R. W. Hartenstein, et al.: A New FPGA Architecture for Word-oriented Datapaths; 4th Int. Workshop On Field Programmable Logic And Applications, FPL'94, Prague, September 7-10, 1994, Lecture Notes in Computer Science, Springer, 1994

A New FPGA Architecture for Word-Oriented Datapaths

Reiner W. Hartenstein, Rainer Kress, Helmut Reinig

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

Abstract. A new FPGA architecture (reconfigurable datapath architecture, rDPA) for word-oriented datapaths is presented, which has been developed to support a variety of Xputer architectures. In contrast to von Neumann machines an Xputer architecture strongly supports the concept of the "soft ALU" (reconfigurable ALU). Fine grained parallelism is achieved by using simple reconfigurable processing elements which are called datapath units (DPUs). The word-oriented datapath simplifies the mapping of applications onto the architecture. Pipelining is supported by the architecture. It is extendable to almost arbitrarily large arrays and is in-system dynamically reconfigurable. The programming environment allows automatic mapping of the operators from high level descriptions. The corresponding scheduling techniques for I/O operations are explained. The rDPA can be used as a reconfigurable ALU for bus-oriented host based systems as well as for rapid prototyping of high speed datapaths.

1 Introduction

Word-oriented datapaths are convenient for numerical computations with FPGAs. A recent trend in FPGA technology moves toward the support of efficient implementation of datapath circuits. The Xilinx XC4000 series [9] provides fast 2-bit addition at each logic cell by a special carry circuit. AT&T's ORCA [4] supports even 4-bit arithmetic operations. A 16 bit adder requires only four function blocks for example. Word-oriented datapaths are not directly supported by FPGAs currently available since these circuits are designed for both random logic control and datapath applications. Word-oriented datapaths in reconfigurable circuits have the additional advantage of operators being mapped more efficiently.
The reconfigurable datapath architecture (rDPA) provides these word-oriented datapaths. It is suitable for evaluation of any arithmetic and logic expression. Statement blocks in inner loops of high performance applications can be evaluated in parallel. The rDPA array is in-system dynamically reconfigurable, which implies also partial reconfigurability at runtime. It is extendable to almost arbitrarily large arrays. Although the rDPA has been developed to support Xputer architectures it is useful for a wide variety of other applications for implementation of numerics by field-programmable media.

First, this paper gives an overview on the rDPA. Section 3 explains a support chip which allows the efficient use of the rDPA in bus-oriented systems. Section 4 presents the programming environment for the automatic mapping of operands and conditions to the rDPA. The scheduling algorithm is described. Section 5 shows the utilisation of the rDPA within the Xputer hardware environment. Finally some benchmark results are shown and the paper is concluded.

2 Reconfigurable Datapath Architecture

The reconfigurable datapath architecture (rDPA) has been designed for evaluation of any arithmetic and logic expression from a high level description. It consists of a regular array of identical processing elements called datapath units (DPUs). Each DPU has two input and two output registers. The dataflow direction is only from west and/or north to east and/or south. The operation of the DPUs is data-driven. This means that the operation will be evaluated when the required operands are available. The communication between the neighbouring DPUs is synchronized by a handshake. This avoids the problems of clock skew and each DPU can have a different computation time for its operator. A problem occurs with the integration of multiple DPUs into an integrated circuit because of the high I/O requirements of the processing elements. To reduce the number of input and output pins, a serial link is used for data transfer between neighbouring DPUs on different chips as shown in figure 1. The DPUs belonging to the converters are able to perform their operations independent of the conversion. Using a serial link reduces the speed of the communication, but simulation results showed that by using pipelining, the latency is increased whereas the throughput of the pipeline is decreased only slightly. Internally the full datapath width is used. For the user this serial link is completely transparent.
A global I/O bus has been integrated into the rDPA, permitting the DPUs to write from the output registers directly outside the array and to read directly from outside. This means, that input data to expressions mapped into the rDPA do not need to be routed through the DPUs. The communication between an external controller, or host, and the DPUs is synchronized by a handshake like the internal communications.
An extensible set of operators for each DPU is provided by a library. The set includes the operators of the programming language C. Other operators such as the parallel pre-

