A New SBST Algorithm for Testing the Register File of VLIW Processors

Davide Sabena, Matteo Sonza Reorda, Luca Sterpone
Dipartimento di Automatica e Informatica
Politecnico di Torino
Torino, Italy

Abstract—**Feature size reduction drastically influences permanent faults occurrence in nanometer technology devices. Among the various test techniques, Software-Based Self-Test (SBST) approaches have been demonstrated to be an effective solution for detecting logic defects, although achieving complete fault coverage is a challenging issue due to the functional-based nature of this methodology. When VLIW processors are considered, standard processor-oriented SBST approaches result deficient since not able to cope with most of the failures affecting VLIW multiple parallel domains. In this paper we present a novel SBST algorithm specifically oriented to test the register files of VLIW processors. In particular, our algorithm addresses the cross-bar switch architecture of the VLIW register file by completely covering the intrinsic faults generated between the multiple computational domains. Fault simulation campaigns comparing previously developed methods with our solution demonstrate its effectiveness. The results show that the developed algorithm achieves a 97.12% fault coverage which is about twice better than previously developed SBST algorithms. Further advantages of our solution are the limited overhead in terms of execution cycles and memory occupation.**

Keywords- Testing, software-based self test, Very Long Instruction Word Processors, Fault Simulation.

I. INTRODUCTION

The continuous aggressive reduction of semiconductor fabrication process permits today the implementation of SoCs with multimillion transistors operating at gigahertz frequencies and integrating on a single chip several hundreds of modules, computational cores and memory blocks. Thanks to the reduced feature size, SoC performances are extremely enhanced. However, due to the highly stressed production process, phenomena like metal migration or aging of the circuit may increase the occurrence of permanent faults. On the other hand, testability is progressively limited due to the aggressive deep-submicron geometries which may also provoke the appearance of new kinds of defects.

Traditional usage of Automatic Test Equipment (ATE) for SoC tests is a valid method for deploying at-speed tests able to obtain high test quality; however, since extremely expensive, it cannot always be considered an economically affordable solution. In the last decade, several Software-Based Self-Test (SBST) techniques have been proposed following this approach [1] [2]. The basic idea of SBST is to generate test-patterns by executing program sequences, where processor instructions are used to test the processor's functionality.

978-3-9810801-8-6/DATE12/©2012 EDAA

Various SBST methodologies have been proposed as an effective or additional solution for the manufacturing test of processors or SoCs. One of the main advantages of SBST is its non-intrusiveness, since it does not require any extra hardware; therefore, cost is reduced, and any critical path delay penalty is avoided, while allowing at-speed testing. SBST methods have been applied to a large set of processors and SoCs [3]. Among the various microprocessor architectures, Very Long Instruction Word (VLIW) processors have been demonstrated to be the most viable solution for several embedded applications characterized by high performance, low cost and low power consumption. Thanks to their architecture, VLIW solutions achieve a very good exploitation of the available instruction level parallelism (ILP). Nowadays, VLIW manufacturing is on-going in several semiconductor companies and the problem of testing their functional architecture is increasing relevant.

A key feature of VLIW processors is the instruction format; in fact VLIW architectures are characterized by grouping several instructions into large macro-instructions called *bundles*, where each instruction within the bundle is executed in parallel distinct computational units referred as *computational domains*. VLIW processors are then organized in *clusters* formed by several computational domains (i.e. generally four or eight). On one hand ILP allows to improve performances, on the other hand it creates instruction interdependence that must be taken into account. This issue is generally solved by software compilers which should be suitable designed in order to address interdependence between different computational domains. A key hardware component which supports the ILP execution is the register file. The VLIW register file is characterized by several multiported registers for a single VLIW processor cluster [4]. It is one of the most resource consuming module of a VLIW processor, since the number of read and write ports in each multiported register increases with the number of computational domains and consequently results in an exponential increase in resources [5]. Internally, the register file has the architecture of a complex cross-bar switch, where each computational domain can write to each single register, and the content of each single register may be transferred to whatever computational domain [6].

