Reiner W. Hartenstein, Michael Herz, Thomas Hoffmann, Ulrich Nageldinger: Using the KressArray for Configurable Computing; Proceedings of SPIE Vol. 3526, Conference on Configurable Computing: Technology and Applications, Boston, USA, November 2-3, 1998

Using the KressArray for Reconfigurable Computing

Reiner Hartenstein, Michael Herz, Thomas Hoffmann, Ulrich Nageldinger
University of Kaiserslautern,
Erwin-Schroedinger-Strasse, D-67663 Kaiserslautern, Germany
Tel.: ++49 631 205 2606, Fax.: ++49 631 205 2640, Email:abakus@informatik.uni-kl.de

ABSTRACT

Multimedia applications commonly require high computation power mostly in conjunction with high data throughput. As an additional challenge, such applications are increasingly used in handheld devices, where also small package outlines and low power aspects are important. Many research approaches have shown, that accelerators based on reconfigurable hardware can satisfy those performance demands.

Most of these approaches use commercial fine-grained FPGAs to implement reconfigurable accelerators. However, it has shown, that these devices are not always well suited for reconfigurable computing. The drawbacks here are the area-inefficiency and the insufficiency of the available design-tools. Besides the fine-grained FPGAs, coarse-grained reconfigurable architectures have been developed, which are more area efficient and better suited for computational purposes. In this paper, an implementation of such an architecture, the KressArray, is introduced and its use in the Map-oriented Machine with Parallel Data Access (MoM-*PDA*) is shown. The MoM-*PDA* is an FPGA-based custom computing machine, which is able to perform concurrent memory accesses by means of a dedicated memory organization scheme. The benefits of this architecture are illustrated by an application example.

Keywords: reconfigurable computing, parallel data access, coarse grained reconfigurable architectures, KressArray, accelerators, Xputer, image processing

1. Introduction

In many research projects, reconfigurable devices have proven to achieve much better computation performance than state of the art microprocessors. Reconfigurable devices have the flexibility to adapt to the application needs instead of providing a fixed hardware for all problems. Thus, parallelism can be exploited at a low level and superfluous computational hardware can be avoided. The resulting speedup allows the use of lower clock frequencies, which results in less power consumption. Further, the adaptability of reconfigurable devices allows the replacement of several standard devices by one reconfigurable device.

However, it has shown, that standard FPGAs have several drawbacks when used for computing. This is illustrated using figure 1, which compares the Gordon-Moore curve of integrated memory circuits versus that of microprocessors and other logic circuits. The curves for memories and processors show an increasing integration density gap, currently by about two orders of magnitude. A main reason of this gap is the difference in design style [3]. The high density of memory circuits mainly relies on full custom style including wiring by abutment and the regularity of memory circuits. Microprocessors, consisting of more irregular structures, include major chip area defined by standard cell and similar design styles based on “classical” placement and routing methods. This is a main reason of the density gap.

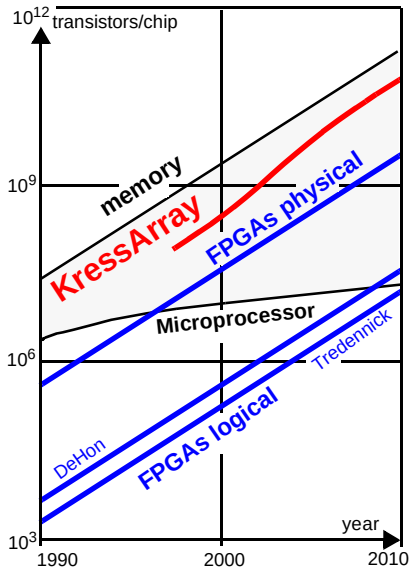

Figure 1. Gordon Moore curve for several devices.

By inserting the physical integration density of FPGAs [25] in figure 1, it can be observed, that FPGA density is growing like the Gordon Moore curve of integrated memories. FPGA integration density is growing such fast that it has already exceeded the density of general purpose microprocessors. The reason is, that FPGAs are also very regular structures and therefore a full custom design style similar to that of memories is used. The gap between the FPGA density and the density of memories depends on the large routing channels and high irregularities inside the logic cells also because of the large amount of hidden RAM.

The major disadvantage of FPGAs is the very high reconfigurability overhead (see logical density of FPGAs in figure 1). Figures having been published [3] indicate up to 200 physical transistors needed for a logical transistor. DeHon states that only 1% of chip area is available for pure application logic [6]. Furthermore, the design process for FPGAs is similar to ASIC design.

Especially, a placement and routing step has to take place, which can take up to hours of computation time. Even worse, FPGAs have only limited routing resources, leaving part of the logic elements unreachable for many applications. In some cases, only about 50% can be used (e.g. see some Xilinx XC6216 examples in [7] or [8]). So FPGAs would hardly be the basis of the mainstream paradigm shift to reconfigurable computing.

An alternative dynamically reconfigurable platform is the KressArray [10], being much less overhead-prone and more area-efficient than FPGAs (figure 1, [3]). The KressArray is a coarse-grained mesh-based reconfigurable architecture operating on 32-bit words and operator-level instead of single-bit and logic-level. No hardware design-techniques are needed like for FPGAs. Operators from a high-level input description are mapped directly onto the array. In contrast to FPGAs the KressArray has no routing facilities. The PEs (processing elements) have only next-neighbor connections, thus avoiding area needed for a reconfigurable routing network. Therefore, also much less configuration data is needed to implement a datapath than for an FPGA solution, where only bit-wide operators are available. This way, also the reconfiguration time is kept short. Necessary routing is done either via the PEs using multiplexers inside or via a global databus also needed for the data in- and output. Due to the mesh-based architecture, the PEs can be arranged using wiring-by-abutment on the chip, thus avoiding long routing channels.

The mapping problem has been mainly reduced to a placement problem. Only a small residual routing problem goes beyond nearest neighbor interconnect, which uses a few PEs also as routing elements. Instead of hours known from FPGA tools the KressArray mapper [19] needs only a few seconds of computation time. Permitting alternative solutions by multiple turn-around within minutes the KressArray tools support very rapid prototyping, as well as profiling methods known from hardware/software co-design. The vision behind this approach is creating and upgrading accelerators by downloading reconfiguration code onto a general purpose reconfigurable accelerator. This reconfiguration code is generated by compilers accepting high level programming language sources.

This paper presents an universal accelerator called Map-oriented Machine with Parallel Data Access (MoM-*PDA*) for reconfigurable computing. It utilizes the KressArray for highly area efficient implementation of a reconfigurable datapath. Further a sophisticated data sequencer hardware performs the address generation for parallel memory banks and is the basis for several speed-up techniques.

The paper is structured as follows. First the KressArray is presented. After that, the MoM-PDA is introduced. The task of the data sequencer is explained and several memory access optimizations are described. After that the compilation framework is summarized. Then an image processing application demonstrates the use of the presented accelerator.