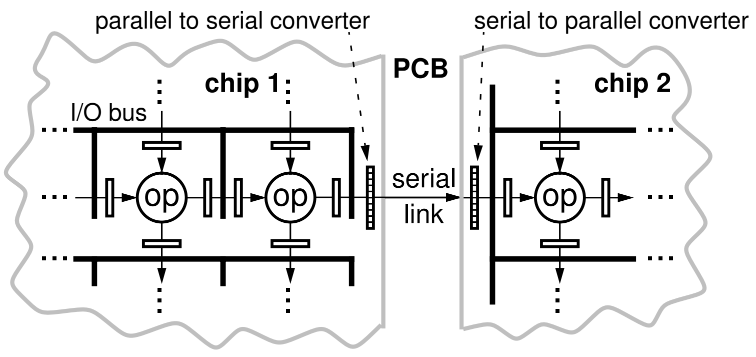


Fig. 1. The extendable rDPA architecture between chip boundaries

fix operator are provided [3]. For example a queue of ‘scan-max’ operators can be used for easy implementation of a hardware bubble sort [7]. The ‘scan-max’ computes the maximum from the input variable and the internal feedback variable and gives the maximum as result and stores the other value internally. In addition to expressions, the rDPA can also evaluate conditions. Each communication channel has an additional condition bit. If this bit is true, the operation will be computed, otherwise not. In each case the condition bit is routed with the data using the same handshake. The ‘false’ path is evaluated very quick, because the condition bit has to be routed only. With this technique also nested `if_then_else` statements can be evaluated (see also figure 4). The `then` and the `else` path can be merged at the end with a merge operator (m). This operator routes the value with the valid condition bit to its output.
The operators of the DPUs are configurable. A DPU is implemented using a fixed ALU and a microprogrammed control, as shown in figure 2. This means, that operators such as addition, subtraction, or logical operators can be evaluated directly, whereas multiplication or division are implemented sequentially. New operators can be added by the use of a microassembler.
As mentioned before the array is extendable by using several chips of the same type. The DPUs have no address before configuration since all rDPA chips are identical. A DPU is addressed by its x- and y-location, like an element in a matrix. The x- and y-location are called addresses later for convenience. A configuration word consists of a configuration bit which distinguishes the configuration data from computational data. Furthermore it consists of the x- and the y-address, the address of the DPU’s configuration memory, and the data for this memory.
Each time a configuration word is transferred to a DPU, the DPU checks the x- and the y-address. Four possible cases can occur:

- the y-address is larger than zero and the x-address is larger than zero
- the y-address is larger than zero and the x-address is zero
- the y-address is zero and the x-address is larger than zero
- both, the y-address and the x-address are zero

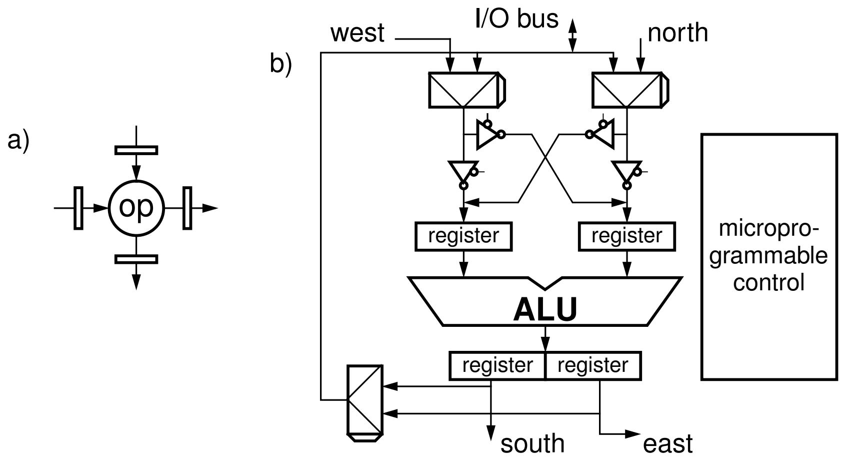


Fig. 2. A datapath unit (a) and its implementation (b)

