A Reconfigurable Data-Driven ALU for Xputers

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 data-driven datapath architecture for ALUs is presented which may be used for custom computing machines (CCMs), Xputers (a class of CCMs) and other adaptable computer systems as well as for rapid prototyping of high speed datapaths. 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. The programming environment allows automatic mapping of the operators from high level descriptions. Two implementations, one by FPGAs and one with standard cells are shown.

1. Introduction

For numerical computations with custom computing machines, word-oriented datapaths are necessary. A recent trend in FPGA architectures moves toward support of efficient implementation of datapath circuits. Xilinx XC4000 series [11] provides fast 2-bit addition at each logic cell by a special carry circuit. AT&T's ORCA [6] supports larger data manipulations. For example a 16 bit adder requires only four function blocks. Word-oriented datapaths are not directly supported by FPGAs currently available, since these circuits are evaluated for both random logic control and datapath applications. Word-oriented datapaths in reconfigurable circuits have the additional advantage of operators being mapped more efficiently. Thus they support programming environments for custom computing machines.

Hardware designers usually have no problem in using custom computing machines, since most of the programming models these machines provide are at hardware level. Algorithms have to be expressed by a hardware description language. Then they have to be synthesized to the dedicated hardware, or the application has to be edited via a schematic editor and mapped by an FPGA vendor tool [2], [1]. People coming from the software side are more used to program with a procedural high level language. To make custom computing machines more interesting to such people, a procedural model for high level programming is needed. Some custom computing machines, consisting of an accelerator board and a host, support function calls from high level languages to be executed on the accelerator board, but they are not able to configure the board from this level, e. g. [3]. Some researchers start to evaluate compilers for their boards to compile a high level input language directly. Gokhale and Minnich [4] for example present such a compiler to program an array of FPGA chips in a high level parallel C using a SIMD programming model. However still problems are the automatic mapping of loops, arithmetic and logic operators.

This paper introduces an Xputer [8] based methodology solving these problems which strongly supports arithmetic applications. The Xputer provides a hardware and a software environment for a reconfigurable ALU (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. The rALU programming environment has to compile arithmetic and logic expressions as well as conditions into the rALU. The machine paradigm of the Xputer is described in [7].

The reconfigurable data-driven ALU proposed here, consists of the rALU control and the reconfigurable datapath architecture (rDPA). The rDPA is a word-oriented scalable regular array of simple processing elements called datapath units (DPUs). The DPUs provide a higher granularity of the basic function units than usual FPGAs. This supports automatic mapping of arithmetic and logic operators to the rDPA. Using the Xputer with the data-driven rALU allows the programming from a high level language. The architecture does not only fit as a reconfigurable ALU for Xputers, it may also be used for any other

kind of adaptable computer system as well as a universal accelerator coprocessor or for rapid prototyping of high speed datapaths.

First, this paper gives a short introduction to the hardware environment. Section 3 introduces the rDPA. An implementation with commercial FPGA chips and one with standard cells is shown and both are compared to each other. The usage as a rALU for Xputers is explained in section 4. The programming environment allows the mapping of operands and conditions to the rDPA automatically (section 5). The final sections present an example and conclude the paper.

2. Hardware environment

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. High performance improvements have been achieved for the class of regular, scientific computations [7], [8].

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, which are accessed or modified within the body of a loop. The data manipulations are done by the rALU subnets, which have parallel access to the scan windows. The scan windows are updated by their corresponding generic address generators, which are the most essential part of the data sequencer. Each generic address generator can produce address sequences which correspond to up to three 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. Pipelining across loop boundaries is supported. Generally, for each nesting level of nested loops a separate rALU subnet is required to perform the computations associated with that nesting level. For most algorithms, only three nested loops are necessary for the computation, this means that in the worst case three rALU subnets are sufficient. The rALU subnets perform all computations on the data in the scan windows by applying a user-configured complex operator to that data. The subnets need not to be of the same type. Subnets can be configured for arithmetic or bit level operations. The data-driven reconfigurable datapath architecture proposed here is well suited for arithmetic, and in the FPGA version also for bit level operations.

