A Dynamically Reconfigurable System for Space-Efficient Computation of the FFT

C. H. Llanos[2,4] , R. P. Jacobi[3,4],M. Ayala-Rincón[1,4], and R. W. Hartenstein[5,6]

[1] Departamentos de Matemática, [2] Engenharia Mecânica e [3] Ciência da Computação,
[4] Universidade de Brasília ayala@mat.unb.br llanos@unb.br rjacobi@cic.unb.br
[5] Fachbereich Informatik, [6] Kaiserslautern University of Technology hartenst@rhhk.uni-kl.de

Abstract

The current explosion of mobile computing, embedded applications, radio defined software and several technologies based on wireless communication has pushed the interest on efficient reconfigurable architectures to deal with communication issues in hardware constrained platforms. Among then, the Fast Fourier Transform is widely used due to the variety of applications it can be applied. In this paper we address an space efficient architecture, which was initially modeled by rewriting-logic, that uses dynamic reconfiguration to reduce the size of the circuit needed to compute the FFT. The architecture implements the algorithm using a single reconfigurable column. An VHDL implementation of the architecture is described the simulation and synthesis results for a FPGA platform are presented.

Key Words: dynamically reconfigurable systolic arrays, fast Fourier transform, morphware, configware

1. INTRODUCTION

The fast evolution of high-bandwidth mobile communication systems and the increasing capacity of integrated circuits fostered the investigation of new design methodologies to provide cost effective solutions that match the processing and transmission requirements of future communication systems. Systems on chip (SoC) and systems on a programmable chip (SoPC) emerge as technological alternatives to integrate a variety of algorithms needed for data processing and communication.

Silicon systems typically include processing components, memories, application specific circuits and, in some cases, reconfigurable modules. Issues as component reuse and adaptation start to play an important role as the design methodologies rely on the integration of *intellectual property* (IP) modules. IP modules can be provided in several flavors, such as a layout description, an PLD bit stream or a synthesizable HDL model. Among those modules, the *Fast Fourier Transform* (FFT) has a large application in several algorithms for signal processing [11]. It may be implemented either as a software module, executed by a digital signal processor (DSP) or by a dedicated hardware device, which may be a hard-IP block or a reconfigurable circuit. The specific solution depends on a variety of factors. Typical mobile communication systems must adapt themselves to changing communication parameters, like bandwidth, protocols, and so on. Moreover, the

M. Arias, R. Cumplido, C. Feregrino (Eds.): International Conference on Reconfigurable Computing and FPGAs ReConFig'04, Colima, México, September 20-21, 2004.

low power requirements [10] for battery powered devices is another key factor for mobile computing.

In this paper we propose a space efficient FFT architecture based on the use of a single reconfigurable vector of processing elements. The classical hardware design the FFT is a matrix of processing elements where the input data is provided to the first column and then traverse the matrix column by column, following a butterfly connection scheme between column elements. In our approach, a single reconfigurable vector is implemented. In the first step, it executes the function of the first column over the input data. The data produced by the first iteration is then fed back to the inputs while the vector is reconfigured to emulate the second column. The process is iterated until the FFT is done. This approach provides a solution that is space and power efficient, scarifying speed.

This architecture was modeled using the rewriting-logic based ELAN environment as presented in [12]. This modeling methodology allows a high level specification and verification through simulation of the architecture. Recently, by applying this rewriting-logic methodology we have also conceived other reconfigurable architectures proved adequate for efficient implementation of general dynamic programming algorithm for sequence comparison [13].

Section 2 review some basic concepts about systolic and reconfigurable architectures. Section 3 describes the FFT algorithm and section 4 presents our implementation and results. Section 5 is the conclusion.

