18

An Embedded Accelerator for Real World Computing

R. W. Hartenstein, J. Becker, M. Herz, U. Nageldinger
University of Kaiserslautern
Erwin-Schrödinger-Straße, D-67663 Kaiserslautern, Germany
Fax: ++49 631 205 2640, email: abakus@informatik.uni-kl.de
URL: http://xputers.informatik.uni-kl.de/

Abstract

This paper presents a new general purpose accelerator based on the Kress ALU Array III (KrAA-III) - a novel field programmable ALU array (FPAA), which efficiently supports high performance arithmetic computations by massive pipelining. The KrAA-III and its underlying concepts will be introduced as a generalization of systolic array structural principles - illustrated by a simple real world computing example. To explain this approach for embedded accelerators the underlying novel machine paradigm, having been published earlier, is recalled as far as needed for comprehensibility.

Keywords

data sequencing, FPAA, FPL-based accelerator, generic address generation, image recognition, parallel memory technology

1 INTRODUCTION

The implementation of the idea of real world computing [RW97] requires supercomputing power within hand-held equipment. To achieve this highly parallel high performance hardware at low cost is required. By general purpose coarse granularity parallel reconfigurable accelerators such achievements become feasible — based on a novel kind of field-programmable devices, which provide high throughput by instruction level parallelism [HB96]. The summing up of very high design and re-design cost because of short product life times as e. g. in the consumer electronics market creates a desire for substantially reduced design cost. The new platform introduced by this paper provides a method of "software-only" (configware-only) implementation of add-on accelerators.

FPGAs currently available commercially and mainly used to implement control and glue logic, use only single bit data paths. In contrast to this fine grain parallelism we face different requirements for implementing computing elements. Although some FPGA manufacturers developed more feasible devices (e.g. Xilinx XC6200, Altera FLEX10k, Atmel AT6000) they are far from meeting such requirements. A wider datapath (e.g. 16 or 32 bits) is needed for arithmetic, relational, or other functions. Reconfiguration should be drastically faster and should also be possible during runtime. Furthermore context switches have to be possible to change efficiently between several tasks during runtime.

VLSI: Integrated Systems on Silicon R. Reis & L. Claesen (Eds.)

In particular this is essential to provide the efficiency and flexibility needed for real world computing. Because of these requirements such a computing element goes more in the direction of processing elements (PEs) like used in systolic array implementation, in contrast to FPGAs, which are similar to the ASIC approach. Therefore we advocate the new class of Field Programmable ALU Arrays (FPAAs) [HB96], which are loosely related to systolic arrays by being a highly area-efficient mesh-connected array through wiring by abutment and extensive use of multiple pipelining (fig. 3). Examples of FPAAs are the Programmable Arithmetic Devices For DIgital Signal Processing (PADDI, [YR93]) or the reconfigurable Datapath Architecture (rDPA, [Kr96]).
This paper presents a novel kind of FPAAs called Kress ALU Array III (KrAA-III, see also [HB97a]). Whereas KrAA-I has been unidirectional, KrAA-III is bidirectional and its rALUs have two ports at each side (fig. 8). Further a new universal accelerator hardware called Map-oriented Machine with Parallel Data Access (MoM-*PDA*) integrating such Kress FPAAs is introduced. This machine enables parallel memory access without any consistency problems and supports also interleaved memory access with burst options. The proposed accelerator works according to the Xputer paradigm [AH94] and allows flexible use of different reconfigurable ALUs (rALUs). Because we use the KrAA-III as a rALU array, we also call this prototype Kress Machine. After the introduction of the hardware, a real world computing example from automotive control demonstrates the use of Xputer-based accelerators applying FPAAs.

2 THE ARCHITECTURE OF THE NOVEL KRESS ALU ARRAY

To support highly computing-intensive applications structurally programmable platforms are needed providing word level parallelism instead of the bit level parallelism like FPGAs. It has shown that on tasks with large data elements, fine-grained devices pay too much area for interconnect than coarser-grained devices. (For an extensive study on area utilization refer to [De96].) For area efficiency this new platform is suitable for full custom design, like known from ASAPs (application specific array processors) operating in SIMD mode. But ASAPs are not structurally programmable and support only problems with completely regular data dependencies (fig. 4).
Therefore FPAAs are as dense as ASAPs but each 'PE' is programmed individually (fig. 4). Each array element of the KrAA-III consists like the rDPA of a reconfigurable datapath units (rDPUs). (Kress has called his approach reconfigurable datapath array, rDPA, [Kr96].) In contrast to the original approach, also the local interconnects between the rDPUs are programmable individually (figure 1). Furthermore the global interconnect between all rDPUs is constructed hierarchically and enables parallel data transfers. For speed up of task switches a context switch mechanism is developed, that allows reconfiguration of unused layers during runtime. Other features have been adopted from the original rDPA [Kr96]. These are:

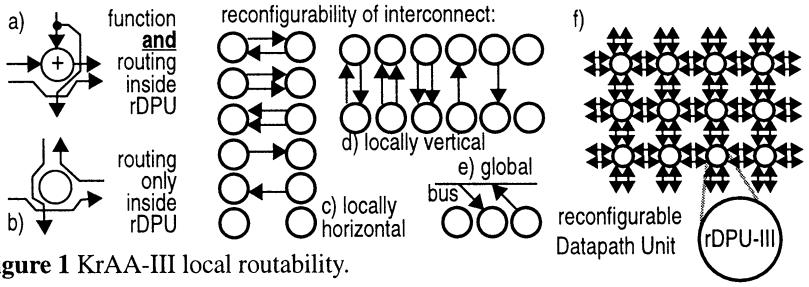

Figure 1 KrAA-III local routability.

	FPGA	micropro.	systolic array
area of circuits active at a time (rough estimation)	~1 %	~2 %	> 50 %

Figure 3 Massive area-inefficiency of main stream technology platforms, currently available commercially.

- transparent scalability
- dynamic partial reconfiguration
- 32-bit datapath width
- the rDPU can serve as routing and/or arithmetic operator
- pipe-like asynchronous inter-rDPU communication
- smart interface for data scheduling (data streams entering and leaving the FPAA)

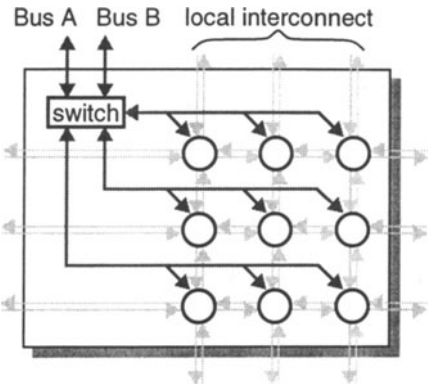

Figure 2 KrAA-III global routability.

The Novel Hierarchical Global Routing Network. The Kress Array provides three levels of routing: routing inside a rDPU (figure 1 a and b, and figure 6), nearest neighbour routing (figure 1 c and d) and a global bus (fig. 1 e) running through an entire KrAA (fig. 2), to support parallel data in- and/or output the "switch" and dual inputs Bus A and Bus B introduced with KrAA-III (fig. 2). The different levels of hierarchy are connected with reconfigurable switches. These switches allow to isolate lower levels and enable parallel inter rDPU communication of rDPUs which are not located side by side. Furthermore the switches allow to connect lower levels with only one bus to one of two parallel buses on the upper levels. Therefore parallel data access from outside the array is made possible. Thus no chip area is wasted for parallel routing resources on the lower levels. Figure 2 shows the KrAA-III chip with its inside global routing resources. The switch on the KrAA-III chip can connect each of the buses to each other. Outside the KrAA-III chip more hierarchy levels and more parallel buses are possible. As shown in figure 2 a maximum of three independent data transfers are possible inside one KrAA-III chip without additional routing resources in comparison to the rDPA. Furthermore two parallel data transfers on the global bus from outside can be performed.

The Improved Local Routing Scheme. In addition to the hierarchical global routing network the KrAA-III has nearest neighbour local routing capabilities. In contrast to the rDPA, which has only one unidirectional connection between two rDPUs, the KrAA-III has full-duplex connections. Therefore the rDPA can route data only in one direction. For the other direction data has to be routed on the global bus. With the ability of bidirectional data transfers the KrAA-III saves a lot of global bus cycles. Because all data paths are 32-bit, the local connections over chip-boundaries are serialized. This reduces the number of chip-pins dramatically.

The New Structure of the Reconfigurable Datapath Unit. The original DPU designed by Kress [Kr96] has been developed further to the rDPU with higher functionality. All arithmetic operations of the language C can be executed on the rDPU. This is supported by a small register file. Additionally, routing operations can be performed in parallel.

