Reiner W. Hartenstein, Juergen Becker, Michael Herz, Ulrich Nageldinger: A Novel Universal Sequencer Hardware; Proceedings of Fachtagung Architekturen von Rechensystemen ARCS'97, Rostock, Germany, September 8-11, 1997

A Novel Universal Sequencer Hardware

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 hardware for controlling computational machines and for structured DMA (direct memory access) applications. The paper introduces the principles and the design of a novel class of this sequencer hardware which supports two-dimensional memory address space or at least the two-dimensional visualization of the traditional one-dimensional address space. From these concepts it derives a classification scheme of computational sequencing patterns and storage schemes.

1. Introduction

In many applications areas, like image processing, digital signal processing, multi media, and many others, a lot of software to hardware migration ideas have been implemented for acceleration by add-on data path hardware: hardwired (as. e. g. in Embedded Systems, PCs and workstations), as well as reconfigurable (e. g. in custom computing machines, CCM). Only very few such innovations are known from the control side. Still the unstructured one-dimensional memory address space is practically the only basis of execution organization. We also are not aware of any common DMA concepts other than based on one-dimensional address space.

With the recent improvements in the area of CCMs using new highly pipelined architectures such as e.g. [KNS96] based on the Xilinx XC6k [Xil96] family, the memory access latency becomes a determining factor. Therefore it is investigated how data can be arranged and sequenced faster to the computational part and how memory cycles can be saved. The required data sequencing is mostly performed either by the CCM itself or by a host computer (for a classification of CCMs see [HB96a]). But both alternatives have serious disadvantages. If a host performs the data sequencing there is no parallel processing possible and additional synchronization mechanisms as well as address computations decrease performance. By integrating the sequencer hardware directly into the configurable part of a CCM, valuable design space is wasted and the design process is unnecessarily complicated. Some systems (e.g. RACE [SB96]) try to compensate the memory bottleneck with several memory banks operating in parallel, connected to a computing network (four Xilinx XC4013 in the case of RACE). This makes the distribution of the data very difficult, because each data has to be placed in a memory bank near to its processing component. If data is needed several times or can‘t be placed in the correct bank, it has to be passed over long distances wasting routing resources. In other cases, by redundant data replication in different banks, consistency problems have to be solved. This

reduces performance even more. Thus, without a sophisticated direct memory access (DMA) an enormous overhead of memory cycles arises through instruction fetch operations and storing of intermediate results. Examples have been published where up to 90% of computation time of a von Neumann system elapses for calculation of data addresses [HHS91]. The major reason is the enormous number of memory accesses needed. This paper introduces a new kind of sequencing method which dramatically reduces the number of memory cycles needed for a large class of algorithms.

For applications from image processing, signal processing, matrix operations and many others, a two-dimensional memory address space or at least the two-dimensional visualization of the traditional one-dimensional address space is very useful. In supercomputing the organization of storage schemes with interleaved memory capability is more comprehensible. Therefore a 2-dimensional memory concept has been implemented in the proposed approach to provide parallel memory access without consistency problems. An universal data sequencing hardware has been developed, which supports two-dimensional memory address space or at least the two-dimensional visualization of the traditional one-dimensional address space. This sequencer hardware is an alternative for controlling computational machines and for structured DMA (direct memory access). It has the advantage that memory address sequences are generated generically by hardware requiring only a few parameters with very short configuration times.

The paper introduces the principles and the design of a novel class of this sequencer hardware and derives from it a classification scheme of computational sequencing patterns and storage schemes. This novel kind of sequencer is the essential architectural component of a new machine paradigm published elsewhere [HHS91]. Further the principal RT level structure of such sequencers and their operation and programming scheme by means of application examples are presented. Finally the paper shows, that this sequencing method adds backbone of a common model for H/S co-design and custom computing machines (CCM) [HB97].

2. 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 [HB96b].

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 [HB96b]). 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 1 shows all necessary components and their interconnect.

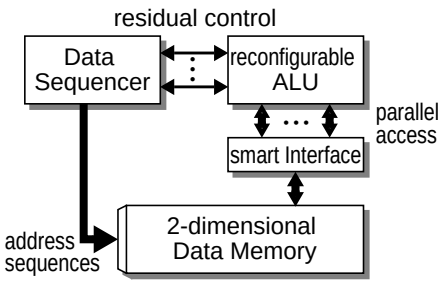


Figure 1. Basic structure of Xputers.