Several SBST approaches have been developed in order to address VLIW processors: most of them rely on software

techniques where the approaches adopt suitable instructions belonging to the processors instruction set to apply the test-patterns, generated off-line by an automatic test pattern generator (ATPG) tool based on the internal components structure. Although effective, these methods have several main drawbacks: first of all, they delegate the pattern generation to an external ATPG, which drastically increases the total test time; secondly, they are applied at the software level before the compiler execution, therefore the generated assembly code it is not always fully compliant with the existing testing constraints; finally, these approaches are all based on a limited version of the VLIW architecture where each computational domain may read / write data only from a given set of registers within the register file [11]. Therefore, most of the faults really affecting the behavior of the VLIW processors are not covered by these methods. In this paper we propose a new SBST algorithm specifically oriented to test the typical register files embedded in VLIW processors. The proposed method does not require any hardware change or addition to the processor architecture. Moreover, since the method is totally functional-based, it does not require the usage of any external ATPG in order to generate the input stimuli.

The main contribution of the developed test method to the advancement of state-of-the-art techniques in the area is the first algorithm able to effectively test a multi-port cross-bar switch embedded into a VLIW register file following an SBST approach. The proposed algorithm has small requirements in terms of memory (to store the test code) and the execution time is very limited. The developed test algorithm is applied at the post-compiler level, therefore it has an extended testing capability with respect to approaches applied before the compile time since it has full control on the execution code. The developed SBST algorithm has been evaluated on a realistic VLIW platform based on the *Delft University r-VEX VLIW* Processor which allows to implement the great part of features generally embedded within industrially manufactured VLIW architectures [6]. The results we achieved clearly demonstrate the efficiency of our approach, since by fault simulation analysis on the VLIW register file we achieved a fault coverage on stuck-at faults higher than 97 %.

The paper is organized as follows. Section II describes the related works on software-based self-test techniques specifically oriented to VLIW processor. Section III gives an overview of the VLIW data path architecture while Section IV outlines the proposed test algorithm. Experimental results and their analysis are presented in Section V. Finally, conclusions and future works are described in Section VI.

II. Related Work

Popular techniques to test processor chips and processor-based System-On-Chip (SoCs) are Built-In Self-Test (BIST) and Software-Based Self-Test (SBST). The methodologies that require external hardware to perform the test are infeasible without the use of multimillion dollar Automatic Test Equipment (ATE); this is due to the increasing gap between ATE frequencies and SoC operating frequencies which makes external at-speed testing problematic and expensive (at-speed testing is needed because some failures can be detected only when the test is performed at the operating frequency of the device). Moreover, external test often involves long time and significant efforts to introduce the required hardware and may be characterized by long test application times [1]. To avoid these drawbacks self-test methodologies can effectively be adopted. In the literature there are many papers related to methods for the functional self-test of processors, but few of them refers to the test of Very Long Instruction Word (VLIW) processors.

BIST moves the testing task from external resources (ATE) to internal hardware: additional hardware and software are integrated into the circuit to allow it to perform self-testing. The use of this technique leads to lower cost of test and shorter tests time, maintaining or improving the fault coverage, at the cost of additional silicon area.

Another way for on-chip testing is SBST that is a non intrusive methodology since it adopts existing processor resources and instructions to perform self-testing. The advantage of this technology is that it uses only the processor functionality and instruction set for both Test pattern Generation and Output Data Evaluation, and thus does not introduce any hardware overhead in the design [8]. However, software-based methods suffer from long program sequences to achieve high coverage of the device under test.

Recently, many techniques have been developed to test a generic superscalar processor [1], [7], [9] and the results obtained show that SBST is a valid and low cost methodology. Moreover, up to now, there are few SBST techniques that refer to VLIW processors, mainly due to the fact that the approaches developed for superscalar processors are not easily implementable on this architecture.

A methodology combining functional test and Built-In Self-Test and Repair for regular data path structures within VLIW processors is described in [11]. In this approach fault detection and localization are performed by software and then a hardware reconfiguration using redundant units is exploited. In this methodology Software-Based Self-Test guarantees high fault coverage; however, the VLIW core used as a test vehicle presents some peculiarities that differentiate it from the most common VLIW processors; the main one refers to the register file: in fact, for each cluster within the processor there is a separate register file and there are not registers shared by all clusters. This implies that this processor does not include a structure that connects all functional units to all registers (i.e. a crossbar); this is an important constraint, since a crossbar is a significant structure in terms of area and the related faults are numerous and difficult to detect using the SBST methodology.

Another technique developed to test VLIW processors combines scan and SBST in order to obtain a good diagnostic resolution at low hardware overhead [10]. The peculiarity of this approach, aimed at detecting faults in the functional units of the processor, is that the same test patterns are loaded directly into the fetch registers of all computational domain. The proper functioning of each domain is tested by comparing the test response of all domain, that should be the same in the fault-free case. This solution involves a hardware overhead of

about 6% and requires that the processor runs in a special self-test mode.

