Effective Software-Based Self-Testing for CMOS VLSI Processors

Nektarios KRANITIS[†]

Abstract. Processor testing approaches based on the execution of self-test programs have been recently proposed as an effective alternative to classic external tester-based testing and pure hardware built-in self-test (BIST) approaches. Software-based self-testing is a non-intrusive testing approach that embeds a “software tester” in the form of a self-test program in the processor on-chip memory. It has the potential to provide high quality at-speed testing using low-cost external ATE without any hardware or performance overheads. In this thesis, we introduce a high-level, software-based self-testing methodology for embedded processors and microprocessors. The proposed methodology aims at high structural fault coverage with low test development and test application cost. We demonstrate the effectiveness and efficiency of the proposed methodology by completely applying it on several processor benchmarks with scaling complexity, implementing different implementations of a popular RISC instruction set architecture with several gate-level implementations.

1. Introduction

Consumer demands for high performance and rich functionality have driven semiconductor manufacturing industry to the integration of multiple complex components on a single chip. Deep-submicron technologies along with the use of the System-on-Chip (SoC) design paradigm have made this possible. System development based on the use of a core-based architecture, where cores are interconnected by an industry standard on-chip bus is very common. This design strategy has been proved to be effective in terms of development time and productivity since it re-uses existing intellectual property (IP) cores. In the multi-million gate SoC era, design and test engineers face signal integrity problems, serious power consumption concerns and maybe more than these, serious testability challenges.

Manufacturing testing of a SoC architecture built around one or more processor cores is a difficult task. Test data volume required for external testing of embedded processors is becoming excessively large [10] resulting in long test application time, while the cost of test time using high-performance Automatic Test Equipment (ATE) is very high. Furthermore, new types of defects appearing in deep-submicron technologies require at-speed testing in order to achieve high test quality. However, the increasing gap between ATE frequencies and SoC operating frequencies makes external at-speed testing almost infeasible. Moreover, ATE accuracy problems can lead to serious yield loss [10]. Thus, high quality external testing is becoming increasingly difficult even with today’s multi-million dollars ATE.

Traditional hardware self-test (or Built-In Self-Test - BIST) moves the testing task from external resources (ATE) to internal hardware, synthesized to generate test patterns and evaluate test responses of the circuit under test. Hardware self-test achieves at-speed testing reducing the overall test costs of the chip [10]. Recent applications of hardware-based commercial Logic BIST techniques in large industrial designs [9] and microprocessors [5] reveal that extensive and manual design changes have to be performed in order to make the design BIST-ready. These include changes to prevent the design from getting to an unknown state that will corrupt the compressed response and extensive test point insertion that is necessary to achieve acceptable fault coverage in random pattern resistance circuits. However, these changes increase the circuit area and degrade its performance. Therefore, Logic BIST has limited applicability to high-performance and power-optimized embedded processors.

Several approaches can be grouped together under the term, software-based self-testing (SBST) and various SBST techniques have been proposed recently as an effective alternative to hardware self-test for embedded processors. SBST has a non-intrusive nature since it utilizes existing processor resources and instructions to perform self-testing. Therefore, SBST can potentially provide sufficient testing quality without any impact on performance, area or power consumption during normal operation. SBST approaches with respect to the single stuck-at faults have been recently proposed and a review of some of them was given in [21]. Apart from this, [21] discusses the application of SBST to testing path delay fault and interconnect crosstalk defects, as well as fault diagnosis. Experimental results provided in [5] demonstrate the superiority of SBST for processors over both Full Scan design and hardware Logic BIST. An outline of the embedded SBST concept is shown in Figure 1.

[†] Thesis Advisor: Associate Professor Antonis Paschalis

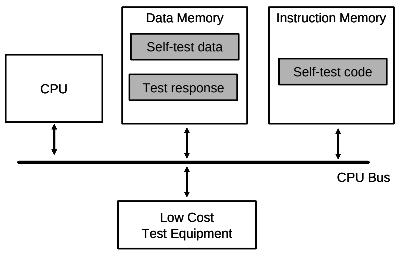

Figure 1: Software-based self-testing concept outline

Self-test routines and data are downloaded into instruction and data memories, respectively, from a low-speed, low-cost external mechanism. Subsequently, these self-test routines are executed at the processor actual speed (at-speed testing) and test responses are stored back in the on-chip RAM. These test responses may be in a compacted (self-test signatures) or uncompacted form and can be unloaded by the low-cost external ATE during manufacturing testing. Since today's microprocessors have a reasonable amount of on-chip cache, execution from on-chip cache is considered a further improvement when targeting at low-cost testers, provided that a cache-loader mechanism exists to load the test program and unload the test response [25]. As an alternative, self-test routines may be executed from an on-chip ROM dedicated to this task when periodic on-line testing is performed for the device.

In this thesis, we first introduce a high-level SBST methodology suitable for complex embedded processors and microprocessors implemented in CMOS VLSI technology. Test development of the proposed methodology is performed at high-level and aims at high structural fault coverage with low test development and test application cost. Low test development cost is accomplished by keeping engineer manual test development effort as low as possible while minimizing computational effort. Low test application cost is accomplished by developing low-cost (small and fast) test routines for the most important and critical-to-test components of the processor. Then, the effectiveness of the introduced methodology with respect to low-cost and high-level of application is demonstrated by successfully applying it on several processor benchmarks with scaling complexity, implementing different implementations of a popular RISC instruction set architecture with several gate-level implementations. The outline of the remainder of this thesis is as follows. In Section 2 an overview of previous related work is given. In Section 3 we present the proposed SBST methodology. In Section 4 we provide experimental results for various implementations of different processor model benchmarks. Finally, Section 6 concludes the thesis summary.

