Scientific Journal Impact Factor (SJIF): 1.711
e-ISSN: 2349-9745
p-ISSN: 2393-8161
International Journal of Modern Trends in Engineering and Research
www.ijmter.com

Automatic generation of software based self test programs for VLIW processors

S.Maniyarasan[1], D.Sathyakala[2]
II-M.E(CS)[1], AP / ECE[2], Dhanalakshmi Srinivasan Engineering College, Perambalur.

Abstract—Software-Based Self-Test (SBST) approaches are the effective solution to detect permanent faults at the end of both the production process and during the operational phase, when Very Long Instruction Word (VLIW) processors are used these techniques require some optimization steps in order to properly exploit the parallelism intrinsic in these architectures. In this paper we present a new method from previously known algorithms and automatically generates an effective test program which can produce high fault coverage on the VLIW processor under test, by reducing the test code size which on effect reduces the test duration. The method consists of four stages of parametric phases and can deal with different VLIW processor models. The main aim of our proposed method is to automatically obtain a test program able to achieve high fault coverage with minimum test time and the required resources. Experimental data gathered shows effectiveness of the proposed approach that this method is able to exploit the intrinsic parallelism of the VLIW processor, reducing the grow thin size, and duration of the test program when the processor size grows.

Index Terms—Very Large Instruction Word(VLIW), Software Based Self Test(SBST).

I. INTRODUCTION

The continuous scaling in the semiconductor manufacturing process and to the increasingly high operation frequency of integrated circuits, processor chips face growing testability problems. Moreover, since the production processes are highly stressed, phenomena like metal migration or aging of the circuit may increase the occurrence of permanent faults in the systems, even during the circuit operational phase. For these reasons, new test solutions are being investigated in order to provide high fault coverage with acceptable costs (e.g., in terms of test time, silicon area over- head, and required test infrastructure). A promising approach for processors and processor-based systems (e.g., systems on a chip, or SoCs) corresponds to the so-called software-based self-test (SBST) : the basic idea is to generate test programs to be executed by the processor and able to fully exercise the processor itself or other components in the system, and to detect possible faults by looking at the produced results. One of the main advantages of SBST lies in the fact that it does not require any extra hardware; therefore, the test cost is reduced and any performance or area penalty is avoided. Moreover, the SBST approach allows at-speed testing, and can be easily used even for on-line testing. For these reasons, SBST is increasingly applied for processors and SoC testing, often in combination with other approaches.

Among the various microprocessor architectures, very long instruction word (VLIW) processors were demonstrated to be a viable solution especially for applications demanding high performance while exposing a considerable amount of parallelism, such as several digital signal processing algorithms used in multimedia and communication applications . VLIW processors are currently adopted in several products, in particular for embedded applications, and the problem of testing them is increasingly relevant.

VLIW processors are characterized by a pipelined architecture with multiple functional units (FUs). Unlike super- scalar processors, VLIW processors do not include any significant control logic, since instruction scheduling is completely performed by the compiler. This implies that the hardware complexity is far lower for superscalar processors, while the compilation steps become more complicated. Consequently, the control hardware of the processor is much more easily testable than in other processors (e.g., the super- scalar ones). Another key feature of VLIW processors is the instruction format. In fact, VLIW processors are characterized by grouping several instructions (named microinstructions) into one large macroinstruction (also called bundle), where each micro-instruction within the bundle is executed in parallel distinct computational units, referred to as computational domains (CDs). In VLIW architectures, the scheduling of the operations is fully performed at compile time; the compiler is responsible for allocating the execution of each instruction to a specific FU.

we focused on a specific issue which must be faced when testing a VLIW processor: the register file characteristics are different than in other processors, since it must be accessed from different domains. In the same paper, we proposed an effective solution to the test of VLIW register files.

In this paper, we focus on the generation of effective SBST test programs for the whole VLIW processor, characterized by minimal duration, minimal size, and maximal fault coverage. The proposed method starts from existing functional test algorithms developed for each single FU type embedded into the processor (e.g., ALUs, adders, multipliers, and memory units). Although the characteristic of the FUs used within a VLIW processor are similar to those used in traditional processors, generating optimized code to effectively test these units is not a trivial task. In fact, by exploiting the intrinsic parallelism of VLIW processors, it is theoretically possible to reduce the increase in duration and size of the test programs when the VLIW processor size grows. For example, testing the ALUs in the different CDs can be performed in parallel, forcing them to perform the required computations in the same clock cycle, provided that a sufficient number of registers are available, and an effective method to check the results is devised. However, generating an optimized test program with minimal size and duration may require a significant manual effort, taking into account both the test algorithms for each FU, and the specific VLIW processor configuration; our test generation procedure provides an automatic solution for test program generation, once the processor configuration and the test algorithms for each FU are known.

