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

A Dynamically Reconfigurable Wavefront Array Architecture for Evaluation of Expressions

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 reconfigurable wavefront array rDPA (reconfigurable datapath architecture) for evaluation of any arithmetic and logic expression is presented. Introducing a global I/O bus to the array simplifies the use as a coprocessor in a single bus oriented processor system. Fine grained parallelism is achieved 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 extendible to arbitrarily large arrays and dynamically in-circuit 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 reconfigurable ALU for bus oriented host based systems as well as for rapid prototyping of high speed datapaths.

1. Introduction

Like other application-specific architectures, the proposed reconfigurable array architecture serves as medium to map algorithms onto hardware. Statement blocks in inner loops of high performance applications should be evaluated as fast as possible. Mono-processor systems need a kind of reconfigurable coprocessor to accelerate such blocks. Reconfigurable wavefront arrays suit well for such applications since their data-driven computation is self-timed and pipelining can easily be performed [5]. The high parallel I/O requirements of the wavefront array make it difficult to connect such an array to a bus oriented mono-processor system. This leads to the idea to integrate the bus as an auxiliary structure into the array. To distinguish this architecture from conventional wavefront arrays, it is called reconfigurable datapath architecture (rDPA). In this architecture the decentralized control of a systolic array is combined with the centralized control for I/O operations.

Mapping statement blocks or algorithms onto wavefront arrays is done for example by simulated annealing with a trade-off between minimizing area (number of processing elements) and performance [7]. These methods map the data flow graph onto the target array and have to consider local data dependencies. In the rDPA all statements can be mapped separately since variables without local dependencies can be routed to different places in the array using the bus. The bus includes a performance drawback since the I/O operations can not be performed in parallel. Optimized scheduling must ensure that the input data is in a perfect sequence for high performance. 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). 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. Xputers require memory accesses for data only, whereas von-Neumann computers require memory accesses for instructions also. 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 [4], [1]. The Xputer provides a hardware and software environment for the rDPA array. Together with a control part (programmable support chip), the rDPA forms a data-driven reconfigurable ALU (rALU). The control part addresses the processing elements, (called datapath units, DPUs) in the rDPA for the I/O operations. A DPU can perform arithmetic, logical, routing and synchronization operations.

First the paper gives an overview on the hardware and software environment. The rDPA array is described in section 3. Section 4 explains the data-driven rALU which uses the rDPA. The programming environment allows the mapping of operands and conditions to the rDPA automatically (section 5). The scheduling algorithm is described in detail. Finally some benchmark results are shown and the paper is concluded.

2. Hardware and software environment

The hardware of an Xputer consists of three main parts: the data memory, the data sequencer and the reconfigurable ALU (rALU). The data memory contains all the data necessary for the computation. It is organized as a two dimensional data map. The data sequencer provides the access to the data. 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. The most essential part of the data sequencer are the generic address generators (GAGs). Each GAG can produce address sequences which correspond to loops under hardware control and each GAG updates one scan window. The scan windows contain all the data which are accessed or modi-