The configuration memory consists of four independent banks. Each bank holds a complete configuration for the rDPU. If the task manager (This is typically done by a runtime system running on the host computer.) decides to execute another task, a context switch is performed. That means that all rDPUs change the configuration memory banks. Therefore also the register file for data in the rDPU has to be implemented four

times. Otherwise all data had to be stored before a context switch is performed. This would cause too high execution times or context switches were only possible if a task had finished. Obviously a stopped and switched-out task has to be finished later, depending on its priority. To gain a further speed up out of the independent configuration banks the configuration control and the channels for configuration data are independent from each other. Thus the reconfiguration of the three idle banks can be performed in parallel to the calculations on the active bank. This is very important for the use as a general purpose accelerator, because calculations and configurations can be performed at the same time. Thus the configuration time is no longer a penalty for the accelerator. If configurations and calculations have to be performed sequentially the configuration time had to be added to the runtime and therefore some applications would not benefit from such an accelerator. A related hardware/software co-design framework (e.g. CoDe-X, [HB97a]) has to regard only the time of the first configuration and reconfiguration times that are longer than the execution times of predecessor tasks. Furthermore tasks executed several times (e.g. nested loops or recurrent functions) can stay in configuration banks permanently.

Figure 6 illustrates the routing programmability of the proposed KrAA-III architecture, which provides non-multiplexed bidirectional interconnect resources between nearest neighbour PEs. The former rDPA provides only a single unidirectional 32-bit channel between nearest neighbour PEs.

The solution illustrated by figure 6 permits to map highly irregular applications onto a regularly structured hardware platform. Figure 5 shows a mapping example based on the rDPA approach only. Eight equations expressed in C language (figure 5 a) are mapped onto the FPAA by the Datapath Synthesis System (DPSS, [Kr96], see figure 5 b). The result of this structural programming effort is the configured data path within the FPAA (figure 5 c: only internal data paths shown). The DPSS is a simulated annealing optimizer, which carries out placement and routing [Kr96]. This example also illustrates also

	characteristics or feature	systolic or wavefront array	Kress Array
1	massive pipelining: parallel, opposite directions, crossing	yes	
2	circuit layout	structured VLSI design (full custom) capable, wiring by abutment, i. e. highly area-efficient	
3	interconnect media	hardwired	programmable at 3levels
4	total interconnect pattern	regular mesh only	no restrictions: individual globally, locally & in PE
5	ensemble of PE functions	uniform only	locally individual
6	pipeline shape	linear only	free form: meandering, feed back, forks, joins, ...
7	data stream synthesis	projection (linear only)	optimizing scheduler
8	array synthesis, (or, (re-) configuration, respectively)	projection (linear only)	simulated annealing optimizer
9	applications executable	problems with regular data dependencies only	any — no restrictions
10	generate running data streams	no systematics published	smart interface available

Figure 4 Kress Array (III) - hyper generalization of the systolic array: it combines the area-efficiency and very high throughput of the systolic array with the universality of the so-called "von Neumann" machine.

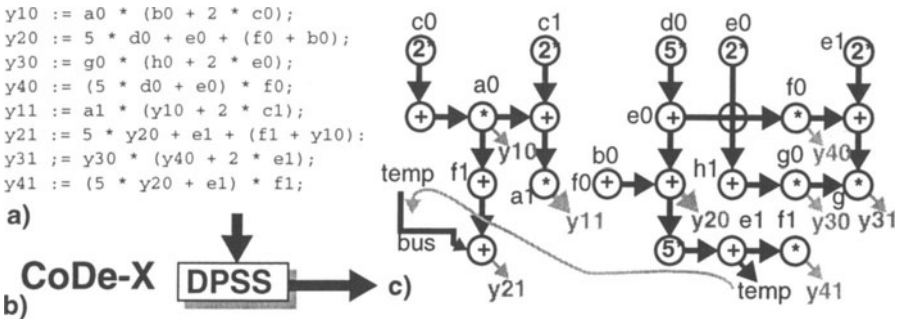

Figure 5 Application Example of an automatic and optimized KrAA Mapping.

part of the role of the CoDe-X application compiler [HB97a].

A Compiler for Structural Software. Like from an ASAP, multiple data streams enter and leave the FPAA. To organize these data streams, *Data Scheduling* [Ha95] is used. Resource scheduling within a configuration is no more needed here, since this has been already done by the placement of the DPSS. A particular data scheduler has been implemented [Ha95], which is supported by the MoM-*PDA* (Map-oriented Machine with Parallel Data Access) Xputer architecture platform and its smart memory interface.

But also a better compiler front end is required. If a hardware expert is needed to configure accelerators (as it is still necessary for FPGA based accelerators), the problem description is not really *structural software*. Structural software being really worth such a term would require a source notation like the C language and a compiler which automatically generates structural code from it. For such a new class of hardware platforms a completely new class of compilers is needed, which generates both, sequential and structural code: partitioning compilers, which separate a source into cooperating structural code (for the accelerator) and sequential code segments (for the host).