2. KressArray

In this section the KressArray-III is introduced. The KressArray-III consists of PEs called rDPU (reconfigurable DataPath Unit) arranged in a NEWS (North, East, West, South) network. The datapath width of the entire architecture is 32 bit. The KressArray-III is built of several identical KressArray-III devices, which are transparently scalable. Figure 2 shows the KressArray-III prototype 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.

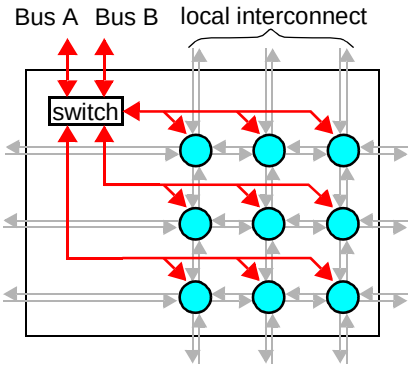

Figure 2. KressArray-III chip structure with hierarchic global

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 routes data from outside to rDPUs of the array and back. 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 in reconfigurable computing 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 KressArray-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 KressArray-III chip. Furthermore two data transfers on the global bus from outside can be performed concurrently.

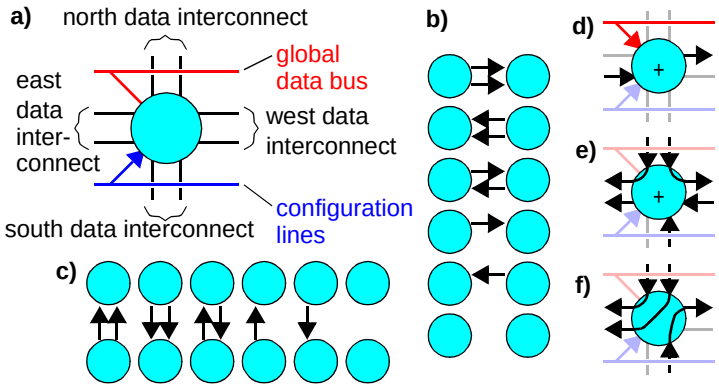

Figure 3. KressArray-III local routability:
a) rDPU with all connections
b) all possible west-east interconnect configurations,
c) all possible north-south interconnect configurations,
d) rDPU serving as arithmetic operator
e) rDPU serving as arithmetic and routing operator, and
f) rDPU serving exclusively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

In addition to the hierarchical global routing network and the configuration lines the KressArray-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 is designed for wiring the local interconnections by abutment. The KressArray-III has two 32-bit 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 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 independently 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 is described later. The arithmetic unit of the rDPU provides a high functionality, covering all arithmetic operations of the language C. The operation configured into the rDPU 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 KressArray-III to form a deep computation pipeline. Furthermore computations are performed asynchronously and different operators can be of different execution time.

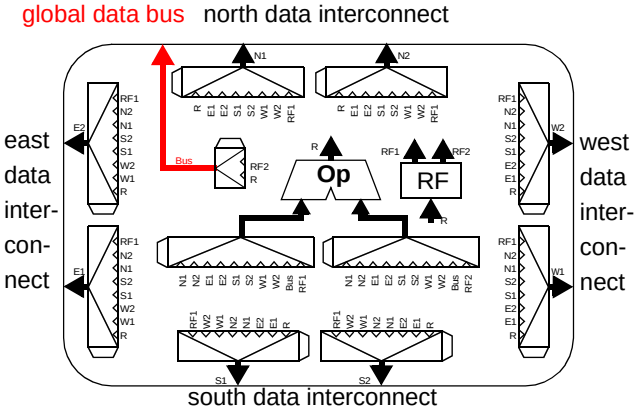

Figure 4. The rDPU architecture.

The performed routing of the multiplexers and the arithmetic function of a rDPU 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 a rDPU of an array changes 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 KressArray-III a related programming framework (see [19]) 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 a host computer 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.

3. MoM-*PDA*

To support highly computing intensive applications structurally programmable platforms providing word level parallelism are needed. Since field-programmable gate arrays (FPGA) provide parallelism on bit-level, field-programmable ALU arrays (FPAA) improve programming on word level. To realize the integration of such *soft ALUs* into a computing machine, a deterministic data sequencing mechanism is needed, because the von Neumann paradigm does not efficiently support *soft* hardware [15]. As soon as a data path is changed by structural programming, a von Neumann architecture would require a new tightly coupled instruction sequencer. A well suited backbone paradigm for implementing such a deterministic reconfigurable hardware architecture is based on data sequencing. In this section the data sequencing paradigm is introduced. Machines based on this paradigm are also well suited to access 2-dimensional memory.

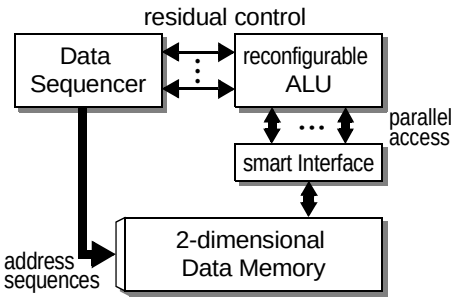

Figure 5. Basic structure of a machine based on the data sequencing paradigm

The main difference between the data sequencing machine paradigm and von Neumann machines is, that the computer is controlled by a data stream instead of an instruction stream (but it is *not* a data flow machine [15]). The program to be executed is determined by first configuring the hardware. As there are no further instructions at run time, only a data memory is required. This data memory is organized 2-dimensionally. At run time an address stream is generated by a data sequencer. The accessed data is passed from the data memory through a smart interface to the reconfigurable ALU (rALU) and back. The smart interface optimizes and reduces memory accesses. It stores interim results and holds data needed several times. Figure 5 shows all necessary components and their interconnect.

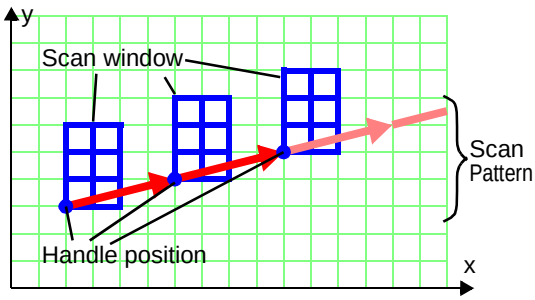

Figure 6. 2-dimensional memory organization and execution model

This principles are derived from the fact that many computation-intensive applications iterate the same operations over a large amount of data. They are accelerated by reducing the addressing overhead. All data needed for one computation step is held in the smart interface and can be accessed in parallel by the rALU. In contrast to von Neumann computers the rALU is not dependent of the sequencer (Von Neumann machines use their ALU for address computations). Therefore, it can be made application specific. The rALU is implemented with the coarse-grained FPAA KressArray [10][18]. Computations are performed by applying a configured complex operator on the data. This operator is also called compound operator. This hardware structure has the big advantage that, if the smart interface is integrated into the rALU, the rALU can be changed easily without modifying the whole machine. The residual control between data sequencer and rALU is needed when the data stream has to be influenced by the result of previous computations.