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 Kress ALU Array (KrAA) [Ha95a] [Ha95b]. 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. 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 [HB97].

To clarify how operations are executed the execution model for Xputers 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 a 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. The scan window is a model for the smart interface which holds a copy of the data 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 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 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 and are generated generically. As a result no memory cycles are needed for address generation.

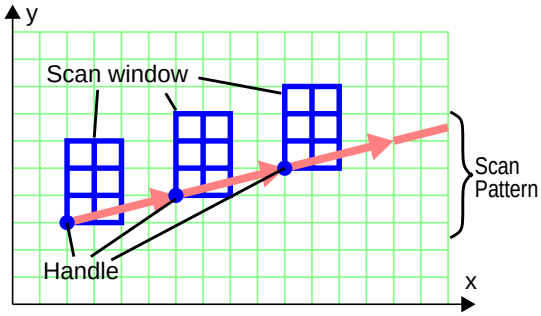


Figure 2. Execution model for Xputers

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. Since the data sequencer is the key component of an Xputer, an implementation is introduced. To justify the proposed implementation a classification of scan pattern is given first.

3. Classification of Scan Patterns

To build a high performance sequencer hardware, first the 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 in view to reconfigurable logic because a small parameter set requires less configuration time. In the following a short discussion of possibilities to describe scan patterns is given.

There may be several ways to design scan patterns. For example the class of all single steps on a 2-dimensional memory can be defined and then concatenations and variations can be built, further nesting and repetitions are possible. All scan patterns can be obtained by this method, but it isn‘t possible to get a short, efficient description of them.

To build a fast (re-)configurable hardware to generate scan patterns, a model to describe scan patterns generically is required. Therefore the linear scan, like shown in figure 2, could be defined as the atomic element. Obviously a linear scan can also be of length 1, consequently all single steps can be modeled. With this scheme a larger class of scan patterns is obtained, which can be described with 3 parameters:

- Starting point of the first step of the linear scan (x- and y-coordinate)
- Ending point of the first step of the linear scan (x- and y-coordinate)
- Number of steps.

Since 2-dimensional memory is available always more than a linear scan is necessary to make use of it. Therefore the video scan (see figure 3a) is introduced as the atomic element of the classification. 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. Of course the number of steps in the second direction can be zero and linear scans are produced. By setting also the number of steps in the first direction to one, a single step is performed. The so-called slider model is used to introduce the parameters required for the generic generation of the class of video scan patterns.

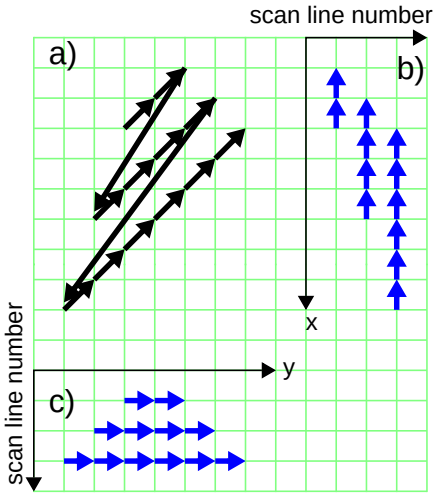


Figure 3. Video scan example (a) with x- and y-components (b&c)

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 (figure 3). First the slider model for one dimension is introduced and later the combination for two dimensions is explained.

The slider model generates gradually linear scans in a 1-dimensional address space. Therefore five sliders are introduced: Base (B), Limit (L), Floor (F), Ceiling (C) and Address (A). Each time a linear scan is generated (i.e. the Address stepper 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 Address stepper loop 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