The Xputer prototype, Map-oriented Machine 3 (MoM-3), has direct access to the host's main memory. The rALU subnets receive their data directly from a local memory or

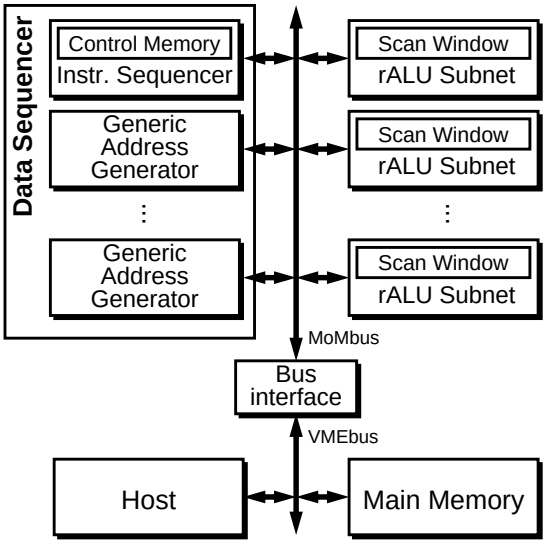

Figure 1. The Xputer prototype Map-oriented Machine 3 (MoM-3)

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 1 shows our prototype MoM-3.

3. Reconfigurable datapath architecture

The benefit of using the reconfigurable datapath architecture (rDPA) within the Xputer paradigm stems from the development environment available. The programmer can benefit from the performance boost of custom operators to match the requirements of the application. He need not know much about hardware description languages or traditional input descriptions for FPGA synthesis tools. Instead the problem is formulated in a familiar procedural programming language and a compiler converts the loops of the application to GAG scan patterns. The statements consisting of expressions and conditions have to be mapped to the rALU only. Section 5 shows the automatic mapping onto the rDPA architecture.

The rDPA is the data manipulator of the data-driven rALU. The rDPA was designed to evaluate any arithmetic and logic expression from a high level description. Therefore the granularity of the basic operation is increased from the bitwise level to wordlevel with possible operations such as addition or multiplication. This greatly simplifies the automatic mapping onto the architecture. A regular structure like in systolic arrays is required for the scalability, also across chip boundaries. Systolic array structures combine intensive local communication and

computation with decentralized parallelism in a compact package. The basic cell of the rDPA is called datapath unit (DPU), see figure 2a. Each DPU has two input and two output registers which may form some part of the scan window of the Xputer paradigm. The dataflow direction is only from west and/or north to east and/or south. The communication between the neighbouring DPUs is synchronized by a handshake. This avoids problems of clock skew and each DPU can have a different computation time for its operator. The operation in the DPU is data-driven, this means that the operation is evaluated when the required operands are available. After completion of the operation, the result in the output register of the DPU is declared to be valid. As soon as the input registers of the succeeding DPU are free, the data will be transferred and the next computation starts. In systolic architectures the high I/O requirements often make the integration into chips a problem. To reduce the number of input and output pins, a serial link is used for data transfer between neighbour DPUs (32 bits are converted into a series of 2 bit nibbles), as shown in figure 2b.

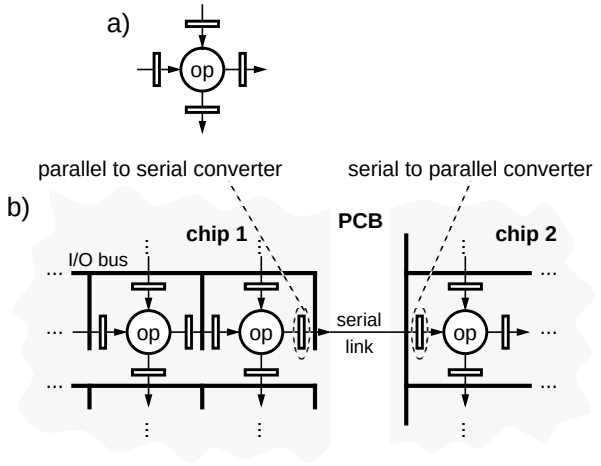

Figure 2. a) datapath unit (DPU), b) the scalable rDPA architecture between chip boundaries

