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

A Novel Sequencer Hardware for Application Specific Computing

Reiner W. Hartenstein, Jürgen Becker, Michael Herz, Ulrich Nageldinger

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

Abstract

This paper introduces a powerful novel sequencer for controlling computational machines and for structured DMA (direct memory access) applications. It is mainly focused on applications using 2-dimensional memory organization, where most inherent speed-up is obtained thereof. A classification scheme of computational sequencing patterns and storage schemes is derived. In the context of application specific computing the paper illustrates its usefulness especially for data sequencing - recalling examples hereafter published earlier, as far as needed for completeness. The paper also discusses, how the new sequencer hardware provides substantial speed-up compared to traditional sequencing hardware use.

1. Introduction

In many computing application areas, like image processing, digital signal processing, multi media and others, a lot of software to hardware migration ideas have been implemented for acceleration by add-on data path hardware as well as application specific computing machines. Still, the one-dimensional memory address space is practically the only basis of execution organization. We are also not aware of any common DMA concepts other than based on this one-dimensional address space.

Two-dimensionally organized memory address space has shown to be useful for many computing applications [HR95]. Therefore a novel sequencer hardware is introduced, which supports two-dimensionally organized memory address space or at least the two-dimensional visualization of the traditional one-dimensional address space. The structure of two-dimensionally organized memories allows to introduce parallel memory access without any consistency problems. An example illustrates how two-dimensional parallel accessible memory can be arranged for application specific computing.

This sequencer hardware is an alternative for controlling computational machines and for structured DMA. Memory address sequences are generated generically, controlled by a few parameters resulting in very short configuration (programming) times. Therefore the sequencer does not need any memory cycle for address calculations. Examples have been published where up to 90% of computation time of regular von Neuman machines elapses for calculation of data addresses [HHS91].

The paper introduces the principles and the design of a novel class of this sequencer hardware derived from a classification scheme of computational sequencing patterns and storage schemes. In the next section the idea of structured data sequencing is introduced. In Section 3, access patterns for 2-dimensional memories and a classification scheme are introduced. In section 4, the novel sequencer hardware is presented and in the last section, its use with a high performance data sequencing example is demonstrated.

2. Structured Data Sequencing

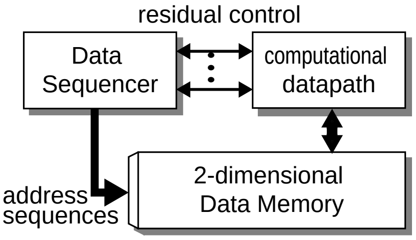


Figure 1. Basic use of the data sequencer.

In this section the data sequencing paradigm is introduced. 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 [HBH96]). 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 generically by a data sequencer. The accessed data is passed from the data memory to the computational datapath and back. Figure 1 shows all necessary components and their interconnect.

This hardware structure has the big advantage that the datapath can be changed easily without modifying the whole machine. The residual control between data sequencer and datapath 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 datapath stays the same, when the datapath is changed [HB97].

To clarify how operations are executed the execution model is pictured in figure 2. 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 the current iteration of a loop. These arrays can be mapped onto a 2-dimensional organized memory. This arrangement of data 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. Each position of the scan window is marked as read, write or read and write. The location of the scan window is determined by the lower left corner, called handle (see figure 2). Operations are performed by moving the scan window over the data map and applying the compound operator of the datapath onto the data in each step. This movement of the scan window, called scan pattern, is the main control mechanism. Because of their regularity, scan patterns can be described generically by only a few parameters. In fact the execution model realizes a 2-level data sequencing. In the first level with the handle position of the scan window all data for one iteration is indicated. On hardware level this position is determined by the x- and y-addresses of the scan pattern. In the second level the data is read from the scan window into the datapath and written back after the computation step. On hardware level the data sequencer computes physical memory addresses for each scan window element.

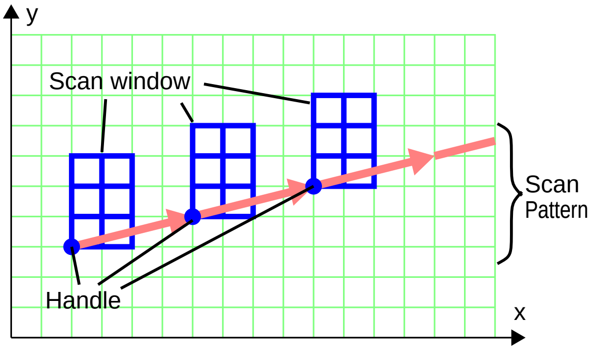


Figure 2. Execution model used by the data sequencer.