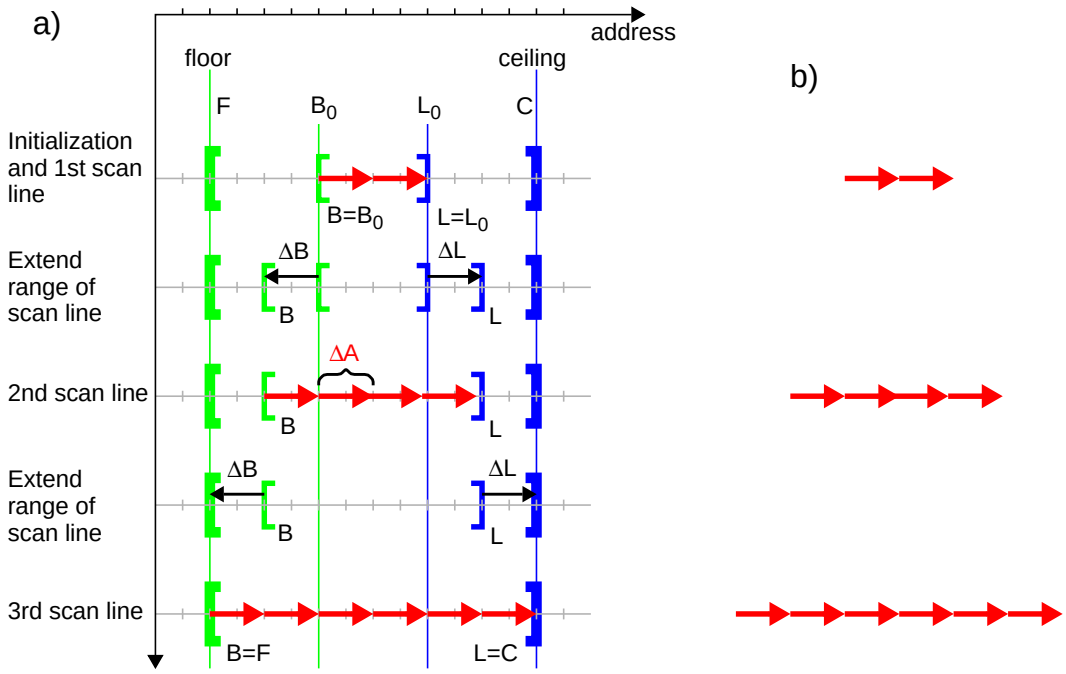


Figure 4. Illustration of the slider model (a) and generated scan sequence (b) for x-component of the scan pattern in figure 3 (figure 3c)

Base meets or passes Floor or if Limit meets or passes Ceiling the address generation is finished. This method is illustrated in figure 4a.

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

- *Synchronous*: at each time step every Address slider performs one operation step. (Table 1(Ia) or figure 3a)
- *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))

The generated video scan ends when the slider model for one dimension indicates the end of address generation, i.e. Base or Limit exceeds Floor or Ceiling.

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 Combinations of Video Scans

By using the video scan as atomic element and building combinations, more scan patterns are classified. This results in four additional classes, 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. 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 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 only a *relative* flag added to the scan characteristics parameter.

Meshed Scans. The class of meshed scans contains all scans where at least two video scans are concatenated similar to compound scans but at least one scan is not finished within one iteration and all concatenated video scans are iterated until one designated video scan is finished. Each video scan except the first step of the first scan starts his first step at the last position of the predecessor 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 with the Zig-zag enumeration example for the run length coding step in JPEG image compression method [HB95]. Therefore a compound scan consisting of two meshed scans is necessary. Each meshed scan contains four video scans (see figure 6).

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 four video scans are combined to a meshed scan, the two meshed scans are