In such an environment parallelizing compilers require two levels of partitioning: host/accelerator partitioning for optimizing performance (first level) and a structural/sequential partitioning (second level) for optimizing the hardware/software trade-off of the Xputer resources. Furthermore the application development environment CoDe-X

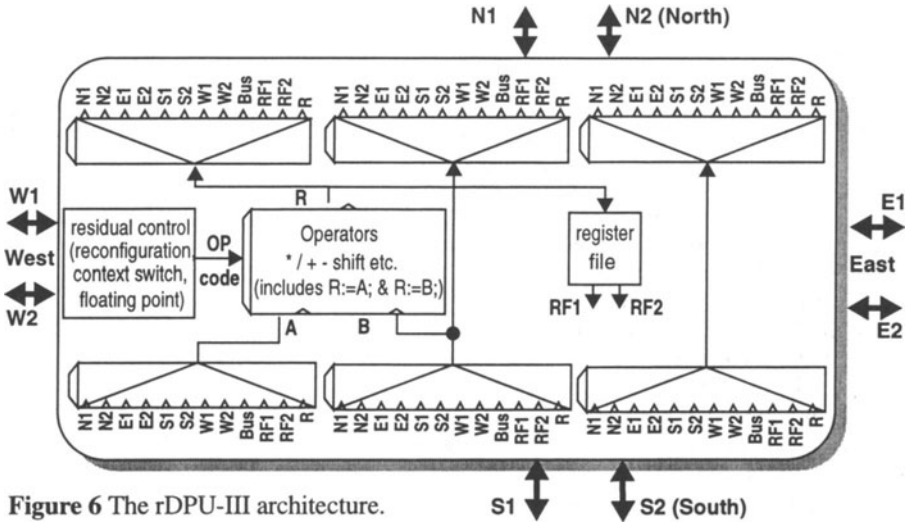

Figure 6 The rDPU-III architecture.

[HB97a] combines three programming paradigms into one more powerful approach: the control procedural paradigm reflected in C language features, the data-procedural paradigm realized in an optional language extension for specifying selected data-procedural application parts executed faster by Xputer hardware and the structural programming paradigm for the reconfigurable Xputer hardware components (FPAA). Section 5 gives a computation-intensive real world computing example. But before, we introduce the target hardware paradigm utilizing the FPAA. Since already one KrAA-III chip supports parallel data streams a data sequencing hardware is required which supports efficient parallel memory access.

3 STRUCTURED DATA SEQUENCING

In this section the data sequencing paradigm is introduced. Also the principles of the necessary hardware structures are described. Since many applications provide inherently structured data, whereas program code storage schemes are usually unstructured, we prefer to explore *data* sequencing applications. Since currently it is unrealistic to believe, that general purpose processing will switch to a new machine paradigm, we used the area of custom computing machines to try out experimental applications of the new sequencing methodology. Machines based on this new paradigm are also called Xputers [HB96].

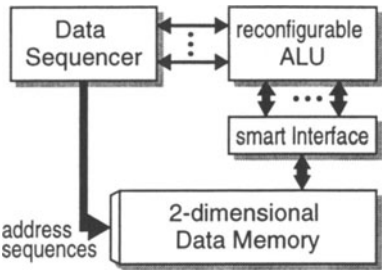

Figure 7 Basic Xputer structure.

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 [HB96]). The program to be executed is determined by a configuration of 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 caches data needed several times. Figure 7 shows all necessary components and their interconnect.

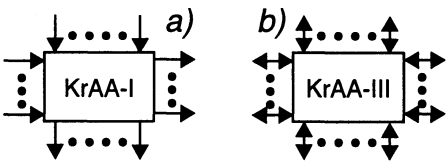

Figure 8 General data stream associated: a) unidirectional, b) bidirectional.

This principles are derived from the fact that many computation-intensive applications iterate the same operations over a large amount of data. Xputers accelerate them 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. The rALU is implemented with the KrAA-III [Ha95]. Computations are performed by applying a configured complex operator on the data, called compound operator.

This hardware structure has further the big advantage that, if the smart interface is integrated into the rALU, the rALU can be changed easily without modifying the whole machine. Therefore other FPAA or FPGA based rALUs can be used. The residual control between data sequencer and rALU is only needed when the data stream has to be influenced by the result of previous computations. Even the whole software environment except the configuration code generation for the rALU stays the same, when the rALU is

changed [HB97a].

To clarify how operations are executed the execution model for Xputers is pictured in figure 9. 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, which is called data map. The part of the data memory which holds the data for the current iteration is determined by a so called scan window, which is a model for the smart interface holding a copy of the relevant data of the memory. Each position of the scan window is marked as read, write or read and write. The location of the scan window is determined by a designated point, called handle (see figure 9). 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 pattern is the main control mechanism of an Xputer. Because of their regularity, scan patterns can be described by only a few parameters.