The research of the thesis summarized here, has been published in [13],[14],[15],[16],[17],[18],[19],[20],[23],[24].

2. Previous work

Traditionally, processor testing resorted in functional testing approaches. Pioneer among those traditional functional testing approaches, [29], is considered a landmark paper in processor functional testing. Based on the RTL description of the processor the authors introduced a functional fault model and considered the processor as a theoretical graph model, the S-graph. Since then, many processor functional testing methodologies were proposed. Those approaches were either based on a functional fault model (many were based on the model of [29]), or based on verification principles without assuming any functional fault models. Those traditional functional test approaches had a high-level of abstraction but required a large amount of manual test writing effort, usually very little fault grading was done on structural processor netlists while high fault coverage was not guaranteed. Furthermore, most of them relied on external ATE to feed the input test patterns and monitor the response in contrast with the SBST approaches that apply at-speed in a self-test mode.

The SBST approaches so far in the literature can be classified in two different categories. The first category includes the SBST approaches [28],[2],[25], that have a high-level of abstraction and are functional in nature. The second category includes the SBST approaches [3],[4],[5],[6] which are structural in nature and require structural fault-driven test development.

In [28], a functional test methodology is proposed that generates a random sequence of instructions that enumerate all the combinations of the operations and systematically selected operands. Test development is performed at a high-level of abstraction based on Instruction Set Architecture (ISA), which is highly desirable. However, since test development is not based on an a-priori fault model, the generated tests - applicable for design validation as well - cannot achieve high fault coverage without the use of large code sequences and a serious amount of effort for the user to determine the heuristics for the operands. Furthermore the use of large code sequences results in excessive test application time and very long fault simulation time required for fault grading. When applied on the GL85 processor model consisting of 6,300 gates and 244 FFs, a test program consisting of 360,000 instructions was derived and the attained fault coverage was 90.2%, dropped to 86.7% when the responses were compressed in a signature.

In [2], a self-test method is proposed which combines the execution of microprocessor instructions with a small amount of on-chip hardware that is used to provide a pseudorandom sequence of instructions, thus creating a randomized test program along with randomized data operands. Besides the fact that the proposed methodology cannot be considered as a "pure" SBST methodology due to the insertion of test hardware, the manual effort required for test program development is high while the pseurorandom philosophy results in very long test application time. Application on a RISC processor core similar to the DLX machine consisting of 27,860 gates resulted in 94.8% fault coverage with 220,000 cycles after an iterative process considering different parameters.

In [25], an automated functional self-test methodology is proposed based on generation of random instruction sequences with pseudorandom data generated by software LFSRs while uses the on-chip cache to be applied. Constraints are extracted and built into the generator to ensure generation of valid instruction sequences ensuring that no cache misses and/or bus access cycles are produced. The high-level functional nature of the proposed approach makes generation of instruction sequences and data that achieve fault coverage goals with reasonable instruction counts very difficult, while the huge amount of cycles inherent to the methodology makes fault grading a non-trivial task. Although compares favorably with classic functional testing manual test generation effort, user expertise and knowledge of the processor units under test is required as well as merging with manual tests in order to achieve high fault coverage. The methodology achieved 70% fault coverage when applied on the Intel® Pentium® 4 processor.

In [3], a partially automated test development approach based on gate-level tuning is proposed. First a library of macros is generated manually by experienced assembly programmers from the ISA, consisting of instruction sequences using operands as parameters. Then, a greedy search and a genetic algorithm are used to optimize the process of random macro selection among the macros set, along with selecting the most suitable macros parameters to build a test program that maximizes the attained fault coverage when the test program is applied on the gate-level netlist of the processor. Thus, test program development is fine-tuned to a specific gate-level netlist. The approach attained 85,2% fault coverage when applied on a 8051 8-bit microcontroller design of 6,000 gates using 624 instructions.

In [4], an automated test development approach is proposed based on evolutionary theory techniques (MicroGP), that maximizes the attained fault coverage when the evolved test program is applied on the gate-level netlist of the processor. It utilizes a directed acyclic graph for representing the syntactical flow of an assembly program and an instruction library for describing the assembly syntax of the processor ISA. Manual effort is required for the enumeration of all available instructions and their possible operands. Experiments on a 8051 8-bit microcontroller design of 12,000 gates, resulted in 90% fault coverage.