In the first case the DPU checks if the neighbouring DPUs are busy. If the neighbouring DPU in y-direction is not busy, the y-address will be decreased by one and the resulting configuration word will be transferred to this DPU. If the DPU in y-direction is busy and the DPU in x-direction is not busy the x-address will be decreased by one and the resulting configuration word will be transferred to this DPU. If both neighbouring DPUs are busy, the DPU waits until one finishes. With this strategy an automatic load distribution for the configuration is implemented. Internally the configuration words are distributed over the whole array and several serial links are used to configure the rest of the chips. An optimal sequence of the configuration words can be determined since these can be interchanged arbitrarily.
In the second case, the y-address will be decreased by one and the configuration word will be transferred to the next DPU in y-direction. In the third case when the y-address is zero and the x-address is larger than zero, the x-address will be decreased by one and the configuration word will be transferred in x-direction. In the last case when both addresses are zero, the target DPU is reached, and the address of the DPU's configuration memory shows the place where the data will be written.
Because of the load distribution in the rDPA array, one serial link at the array boundary is sufficient to configure the complete array. The physical chip boundaries are completely transparent to the user. The communication structure allows dynamic in-system reconfiguration of the rDPA array. This implies partial reconfigurability during runtime [6]. Partial reconfigurability is provided since all DPU can be accessed individually. The configurability during runtime is supported because each DPU forwards a configuration word with higher priority than starting with the next operation. The load distribution takes care of that most of the configuration words avoid the part of the rDPA array which is in normal operation. Further the configuration technique allows to migrate designs from a smaller array to a larger array without modification. Even newer generation rDPA chips with more DPUs integrated do not need a recompilation of the configuration data. The configuration is data-driven, and therefore special timing does not have to be considered.
With the proposed model for the DPA, the array can be expanded also across printed circuit board boundaries, e. g. with connectors and flexible cable. Therefore it is possible to connect the outputs of the east (south) array boundary with the west (north) one, to build a torus.

3 Support Chip for Bus-Oriented Systems

With the rDPA, a programmable support chip for bus-oriented systems is provided. Together they form a data-driven reconfigurable ALU (rALU). The support chip consists of a control unit, a register file, and an address generation unit for addressing the DPUs (figure 3).
The register file is useful for optimizing memory cycles, e. g. when one data word of a statement will be used later on in another statement. Then the data word does not have to be read again over the external bus. In addition, the register file makes it possible to use each DPU in the rDPA for operations by using the internal bus for routing. If different expressions have a common subexpression, this subexpression has to be computed only once. If the rDPA does not provide the routing capacity for this reduction,

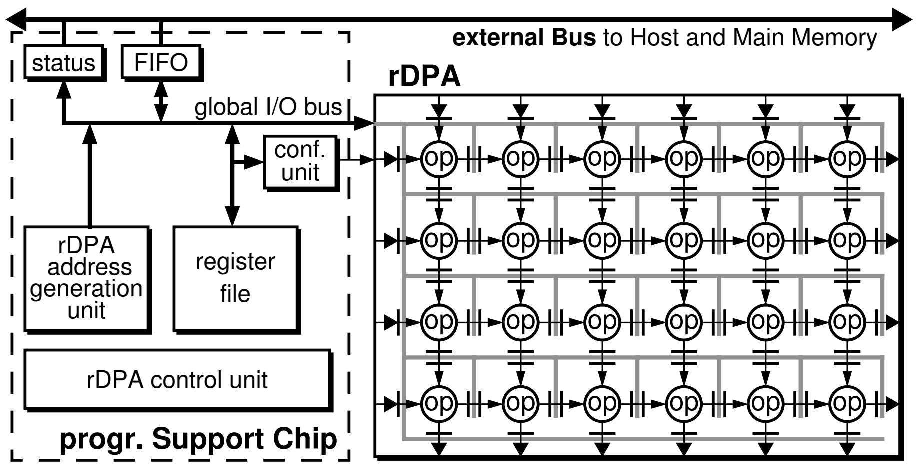


Fig. 3. The reconfigurable datapath architecture (rDPA) with the programmable support chip

e. g. if three or more subexpressions are in common, the interim result can be routed through the register file.
The address generation unit delivers the address for the DPU registers before each data is written into the rDPA over the bus. Usually the address is increased by one but it can also be loaded directly from the rDPA control unit.
The rDPA control unit holds a program to control the different parts of the data-driven rALU. The instruction set consists of instructions for loading data into the rDPA array to a special DPU from the external units, for receiving data from a specific DPU, or branches on a special control signal from the host. The rDPA control unit supports context switches between three control programs which allows the use of three independent virtual rALU subnets. The control program is loaded during configuration time. The reconfigurable data-driven ALU allows also pipelined operations.
A status can be reported to the host to inform about overflows, or to force the host to deliver data dependent addresses. The input FIFO is currently only one word deep for each direction. The datapath architecture is designed for an asynchronous bus protocol, but it can also be used on a synchronous bus with minor modifications of the external circuitry.