A global I/O bus has been implemented into the rDPA permitting the DPUs to write from the output registers directly outside the array and to read directly from the outside. This means input data to expressions mapped into the rDPA do not need to be routed through the DPUs. Principally, the global I/O bus is used for reading and writing the scan window contents to the array. A single bus is sufficient for our Xputer prototype MoM-3 since the data from the main memory of the host is made available by the bus also. For other systems it may be better to have a dedicated input and output bus. The communication between the outside control unit and the DPUs is synchronized by a handshake like the internal communications.

Each DPU which has to communicate using the global I/O bus gets an address for its input and/or output register during configuration. The rDPA control unit can address a DPU register directly using the bus. The DPU where the address matches performs the handshake with the outside control unit and receives or sends the data. The propagation of interim results, which accounts for the largest bunch of communication, is handled through the internal communication paths, and therefore fully parallel.

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 prefix operator are provided. The parallel prefix operator [5] has an internal feedback. For example a queue of scan-max operators can be used for easy implementation of a hardware bubble sort [10]. 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 multiply `a` with `b` and write the result to the south, then route variable `a` to the east.

Conditions are implemented in the rDPA in the following way. 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 condition operator sends to one neighbouring DPU a true condition bit, to the other one a false bit. The 'false' path is evaluated very quick, because the condition bit is routed only. With this technique also nested `if_then_else` statements can be evaluated. An example is shown in figure 3. The `then` and the `else` path can be merged at the end with a merge

a)
```
if (a < b)                (1)
  if (a > c)              (2)
    x = u + v * w;        (3)
  else                    (4)
    x = u - s * w;        (5)
else                      (6)
  x = w * t + v * z;      (7)
```

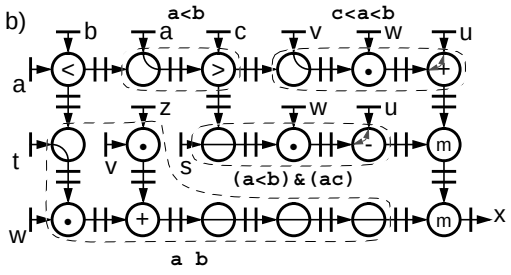

Figure 3. Example of a nested `if_then_else` statement

operator (m). This operator routes the value with the valid condition bit to its output.

With the proposed scalable model for the rDPA, the array can be expanded also across printed circuit board boundaries, e. g. with connectors and flexible cable. This makes it possible to connect the outputs of the east array boundary with the west one, to build a torus. The same is possible for the south and the north. With the data-driven concept, the user need not worry about synchronization.

For the implementation of the rDPA two possibilities are presented. The first is based on FPGAs and the second one is the based on standard cells.

3.1 FPGA implementation of the rDPA

The communication model of the DPU can be implemented into SRAM based FPGAs. A large reconfigurable datapath architecture can be implemented into multiple FPGA chips. The FPGA chips are arranged in an array with a single bus and configuration lines to each chip. All other wiring is local, therefore no field programmable interconnect component (FPIC) is required. On each FPGA, an array of DPUs is implemented depending on the size of the FPGA and the complexity of the operators. If a complex operator like a multiplication is implemented into an FPGA, the other DPUs can only be used for routing or simple operators in this chip. The rDPA programming environment, presented in section 5 takes this fact into consideration.