2. SYSTOLIC ARRAYS AND RECONFIGURABLE SYSTEMS

The term "systolic array" has been coined probably by H. T. Kung around 1979 [18]. A systolic array is a mesh-connected pipe network of DPUs (datapath units), using only nearest neighbor (NN) interconnect. DPU functional units usually operate synchronously, processing streams of data that traverse the network (also asynchronous mode of operation is possible, where sometimes also the term "wavefront array" is used instead of "systolic array"). Systolic arrays provide a large amount of parallelism and are well adapted to a restrict set of computational problems: those which present strictly regular data dependencies. Linear projection has been the reason, why systolic arrays allowed only uniform linear pipes (uniform means: only a single type of DPU allowed): systolic arrays can be used only for applications with strictly regular data dependencies. The restriction of systolic arrays to applications with strictly regular data dependencies was due to the synthesis algorithms used by the systolic array scene during the eighties. Because of this lack of flexibility reconfigurability did not made very much sense. From this point of view, the first version DPSS (datapath synthesis system) of the KressArray used simulated annealing instead of algebraic methods, which allows any wild form pipes like zig-zag, spiral, fork, join, and any completely irregular shape of pipes, and allows non-uniform functionality of DPUs [14,15]. This array has become general purpose, so that reconfigurability makes sense. It allows non-uniform functionality of DPUs. Only by discarding linear projection or algebraic synthesis methods, reconfigurability makes sense. The restriction to applications with only regular data dependencies has disappeared. Although the first platform example by R. Kress did not reveal this: now any non-linear wild patterned pipes and non-uniform DPU architectures are possible. This has been implemented later along with a new version of the KressArray by U. Nageldinger, which is called the KressArray Xplorer [16 ,17]. This also features programmable interconnect: both,

multiple nearest neighbour interconnect, and auxiliar second level interconnect going beyond nearest neighbours.
The generalization provided by reconfigurable arrays require new synthesis methods to efficiently map applications to non-uniform DPUs interconnected through non-linear pipes
The variety of implementations that arise combining systolic architectures and reconfigurable computing requires adequate tools for modeling and simulating design decisions, providing a framework for design space exploration.

A systolic array is a mesh-connected pipe network of DPUs (datapath units), using only nearest neighbour (NN) interconnect [7, 8, 5]. DPU functional units operate synchronously, processing streams of data that traverse the network. Systolic arrays provide a large amount of parallelism and are well adapted to a restrict set of computational problems, i.e., those which can be efficiently mapped to a regular network of operators. Figure 1 shows a simple systolic example of a matrix-vector multiplication. The vector elements are stored in the cells and are multiplied by the matrix elements that are shifted bottom-up. On the first cycle, the first cell (DPU1) computes x_1*a_{11}, while the second and third cells (DPU2 and DPU3) multiply their values by 0. On the second cycle, the first cell computes x_1*a_{21}, while the second cell computes $x_1*a_{11} + x_2*a_{12}$, where the first term is taken from the first cell and added to the product produced in second cell. In the third cycle, the third cell produces the first result: $y_1 = x_1*a_{11} + x_2*a_{12} + x_3*a_{13}$. In the following two cycles y_2 and y_3 will be output by the third cell. Thus, by the end of the third cycle the first result is produced and the remaining values are produced in the following cycles.

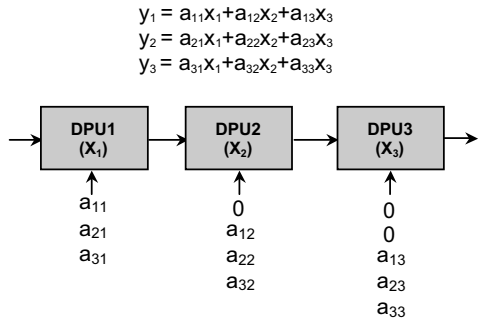

Figure 1: Vector – matrix computation

Systolic arrays provide a large degree of parallelism but are restricted to a set of applications that efficiently maps to this architecture. The FFT is an algorithm that may be efficiently mapped to a systolic array. A pipeline implementation of the FFT would allow one computation by clock cycle. However, this approach takes a lot of space and consume more power. To allow a single vector to compute the complete FFT we must modify its architecture in order to allow the vector function change according to the algorithm's steps.