4 Programming Environment

Statements which can be mapped to the rDPA array are arithmetic and logic expressions, and conditions. The input language for programming the rALU including the rDPA array is the rALU programming language, called ALE-X (arithmetic & logic expressions for Xputers). The syntax of the statements follows the C programming language syntax. A part of an ALE-X example is shown in figure 4.
A data dependency analysis is performed to recognize possible parallelization and to find dependencies between the statements. The statements are combined to larger expressions and a data structure which is a kind of an abstract program tree is built (figure 5). Then the data structure is mapped onto the rDPA array structure. The map-

```
a = b + c * d; (1)
if (e < 10)    (2)
  f = g + h;   (3)
else           (4)
  f = g - h;   (5)
i = c + f;     (6)
```

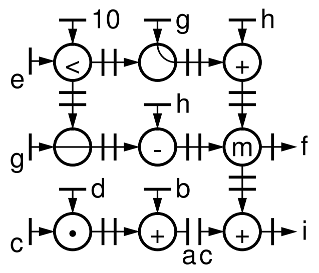


Fig. 4. Part of an ALE-X program example mapped onto the rDPA array

ping algorithm starts at the leaf cell nodes of the data structure for each expression. These nodes are assigned to DPUs in a first line of the rDPA array. A second line starts if there is a node of higher degree in the data structure. The degree of a node increases if both sons of the node are of the same degree. After that the mapped structure is shrunk by removing the nodes which are used only for routing. There are several possibilities for the mapping of each expression. Finally the mapped expression with the smallest size is chosen. Figure 5 shows an example of the mapping. Now the mapped expressions are allocated in the rDPA array, starting with the largest expression. If the expressions do not fit onto the array, they are split up using the global I/O bus for routing. If the number of required DPUs is larger than the number of DPUs provided by the array, the array has to be reconfigured during operation. Although this allocation approach gives good results, future work will be done in the optimization of this algorithm to incorporate the scheduling process for advance timing forecast.
Due to the global I/O bus of the rDPA array, the loading of the data and the storing are restricted to one operation per time. An optimal sequence of these I/O operations has to be determined. For the example in figure 4, starting with loading the variables `c` and `d` is better than starting with `h`. The operators do not have to be scheduled, since they are available all the time. The operands have to be scheduled with the additional restriction that operands used at multiple locations have to be loaded several times at succeeding time steps. For example, when the variable `c` is scheduled, the `c` of the multiplication and the `c` of the last addition have to be loaded in direct sequence. To find the time critical operations, first an ‘as soon as possible’ schedule (ASAP) and an ‘as late as possible’ schedule (ALAP) are performed. No other resource constraints

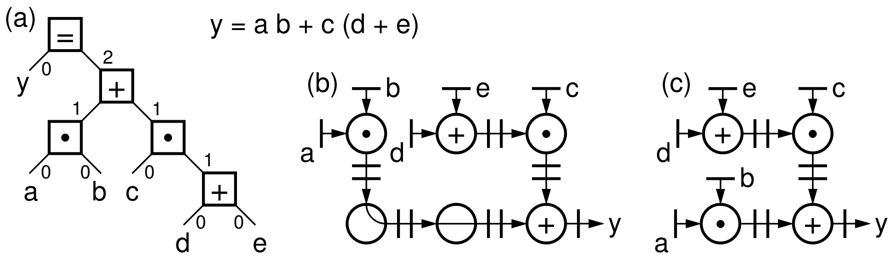


Fig. 5. Example of the mapping process: a) data structure with the degree of the node, b) mapped structure, c) shrunk mapped structure

