Reiner W. Hartenstein, Helmut Reinig: Novel Sequencer Hardware for High-Speed Signal Processing; Workshop on Design Methodologies for Microelectronics, Smolenice Castle, Slovakia, September 1995

NOVEL SEQUENCER HARDWARE FOR HIGH-SPEED SIGNAL PROCESSING

Reiner W. Hartenstein, Helmut Reinig

Dept. of Computer Science, University of Kaiserslautern
P.O.Box 3049, 67653 Kaiserslautern, Germany
e-mail: abakus@informatik.uni-kl.de, FAX: ++49 631 205 2640

Abstract. *The paper presents a novel kind of sequencer hardware, called data sequencer. For many signal processing, multimedia, and other high-performance applications the new sequencer drastically reduces processor-to-memory bandwidth requirements, compared to conventional instruction sequencers. The data sequencer coordinates the parallel operation of multiple address generators, each providing independent access to structured data completely under hardware control. The paper describes principles and the hardware design of the data sequencer and draws comparisons to related speed-up approaches.*

Keywords: *data sequencing, high-performance, address generator, structured data access*

1 INTRODUCTION

Many applications in signal and image processing require fast access to structured data. Most of the time, the data structures are arrays, either of simple scalar values or arrays of records. In many algorithms (e.g. FFT, or FIR filtering), the access sequence to the data is already known at compile time, because the index computations to the array elements are done by nested loops, which do not depend on the actual data values. This is the reason, why most digital signal processors [3, 4, 6] contain independent address generating ALUs. They are used to compute the addresses of data for future use in parallel to the data manipulations on already fetched data. Typically, these address generators are capable to produce a linear address sequence with constant offsets between subsequent addresses, cyclic or modulo addressing for cyclic buffers, and bit-reversed addressing for FFT-type applications. Harris Semiconductor offers a similar address sequencer as a stand-alone device in a single package [5]. In addition to linear address sequences, it is capable to access data in two-dimensional arrays from a single set of parameters. All of the address generators described above are capable to provide addresses for scalar values only. The frequent cases, where an algorithm requires access to structured data (e.g. a subarray moving as a sliding window across the whole data array in two-dimensional FIR filtering) require a new setup of the address sequence parameters quite often. This is because these address generators cannot make use of the hierarchy information, where a simple data access sequence (the accesses to the subarray or record elements) is repeated in a regular manner to access a large amount of structured data. This is the motivation to develop an address generating device to support structured data access for multi-dimensional arrays. The key issues are: support for structured data access, data organization in one-, two-, or three-dimensional arrays, coordinated operation of multiple address generators, and versatility of the possible address patterns to reduce the reconfiguration overhead. We call this device Generic Address Generator, because it is used to produce generic address sequences (known at compile-time) ahead of the data manipulations without consuming instruction or memory bandwidth apart from a few parameter

transfers. Multiple Generic Address Generators combine to a Data Sequencer, which controls data manipulations by the sequence in which data arrives at the data processing hardware.

The following section introduces the Generic Address Generator hardware. An application example demonstrates its use in a high-speed image analysis environment for automotive applications. A final section provides some technical information on the device and concludes the paper.

2 THE GENERIC ADDRESS GENERATOR

The Generic Address Generator is a single device, which computes a sequence of addresses from a set of parameters. The parameters can be downloaded via a simple processor bus interface. During operation, a Generic Address Generator produces memory addresses only - the data manipulations have to be performed by other devices, which operate on the data according to the order in which the data elements arrive. Since the Generic Address Generators do not produce opcodes or addresses for the data manipulating devices, these can be chosen from a wide variety of alternatives. They may either be general purpose microprocessors interfaced to the Generic Address Generator(s) by queues, or a reconfigurable computational hardware, or application specific computational devices designed for high throughput applications.

The Generic Address Generator provides a logical view to the memory as a two-dimensional map. Higher dimensional data arrays can be mapped by slicing them into planes, which can be placed adjacently onto the two-dimensional memory organization. Large linear arrays (vectors) are constructed by concatenating multiple rows in the address computations. Physically, a linear memory is accessed, of course, for compatibility with conventional memory organizations. The Generic Address Generator can be programmed to several variations how to combine two address parts to a single linear memory address. To support efficient access to structured data, a Generic Address Generator operates in a two-stage pipeline. The first stage computes handle positions for so-called scan windows, which represent a neighbourhood of data elements around the handle position (Handle Position Generator). A handle position consists of two 16-bit values for the two dimensions of the data memory. The sequence of handle positions describes how the corresponding scan window moves across the data memory (figure 1). Such a sequence of handle positions is called scan pattern.