III. Background on the R-VEX VLIW Architecture

A generic VLIW processor may have different numbers of functional units (FUs). Generally, the VLIW processor architecture is parametric, so that different options such as the number and type of the functional units, the number of multi-ported registers (i.e. the size of the register file), the width of the memory buses and the type of different accessible FUs can be modified depending on the application's requirements [12]. In the present work, we addressed on the Delft Reconfigurable VLIW processor (r-VEX) which includes all the features of the generic VLIW processors existing on the market [5]. The r-VEX processor is based on 32-bit data registers with the main processor architecture based on 4 FUs organized in four different computational domains. The r-VEX main processor architecture consists of four stages pipeline organization: fetch, decode, execute and write-back stages. The fetch unit fetches a VLIW instruction from the attached instruction memory and splits it into four syllables that are passed to the decode unit. In this stage, registers used as operands are fetched from the register file. The actual operations take place either in the execution unit, or in one of the parallel branch/control (CTLR) or load/store memory (MEM) units. The arithmetic logic unit (ALU) functions and multiplier (MUL) operations are performed in the execution stage, while CTRL unit manages branch operations. Vice versa the MEM unit handles all data memory load and store operations. All write activities are performed by the write-back unit at the same time. These operations are performed in specific computational domains, since ALU operations (A) are performed by all the execution units, while CTRL (B) and MEM (S) operations are executed only by the first and fourth computational units and MUL operations are executed by the second and third units. The data are stored in two different register files: the general register file (GR) and the branch register file (BR). The two register files are included into the Decode unit, where the register input and output lines are controlled respectively by an input and an output cross-bar switch. The data memory stores the results of the execution.

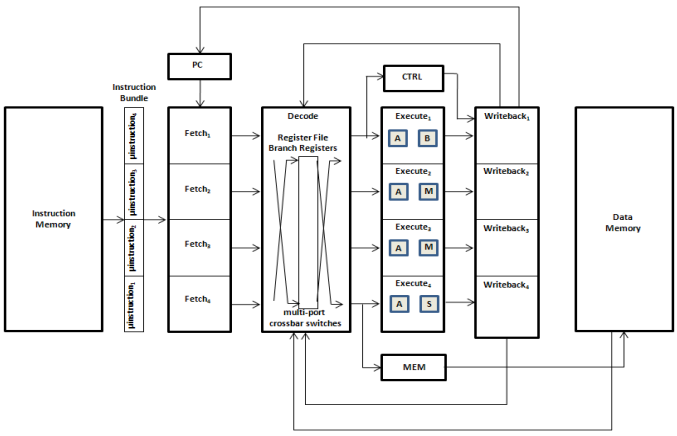

Figure 1. The VLIW processor data path.

As outlined in the previous sub-section, the VLIW data path includes the operational register file and all the functional modules. Starting from the fetch stage the VLIW instructions are splitted into four different computational domain that will be executed in parallel. The computational domain division is maintained in all the processing units.

In order to using the r-VEX processor as a test vehicle for our experiments, we synthesized and implemented it on a standard ASIC gate library. In order to develop a suitable SBST algorithm, it is important to understand which components of the processor have the largest contribution to the overall processor fault list [8]. As shown in [5] and [13], the register file of the r-VEX VLIW processor occupies the large percentage of logic resources of the entire VLIW architecture. In details, when the number of registers within the register file increases, the register file resource area exponentially increases. For example, a register file having 32 or 64 registers requires 40% or 59% of the overall VLIW area respectively. This is mainly due to the high number of general purpose registers and to the logic circuitry necessary to provide their accessibility: indeed, registers are shared by all computational domains of the VLIW processor and several multi read and write ports are necessary with respect to traditional processor register files. This confirms that the register file of a VLIW processor is one of the components with the largest differences if compared to those of superscalar processors [5]. For this reasons we focused our efforts on this module.

IV. The Proposed SBST Algorithm

The key characteristics of our SBST algorithm for VLIW processors are essentially two: the first is that in order to develop the test routine, it considers each component of the processor as a single independent unit, the second feature is that the test development is based only on the Instruction Set Architecture (ISA) of the VLIW processor. In this section we first describe the details of the register file and outlines some basic idea for testing it, then we focus on the description of the developed SBST algorithm.