Before consolidation of this method for computing with processor arrays, an implementation of the data sequencer is introduced. To justify the proposed implementation a classification of scan patterns is given first.

3. Scan Patterns

To build a high performance sequencer hardware, first different types of scan patterns have to be determined. It is necessary to find suitable classes of scan patterns to have only a small parameter set for their description. This is very important for reconfigurable systems because a small parameter set requires less configuration time. Since there is 2-dimensional memory always more than a single step or one linear scan, i.e. a straight, one-dimensional sequence of steps, is required to make use of it. Therefore as the atomic element of the classification the video scan is used rather than the linear scan (see Table 1 (I)). This is a regular scan, which goes in both directions over the 2-dimensional memory. Regular means, that the scan has a fixed step width for each direction. Note, that both single steps and linear scans are obviously a subset of the video scans. The so-called slider model is used to introduce the parameters required for the generic generation of the class of video scan patterns.

3.1 The Slider Model

The slider model describes generic address generation for one dimension in the address-/time-space. To obtain a 2-dimensional video scan the slider model has to be used for both dimensions. First the slider model for one dimension is introduced and later the combination for two dimensions is explained.

With the slider model linear scans are generated gradually in a 1-dimensional address space. Therefore five sliders and according parameters are introduced: Base (B), Limit (L), Floor (F), Ceiling (C) and Address (A). Each time a linear scan is generated (i. e. the Address slider performs one scan line) the Address is initialized at the Base and addresses are generated in ΔA steps till the Limit is reached. Before the first scan line can be performed the Base is placed at B_0 and the Limit is placed at L_0. Each time one scan line is completed, Base is moved by ΔB and Limit is moved by ΔL. If Base meets or passes Floor or if Limit meets or passes Ceiling the address generation is finished. This method is illustrated in figure 3.

For the generation of 2-dimensional scan patterns x- and y-addresses are needed. For each dimension a separate Address slider based on the slider model performs the address calculations. There are three separate modes for synchronization:

- Synchronous: at each time step every Address slider performs one operation step. (Table 1(Ia))
- y wait x: alternately the x-Address slider performs a complete scan line and the y-Address slider one step. (Table 1(Ib))
- x wait y: alternately the y-Address slider performs a complete scan line and the x-Address slider one step. (Table 1(Ic))

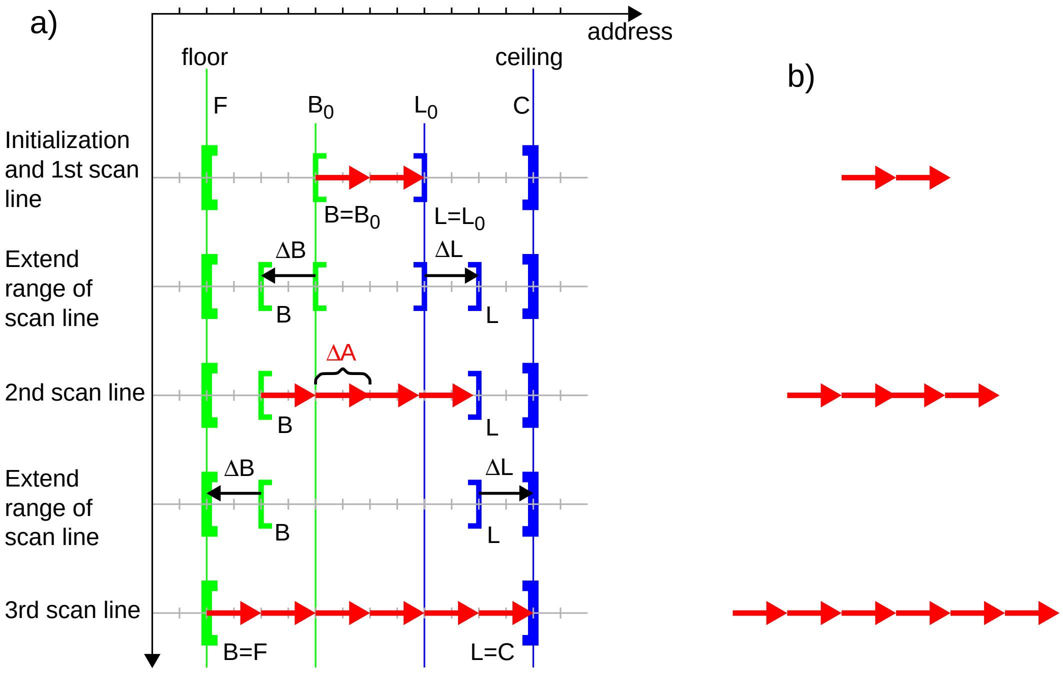


Figure 3. Illustration of the slider model (a) and generated scan sequence (b).