The implementation of the DPUs using FPGAs has the advantage that speed critical operators can be implemented fully or partly in parallel if necessary. Also the operations itself can be pipelined. The programming environment detects the critical path with the critical operations. Slow ripple carry adders can be substituted by conditional sum adders or other faster adders. A library of operators, which can be extended, is available in the rDPA programming environment. The communication channels are all the same for each DPU. The programmer has to put the macros for the operators and the communication together in a schematic editor. Then the DPUs are mapped with a vendor tool to the FPGA structure. This is done for each FPGA, no partitioning is necessary. Especially bit level operators are well suited for FPGA implementation. The DPUs are smaller, faster and a larger number can be implemented on a chip. Different widths of the datapath can be used also.

3.2 Standard cell implementation of the rDPA

The rDPA implemented with standard cells has a 32 bit datapath. The operators of the DPUs are configurable with a fixed ALU and a microprogrammed control. This means operators such as addition, subtraction or logical operators can be evaluated directly, whereas multiplication or division are implemented sequentially. A library of operators is available and new operators can be build with a microassembler. The standard cell implementation of the proposed model has a higher density of the arithmetic operations and the delay times are known for each operation. The time for synthesis of the rALU board is reduced because the operators in the DPUs are microprogrammed.

As mentioned before the array is scalable by using several chips of the same type. The DPU registers have no address before configuration. The only identification of the DPUs is their location in the rDPA array. Each DPU has an x- and a y-address like the elements in a matrix. A configuration word consists of a configuration bit which distinguish the configuration data from computational data. Further 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. If the y-address is larger than zero, the address will be decreased by one and the configuration word will be transferred to the next DPU in y-direction. If 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. If 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.

One serial link at the array boundary is sufficient to configure the complete array, but multiple ports can be used to save time. The physical chip boundaries are completely transparent to the user. The communication structure allows dynamic in-circuit reconfiguration of the rDPA. This implies partial reconfigurability during runtime [9]. Filter coefficients, for example, can be changed during runtime to build adaptive filters. 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.

4. Reconfigurable data-driven ALU

A subnet of the reconfigurable data-driven ALU consists of two main parts: the control for the rALU and the reconfigurable datapath architecture, as shown in figure 4. Subnets can run in parallel. Here the connection to the MoMbus is shown only.

The register file is necessary for optimizing memory cycles when the generic address generator runs with overlapping scan windows. This means that data of the actual scan window position is required in the consecutive posi-