A. Register File

The register file implemented within the r-VEX VLIW processor consists of 64 32-bit wide registers; 3 of them are special register: r0.0 is constant always settled at '0', r0.1 is the *stack register*, and r0.63 is the *link register*. The rest of the registers are general-purpose. Each general-purpose register can be accessed by each computational domain of the r-VEX VLIW processor. In details, a single computational domain has one write-port to access to the register file to store a data in a register; and two read-ports by which it can read from two different registers at the same clock cycle as illustrated in Figure 2. This implies that within the register file there is a write address decoder, a read address decoder and two read-ports for each domain. This architecture is taken into account in our developed test methodology in order to achieve high fault coverage.

This structure, corresponding to a *cross-bar* can be implemented through a set of 2-to-1 *multiplexers* (for data output) and 1-to-2 *demultiplexers* (for data input); this allows to implement a cross-bar switch architecture where each domain's data input can be stored in whatever register, and can be read by whatever domain's read-port. In order to fully test this particular structure of the register file each register must be accessed for at least a read and a write operation by each of the four VLIW domains; moreover, with regard to reading operations, all registers must be accessed by *read_port_1* and *read_port_2* of each domain as shown in the pseudo code depicted in Figure 3. In this way, all the possible register datapaths inside the register file are exercised. Moreover, it is necessary that each register is assigned with a value and then with the complemented value in order to guarantee that each possible stuck-at fault in the register bits is detected. Therefore, considering that the r-VEX VLIW processor has four domains, 8 write operations and 16 read operations are necessary for each register.

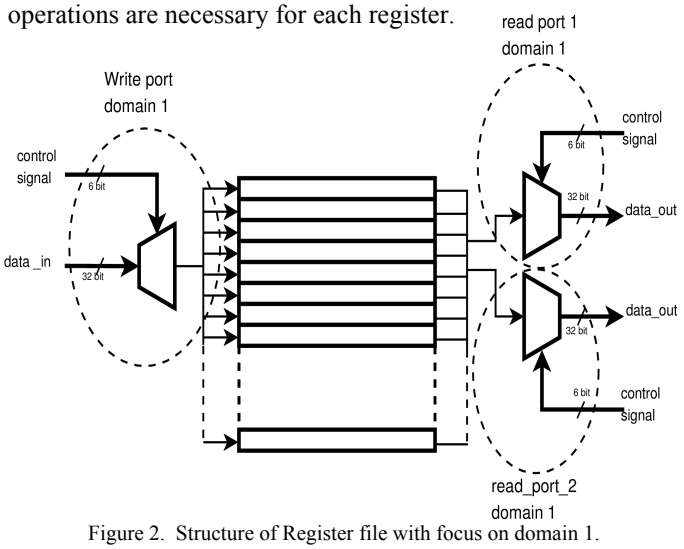

Figure 2. Structure of Register file with focus on domain 1.

B. *The proposed SBST algorithm*

The SBST algorithm we propose is intended to be implemented in assembly code. For convenience, in this section we formalize it describing the pseudo-code of the main procedure. The algorithm essentially focuses on the generation of test instructions for addressing the register file crossbar switches architecture. Since the crossbar switches are generally implemented using multiplexers and demultiplexer modules, which physical implementation depends on the selected technology, it is important to analyze the physical implementation of this components in order to cope with the internal stuck-at faults. The method we used to assign values to register were developed inspiring from the methodology previously developed in [1]. The main procedure, which is illustrated in Figure 3, consists of two parts, in each part, half of the registers are under test while the others are used to manage the test execution. This allows using all the VLIW registers without adding external hardware resources in order to support the test execution. The two parts are executed for a given number of phases, in our cases eight times according to the register file functional analysis illustrated in the previous section. For each phase execution, the logic value assigned to the registers changes.

The logic value assignment is performed by the function `assign_value_to_reg_R_R` that, on the basis of the received parameters, selects which type of assignment has to be used. For instance, considering the register assignment rules, illustrated in Table I, if the phase A_1 is considered, the set of registers r0.0 to r0.31 are assigned as illustrated in Figure 4.

```
//part 1
assign_value_to_reg_R0_R31(Phase);
for (each domain D){
  for (each register R ε(R0…R31){
    for (each read_port P){
       read value V of register R using domain D and
         read_port P;
       use V to compute a signature S;
    }
  }
   store S in memory;
}
//part 2
assign_value_to_reg_R32_R63(Phase);
for (each domain D){
  for (each register R ε(R0…R31){
    for (each read_port P){
       read value V of register R using domain D and
         read_port P;
       use V to compute a signature S;
    }
  }
store S in memory;
}
```

Figure 3. The pseudo-code of the proposed algorithm.