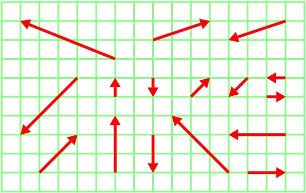
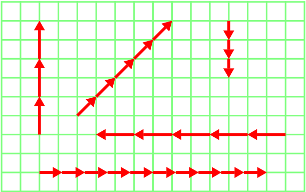
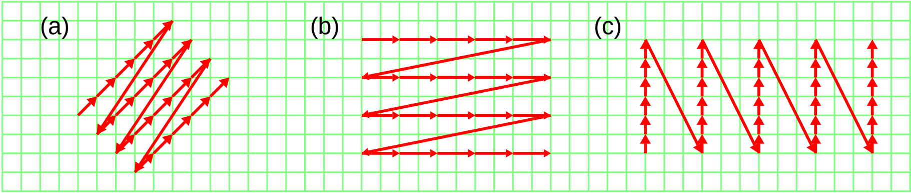
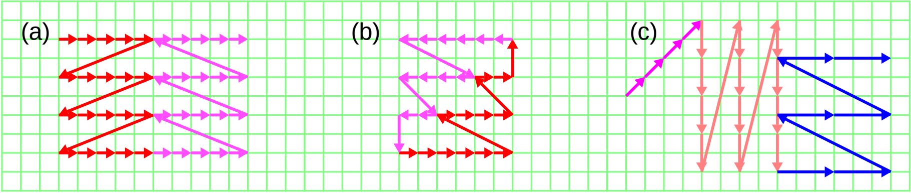
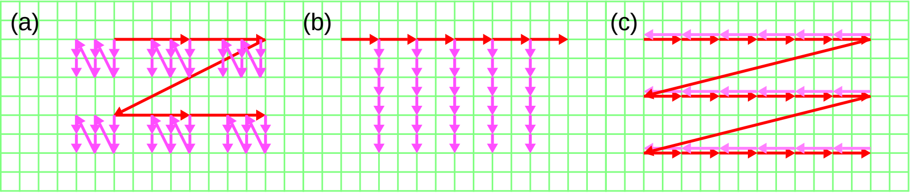
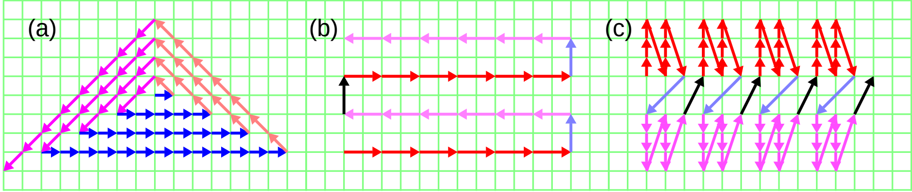
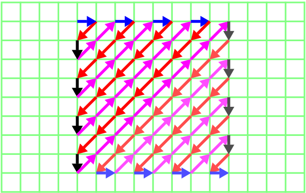
Classes	Examples
single steps	
linear scans	
(I) video scans	
(II) compound scans	
(III) nested scans	
(IV) meshed scans	
(V) complex scans	

Table 1. Scan pattern classes.

For each mode the video scan is finished when one designated Address slider is finished. For the address slider without termination condition also infinite configurations are allowed. With the slider model a complete video scan can be described by only 15 parameters: the synchronization mode and for both dimensions Floor (F), initial Base (B_0), step width for Base (ΔB), Ceiling (C), initial Limit (L_0), step width for Limit (ΔL) and step width for the Address steps (ΔA).

3.2 Classification of Scan Patterns

To produce more scan patterns multiple video scans can be combined. Doing this, four important classes can be pointed out, which are introduced in this section. For the following classes the parameter set of the video scan is extended by a scan characteristics parameter which designates when a scan is stopped or finished and what to do then. Therefore the count of parameters of the following scans is the sum of the parameters of the combined video scans.