In [5], a SBST methodology is proposed which is structural in nature, targeting specific components and fine-tuning the test development to the low, gate-level details of the processor core. First, long pseudorandom pattern sequences are developed for each processor component taking into consideration manually extracted constraints imposed by its instruction set. Then, test sequences are encapsulated into self-test signatures that characterize each component. The component self-test signatures are afterwards expanded on-chip by a software LFSR into pseudorandom test patterns stored in embedded memory and are finally applied to the component by software test programs. The pseudorandom approach in [5] does not consider the regular structure of critical processor components and hence leads to large self-test code, large memory requirements and excessive test application time even when applied to a small processor model. Application to an accumulator based CPU core, Parwan, consisting of 888 gates and 53 FFs, resulted in 91.4% fault coverage in 137,649 cycles using a test program of 1129 bytes. Nevertheless, the gate level based test strategy with manual constraint extraction, as it is presented in [5], is a generalized approach targeting every processor component. However, in the case of large "real" processors, manual constraint extraction for the targeted components is rather impractical. Furthermore, the step that follows constraint extraction, constraint-based test generation for the processor modules using sequential ATPG is a very time consuming task, if at all achievable since usually only combinational components constrained test generation can be easily handled by commercial ATPG tools.

In [6], the authors propose a methodology that extends and improves their previous work by automating the complex constraint extraction phase, while emphasizing in ATPG-based test development instead of a pseudorandom one. Statistical regression analysis is applied on the RTL simulation results using manually coded instruction templates, to derive a model of the surrounding logic of the module under test (MUT). The learned model is converted into Virtual Constrained Circuit (VCC) followed by ATPG on the VCC-MUT in an iterative way. Although automation is achieved in several steps with manual effort is still required for the selection and construction of a set of representative test program templates among a quite large space of possible templates. Furthermore, this structural SBST methodology based on gate-level ATPG applies only to combinational processor components without exploiting any possible regular structure of critical-to-test combinational components. Moreover, it results in large test program sizes (thus high test cost) due to the fact that the test programs cannot be in a compact form since they are automatically generated using test programs templates and also because of the large amount of ATPG test patterns required to be downloaded from the slow external ATE to the on-chip memory. Application of the methodology to the combinational logic in the execution stage of a commercial processor from *Tensilica (Xtensa™)* with 24,962 faults resulted in 288 ATPG test patterns and 90.1% fault coverage after constrained ATPG. When the tests are applied using processor instructions in a test program of 20,373 bytes, the fault coverage for the targeted component is increased (due to collateral coverage) to 95.2% in 27,248 cycles.

The requirement of a gate-level netlist cannot be considered as a negative attribute in general, since it helps to improve test quality (higher fault coverage). However, test generation tuned on gate-level can be very computationally costly if applied in an iterative way with successive fault simulations (especially at the total processor level) in the test generation loop. Thus, SBST methodologies based on gate-level details can achieve high fault coverage, but require significant test development cost, due to the tremendous increase of the circuit size and complexity of modern processors. Therefore, it is highly desirable that a SBST methodology is based on a high Register Transfer (RT) level description of the processor and its instruction set architecture, thus providing a low-cost and technology independent test development strategy.

In [14],[15] a *high-level structural* SBST methodology was proposed, showing for the first time in the literature that small deterministic test sets, applied by compact test routines provide significant improvement when applied to the same simple accumulator-based processor design, Parwan, which was used in [5]. Compared to [5], the methodology described in [14],[15] *requires 20% smaller test program using 923 bytes, 75% smaller test data and almost 90% smaller test application time using 16,667 cycles*. Both methodologies achieve single stuck-at fault coverage slightly higher than 91% for the simple accumulator-based Parwan processor.

Despite the new perspective [14],[15] of deterministic test development at high-level for SBST of embedded processors, scaling from simple accumulator-based processor architectures to more realistic ones in terms of complexity like contemporary complex processors implementing commercially successful ISAs (i.e. RISC ISAs), brings out several test challenges that remain to be solved. These challenges arise when high-level test development is applied to these complex processor architectures that contain large functional components (i.e. fast parallel multipliers, barrel shifters, etc), large register banks and pipeline mechanisms, while trying to keep the test-cost as low as possible.

In [19],[20], the SBST methodology proposed, addressed these test challenges successfully by defining different test priorities for processor components and classifying them according to the defined priorities, proving that high-level test development based on ISA and RTL description of a complex processor ISA can lead to low test cost without sacrificing high fault coverage, independently of complex processor implementation and post-synthesis (gate-level) result.

The SBST methodology proposed in this thesis, shares desirable characteristics with functional-based SBST methodologies like test development at high-level using the ISA, but goes one step deeper using RTL information and a divide-and-conquer approach targeting individual components with respect to the stuck-at fault model, thus providing very high fault coverage. The proposed SBST methodology is also fundamentally different when compared with other SBST methodologies targeting structural faults [3],[4],[5],[6] since it is based only on the high RTL description of the processor and its ISA and thus it provides a technology independent test development strategy with gate-level netlist required only for fault grading purposes. On the contrary, in other structural-based SBST methodologies, gate-level netlist is not just a simple step of the test generation process (in the traditional way of ATPG or pseudorandom TPG) but it is also a key part of iterative test generation loops, which leads to increased computational cost. Focus at low test application cost (test program size and test program cycles) is another fundamental difference between this work and previous works in SBST targeting structural faults [3],[4],[5],[6]. Assigning different test priority to the processor components and then developing low-cost (small and fast) test routines for the most critical-to-test components of the processor results in smaller on-chip memory requirements, shorter test program download and application time while fault simulation time required for fault grading is minimized, thus providing an efficient low-cost alternative structural approach.