The second pipeline stage computes a sequence of offsets to the handle positions, to obtain the effective memory addresses for the computations. Therefore this stage is called Memory

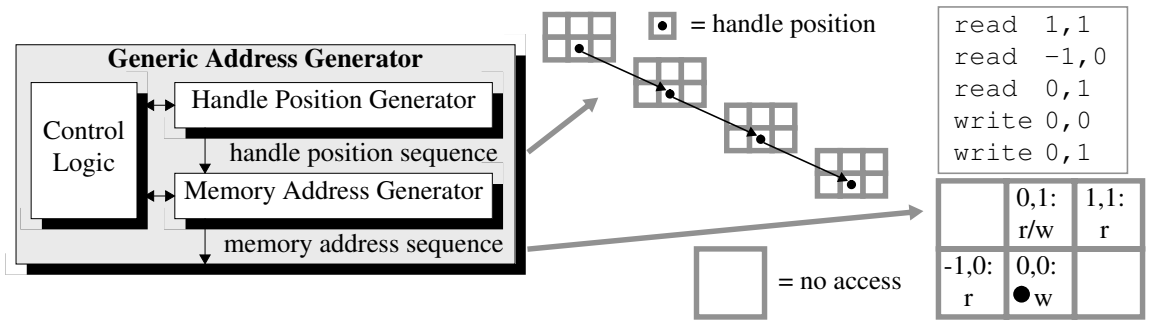

Figure 1: Block diagram of the Generic Address Generator

Address Generator. The offsets may be programmed to appear in arbitrary order, because the memory addresses are computed from simple RISC-like instructions (figure 1). With a proper instruction pipelining, addresses can be delivered one in each clock cycle, so that the introduction of instruction sequencing at this level does not introduce a bottleneck.

Multiple Generic Address Generators may be run in parallel on a single memory bus or multiple memory buses. They synchronize their scan patterns through the activations of the data processing device(s). All scan patterns proceed until either they detect an explicit synchronization point in the offset sequence of the Memory Address Generator, or they are blocked by a write operation to memory, waiting for the data processing device(s) to compute the desired result. In case of a single memory bus, explicit synchronization instructions in the Memory Address Generator program are mandatory to allow bus mastership to change from one Generic Address Generator to the next.

The following sections describe the pipeline stages of the Generic Address Generator in more detail. Afterwards, the capabilities of a Generic Address Generator are illustrated by examining the set of scan patterns that a Generic Address Generator can produce without requiring a reconfiguration.

2.1 The Handle Position Generator

The control structures of an algorithm are mapped onto the scan patterns produced by the Handle Position Generator. Therefore, a Handle Position Generator should be as versatile as possible, so that many kinds of nested loops can be transformed to single parameter sets for long periods of hardware controlled address generation. A Handle Position Generator can produce a scan pattern corresponding to four nested loops at the most. It is programmed by specifying a set of parameters, such like starting position, increment value, and end position of a loop, each both for the x and y dimension of the data memory. A closer look to the hardware organization makes clear, how this can be achieved.

The programming model of the memory is a two-dimensional map. Therefore the Handle Position Generator consists of two identical parts, one for each dimension (figure 2). This allows to use both dimensions at will, since the hardware structure induces no preference to assign specific characteristics only to one dimension. The so-called 1–D Address Generators operate in parallel and synchronize through a trigger logic, which preserves the symmetry of the design. The programmer decides, which (if any) 1-D Address Generator operates as master, triggering the other 1-D Address Generator to perform a step at the appropriate positions. The trigger logic simply routes the trigger signals according to the programmer's specification. Furthermore, it evaluates all conditions, which require a knowledge of the state of the complete system (e.g. when a scan pattern terminates).

