Devising an RT-Level ATPG for μProcessor Cores

F. Corno, G. Cumani, M. Sonza Reorda, G. Squillero
Politecnico di Torino
Dipartimento di Automatica e Informatica
Corso Duca degli Abruzzi 24
10129 Torino, Italy

Abstract

The issue of SOC testing is one of the most crucial in their design and production process. A popular solution for SOCs including microprocessor cores is based on letting them execute a test program, thus implementing a very attracting BIST solution. This paper describes a method for the generation of effective programs for the self-test of a processor starting from its RT-level description. The method can be partially automated, and combines ideas from traditional functional approaches and from the ATPG field. We are preliminary assessing the feasibility and effectiveness of the method by applying it to an 8051 core.

1. Introduction

Current technology allows integrating enormous numbers of transistors, embedding entire systems on single chips, building the so-called *systems-on-a-chip* (SOCs). These SOCs may be fairly elaborated and include complex cells like microprocessors or MPEG encoders-decoders, together with ASICs, peripherals and memories. Many of these complex cells are readily available as Intellectual Property (IP) cores, designed by third parties, and easily includable in designs.

Today, it is a common practice to design, simulate and synthesize SOCs entirely at the RT level. The increasing demands for tools enabling the design of digital circuits at high levels of abstraction already pushed the development of synthesis and simulation technologies. However, not all design activities have already migrated from gate to RT level (or are not yet mature enough to). Despite many efforts in high-level design for testability, testable synthesis and test pattern generation [1], tackling testability at high levels can be still considered an open problem.

Additionally, testability at RT level is particularly critical for IP cores implementing microprocessors, or microcontrollers, since standard ATPG techniques are seldom exploitable. In fact, microprocessor test requires semantically meaningful and syntactically correct sequences of instructions. Moreover, microprocessor internal structure is often based on a sequentially complex *decode and control unit* that decodes instructions and sends the appropriate control signals to a large *data-path*, which may also include hard-to-test elements such as multipliers and dividers. It may be maintained that testing microprocessors cores at the gate level is a challenging task, but automatically testing them from the RT level is still more of a dream than a reality. However, several research proposals in this field already present some promising results: in this paper, we propose an approach based on manual creation of macro-instructions and on automatic simulation-based selection of most promising macros. The approach is fully based on the RT-level VHDL description of the processor and exploits a custom RT-level fault model.

The paper is organized as follows. Section 2 outlines some basic concepts about the adopted test generation strategy as well as about the adopted RT-level fault model. Section 3 presents an overview of the test program generation approach we propose. Section 4 reports some preliminary experimental results assessing the effectiveness of our approach, and Section 5 draws some conclusions.

2. Test Strategy

Traditionally, microprocessors were tested by resorting to functional approaches based on exciting all the functions and resources described in data-sheets [9], but this approach involves a extremely high amount of manual work performed by skilled programmers. Recently, Dey et al. proposed a deterministic method

named DEFUSE [4] to generate test programs able to reach a good Fault Coverage on the ALU of a microprocessor, and to compact the result. The approach is very effective with easily testable parts (e.g., simple ALUs), but shows some limitation when hard-to-test modules, such as Control Units, are addressed. Another approach has been proposed by Batcher and Papachristou [2] that is based on a random sequence of instructions, but it also requires the insertion of additional hardware in the microprocessor under test. Presently, Bose et al. are working on biased random instruction generators for architectural verification and on-line testing of microprocessors [3].

In [7], we proposed an approach that requires only a limited amount of manual work and that is applicable whenever the netlist, although encrypted, is available for simulation. The approach is based on the automatic cultivation of high-level test programs. These test programs are built from a set of generic pre-computed high-level macros, exploiting a greedy search and an evolutionary optimization algorithm. Once the test program is generated, it is fault-simulated against the gate-level netlist.

In detail, the approach was based on three steps:

- Designers, referring to microprocessor data-sheets, list available addressing modes and categorize instructions in classes.
- Designers identify a library of macros, each composed of a short sequence of instructions, and able of testing a part of control unit and data-path. Each macro owns several parameters, corresponding to the operands of the instructions it is composed of.
- The test program attaining a maximal Fault Coverage is generated without further human intervention, choosing macros from the library and setting their parameters.

We are now modifying this approach in order to handle RT-level microprocessor cores. This work presents the first results in this migration and a consistent roadmap for the implementation of a working microprocessor-cores RT-level ATPG.

In [7], while generation did not take into account any low-level structural information, resulting test programs were simulated against the IP-core netlist to compute their exact fault-coverage figures. Thus, in order to fully migrate to the RT level, the first step is to replace this gate-level fault simulation with a reliable RT-level test metric.

We adopt *observability-enhanced statement coverage* [8] and we refine it by using explicit *RT-level single-bit stuck-at's* instead of tags, as present in [5]. An RT-level single-bit stuck-at fault is defined as a single-bit stuck-at in the effect of an RT-level assignment operation: when a fault is present, the affected object (signal or variable target of an assignment statement) loads the correct value, except for one bit that remains stuck to 0 or 1.

