Automatic Generation of Test Codes To Detect Permanent Faults in VLIW Processor

Kalitha Nashkath. M
PG Scholar, ECE, SBM CET,
Dindigul, Tamil Nadu, India.

Manikandan. V
Assistant Professor, ECE Dept., SBM CET,
Dindigul, Tamil Nadu, India.

***Abstract*— Very long instruction word (VLIW) processors have many advantages such as high performance with reduced clock rate and low power consumption, due to this quality they are widely used in embedded signal processing applications. Similarly there is an increased need for efficient techniques to detect permanent faults due to aging or during the manufacturing process in VLIW processors. Software-based Self-Test (SBST) methods are considered as an effective solution for detecting faults in processors. If the traditional SBST technique is applied to the VLIW processor it might prove to be inefficient mainly due to its size, duration and the inability of the technique to exploit parallelism of these architectures. Here we present a new method for the automatic generation of efficient test programs especially oriented for VLIW processors. This method begins from extending the test programs based on characteristic of SBST algorithms and automatically generates effective test programs to reach maximum fault coverage, minimum test code size and duration. The main aim is to show that it is possible to generate effective test program. In addition to this we use a technique called tiling technique, by using this technique it is possible to avoid the place where the error has occurred by the testing controller thus the unwanted generation of test pattern for the faulty zone can be avoided**

Keywords*— *Very long instruction word (VLIW) processors; Software-Based Self-Test (SBST)

I. INTRODUCTION

Mainly due to 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 favorable 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, with the produced result we can detect possible faults. 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. Currently VLIW processors are adopted in several products, in particular to embedded applications, and the problem of testing them is progressively relevant.

VLIW processors are characterized by a pipelined architecture with multiple functional units. Unlike super scalar processors, VLIW processors do not include any significant control logic since instruction scheduling is completely performed by the compiler. This involves 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 micro- instructions) into one large macroinstruction (also called bundle), where each microinstruction 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.

A few SBST approaches were proposed in the literature in order to properly test VLIW processors against permanent faults. Some of them rely on suitable instructions belonging to the processor instruction set to apply the test patterns previously generated by a conventional automatic test pattern

generator (ATPG) targeting each internal component. Although effective, these methods have several drawbacks. First of all, transforming the test patterns generated by the ATPG into test programs is not always straightforward. Secondly, the resulting test programs are far from being optimized, especially in terms of test length, and finally, the attainable fault coverage is not always as high as it may be required. 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 size, minimal duration, 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 generates optimized code to effectively test these units is not a trivial task. In fact by exploiting the essential 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 (e.g) testing the ALUs in the different CDs can be performed in parallel, forcing them to perform the required computations in the same clock cycle provides 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. Considering this 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), it is entirely based on a combination of common FUs. Exploiting this characteristic, our solution allows to generate and optimize test 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 typically 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. More in general, the test of a specific unit in a VLIW processor requires performing a well-defined sequence of instructions in a well- defined CD. If we encode the test program in a high-level language and then launch the compiler, we do not have any way for forcing it to generate a code, which executes the given sequence of instructions on the FUs of a given CD. Consequently, it is not possible to use a VLIW compiler to generate the machine code for testing the processor, nor to use it to optimize the test code.

The proposed method was experimentally evaluated on a VLIW platform based on the Delft University ρ-VEX VLIW processor which supports most of the features of industrial VLIW architectures. The results we achieved clearly demonstrate the effectiveness of our approach on three different VLIW configurations. The required test time taken for the 4, 6, and 8 CDs configurations of the ρ-VEX processor decreased of about 54%, 56%, and 59% with respect to the corresponding non optimized solution, respectively; considering, instead, the size of the test programs, the reductions are 58%, 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 a general overview on VLIW architectures and on the most important scheduling techniques. Section III describes the related works on SBST techniques specifically oriented to VLIW processors. Section IV explains the flow and the details of the proposed method. The experimental analysis we performed and the gathered results are presented in Section V, while conclusions and future works are finally described in Section VI.

II. RELATED WORK

In this paper we proposed on the generation of effective SBST test program for the VLIW processors, characterized by maximal fault coverage, minimal duration and size.

This method starts from existing functional test algorithm developed for each single FU type embedded into the processor. Although the characteristic of the fuse used within a VLIW processor are similar to those used in traditional processors to generate optimized code to effectively test these units is not a trivial task.