3. The Proposed High-Level Component-based SBST Methodology

The key characteristics of the proposed high RT-level, component-based, SBST methodology are the following:

- A divide-and-conquer approach is applied using component-based test development.
- Test development is based only on the Instruction Set Architecture (ISA) and the RT-level description of the processor without the need of low gate-level fine-tuning.

Although the main target always remains the high structural fault coverage (stuck-at fault model), test cost should be considered as a very important aspect. The proposed methodology has two main objectives both aiming to *low test cost*: (a) *the generation of as small and as fast as possible code routines* with (b) *as small as possible engineering effort and test development time*. The first objective leads to smaller download times at the low frequency of the external tester as well as to smaller test execution times of the routines, therefore reducing the total processor test time. The second objective reduces test development cost and time-to-market leading to significant improvements in product cost-effectiveness and market success.

The high RT-level test development of the proposed approach is well suited to the high RT-level flow of the design cycle. Since design, simulation, and synthesis are usually carried out at the high RT-level, test development can also be carried out at the same level providing high convenience and flexibility. In this case, processor cores can be easily integrated into a SoC environment, configured and re-targeted in a variety of silicon technologies without any specific need for fine-tuning the test development to specific synthesis optimization parameters using a specific technology library. Experimental results show that the gate-level independent proposed test strategy is very efficient and achieves more or less the same high fault coverage results for different gate-level implementations of the RTL processor core.

The proposed SBST methodology is shown in Figure 2.

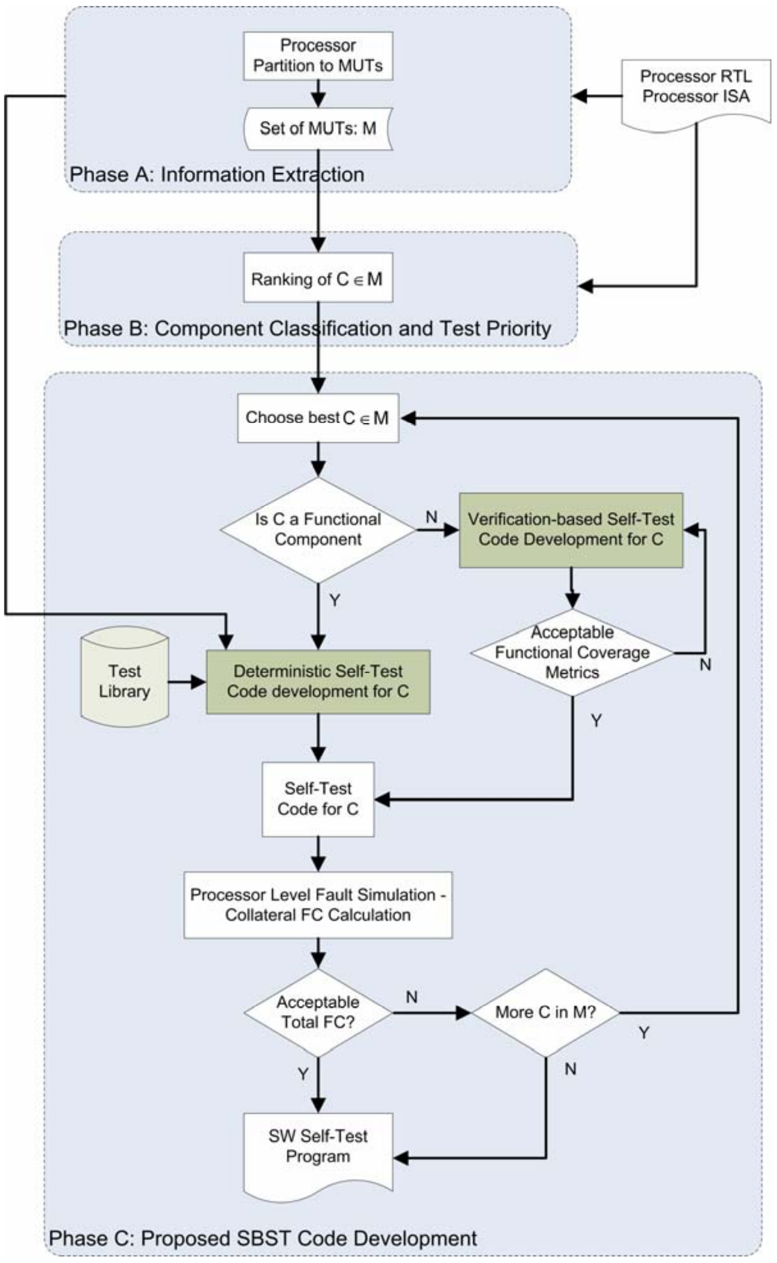
Processor Partition to MUTs
Set of MUTs: M
Processor RTL Processor ISA
Phase A: Information Extraction
Ranking of C ∈ M
Phase B: Component Classification and Test Priority
Choose best C ∈ M
Is C a Functional Component
N
Verification-based Self-Test Code Development for C
Y
Acceptable Functional Coverage Metrics
N
Test Library
Deterministic Self-Test Code development for C
Y
Self-Test Code for C
Processor Level Fault Simulation - Collateral FC Calculation
Acceptable Total FC?
N
More C in M?
Y
Y
N
SW Self-Test Program
Phase C: Proposed SBST Code Development

Figure 2: Proposed SBST Methodology

The proposed SBST methodology consists of three phases. These phases are summarized as follows:

Phase A

In *Phase A (Information Extraction)* the processor is partitioned to a set (M) of Modules Under Test (MUTs) using the processor RTL. Furthermore, the following information is extracted using the ISA and the RTL:

- the sets of component operations O_C
- the sets of *Basic Test Instructions I* that excite these component operations $I_{C,O}$
- the sets of *Peripheral Test Instructions* (or instruction sequences) for controlling or observing processor registers

Phase B

In *Phase B (Component Classification and Test Priority*), the processor components (C) are categorized in classes with the same properties and ranking is performed among the components $C \in M$ to determine the test priority.