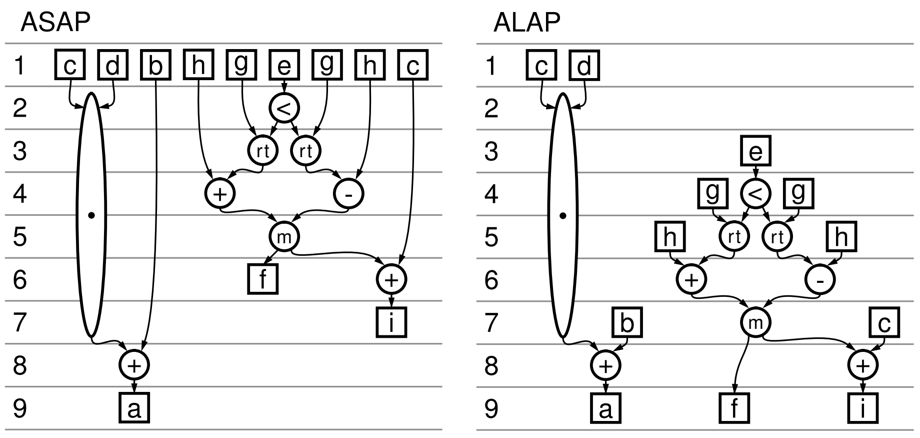


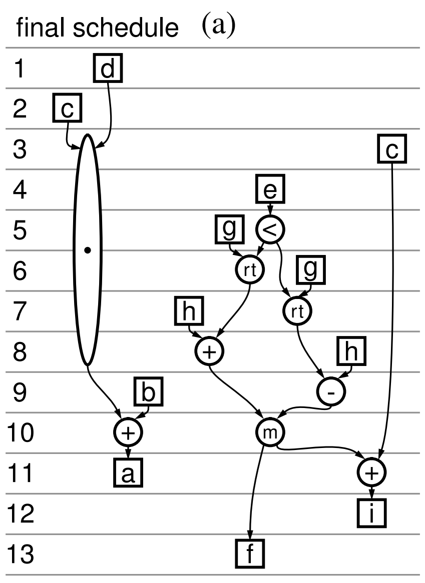
Fig. 6. ASAP and ALAP schedules for the program example

(like only a single I/O operation at a time) are considered at this moment. For simplicity in our example, all operations, including the route operations (rt) are assumed to need a single time step for finishing in the worst case. The multiplication is assumed to need six time steps. The rALU compiler considers the real time delays in the worst case. Due to the self timing of the data-driven computation, no fixed time step intervals are necessary. Figure 6 shows the ASAP and ALAP schedules for the program example.

Comparing the ASAP with the ALAP schedule, the time critical path is found. It is the multiplication of `c` and `d` with the succeeding addition of `b`. The range of this operation is zero. A priority function is developed from these schedules which gives the range of the I/O operations of the operands. This is the same as in a list based scheduling [2]. The highest priority i. e. the lowest range have the variables `c` and `d`. Since `c` has to be loaded twice, `d` is loaded first. The complete priority function is listed in figure 7b.

When the variable `c` is scheduled twice in direct sequence the ASAP and the ALAP schedule may change because of the early scheduling of `c` in the addition operation. Then the schedule of `d`, `c`, and `c` is kept fixed and a new priority function on the remaining variables is computed to find the next time critical operation. For simplicity this is not done in the illustration. Figure 7a shows the final schedule of the program example.

In time step 10 no I/O operation is performed. If the statement block of the example is evaluated several times, the global I/O bus can be fully used by pipelining the statement block. The pipeline is loaded up to step 9. Then the variable `d` from the next block is loaded before the output variables `a`, `i` and `f` are written back. The statement block is computed several times (step 10 to 21, figure 7c) until the host signals the rALU control to end the pipeline. Step 22 to the end is performed, and the next operators can be configured onto the rDPA array.



