VIRTUAL-SCAN : A NOVEL APPROACH FOR SOFTWARE-BASED SELF-TESTING OF MICROPROCESSORS

Giorgos Dimitrakopoulos[†] *, Xrisovalantis Kavousianos*[‡] *, and Dimitris Nikolos*[†§]

[†]Computer Engineering and Informatics Dept., University of Patras, 26500 Patras, Greece
[‡]Dept. of Computer Science & Technology, Univ. of Peloponnese, 22100, Tripoli, Greece
[§]Computer Technology Institute, 3 Kolokotroni Str., 26221 Patras, Greece

ABSTRACT

A systematic methodology for generating software-based self-tests for microprocessor cores is introduced in this paper. The produced software tests emulate the functionality of a scan path design, and can be applied during the normal-operation mode of the microprocessor, thus enabling at-speed testing. A major advantage of the proposed approach lies in the fact that the generation of the software tests does not require any knowledge about the low-level implementation of the microprocessor and is only based on its RT-level description and its instruction set architecture.

1. INTRODUCTION

Modern microprocessors impose significant challenges to the test community, due to their high complexity and heterogeneity. The gap between the operating frequencies of microprocessors and that of the Automatic Test Equipment (ATE) is steadily increasing, a fact that leads to the escape of failures, which can only be detected, when testing is applied *at-speed*.

One of the most promising solutions to at-speed testing is the application of self-testing techniques such as Built-In Self-Test (BIST). In BIST both test pattern generation and response monitoring and evaluation are performed on-chip by dedicated hardware modules such as Linear-Feedback Shift Registers (LFSRs). The application of BIST can significantly reduce the design and test generation time improving in this way the time-to-market of the final product. Nevertheless, in order for BIST to be able to generate high quality tests, certain modifications should be applied to the microprocessor-under-test, such as test-point insertion, which significantly increase the circuit area and may lead to performance degradation.

An alternative to hardware-based self testing is software-based self testing, which involves the testing of a microprocessor using its instruction set. The main benefit of software-based self-test is that it can be applied in the normal operation mode of the microprocessor, thus applying the required tests at-speed. Additionally, software-based self-testing does not require any design changes neither the insertion of any additional test hardware, and its efficiency has been proven both for stuck-at [1]–[3] and delay faults [4].

Recently, several methods for generating software tests that target structural faults of the microprocessor were presented [1]–[3]. In [1] the application of random instruction sequences with randomly selected operands was investigated, but the obtained fault coverage remained low, due to the lack of information concerning the internal structure of the microprocessor. In [2] a method to construct software tests composed of predetermined and manually-generated test macros was proposed. On the contrary, in [3] a method which relied both on random and deterministic test data was presented. The generated instruction sequences are very effective but their generation relies on detailed structural information of the microprocessor-under-test and cannot be applied to a general design in a straightforward manner.

In this paper a new methodology for software test program generation is introduced. The approach followed for the generation of software-based structural tests does not require any low-level implementation information of the microprocessor, as needed in [3], and it is only based on its RT-level description and its instruction set architecture. The software-tests generated according to the proposed methodology *emulate* the functionality of a scan-chain architecture to the microprocessor-under-test. In order software tests to be generated the microprocessor is enhanced with a virtual scan chain. During test time the software tests are applied in normal operation mode, emulating the virtual scan design. The test responses (registers' values) are stored to the memory from where they can be later retrieved by a slow-speed ATE.

The rest of the paper is organized as follows. Section 2 introduces the proposed methodology for generating

[†] This work has been supported by GiGA Hellas S.A. an Intel Company.

[‡] This work has been supported by the State Scholarships' Foundation of Greece via its Post-doctoral research scholarship programs.