To clarify how operations are executed the execution model is pictured in figure 6. A large amount of input data is typically organized in arrays (e.g. matrix, pictures) where the array elements are referenced as operands of a computation in a current iteration of a loop. These arrays can be mapped onto a 2-dimensional organized memory without any reordering. This arrangement of data is called data map. The size of the data map depends on the application and varies in height and width. The part of the data memory which holds the data for the current iteration is determined by a so called scan window, which can be of any size and shape. The scan window can be seen as a model for the smart interface which holds a copy of the data memory. Each position of the scan window is labeled as read, write or read and write. The labels indicate the performed operation to the specified memory location. The position of the scan window is determined by the lower left corner, called handle (see figure 6). Operations are performed by moving the scan window over the data map and applying the compound operator on the data in each step. Thus this movement of the scan window called scan

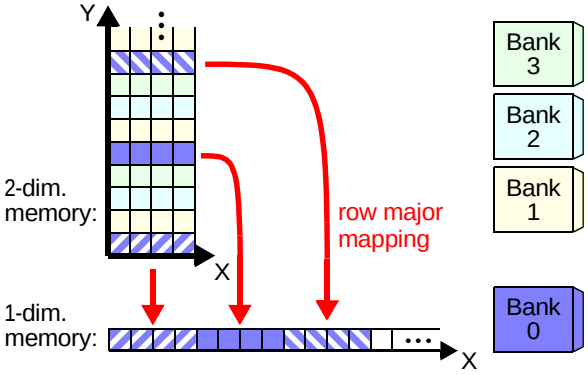

Figure 7. Mapping of 2-dim. memory to linear banks

pattern is the main control mechanism. Because of their regularity, scan patterns can be described by only a few parameters. As a result no instruction cycles are needed for address generation.

In fact the execution model realizes a 2-level data sequencing. In the first level with the position of the scan window all data for one iteration is indicated. On hardware level the position of the scan window is determined by the x- and y-addresses of the scan pattern. On the second level the data is sequenced from the scan window into the rALU and back. This is done for each position depending on the read/write labels. On hardware level the data sequencer computes physical memory addresses for each scan window position.

3.1 Physical Implementation Issues

Since no 2-dimensional organized memories modules are commercially available it has to be mapped onto 1-dimensional memories. Therefore the 2-dimensional memory is cut in slices. Depending on the width of the data map linear memory segments are obtained, which can be appended and mapped to a linear data memory (row major mapping). Mappings to any number of physical memories are possible. By mapping the 2-dimensional memory only to one linear memory module a 2-dimensional visualization of data memory is obtained. Figure 7 illustrates the mapping of a 2-dimensional data map onto 4 parallel memory banks. For bank 0 row major mapping is shown.

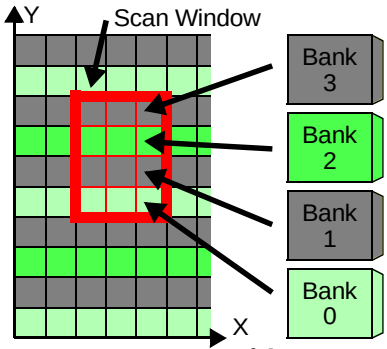

Figure 8. Mapping of the 2-dimensional memory to four linear memory banks

Further figure 8 demonstrates how scan window positions are assigned to several memory modules (4 in the example). The neighboring memory slices are mapped to parallel memory banks. If the number n of parallel memory banks is smaller than the height of the memory map, also several slices are appended and mapped to one memory bank as described before. Because of the scan window holds only a small part of the data map, only a few number of parallel memory banks are sufficient to have each row of the scan window accessible in parallel.

Because data needed in one iteration of a loop is mapped to parallel memory banks, instruction level parallelism (ILP) on hardware level becomes more feasible. By providing multiple datapaths between memory banks and a parallel ALU, loop bodies can be computed in concurrently on hardware level.

An algorithm is executed by the movement of the scan window over the datamap, whereas the scan window holds the source and destination data. Depending on the order in which memory locations are accessed by the scan window a specific scan pattern results. Because of the regularity of memory accesses of loops regular scan patterns are achieved. Figure 10a shows a scan pattern called video scan [11][13], which often occurs in image processing. The data map contains an input picture, where all pixels will be processed during computations.

Because of the scan window size may be larger than the scan pattern step width, scan window overlapping during computation occurs (e.g. in image processing, see example section 5). Figure 10 also illustrates the different cases for scan window overlapping. While in figure 10e no overlapping occurs because of the scan step is longer than the scan window width, the examples in figure 10b,c,d demonstrate overlapping in x-, y- and x/y- directions. While non-overlapping scan window positions reference data, which was not used in the near past, overlapping scan window positions access data used directly before. This is the base of a deterministic *cache-like* memory access optimization. Because of the same scan window overlapping occurs at each scan window movement, the optimization can be implemented with less hardware effort. The overlapping positions can be figured out at compile time, when the scan window for the application is determined.

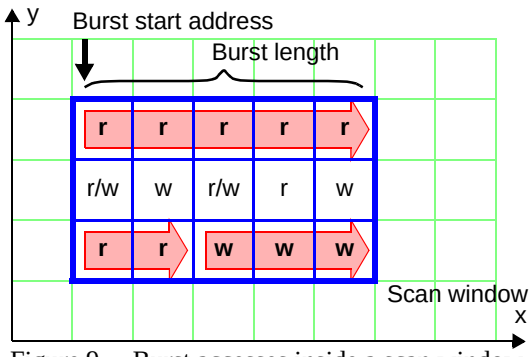

Figure 9. Burst accesses inside a scan window

Because of traditional caches rely on the locality principle cache hits can only be expected for an application, but not guaranteed. In contrast to this, the presented approach certainly provides the calculated reduction of memory accesses. This could be compared with a *hit-rate* of 100%. Because of the access optimization depends on the application, a programmable *cache* is required. The hardware implementation of the scan window is called smart interface, which is integrated into the rALU.

Since modern DRAMs provide burst accesses this optimization can also be adopted to 2-dimensional memory. Originally burst read operations where introduced to fill complete fixed-width cache lines within only one memory access. Multibank DRAMs (MDRAM, [23]) allow variable length burst read or write operations. Burst accesses are performed by setting a start address and the memory generates automatically accesses to the following memory locations until a stop-command is per-