The use of explicit RT-level single-bit stuck-at's enhances the resolutive power of the metric, but prevents several optimizations: each possible stuck-at must be simulated individually instead of calculating the delta variations.

3. Test Program Generation

To perform Test Program Generation we need an environment that, starting from the analysis of the VHDL description, selects the best macros and their parameters in order to create a program able to detect the best number of faults.

After generating the Fault List, analyzing the VHDL description according to the *RT-Level single-bit stuck-at* fault model, the faults are injected during the simulation whenever the corresponding statement is executed. All the faults corresponding to a statement which has been executed at least once by the test program are labeled as *executed*.

The injection of the fault forces a given bit to assume the stuck-at value, thus if the value of the bit is always equal to the faulty one the faults is not excited; so not all the executed faults are excited. When an injected fault produces one difference in the behavior of the processor (in terms of produced and observable results) it is marked as *detected*.

As we work on a microcontroller description, we can group faults in two classes:

- detectable independently from macro operands;
- detectable only using a specific set of macro operands.

In this paper, the faults that belong to the first class are called *control-dependent* faults and the ones belonging to the second class are called *data-dependent* faults.

Intuitively, we can say that most of the *control-dependent* faults are located in the Control Unit and in the Instruction Decoder, where the systems decides *how* to elaborate the instruction data. Instead most of the *data-dependent* faults are located where data are elaborated, such as in the Arithmetic and Logical Unit.

The algorithm we propose is based on two phases:

- *control-dependent* fault detection phase
- *data-dependent* fault detection phase.

The detection of *control-dependent* faults is based on the correct selection of the operative code and the addressing mode. The detection of these faults depends on which instructions (i.e., which macros) have been executed by the microprocessor, independently from a specific set of data.

In this phase each macro in the library is simulated with random operands. By means of this procedure the VHDL statements executed during the fault-free simulation of each macro are identified. The macro that executes statements cumulatively corresponding to the highest number of undetected faults is selected. The selected macro is then fault simulated and, if at least one new fault is detected, it is added to the final test program.

When all the macros of the library have been selected and fault simulated, all the macros become selectable again and the second phase starts.

The goal of the second phase is to detect *data-dependent* faults. The coverage of these faults depends on the arguments of each instruction (i.e., macro operands) executed by the microprocessor.

As in the first phase, the instructions executed by each macro of the library are first identified via fault-free simulation. The macro that executes the VHDL statements corresponding with the highest number of undetected faults is selected.

A *hill-climbing* algorithm is then activated, whose goal is to find the values for the macro operands that maximize the number of faults activated by the macro. The *hill-climber* runs until the number of activated faults reaches a given threshold, or the maximum number of iterations has been reached.

For each fault activated by the selected macro, a Genetic Algorithm, detailed in the following, is then executed, whose goal is to find the values for the macro operands that detect the target fault.

If the target fault is detected, the macro is added to the final test program and a fault dropping phase is activated; otherwise, the fault is discarded, to avoid being considered again with this macro.

When all the activated faults have been detected or discarded, the algorithm returns to the *hill-climber* in order to try to activate others faults.

The whole algorithm stops when either the Fault Coverage reaches a given threshold, or all the macros have been selected and fully considered.

4. Experimental Results

In order to practically assess the effectiveness of the proposed approach we implemented an Automatic Test Program Generator System (ATPGS) whose architecture is sketched in Figure 1.

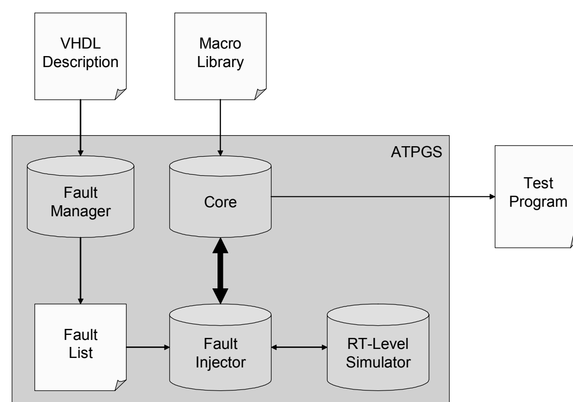

Figure 1: ATPGS Architecture.

The ATPG system amounts to about 11,000 lines of C code including an in-house developed RT-Level Fault Simulator based on a commercial VHDL Simulator (ModelSim 5.5a by Mentor Graphics).

The system has been evaluated on a description of the Intel 8051 microcontroller, containing the core system without peripherals, whose main characteristics are summarized in Table 1.

The Fault Simulator is able to simulate the entire 8051 while it executes the program stored in the embedded ROM, injecting RT-level single bit stuck-at faults in VHDL code. A library of 115 macros is exploited, each composed of a number of instructions that ranges from 3 to 6.

Primary inputs	41
Primary outputs	45
VHDL lines	13,583
Processes	6
Procedures	29
RT-level faults	15,387
Gates	12,134
Flip flops	1,325
Gate-level faults	28,792

Table 1: 8051 description characteristics.

The experiments have been performed on a Sun Enterprise 250 running at 400 MHz and equipped with 2 GBytes of RAM.