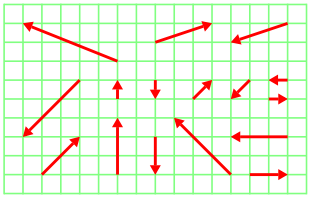
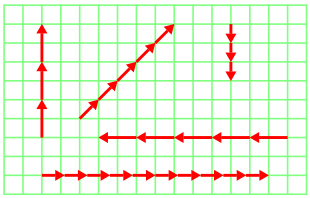
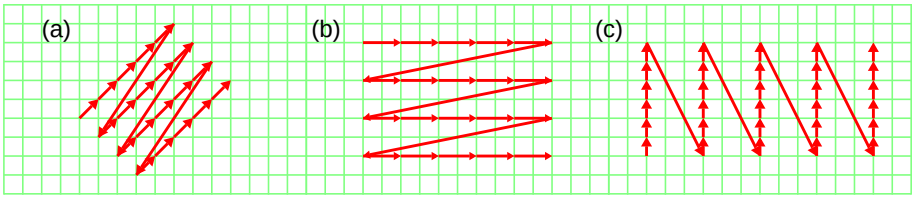
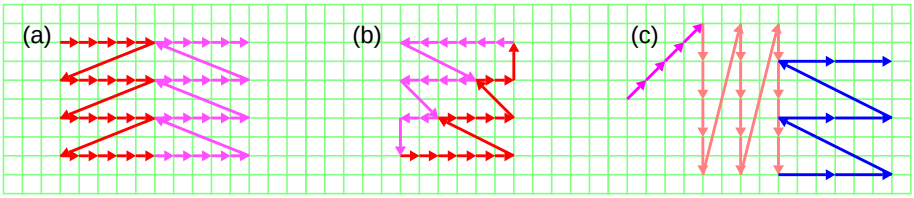
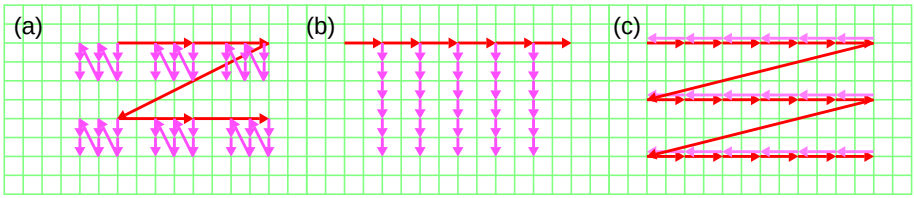
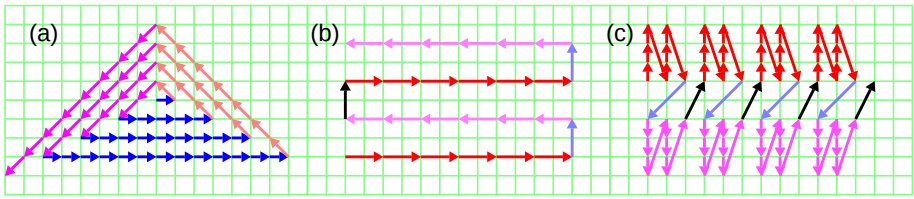
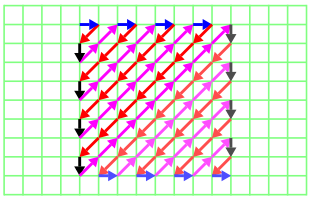
Classes	Examples
single steps	
linear scans	
(I) video scans	
(II) compound scans	
(III) nested scans	
(IV) meshed scans	
(V) complex scans	

Table 1. Scan pattern classes

compound (=8 video scans) and nested in an outer scan (plus one video scan), 9 video scans with 16 parameters for each are used. Therefore the run length coding step of JPEG image compression method requires 144 parameters. The number of parameters does not depend on the size of the image.

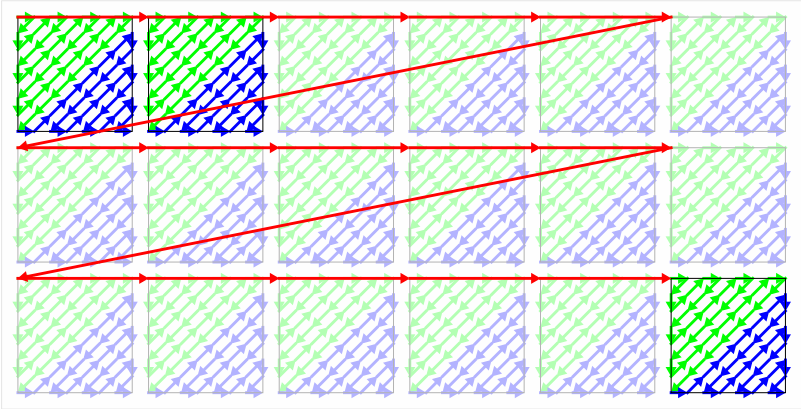


Figure 5. Run length coding scan for a 48x24 pixel image in JPEG image compression method.