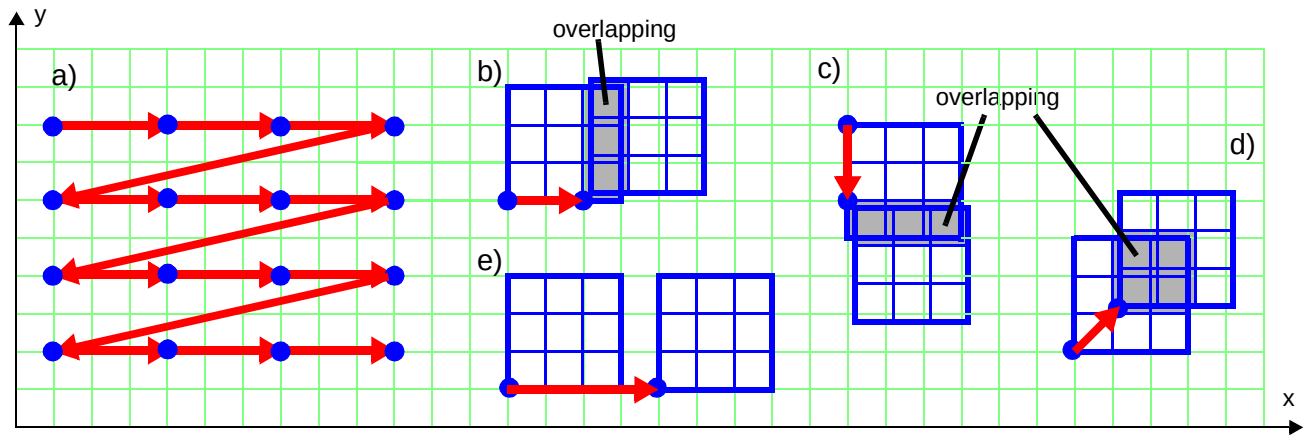

Figure 10. (a) Scan Pattern Example,
(b) (c) (d) Scan Steps with Scan Window overlapping, (e) Scan Step without Scan Window overlapping

formed. The number of performed accesses is called the burst length.

The technique of burst memory access is used inside a scan window, if accesses of the same kind (read or write) are located horizontally side by side. Figure 9 shows several examples for burst accesses inside a scan window. Because of only an initial address has to be set, the memory accesses are performed faster.

3.2 Summary of Memory Optimization Techniques

Three different techniques to accelerate memory accesses are introduced: scan window overlap, parallel memory and burst access. These methods complement one another as they are effective in different directions. While parallel memory accelerates memory accesses in y-direction, burst accesses work in the x-direction. Scan window overlappings act in the direction of the scan pattern. The three methods have different degrees of optimizing the memory access. This results in an order of priority based on this degree: Scan window overlapping has the highest priority, because unnecessary memory cycles are saved completely. The degree of optimization increases with the overlapping area. Parallel memory performs memory accesses concurrently. The degree of optimization increases with the number of parallel memory banks. Burst accesses save the time to set the memory addresses but the accesses are performed sequentially. The degree of optimization increases with the burst length.

3.3 Data Sequencer Implementation

The data sequencer [11][13] is the basis for exploiting efficiently the proposed memory organization. It is a generic address generation unit with 32 bit address range. Generic address generation has the advantage that no instruction and therefore no memory cycles are needed for address generation. Addresses are not computed by command sequences but generated out of generic parameters.

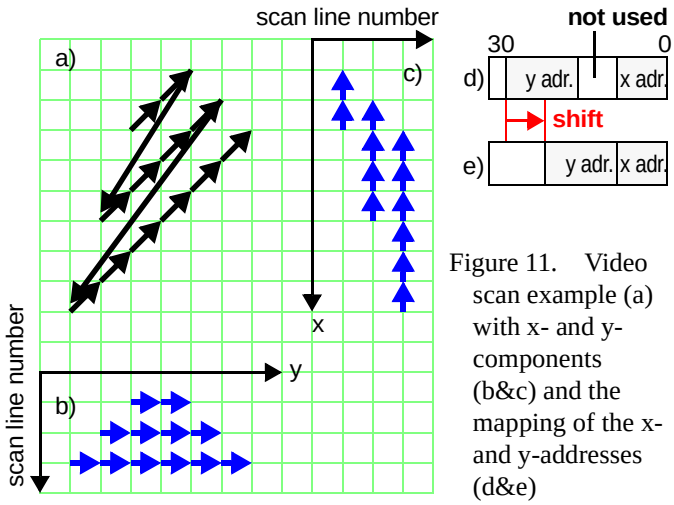

Figure 11. Video scan example (a) with x- and y-components (b&c) and the mapping of the x- and y-addresses (d&e)

Video scans are composed of an x- and y-part with 16 bit for each (figure 11). The address generation for both parts is identical. After generation of the x- and y-address parts, they are merged to one physical memory address. But simply merging two 16 bit x- and y-addresses to a 32 bit memory address would require an enormous memory size of 4 giga words with a fixed row length of 64 kilo words for any application. To reduce fragmentation caused by a fixed row length the output addresses are shifted according to the size of the actual data map, i.e. not every application requires the complete address range of 16 bits for each dimension. Often some leading bits of the x and y addresses are unused. This results in different data map sizes and shapes. The leading zeros of the x-address are the reason for this waste as can be seen in figure 11d. Therefore the higher bits of the address register are shifted to the lower positions until all unused lower zero bits are eliminated (figure 11e).

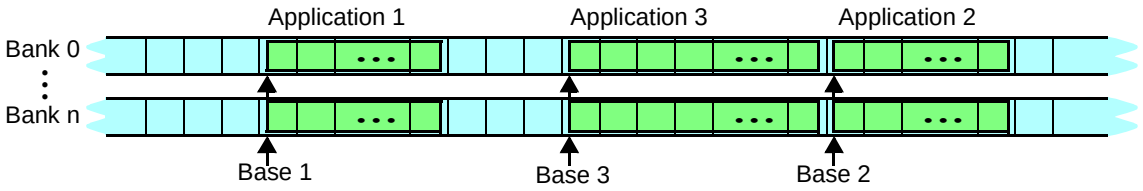

Figure 12. Data of several applications mapped to the linear memory banks

This strategy leads to an own address space for each data block in the 2-dimensional memory. If there is data for several applications in the physical memory the hardware has to secure that there is no memory violation. For each application a base address is added and the necessary space is allocated. Figure 12 shows how multiple applications are mapped onto the parallel memory banks with linear view of the banks.

The data sequencer hardware can be divided into a central control unit and an address generation data path. The data path is a pipelined structure with two stages; the Handle Position Generator (HPG) and the Scan Window Generator (SWG). Each stage of the pipeline performs one level of the 2-level data sequencing. After the two pipeline stages a memory map function (Memory Mapper, MemM) and the Burst Control Unit (BCU) process the generated memory accesses. The complete structure is illustrated in figure 13.

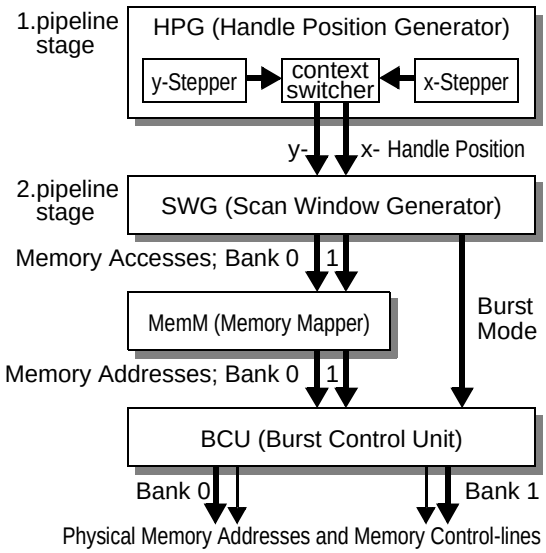

Figure 13. Address generation datapath of the Data Sequencer

Fragmentations of the physical memory are avoided be shifting the memory accesses according to figure 11d,e. This is done by the memory mapper unit. To support memory with burst options an additional unit has to control this operations. The required signals for variable burst lengths are generated by the Burst Control Unit (BCU, [4]) based on the burst information provided by the SWG. Because the memories are accessed in parallel the BCU hardware is instantiated for each memory bank. A more detailed description of the data sequencer hardware can be found in [11] or [13].