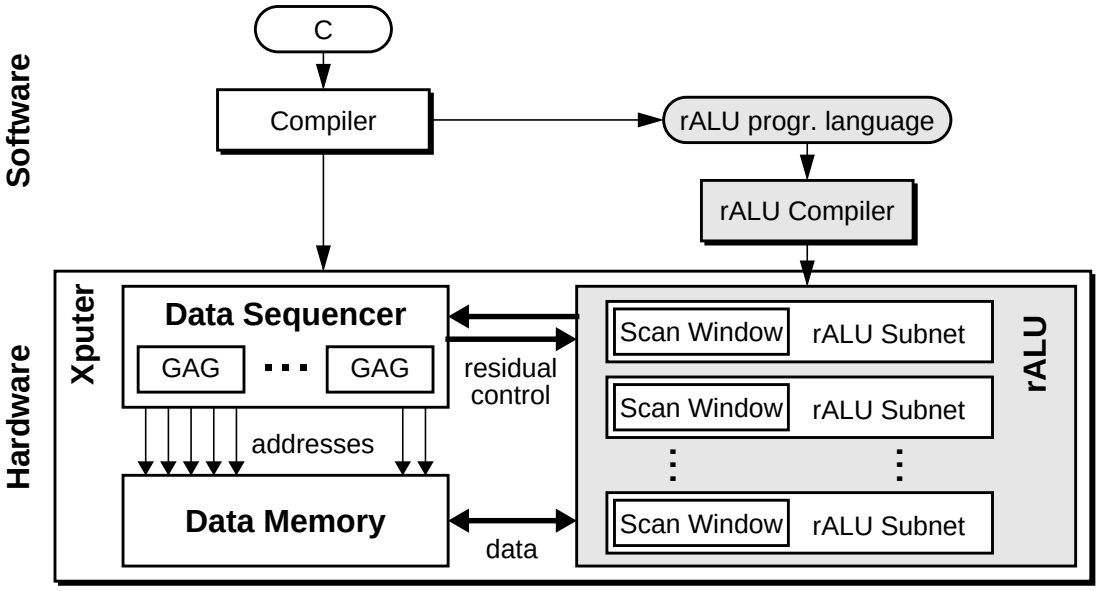


Figure 1. Hardware and software environment for the rDPA array

fied by the subnets. They are a kind of window to the data memory. The reconfigurable ALU is the data manipulator of the Xputer. It consists of several subnets. Each subnet has parallel access to the scan window. The complex operator in the subnet is reconfigurable. Subnets need not to be of the same type. Subnets can be configured for arithmetic or bitlevel operations (figure 1).

Configuration of the rDPA array and its support chip is done with the rALU programming language. The syntax of the statements follows the C programming language syntax (see also figure 6). 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 rALU programming language can be computed with a compiler from the C programming language or it can be written by hand. The C compiler [9] arranges the way data is distributed in the data memory, so that data can be accessed efficiently. It computes also the code for the data sequencer to perform the loops.

3. Reconfigurable array architecture

The reconfigurable rDPA array architecture 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 two neighbouring DPUs is synchronized by a handshake like in wavefront arrays. This avoids 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 2. The 32 bits of the parallel internal communication path are converted into series of two bit nibbles. 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.

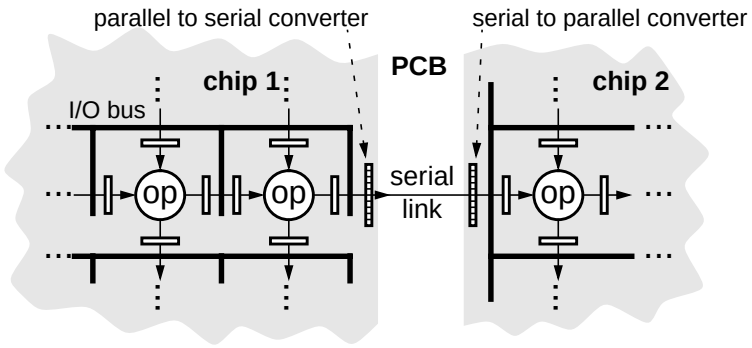


Figure 2. a) processing element DPU (datapath unit), b) the extendible rDPA array between chip boundaries

A global I/O bus has been integrated into the rDPA array, permitting the DPUs to write from the output registers directly to 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. A traditional wavefront array has a decentralized control, whereas the rDPA array has a centralized control for the I/O operations and a decentralized control for the execution of the expressions. A single bus is sufficient for our prototype implementation since it is targeted to a bus-oriented host system with a single memory bus. The communication between the control unit and the DPUs is synchronized by a handshake like the internal communications. Each DPU which has to communicate by using the bus gets an address during configuration. The rDPA control unit can access a DPU directly using the bus. The DPU where the address matches performs the handshake with the control unit and receives or sends the data. The largest bunch of communication occurs during the computations and uses the internal communication paths. Therefore this is done completely in parallel.

The operators of the DPUs are configurable. A DPU is implemented using a fixed ALU and a microprogrammable control as shown in figure 3. 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.

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 is 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. A simple example is shown in 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.

An extensible set of operators for each DPU is provided by a library. The set includes the operators of the programming language C. Also other operators such as parallel prefix operators are provided. The parallel prefix operator [3] has an internal feedback. For example a queue of ‘scan-max’ operators can be used easily for implementation of a hardware bubble sort [8]. 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. Operators can also be used for routing, even mixed with other operations, e. g. first shift variable `a` to the east, then multiply `a` with `b` and write the result to the south.