Compound Scans. This class contains all scan patterns, which can be formed by concatenation of complete video scans, i.e. each video scan has to be finished before the next is started. Each successor scan has the last position of his predecessor scan as starting position. A compound scan is finished when the last video scan in the concatenation is finished. Table 1(II) gives some examples for compound scans. Obviously the class of compound scans contains all 2-dimensional scan patterns, because each single step is also a video scan. By forming concatenations all scan patterns can be build. But, because of configuration times having the number of parameters in mind, only *short* concatenations are interesting.

Nested Scans. Another important class of scan patterns are the scans where an outer video scan performs only one step or a scan line and then an inner video scan is performed completely each time (see table 1(III)). All parameters of the inner video scan have to be relative to the actual position of the outer video scan. But this does not increase the number of parameters of the inner video scan. There is a *relative* flag provided in the scan characteristics parameter.

Meshed Scans. The class of meshed scans are all scans where at least two video scans are concatenated similar to compound scans. But in contrast, at least one of the single scans performs only one scan line before the next one starts, rather than executing to completion. The whole sequence of scans is then repeated this way until one designated scan is finished. Each video scan except the first step of the first video scan starts his scan line at the last position of the predecessor video scan. Table 1(IV) shows 3 examples for meshed scans.

Complex Scans. To finish the classification a last class of scan patterns called complex scans is introduced. Complex scans are all arbitrary combinations of above classes. How a complex scan is combined out of the above classes is shown by the Zig-zag enumeration example for the run length coding step in JPEG image compression method [HBK95]. For this, compound scan consisting of two meshed scans is needed. Each meshed scan contains four video scans (see figure 4).

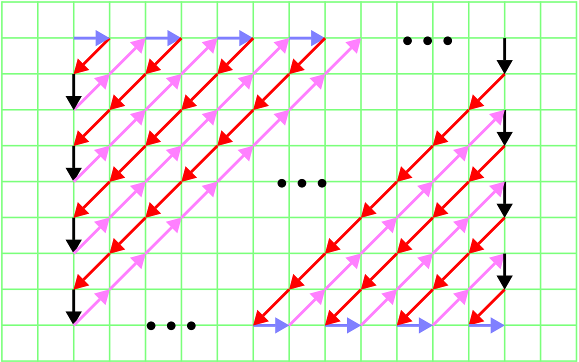


Figure 4. Scan pattern example: the JPEG Zig-zag enumeration.

To process a complete picture the compound scan for the Zig-zag enumeration has to be nested in an additional video scan covering the whole picture. All parameters of the above compound scan (and the inner meshed scans) have to be relative. Figure 5 displays the complete scan pattern. Since there are four video scans for each meshed scan, two meshed scans compound (=8 video scans) and this compound scan nested in an outer scan (plus one video scan), 9 video scans with 16 parameters for each are necessary. Therefore the run length coding step of JPEG image compression method requires 144 parameters.

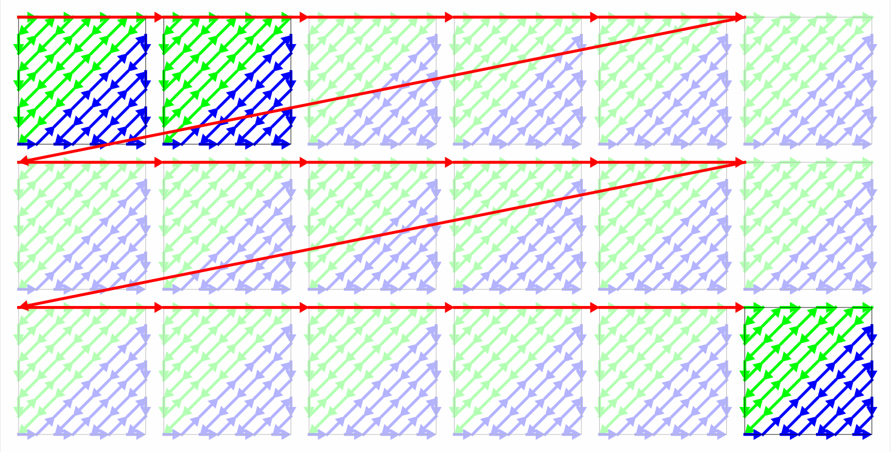


Figure 5. Run length coding scan for a 48x24 pixel image in JPEG image compression method.

The number of parameters does not depend on the size of the image. Furthermore this method has the big advantage that the parameters can be easily scaled to different image sizes. In the JPEG example only the two Ceiling values (see figure 3) of the outer video scan (figure 5) have to be set with the x- and y-size of the image. All other parameters stay the same for this application.

