Design and Test Space Exploration of Transport-Triggered Architectures

V. A. Zivkovic, R. J. W. T. Tangelder, H. G. Kerkhoff
MESA+ Research Institute, University of Twente,
P.O. Box 217, 7500 AE Enschede, the Netherlands

Abstract

This paper describes a new approach in the high level design and test of transport-triggered architectures (TTA), a special type of application specific instruction processors (ASIP). The proposed method introduces the test as an additional constraint, besides throughput and circuit area. The method, that calculates the testability of the system, helps the designer to assess the obtained architectures with respect to test, area and throughput in the early phase of the design and selects the most suitable one. In order to create the templated TTA, the "MOVE" framework has been addressed. The approach is validated with respect to the "Crypt" Unix application.

1. Introduction

Transport triggered architectures (TTA) [1-3] have given emerged importance in high-performance ASIC applications. They support a high degree of instruction-level parallelism [4] and offer a possibility to parameterize the architecture, thus leaving the designer more degrees of freedom in the design. TTAs are basically derived from VLIW (Very Long Instruction Word) Processor architectures. The difference between the two architectures is that the TTA have the implicit data control i.e. they are programmed by specifying the data transports instead of the operations. This will result in a more efficient use of hardware resources and in a less complex hardware structure. There are also additional advantages of using the TTA such as, e.g., cycle minimization, implementation flexibility, performance scalability, etc. They are all explained in more details in [1]. Hence, these architectures have serious potential in important applications in the future. Addressing the test in the early design phase of these architectures will definitely lower their cost, making them more robust and easier to test. Our proposal is to introduce the test cost as an additional constraint during the design of the TTA templates.

2. TTA design within the "MOVE" Codesign Environment

In order to generate the TTA template, the MOVE hardware/software codesign system has been used [5]. It accepts C/C++ applications as input and produces parallel code that is supported by an instruction level parallel-type TTA.

A typical TTA template is shown in figure 1, where FU denotes functional units such as adders, comparators, multipliers, etc., while RF denotes register files. The two other parameters of the datapath are busses and sockets.

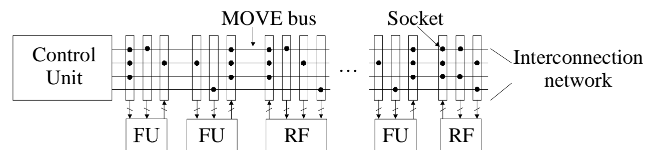

Figure 1. The TTA structure

The sockets perform the control of the operations and the control unit is actually distributed over the sockets. There is only one type of operation, being the "*move"* operation through the network, from one FU (or RF) to another one. It is possible to perform as many move operations in parallel as busses are contained within the TTA. The FU's are triggered with the arrival of the data at their inputs, performing their function(s) subsequently.

The exact match of the number and type of functional units, register files, sockets and busses is the subject of *design space exploration* provided within the MOVE environment. The main design evaluation criteria within MOVE have been circuit area and performance, so far. The exploration is performed with iterative generation of different architectures. Hence, for particular architectures, the area and throughput costs are generated. The solution space is bounded by local optimal solutions, so called *Pareto points* [6]. For example, figure 2 depicts the result of the design space exploration for a "Crypt" application [7].