This can be achieved through the use of reconfigurable circuits [3,6]. The most popular ones are the *Field Programmable Gate Arrays* (FPGAs). These circuits provide a high degree of flexibility, allowing the configuration of the design at gate level. However, this flexibility has a drawback: the efficiency of the use of the logic is reduced, since a lot of area is wasted with connections [5].

Devices that allow a fine grain reconfiguration usually suffer from this problem. On the other side, coarse grain reconfigurable architectures [9] allow reconfiguration at word level, reducing the flexibility but providing more efficient implementations.

The FFT design presented here could be implemented either with fine grain or with coarse grain architectures. In the case of coarse grain circuits, the reconfigurable module in the SoPC would be specialized and dedicated to a subset of signal processing applications, those that could be implemented efficiently implemented by a vector of processing elements that work with the same data type (fixed point numbers) and uses a restricted set of operations, like sum, subtraction and multiplication.

3. THE FAST FOURIER TRANSFORM IN A SISTOLIC ARRAY

The FFT is an implementation of the Discrete Fourier Transform - DFT, which is widely used in signal processing. Given an n-array of complex numbers $a = (a_0, ..., a_{n-1})$, its DFT, $F_n \times a$, is the n-array $(b_0, ..., b_{n-1})$, where

$$b_j = \sum_{k=0}^{n-1} a_k \cdot \omega_n^{kj} \quad for \quad j = 0,1,...,n-1$$

and $\omega_n = e^{i\frac{2\pi}{n}}$ is a primitive n^{th} complex root of the unity. The basic operations are multiply-accumulate, executed over complex numbers. The DFT has a time complexity of $O(n^2)$, which is too excessive for large sequences. The FFT is an $O(n \ ln \ n)$ run time implementation of DFT based on a recursive algorithm proposed by Cooley-Tukey. This algorithm can be implemented in dataflow hardware as it is shown in the Figure 2 [4,1].

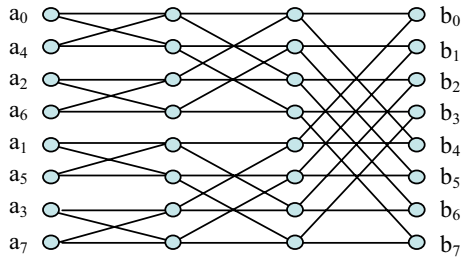

Figure 2. FFT circuit for n = 8

The number of data points is a power of 2. The network of nodes is a butterfly circuit. Each node implements a complex number multiplies-accumulate operation on its inputs:

$$b_j = u_j + z \cdot v_j$$

Details of this implementation can be found in classical text books, such as in [4,1].

The systolic array for FFT implementations is composed of MACs. The reconfigurable architecture of a MAC is showed in figure 3. It was implemented in such a way that it could be reused to implement other array processing applications like matrix multiplication.

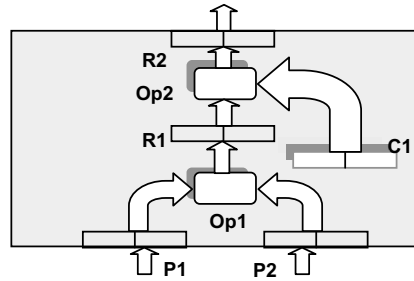

Figure 3: node architecture for FFT

We distinguish between reconfigurable (shadowed) and fixed components. The formers are the two operational units, Op1 and Op2; and the constant, C1. The latter are the two ports, P1 and P2; and the two registers, R1 and R2. In any reconfiguration the constant part of each `MAC` is set with arbitrary complex numbers. For implementing FFT, we will set these constants with the adequate complex roots of the unity.

The registers, ports and the constant store complex numbers and consist of two components: the real and the imaginary part. The two operators can be reconfigured to be any operation over complex numbers. In particular, for implementing FFT we will use only addition (+), subtraction (-) and multiplication (×) of complex numbers.