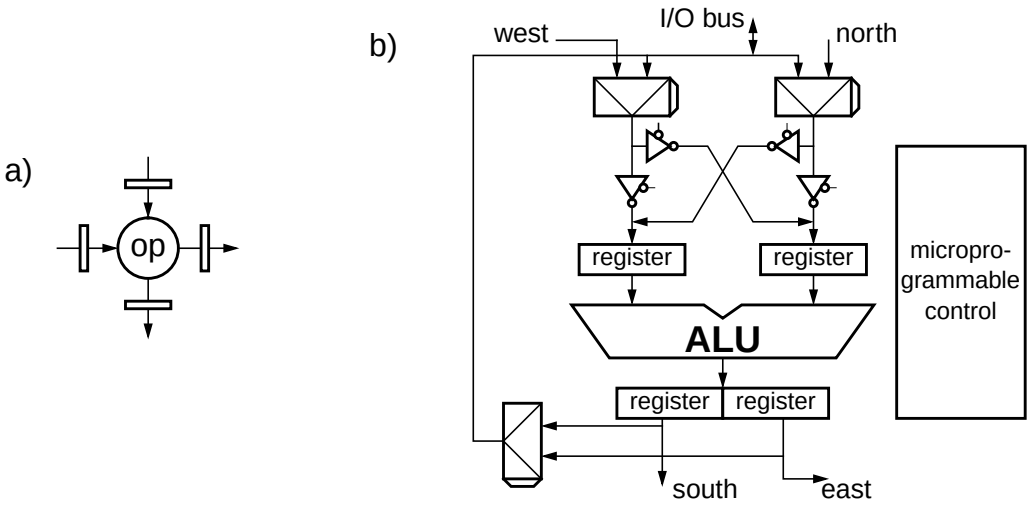


Figure 3. A datapath unit (a) and its implementation (b)

```
if (a < b)        (1)
  c = d + e;      (2)
else              (3)
  c = d - e;      (4)
```

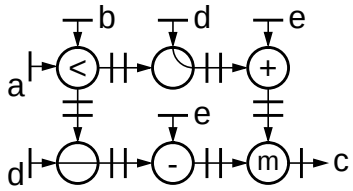


Figure 4. Example configuration to implement an `if_then_else` statement

As mentioned before the array is extendible by using several chips of the same type. The DPUs have no address before configuration since all chips are identical. The DPUs are identified by their location in the rDPA array. Consequently each DPU has an x- and a y-address like the elements in a matrix. 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

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 a 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-circuit 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.

4. The rDPA array used as a reconfigurable ALU

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 5).

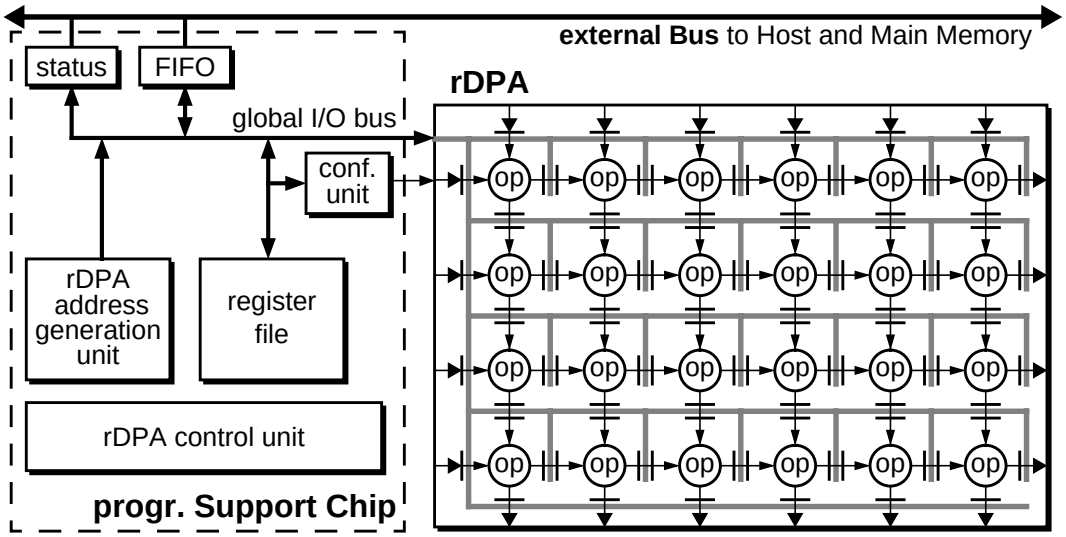


Figure 5. The reconfigurable data-driven ALU

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, e. g. if a subexpression is common to three or more expressions, 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. The addresses of the DPU registers are configured in a way, that the address has to be increased by one only, but it can be loaded directly from the rDPA control unit also.

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, which are all configured in the same rDPA array. 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.