(b)

priority	read	write
0	d	a
0’	c	-
0”	c	-
1	-	-
2	e	i
3	g, g	f
4	h, h	-
5	-	-
6	b	-

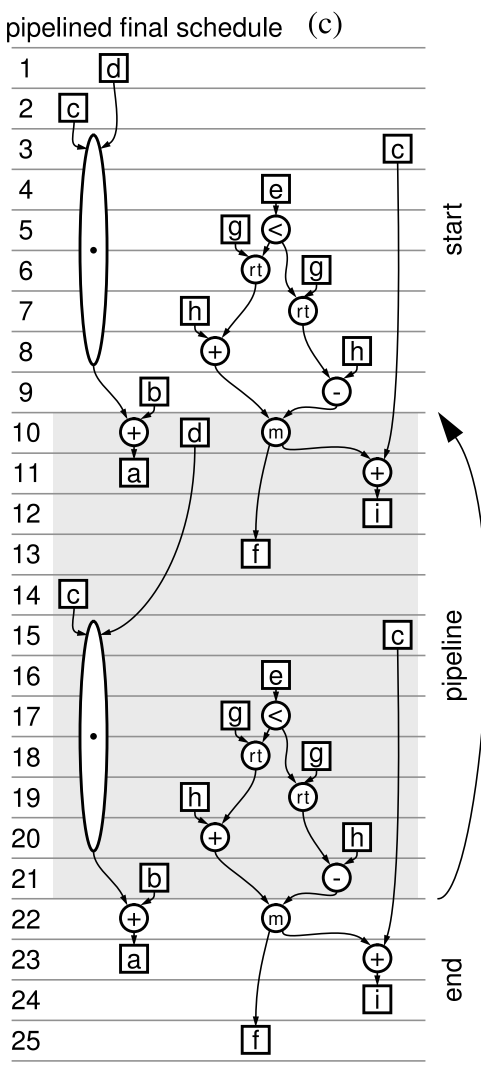


Fig. 7. The final schedule (a), the priority function (b) and the pipelined final schedule (c) for the program example

The rDPA configuration file is computed from the mapping information of the processing elements and a library with the microprogram code of the operators. The configuration file for the rALU control unit is extracted from the final schedule of the I/O operators.

5 Utilisation with the Xputer Hardware Environment

Although the proposed rALU can be used for any bus-oriented host based system, it is originally build for the Xputer prototype Map-oriented Machine 3 (MoM-3). The Xputer provides a hardware and a software environment for a rALU. The rALU has to compute a user defined compound operator only. A compiler for the Xputer supports the high level language C as input [8]. The rALU programming environment has to compile arithmetic and logic expressions as well as conditions onto the rALU.
Many applications require the same data manipulations to be performed on a large amount of data, e. g. statement blocks in nested loops. Xputers are especially designed to reduce the von-Neumann bottleneck of repetitive decoding and interpreting address and data computations. In contrast to von Neumann machines an Xputer architecture strongly supports the concept of the "soft ALU" (rALU). The rALU allows for each application a quick problem-oriented reconfiguration. High performance improvements have been achieved for the class of regular, scientific computations [5], [1].
An Xputer consists of three major parts: the data sequencer, the data memory and the rALU including multiple scan windows and operator subnets. Scan windows are a kind of window to the data memory. They contain all the data words, which are accessed or modified within the body of a loop. The data manipulations are done by the rALU subnets, which provide parallel access to the scan windows. The scan windows are updated by generic address generators, which are the most essential part of the data sequencer. Each generic address generator can produce address sequences which correspond to nested loops under hardware control. The term data sequencing derives from the fact that the sequence of data triggers the operations in the rALU, instead of a von-Neumann instruction sequence. Generally, for each nesting level of nested loops a separate rALU subnet is required to perform the computations associated with that

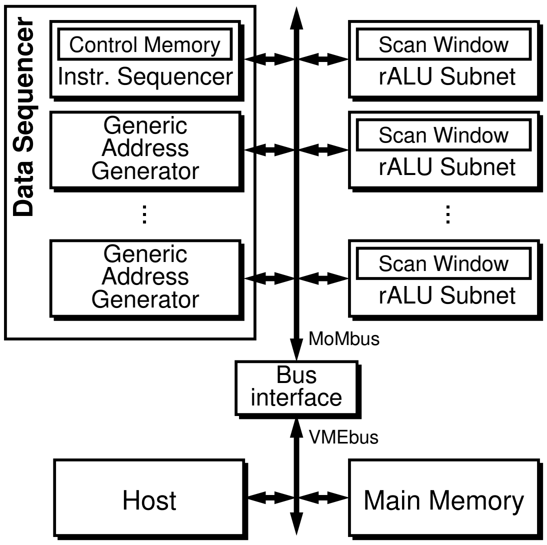


Fig. 8. The Xputer prototype Map-oriented Machine 3 (MoM-3)