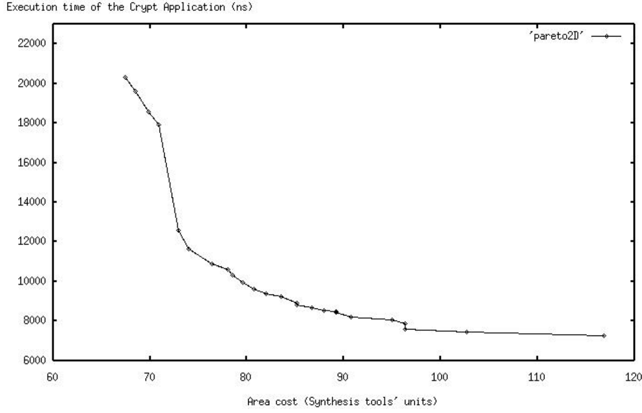

Figure 2. The solution space limited with Pareto points for Crypt application

Hence, the designer may choose the solution that satisfies his initial specifications in terms of the area and the performance. However, the existing high-level design approach, so far, does not offer the possibility to assess the overall test cost of the resulting TTA before the actual hardware implementation. For example, even though the optimal solution in terms of area/timing has been found, its cost after the test synthesis could increase. This can be reflected with a large number of test patterns necessary to test the resulting structure. Furthermore, the introduction of DfT circuitry or BIST may lead to the degradation of the already achieved performances or complicate the design and increase its design time. Therefore, it would be more convenient if the designer of the TTA has a notion of test implications during the early phase of the design. Our proposal is to introduce an additional test constraint during the design space exploration phase in order to decrease the number of the test patterns while preserving completely the already achieved area-throughput ratio. The next section clarifies the method.

3. Test costs in the design space exploration of the TTA architectures

The concept of the test cost as a merit of the testability of the system has also been addressed in some recent papers in the area of the high-level test synthesis [8-13]. However, none of them was acceptable in the TTA environment. For example, the testability costs described in [8] and [9] are appropriate for random logic, not for the regular structure that characterizes the TTA. [10] introduces the test cost during the tradeoff in hardware/software codesign partitioning. However, the hardware test patterns were developed according to the high-level hardware description, not its actual implementation. The approach given in [11] copes with the test cost introduced for the sake of the testable ALU design and it might be useful as a test cost of a particular FU within TTA. The test costs stated in [12] have been given for VLIW applications and might be considered useful for the derivation of the approach stated in this article. Reference [13] copes with the implementation of BIST in the datapath. However, the idea behind our approach is not to use any additional circuitry for the test, except flip-flops (functional) with scan. As a boundary condition, the obtained architectures that are on the Pareto curve with respect to area and performance should not be deteriorated. Hence, given the solution space, an appropriate test cost has to be introduced, different from any above mentioned. Our overall test cost is a function of the architectural parameters only, i.e., it depends on the match of the functional units, registers, busses and sockets.

Pareto points limit the design space such that $\forall\ (a, t) \in \vartheta^2(a, t)$, $(a \geq a_p \vee t \geq t_p)$ where $\vartheta^2(a, t)$ is the two-dimensional area-throughput solution space and (a_p, t_p) is a Pareto point. Therefore, only the architectures that correspond to the Pareto points in the design space are evaluated in terms of testing. The regularity of the architecture, i.e., the fact that each FU and RF (in the further text also denoted as components) from datapath is accessible via a MOVE bus and the sockets, is also used for the calculation of the test cost. The design space exploration in the MOVE environment is performed on a limited set of components. Hence, each available component from the datapath has its unique test cost function $f_t(\underline{p})$ expressed as:

$$\underline{p} \rightarrow f_t(\underline{p}),\ \underline{p} = p(c, n_p, n_b, n_{conn}), \qquad (1)$$

where c denotes the type of the component, n_p the number of the test patterns targeting the stuck-at faults of the components, n_b is the total number of busses within the architecture, n_{conn} is the number of the component's ports (connectors). The components are already predesigned up to the gate-level using the Synopsys synthesis package. Hence, the numbers of the test patterns for each functional unit (and register file) is back-annotated with an automatic test pattern generation (ATPG) tool. Not only the test patterns, but also the information regarding the actual area and delay of each component are used during the design space exploration phase resulting in Pareto point curve.

Before the analytical expressions for the test cost of the components are given, the data transport mechanism through the components and sockets has to be explained.

3.1 Data transport scheduling within the TTA

During the execution of an operation, the scheduled, pipelined data transport from a MOVE bus through the sockets and components to another (or the same) MOVE bus takes place (see figure 1). The function units and

registers files are implemented using the so-called *hybrid-pipelining* mechanism, whose structure is shown in figure 3 [1]. The details of the input socket implementation, is depicted in figure 4 (the architecture of the output socket is very similar). The control signals refer to the instruction decoding and matching, i.e., selecting of the appropriate component for the execution of the operation.