The ATPGS is based on using a set of heuristics (i.e., greedy, hill climber and genetic), select the most suitable macros and the values for their parameters to create the test program.

To assess the effectiveness of the technique we propose, we used it to create a test program: the RT-level ATPG was first run, with the goal of maximizing the Fault Coverage based on the RT-level fault model and a test program was obtained. This test program was also simulated at gate-level to obtain gate-level Fault Coverage. Results in Table 2. For comparison purposes, a second set of experiments was then performed: the RT-level description of the 8051 was synthesized and we ran the gate-level ATPG described in [7]. Result in Table 3. The whole procedure adopted for the experiments is outlined in Figure 2.

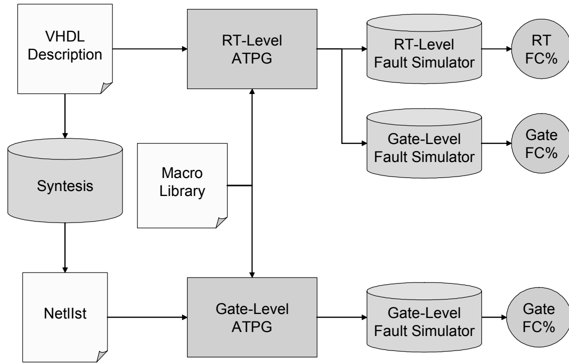

Figure 2: Experimental setup for comparison proposes.

The reported results show that no matter the fact that it exploits the proposed technique provides Fault Coverage figures higher than the gate-level ones, with a slight increase in the length of the final test program (in terms of number of instructions).

Before the proposed method is evaluated in terms of required computational effort, it must be first emphasized that in the current implementation of the tool RT-level Fault Simulation is performed exploiting a commercial VHDL simulator. The interaction with it is necessarily loose, and therefore slow. However, if the method were be integrated in the code of the simulator, a much higher efficiency would be attained. For this reason, we adopted as a parameter the number of 8051 instructions simulated by ATPGS during the test program generation phase. This number is equal to about two million instructions, and roughly corresponds to the number of instructions simulated by the gate-level ATPG described in [7].

RT-level faults [#]	15,387
Executed faults [#]	13,364
Excited faults [#]	12,263
Detected faults [#]	12,122
Test Program Instructions [#]	883
RT-level Fault Coverage [%]	78.78
Gate-level Fault Coverage [%]	89.47

Table 2: Test Program generation from RT-level description.

Gate-level faults [#]	28,792
Detected faults [#]	25,759
Test Program Instructions [#]	624
Gate-level Fault Coverage [%]	85.19

Table 3: Test Program generation from gate-level description.

5. Conclusions

We introduced a new method for generating test programs for microprocessors and microcontrollers. The main novelty of the proposed approach lies in the fact that it only relies on the RT-level description of the device, and does not exploit any knowledge about lower-level implementation details. The method requires the availability of a small library of macros, whose development should be performed by hand, based on the mere knowledge of the instruction set. An optimization algorithm is outlined for selecting the minimal subset of macros, and their parameters. The algorithm entirely works on the RT-level description, and therefore exploits a suitable RT-level fault model.

Experimental results gathered on the Intel 8051 microcontroller using a prototypical implementation of the method show that the generated test program attains higher fault coverage figures (in terms of gate-level stuck-at faults) than the test program generated starting from the gate-level description, thus demonstrating the practical viability of the approach.

6. References

[1] *High Time for High-Level Test Generation*, Panel at ITC99: International Test Conference, 1999

[2] K. Batcher, C. Papachristou, "Instruction Randomization Self Test For Processor Cores", *Proceedings IEEE VLSI Test Symposium*, 1999

[3] M. Bode, E. M. Rudnick, M. Abadir, "Automatic Bias Generation using Pipeline Instruction State Coverage for Biased Random Instruction Generation", *Proceedings 7th On-Line Test Workshop*, July 2001, pp. 65-17

[4] L. Chen, S. Dey, "DEFUSE: A Deterministic Functional Self-Test Methodology for Processors", *Proceedings IEEE VLSI Test Symposium*, 2000

[5] F. Corno, G. Cumani, M. Sonza Reorda, G. Squillero, "An RT-level Fault Model with High Gate Level Correlation", *IEEE International High Level Design Validation Workshop*, The Claremont Resort & Spa, Berkeley, California, November 8-10 2000

[6] F. Corno, M. Sonza Reorda, G. Squillero, "RT-Level ITC 99 Benchmarks and First ATPG Results", *IEEE Design & Test of Computers*, July-August 2000, pp. 44-53

[7] F. Corno, M. Sonza Reorda, G. Squillero, M. Violante, "On the Test of Microprocessor IP Cores", *DATE, IEEE Design, Automation & Test in Europe Conference*, Munich (Germany), 13-16 March 2001, pp. 209-213

[8] F. Fallah, S. Devadas, K. Keutzer, "OCCOM: Efficient Computation of Observability-Based Code Coverage Metrics for Functional Verification," *Proceedings 34th Design Automation Conference*, 1998

[9] S. Thatte, J. Abraham, "Test Generation for Microprocessors", IEEE Transaction on Computers, Volume 29, 1980