3.3 Summary of the Classification

The video scan as the basic scan pattern is obtained out of the classification. Single steps and linear scans are subsets of video scans. Further there are 3 important classes of combinations of the video scan. These are the compound scan, the nested scan and the meshed scan. It can be proven that the class of compound scans contains all arbitrary scans but the nested and meshed scans give the opportunity to describe important scans much more efficiently. The last class contains all combinations of compound, nested and meshed scans.

4. Xputer Data Sequencer Structure and Memory Interface

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

4.1 Memory Organization

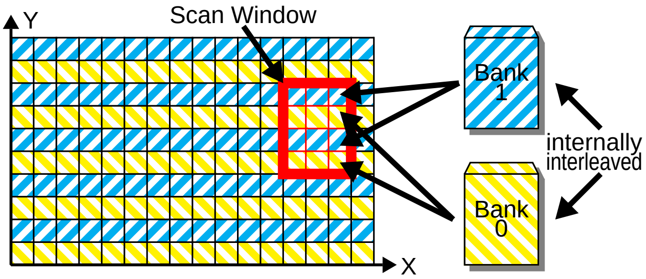


Figure 6. Memory distribution.

Since there is 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 6). With this organization accessing different rows in parallel is possible. Depending on the handle position the hardware has to determine which row inside the scan window is assigned to which memory bank. This is done at the second level of the 2-level data sequencing. There are n parallel memory banks possible but to simplify the explanation only 2 parallel memories are considered in this paper.

The data map for a task can be any rectangle of any size. Several tasks with 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.

4.2 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 7.

As illustrated in figure 7, 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 the data sequencer structure scalable, an FPGA implementation is considered.

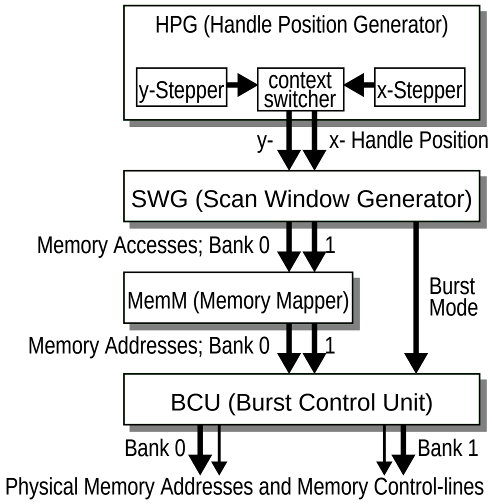


Figure 7. The Data Sequencer address generation datapath.

Handle Position Generator. The HPG performs the first level of the 2-level data sequencing. It provides two identical steppers for the x- and y-address generation according to the slider model. Further there is a context switcher unit (figure 7) 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 memory is required for each stepper.

Scan Window Generator. The position of the scan window is determined by the handle position generated by the HPG. The SWG (figure 8) has to access all memory locations inside the scan window (see figure 2 or figure 6). 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. All parameters for second level of the 2-level data sequencing are provided by the offset generator (figure 8). They are stored in a look-up table and sequenced in succession by a finite state machine. The look-up table provides capacity to hold 16

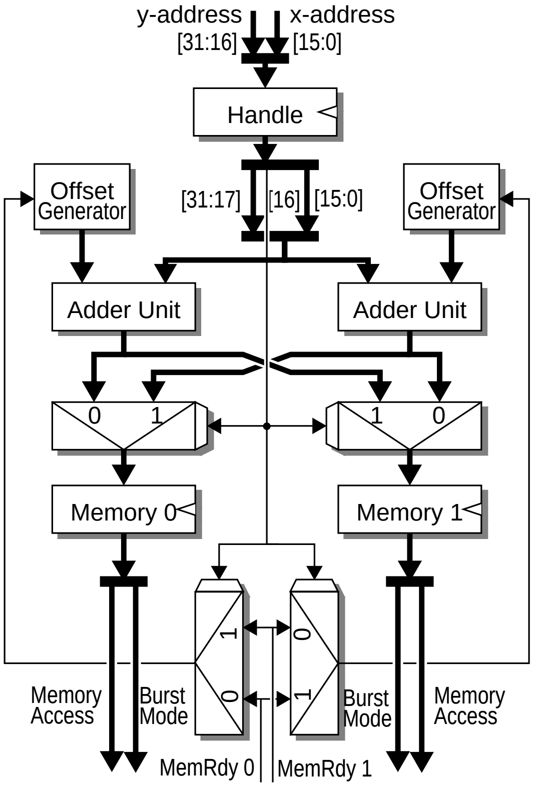


Figure 8. The Scan Window Generator.