The eight types of logic value assignment performed are illustrated in Table I where each row of the table represents a single macro-instruction of the r-VEX VLIW processor: the first instruction is executed by the first domain, the second by the second domain and so on. These executions are performed during the same clock cycle. Please notice that within each macro-instruction the order of the registers is changed between phase A_1, B_1, C_1 or D_1, whereas the data assigned to the registers are the same. Vice versa, the difference between phase A_1 and A_2, or between B_1 and B_2, or C_1 and C_2, or D_1 and D_2 are the data used to assign the registers, that in phases A_2, B_2, C_2 and D_2 are complemented respect to A_1, B_1, C_1 and D_1.

It is mandatory that the main procedure is performed with this eight types of assignment in order to exploit all possible datapaths within the register file, using first a value and then the complemented one. This procedure allows effectively covering stuck-at faults affecting the crossbar switches circuitry. The procedure is based on the idea that *n-to-1* multiplexers are generally decomposed and implemented by smaller muxes ordered in a tree structure [1]; therefore, in order to minimize the probability that a fault occurred in a crossbar cannot be traced it is important that the registers are assigned following the rules reported in Table I. In particular, it is mandatory that registers encoded with a bit string having Hamming distance equal to 1 are assigned with different logic values, in order to detect stuck-at faults that occur on the

selection of a register in both writing or reading operations. Since the register index in encoded with 6 bits, 8 different logic values are used. They are represented with *A,B,C,D,E,F,G,H* where B=NOT(A), D=NOT(C), F=NOT(E) and H=NOT(G) and A≠C≠E≠G. The actual values for A, C, E and G are not important.

```
0]  -------------------------
mov $r0.32,01010101010101010101010101010101   #A
mov $r0.33,10101010101010101010101010101010   #B
1]  -------------------------
mov $r0.34,00110011001100110011001100110011   #C
mov $r0.35,11001100110011001100110011001100   #D
2]  -------------------------
mov $r0.36,00001111000011110000111100001111   #E
mov $r0.37,11110000111100001111000011110000   #F
3]  -------------------------
mov $r0.38,00000000111111110000000011111111   #G
mov $r0.39,11110000111100001111000011110000   #H
4] -------------------------
mov $r0.0 = $r0.32     #A
mov $r0.1 = $r0.33     #B
mov $r0.2 = $r0.33     #B
mov $r0.3 = $r0.32     #A
5] -------------------------
mov $r0.4 = $r0.34     #C
mov $r0.5 = $r0.35     #D
mov $r0.6 = $r0.35     #D
mov $r0.7 = $r0.34     #C
6] -------------------------
…
8] -------------------------
mov $r0.24 = $r0.38    #G
mov $r0.25 = $r0.39    #H
mov $r0.26 = $r0.39    #H
mov $r0.27 = $r0.38    #G
9] -------------------------
mov $r0.28 = $r0.36    #E
mov $r0.29 = $r0.37    #F
mov $r0.30 = $r0.37    #F
mov $r0.31 = $r0.36    #E
10] ------------------------
```

Figure 4. Assembly code example used during the register file logic value assignment.

TABLE I. LOGIC VALUE REGISTER ASSIGNMENT RULES FOR THE FIRST TWO MACRO-INSTRUCTIONS

Phase A_1	**Phase B_1**	**Phase C_1**	**Phase D_1**
R0 = A	R1 = B	R2 = B	R3 = A
R1 = B	R0 = A	R3 = A	R2 = B
R2 = B	R3 = A	R0 = A	R1 = B
R3 = A	R2 = B	R1 = B	R0 = A
R4 = C	R5 = D	R6 = D	R7 = C
R5 = D	R4 = C	R7 = C	R6 = D
R6 = D	R7 = C	R4 = C	R5 = D
R7 = C	R6 = D	R5 = D	R4 = C
Phase A_2	**Phase B_2**	**Phase C_2**	**Phase D_2**
R0 = B	R1 = A	R2 = A	R3 = B
R1 = A	R0 = B	R3 = B	R2 = A
R2 = A	R3 = B	R0 = B	R1 = A
R3 = B	R2 = A	R1 = A	R0 = B
R4 = C	R5 = C	R6 = C	R7 = C
R5 = D	R4 = D	R7 = D	R6 = D
R6 = D	R7 = D	R4 = D	R5 = D
R7 = C	R6 = C	R5 = C	R4 = C