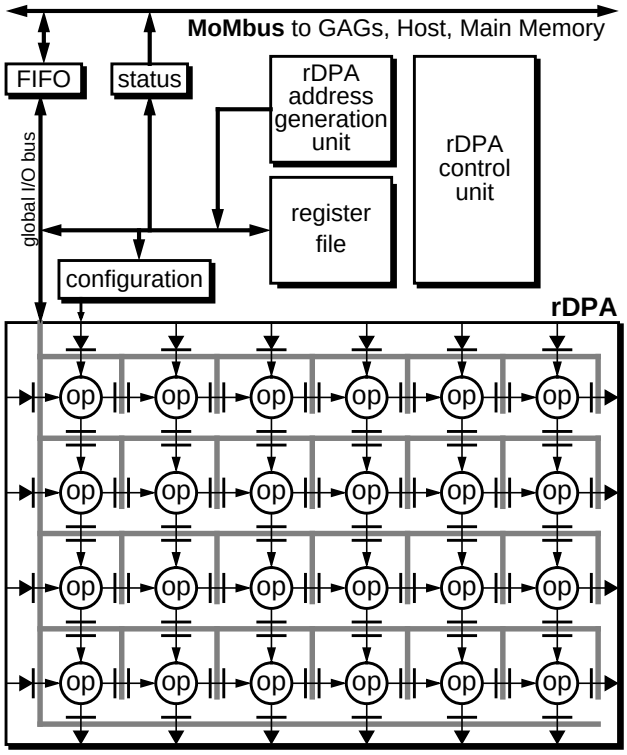

Figure 4. One subnet of the reconfigurable data-driven ALU

tion. This data can be stored in the register file and written back to main memory when its not needed any more. If there are flow dependencies between the variables in the scan window, the register file can also be used for storing the values. Additionally, the register file is used to store auxiliary variables. If a large statement has to be broken down because it does not fit into the current size of the rDPA, the auxiliary variable can be stored in the register file and read back over the global I/O bus, to the place where the computation of the statement is finished. The register file makes it possible to use each DPU in the rDPA for operations by using the 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 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. Normally the address is only 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 units of the data driven rALU. The instruction set consists of instructions for loading data into the rDPA to a specific DPU from the MoMbus or the register file, for receiving data from a specific DPU, or branches on a special control signal from the generic address generators. The rDPA control supports context switches between three control programs which allows to use three independent virtual rALU subnets. The control program is loaded during configuration. The reconfigurable data-driven ALU allows also pipelined operation as shown in the example in section 6.

A status can be reported to the generic address generators (GAG) to inform on overflows or to force the GAGs to compute data dependent addresses. The input FIFO is currently only one word deep for each direction. This is sufficient because the data flow is regular to the main memory. Normally the rALU does not have to wait for data.

5. Programming environment

Statements which can be mapped to the rDPA array are arithmetic and logic expressions as well as conditions. loops are handled by the generic address generators. The input language for programming the rALU is the rALU programming language. The syntax of the statements follows the C programming language syntax (see also figure 3). In addition, the language provides the size of the used scan windows and the next handle position (relative to the actual position). The handle position is the lower left corner of the boundary of the scan window. By providing the handle position the rALU gets the necessary information for pipelining the complete statement block.

The rALU programming language file is parsed and a data structure like an abstract program tree is computed. Common subexpressions are taken into consideration. The operators of each statement are allocated with a straight forward algorithm. The number of DPUs used is optimized. Each allocated operator is then associated to a DPU in the rALU array. To recognize possible parallelization and to find data dependencies between the statements, a data dependency analysis is performed. 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. For best performance, an optimal sequence of these I/O operations has to be determined. Comparing an ‘as soon as possible’ (ASAP) with an ‘as late as possible’ (ALAP) schedule, the time critical path is detected. A priority function is developed from these schedules which gives the range of the I/O operations of the operands. With a list based scheduling method the optimal sequence of the I/O operations is found. 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 with the code of the operators. The configuration file for the rALU con-

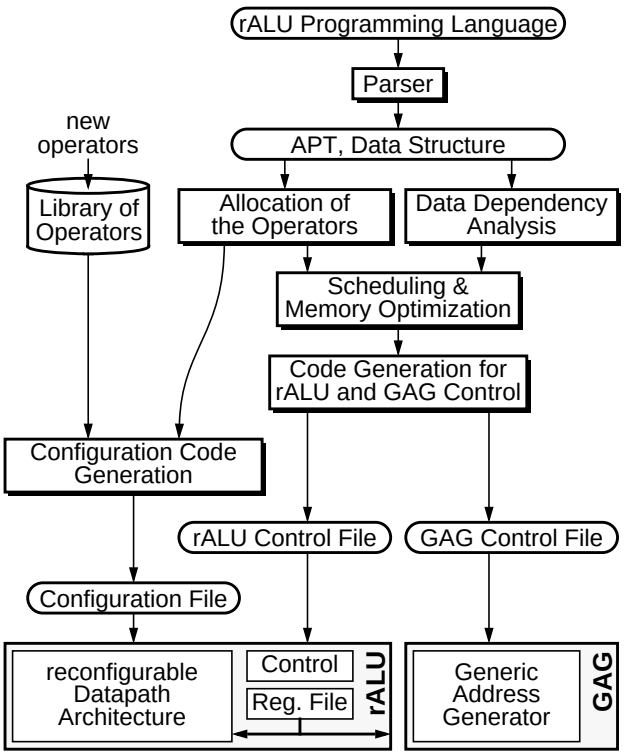

Figure 5. The rALU programming environment

trol unit and the configuration file for the GAGs is extracted from the final schedule of the I/O operators. For the FPGA implementation, the macros for the necessary operations are provided. The user has to put them together with a template of the communication in a schematic editor. The mapping is done by a vendor tool e. g. from Xilinx. Alternatively the environment provides a hardware description language file which has to be synthesized with a synthesis tool, e. g. Synopsys. The programming environment of the rALU is shown in figure 5.