We classify the processor components in the following three classes:

- *Functional components*, which are directly related to the execution of instructions (data processing / data storage) implementing the micro-operations and visible to the assembly language programmer. These are either:
  - *Computational components*, which perform specific arithmetic/logic operations on data. Components classified in this sub-category include: Arithmetic Logic Units (ALUs), shifters, barrel shifters, multipliers, dividers, etc.
  - *Interconnect components* between processor components, which serve the data flow in a processor datapath. Components classified in this sub-category include: multiplexers controlled by appropriate control signals.
  - *Storage components*, which serve as data storage elements that feed the data to the inputs of the computational components and capture their output. Components classified in this sub-category include: special processor registers visible to the assembly language programmer, the Register file, pipeline registers, etc.
- *Control components*, which control either the flow of instructions/data inside the processor core or from/to the external environment (memory, peripherals). These components include the processor controller, the instruction and data memory controllers that implement instruction fetching and similar components.
- *Hidden components,* which are the components inserted in the processor architecture usually to increase its performance but they are not visible to the assembly language programmer. These components can be either Functional or Control as well, and include pipeline logic (pipeline registers, pipeline multiplexers and control) and other performance increasing components related to Instruction Level Parallelism, branch prediction techniques, etc.

Component ranking is performed to determine the order in which test routines will be developed for each component. High priority components will be considered first while low priority components will be considered afterwards and only if the achieved overall fault coverage result is not adequate. In many cases test development for top priority components leads to sufficient fault coverage for not targeted components as well due to collateral coverage. This is particularly true in processor architectures where the execution of instructions in functional units excites many of the control subsystem components as well. The characteristics of a module that determine its priority in our methodology are the relative size and the accessibility (controllability and observability) of the component by processor instructions.

Therefore, according to the proposed methodology, the functional components of the processor have the highest test priority for test development since their size dominates the processor area and they are easily accessible. When total processor fault coverage is not sufficient, test development should proceed with components with lower test priority.

Phase C

In *Phase C (Self-test Code Development),* self-test routines are developed for each processor component C and the most critical components (components C that ranked with higher test priority) are targeted first. The self-test program consists of several self-test routines. Processor level fault simulation is performed to evaluate collateral coverage and determine if the total fault coverage is acceptable.

If C is a *functional component*, we follow a *Deterministic Self-Test Code Development* approach. The proposed methodology applies all possible component operations O_C with deterministic operands. Application of component operations O_C is performed by selecting the *Basic test instruction* $i \in I_{C,O}$ which requires the shortest instruction sequences (*Peripheral test instructions*) to apply the specific operand to component inputs and propagate the component outputs to the processor primary outputs. The key for selecting the most appropriate deterministic operands lies beneath the architecture of most critical-to-test processor components. Such components have an inherent regularity, which can lead to very efficient test algorithms for any gate-level implementation. This inherent regularity is not exploited either by pseudorandom test development or by ATPG-based test development approaches. Many processor components, in particular the vast majority of functional components like computational (arithmetic and logic operation modules), interconnect (multiplexers) and storage components (registers, register files) have a very regular or semi-regular structure. Regular structure appears in several forms like in the form of arrays of identical cells (linear or rectangular), tree-like structures of multiplexers, memory element arrays, etc. Such components can be efficiently tested with small and regular test sets that are gate-level independent, i.e. provide high fault coverage for any different gate-level implementation.

We have developed a component *Test Library* of test algorithms [7],[8],[11],[12],[22] that generate small deterministic tests and provide very high fault coverage for most types and architectures of functional processor components. The nature of these deterministic tests is fundamentally different from ATPG-generated test patterns since a gate-level ATPG tool is not "smart" enough to identify regular structures and generate patterns optimized for compact-code software-based test application. Based on these small deterministic tests we have developed test routines that, besides the small number of tests, take advantage of test vectors' regularity and algorithmic nature resulting in efficient compact loops. These loop-based compact test routines require a very small number of bytes while the small number of tests results in low test routine execution time. The *Test Library* routines that describe a test algorithm in assembly pseudocode, for almost all generic functional components, are tailored each time to the processor's under test instruction set and assembly language. We remark that although constructing such a *Test Library* seems like a time consuming task that requires a substantial manual effort, it is a *one-time-cost* that cannot be directly associated with manual effort and test development cost for a specific processor under test, since it provides reusability. Testing different processors requires only tailoring the generic processor component tests to their ISA. Of course, the component *Test Library* can be enriched anytime with new deterministic tests, like tests that deal with new architectures and algorithms implementing for example arithmetic/logic operations.