A classical implementation of the FFT for eight inputs (as it's showed in figure 2) it would require four MAC arrays. Otherwise, it's possible to implement the algorithm in a optimal space approach by using only one array as it's showed in figure 4.

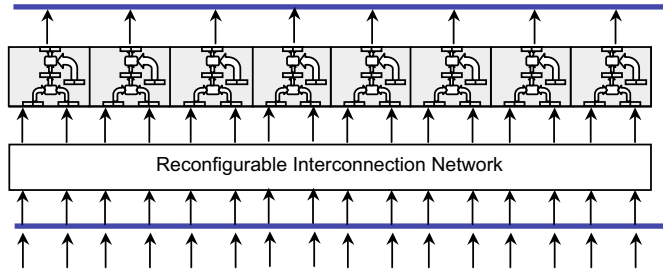

Figure 4. Reconfigurable 8-array FFT

This architecture was modeled using *term rewriting systems* (TRS) and presented in [12]. The TRS modeling allowed a high level specification and verification through simulation of the architecture. In this case the butterfly operation is implemented as a reconfiguration step of the reconfigurable interconnection network.

4. VHDL DESIGN OF THE RECONFIGURABLE FFT ARCHITECTURE

The architecture of the reconfigurable FFT was designed in VHDL and synthesized with the Altera design tools. The architecture consists of the FFT array that includes the reconfigurable interconnection network, a ROM for storing the data for reconfiguration and the control unit. The circuit architecture is showed in figure 5.

4.1 The Functional Units implementation

Since the operands in the FFT algorithm are complex, the registers are duplicated for storing real and imaginary parts of the complex.

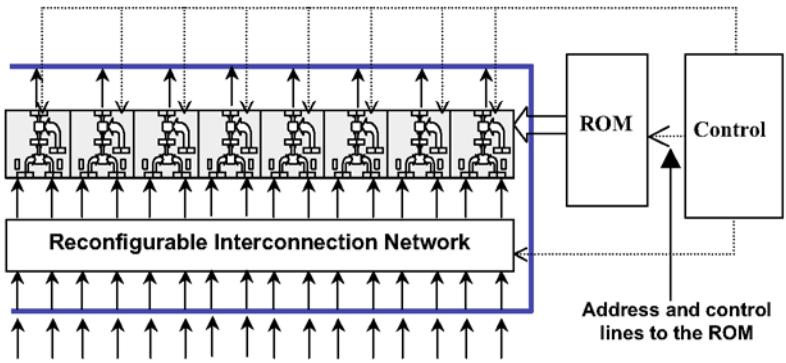

Figure 5. The architecture of the FFT circuit

In the same way the functional units need to deal with complex operands. The operation in the functional units are selected by two bits as is showed in figure 6a.

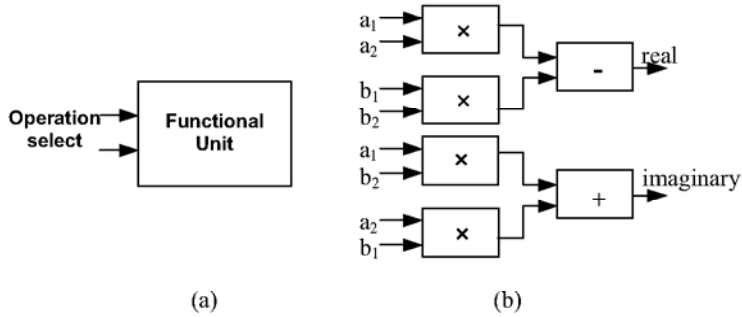

Figure 6. (a) The functional unit block. (b) The complex multiplier

The multiplication of two complex numbers p: $(a_1 + ib_1)$ and q: $(a_2 + ib_2)$ is defined as: $p \times q = (a_1 \times a_2 - b_1 \times b_2) + i(a_1 \times b_2 + a_2 \times b_1)$. The modules that perform the complex multiplication are described in figure 6b. Each complex multiplier consists of four-carry save array multipliers and two ripple carry adders.