3.4 The Architecture of the Map-oriented Machine with Parallel Data Access (MoM-*PDA*)

The MoM-PDA is an accelerator to be connected to a host computer via a PCI interface. It is based on the data sequencing paradigm ([16], also called Xputer paradigm) and utilizes the KressArray for reconfigurable computing. This section gives an short overview on the overall hardware structure of the MoM-*PDA*.

The most important new feature of the MoM-*PDA* prototype is parallel high speed access to the data as described above. Therefore it has 2 parallel banks of MDRAM. Addresses for the MDRAMs are computed by the data sequencer and extended with burst information by the BCU. In the current prototype implementation the data sequencer of the MoM-*PDA* is mapped to an Altera FLEX10K100 device [1]. The novel data sequencer structure handles up to 16 parallel tasks each consisting of an complex scan pattern [11][13]. Therefore up to 16 scan windows can operate on the data memory concurrently. The computation of the parallel tasks is done like known from multi-tasking systems. As described above the MoM-*PDA* has parallel data access capabilities. Therefore the data sequencer generates two parallel address streams. These addresses are passed to the Burst Control Unit (BCU, [4]), which initiates the accesses to the MDRAMs [23]. Computations are normally performed by the KressArray. Data is first routed to a reconfigurable ALU port (rAP, figure 14). It is implemented with a Xilinx XC6216 [24], which is connected to data lines of the data memory. The programmable space of the XC6216 device will be partitioned

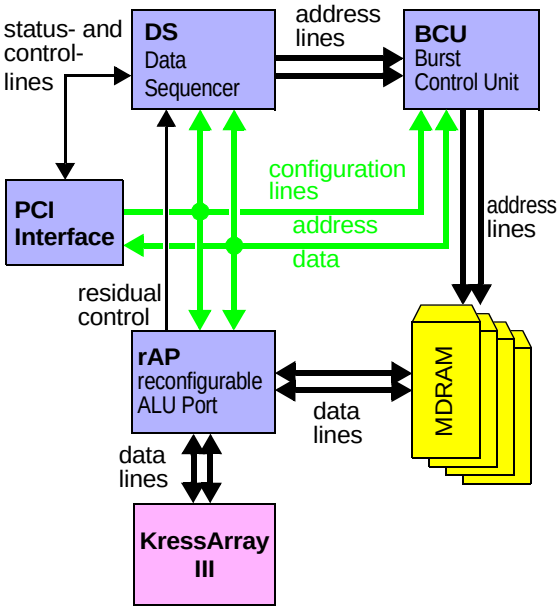

Figure 14. MoM-PDA machine overview

into two functional units. One unit will be the parallel memory interface for the MDRAMs [23] and the smart interface. This unit is the same for every application and is configured once at power up. The remaining programmable space can be used in two different ways:

- for calculations it is (re-)configured for every task. Computations are performed by applying the configured operations on the data in the smart interface. This allows to build a small version of the MoM-*PDA* without the KressArray. Simple problems may be computed in the XC6216 only. Therefore a operator library has been implemented, mainly containing image processing applications (see [7] and [8]).
- as a connection to KressArray [12]. In that operation mode the XC6k optimizes data exchange between the KressArray and the data memory.

4. Application development for the MoM-*PDA*

In the previous sections, a computing paradigm has been introduced, which uses strict separation of control-flow and datapath. As this paradigm is complementary to the von-Neumann paradigm, special compilation techniques are needed. For the mapping of applications onto the MoM-*PDA* the partitioning compilation environment CoDe-X [2] has been developed. CoDe-X is capable to compile applications specified in a high level language onto a system comprising a host and an Xputer-based accelerator. In this paper, only a brief overview about the CoDe-X framework is given. For a detailed description, see [2].

CoDe-X accepts a description in a superset of the C language. The input program is in the first step partitioned in a set of tasks for the host and a set of tasks for the accelerator. This partitioning is driven by a performance analysis, which minimizes the overall execution time. Also, the first level partitioner applies loop transformations on the code to improve the performance.

The accelerator tasks are then processed by the X-C [22] compiler, which accepts a subset of C as input language. A course sketch of this second compilation step is illustrated by an example in figure 15. The X-C compiler performs a second level of partitioning by dividing the input up into code for the data sequencer and code for the rALU comprising the KressArray. First, the data arrays of the application are mapped onto the data memory. Basically, each array becomes a two-dimensional area and needs a scan pattern to access the data. If the loop structure allows, several arrays can be aligned [22] to form one data area, so only one scan pattern is needed (see figure 15).

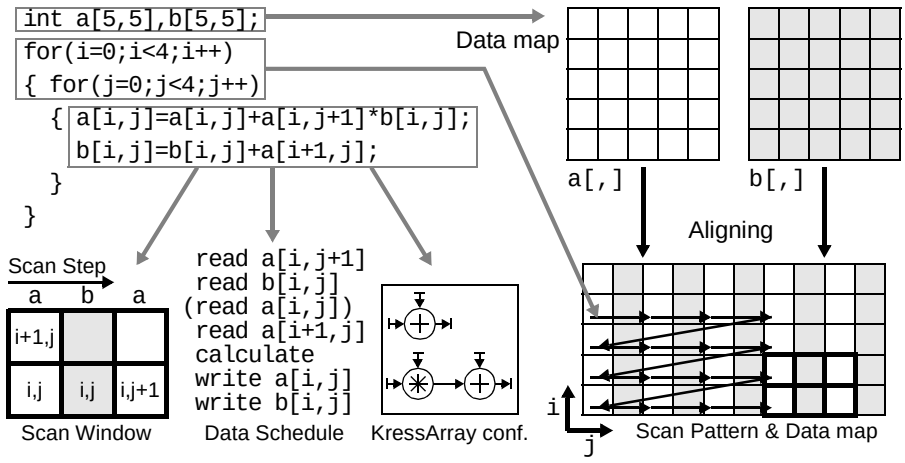

Figure 15. Mapping of an application onto an Xputer

Due to the architecture of the data memory, the arrangement of the data affects the memory bandwidth. Thus, the mapping of the application data is combined with an optimization step. Currently, a system is implemented, which uses two approaches for exploiting the data memory burst mode to provide high memory bandwidth. The first approach is suitable for algorithms, where the scan window is sized 1 by 1, which means, that the scan pattern directly corresponds to the access sequence of single data words. From the scan pattern parameters, a transformation can be derived, which is used to rearrange the data map in a way, that the scan direction of the innermost loop has the same direction as the burst mode. This way, very high data transfer rates can be achieved. A more detailed description of the algorithm can be found in [9]. The second optimization approach is suited for algorithms with big scan windows, especially if the data arrays are aligned. In this approach, the data is rearranged such that the access sequence of data words inside the scan window is optimized. This is also done by data rearrangements. As the dependencies between data words of consecutive steps (see e.g. the value of a[i,j] in figure 15 and the discussion below) have to be retained, the rearrangement possibilities are normally restricted. Thus, in a first step, the limits of the valid rearrangements are determined. Then, an optimization tries to find a new scan window, which makes optimal use of the parallel access to adjacent rows and the burst mode inside a row.