If *C* is a *control component*, we adopt a *Verification-based Self-Test Code Development* approach with test development still performed at high-level. Often, verification-based functional tests cannot guarantee acceptable fault coverage while the manual effort required to derive simple instruction sequences that verify control subsystem functionality is higher than the effort required for processor functional components where reusing our *Test Library* minimizes costly manual effort. Satisfaction of high-level RTL verification metrics supported by industry standard simulation tools like RTL statement, branch, condition and expression coverage helps to improve verification manual effort. Besides the overhead in test development cost, testing control and hidden subsystems impose a small overhead in the test program size and test application time as well. Such overheads characterize functional testing approaches, since it is very difficult for such components to be tackled by small and fast deterministic routines. Thus, we adopt such verification-based functional testing techniques for the control and hidden subsystems with the test cost/fault coverage trade-off in mind. We should remark that in many practical cases, verification-based test routines are developed at the design verification phase, therefore can be reused for the testing of control and hidden components with no additional manual effort involved or in the worst case substantially alleviating any manual self-test routine development effort.

4. Experimental results

The proposed methodology is demonstrated through its complete application to different processor benchmarks with scaling complexity. Some of these processor models are distributed under the GPL license agreement while other were developed to overcome architectural and functional limitations of the publicly available benchmarks. In order to evaluate the fault coverage, we developed a Test Evaluation Framework that was used for every processor benchmark case study. Tools from Mentor Graphics suite were used for VHDL synthesis, functional and fault simulation (*Leonardo*™, *ModelSim*™ and *FlexTest*™ products, respectively). Mentor Graphics suite and technology libraries were provided through Europractice service for academic institutions. For each processor benchmark, we first present logic synthesis results using different technology libraries. Then we present basic self-test program statistics like test program size and test program application time in clock cycles. Finally, we provide fault simulation results under the stuck-at fault model. The scope of this section is to prove the effectiveness of the proposed methodology by experimental results on different processor benchmarks with scaling complexity.

4.1 Case study A: Plasma/MIPS

Plasma/MIPS is a public available RISC processor model that supports interrupts and all MIPS I^{TM} user mode instructions except unaligned load and store operations (which are patented) and exceptions. The synthesizable CPU core is implemented in VHDL with a 3-stage pipeline [27]. We have enhanced the CPU model of [27] by adding a parallel multiplier to provide closer correspondence with real-world high-performance RISC processor core models. The fast multiplier module was generated using a public available module generator [26] and has the following characteristics: Booth recoding, Wallace trees for partial product summation and fast carry look-ahead addition at the final stage.

The *Plasma/MIPS* processor VHDL model was synthesized in two different technology libraries with different synthesis parameters. Synthesis A was optimized for area, targeted a 0.35 um technology library and the design runs at a clock frequency of 57 MHz. Synthesis B was also optimized for area, targeted a 0.50 um technology library and the design runs at a clock frequency of 42 MHz. Finally, Synthesis C was optimized for delay, targeted a 0.35 um technology library and the design runs at a clock frequency of 74 MHz. The resulting gate counts for each component and the processor overall, are shown in Table 1. A 2-input NAND gate is the gate count unit.

Component Name	Synthesis A 0.35 um/ 57MHz	Synthesis B 0.50 um/42 MHz	Synthesis C 0.35 um/74 MHz
Register File	9,905	11,558	11,905
Paral.Mult/ Serial Div.	11,601	11,654	13,358
ALU	491	558	900
Shifter	682	636	834
Memory Controller	1,119	1,120	1,163
PC Logic	444	449	493
Control Logic	230	244	361
Bus Multiplexer	453	431	623
Pipeline	885	876	961
Total CPU	26,080	27,824	30,896

Table 1: Plasma/MIPS components gate counts

According to the proposed methodology, the components of highest priority for test development are the five functional components of the Plasma/MIPS processor: register file, parallel multiplier/serial divider, ALU and shifter. Although very high total fault coverage (>94.5%) was achieved by targeting these functional components due to collateral coverage, the test program was enhanced by verification-based functional testing routines targeting memory controller and control logic.

Component	Size (words)	Clock Cycles
Register File	319	582
Parallel Multiplier	28	3,122
Serial Divider	41	1,154
ALU	79	275
Shifter	210	340
Memory Controller	76	160
Control Logic	100	164
Total CPU	853	5,797

Table 2: Self-test routines statistics on Plasma/MIPS

The self-test routine statistics are given in Table 2 in number of words and clock cycles for self-test routine execution. Along with total self-test routine statistics, partial statistics are given for each targeted component as well. Fault simulation results after the application of the developed test routines to processor netlists led to the fault coverage results given in Table 3. The components that have been targeted for test development are marked on their left side in the table.

		Fault coverage (%)		
	Component Name	Synthesis A	Synthesis B	Synthesis C
√	Register File	97.8	97.8	97.8
√	Multiplier/Divider	96.3	96.1	95.2
√	Arithmetic-Logic Unit	96.8	97.5	95.8
√	Shifter	98.4	99.9	95.3
√	Memory Control	87.9	88.5	90.3
	Program Counter Logic	89.3	88.3	85.3
√	Control Logic	54.9	54.9	55.9
	Bus Multiplexer	71.8	72.0	71.3
	Pipeline	98.4	96.9	96.0
	Total CPU	**95.3**	**95.3**	**94.5**

Table 3: Fault coverage results on Plasma/MIPS

As it is shown in Table 3, we obtained very similar fault coverage results when the processor was synthesized in different technology libraries with different optimization scripts, optimizing either for area or performance. This set of experiments was performed to show that test development for the targeted components which is performed at high RT-level results in high structural fault coverage that does not depend in the gate-level synthesis implementation. The application of the proposed SBST methodology to a 30,000 logic gates RISC processor benchmark can be summarized as follows:

Using a self-test program of less than 1K words that is executed in less than 6,000 clock cycles which leads to very low test application cost, we can achieve high quality at-speed testing with fault coverage >95%. The self-test program is gate-level independent, which means independent of the logic synthesis parameters and technology library used.