VLIW processors do not include any specially designed hardware module (as it happens for other processor types), but are rather based on a combination of common FUs. Exploiting this characteristic, our solution allows test program generation and optimization to be performed autonomously and automatically, without any manual effort.

The test programs generated by the proposed method are highly optimized and exploit the VLIW processor features in order to minimize the test time and the test program size. Moreover, since the method is totally functional, it does not require the usage of any ATPG tool, nor the adoption of any design for testability (DfT) technique.

In principle, the scheduling technique we propose is based on the same approach typically used by the VLIW compilers for optimization purposes. However, the use of a compiler is not feasible for optimizing a test program; in fact, in our case, the optimization should maintain unchanged the fault coverage of the original test program, while a compiler typi- cally optimizes the code by analyzing the function performed by the code (which in the case of a test program is meaning- less) and selecting the most suitable resources to be used at each time step. For example, a typical VLIW compiler tries to optimize the parallelism of the instructions exploiting the VLIW resources without any external constraints; in our case, we are considering test programs and thus the instructions composing each

piece of code have to be executed in a specific CD and cannot be moved from one to another without modifying the corresponding fault coverage.

The results we achieved clearly demonstrate the effectiveness of our approach on three dif- ferent VLIW configurations. The required test time for the 4, 6, and 8 CDs configurations of the ρ-VEX processordecreased of about 54%, 56%, and 59% with respect to the corresponding nonoptimized solution, respectively; consider- ing, instead, the size of the test programs, the reductions are58%, 60%, and 63%, respectively. The reached fault coverage, for all the processor configuration is about 98%.

This paper is organized as follows. Section II provides ageneral background on VLIW architectures. Section III describes the works based on SBST techniques specifically oriented to VLIW processors. Section IV explains the flow diagram and the proposed method. The experimental results are gathered and are presented in Section V, while conclusions and future works are finally described in Section VI.

II. VLIW BACKGROUND

A VLIW processor is characterized by the operations that are executed by parallel CDs which are characterized by its own FUs as shown in Fig.1.The assembly code for a VLIW processor is different from the machine code of a superscalar processor: instructions are grouped together into a single macro-instruction (also called Bundle) and each instruction is assigned to a specific CD for execution. In the vlsi processor there is no hardware scheduler and the operation was performed by the compiler, thereby reducing the power consumption and area coverage is less than that of traditional superscalar processors. The instruction level parallelism can be sufficiently exploited, since a good compiler is able to execute the whole program and optimize it correctly.

The VLIW architectures can be easily customized to perform any given application efficiently. There may be different number of CDs and FUs in generic VLIW processor architecture such as type of FUs number of multiported registers and type of accessible FUs. The features of VLIW processor are grouped together and listed in VLIW manifest. The manifest specifies the features such as number of CDs, number and type of FUS embedded into each CD,size and other features during the development of code.

The VLIW code is composed of a sequence of bundles, each of which contains a number of instructions equal to the number of CDs composing the VLIW where each instruction is responsible for execution. The VLIW code is more complex than a traditional assembly code, and the size of the code is larger. The mapping of instruction is performed entirely at compile time. It is up to the compiler to include the most suitable instruction in each bundle and guaranteed that they can be executed in parallel by the different CDs.

By considering the particular architecture of the VLIW processor, several methods and solutions were proposed to produce the assembly code suitable for that kind of processors. While generating the code for a VLIW processor, the programmer or the compiler is faced the issue of schedulimg independent operations and extracting parallelism from a sequential code concurrently, to the embedded FUs. For this reason, the scheduling algorithms are critical for the performance of a VLIW processor. For the first-generation of VLIW processors many compilers used a three phases method to generate code: first they generate a sequential program, then they analyze each basic block in the sequential program looking for independent operations, and they finally schedule independent operations within the same block in parallel. In many cases the instructions in a basic block are dependent on each other; hence, insufficient ILP may be available within a single basic block, which is the main problem of this approach. If VLIW processors are considered, the trace scheduling is the most important scheduling algorithm, where it is a profile driven method where the whole trace is scheduled together.