6. Example: two dimensional FIR filter

The two dimensional FIR (finite impulse response) filter is one whose impulse response processes only a finite number of nonzero samples. The equation for a general two dimensional FIR filter is

$$y_{nm} = \sum_i \sum_j k_{ij} \cdot x_{n-i,\,m-j}. \qquad (1)$$

In this example a two dimensional filtering of second order is shown. Since the focus on this paper is only on the rALU for Xputers, the address generation for the data is not explained, please refer to [8]. Suppose the processed data is written into the same memory from which the data is read, one scan window is sufficient. It is of the size 3 by 3 as shown in figure 6c. All 9 scan window positions are for reading and the middle is also for writing. The window scans over the data map with a video scan, step width one. After each step it performs the equation:

$$\begin{aligned} w_{11new} = {} & w_{22}k_{22} + w_{12}k_{12} + w_{02}k_{02} \\ & + w_{21}k_{21} + w_{11}k_{11} + w_{01}k_{01} \\ & + w_{20}k_{20} + w_{10}k_{10} + w_{00}k_{00} \end{aligned} \qquad (2)$$

The k_{ij} are the coefficients of the filter and the w_{ij} contain the data at the corresponding scan window positions. Figure 6a shows equation (2) mapped into the rDPA. The input registers of the DPUs named with w_{ij} represent the scan window positions. The input registers named with k_{ij} represent the registers with coefficients of the FIR filter loaded at configuration.

At each step of the scan window three new data words are loaded, three old ones are removed and one output word is written back. Figure 6c shows the loading of the data. At the beginning, data word a_0 is loaded across the bus directly to w_{00}, b_0 to w_{01}, c_0 to w_{02}, a_1 to w_{10} and so on. As soon as data words arrive in the input registers of the DPUs the operation will be performed (data-driven). As shown in figure 6b the multiply-route operation evaluates the multiplication and routes the input data from west to the next DPU east. So the data required in the next step need not to be loaded again across the bus. The pipeline is filled by loading the first nine input data words over the bus, then the internal routing resources are used. Only the three new data words have to be routed by the bus. The execution is shown in figure 7. The first line shows the read and write operations using the global I/O bus of the

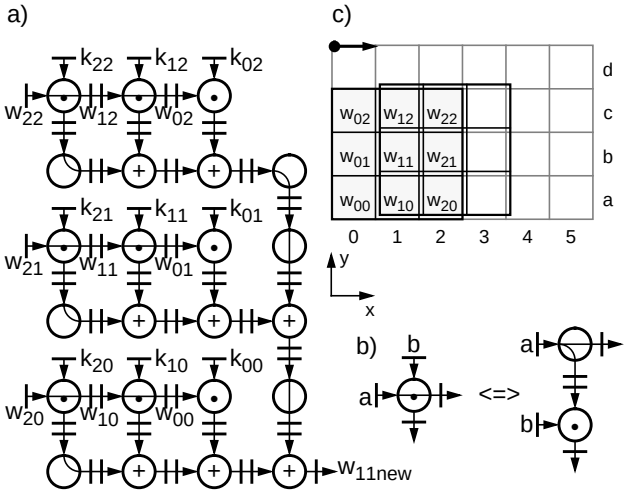

Figure 6. a) Two dimensional FIR filter mapped into the rDPA, b) equivalent operation of the multiply-route operation, c) the corresponding scan window