5. Programming the rDPA array

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. The syntax of the statements follows the C programming language syntax. A part of a rALU programming language example is shown in figure 6.

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

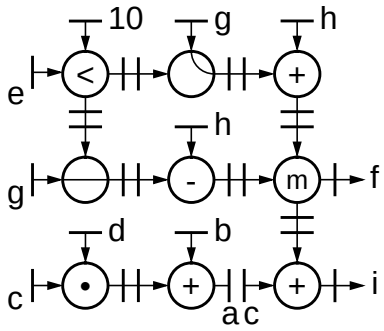


Figure 6. rALU program example mapped in the rDPA array

The rALU programming language is parsed. 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 7). Then the data structure is mapped onto the rDPA array structure. The mapping 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 7 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

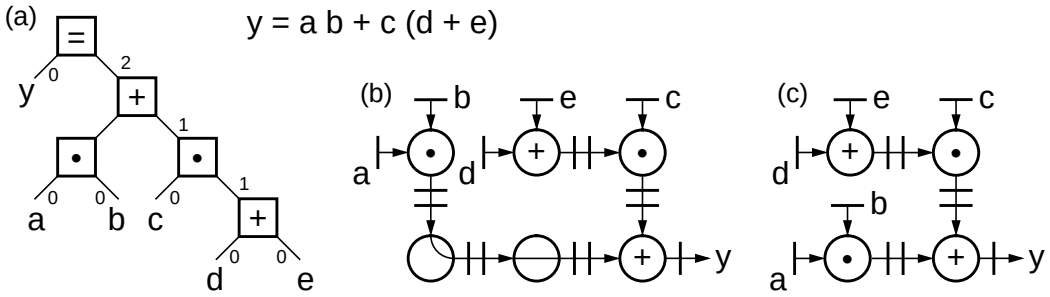


Figure 7. Example of the mapping process: a) data structure with the degree of the node, b) mapped structure, c) shrunk mapped structure

determined. Returning to the example in figure 8, starting with loading the variables c and d is better than starting with h because the multiplication of c and d is in the critical path. 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 more than one time have to be loaded several times at consecutive time steps. So the same variable does not have to be loaded twice via the global I/O bus. 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 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 8 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, succeeded by the 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 9b. 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 9a 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 9c) 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.