Once the initialization is executed, a set of permutations are performed in order to stimulate the access to the register files in all the possible access permutations. In Figure 5 an example of assembly code is reported by which it is possible to read a register with all *read_port*s of the selected domain. Particularly, considering the domain 1 and the register r0.0 we notice that this register is used as operand 1 in the first instruction of the macro-instruction 1 and then as operand 2 in the first instruction of macro-instruction 2. Therefore, the register r0.0 is accessed by all the *read_port*s of domain 1. This procedure must be repeated for each register in order to consider all paths.

```
1]  -------------------------
  xor $r0.34 = $r0.0, $r0.1
  xor $r0.35 = $r0.2, $r0.3
  xor $r0.36 = $r0.4, $r0.5
  xor $r0.37 = $r0.6, $r0.7
  2]  -------------------------
  xor $r0.34 = $r0.1, $r0.0
  xor $r0.35 = $r0.3, $r0.2
  xor $r0.36 = $r0.5, $r0.4
  xor $r0.37 = $r0.7, $r0.6
  3]  -------------------------
```

Figure 5. Assembly code by which is possible to read r0.0 using the two *read_port* of the first domain.

Finally, in order to minimize the number of memory accesses and therefore limiting the length of the whole test program, the proposed algorithm compute a signature calculation using a small LFSR algorithm implemented in few assembly instructions. The implemented LFSR algorithm avoid logic fault masking.

The major advantage of the proposed algorithm is that it is designed to be adopted to test a generic register file for VLIW processors and it not specific for the register file of the r-VEX processor. This is due to the fact that the algorithm is fully parametric and can be used to test a register file with different number of read- and write- ports and a different number of computational domains.

V. EXPERIMENTAL RESULTS

We analyzed the efficiency of the proposed SBST algorithm by performing several fault simulation campaign injecting stuck-at faults into the r-VEX VLIW model. We firstly analyzed the structure of the register file in order to identify the instructions that properly excite this component operations and the instructions for controlling and observing the registers.

A. Fault simulation results

Fault simulation is performed with respect to the stuck-at fault model. The results of the fault simulation experiments, related to the register file, are reported in Table II, where for each test program, we showed the duration in terms of number of clock cycle and reached fault coverage. The whole fault list is composed of 259,716 faults.

The test program 1 consists of a simple assignment of all registers with a value and then with a complemented value; the coverage reached using this basic method is very low. The test program 2 implements a test methodology developed to test a generic register file of a superscalar processor. It is

possible to notice that there are problems similar to those of the previous algorithm, although the coverage is increased, it still remains overall low. This is mainly due to use of the write and read ports that, using a generic techniques cannot be fully tested.

TABLE II. FAULT SIMULATION RESULTS

Algorithm	Coverage	# Clock Cycles
Test Program 1	35.74%	130
Test Program 2 [1]	56.26%	190
Test Program 3	72.44%	477
Test Program 4	91.49%	949
Test Program 5	95.44%	849
Test Program 6	95.71%	1050
Our SBST Algorithm	97.12%	760

The test program 3 makes write operations exploiting all possible datapaths; however it has the drawback that each register is read by only one read port of each domain: in this manner the coverage slightly increases. In the test program 4 we took into account the need of using all the *read_port*, thus increasing the coverage by 19.5%. The test program 5 improves the previous ones by including the assignment methodology derived from the approaches reported in [1]; special registers r0.63 and r0.1 are also covered. The test program 6 uses eight different values assigned to the registers in order to guarantee that registers whose index has a Hamming distance equal to 1 are assigned with different logic values. Finally, we wrote the last program according to the proposed algorithm described in Section IV. Please note the corresponding reduction in the test duration, due to the optimizations introduced by the algorithm. When analyzing the faults that remain untested, it is important to note that the large part of them are located on the reset signal of the 2,272 Flip-Flops composing the register file, and this signal cannot be controlled via software access, resulting in a 0.87% of faults that are untestable. Moreover, another feature that avoids that the coverage reaches the maximum is the register r0.0, which by its nature cannot be written.