4.2 The Reconfigurable Interconnection Network (The butterfly operation)

The butterfly is a basic operation in the FFT computation which produces a pair of complex values in a stage m from a pair or values from the preceding stage $m - 1$ following the equations:

$X_m[n] = X_{m-1}[n] + X_{m-1}[n + r]$ and $X_m[n + r] = X_{m-1}[n] - X_{m-1}[n + r]$

where $r = N/2^m$ and $m = 1,..., \log_2 N$.

For the 8-array FFT the sequence of the butterfly operation for each step can be described as shown in table 1.

Table 1. Mapping the interconnections on the FFT steeps

	Input Port	St_0	St_1	St_2	St_3
MAC0	P1	Extern	MAC0	MAC0	MAC0
	P2	Extern	MAC1	MAC2	MAC4
MAC1	P1	Extern	MAC0	MAC1	MAC1
	P2	Extern	MAC1	MAC3	MAC5
MAC2	P1	Extern	MAC2	MAC0	MAC2
	P2	Extern	MAC3	MAC2	MAC6
MAC3	P1	Extern	MAC2	MAC1	MAC3
	P2	Extern	MAC3	MAC3	MAC7
MAC4	P1	Extern	MAC4	MAC4	MAC0
	P2	Extern	MAC5	MAC6	MAC4
MAC5	P1	Extern	MAC4	MAC5	MAC1
	P2	Extern	MAC5	MAC7	MAC5
MAC6	P1	Extern	MAC6	MAC4	MAC2
	P2	Extern	MAC7	MAC6	MAC6
MAC7	P1	Extern	MAC6	MAC5	MAC3
	P2	Extern	MAC7	MAC7	MAC7

For example, in the step 0 the operands for the two ports (for each MAC) are the coefficients of the input polynomial (see section 4) just as described in column 1 of the table1 (external inputs). In the following step the butterfly operation defines the new sources of the operands for each MAC. For example, in the step 0 for the MAC0 receives the inputs from MAC0 (in the port P1) and MAC1 (in the port P2). Thus the butterfly operation represents a reconfiguration step of the interconnections in the 8-array.

The reconfigurable interconnection network was implemented using 4×1 multiplexers. Figure 7 shows the multiplexer interconnection for routing the data to the input ports in a MAC.

To simplify the reconfiguration task, the butterfly operation was implemented using sixteen (16) 4×1 multiplexers for routing the complex input data, one for each port (as shown for one MAC in figure 7). This way optimizes the butterfly operation performance by switching all the interconnections simultaneously among all the eight MACs. The two bits s_1 and s_2 to control the set of sixteen multiplexers are generated by the control unit of the system.

4.3 The reconfiguration of the operational Units and the constant register

The data for reconfiguring the functional units and the constant register of each MAC are stored in a ROM memory. The ROM was implemented with four parameterized words as it's showed in figure 8. Each word stores the coefficient values for each register Ci (eight complex numbers). Besides this information the memory store a pair of two bits to configure the functional units of each MAC. The word length is parameterized and it allows to adapt the design to any coefficient size.

4.4 The control Unit

The operation of the modules integrating the whole of the system is scheduled by a 12-state synchronous FSM. In order to execute each step of the FFT algorithm

the FSM execute the propagate, reconfigure and execute state. The *propagate-state* deals with the control lines of the multiplexer and controls de enable lines for the inputs/outputs registers.