4.2 Case study B: ASIP Meister/MIPS

We developed a MIPS R3000 compatible processor with a 5-stage pipeline using the *Application Specific Instruction set Processor (ASIP)* design environment *ASIP Meister* [1] for evaluating the proposed methodology in a different processor implementation of the same MIPS I ™ ISA. A 52 instruction subset of the MIPS R3000 instruction set was developed while coprocessor and interrupt instructions were not implemented. It should be noted, that in the current educational release version of the *ASIP Meister* design environment [1], data hazard detection and register bypassing control logic are not implemented for IP security reasons. The automatically generated RTL VHDL model has been synthesized, optimized for area, targeting a 0.35 um technology library and the design runs at a clock frequency of 44 MHz with a total gate count of 37,400 gates. The resulting gate counts for each component and the processor overall, are shown in Table 5.

A test program was constructed targeting the functional components and was further enhanced by instruction sequences targeting the processor controller following a verification-based functional testing approach. Test program statistics are illustrated in the following Table 4. It should be noted that although this processor benchmark implements almost the same MIPS I ™ ISA as the previous one with more complex control and pipeline, the fact that the VHDL RTL design generated by the ASIP design environment does not support data hazard detection and register bypassing, a requirement for careful assembly instruction scheduling is imposed along with insertion of *nop* instructions wherever it is necessary. This resulted to increased test program size and test execution time when compared to the previous case study A.

Component	Size (words)	Clock Cycles
Register File	720	859
Parallel Multiplier	68	5855
Serial Divider	65	1396
ALU	192	1188
Shifter	378	437
Hi-Lo Registers	30	35
Control Logic	275	291
Total CPU	1,728	10,061

Table 4: Self-test routine statistics on ASIP Meister/MIPS R3000 compatible processor

In Table 5, one should notice the very high fault coverage percentage for the targeted processor functional units. For example, the computational functional units (ALU, Multiplier, Shifter) were targeted by simply tailoring our *Test Library* algorithms that apply precomputed gate-level independent deterministic tests, to the MIPS assembly language and fault coverage of 98.4%, 99.0% and 99.8% respectively was achieved. Similarly, the storage functional units (Register file, HI-LO registers) were targeted by our precomputed gate-level independent deterministic tests and fault coverage of 99.8% and 100% respectively was achieved. The total processor fault coverage is 92.6% which is less that the previous case study benchmark due to the more complex pipeline (5-stage pipeline) and pipeline control, which are not targeted in this case study. If there were no pipeline control logic implementation limitations in the *ASIP Meister* suite due to IP protection, much higher fault coverage could be attained by targeting the pipeline logic.

Component	Gates	FC (%)
√ Register File	11,414	99.8
√ Parallel Multiplier/Serial Divider	12,564	95.2
√ ALU	658	98.4
√ Shifter	633	99.8
√ Hi-Lo Registers	536	100.0
√ Controller	2,352	79.2
Data Memory Controller	1,086	58.2
Program Counter Logic	644	58.5
Instruction register	275	97.4
Pipeline registers	5,693	91.0
Total CPU	37,402	92.6

Table 5: Gate count and fault coverage statistics on ASIP Meister/MIPS R3000 compatible processor

4.3 Case study C: Athena/MIPS

Athena is an in-house developed processor developed to fulfill the requirements of a fully functional pipelined processor benchmark. *Athena's* complexity is the higher in the complexity scale among the processor benchmarks used for the experimental results. *Athena* is a 32-bit embedded RISC processor core that implements a 5-stage pipeline with hazard detection and forwarding mechanisms. It implements the full MIPS-I ISA with the sole exception of the unaligned load and store operations that are patented. The RTL processor model was synthesized targeting a 0.18um technology library. Synthesis was optimized for area and the netlist gate-count was 26,122 gates with 1,515 FFs. The design runs at a clock frequency of 82 MHz. The resulting gate counts for each component and the processor overall, are shown in Table 7.

A self-test program was composed targeting the computational functional components (parallel multiplier, ALU, shifter), the interconnecting functional components (pipeline forwarding multiplexers) and the storage fuctional components (Register file, HI,LO registers and pipeline registers). All functional components were targeted by simply tailoring our *Test Library* algorithms that apply precomputed gate-level independent deterministic tests, to the MIPS assembly language. The self-test program was enhanced by self-test routines targeting control components including pipelined control, the pipeline stage controllers and the hazard detection unit. Control components were targeted using a coverage driven, verification-based self-test code development approach aiming at maximizing functional coverage metrics by taking advantage of existing software platforms. *Athena Processor Verification Suite (APVS)* which was developed at the design phase of *Athena*, automatically generated verification-based functional test code. Through re-usability, any manual self-test routine development effort was substantially alleviated. Test program statistics are illustrated in the following Table 4.