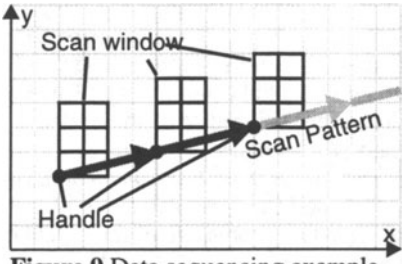

Figure 9 Data sequencing example.

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. In 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. On this level the data scheduling described in section 2 is performed.

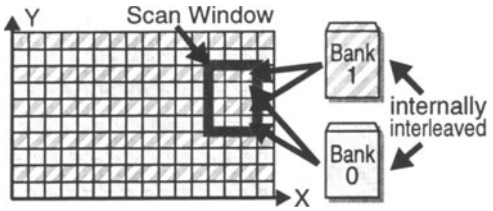

Figure 10 Memory distribution.

4 XPUTER DATA SEQUENCER HARDWARE

The data sequencer is a specialized hardware for generic address generation with a small parameter set. This chapter explains the hardware realization and the underlying concepts. First, the physical memory organization for the 2-dimensional model is shown.

The Memory Organization. Since we have 2-dimensional memory accessible through a 2-dimensional scan window the memory can easily be cut in slices assigning the rows to different memory banks (figure 10). With this organization, access to different rows in parallel is made possible. This is done at the second level of the 2-level data sequencing. Depending on the handle position the hardware has to determine which row inside the scan window is assigned to which memory bank. There are n parallel memory banks possible but to simplify the explanation only 2 parallel memories are illustrated. Each of the parallel memory banks is physically assigned to one of the parallel KrAA-III buses (figure 2).

The data map for a task can be any rectangle of any size. Several tasks of different data maps may be mapped onto the physical memory at the same time. A special hardware is required to avoid big areas of unused memory. Because the memory is organized 2-

dimensionally, it has to be mapped by the data sequencer to commercially available 1-dimensional memories.

The Data Sequencer Hardware. The data sequencer hardware can be divided into a central control unit and an address generation data path. This 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. The third part of the data path is a *Memory Map* (MemM) function followed by the *Burst Control Unit* (BCU) which generates the memory control lines. The complete structure is pictured in figure 11.

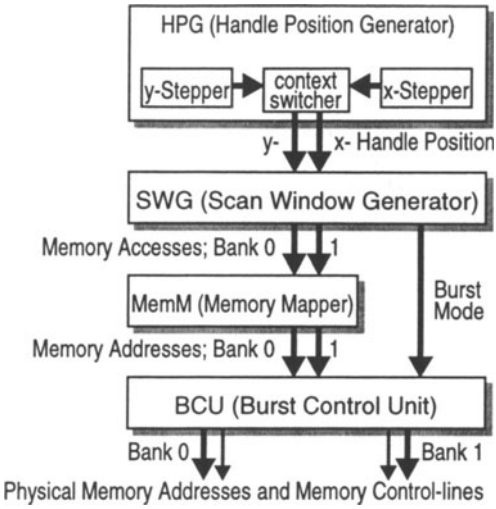

Figure 11 The Data Sequencer Datapath.

As illustrated in figure 11, the number of memory banks affects only the second pipeline stage. The handle position is the same for all parallel memories. Though the Memory Map function and the Burst Control Unit are instantiated for every memory bank, their structures are not influenced. To have this structure scalable, an FPGA implementation is considered.

As the KrAA-III structure provides context switching for multitasking applications and for fast task switching through simultaneous configuration, the data sequencer has to extent this functionality. This is because of large FPAAs can be configured with several small tasks and process them in parallel. For this the data sequencer datapath provides a deep stack to store the parameter sets of all active tasks running in parallel as well as of all tasks switched out on the idle context layers of the KrAA-III. The size of the stack benefits from the small parameter sets required for generic address generation. The control of the tasks executed on the active layer of the KrAA-III is performed by the central control unit of the data sequencer. Context switches on the KrAA-III and the required change of the parameter sets inside the data sequencer datapath are initiated by the runtime system running on the host computer.