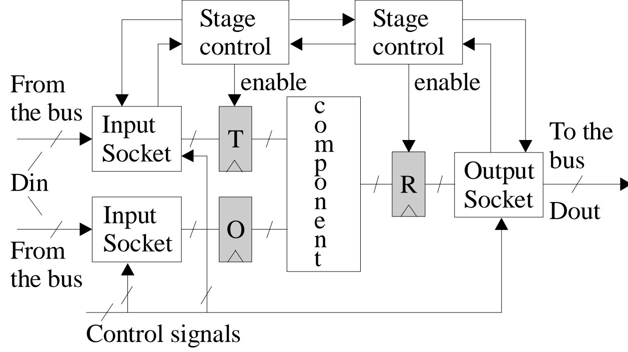

Figure 3. The control structure of the pipelined component

Figure 3 shows the arbitrary two-input, one-output component and its connections with the other architectural parameters. Of course, it is always possible to have the components with arbitrary number of input and output ports, followed with the appropriate registers and sockets. However, there can be exactly one and only one trigger register (*T*) that actually triggers the operation of the component after the data arrives at its input. The *O* and *R* are the operand and the result registers, respectively. The other flip-flops influencing the data transport mechanism are F_{in} in the input socket (see figure 4) and F_{out} in the output socket (not shown in figures).

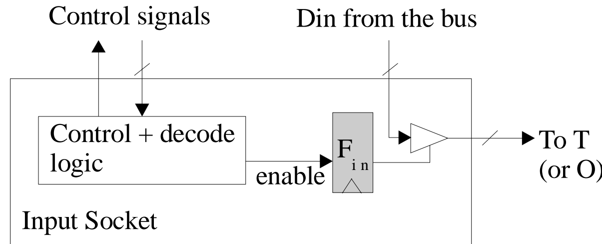

Figure 4. The implementation of the input socket

If $C_i(r)$ denotes the cycle when the actual data transport of the *i-th* operation to the register r, $r \in \{O, T, R, F_{in}, F_{out}\}$ takes place, the timing-transport relations among the registers are expressed as:

$$C_i(T) - C_i(O) \geq 0, \quad (2)$$
$$C_i(R) - C_i(T) \geq 1, \quad (3)$$
$$C_i(T) > C_j(T) \Leftrightarrow C_i(R) > C_j(R) \quad (4)$$
$$C_i(T) > C_j(T) \Leftrightarrow C_i(O) > C_j(T) \quad (5)$$
$$C_i(O) - C_i(F_{in}) \geq 1 \quad (6)$$
$$C_i(T) - C_i(F_{in}) \geq 1 \quad (7)$$
$$C_i(F_{out}) - C_i(R) \geq 1 \quad (8)$$

The data can not appear in the trigger register before the data is ready in the operand register (inequality 2). In addition, the data processing inside the component itself requires at least one cycle (3). Also, the instructions in the same functional unit are executed sequentially and the operand value must not be overwritten unless it is previously used, as the relations (4) and (5) reveal. The stage control block from figure 3, implemented as a finite state machine, ensures these conditions are fulfilled. Finally, the instruction decoding takes also at least one cycle, according to the inequalities (6-8).