0-7803-7762-1/03/$17.00 ©2003 IEEE

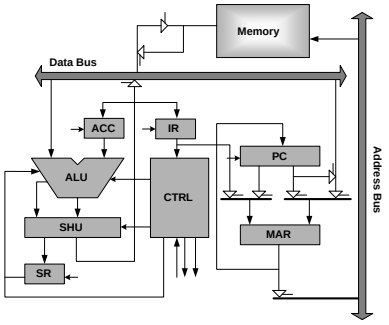

Figure 1: PARWAN Microprocessor Core

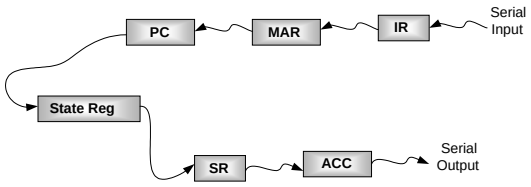

Figure 2: The virtual scan chain of PARWAN.

software tests based only on a RT-level description (VHDL) of the microprocessor. In Section 3 the proposed test generation and validation flow is presented while in Section 4 experimental results are discussed. Finally, conclusions are drawn in Section 5.

2. THE PROPOSED METHODOLOGY

The main goal of the proposed software-test generation methodology is to generate software tests that can be applied at-speed, based only on a RT-level description of the microprocessor and its instruction set. In particular, we are only interested in extracting from the microprocessor's RTL description, the nominal values assumed by its registers during normal-operation mode.

In order to present our approach a simple accumulator-based microprocessor will be used, named PARWAN [5]. The architecture of PARWAN includes the following components, as shown in Fig. 1: An arithmetic logic unit (ALU), an accumulator (ACC), a program counter unit (PC), a memory address register (MAR), a shifter unit (SHU), a status register unit (SR), and a controller (CTRL), which controls the functionality of all the aforementioned components and is implemented as a Finite-State Machine (FSM). Additionally it consists of an 8-bit data bus and 12-bit address bus that covers a total of 4KBytes of memory space and it is divided into 16 segments.

The scan chain design-for-testability technique connects the registers of the microprocessor-under-test to a single shift register. In this way the the values of the registers of the circuit can be directly controlled and observed through the scan input and output pins respectively, in a bit-serial manner. The insertion of a shan chain to the PARWAN microprocessor would connect its registers according to Fig. 2. It should be noted that except the state register of the controller, all the other flip-flops of the scan chain belong to functional registers of the microprocessor. In this case the Instruction Register (IR), the Program Counter (PC), or the Accumulator (ACC), can be set to specific values via an instruction or a certain sequence of instructions assuming certain values for their operands. For example the value of the ACC can be directly set via a load-accumulator instruction and its contents can be observed by applying a store-accumulator instruction. In a similar manner the contents of the PC can be controlled by a jump instruction to a certain memory address.

Therefore the aim of the proposed methodology is at first to generate test vectors as if the microprocessor employed a scan chain and second to generate the proper sequence of instructions that would emulate the functionality of the scan chain to the *non-scan* microprocessor. Note that the microprocessor does not contain any scan chain. The scan chain insertion takes place so that the constrained test vectors to be derived by the Automatic Test Pattern Generation (ATPG) engine.

2.1. Constrained Test Pattern Generation

In order the generated ATPG vectors to be *realizable* by processor's instructions, the values that appear on the registers of the microprocessor during test application time should correspond to nominal values of the normal-operation mode. For example, the IR can only assume values that correspond to valid opcodes. In a similar manner, the zero (z) and the negative (n) flags of the status register cannot be asserted simultaneously.

The constraints extraction leads to a set of logic equations that the bits of microprocessor's registers should satisfy. For example the constraint that the zero (z) and the negative (n) flags of the status register cannot be asserted simultaneously is translated to to $z \oplus n = 1$, where $\oplus$ denotes the logical exclusive-OR operation. The produced set of logic equations that describe the imposed constraints is given to the ATPG tool in order the constrained test vectors to be generated. Finally, the value assumed by the state register of the controller is chosen to be equal to the encoding bits of the execution state. This constraint is imposed since the execution phase of an instruction resembles better the capture cycle of a scan chain design, than other machine cycles, i.e., instruction fetch.

It should be noted that the imposed constraints

do not rely on detailed gate-level information of the microprocessor but concern only nominal registers' values (during normal operation), and thus can be easily extracted from the RT-level description.