III. RELATED WORK

The most popular testing techniques for processor chips and socs are sbst and built-in self-test (bist). Some methods may require very expensive automatic test equipment (ate) while the increasing difference between device-operating frequencies and maximum ate frequencies may results in at speed testing expensive and complex. The failures are detectable only when the test is performed at the operating frequency of the device, said to be at speed testing.

In bist testing task from external resources (ate) are moved to internal hardware and additional hardware is integrated into the circuit to allow it to perform self-testing. By using this technique the timing for testing and maintaining are decreased and the fault coverage is increased.

SBST is a special type of on-chip testing, that is a nonintrusive methodology, since it adopts existing processor instructions and resources to perform self-testing. An important advantage of this technology is that it uses only functionality of the processor and its instruction set for both test pattern application and output data evaluation, thus it does not introduce any hardware overhead in the design. Even though, sbst methods may suffer from long program sequences to get high fault coverage, and may require effective techniques for generating suitable test programs.

In the literature, there are many papers related to functional self-test of processors, but only a few of them refer to the test of vliwprocessors .

we presented a novel sbst algorithm aimed at testing the register file of a generic vliw processor. The register file of a vliw processor has the architecture with multiports, since this component must be accessed by all the cds. At the same time, a single cd can access each register both in reading and writing by the use of read ports and write ports . This means that the architecture of register file of a genericvliw processor has the complex cross-bar. By considering this structure, we developed a new sbst algorithm able to achieve high fault coverage. The obtained experimental results show that this algorithm achieves up to 97.12% of fault coverage with respect to stuck-at faults. The method proposed in was developed addressing stuck-at faults, but it is necessary to deal with different fault models, such as transition delay faults.

Variant technique aimed at testing vliw processors in order to obtain a good diagnostic resolution with a low hardware overhead, is proposed in .the characteristic of that approach is aimed at detecting faults in the functional units of the processor, is that the same test patterns are loaded into the fetch registers directly, of all cds. The adequate effect of each domain is tested by comparing the test response of all domains, which should be like in the fault-free case. This solution involves a hardware overhead of about 5% and requires that the processor runs in a special self-test mode.

In goal and vierhaus propose a built-in self-repair strategy based on sbst for vliw processors; this approach is able to detect faults and identify the most convenient configuration able to recover them, if faults are located in components. By considering the sbst technique, the purpose of that approach is to use the isa in order to apply the test patterns generated off-line by an atpg for the units embedded into the processor.

A software methodology is proposed for vliw processors based on the execution of each operation twice on two different fus for detecting faults in computational resources and checking the adjacent results through control instructions.

We first proposed a method for generating sbst programs for a full vliw processor, starting from existing test algorithms developed for traditional processors. In deep, the method addresses the functional units (such as the alus and the muls), embedded into a vliw processors and explains to

embedded them for the whole processor in a single test program. However, the aim of the work was not the optimization of the sbst programs in terms of test duration and size.

IV. PROPOSED METHOD

The method starts from existing test programs based on generic SBST algorithms and automatically generates effective test programs able to reach the same fault coverage, while minimizing the test duration and the test code size. The method consists of four parametric phases and can deal with different VLIW processor models. The main goal is to show that in the case of VLIW processors, it is possible to automatically generate an effective test program able to achieve high fault coverage with minimal test time and required resources. Experimental data gathered on a case study demonstrate the effectiveness of this approach; results show that this method is able to exploit the intrinsic parallelism of the VLIW processor, taming the growth in size, and duration of the test program when the processor size grows.

VLIW architectures can be easily customized to efficiently perform any given application. In fact, a generic VLIW processor parametric architecture may have a variable number of CDs and FUs, so that different options, such as the number and type of FUs, the number of multiported 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 to best fit the application requirements . All the features of a VLIW processor are grouped together and listed in the so-called VLIW manifest. The manifest specifies the number of CDs, the number and type of FUs embedded into each CD, the size and the access mode of the register file, and any other feature that must be taken into account when developing the code, such as the memory size and the memory access mode. Moreover, this file also contains the description of the instruction set architecture (ISA) of the considered VLIW processor, which is clearly crucial in order to write the corresponding assembly code.