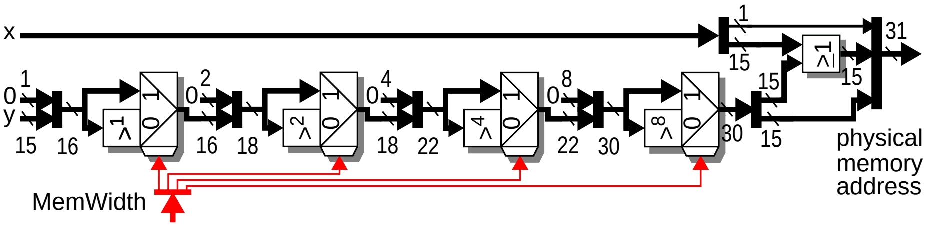


Figure 10. The Memory Mapper.

different scan windows at the same time. The switching between different tasks is initiated by the central control unit (figure 7).

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 8 supports 2 parallel memories and has to be extended for more parallel banks.

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. This is done by merging the two 16 bit addresses for x- and y-location to one 32 bit address (see figure 9a). When only one memory bank exists, all rows are mapped onto this bank. Consequently in that case only a 2-dimensional visualization of the traditional 1-dimensional memory is performed.

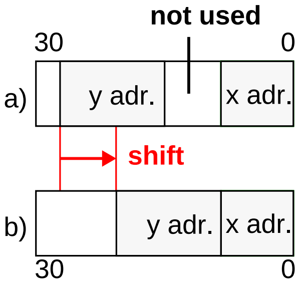


Figure 9. Address mapping of the Memory Mapper.

Not every application requires the complete address range of 16 bits. Often some leading bits of the x and y addresses are unused (figure 9a). 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 9a. Therefore an additional shift operation is implemented (figure 10). The number of bits to be shifted depends on the application (figure 9b). It is known and fixed at compile time. If there are several tasks in the physical memory the context switcher in the HPG (figure 7) secures by adding an offset that there is no memory violation.

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 High Performance Data Sequencing Example

In this chapter the use of the novel data sequencer hardware will be demonstrated with a classic example in application specific array processing. For this, the outer connections of a processor array (e.g. the Kress ALU Array, KrAA, [Har94]) are connected directly with memory modules like demonstrated in figure 11. All address lines of the memory modules are connected to one data sequencer. The 2-dimensional address space is assigned to the memory banks like illustrated in figure 6. Therefore the *Scan Window Generator* has to be

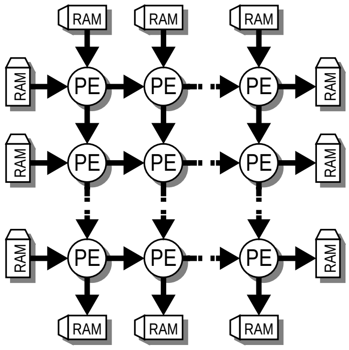


Figure 11. Configuration of an array with processing elements (PE) connected to parallel memory banks.

$$\vec{c} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

whereas:

$$\begin{aligned} c_1 &= a_{11} \cdot b_1 + a_{12} \cdot b_2 + a_{13} \cdot b_3 \\ c_2 &= a_{21} \cdot b_1 + a_{22} \cdot b_2 + a_{23} \cdot b_3 \\ c_3 &= a_{31} \cdot b_1 + a_{32} \cdot b_2 + a_{33} \cdot b_3 \end{aligned}$$

Figure 12. Matrix vector multiplication.

adapted to the number of memories and the *Memory Mapper* and the *Burst Control Unit* have to be instanced for every memory module.

As an application example the matrix-vector multiplication is chosen. It is assumed, that the PEs can be configured to execute the necessary function (see below or figure 13). Furthermore, unused PEs can be programmed to serve as routing elements. For demonstration purposes the formula in figure 12 is chosen.

For implementation a 3 by 3 processor array is sufficient. Each array processor implements the function $c_i'' = c_i' + a_{ij} \cdot b_j$. For a pipelined execution the data has to be scheduled like pictured in figure 13. The lower index indicates the position of the vector elements in the vector and the upper index indicates the number of matrix-vector multiplication. This is because several matrix-vector multiplications are pipelined. While the matrix elements are configured into the processor elements, the source vector elements b_j are sequenced into the processor array and the resulting vector elements c_i out of the array. The interim results c_i‘ are passed between the array processors. The implementation of the processor element is shown in figure 13.