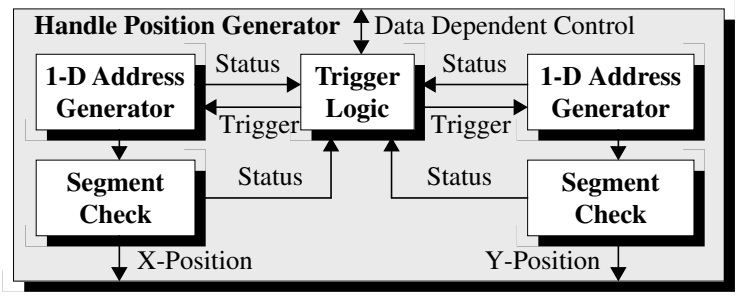

Figure 2: Block diagram of the Handle Position Generator

The one-dimensional positions produced by each of the 1-D Address Generators are checked against segment limits, before they are presented to the Memory Address Generator as valid handle positions. This serves as a kind of memory protection scheme, which is especially useful, if the handle positions are computed under control of the data manipulating devices, dependent on the data processing results. The segment check units make sure, that all handle positions of a Generic Address Generator remain within a programmer-defined orthogonal bounding box in the two-dimensional memory map. By providing the maximum and minimum offsets that occur in the Memory Address Generator program, the segment checks can even evaluate whether the bounding-box would be left during the generation of memory addresses and invalidate such a handle position before it is passed on in the pipeline.The actual address computations are done by the 1-D Address Generators. Each 1-D Address Generator is capable of handling two nested loops, where the number of iterations is known at compile time.

2.2 The Memory Address Generator

The Memory Address Generator resembles a kind of RISC processor with a special purpose instruction set and on-chip instruction memory. The only task that has to be performed by the Memory Address Generator, is to provide an arbitrary sequence of offsets to be added to the handle position to perform memory accesses in a local neighbourhood around the scan window handle position. The most efficient way to support arbitrary offset sequences, is to make the offsets directly programmable in a memory that is scanned linearly. At every new handle position, the memory is scanned again from the beginning. But since the length of the offset sequence varies from application to application, an escape mechanism has to be programmed into the offset memory, to signal the end of an offset sequence. Interpreting the offsets as read or write instructions and the escape code as branch to the beginning, the Memory Address Generator can be programmed with a small and simple instruction set. Additional instruction codes have to be introduced to support pipelining in the data processing devices with the requirement of conditionally executed read or write operations at the beginning and at the end of loops, to fill the pipeline, and to flush results remaining in the pipeline.

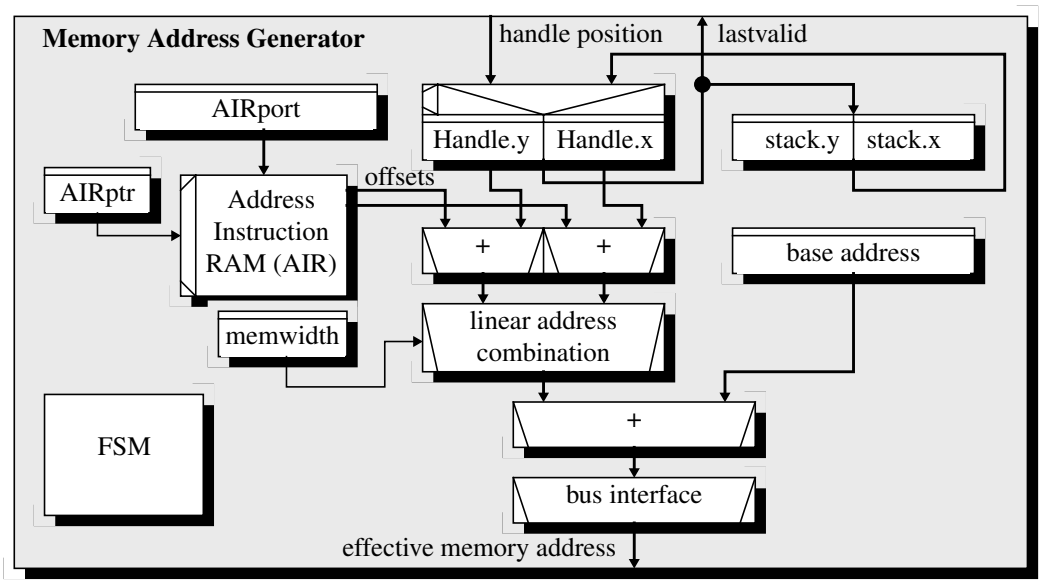

Figure 3: Block diagram of the Memory Address Generator

The range of offsets may be –32 to +31 in both dimensions of the data memory. The memory to store the instructions (address instruction RAM, AIR) has 256 entries, so that at most about 250 references to the data memory can be made from a single handle position. At least three instructions are overhead, which have to be inserted to accept a new handle position from the Handle Position Generator, to signal the start of computations to the data processing devices, and to jump to the beginning at the end of an offset sequence.