nesting level. The rALU subnets perform all computations on the data in the scan windows by applying a user-configured complex operator to that data. Pipelining across loop boundaries is supported. The subnets need not to be of the same type. Subnets can be configured for arithmetic or bit level operations.
The Xputer prototype MoM-3 has direct access to the host's main memory. The rALU subnets receive their data directly from a local memory or via the MoMbus from the main memory. The MoMbus has an asynchronous bus protocol. The datapath architecture is designed for the asynchronous bus protocol of the MoMbus, but it can also be used by a synchronous bus with minor modifications. Figure 8 shows our prototype MoM-3.
A complete rALU programming environment is developed for the rALU when using it with the Xputer prototype. The input language for programming the rALU is the ALE-X programming language. The syntax of the statements follows the C programming language syntax (see also figure 4). In addition, the language provides the size of the scan windows used and the next handle position which is the lower left corner of the boundary of the scan window. Providing the handle position gives the necessary information for pipelining the complete statement block in the rALU.
The ALE-X programming language file is parsed and a data structure like an abstract program tree is computed. Common subexpressions are taken into consideration. The

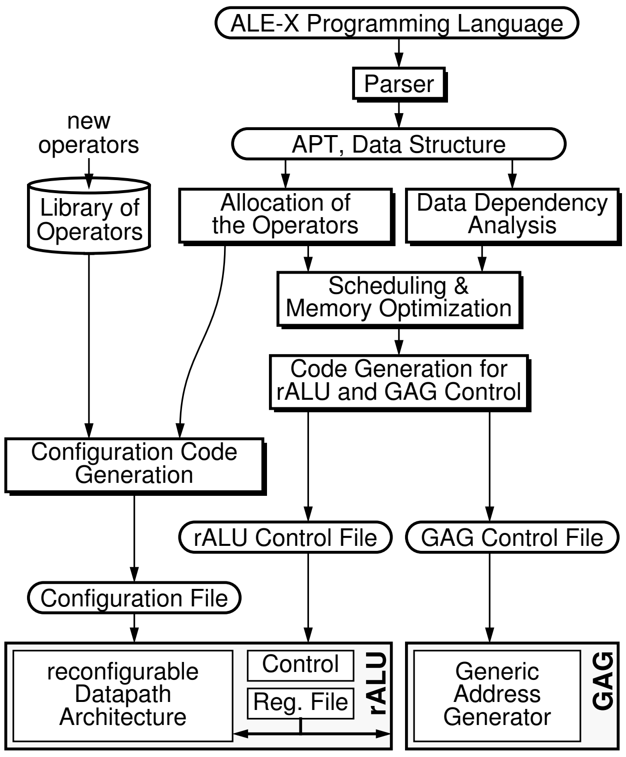

ALE-X Programming Language
Parser
new operators
APT, Data Structure
Library of Operators
Allocation of the Operators
Data Dependency Analysis
Scheduling & Memory Optimization
Code Generation for rALU and GAG Control
Configuration Code Generation
rALU Control File
GAG Control File
Configuration File
reconfigurable Datapath Architecture
Control
Reg. File
rALU
Generic Address Generator
GAG

Fig. 9. The rALU programming environment

operators of each statement are associated to a DPU in the rALU array as described in section 4. Memory cycles can be optimized using the register file when the scan pattern of the GAGs works with overlapping scan windows.
The rDPA configuration file is computed from the mapping information of the DPUs and a library containing the code of the operators. The configuration file for the rALU control unit and the configuration file for the GAGs is extracted from the final schedule of the I/O operators. The programming environment of the rALU is shown in figure 9.

6 Results

The prototype implementation of the rDPA array works with 32 bit fixed-point and integer input words. Currently the host computer's memory is very slow. The clock frequency of the system is 25 MHz. In a single chip of the rDPA array fits at least a 3×3 matrix of DPUs. In many applications the coefficients in e. g. filter implementations are set up in such a way that shift operations are sufficient and multiplications are not necessary. If high throughput is needed the DPU processing elements can be linked together to implement a pipelined multiplier for example. Benchmark results are given in table 1. The performance figures are a worst case estimation of our prototype. They give the duration of the operation time per data word. The speed of the examples 2 to 5 does not depend on the order of the filter as long as the necessary hardware (number of DPUs) is provided. The same applies for example 6.