In addition, let $CD_c(t_i, t_j)$ denotes the difference in clock cycle between the transports in t_i*-th* and t_j*-th* timing slots for *c-th* component. In that case, the minimum number of clock cycles that are necessary for the completion of the *i-th* operation of the component from figure 3 is equal to 3, referring to the (2-8):

$$CD_c(t_{Din}, t_{Dout}) = (C_i(T) - C_i(F_{in})) + (C_i(R) - C_i(T)) + (C_i(F_{out}) - C_i(R)) \geq 3. \quad (9)$$

t_{Din} and t_{Dout} denote the cycles in which the test data is applied to or read from MOVE bus, respectively. It is assumed that the data arrives at the same time in the operand and trigger register, according to relation (1). Hence, the *n-th* operation of the same component may be completed in the timing slot $n+3$,expressed with (3) and (4). However, the execution time increases if the operand and trigger registers are connected to the same bus since the data can not appear at the same time in the operand and the trigger register in that case. The architecture demands the additional timing slot for data fetching into the operand register before the data may be set into the trigger register (relations (2), (5)):

$$CD_c(t_{Din}, t_{Dout}) = (C_i(O) - C_i(F_{in})) + (C_i(T) - C_i(O)) + (C_i(R) - C_i(T)) + (C_i(F_{out}) - C_i(R)) \geq 4. \quad (10)$$

The similar consideration may be applied if the result register is tied to one of the operands. The number of cycles for the execution will further increase if all of the registers are tied to the same bus.

3.2 Test cost in TTA structures

The structure of the pipelined component itself from the figure 3 and their convenient accessibility from the busses offers an ideal workaround for the application of the functional test without adding any extra circuitry. Actually, the test itself is not completely functional since the test patterns of each component are precalculated,

derived on the structural way, by means of the ATPG. In order to apply these structural test patterns, the components' surrounding circuitry i.e. the functional signals of the sockets has to be set accordingly (Figure 5).

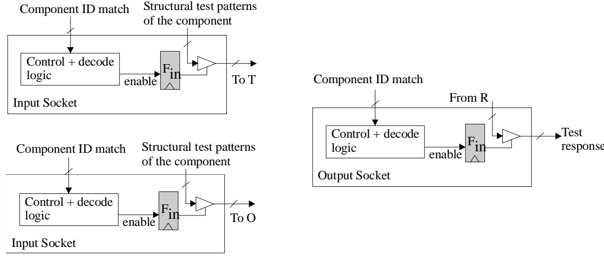

Figure 5. The functional test path through the sockets of the components

Our proposal is to express the test cost function with respect to the number of patterns and the number of cycles per patterns necessary to execute the tests, i.e., the cost is related to the testing time. Separate costs have to be distinguished for the functional units, f_{tfu}, and register files, f_{trf}. The analytical expressions for the test cost of the functional unit is given by:

$$f_{tfu} = n_p CD_{fu}(t_{Din}, t_{Dout}) \left\lceil \frac{n_{conn}}{n_b} \right\rceil \qquad (11)$$

Therefore, the test cost of the functional unit will increase with the ratio n_{conn}/n_b in the case when the number of components' ports exceeds the number of available busses in the architecture. In that case, there will be at least two registers connected to the same bus. For example, the figure 6 shows the two identical components ($FU_1 = FU_2$) where the $f_{tf1} < f_{tf2}$ due to their different ports' connectors.

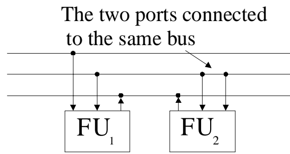

Figure 6. Two identical functional units with the different test cost

It is assumed that CD_{fu} is constant expression for particular component.

For the register files, we have proposed the following formula in order to assess their test cost:

$$f_{trf} = \begin{cases} \dfrac{n_p}{\min(n_{in}, n_{out})} CD_{rf}(t_{Din}, t_{Dout}) & n_{in} \vee n_{out} \le n_b \\ n_p \left\lceil \dfrac{\max(n_{in}, n_{out})}{n_b} \right\rceil CD_{rf}(t_{Din}, t_{Dout}) & otherwise \end{cases} \qquad (12)$$

where n_{in} and n_{out} denote the number of input and output ports, respectively. Hence, n_p in the latter equation is the number of the marching test patterns [14], necessary to test the registers within the register bank, and depends on the number of registers within the register file. The numerous register connectors might facilitate the test allowing the test vectors to be applied in parallel. However, it does not hold if both n_{in} and n_{out} are greater than n_b. In that case, the testing time will increase since the marching test patterns coming at the inputs (outputs) will be scheduled in different timing slots, thus increasing the test cost, as well. The cost for the register files is derived for the case of their implementation using a multi-ported memory [15], not a set of flip-flops. For the latter case, the test cost (as well as performance and area) will be different.