After the instructions have been fetched and decoded, the offsets have to be added to the current handle position. A number of steps are performed with each of the resulting addresses, to transform them into physical memory addresses. These tasks are completely hardwired, because they have to be applied to each address. The first is an address modification to allow for cyclic addressing. For each dimension, a CycleMask word is used to keep selected bits of the address from changing, and a CyclePattern word provides default values for the masked bits. If the CycleMask is partitioned into a leading block of 16–k zeroes and a trailing block of k ones, for example, the resulting addresses automatically wrap around at the 2^k boundary, because the values of the higher (16 – k) bits are masked. Input to the following step are still two 16-bit address words, one for each dimension. To be able to access a conventional linear memory, the address parts have to be combined to a linear memory address. The address parts of the two dimensions may be combined to a real linear memory address in four ways: one row of two-dimensional memory (x dimension) may consume an address space of 10, 12, 14, or 16 bits. This allows to adjust the “size” of the data memory to the size of the processed data, to reduce wastage of address space. After the concatenation of the two address parts to a linear address, a 32-bit base address is added, to obtain the effective memory address. The base address typically is the starting address of the data array referenced by this Generic Address Generator. Finally, the bus interface handles the protocol for the actual data transfers between the data memory and the data manipulating devices, using the effective memory address to access the data memory. A block diagram (figure 3) illustrates these tasks, as they are performed in the Memory Address Generator. The AIRport register has been introduced for two reasons. First, the instructions in the Address Instruction RAM (AIR) are only 16 bits wide, so that the AIRport register serves as an interface register, to transfer two instructions with one configuration data transfer. Second, to the processor which downloads the parameters, the Address Instruction RAM is hidden behind the AIRport register and consumes only a single address. The programming model corresponds to a shift register, allowing to write a complete instruction stream to the same AIRport address during configuration. This can be done efficiently using block transfers. This concept improves the scalability of the Generic Address Generator, because different sizes of the Address Instruction RAM do not require a different layout of configuration register addresses, but only a different length of the block transfers. This is true for the number of instructions that can be stored as well as for the format of the instructions (32-bit instructions would allow for a larger offset range, for example). The resulting changes to the download software can easily be parameterized, whereas the processor interface remains unchanged for all these variations of Memory Address Generators.

A chip photograph of the Generic Address Generator can be seen in figure 4. It has been fabricated in a 1.0 m CMOS standard cell process at European Silicon Structures (ES2). The circuit contains 6821 standard cells, being equivalent to 126702 transistors, excluding the RAM. The macro-block in the upper left corner is the 256 word Address Instruction RAM. The die size is 8.3 by 7.4 mm^2, including the pads. It is housed in a 101 pin PGA package and operates at 20 MHz. Spending additional efforts in the place and route process and inserting some more stages to the pipeline, higher operation frequencies will be possible.