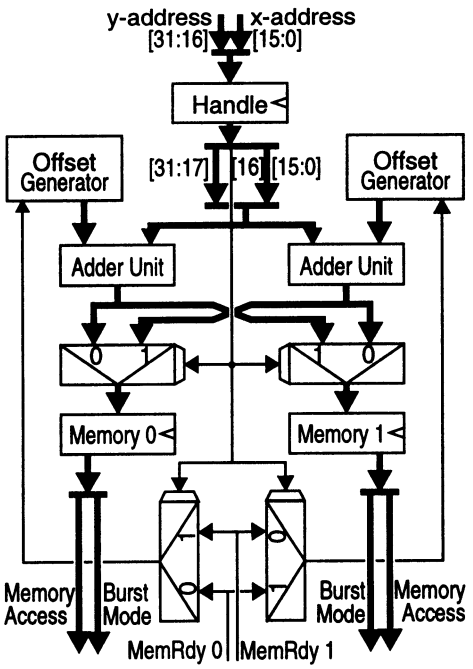

Figure 12 The Scan Window Generator.

The Handle Position Generator. The HPG performs the first level of the 2-level data sequencing. It provides two identical step-

per units for the x- and y-address generation [HB97b]. Further there is a context switcher unit (figure 11) which adds an offset address for each scan pattern. This provides the possibility for several data maps at the same time in the physical memory. The entire stepper hardware is designed to store several parameter sets for nested or meshed scans and for independent scans running in parallel. Therefore the scan parameters have to be exchanged very fast. Most of the parameters are constants which have only to be read. Only the actual position of the slider and an offset address for relative scans have to be stored as interim results. Therefore a parameter stack is required for each address generator.

The Scan Window Generator. The position of the scan window is determined by the handle position generated by the HPG. The SWG (figure 12) has to access all memory locations inside the scan window (see figure 9 and figure 10). This is realized by adding offsets to the handle position. Further the flags for read and write operations have to be set. If the data memory supports burst read or write operations, this is initiated. Since several physical memories are accessed in parallel all these actions have to be done in parallel. All parameters for second level of the 2-level data sequencing are provided by offset generators (figure 12). They are stored in a look-up tables and sequenced in succession by a finite state machine. The look-up tables provide capacity for 16 different scan windows at the same time. The switching between different tasks is initiated by the central control unit. Further the SWG has to determine which adder unit is assigned to which memory bank, depending on the handle position. Since the 2-dimensional memory is cut in rows, this can be easily done by evaluating the LSB of the y-address. The SWG pictured in figure 12 supports 2 parallel memories and has to be extended for more parallel banks.

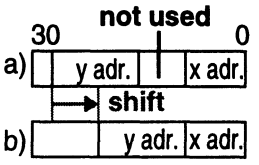

Figure 13 Address mapping of the Memory Mapper.

The Memory Mapper. All memory banks are commercially available 1-dimensional memories. In cases where the number of rows in the memory map exceeds the number of parallel memory banks, several rows have to be mapped onto one memory. If only one memory bank exists, all rows are mapped onto this bank. In that case only a 2-dimensional visualization of the traditional 1-dimensional memory is performed. To have several rows of the memory map mapped onto the same physical memory device the x- and y- addresses have to me merged to one physical memory address. To eliminate unused memory areas in the physical memory the output addresses are shifted according to the exact size of the actual data map, i.e. not every application requires the complete address range of 16 bits. Often some leading bits of the x and y addresses are unused. This results in different data map sizes and shapes. Therefore simply merging the 16 bit x- and y-address to a 32 bit physical memory address would cause a waste of memory, as both 16 bit addresses do hardly exploit the 16 bits. The leading zeros of the x-address are the reason for this waste as can be seen in figure 13a. Therefore an additional shift operation is implemented in the Memory Mapper unit. The exact number of shifts depends on the application. It is known and fixed at compile time. If there are several tasks in the physical memory the context switcher in the HPG secures that there is no memory violation. Because the memories have to be accessed in parallel this hardware is required for each memory bank.

The Burst Control Unit. Since interleaved memory with burst options is supported an additional unit has to control the burst operations. The required signals for variable burst lengths are generated by the Burst Control Unit. Because the memories have to be accessed in parallel this hardware is required for each memory bank.

5 A REAL WORLD COMPUTING APPLICATION: AUTOMOTIVE COLLISION DETECTION

An embedded accelerator example for real-time automotive collision detection has been described in [Ha96]. Because of the new parallel memory access and context switch features of the data sequencer and the KrAA-III, this application gains a further speed-up. Therefore the improved version of this approach is introduced.

The availability of high-dynamic-range CMOS image sensors (HDRC) with random access to the pixel array serves as a prerequisite to automotive applications like automatic collision prevention. A sensor with a high dynamic range is necessary to avoid the saturation known from CCD cameras in the case when a car leaves a tunnel, where the sensors have to be able to detect an approaching car against the bright sunlight. The real time analysis of the incoming data can be done more efficiently, if a random access to the sensor array allows to concentrate on the interesting portions of the image, which greatly reduces the data transfer rates required between the sensor and the image processing hardware. The design of a hardware to perform such a collision detection can be seen in figure 14.