The sockets, being the control unit of the architecture, also influence the total test cost. The test of the sockets, including the stage controller modules given in the figure 3, can be performed using full scan. The test cost function of the sockets is labeled f_{ts}. Since the length of the scan chains determines the number of cycles for test, f_{ts} is proportional to the number of the patterns to test one socket (n_p), and to the length of the scan chains inside the socket (n_l):

$$f_{ts} = n_p \, . \, n_l \qquad (13)$$

The total test cost of the architecture is expressed as:

$$f_t(\underline{p}) = \sum_{i=1}^{n_{fu}} f_{tfu_i}(\underline{p}) + \sum_{j=1}^{n_{rf}} f_{trf_j}(\underline{p}) + \sum_{k=1}^{n_s} f_{ts_k} \qquad (14)$$

where n_{fu}, n_{rf} and n_s represent the total number of functional units, register files and sockets, respectively.

The test of the sockets also tests all interconnections inside the datapath. Note that the order of test is important for these architectures, i.e. it is necessary to perform the interconnect test of the sockets and busses before carrying out the functional test of the components. In that sense, our approach has some similarities with *Core-Based Test* [16] strategy where the test consists of two steps: interconnect and IP test. The functional test of TTA may be regarded as an IP test in the Core-Based test when predefined patterns are applied to test the embedded cores with properly configured surrounding infrastructure.

Another advantage of our approach is that the functional test of the components may also be used for delay fault tests, since it basically checks not only the structure of the components but also their timing relations (2-8). In addition, our approach can be extended to any type of regular bus-oriented VLIW ASIP architectures [17,18]. The figure 7 shows the general structure of such architectures.

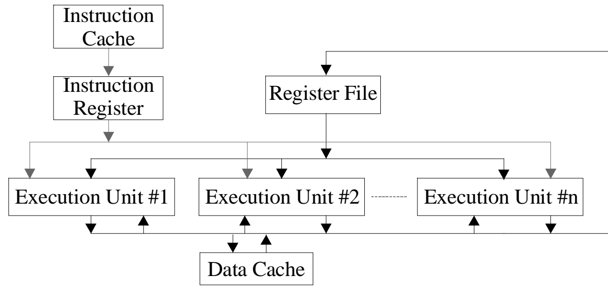

Figure 7. The bus-oriented VLIW ASIP template

Since most of the components are directly accessible from the bus, it is obvious that their test can be done by means of the functional application of structural test patterns. A few modifications are, however, required if the components are connected to the bus through the other components. For example, figure 7 shows the VLIW architecture where the output of the register file is connected to the bus through one or more functional units. In these situations, the order of testing the components becomes relevant and also a different set-up of the control signals has to take place.

4. The Architecture Selection

Our approach has been validated with regard to the particular "Crypt" application written in C++. The design space exploration within the MOVE framework has been carried out first, after which the set of Pareto points shown in figure 2 is obtained. Next, the test cost calculation, applying the analytical formulas as previously presented, has been performed for each architecture. Procedures (external from MOVE), written in the script language *gawk* and C++, are used for that purpose. The resulting set of Pareto points in $\mathcal{R}^3$ vector space with respect to the circuit area, throughput and test cost is given in figure 8. The already achieved area-throughput ratio is preserved since the first projection of the 3D curve in the area-execution-time plane is still the curve from figure 2.