By utilizing 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.

A. Advantages

- Automatically generates the test program for the VLIW processor.
- Maximal fault coverage.
- Reduce the program code size.
- Reliable method.
- Minimizing the test test code size and duration.
- Automatically generates optimized test program and able to achieve high fault coverage with minimal test time and required resource. Tailing technique will be used to reduce a testing time.

B. Implementation

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.

This method aims at automatically generating the test program for a given VLIW processor starting from the VLIW manifest and from a library of existing SBST programs. The generation of the test program is automatically performed on the basis of the VLIW configuration, and is therefore autonomously tuned depending on the VLIW manifest features. The flow supports some optimizations concerning the execution time, the fault coverage, and the code length, and these optimizations are not correlated to the characteristics of the original test routines. The execution flow is based on four main steps: the fragmentation, the customization, the selection, and finally scheduling.

The flow has three initial inputs, which include two global requirements (the VLIW manifest and the library of generic SBST programs), and a specific input (the SBST program for testing the VLIW register file). The VLIW manifest contains all the features of the processor under test, while the SBST library contains a set of programmable to test the different modules within the processor itself. These two requirements are defined as global since they are configurable depending on the characteristics of the addressed VLIW processor. In particular, the library is a collection of generic SBST programs based on literature test algorithms, it contains the functional test code able to test the most relevant FUs of a generic VLIW processor. The test codes stored into the library are purely functional (i.e., do not require any DFT feature) and are completely independent on any physical implementation of the FU they refer to. These codes may be based on the techniques used to test the same FUs when used in conventional processors. Their description exploits a pseudo-code based on C-language. On the other side, when t he register file is considered, the algorithm can be successfully used, since the structure of a register file of a superscalar processor is very different with respect to the register file of a VLIW processor, especially considering the access mode to the registers. In a typical VLIW processor, the register file is composed of a variable number of read and writes ports, depending on the number of CDs embedded into the processor; consequently a special algorithm must be used in order to reach high fault coverage.

C. 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 and 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. The fragmentation phase simply separates them in a series of short test programs using the loop unrolling technique.

D. Customization

The customization step, illustrated in Step B, is responsible for the translation of the generic architecture independent test programs into the VLIW code, developing the 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.

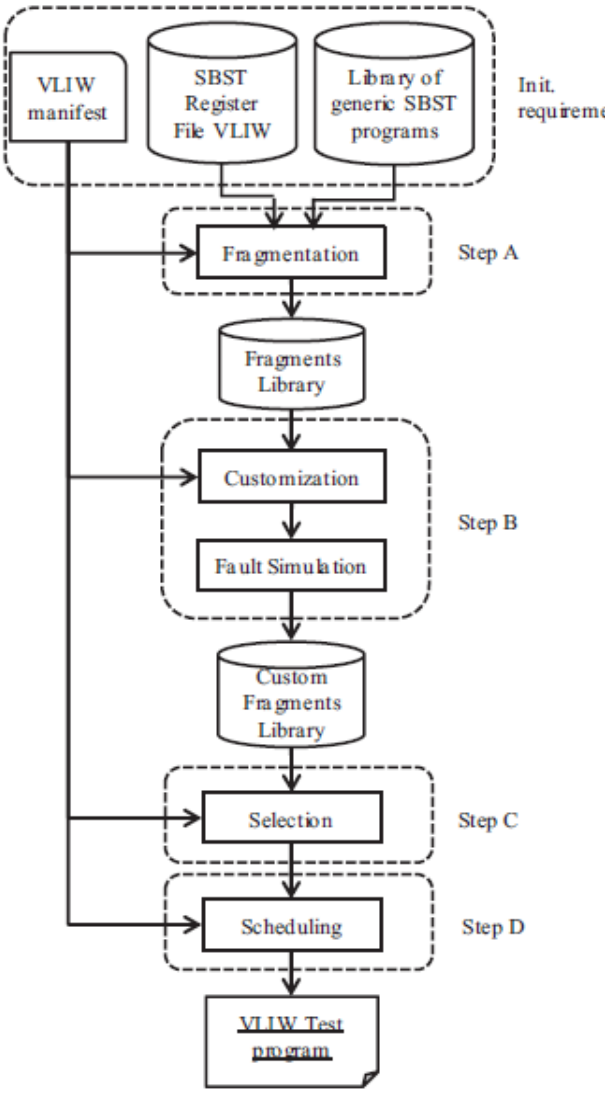
E. Selection of the custom fragments

The selection of the custom fragments, explained 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 filtering of 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.