2.2. Test Program Synthesis

For each test vector generated through the constrained ATPG procedure a certain software routine is created, which is called the test-vector application routine. The test-vector application routine is responsible for the emulation of the application of the corresponding test vector to the microprocessor-under-test, as if the microprocessor included a scan chain.

The instructions that constitute each test vector application routine are divided into three groups. The first group sets the proper values to the registers, the second applies the virtual scan test vector, and the third group of instructions writes back to memory the new state of the microprocessor. Proper care is taken in order the values of certain registers not to change until their values are observed, that is their contents are written back to memory in order to be compared with the pre-computed values of the non-fault behaviour.

In order to clarify the generation of a test-vector application routine, assume that the ATPG tool has generated a vector V, which includes the following values for PARWAN's registers: The PC is set to 007H, the IR contains the value 40H, the SR the value 1H, the MAR is set to address 028H, and the ACC contains the value 15H. The software routine that emulates the application of vector V to PARWAN is shown in Fig. 3. The registers' values, as selected by vector V, will appear during the execution cycle of the instruction on line 6, which emulates the capture cycle of a virtual scan chain.

```
1: [028]  int var 0001    // Variable Declaration
            .
            .
            .
2: [001]  lda 1100_0000   // Load Accumulator
3: [003]  asl             // Arithmetic Shift Left
4: [004]  cmc             // Complement Carry
5: [005]  lda 0001_0101   // Load Accumulator
6: [007]  add var         // Add var to ACC
7: [009]  jsr observe     // Jump to Observe routine
```

Figure 3: A sample test-vector application routine.

At first the value of the status register is set to 1H, through the code shown in lines 2–4, which corresponds to a negative flag equal to 1 while the rest flags are equal 0. In the following the value of ACC is set by a load instruction in line 5, while the `add` instruction that appears on line 6 corresponds to the value assumed by the IR, according to test vector V. Additionally, since the value of the PC needs to be equal to 007H, when instruction in line 6 is executed (virtual capture cycle) the software routine starts from address 001H. In a similar manner the data variable (`var`) needed by the `add` instruction of line 6 is placed on address 028H so that the value of MAR during the virtual capture cycle to be equal to 028H, as dictated by vector V. Note that although the value of the ACC can be directly controlled by an `lda` instruction, the values of the PC and MAR are implicitly controlled by the address space selected for storing the instructions and the data of the test-vector application routine.

After the application of each test vector, via its equivalent software routine, an observation routine is called (line 7), which is responsible, according to [3], to store the contents of certain registers, such as the ACC or the SR, back to memory in order to be observed. It should be noted that this observation routine is produced once and is common for all test-vector application routines.

As shown by the previous example, during the test application phase the instruction groups of each test vector lie in specific memory locations. In case that more than one test vectors impose their corresponding instruction sequences to lie in the same memory area, then a conflict is caused. Hence, the conflicting instruction groups are scheduled to different test sessions, i.e., different assembly programs, that are applied separately to the microprocessor-under-test.

3. SOFTWARE-TEST GENERATION FLOW

In this section the proposed software-based test generation flow, shown in Fig.4, will be presented. The main steps of the proposed flow are the following:

A. Virtual Scan Chain Insertion : In this step the gate-level netlist of the microprocessor is enhanced with a scan chain in order the test vectors to be generated. The gate-level netlist is produced by synthesizing the RTL VHDL description.

B. Constraints Extraction : The RT-level description of the microprocessor is analyzed in order the nominal values assumed by its registers to be identified. The output of the constraints extraction procedure is a set of logic equations that can be directly used by the ATPG tool.

C. Test Vectors Generation : This step involves the generation of the constrained test vectors.

D. Test Program Synthesis : The test vectors derived by the ATPG engine are transformed automatically to test-vector application routines, in the form of assembly programs, according to the methodology presented in Section 2.2. In general the sequence of instructions used to emulate the application of one test vector to the microprocessor, assuming a virtual scan path, consists of 5–7 instructions.