#	Algorithms	Operations	number of active DPUs	number of necessary chips	Time Steps per Operation	Performance
1	1024 Fast Fourier Transform	*, +, -	10	2	$16 \cdot 10240$	20 ms
2	FIR filter, nth order	*, +	2(n+1)	$\left\lceil \frac{n+1}{3} \right\rceil$ [a]	15	1800 ns / data word
3	FIR filter, nth order	shift, +	2(n+1)	$\left\lceil \frac{n+1}{3} \right\rceil$	4	500 ns / data word
4	n × m two dim. FIR filter	*, +	2(n+1)(m+2)-1	$\left\lceil \frac{n+1}{3}(m+2) \right\rceil$	15	1800 ns / data word
5	n × m two dim. FIR filter	shift, +	2(n+1)(m+2)-1	$\left\lceil \frac{n+1}{3}(m+2) \right\rceil$	4	500 ns / data word
6	Bubblesort, length n	scan-max	n-1	$\left\lceil \frac{n-1}{9} \right\rceil$	2	240 ns / data word

Table 1. Benchmark results

a. $\lceil x \rceil$ = the smallest integer, which is greater or equal to x

7 Conclusions

An FPGA architecture (reconfigurable datapath architecture, rDPA) for word-oriented datapaths has been presented. Pipelining is supported by the architecture. The word-orientation of the datapath and the increase of the fine granularity of the basic opera-

tions extremely simplifies the automatic mapping onto the architecture. The extendable rDPA provides parallel and pipelined evaluation of the compound operators. The rDPA architecture can be used as reconfigurable ALU for bus-oriented host based systems as well as for rapid prototyping of high speed datapaths. It suits very well for the Xputer prototype MoM-3. The architecture is in-system dynamically reconfigurable, which implies also partial reconfigurability at runtime.
A prototype chip with standard cells has been completely specified with the hardware description language Verilog and will be submitted for fabrication soon. It has 32 bit datapaths and provides arithmetic resources for integer and fixed-point numbers. The programming environment is specified and is currently being implemented on Sun SPARCstations.

References

1. A. Ast, R. W. Hartenstein, H. Reinig, K. Schmidt, M. Weber: A General purpose Xputer Architecture derived from DSP and Image Processing; in M. A. Bayoumi (Ed.): VLSI Design Methodologies for Digital Signal Processing Architectures; Kluwer Academic Publishers, Boston, London, Dordrecht, pp. 365-394, 1994
2. D. D. Gajski, N. D. Dutt, A. C.-H. Wu, S. Y.-L. Lin: High-Level Synthesis, Introduction to Chip and System Design; Kluwer Academic Publishers, Boston, Dordrecht, London, 1992
3. S. A. Guccione, M. J. Gonzalez: A Data-Parallel Programming Model for Reconfigurable Architectures; IEEE Workshop on FPGAs for Custom Computing Machines, FCCM'93, IEEE Computer Society Press, Napa, CA, pp. 79-87, April 1993
4. D. Hill, B. Britton, B. Oswald, N.-S. Woo, S. Singh, C.-T. Chen, B. Krambeck: ORCA: A New Architecture for High-Performance FPGAs; in H. Grünbacher, R. W. Hartenstein (Eds.): Field-Programmable Gate Arrays, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1993
5. R. W. Hartenstein, A. G. Hirschbiel, M. Riedmüller, K. Schmidt, M. Weber: A Novel ASIC Design Approach Based on a New Machine Paradigm; IEEE Journal of Solid-State Circuits, Vol. 26, No. 7, July 1991
6. P. Lysaght, J. Dunlop: Dynamic Reconfiguration of Fieldprogrammable Gate Arrays; Proceedings of the 3rd International Workshop on Field Programmable Logic and Applications, Oxford, Sept. 1993
7. N. Petkov: Systolische Algorithmen und Arrays; Akademie-Verlag, Berlin 1989
8. K. Schmidt: A Program Partitioning, Restructuring, and Mapping Method for Xputers; Ph. D. Thesis, University of Kaiserslautern, 1994
9. N. N.: The XC4000 Data Book; Xilinx, Inc., 1992