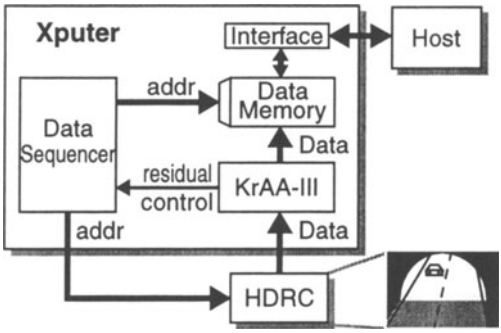

Figure 14 Automotive collision detection using Xputer accelerator.

Since the HDRC can be accessed like a regular memory device, its address lines are directly connected to the data sequencer. The data output is connected to one of the KrAA-III buses. Though the KrAA-III buses are bidirectional they are pictured directed in figure 14 to illustrate the flow of data in this example. The second KrAA-III bus is connected to the data memory. Because of the HDRC provides only one databus parallel memory access is only possible for parallel HDRC read and data memory write operations. The according memory organization is illustrated in figure 15. This corresponds to the memory organization in section 4 where the second memory bank is replaced by the HDRC.

The way operations are performed is as follows. The Xputer data sequencer computes the address sequences for the HDRC and the data memory. On the sequenced data the KrAA-III performs an edge detection by applying a two-dimensional FIR filter with appropriate weights. The filter equation is shown in figure 16. The k_{ij} are the coefficients of the filter and the w_{ij} contain the data at the corresponding scan window positions (grey fields in figure 15). The result

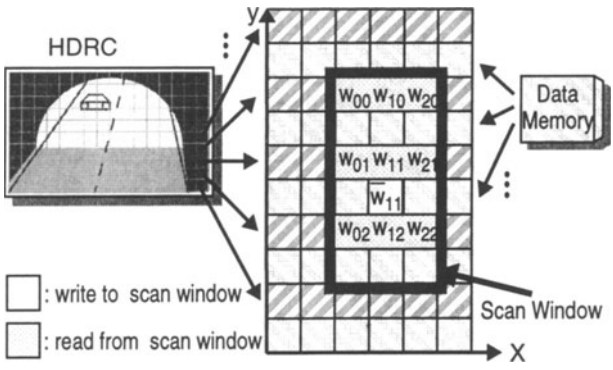

Figure 15 Memory organization for automotive collision detection example.

$\overline{w}_{11}$ is written back into the scan window (white field). This scan window position is placed in another row assigned to the data memory. Figure 17 shows the equation mapped into the KrAA-III. The input registers of the rDPUs named with w_{ij} and the output register named with $\overline{w}_{11}$ represent the scan window positions. In figure 17 the input registers named with k_{ij} represent the registers with coefficients of the FIR filter loaded at configuration. For more details see [Ha96].

The pixel array of the HDRC image sensors is accessed directly by the data sequencer (figure 14). First the whole pixel array is scanned with a video scan to detect the borders of the street. Dependent on their location data dependent scans are performed, which corresponds to local video scans along these borders. So only the relevant parts of the image are read. The KrAA-III also transfers data to the data sequencer for controlling these data dependent scans (residual control, figure 14 or figure 7) [AH94]. Always after 5 frames of scanning the relevant street borders, a video scan over the complete image is performed by the data sequencer and the edge detection algorithm is applied. To process the complete image is necessary for identifying approaching cars etc. (See image analysis task of the host below.)

Because of a different algorithm is required, a context switch is performed. As a result no reconfiguration is required and no hardware resources have to stay idle. (One of this two points would be the case if a FPAA with no context switch capabilities were used.) The data memory is also accessible by the host for data exchanges with the embedded accelerator. After this image preprocessing the host CPU performs the image analysis tasks, where the edges have to be combined to objects. These are identified to detect obstacles, like an approaching car or the borders of the street, for example. Since these algorithms do not reveal address patterns known at compile time, they can be handled with less overhead by a von Neumann processor than by a data sequencer. If the car would go out of the borders (street), for example a controller connected to the steering mechanism of the car could correct the wrong way. Also if an approaching car is identified, this controller could react in a suitable way.

$$\begin{aligned}\overline{w}_{11} = {} & w_{22}k_{22} + w_{12}k_{12} + w_{02}k_{02} \\ & + w_{21}k_{21} + w_{11}k_{11} + w_{01}k_{01} \\ & + w_{20}k_{20} + w_{10}k_{10} + w_{00}k_{00}\end{aligned}$$

Figure 16 Equation of 3 by 3 FIR filter.