In the following the test programs produced by the test-program synthesis procedure are simulated in order

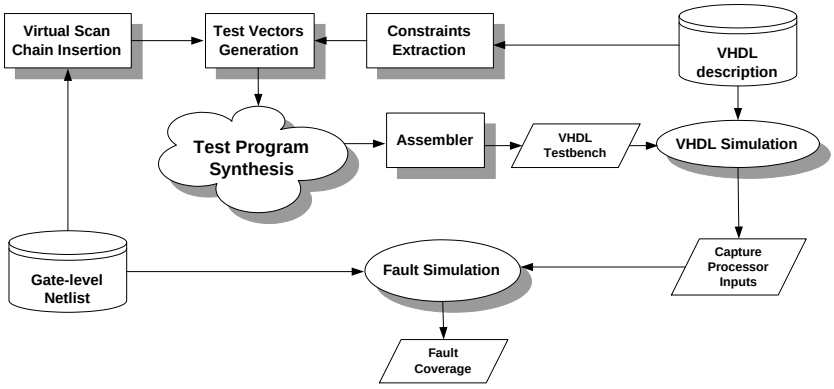

Figure 4: The proposed flow for the generation of software-based test programs.

	ATPG	Deterministic Software	Cycles	Determ. and Random Software	Cycles
Proposed	88.3%	86.7%	1018	87.8%	46692
[3]	—	—	—	91.4%	137649

Table I: The fault coverage obtained by the proposed software-test generation method and the one presented in [3], along with the number of clock cycles needed by each test.

all the values that appear on the primary inputs of the microprocessor to be captured. The captured vectors are used as test stimuli for the fault simulation procedure, which derives the fault coverage obtained by the software tests.

4. EXPERIMENTAL RESULTS

We applied the design flow of Fig. 4 to the PARWAN microprocessor. At first, the RTL VHDL description was synthesized using the Design Compiler tool set of Synopsys. In the following the gate-level netlist was enhanced with a scan chain in order the constrained ATPG vectors to be generated. The ATPG tool used was the FastScan of Mentor Graphics, and the obtained fault coverage is shown in the first column of Table I. In the following the produced constrained test vectors were transformed to software routines that were simulated using the FlexTest of Mentor Graphics.

The final obtained fault coverage is shown in the second column of Table I, along with the needed clock cycles. The fault coverage obtained by the test-vector application routines is very close to that obtained by the ATPG test vectors. The 1.6% difference in the final fault coverage obtained by the proposed software tests and the ATPG test vectors is due to the reduced observability of the microprocessors' internal state, when the test is applied during normal-mode operation.

It is evident that the number of clock cycles needed by the proposed approach is impressively less than that of the approach presented in [3]. Therefore, in an attempt to increase the final fault coverage 1000 random test vectors were generated, along with their corresponding software routines. The randomly generated software tests required 45674 additional clock cycles in order to be applied, and resulted in an improved fault coverage equal to 87.8%.

5. CONCLUSIONS

A novel methodology for the generation of software tests for microprocessors is introduced in this paper. The main benefit of the introduced approach is that the only knowledge required for the microprocessor-under-test lies in the architectural level and thus can be very easily extracted by its RT-level description.

6. REFERENCES

[1] K. Bachter and C. Papachristou, "Instruction randomization self-test for processor cores," in *Proc. of IEEE VLSI Test symposium*, May 1999, pp. 34–40.

[2] F. Corno, M. Sonza Reorda G. Squillero, and M. Violante, "On the test of microprocessors IP cores," in *Design Automation and Test in Europe (DATE)*, March 2001, pp. 209–213.

[3] L. Chen and S. Dey, "Software-based self-testing methodology for processor cores," *IEEE Trans. Computer-Aided Design*, vol. 20, no. 3, pp. 369–380, Mar 2001.

[4] W. C. Lai *et al.*, "On testing the path delay faults of a microprocessor using its instruction set.," in *Proc. 18th IEEE VLSI Test symposium*, 2000, pp. 15–20.

[5] Z. Navadi, *VHDL: Analysis and Modeling of Digital Systems*, McGraw-Hill, New York, 1993.