3.3 Summary of the Classification

Out of the discussion above the video scan as the basic scan pattern is obtained. A video scan can also be a single step or a linear scan. Further 3 important classes of combinations of the video scan are figured out. These are the compound scan, the nested scan and the meshed scan. It can be proofed 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 efficient. The fifth class contains all combinations of compound, nested and meshed scans.

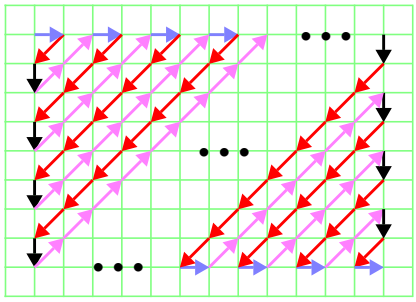


Figure 6. Zig-zag enumeration

4. Xputer Data Sequencer Structure

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 is shown.

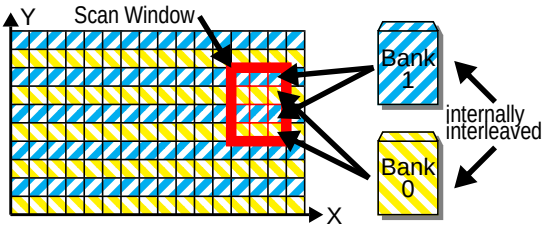


Figure 7. Memory distribution

4.1 Memory Organization

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 7). With this organization accessing different rows in parallel is made possible. This is done at the second level of the 2-level data sequencing. Depending of 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.

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 memory fragmentation. Because the memory is organized 2-dimensionally, it has to be mapped by the data sequencer to commercially available 1-dimensional memories.

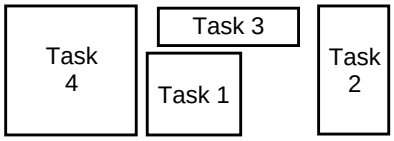


Figure 8. Several tasks mapped onto the data memory

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. 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 9.

4.2.1 Handle Positioning

The Handle Position Generator (HPG) performs the first level of the 2-level data sequencing. It provides two identical address generators for the x- and y-address generation according to the slider model. Further there is a context switcher (Figure 9) unit 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 like pictured in figure 8.

For each slider (base-, limit- and address slider) of the slider model the address generator hardware provides a corresponding stepper called base-, limit- and address- stepper (Figure 10). All steppers are built with the same hardware (Figure 11), implementing a slider which gives the actual address. This slider is positioned at an initial position and steps with a defined step with until an end value is reached.

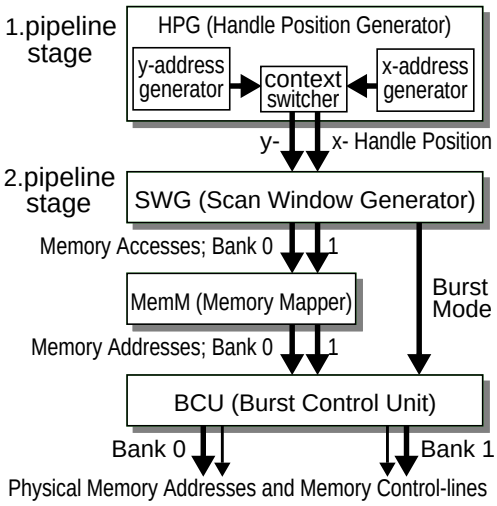


Figure 9. The Data Sequencer address generation datapath

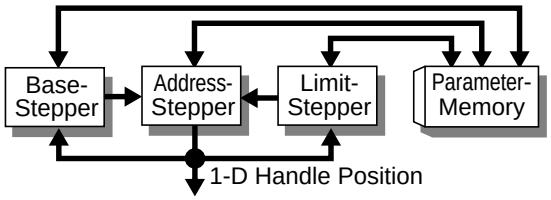


Figure 10. 1-D address generator

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 address generator (Figure 10).

Figure 11 shows the stepper hardware. For illustration purposes the registers are named for the base slider/stepper implementation. The limit value (called Floor in the case of the Base stepper) is relative to the initial value (B_0). Further, the initial value can be absolute or relative to *A'*. In this case *A'* is loaded by the last valid address of the previous or outer scan pattern (e.g. for nested scans). The first position is generated by initializing the output register B by B_0 (or by B_0+*A'* for relative scans). Address generation is performed by adding ΔB to B each iteration until the limit value (B_0+F) is reached. This is detected by the End Detection Unit.