The hardware for the image preprocessing might be a FPAA like the rDPA [Kr96], the KrAA-III or the reconfigurable data-driven multiprocessor architecture proposed in [YR93]. Generally FPAAs are superior for the image preprocessing over special purpose hardware with regard to flexibility in the choice of algorithms to be run. Using the MoM-*PDA* (Map-oriented Machine with Parallel Data Access) and a host allows to perform the image analysis in a pipelined fashion, where the image acquisition, the image preprocessing, and the image analysis are done in parallel in different stages of the pipeline. This does not even increase the latency from the image acquisition to the collision detection, because all these steps would have to be done sequentially as well in a non-pipelined fashion, only at lower overall data rates or higher bandwidth requirements. The versatility of the scan patterns allows to run a wide variety of image preprocessing algorithms in an endless loop without requiring any download of parameters after the initial power-up configuration.

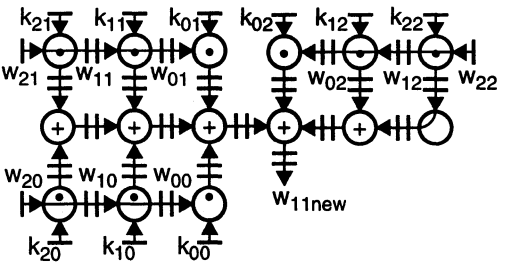

Figure 17 Two dimensional FIR filter mapped into the KrAA-III.

6 CONCLUSIONS

The use of the data driven Xputer architecture as an embedded reconfigurable accelerator for real world computing and many other applications has been presented. The data driven mechanism has been realized by a novel kind of sequencer hardware. This data sequencer is superior to the address generation units found in digital signal processors as well as other separately available address generation devices on the market (e.g. HSP45240 [Ha92]). It provides the support to run complete image and signal processing algorithms on structured data without requiring an update of parameters after an initial configuration, even though such an update is possible. Multiple tasks can be executed on a data sequencer. The usefulness of the novel universal accelerator approach has been illustrated by a real time application example of automatic protection a car from leaving the lane and collision prevention.

The novel KrAA-III has been explained in detail and several new mechanisms (improved rDPU, context switch mechanism and hierarchically routing structure) for speed up have been introduced. Especially the context switch mechanism recommends the KrAA-III architecture because of the ability to keep several tasks resident simultaneously. Furthermore reconfigurations of idling configuration layers can be performed in parallel to calculations of an active layer.

This general purpose accelerator approach promises high speed-up factors to a cheap price for a wide variety of computation intensive applications in real world computing and other embedded systems as well as in desk top and other in scientific computing [AH94].

7 REFERENCES

[AH94] A. Ast, R. Hartenstein, et al.: A General Purpose Xputer Architecture derived from DSP and Image Processing. In Bayoumi (Ed.): VLSI Design Methodologies for Digital Signal Processing Architectures, Kluwer Academic Publishers 1994

[De96] A. DeHon: Reconfigurable Architectures for General-Purpose Computing; Technical Report 1586, MIT Artificial Intelligence Laboratory, Sept., 1996

[Ha92] N.N.: Harris Semiconductor Data Book; pp. 6,3-6,15, Harris Corp., 1992

[Ha96] R. Hartenstein, et al.: An Embedded Accelerator for Real Time Image Processing; 8th EUROMICRO Workshop on Real Time Systems, L'Aquila, Italy, June 1996

[Ha95] R. W. Hartenstein, et al.: A Scalable, Parallel, and Reconfigurable Datapath Architecture; Sixth International Symposium on IC Technology, Systems & Applications, ISIC'95, Singapore, Sept. 6-8, 1995

[HB97a]R. Hartenstein, J. Becker: A Two-level Co-Design Framework for data-driven Xputer-based Accelerators; Proc. 30th Ann. Hawaii Int'l Conf. on System Sciences (HICSS-30), Wailea, Maui, Hawaii, Jan. 7-10, 1997

[HB97b]R. Hartenstein, J. Becker, M. Herz, et al.: A Novel Sequencer Hardware for Application Specific Computing; Proc. of 11th International Conference on Application-specific systems, Architectures and Processors, Zurich, Switzerland, July ,'97

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

[Kr96] R. Kress: A fast reconfigurable ALU for Xputers; Ph. D. Thesis, Kaiserslautern University, 1996

[RW97] N.N.: Proc. of the 1997 Real World Computing Symposium, Tokyo, Japan, '97

[YR93] A. Yeung, J. Rabaey: A Reconfigurable Data-Driven Multiprocessor Architecture for Rapid Prototyping of High Throughput DSP Algorithms; Proc. 26th Hawaii Int'l. Conf. on System Sciences, Jan. 1993