After the mapping of the application data, the scan pattern is generated from the loop structure [22]. In the example, there are two nested loops whereof a so-called video scan is derived. The loop body holds the information for the calculation to be performed onto the data. The variable references in the loop body combined with the scan pattern determine the scan window

for the application. In the example in figure 15, there is only one scan window due to the alignment of the two data arrays. The functions in the body itself are then transformed into a KressArray configuration by the Datapath Synthesis System (DPSS, [19]). The DPSS determines a mapping of the application datapath onto the KressArray. For this task, a simulated annealing algorithm is used to optimize the resulting structure in terms of needed routing elements and bus connections. In a second step, the DPSS determines an optimal data schedule, describing the sequence for reading and writing the data words inside the scan window, thus implementing part of the memory optimization mentioned earlier. In the example in figure 15, the value of a[i,j] is the same as a[i,j+1] from the previous iteration, so it has to be read from memory only at the beginning of each scan line. For all other positions, the according value from the last step is taken, which has been stored in the smart memory interface. In figure 15, this situation is illustrated by the parentheses around the read operation for a[i,j] in the data schedule.

5. Application

In this section a generic 3x3 linear filter for image-processing is presented as an example design for the MoM-*PDA*. This application is also implemented for execution in the reconfigurable ALU port (figure 14), which is presented in [8]. Here a much more powerful KressArray implementation of this application is given.

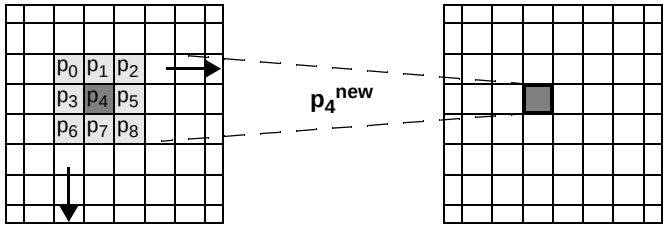

Figure 16. 3x3 Image-Transformation

A filter φ for image-transformation is an operator, which assigns a new value to pixel p_4^{new} depending on the pixel-value of p_0 and N of its neighbor pixels (figure 16).

If calculation of the modified pixel-value is independent from its position, the filter φ is called homogenous. The selection of neighbor pixels is done by moving a window from left to right and from top to bottom over the complete image. All pixel covered by this window are used. In the case of a 3x3 filter, an array of 3 times 3 pixels is chosen with p_4 as the center of the array (see figure 16). In case of a linear filter, the operation is a linear function of all elements in the defined array.

For a generic 3x3 linear filter, p_4^{new} is calculated as follows:

$$p_4^{new} = \frac{1}{j} \cdot \sum_{i=0}^{8} p_i \cdot k_i$$

Figure 17a shows how an image is stored in the 2-dimensional data memory and how the image is processed by moving the scan window over it. The according scan pattern is generated by the data sequencer. To benefit from the parallel data access capabilities of the MoM-*PDA* architecture, two pixels of the image are processed concurrently. Depending on the available hardware resources also more pixel can be computed in parallel.

In the example p_4^{new} and p_7^{new} are computed at the same time. Therefore the pixel p_0 - p_{11} are needed as input data. Because the scan window overlaps during movement according to the scan pattern (figure 17b), only half of the input data has to be accessed. The data accessed in the iteration before is stored in the smart interface (figure 17c). As a result all input data for one computation step can be accessed in two concurrent burst read operations. After processing the data the results are written back in parallel.

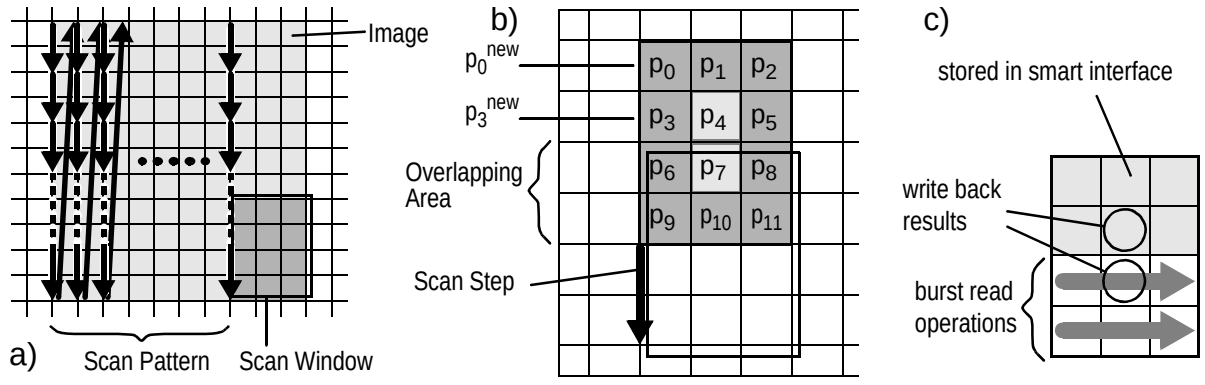

Figure 17. Overlapping scan Windows. a) Data map with image covered by a scan pattern, b) scan window contents with source data (p_0 - p_{11}) and results (p_4^{new}, p_7^{new}) of one computation step, c) memory accesses within the scan window during one computation step

5.1 Performed Memory Access Optimizations

In this section the presented memory access technique is compared to an non-optimized method. The time to set an address to a memory module is assumed to be *a*. Read and write access times are assumed to be of the same length *d*. For each read or write operation an address has to be set. The only exception are burst operations, where only one address for the whole burst has to be set. For the final access time t_{acc}, *a=d* is assumed.

The non-optimized system has to read all 9 input pixels to compute a result each time. Therefore the memory access time is computed as follows: $t_{acc} = 9(a+d) + (a+d) = 20d$

The proposed method benefits of the parallel data access capabilities and computes 2 pixels at a time. Furthermore, data needed in the next iteration is stored in the smart interface. As input data can be read in a burst mode, some set address cycles can be saved: $t_{acc} = ((a+3d) + (a+d)) / 2 = 3d$

As a result a non-optimized memory access scheme takes 6.6-times longer memory access time than the proposed method.