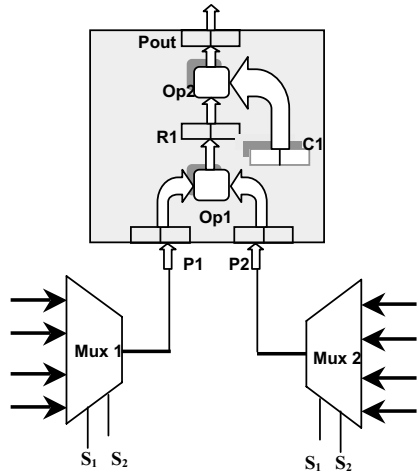

Figure 7. Implementation of the interconnection network

Mac0 coefficient, steep 0	Unit 1 operation MAC0	Unit 2 operation MAC0	Mac1 coefficient, steep 0	.....
Mac0 coefficient, steep 3	Unit 1 operation MAC0	Unit 2 operation MAC0	Mac1 coefficient, steep 3	.....

Figure 8. Data structure of the ROM

The *reconfigure-state* controls the ROM memory address lines in order to configure the functional units and all of the constant registers. The *execute-state* controls the port out enable lines to obtain the result for each steep.

5. Area and running time results

The FFT circuit was implemented as a hierarchic description. The MAC array and the interconnection network implement the FFT-processor module. The top of the design consists of the FFT-processor, the ROM and the control unit. Each module of the architecture was implemented and tested for validation. The modules were coded using parameterized RTL-level VHDL allowing the reusability and easy configuration. The complete description was compiled and simulated in the Quartus II system from Altera Corporation [2]. The area and timing results for a 8 bits length words coefficients using an APEX EP20K400 as target device are showed in table 2.

The computation of one step of the FFT takes 2 cycles, one cycle to load data into the input registers and compute de complex result and a second cycle to store the

result in the MAC output registers, to feedback the results through the reconfigurable network and to reconfigure the MAC operations. The design presented here receives as inputs 8 coefficients. The complete FFT is performed in 3 steps. Thus, we have that the global time to produce the outputs is 29,64 / 6 = 4,94 millions of FFT by second.

Table 2. Synthesis Results		
Feature	Reconfigurable FFT	Pipelined FFT
Family	APEX	APEX
Device	EP20K400FC672-2XV	EP20K400FC672-2XV
Operating frequency	29,64 MHz	35,11 Mhz
Logic elements number	1340 (8%)	10820 (65%)
Delay	33.7 ns	28,48 ns
Throughput	4,94 M FFT/s	35,11 M FFT/s

For comparison purposes we implemented a pipeline version of the FFT, which is built using the same structure of the reconfigurable FFT but without feedback. It is able to compute one FFT by cycle after the pipe is filled. The synthesis results are shown in the last column of table 2. As expected, the pipelined version produces faster results, around 7 times the throughput of the reconfigurable FFT but consuming around 8 times more hardware resources.

6. CONCLUSIONS

This paper described a space efficient FFT implementation by using reconfigurable systolic array approach. The architecture proposed here was developed taking into account its possible reuse for different applications, like matrix multiplication, convolution, and signal processing algorithms in general. All of the modules of the circuit were parameterized for easy reuse and configuration.

One of the advantages of this approach is that this reconfigurable module can fit into systems on chip with area and reconfigurable resource restrictions and still produce high throughput. A course grain implementation of this architecture could be suitable for a variety of signal processing problems with a much higher throughput than that obtained with standard FPGAs or by software.

Future work will address new reconfiguration schemes, since the reconfiguration time in this case is constrained by the ROM access time. Alternatives like a pipeline between reconfiguration and operation using two vectors, as suggested in [*12*], will be studied.

One of the referees reports the existence of implementations of FFT using only $O(log_2 n)$ *Units*. Instead of having n units all computing a level of the FFT in parallel, we have one *Unit* per level. Specifically, *Unit_1* does all processing of the first level of butterfly, feeding results to *Unit_2*. *Unit_2* does the butterfly, and the processing, and feeds on to level 3. Since the butterfly at each level is regular, the data movement becomes trivial to implement the butterfly (i.e. a simple shift register will do it). Notice that the proposed *Unit* should execute all processing that our array of n MACs does at each level, which implies that the space used for *Unit* is $O(n)$. Then the used space of this solution coincides with the classical one: $O(nlog_2 n)$. From this point of view, a sole more elaborated MAC, which executes all operations in parallel could implement FFT in $O(1)$ space. But notice that this MAC is itself of space complexity $O(n)$, which does not improve our proposed solution of FFT.