The Limit- and Base-Stepper and the Address-Stepper are working alternately. First the Limit- and the Base-Stepper generate an initial (Base-Stepper) and a limit value (Limit-Stepper). Then the Address-Stepper performs a whole scan line starting at the initial value (base), incrementing by ΔA and running until the limit value is reached. This is iterated until either the Base- or the Limit-Stepper has finished.

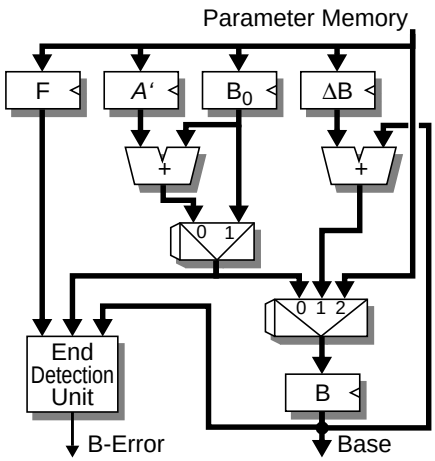


Figure 11. The stepper hardware

4.2.2 The Scan Window Generator

The position of the scan window is determined by the handle position generated by the HPG. The scan window generator (SWG, figure 12) has to access all memory locations inside the scan window and performs the mapping of the rows of the 2-dimensional memory to the different memory banks (figure 7).

The generation of the scan window is realized by adding address 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, they are initiated. All parameters for second level of the 2-level data sequencing are provided by the offset generator (figure 12). They are stored in a look-up table and sequenced in succession. The look-up table provides capacity for 16 different scan windows. The offset generator optimizes scan window operations of overlapping windows by reading only the 'new', non-overlapping content. The sum of the handle and the offset is built by an adder unit which consists of two separate adders for each dimension.

The address mapping to the parallel memory banks is performed by evaluating the LSB (least significant bits) of the y-address. As mentioned earlier the hardware design is explained only for two parallel memories. For each parallel RAM bank the SWG provides a separate offset generator and an adder unit. The SWG has to determine which adder unit is assigned to which RAM bank, depending on bit 0 of the y-address. This optimizes memory access in the second dimension (y) by having several banks accessible in parallel, while memory access in the first dimension (x) is optimized by burst operations.

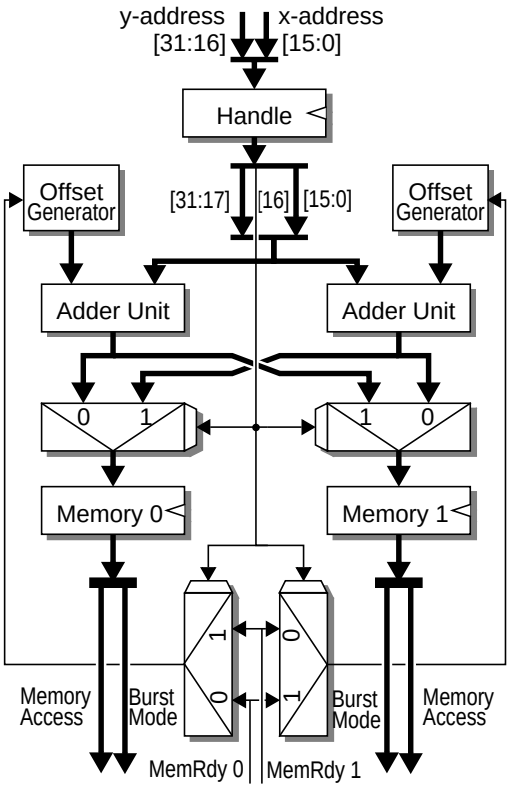


Figure 12. Scan Window Generator

4.2.3 The Memory Mapper

To reduce fragmentation of 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 (Figure 8).

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 13b. The higher bits of the address register are shifted to the lower positions until all unused lower zero bits are eliminated (Figure 13c).

If there are several tasks in the physical memory the context switcher in the HPG secures that there is no memory violation. The shift operations are performed with little hardware by a modified barrel shifter. The memory mapper hardware (Figure 13a) is required for each memory bank.