5.2 KressArray Implementation

Figure 18 shows an optimized KressArray-III implementation to calculate the generic 3x3 linear filter for two pixels in parallel. It is especially designed to reduce the KressArray-bus traffic. Therefore rDPUs are programmed to work as a register. Each of these rDPUs stores a pixel and routes it to both processing units via the local interconnect. Further the pixels of the overlapping area of the scan window are stored and routed via the local interconnect to their position for the next iteration. The register rDPUs are labeled in figure 18 by the scan window position (figure 17b) of the data they hold. The processing unit for p_4^{new} is shown on the right side of figure 18 and the unit for p_7^{new} is located on the left side. The coefficients for the computations are directly mapped into the array. The sum of the products is built in two stages because one rDPU does not have enough inputs to read all products in parallel.

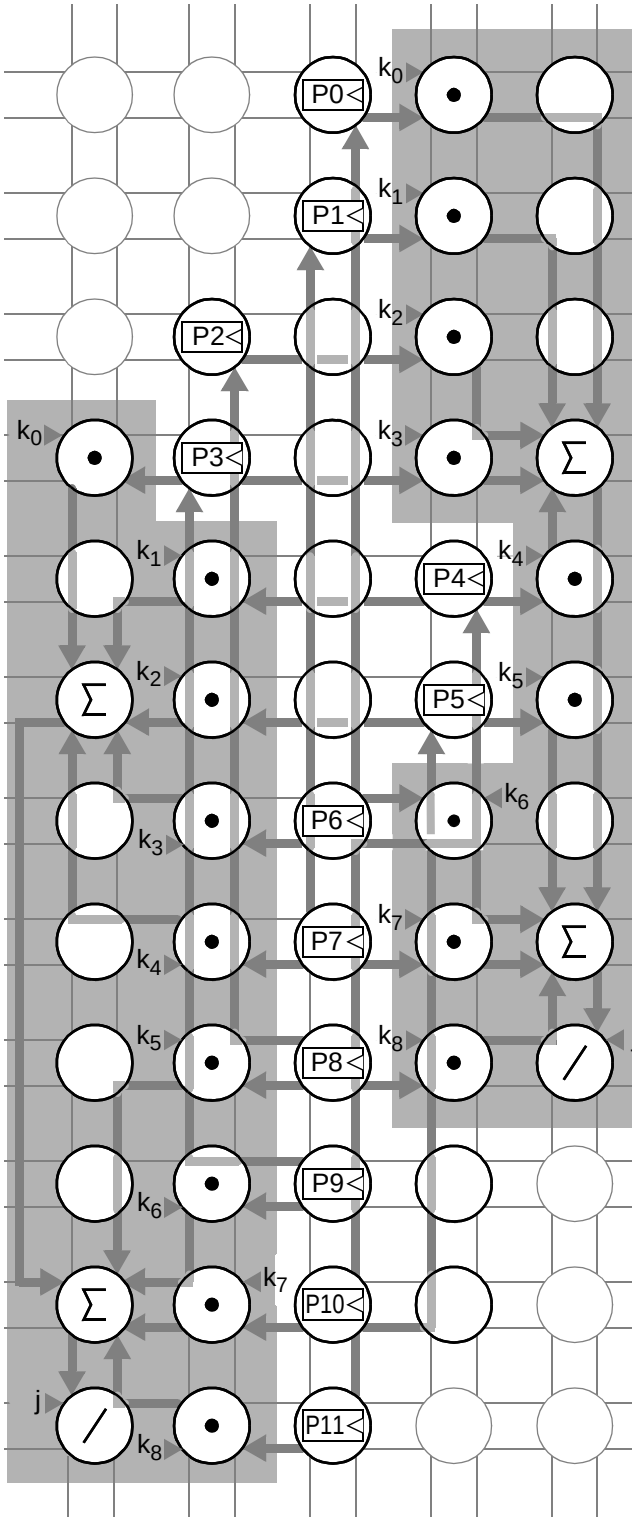

Figure 18. Optimized KressArray-III implementation of a generic 3x3 linear filter.

5.3 Performance Results

The KressArray is designed to operate in a pipelined fashion. Because the filter execution time is longer than the memory access time, pipelined execution allows to make more use of the memory- and KressArray-busses. To make full use of their capacity a parallel computation of 4 pixels is necessary.

Figure 19 illustrates the pipelined execution of the generic 3x3 filter implementation. The utilization of the two memory- and the two KressArray-busses is shown as well as the parallel operation of the rDPUs inside the KressArray. Data fetches from the register rDPUs are shown separately.

Each box in figure 19 represents one clock cycle at a clock speed of 25MHz of our experimental prototype. The computation of two pixels would need 19 clock cycles. Because of the pipelined execution 4 pixels are processed. With a larger KressArray the prototype has a capacity to compute a maximum of 8 pixel in 19 clock cycles, which is a computation power of 10.5 Mpixel/s. To compute more than 8 pixel is not possible because of the capacity of the memory busses. For higher performance more accelerator boards can be used in parallel or a

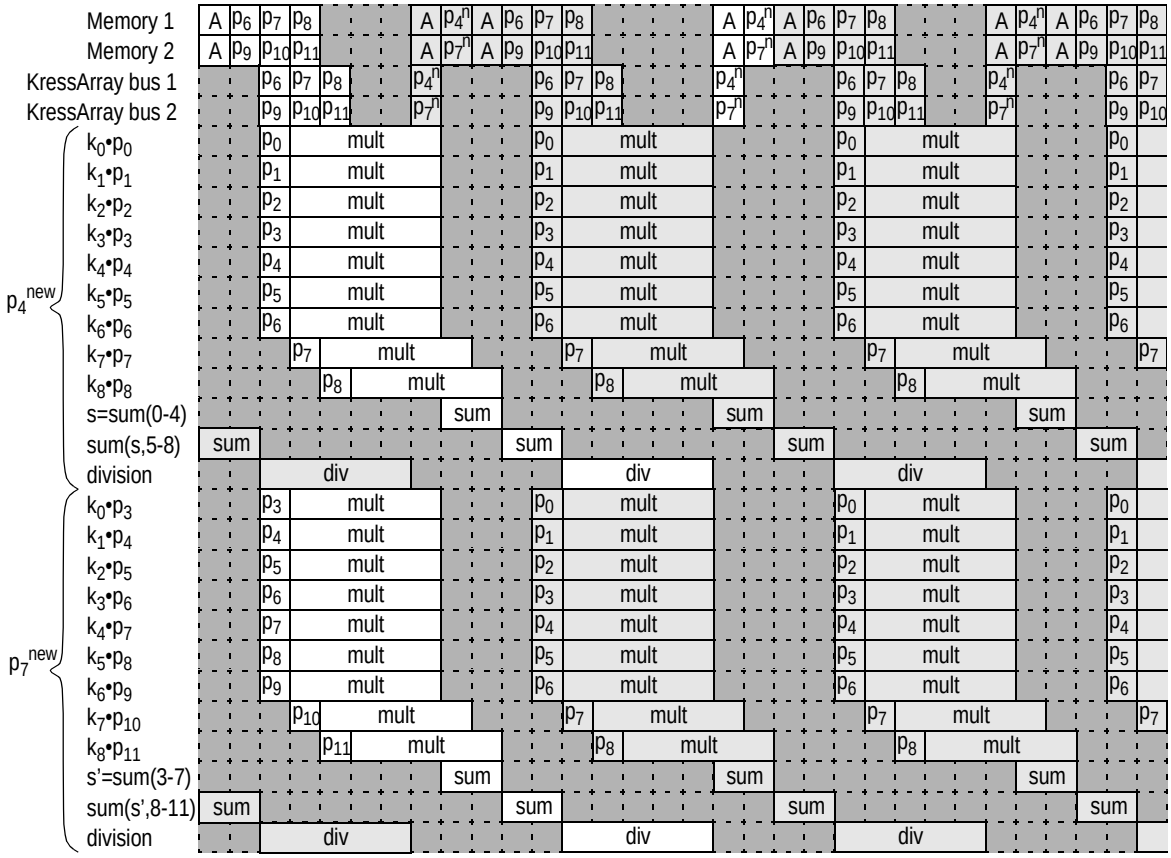

Figure 19. Pipelined execution of the generic 3x3 linear filter implementation.

board with more parallel memory banks has to be designed. Also the prototypes clock frequency could be increased by using better technology.