7. REFERENCES

[1] S. G. Akl. *Parallel Computation: Models and Methods*. Prentice-Hall, 1997.

[2] Altera® Corporation. Quartus II User Guide. Available at http://www.altera.com. Accessed in 2004.

[3] J. Becker and R. W. Hartenstein, *Configware and morphware going mainstream*, Journal of Systems Architecture 49:127-142, 2003.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*, The MIT Press, 2001.

[5] A. DeHon: Reconfigurable Architectures for General Purpose Computing; report no. AITR 1586, MIT AI Lab, 1996.

[6] R. Hartenstein, R. Kress and H. Reinig. *A Scalable, Parallel and Reconfigurable Datapath Architecture.* 6th Int. Symposium on IC Technology, Systems and Applications - ISIC'95, Singapore, 1995. Available at www.kressarray.de.

[7] H.T. Kung, C. E. Leiserson, Systolic Arrays for VLSI; Sparse Matrix Proc. 1978, Society for Industrial and Applied Mathematics,: 256-282,.

[8] S. Y. Kung. VLSI Array Processors. Prentice-Hall, 1987.

[9] U. Nageldinger. *Coarse-Grained Reconfigurable Architecture Design Space Exploration.* These, Universität Kaiserslautern, 2001.

[10] A. Abnous, K. Seno, Y. Ichikawa, M. Wan, J. M. Rabaey. *Evaluation of a Low-Power Reconfigurable DSP Architecture*. IPPS/SPDP Workshops 1998: 55-60.

[11] M. Vergara, M. Strum, W. Eberle and B. Gyselinckx. A 195KFFT/s (256-points) high performance FFT/IFFT processor for OFDM applications. In Proc. of SBT/IEEE Int'l telecommunications Symposium (SP - Brasil), 1998.

[12] M. Ayala-Rincón, R. B Nogueira, C.H. Llanos, R.P. Jacobi, and R.W Hartenstein. Modeling a Reconfigurable System for Computing the FFT in Place via Rewriting-Logic. In IEEE CS Proc. of SBCCI03, 205-209, 2003.

[13] M. Ayala-Rincón, R.P. Jacobi, L.G.A. Carvalho, C.H. Llanos, and R.W Hartenstein. Modeling via Rewriting-Logic and Prototyping Reconfigurable Systems for Efficient Computation of Dynamic Programming Methods by Rewriting-Logic. In ACM Proc. of SBCCI04, 2004.

[14] R. Hartenstein, R. Kress and H. Reinig. *A Scalable, Parallel and Reconfigurable Datapath Architecture*. 6th Int. Symposium on IC Technology, Systems and Applications - ISIC'95, Singapore, 1995.

[15] R. Hartenstein, M. Herz, T. Hoffmann, U. Nageldinger. *Kress Array Explorer: A New CAD Environment to Optimize Reconfigurable Datapath Array Architectures*. 5th Asia and South Pacific Design Automation Conference - ASP-DAC 2000, Yodohama, Japan, 2000.

[16] R. Hartenstein, M. Herz, T. Hoffmann and Ulrich Nageldinger. *Mapping Applications onto Reconfigurable Kress Arrays*. 9th International Workshop on Field Programmable Logic and Applications - FPL 1999: Springer Lecture Notes in Computer Science, Vol. 1673, pages 385-390, 1999.

[17] U. Nageldinger. *Coarse-grained Reconfigurable Architecture Design Space Exploration*, PhD Thesis, TU Kaiserslautern, 2001.

[18] S. Y. Kung. VLSI Array Processors. Prentice-Hall, 1987.