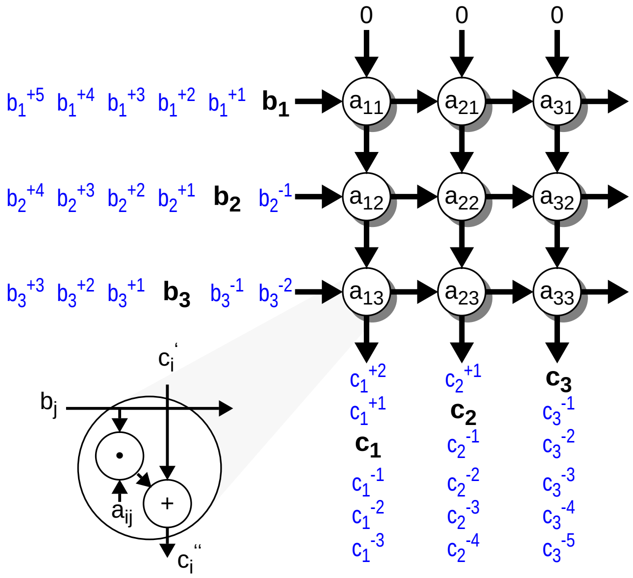
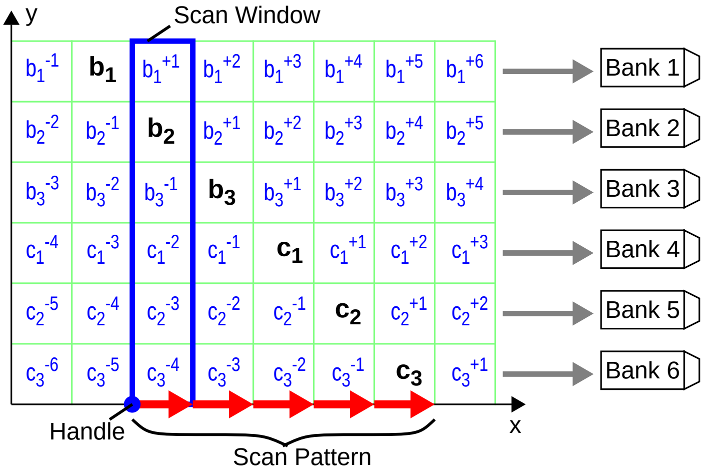

0
0
0
b_1^{+5} b_1^{+4} b_1^{+3} b_1^{+2} b_1^{+1} b_1
b_2^{+4} b_2^{+3} b_2^{+2} b_2^{+1} b_2 b_2^{-1}
b_3^{+3} b_3^{+2} b_3^{+1} b_3 b_3^{-1} b_3^{-2}
a_{11}
a_{21}
a_{31}
a_{12}
a_{22}
a_{32}
a_{13}
a_{23}
a_{33}
c_1^{+2} c_1^{+1} c_1 c_1^{-1} c_1^{-2} c_1^{-3}
c_2^{+1} c_2 c_2^{-1} c_2^{-2} c_2^{-3} c_2^{-4}
c_3 c_3^{-1} c_3^{-2} c_3^{-3} c_3^{-4} c_3^{-5}
c_i‘
b_j
a_{ij}
+
c_i“

Figure 13. Configuration of a 3 by 3 processor array for a matrix-vector multiplication example with data streams.

In order to distribute each b_j and each c_i to the correct memory bank, they have to be placed in the assigned row of the 2-dimensional data map (see figure 14). To have the required data sequenced at the correct time step, the vector elements of one iteration are distributed to several columns. Note that only the b_j elements are source data and read from data memory. The c_i elements are written after calculations. The scan window holds always 6 vector elements of 6 different matrix-vector multiplications in the pipeline.