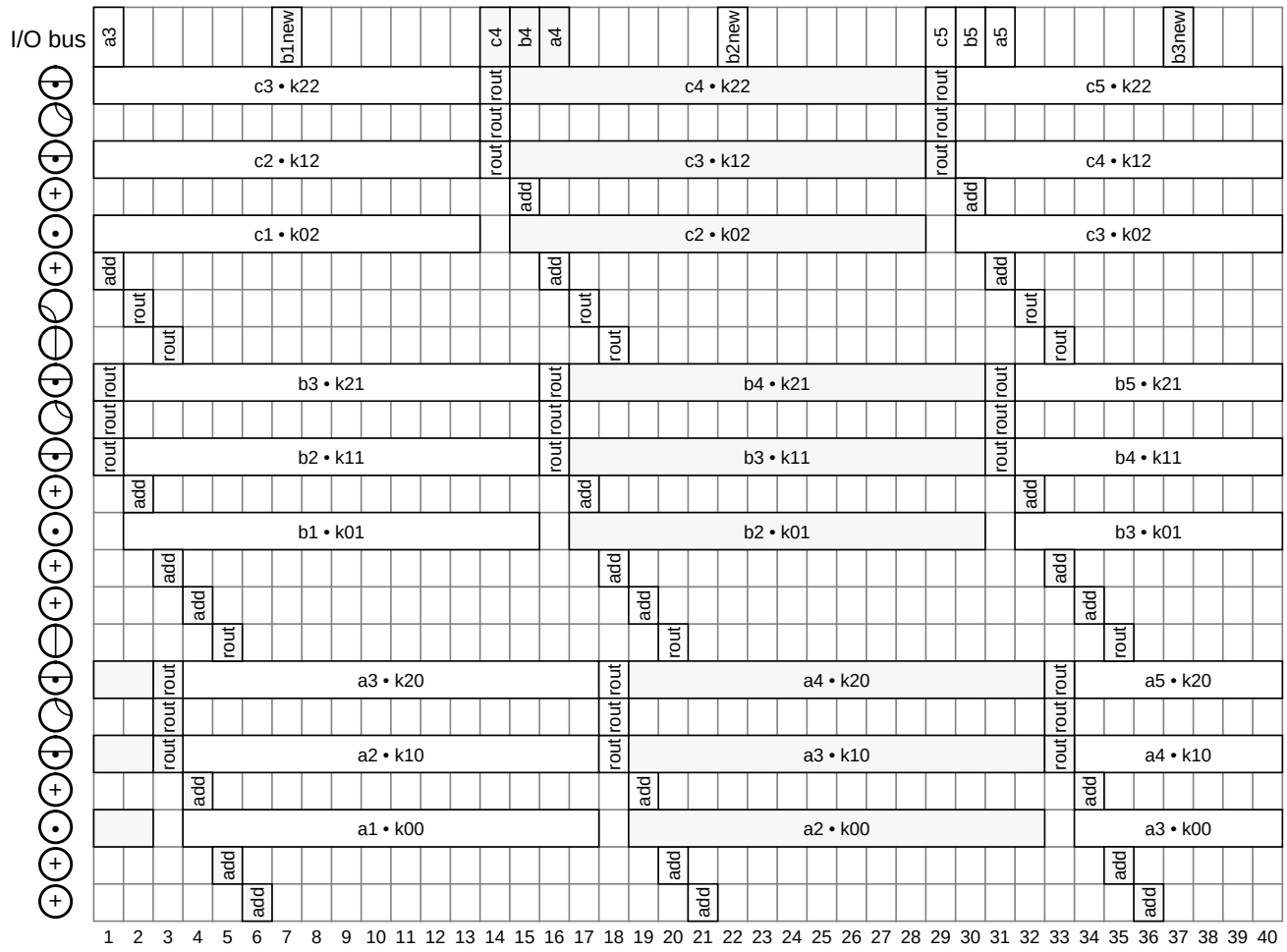

Figure 7. Pipelined execution of the two dimensional FIR filter

rDPA. The following lines describe the operation of the multipliers, routing units and adders. Each column belongs to one time step. All boxes in one column are evaluated in parallel.