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

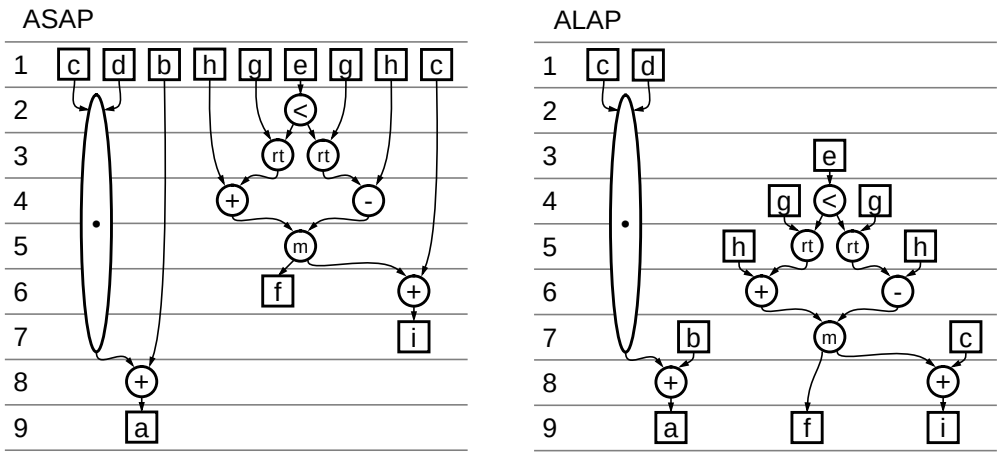
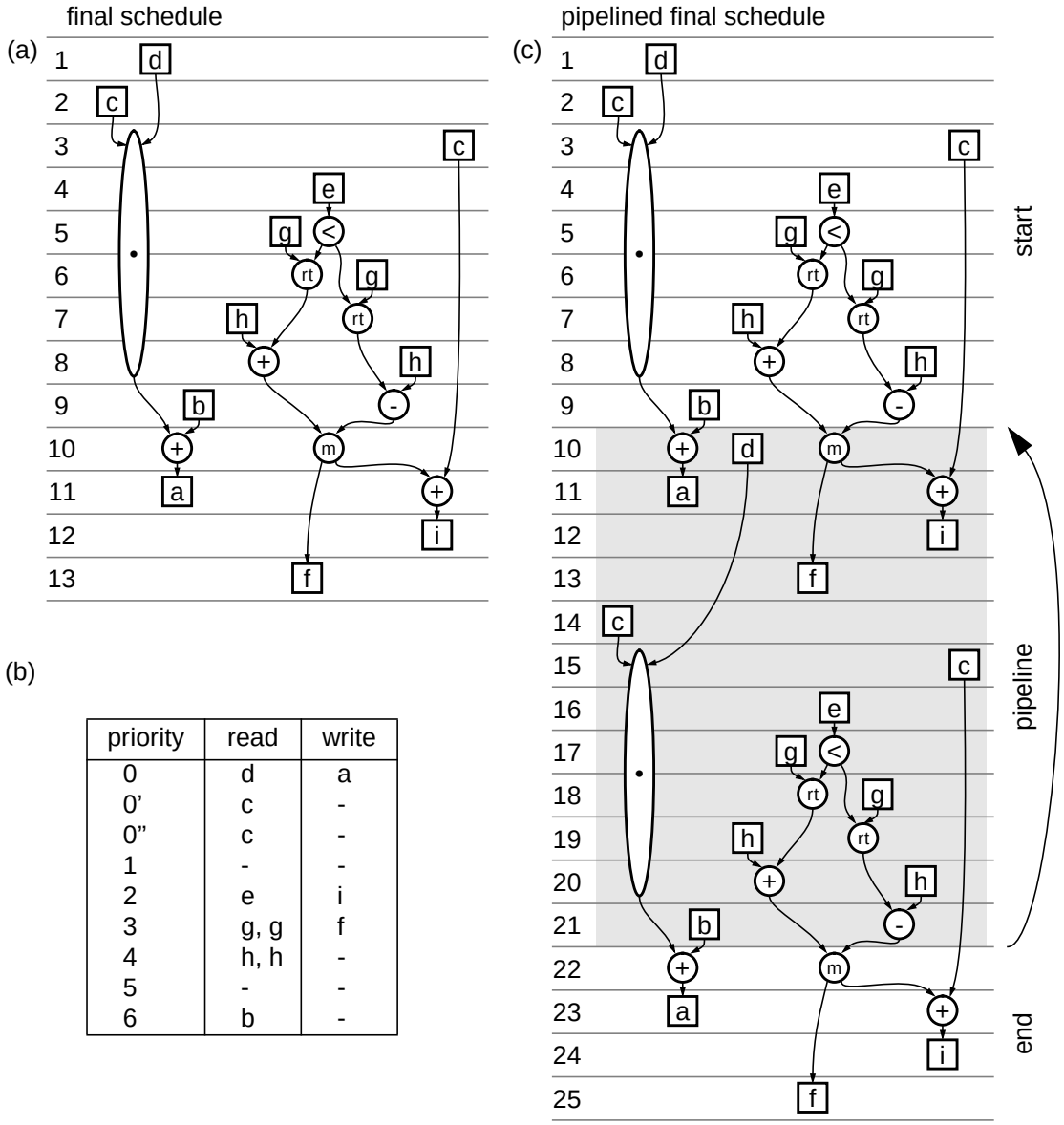


Figure 8. ASAP and ALAP schedule for the program example



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

Figure 9. The final schedule (a), the priority function (b) and the pipelined final schedule (c) for the program example

the rALU control unit is extracted from the final schedule of the I/O operators. An overview of the programming environment is given in figure 10.

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 used from the host is very slow. The clock frequency of the system is 25 MHz. In many applications the coefficients in e. g. filter implementation 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