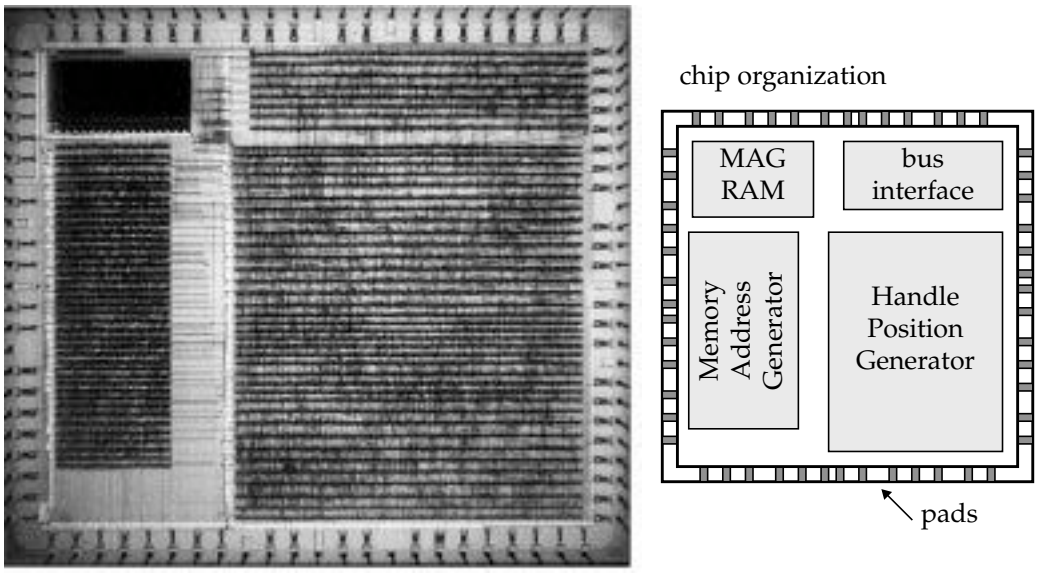

Figure 4: Generic Address Generator chip photograph

2.3 Scan Patterns

The Generic Address Generator is capable to compute a wide variety of scan patterns from a single parameter set, without requiring further interaction with a host processor. To give an impression of the Generic Address Generator's capabilities, an incomplete choice of examples has been collected in figure 5. It is important to notice, that these scan patterns describe the sequence of handle positions only. At each position, a large number of nearby data elements can be accessed in arbitrary order.

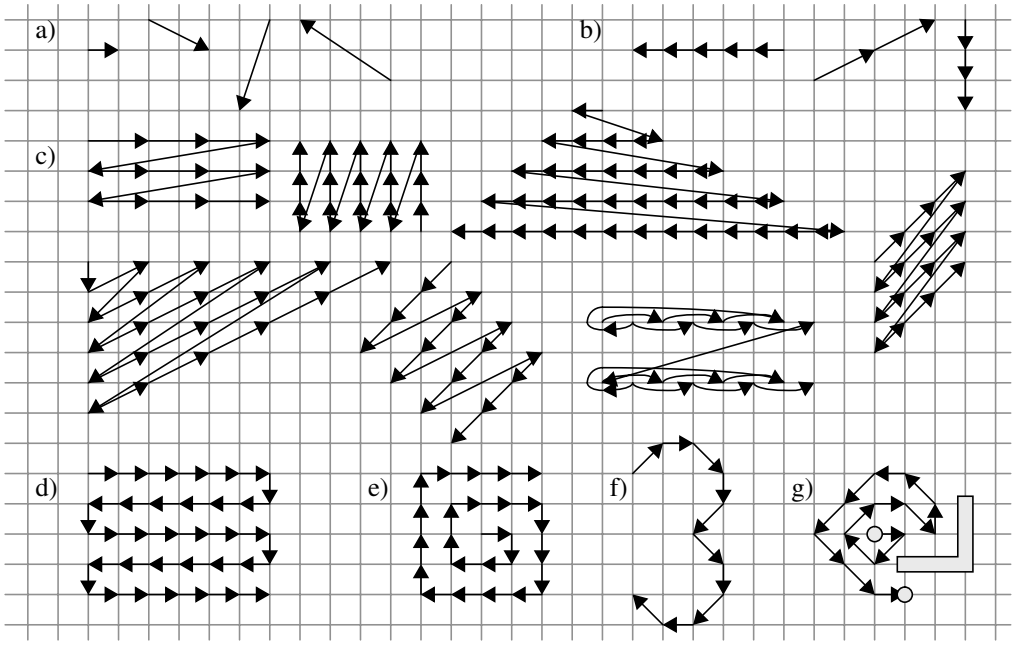

Figure 5: Scan pattern repertory: a) single steps; b) linear scans; c) video-scan variations; d) zig-zag scan; e) spiral; f) and g) data dependent scans

For all scan patterns, the stepwidths can be chosen arbitrarily in the range of 1 to 65536. They have been reduced to one for the more complex patterns only for compactness of the presentation. And since both dimensions are completely symmetric, all variations of the scan patterns, where x and y dimension are interchanged, are valid scan patterns as well. Even the interleaved video-scan in figure 5c, where the odd x positions are visited first and the even ones next before proceeding with the next line can be changed e.g. to a pattern, where first the odd and then the even lines are processed, like an interlaced video display does. The data dependent scan patterns in figure 5f and g can be handled with little overhead, because the data manipulating devices only have to signal the sign bits of the single step vectors, using the programmed stepwidths of the Generic Address Generator. This simplifies curve following and Lee routing applications as examples of algorithms requiring nearest neighbour data dependent processing. Furthermore, the arbitrary sequence of addresses programmable in the Memory Address Generator allows to produce any scan pattern made up of up to 250 references to addresses in a 64 by 64 words wide area. This includes all scan patterns required for the JPEG compression algorithm, which operates on blocks of 64 data words in an eight by eight array organization.