VLIW processor, several solutions were proposed in order to produce the assembly code suitable for this kind of processors. When generating code for a VLIW processor, the programmer or the compiler is faced with the issue of extracting parallelism from a sequential code and scheduling independent operations, concurrently, to the embedded FUs. For this reason, the scheduling algorithms are critical to the performance of a VLIW processor. Many compilers for the first-generation of VLIW processors used a three phases method to generate code: first of all they generate a sequential program, then they analyze each basic block (a basic block is a sequence of instructions with a single entry point and a single exit point) in the sequential program looking for independent operations, and finally they schedule independent operations within the same block in parallel.

The main problem of this approach is that in many cases the instructions in a basic block are dependent on each other; hence, insufficient ILP may be available within a single basic block, especially considering the large number of parallel resources of a typical VLIW processor. The trace scheduling is the most important scheduling algorithm, if VLIW processors are considered; it is a profile driven method where a set of commonly executed sequence of basic blocks embedded in the control flow is gathered together into a trace and the whole trace is scheduled together. In this way, the probability of assigning an operation to each FU increases since in a trace the possibility to find instructions that can be executed together at the same clock cycle is greater than when considering a single basic block.

The execution flow is based on four main steps: the fragmentation,the customization, the selection, and the scheduling.

A. Fragmentation

The purpose of the fragmentation phase is to minimize the number of test operations in order to create efficient and optimized test programs. The fragmentation phase, illustrated Step A, performs two main tasks. The first is the selection from the library of the test programs needed to test the VLIW processor under test, ignoring those which refer to FUs that are not part of the processor itself. The second task performed by this step is to fragment each selected test program into a set of small pieces of code, called fragments, containing few test operations and the other instructions needed to perform an independent test. The result of the fragmentation phase is a set of unique test fragments, where each fragment is normally built around a single test instruction and includes some preliminary instructions, required to correctly perform it, and some additional instructions to forward the produced results into observable locations; a fragment is described through architecture-independent code. A test program is typically composed of a set of test operations enclosed in a loop; thefragmentation phase simply separates them in a series of short test programs using the loop unrolling technique.

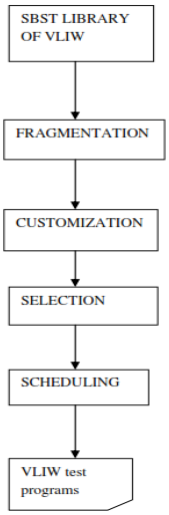

Fig 1 flow diagram

B. Customization

The customization step, illustrated in Step B, is responsible for the translation of the generic architecture independent test programs into the VLIW code, exploitingthe ISA of the considered processor. In particular, starting from the fragments library and from the VLIW manifest, the method translates each generic fragment in a custom fragment that can be executed by the processor under test. A custom fragment is defined as a set of instructions belonging to the ISA of the processor under test, which perform several operations in order to test the addressed FU. An example of the customization process, where the code of a Fragment before and after the customization phase is reported.

C. Selection of the Custom Fragments

The selection of the custom fragments, illustrated in Step C, consists in the choice of the test fragments that optimize a set of rules dependent on the requirements desired for the final SBST program. The optimization is performed by the execution of the algorithm described; the algorithm is able to implement two alternative rules. The former aims at selecting the minimum number of custom fragments that allow to reach the maximum fault coverage with respect to all the resources of the processor under test. During this phase, all the fragments are filtered depending on their fault coverage on the full VLIW processor. The filteringof the fragments is performed by the execution of multiple algorithm iterations. At each iteration, the algorithm adds to the selected fragment list the fragments that maximize the fault coverage with respect to all the resources of the processor under test. In this way, at the end of the execution, several custom fragments are not selected, since the faults covered by these fragments are already covered by the fragments chosen by the algorithm.

D. **Scheduling**

The scheduling phase first elaborates the selected custom fragments obtained from the selection phase. This process is responsible for the integration of the custom fragments in order to obtain an optimized and efficient final test program. In order to reach this goal, we developed a scheduler that optimizes and merges the codes contained into the custom fragments exploiting the VLIW features; in particular, it compacts the test programs trying to maximize the ILP of the VLIW processor by an optimal usage of the parallel CDs. In order to optimize the execution of the test instructions composing the custom fragments, we developed a new scheduling algorithm based on the trace scheduling algorithm. The developed scheduling algorithm aims at optimizing the execution of the testing code in a generic VLIW architecture, taking into account the possibility of computing several instructions in a single clock cycle while maintaining the fault coverage capabilities of the compacted code unaltered with respect to the generated custom fragments. Our solution organizes the code belonging to the custom fragments in traces, which are loop- free sequences of basic blocks (a basic block is a sequence of instructions with a single entry point and a single exit point) and then squeezes a trace into few VLIW instructions. The scheduling algorithm we developed is restricted with respect to the original version of the trace scheduling algorithm, since in our test code (which is composed of several custom fragments that must be performed only once), we neglected the loop management.