F. Scheduling Phase

The last step of the proposed automatic test program generation flow is the scheduling, illustrated in Fig. 3 Step D. 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 described in [8] and [9]. 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.

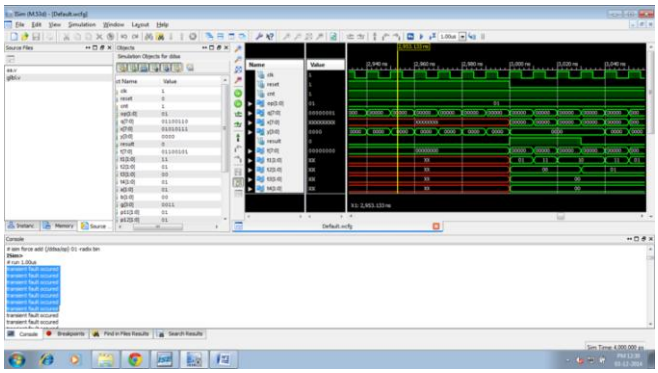


System Flow Diagram

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, a node 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. 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 micro-instructions 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.

Aside from the IDG generated, a table called resource usage table, containing the details of the instructions executed is created. For each instruction, a new entry in the table is instantiated, containing the instruction ID and the identifier of the CD where the instruction must be executed; finally each entry has a priority field, which is initialized to 1.

III. EXPERIMENTAL RESULT



IV. CONCLUSION

In this paper, we presented the first method able to generate optimized SBST programs for VLIW processors by exploiting the intrinsic parallelism existing in the VLIW processors. The method is fully automatic and allows to drastically reducing the effort of generating test programs for VLIW processors. The method has been experimentally validated resorting to a representative VLIW processor and generating test programs using a prototypical tool. The obtained results clearly demonstrate the efficiency of the method that allows reducing significantly both the number of clock cycles and the memory resources with respect to test programs generated by applying generic SBST methods to the specific case of the VLIW processors. 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.

REFERENCES

1. M. Psarakis, D. Gizopoulos, E. Sanchez, and M. Sonza Reorda, “Microprocessor software-based self-testing,” IEEE Design Test Comput., vol. 2, no. 3, pp. 4–19, May–Jun. 2010.
2. J. A. Fisher, P. Faraboschi, and C. Young, Embedded Computing: A Vliw Approach to Architecture, Compilers and Tools. Berkeley, CA, USA: Univ. California Press, Dec. 2004.
3. M. Beardo, F. Bruschi, F. Ferrandi, and D. Sciuto, “An approach to functional testing of VLIWarchitectures,” in Proc. IEEE IEEE Int. High-Level Design Validation Test Workshop, Jul. 2000, pp. 29–33.
4. D. Sabena, M. Sonza Reorda, and L. Sterpone, “A new SBST algorithm for testing the register file of VLIW processors,” in Proc. IEEE Int. Conf. Design, Autom. Test Eur., Mar. 2012, pp. 412–417.
5. S. Wong, F. Anjam, and F. Nadeem, “Dynamically reconfigurable register file for a softcore VLIW processor,” in Proc. IEEE Int. Conf.Design, Autom. Test Eur., Mar. 2010, pp. 962–972.
6. S. Wong, T. Van As, and G. Brown, “ρ-VEX: A reconfigurable and extensible softcore VLIW processor,” in Proc. Int. Conf. ICECE Technol., Dec. 2010, pp. 369–372.
7. Hewlett-Packard Laboratories. EX Toolchain [Online]. Available: http://www.hpl.hp.com/downloads/vex/
8. J. Fischer, “Very long instruction work architectures and the ELI-512,” IEEE Solid, State Circuits Mag., vol. 1, no. 2, pp. 23–33, Sep. 2009.
9. J. A. Fischer, “Trace scheduling: A technique for global microcode compaction,” IEEE Trans. Comput., vol. 30, no. 7, pp. 478–490, Jul. 1981.
10. N. Kranitis, A. Paschalis, D. Gizopoulos, and G. Xenoulis, “Softwarebased self-testing of embedded processors,” IEEE Trans. Comput., vol. 54, no. 4, pp. 461–475, Apr. 2005