As a result of the parallel memory banks all data input and data



b_1^{-1}	b_1	b_1^{+1}	b_1^{+2}	b_1^{+3}	b_1^{+4}	b_1^{+5}	b_1^{+6}	Bank 1
b_2^{-2}	b_2^{-1}	b_2	b_2^{+1}	b_2^{+2}	b_2^{+3}	b_2^{+4}	b_2^{+5}	Bank 2
b_3^{-3}	b_3^{-2}	b_3^{-1}	b_3	b_3^{+1}	b_3^{+2}	b_3^{+3}	b_3^{+4}	Bank 3
c_1^{-4}	c_1^{-3}	c_1^{-2}	c_1^{-1}	c_1	c_1^{+1}	c_1^{+2}	c_1^{+3}	Bank 4
c_2^{-5}	c_2^{-4}	c_2^{-3}	c_2^{-2}	c_2^{-1}	c_2	c_2^{+1}	c_2^{+2}	Bank 5
c_3^{-6}	c_3^{-5}	c_3^{-4}	c_3^{-3}	c_3^{-2}	c_3^{-1}	c_3	c_3^{+1}	Bank 6

Figure 14. Scan window, scan pattern and data map storage scheme for the matrix-vector multiplication example.

output lines of the processor array can be served in parallel. Because of the pipelined execution a complete matrix-vector multiplication can be performed at each time step. The application requires only one video scan and can be performed with only 16 parameters for the scan pattern. Additionally for every scan window position an offset (+6 configuration words) has to be configured to the SWG.

6. Conclusions

A novel universal sequencer hardware has been introduced - along with a methodology of its application for instruction level parallelism. Other speed-up mechanisms are the support for burst memories and a pipelined hardware structure. Its usefulness has been illustrated by means of application specific processor array example applications. Several sources of acceleration provided by this novel sequencer have been described, such as e. g. the drastic reduction of processor / memory communication bandwidth requirements. The flexible structure allows to adopt the hardware to specific applications. Even a high level of parallelism in memory accesses can be achieved, through generating addresses for several memory banks in parallel. An easy description of the parallel access sequences is achieved by using the 2-dimensional memory concept. The generic programming scheme allows complex memory accesses with only a few parameters and enables very fast reconfiguration times. Furthermore, the parameter set for an application can easily be adopted to different data sizes (e.g. in image processing, since the parameter set for an application is compiled it can be scaled to any image size).

The presented data sequencer version is being implemented on an Altera FLEX10k FPGA [Alt95], as part of Kaiserslautern's next generation Xputer prototype MoM-4, including a data distribution network for parallel memory access by the reconfigurable Kress ALU array. The advantage of the FPGA implementation will be the flexible structure of the SWG, which can be adjusted depending on the application to number of required memory banks.

7. References

[HB97] R. Hartenstein, J. Becker: A Two-level Co-Design Framework for data-driven Xputer-based Accelerators; published in Proc. of 30th Annual Hawaii Int'l Conf. on System Sciences (HICSS-30), Jan. 7-10, Wailea, Maui, Hawaii, 1997.

[Rit96] U. Rith: Neue Speicherarchitekturen für Workstations und PCs; APS Nachrichten, Mitteilungen der ITG/GI Fachgruppe Arbeitsplatzrechensysteme, ISSN 0941-9519, October, 1996.

[Sie96] N.N.: Siemens Multibank DRAM, Ultra-high performance for graphic applications; Siemens Semicond. Group, Oct., 1996.

[HBH96] 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.

[HR95] R. Hartenstein, H. Reinig: Novel Sequencer Hardware for High-Speed Signal Processing; Worksh. on Design Methodologies for Microelectronics, Smolenice Castle, Slovakia, Sept. 1995.

[Alt95] N.N.:FLEX10k Embedded Programmable Logic Family, Data Sheet; Altera Corp., July 1995.

[HBK95] R. W. Hartenstein, J. Becker, R. Kress, H. Reinig, Karin Schmidt: A Reconfigurable Machine for Applications in Image and Video Compression; Conference on Compression Technologies and Standards for Image and Video Compression, Amsterdam, The Netherlands, March 1995.

[Har94] R. W. Hartenstein, et al.: A Dynamically Reconfigurable Wavefront Array Architecture for Evaluation of Expressions; Proceedings of the Int. Conference on Application-Specific Array Processors, ASAP'94, San Francisco, IEEE Computer Society Press, Los Alamitos, CA, Aug. 1994.

[HHS91] R. Hartenstein, A. Hirschbiel, K. Schmidt, M. Weber: A Novel Paradigm of Parallel Computation and its Use to Implement Simple High-Performance-HW; Future Generation Computer Systems 7 91/92, p. 181-198, (Elsevier Scientific).

[Ble90] Guy E. Blelloch: Vector Models for Data-Parallel Computing. MIT Press, Cambridge, MA, 1990.

[Har87] D. T. Harper III and J. R. Jump: Vector access performance in parallel memories using a skewed access scheme; IEEE Transactions on Computers, Vol. C-36, No. 12, pp. 1440-1449, Dec. 1987.