First of all, the selected custom fragments are analyzed, looking for data dependencies among the instructions. For each fragment an instruction dependency graph(IDG)is created. More in details, anode exists in the IDG for each instruction in the fragment code, while each edge between two nodes corresponds to a data dependency between the corresponding instructions. Hence, each edge corresponds to a physical resource(i.e., a register or memory location)used to store the data produced by one instruction and used by the other. During this phase, it is possible that two or more instructions, belonging to different custom fragments and related to the same CD, are identified as operations that perform the same job(e.g., they write the same value into the same register) if this behavior is detected, a unique IDG will be defined for the considered custom fragments, where only one of these microinstructions will be considered, while the others will be neglected in this way the code functionality of the custom fragments remains unchanged, while the number of instructions is reduced.

V. EXPERIMENTAL RESULTS

In this section, we present the experimental results obtained to validate the proposed method. For this purpose, we selected the ρ-VEX VLIW processor as a case study.

The ρ-VEX processor is a generic and reconfigurableVLIW processor whose VHDL description has been released by researchers from delft university of technology it includes most of the features of the VLIW processors used by industry. The processor standard configuration consists of a four stages pipeline organization: fetch, decode, execute, and write-back. The fetch unit fetches a VLIW macro-instruction from the attached instruction memory and splits it into micro-instructions, that are passed, in parallel, to the decode unit. In this step, the decoding operations are performed and the registers used as operands are fetched from the register file. The micro-instructions are then forwarded to the parallel execution units.For each execution unit there is an ALU unit (A); within the second and the third execution units there is also a MUL unit (M).

In order to prove the effectiveness of the proposed method, we developed three different configurations of the ρ-VEX VLIW processor. The key difference between these versions is the number of CDs composing the processor itself, which was varied from 4 to 8. Each of these processor configurations are described in a manifest, corresponding to a ASCII file containing the features of the resources and the ISA of the processor. We generated one test program for each of these configurations applying the method previously described using a prototypical tool we developed, composed of about 3000 lines of C code.

For testing the FUs embedded in the processor. Each of them was encoded in architecture-independent pseudo-code and was inserted in the starting library. The total time required to manually prepare the three input files required by the method was approximately 30 h. At the end of the fragmentation step, we obtained a fragments library composed of 520 architecture- independent fragments.

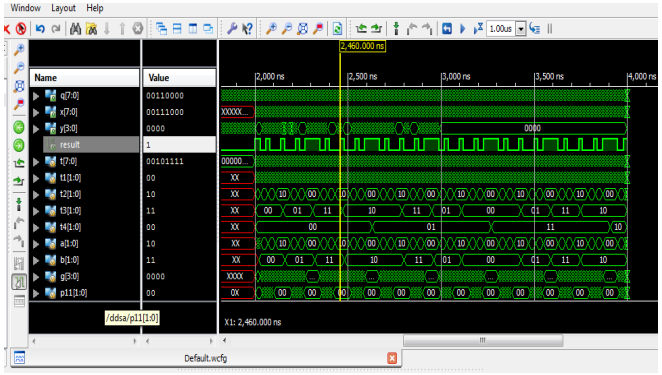

Fig 2 Output result with fault coverage

The results obtained adding two further CDs (1 ALU and 1 MUL) are reported; finallywe present the results related to the ρ-VEX processor composed of eight CDs, where the number of embedded ALUs has been increased to eight and that of MULs to four. As the reader can notice, the reached fault coverage is almost the same for the three configurations of the processor.

The details about the test programs generated through the proposed method and used to test the ρ-VEX processor are reported in , where the required number of clock cycles, the reached coverage, and the size of the test pro- grams are shown. The generation time for each test program was approximately 40 h, of which about 95% used for the fault simulation of the custom fragments.