The prototype implementation of the rDPA 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 many applications the coefficients are set up in such a way that shift operations are sufficient and multiplications are not necessary. If high throughput is needed, DPUs can be linked together to implement a pipelined multiplier. Benchmark results are given in table 1. The performance figures are a worst case estimation of our prototype. The speed of the examples 2 to 5 depend not on the order of the filter as long as the necessary hardware (# of DPUs) is provided. The same is applies for example 6.

#	Algorithms	Operations	# of DPUs	Throughput (time steps)	Performance
1	1024 Fast Fourier Transformation	*, +, -	10	$16 \cdot 10240$	50 Hz
2	FIR filter, n^{th} order	*, +	2(n+1)	15	0.6 MHz
3	FIR filter, n^{th} order	shift, +	2(n+1)	4	2 MHz
4	$n \times m$ two dim. FIR filter	*, +	2(n+1)(m+2)-1	15	0.6 MHz
5	$n \times m$ two dim. FIR filter	shift, +	2(n+1)(m+2)-1	4	2 MHz
6	Bubblesort, length n	scan-max	n-1	2	4.2 MHz

Table 1. Benchmark results

7. Conclusions

A programming model for an FPGA based hardware platform for a data-driven reconfigurable datapath architecture (rDPA) has been presented which strongly supports numerical applications. It may be used as a reconfigurable ALU (rALU) for custom computing machines (CCMs), Xputers (a class of CCMs) and other adaptable computer systems as well as for rapid prototyping of high speed datapaths. Together with the Xputer hardware and software environment a system has been implemented which is programmable in a high level language. 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 scalable rDPA provides parallel and pipelined evaluation of the compound operators. An implementation with FPGA chips and one with standard cells is shown. The FPGA implementation supports variable width of datapaths, also for bit level operations. The standard cell version of the implementation of the rDPA is microprogrammable. It provides a higher density of the operators. The architecture is in-circuit dynamically reconfigurable, which implies also partial reconfigurability at runtime.

A prototype chip with standard cells has been completely specified in the hardware description Verilog and will be submitted for fabrication to the Eurochip[1] organization soon. It has 32 bit datapaths and provides arithmetic resources for integer and fixed-point numbers. The FPGA version is being implemented by using Xilinx software tools. The programming environment is specified and is being implemented on Sun SPARCstations.

1. Eurochip is a microelectronics project of the CEC.

References

[1] J. M. Arnold: The Splash 2 Software Environment; IEEE Workshop on FPGAs for Custom Computing Machines, FCCM'93, IEEE Computer Society Press, Napa, CA, pp. 88-93, April 1993

[2] S. Casselman: Virtual Computing and The Virtual Computer; IEEE Workshop on FPGAs for Custom Computing Machines, FCCM'93, IEEE Computer Society Press, Napa, CA, pp. 43-48, April 1993

[3] N. N.: Giga Ops: Technical Overview of C to Hardware Compilation and Development System for FPGA Computing; Pre-Release Documentation, Rev. 0.7, Giga Operations Corporation, Berkeley, 1993

[4] M. Gokhale, R. Minnich: FPGA Computing in Data Parallel C; IEEE Workshop on FPGAs for Custom Computing Machines, FCCM'93, IEEE Computer Society Press, Napa, CA, pp. 94-101, April 1993

[5] 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

[6] 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

[7] 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

[8] R. W. Hartenstein, A. G. Hirschbiel, K. Schmidt, M. Weber: A Novel Paradigm of Parallel Computation and its Use to Implement Simple High-Performance Hardware; Future Generation Systems 7 (1991/92), p. 181-198, Elsevier Science Publishers, North-Holland, 1992

[9] P. Lysaght, J. Dunlop: Dynamic Reconfiguration of Field-programmable Gate Arrays; Proceedings of the 3rd International Workshop on Field Programmable Logic and Applications, Oxford, Sept. 1993

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

[11] N. N.: The XC4000 Data Book; Xilinx, Inc., 1992