6. Results

We have introduced the KressArray which allows to implement effiently computational datapaths. The KressArray is a coarse-grained reconfigurable architecture in contrast to FPGAs which are more fine-grained. It has been shown that it is more area efficient and well suited for reconfigurable computing. Due to the coarse structure featuring 32-bit arithmetic operators the complexity of the compilation step is reduced. Especially the application mapping is mainly reduced to a placement algorithm, based on simulated annealing. A compiler framework has been briefly introduced, which maps applications specified in C onto a host- / accelerator- system.

Based on the KressArray a novel machine architecture, the MoM-*PDA*, has been presented, which relies on the data sequencing paradigm. It comprises a 2-dimensional data memory and a data sequencer. The data sequencer is a dedicated device to drive accelerators for reconfigurable computing. It performs the data sequencing task without needing any memory cycles. Further it is the base for several memory access optimizations. The 2-dimensional memory of the MoM-*PDA* features two parallel banks and burst mode providing a high bandwidth for data transfers.

7. References

1. N.N.: Altera 1995 Data Book; Altera Corporation, San Jose, California, USA, 1995.
2. J. Becker: A Partitioning Compiler for Computers with Xputer-based Accelerators; Ph.D. dissertation, University of Kaiserslautern, 1997.
3. J. Becker, R. Hartenstein, M. Herz, U. Nageldinger: Parallelization in Co-Compilation for Configurable Accelerators; in proccedings of Asia and South Pacific Design Automation Conference, ASP-DAC'98, Yokohama, Japan, Feb. 10-13, 1998
4. M. Bednara: A Burst Control Unit to Perform Optimized Access to Multibank DRAMs; Diploma Thesis, University of Kaisrslautern, Kaiserslautern, Germany, May 29,1998
5. P. Budnik, D. J. Kuck: The Organization and Use of Parallel Memories; IEEE Transactions on Computers, Dec. 1971.
6. A. DeHon: Reconfigurable Architectures for General Purpose Computing; report no. AITR 1586, MIT AI Lab, 1996
7. F. Gilbert: Development of a Design Flow and Implementation of Example Designs for the Xilinx XC6200 FPGA Series; Diploma Thesis, University of Kaiserslautern, Kaiserslautern, Germany, May 29, 1998. Download from http://xputers.informatik.uni-kl.de/reconfigurable_computing/XC6k/index_xc6k.html
8. R. Hartenstein, M. Herz, F. Gilbert: Designing for the Xilinx XC6200 FPGAs; 8th International Workshop on Field Programmable Logic and Applications, FPL'98, Tallinn Technical University, Estonia, Aug. 31 - Sept. 31, 1998
9. R. W. Hartenstein, M. Herz, T. Hoffmann, U. Nageldinger: Exploiting Contemporary Memory Techniques in Reconfigurable Accelerators; 8th International Workshop on Field Programmable Logic and Applications, FPL'98, Tallinn Technical University, Estonia, Aug. 31 - Sept. 31, 1998
10. R. Hartenstein, M. Herz, T. Hoffmann, U. Nageldinger: On Reconfigurable Co-Processing Units; Proceedings of Reconfigurable Architectures Workshop (RAW98), held in conjunction with 12th International Parallel Processing Symposium (IPPS-98) and 9th Symposium on Parallel and Distributed Processing (SPDP-98), Orlando, Florida, USA, March 30,1998
11. R. Hartenstein, J. Becker, M. Herz, U. Nageldinger: A Novel Universal Sequencer Hardware; Proceedings of Fachtagung Architekturen von Rechensystemen ARCS'97, Rostock, Germany, September 8-11, 1997
12. R. Hartenstein, J. Becker, M. Herz, U. Nageldinger: An Embedded Accelerator for RealWorld Computing; in Proceedings of IFIP International Conference on Very Large Scale Integration, VLSI'97, Gramado, Brazil, August 26-29, 1997
13. R. Hartenstein, J. Becker, M. Herz, U. Nageldinger: A Novel Sequencer Hardware for Application Specific Computing; Proceedings of 11th International Conference on Application-specific systems, Architectures and Processors, ASAP'97, Zurich, Switzerland, July 14-16, 1997
14. R. Hartenstein, J. Becker: A Two-level Co-Design Framework for data-driven Xputer-based Accelerators; Proc. of 30th Annual Hawaii Int'l Conf. on System Sciences (HICSS-30), Jan. 7-10, Wailea, Maui, Hawaii, 1997.
15. 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.
16. R. Hartenstein, A. Hirschbiel, M. Weber: A Novel Paradigm of Parallel Computation and its Use to Implement Simple High Performance Hardware; InfoJapan'90, Int'l. Conf. memorizing the 30th Anniv. of Computer Society Japan, Tokyo, Japan, 1990.
17. K. Hwang, F. A. Briggs: Computer Architecture and Parallel Processing, International Edition; McGraw-Hill, 1985.
18. R. Kress: A Fast Reconfigurable ALU for Xputers; Ph. D. Thesis, University of Kaiserslautern, 1996.
19. R. Kress et al.: A Datapath Synthesis System for the Reconfigurable Datapath Architecture; Asia and South Pacific Design Automation Conference 1995 (ASP-DAC'95), Chiba, Japan, Aug. 29 - Sept. 1, 1995
20. N.N: Reduce DRAM Cycle Times With Extended Data-Out; Technical Note, Micron Technology, 1995.
21. D. A. Patterson, J. L. Hennessy: Computer Architecture A Quantitative Approach; Morgan Kaufmann Publishers, 1990.
22. K. Schmidt: A Program Partitioning, Restructuring, and Mapping Method for Xputers; Ph.D. dissertation, University of Kaiserslautern, 1994.
23. N.N.: Siemens Multibank DRAM, Ultra-high performance for graphic applications; Siemens Semicond. Group, Oct., 1996.
24. N.N.: The Programmable Logic Data Book; Xilinx Inc., San Jose, California, USA, 1996.
25. N.N.: Programmable Logic Breakthrough '95, Technical Conference and Seminar Series; Xilinx, San Jose, CA, USA, 1995.