3 APPLICATION EXAMPLE

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 [1]. 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 6.

The first Generic Address Generator computes the address sequence for the image preprocessing hardware, which performs an edge detection (e.g. by applying a two-dimensional FIR filter with appropriate weights). The pixels are read directly from the sensor array in the order which is important to the selected algorithm. The second Generic Address Generator reads the resulting edge enhanced image from the preprocessing hardware and stores it into a dual-ported data memory. There a general-purpose CPU or a digital signal processor may perform the image analysis tasks, where the edges have to be combined to objects. These are identified to detect obstacles, like the approaching car, and 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 style processor than by an address generator.

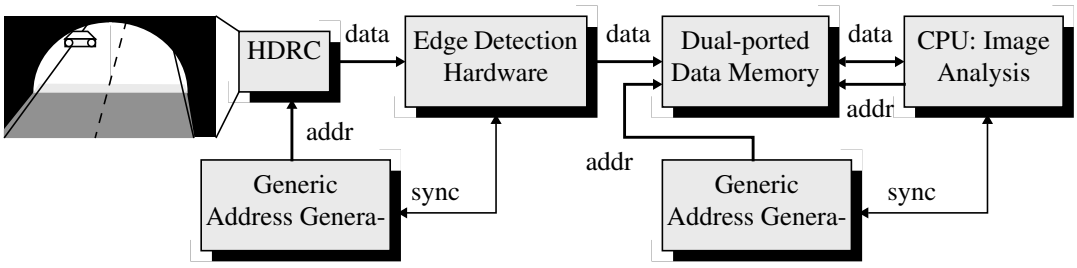

Figure 6: Automotive collision detection using two Generic Address Generators

The hardware for the image preprocessing might be a reconfigurable hardware like the rDPA [2], or the reconfigurable data-driven multiprocessor architecture proposed in [7]. Both would be superior over special purpose hardware with regard to flexibility in the choice of algorithms to be run. The required data rates of 3.2 MBytes per second to process the contents of the complete sensor array of 128 by 256 pixels 100 times a second can easily be handled by both kinds of devices. This allows to process even larger sensor arrays built from multiple devices, or higher frame processing rates than 100 per second.

Using two independent Generic Address Generator 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 in combination with the two-level hierarchy of address generation 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.

4 CONCLUSIONS

The Generic Address Generator developed at Kaiserslautern University is superior to the address generation units found in digital signal processors as well as other separately available address generation devices on the market. 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 at power-up time. Multiple Generic Address Generators can operate as a Data Sequencer in a system, either on shared memory buses or on separate buses, and automatically synchronizing for the highest collision-free data transfer rates. The simple interface to reconfigurable computing devices makes the Generic Address Generator an ideal candidate to build application specific signal processing hardware from off-the-shelf components.

The Generic Address Generator has been successfully designed and fabricated in a student project. The 1.0 m CMOS standard cell process allows an operation at 20 MHz. The circuit fabrication and the design software have been made available by the Eurochip project of the CEC.

References

[1] GRAF, H.G. – HÖFFLINGER, B. – SEGER, U. – SIGGELKOW, A.: Elektronisch sehen. Elektronik, Vol. 3/95, Franzis-Verlag 1995, pp. 52 - 57.

[2] HARTENSTEIN, R.W. – KRESS, R. – REINIG, H.: A Reconfigurable Data-Driven ALU for Xputers. Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines (FCCM'94), Napa, CA, April 1994.

[3] N.N.: ADSP-21020 User's Manual. Analog Devices 1991.

[4] N.N.: DSP96002 IEEE Floating-Point Dual-Port Processor User's Manual. Motorola 1989.

[5] N.N.: Harris Semiconductor Data Book. Harris Corporation 1992, pp. 6-3 - 6-15.

[6] N.N.: TMS320C3x User's Guide. Texas Instruments 1990.

[7] YEUNG, A.K.W. – RABAEY, J.M.: A Reconfigurable Data-Driven Multiprocessor Architecture for Rapid Prototyping of High Throughput DSP Algorithms. Proceedings 26th Hawaii Int'l. Conf. on System Sciences, Vol. 1, IEEE Computer Society Press 1993, pp. 169 - 178.