VI. CONCLUSIONS AND FUTURE WORKS

In this paper, we first showed that the register file is a significant and critical component for most VLIW processors in terms of testing, and then presented an SBST algorithm specifically developed to test it.. The algorithm provides an advancement with respect to state-of-the-art techniques in the area, since it is the first algorithm able to effectively test cross-bar switch-based register files embedded into VLIW processors. The proposed algorithm has a small impact in terms of memory used to store the test code and the execution time is drastically limited with respect to other solutions. It extends test capabilities with respect to previously developed approaches since it is applied after the compile-time, therefore having full control on the execution code. We checked the effectiveness of this algorithm on a real VLIW platform based on the r-VEX VLIW processor [6]. The obtained results clearly demonstrate the efficiency of our algorithm, since we achieved a fault coverage on stuck-at faults higher than 97%. As future works we plan to better evaluate the performances of the proposed solution and to investigate its applicability to on-line testing and to evaluate the fault coverage with respect to transition delay faults.

REFERENCES

[1] Nektarios Kranitis, Antonis Paschalis, Dimitri Gizopoulos, George Xenoulis, *Software-based self testing of embedded processors,* IEEE Transaction on Computers, vol. 54, no. 4, pp. 461 – 475, April, 2005.

[2] Psarakis, M.; Gizopoulos, D.; Sanchez, E.; Reorda, M.S., *Microprocessor software-based self-testing,* Design & Test of Computers, vol. 27, no. 3, pp. 4 – 19, June 2010.

[3] N. Kranitis, A. Merentitis, G. Theodorou, A. Paschalis, D. Gizopoulos, *Hybrid-SBST Methodology for Efficient Testing of Processor Cores*, Design & Test of Computers, Vol. 25, no. 1, pp. 64 – 75, February 2008.

[4] J. A. Fisher, P. Faraboschi and C. Young, *Embedded Computing: A VLIW Approach to Architecture, Compilers and Tools*, Morgan Kaufmann, pp. 671, 2004.

[5] S. Wong, F. Anjam and F. Nadeem, *Dynamically Reconfigurable Register File for a Softcore VLIW Processor*, IEEE International Conference on Design, Automation and Test in Europe, pp. 962 – 972, March, 2010.

[6] S. Wong, T. Van As, G. Brown, *σ-VEX: A reconfigurable and extensible softcore VLIW processor*, International Conference on ICECE Technology, pp. 369 – 372, December, 2010.

[7] M. Schölzel, *HW/SW Co-Detection of Transient and Permanent Faults with Fast Recovery in Statistically Scheduled Data Paths*, International Conference on Design, Automation and Test in Europe (DATE), pp. 723 – 728, March, 2010.

[8] Krstic, A.; Wei-Cheng Lai; Kwang-Ting Cheng; Chen, L.; Dey, S.; *Embedded software-based self-test for programmable core-based designs*, Design & Test of Computers, vol. 19, no. 4, pp. 18 – 27, August, 2002.

[9] Kranitis, N.; Gizopoulos, D.; Paschalis, A.; Zorian, Y., *Instruction-based self-testing of processor cores,* VLSI Test Symposium, pp. 223 – 228, August, 2002.

[10] Ulbricht, M.; Scholzel, M.; Koal, T.; Vierhaus, H.T, *A new Hierarchical Built-In Self-Test with On-Chip Diagnosis for VLIW processors,* Design and Diagnostic of Electronic Circuits and Systems (DDECS), pp. 143 – 146, April, 2011.

[11] Koal, T.; Vierhaus, H.T., *A Software-based self-test and hardware reconfiguration solution for VLIW processors,* Design and Diagnostic of Electronic Circuits and Systems (DDECS), pp. 40 – 43, April, 2010.

[12] J. Liu, B. Bell, T. Troug, *Analysis and Characterization of Intel Itanium Instruction Bundles for Improving VLIW Processor Performance*, International Multy-Symposium on Computer and Computational Sciences (IMSCCS'06), vol. 1, pp. 389 – 396, June, 2006.

[13] A. K. Jones, R. Hoare, D. Kusic, J. Fazekas and J. Foster, *An FPGA-based VLIW Processor with Custom Hardware Execution*, in Proceedings of the ACM/SIGDA 13th Internal Symposium on Field Programmable Gate Arrays (FPGA'05), pp. 107 – 117, 2005.