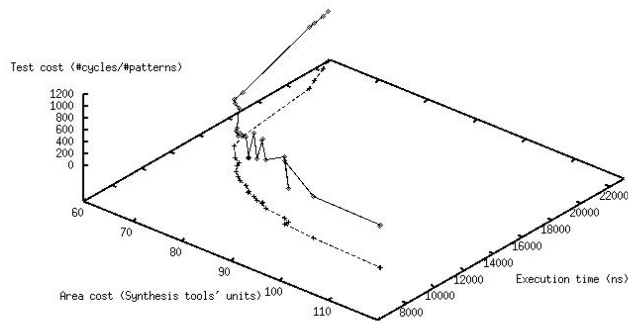

Figure 8. 3D Pareto points with respect to circuit area, performance and test

The figure 8 shows that the test cost may vary significantly even for the architectures that are close each other at the 2D Pareto curve. The selection of the most appropriate architecture can be done using any of the standard weighted norm techniques within the vector space $\mathcal{R}^3$ [19]. The weights are expressing the significance of a constraint over other constraint. The standard Euclid norm $|\underline{x}|_E$ $\underline{x} \rightarrow x(a, t, f_t)$ with equal constraint weights has been used, i.e., no preferences have been given neither to the minimum test, nor area, nor throughput (execution time). The resulting architecture with minimal norm is depicted in figure 9. The data-bus width of the architecture in figure 9 is 16 bit. The control signals and bits are not shown, they are adjoined to the data-bus. The ALU unit is capable of performing the operations of the addition, subtraction, shifting and basic logical operations (AND, OR, XOR).

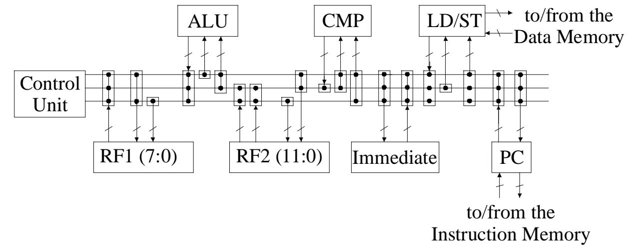

Figure 9. The architecture with the best circuit area-execution time-test cost (equal weighting factors)

Table 1 shows the number of the test patterns using our approach for the architectures' components from figure 9 and compares it with the test patterns obtained with full scan technique. Note that the number of the test patterns in this case corresponds to the number of cycles for their application. It is obvious that our approach requires significantly fewer cycles for the component's testing than the full scan. The sockets' patterns are included inside each component patterns. Table 1 also shows the length of the scan chains (n_l), the value of the test cost for functional units and register files (f_{tfu}, f_{trf}) and the value of the test cost for the sockets (f_{ts}) of each component. It has been adopted that all scan chains are connected to one single scan chain, so that the total test cost of the architecture equals to the sum of the test cycles of the components in the architecture. Of course, in the case of multiple scan chains, the total test cost will change due to the scheduling of test patterns. However, that change will be reflected equally to both full scan and our approach since the test of the sockets is scan-based in our approach. Hence, our method still retains the advantage over the full scan. Fault coverage of the datapath components in the fourth column of the table refers to our approach.

Table 1. The comparison of full scan and our methodology for the components from Figure 9.

Component	full scan	our approach	n_l	f_{tfu}	f_{trf}	f_{ts}	FC (%)
ALU	7208	877	58	65	-	812	99.72
CMP	4556	884	58	72	-	812	99.78
RF1	1912	882	58	-	70	812	99.78
RF2	2083	1144	75	-	94	1050	99.48
LD/ST	964	(964)	58	-	-	-	99.78
PC	1112	(1112)	58	-	-	-	99.78

Our approach does not take the Load/Store unit (LD/ST) and program counter (PC) into account during the test cost calculation. They always appear once for arbitrary architecture and application; hence, they contribute equally to any of the remaining architectural parameters in any sense. The same holds for the Immediate unit. RF1 and RF2 could not have been tested with full scan, unless implemented as a set of flip-flops. However, the flip-flop implementation of register files would increase its area after DfT scan insertion considerably compared to our approach. This is another justification for using our approach to test these architectures.