4.2.4 Burst Operations

Since memory with burst options is supported an additional unit has to control this operations. The required signals for variable burst lengths are generated by the Burst Control Unit (BCU) 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.

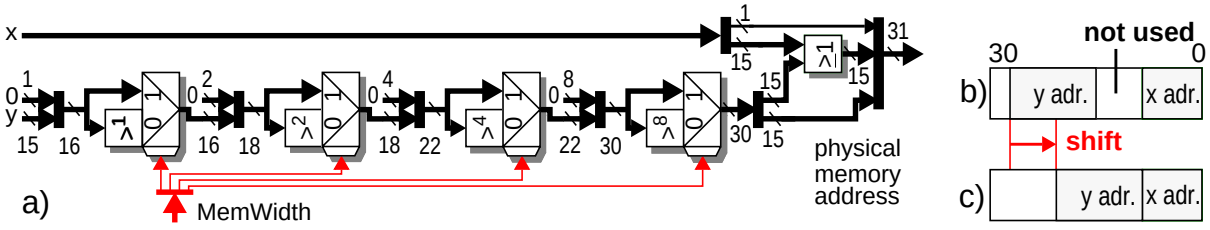


Figure 13. The Memory Mapper (a) and its address mapping (b&c)

5. Conclusions

The hardware of a novel universal sequencer hardware has been introduced. The performing of scan patterns guarantees efficient memory accessing, resulting in very short configuration times. Several sources of acceleration provided by this novel sequencer are introduced. The pipelined structure parallelizes the address generation. Since this data sequencing approach generates memory accesses generically, no instruction cycles for their computation are necessary. Moreover the scan window generator saves memory cycles by optimizing accesses to data needed several times, which is supported by the smart interface introduced in section 2. The 2-dimensional memory enables two speed-up mechanisms. Data accesses in x-direction can be accelerated with a burst mode, while accesses in y-direction are performed in parallel.

The presented data sequencer is being implemented on Altera FLEX10k FPGAs [Alt95], as part of Kaiserslautern's next generation Xputer prototype MoM-*PDA* (Map-oriented Machine with Parallel Data Access), including a data distribution network for parallel memory access to the reconfigurable Kress ALU Array.

6. References

[HB97] 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.

[HB96b] 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.

[HB96a] R. Hartenstein, J. Becker, et al.: Custom Computing Machines vs. Hardware/Software Co-Design: from a globalized point of view; Proc. of 6th International Workshop On Field Programmable Logic And Applications, FPL'96, Darmstadt, Germany, Sep. 23-25, 1996, Lecture Notes in Computer Science, Springer Verlag, 1996

[KNS96] T. Kean, B. New, B. Slous: A Fast Constant Coefficient Multiplier for the XC6200; Proc. of 6th International Workshop On Field Programmable Logic And Applications, FPL'96, Darmstadt, Germany, Sep. 23-25, 1996, Lecture Notes in Computer Science, Springer Verlag, 1996

[SB96] D. Smith, D. Bhatia: RACE: Reconfigurable and Adaptive Computing Environment; Proc. of 6th International Workshop On Field Programmable Logic And Applications, FPL'96, Darmstadt, Germany, Sep. 23-25, 1996, Lecture Notes in Computer Science, Springer Verlag, 1996

[Har96] R. Hartenstein: High-Performance Computing: Über Szenen und Krisen; GI/ITG Workshop on Custom Computing, Schloß Dagstuhl, Germany, June 1996.

[Xil96] N.N.: The Programmable Logic Data Book; Xilinx, 1996.

[Ha95b] R. W. Hartenstein, et al.: A Scalable, Parallel, and Reconfigurable Datapath Architecture; Sixth International Symposium on IC Technology, Systems & Applications, ISIC'95, Singapore, Sep. 6-8, 1995

[Ha95a] R. W. Hartenstein, et al.: A Datapath Synthesis System for the Reconfigurable Datapath Architecture; Asia and South Pacific Design Automation Conference, ASP-DAC'95, Nippon Convention Center, Makuhari, Chiba, Japan, Aug. 29 - Sept. 1, 1995

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

[HB95] R. Hartenstein, J. Becker, et al.: 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

[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).