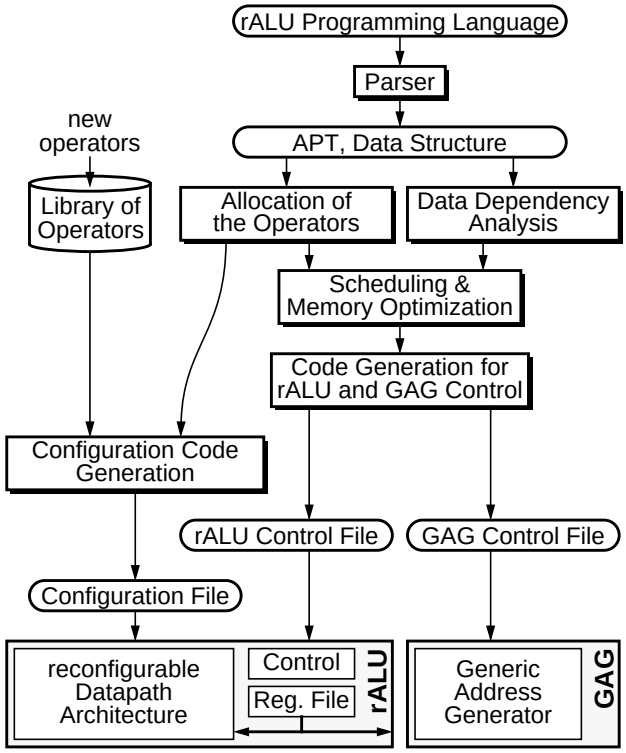


Figure 10. The rALU programming environment

implement a pipelined multiplier. 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. For the Fast Fourier Transform (FFT) one butterfly operation is implemented for complex figures. The weights are loaded by the I/O bus like the data words. One 1024 point FFT requires 20 ms for completion. The finite impulse response (FIR) filter is implemented as shown in figure 11. The 'h' are the coefficients and the 'x[ ]' is the input data stream. The multiplications with the coefficients 'h_0' till 'h_{N-1}' perform first a multiplication and route the result to the south, and then they route the 'x' input word to the next DPU at the east. At the beginning all DPUs with multiplications are preloaded with zero via the I/O bus to fill the pipeline. Then the 'x' input words have to be provided at the west input only. The filter produces a new valid output word every 500 ns by using 'shifts' instead of 'multiplications'. The bubble-sort works with a linear chain of 'scan-max' operators producing a new sorted data word every 240 ns. The speed of the examples 2 to 5 depend not on the order of the filter as long as the necessary hardware (number of DPUs) is provided. The same is true for example 6.

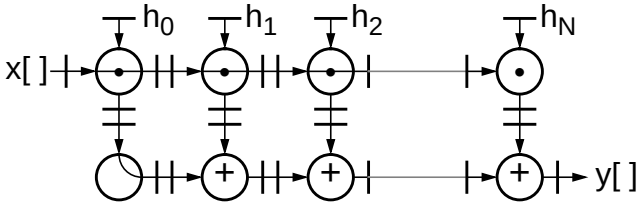


Figure 11. Finite impulse response (FIR) filter implementation

#	Algorithms	Operations	number of DPUs	Time Steps per Operation	Performance
1	1024 Fast Fourier Transf.	*, +, -	10	16 · 10240	20 ms
2	FIR filter, n^{th} order	*, +	2(n+1)	15	1800 ns/data word
3	FIR filter, n^{th} order	shift, +	2(n+1)	4	500 ns/data word
4	n × m two dim. FIR filter	*, +	2(n+1)(m+2)-1	15	1800 ns/data word
5	n × m two dim. FIR filter	shift, +	2(n+1)(m+2)-1	4	500 ns/data word
6	Bubblesort, length n	scan-max	n-1	2	240 ns/data word

Table 1. Benchmark results

7. Conclusions

A reconfigurable wavefront array rDPA (reconfigurable datapath architecture) for evaluation of any arithmetic and logic expression 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 operations greatly simplifies the automatic mapping onto the architecture. The extendible rDPA provides parallel and pipelined evaluation of the compound operators. The rDPA array is originally built for the Xputer prototype MoM-3 but it can be used also as reconfigurable ALU for bus oriented host based systems as well as for rapid prototyping of high speed datapaths. The architecture is in-circuit dynamically reconfigurable, which implies also partial reconfigurability at runtime.

A prototype chip with standard cells has been completely specified with the hardware description 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 being implemented on Sun SPARCstations.

Future work is the implementation of a pipelined version of the processing elements of the rDPA and the optimization of the mapping algorithm.

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, Dortrecht, 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] 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

[5] S. Y. Kung: VLSI Array Processors; Prentice Hall, Englewood Cliffs, New Jersey, 1988

[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] B. Mendelson, I. Koren: Mapping onto a Multiple-Chip Data-Driven Array; Proceedings of the International Conference on Application-Specific Array Processors, IEEE Computer Society Press, Los Alamitos, California, Oct. 1993

[8] N. Petkov: Systolische Algorithmen und Arrays; Akademie-Verlag, Berlin 1989

[9] K. Schmidt: A Restructuring Compiler for Xputers; Ph. D. Thesis, University of Kaiserslautern, 1994