We also compared the test program (denoted as TP) generated by our approach with a test program (denoted as Plain TP) consisting of several test programs developed using some algorithms taken from the literature for the FUs of traditional processors. In the plain TP, these test programs have simply been queued in a unique test program, without performing any selection or scheduling steps, therefore adopting a realistic test estimation of what can be achieved with previously developed test algorithms

without any optimization method. This is the only possible approach to have a comparison for the proposed method. To the best of our knowledge in the literature, there is no method aimed at optimizing the SBST routines for VLIW processors exploiting the parallelism that characterizes these architectures. In order to fairly evaluate the two solutions, these test programs have been applied using the loop-unrolling technique, as it is common for any VLIW applicaton.

Table1: comparison of existing and proposed approach

Partition	OptimizedSBST	ProposedApproach
RegisterFile	62.82%	87.17%
CD 0	77.12%	83.74%
CD 1	80.12%	88.39%
CD 2	79.99%	88.23%
CD 3	70.80%	81.65%

Moreover, the user's effort is minimized, since it is enough to specify the processor features in the manifest in order to generate the appropriate test program for the generic VLIW processor. Finally, the advantage of our approach is signifi- cantly greater with respect to traditional scan test techniques: in fact, in order to reach 100% of stuck-at fault coverage, traditional scan test technique requires a number of clock cycles greater than three orders of magnitude on the average, with respect to our solution, for all the three configurations of the considered VLIW.

V. CONCLUSIONS

In this paper, we presented the first method able to achieve optimized SBST programs in the VLIW processors. The method is fully automatic and the obtained results clearly shows the efficiency of the method that allows to reduce both the number of clock cycles and the memory resources. More in particular, the method shows that it is possible to develop test programs whose duration and size grows less than linearly with the VLIW parallelism.As future work, we plan to better evaluate the performances of the proposed solution with the use of other VLIW models with different functional units.

REFERENCES

[1] M.Psarakis,D.Gizopoulos,E.Sanchez,andM.SonzaReorda, "Microprocessorsoftware-basedself-testing,"IEEE Design TestCom- put.,vol.2,no.3,pp.4–19,May–Jun.2010.
[2] J.A.Fisher,P.Faraboschi,andC.Young,EmbeddedComputing: AVliwApproachtoArchitecture,CompilersandTools.Berkeley,CA,USA: Univ.CaliforniaPress,Dec.2004.
[3] S.Wong,T.VanAs,andG.Brown,"ρ-VEX:AreconfigurableandextensiblesoftcoreVLIW processor," inProc.Int.Conf.ICECE Tech- nol.,Dec.2010,pp.369–372.
[4] Hewlett-Packard Laboratories. EX Toolchain[Online]. Available:http://www.hpl.hp.com/downloads/vex/
[5] J.Fischer,"VerylonginstructionworkarchitecturesandtheELI-512," IEEESolid,StateCircuitsMag.,vol.1,no.2,pp.23–33,Sep.2009. J. A. Fischer,"Trace scheduling: A technique for global microcodecompaction," IEEE Trans. Comput., vol. 30, no. 7, pp. 478–490, Jul.1981.
[6] N.Kranitis,A.Paschalis,D.Gizopoulos, andG.Xenoulis,"Software- based self-testing ofembedded processors," IEEE Trans. Comput.,vol.54,no.4,pp.461–475,Apr.2005.
[7] T.KoalandH.T.Vierhaus,"Asoftware-basedself-testandhardware reconfiguration solutionforVLIWprocessors,"inProc.IEEESymp. DesignDiag.Electron.CircuitsSyst.,Apr.2010,pp.40–43.
[8] M.Ulbricht,M.Scholzel,T.Koal,andH.T.Vierhaus,"Anewhier- archicalbuilt-inself-testwithon-chipdiagnosisforVLIWprocessors," inProc.IEEESymp.DesignDiag.Electron.CircuitsSyst.,Apr.2011, pp.143–146.
[9] Pillai,W.Zhang,andD.Kagaris,"DetectingVLIW harderrorscosteffectivelythroughasoftware-basedapproach,"inProc. Adv.Inf.Netw. Appl.Workshops,May2007,pp.811–815.
[10] Bolchini"Asoftwaremethodologyfordetecting hardwarefaultsinVLIWdatapaths,"IEEETrans.Rel.,vol.52,no.4,pp.458–468, Dec.2003.