5. Conclusion

The test space exploration of transport triggered architectures has been addressed in this paper. The approach that assesses the architecture with respect to the test has been explained. In addition, our approach minimizes the extra DfT circuitry necessary for test. Furthermore, it does not hinder already achieved area/performance ratio, which is one of the most important issues of the TTA. The fact that allows one to minimize the extra circuitry is the regularity of the TTA, i.e., the direct accessibility of the components in the datapath via busses. The analytical test cost functions for the functional test of the components are derived according to the transport-timing relations and they are dependent of the architectural parameters, only. Our approach has been validated with respect to the example of "Crypt" Unix application where the advantages of our methodology over the classical full-scan are shown.

References:

[1] H. Corporaal, *Transport Triggered Architectures; Design and Evaluation*. PhD thesis, Delft University of Technology, September 1995.

[2] H. Corporaal, "*Microprocessor Architectures from VLIW to TTA,*" ISBN 0-471-97157-X, John Wiley, 1998.

[3] H. Corporaal, M. Arnold, "Using Transport-Triggered Architectures for Embedded Processor Design," *Integrated Computer-Aided Engineering,* vol. 1998, no. 1, pp. 19-37, 1998, ISSN: 1069 – 2509.

[4] B. R. Rau, J. A. Fisher, "Instruction-Level Parallel Processing; History, Overview and Perspective," *Journal of Supercomputing*, vol. 7, no. 2, May 1993, pp. 9-50.

[5] H. Corporaal, J. Hoogerbrugge, "Cosynthesis with the MOVE Framework," CESA'96, IMACS Multiconference, Lille, France, 1996.

[6] R. Brayton, R. Spence, *Sensitivity and Optimisation*, Elsevier 1980.

[7] UNIX password and DES encryption, ref. "*Mathematical Cryptology for Computer Scientists and Mathematicians*", W. Patterson, 1987.

[8] R. P. van Riessen, *Module Generation for Self-Testing Integrated Systems*, PhD thesis, University of Twente, January 1992.

[9] C. H. Chen, D. G. Saab, "A Novel Behavioral Testability Measure," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 12, December 1993, pp. 1960 - 1970.

[10] Y. Le Traon, G. Al-Hayek, C. Robach, "Testability-oriented Hardware/Software partitioning," *Proceedings of the International Test Conference,* October 20-25, 1996, Washington DC, USA, pp. 727-731

[11] R. D. Shawn, J. P. Hayes, "Design of a fast, Easily Testable ALU," *Proc. of the 14th IEEE VLSI Test Symposium*, 1996, pp. 9-16.

[12] V. A. Zivkovic, R. J. W. T. Tangelder, H. G. Kerkhoff, "Codesign and Test of VLIW Processor", *Proceedings of ProRISC Circuits, Systems and Signal Processing CSSP 98*, November 1998, Mierlo, the Netherlands, pp. 625-630.

[13] D. Gizopoulos, A. Paschalis, Y. Zorian, "An Effective BIST Scheme for Datapaths," *Proceedings of the International Test Conference*, October 20-25, 1996, Washington DC, USA, pp. 76-85.

[14] A. J. van de Goor, "Testing of the Semiconductor Memories", Kluwer Publishers, 1991.

[15] S. Hamdioui, A. J. van de Goor, "Consequences of Port Restrictions on Testing Two-Port Memories," *Proceedings of the International Test Conference*, October 18-23, 1998, Washington DC, USA, pp. 63-72.

[16] E. J. Marinissen, e.a., "A structured and Scalable Mechanism for Test Access to Embedded Reusable Cores", *Proceedings of the International Test Conference*, October 18-23, 1998, Washington DC, USA, pp. 284-293.

[17]http://www.semiconductors.com/trimedia/news/index.html# articles "Tri Media Update"; Volume 1, Issue 1, 1st Quarter 1998.

[18] J. Wilberg, *Codesign for Real-Time Video Applliccations*, Kluwer Academic Publishers, Dordrecht, the Netherlands, ISBN 0-7923-8006-1, 1997.

[19] G. Milovanovic, *Numerical Analysis, part I*, Scientific Book Edition, 1991.