Self-test program statistics	
Size (words)	3,014
Execution time (cycles)	11,637

Table 6: Self-test program statistics on Athena/MIPS processor

Again, the effectiveness of our SBST methodology for the most critical-to-test processor functional components using high-level Deterministic Self-Test code development is clearly shown in the fault simulation statistics given in Table 7. For example, complete or almost complete coverage was achieved for the computational functional components (ALU, Multiplier, Shifter), the storage functional components (Register file, HI-LO registers) and the interconnect functional components (pipeline forwarding multiplexers). Verification-based self-test code development for the least critical-to-test control components can be performed at high-level using functional metrics can but cannot guarantee very high fault coverage.

The application of the proposed SBST methodology to *Athena*, a fully 5-stage pipelined RISC processor benchmark can be summarized as follows:

Using a self-test program of about 3K words that is executed in less than 12,000 clock cycles, we can achieve high quality at-speed testing with fault coverage >96%.

Component	Gates	FC (%)
Register File	9,669	100.0
Parallel Multiplier	8,746	98.7
Arithmetic-Logic Unit	579	99.7
Shifter	861	100.0
HI-LO Registers	484	100.0
Pipelined Control	196	84.0
Pipeline registers	2,968	94.8
Pipeline stage controllers	239	94.7
Hazard Detection Unit	26	92.9
Forwarding Unit	61	87.8
Forwarding Multiplexers	483	100.0
Total CPU	26,122	96.3

Table 7: Gate count and fault coverage statistics on Athena/MIPS processor

5. Conclusions

In this thesis, we presented a high-level component-based software-based self-test methodology and demonstrated its application on three complex embedded processor architectures that implement a very popular RISC instruction set including different gate-level implementations of these processors.

The proposed SBST methodology does not require either manual or automatic constraint extraction for the processor components or any sequential ATPG at the gate level. It can be applied when just an RT-level description and the Instruction Set Architecture of the processor are available. Post synthesis gate-level netlist is required only for fault grading and it does not consist a part of any test generation iterative loop. The methodology aims to construct small and fast self-test routines for the most important and test critical components of the processor in order to achieve a low test application cost. Low test development cost is accomplished by keeping engineer manual test writing effort as low as possible with test routine library *reuse* while at the same time minimizing computational effort. Low-speed, low-cost external testers are excellent utilized by the proposed methodology, since the test program downloading from the tester memory to the on-chip memory, performed at the low frequency of the tester, is completed in very short time intervals.

We have demonstrated that high test quality (more than 95%, 92.6% and 96.3% respectively) fault coverage is achieved for the three benchmark processors implementing a popular RISC ISA with scaling complexity and different gate-level implementations while keeping total test application cost as low as possible.

References

[1] ASIP Meister http://www.eda-meister.org

[2] K. Batcher, C. Papachristou, "Instruction randomization self test for processor cores", *in Proc. of the IEEE VLSI Test Symposium (VTS) 1999*, pp. 34-40.

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

[4] F. Corno, G. Cumani, M. Sonza Reorda, G. Squillero, "Fully Automatic Test Program Generation for Microprocessor Cores", *in Proc. of the IEEE Design Automation & Test in Europe Conference (DATE) 2003*, pp.1006-1011.

[5] L. Chen, S. Dey, "Software-Based Self-Testing Methodology for Processor Cores", *IEEE Transactions on CAD of Integrated Circuits and Systems*, Vol.20, No.3, pp. 369-380, March 2001.

[6] L. Chen, S. Ravi, A. Raghunathan and S.Dey, "A scalable software-based self-test methodology for programmable processors," *in Proc. of the Design Automation Conference (DAC) 2003*, pp 548-553.

[7] D. Gizopoulos, N. Kranitis, A. Paschalis, M. Psarakis, and Y. Zorian. "Low Power/Energy BIST Scheme for Datapaths", *in Proc. of the 18th IEEE VLSI Test Symposium (VTS) 2000*, pp. 23-28

[8] D. Gizopoulos, N. Kranitis, A. Paschalis, M. Psarakis, and Y. Zorian, "Effective Low Power BIST for Datapaths", *in Proc. of the IEEE Design Automation & Test in Europe Conference (DATE) 2000*, p.757

[9] G. Hetherington, T. Fryars, N. Tamarapalli, M. Kassab, A. Hassan and J. Rajski, "Logic BIST for large industrial designs: Real issues and case studies", *in Proc. of the IEEE Intl.Test Conference (ITC) 1999*, pp.358-367

[10] ITRS, 2001 edition, http://public.itrs.net/Files/2001ITRS/Home.htm

[11] N. Kranitis, D. Gizopoulos, A. Paschalis, M. Psarakis and Y. Zorian, "Power/Energy Efficient Built-In Self-Test Schemes for Processor Datapaths", *IEEE Design & Test of Computers, Special Issue on Microprocessor Test and Verification*, vol. 17, no. 4, pp. 15-28, October-December 2000

[12] N. Kranitis, A. Paschalis, D. Gizopoulos, M. Psarakis and Y. Zorian, "An Effective Deterministic BIST Scheme for Shifter/Accumulator Pairs in Datapaths", *Journal of Electronic Testing, Theory and Applications (JETTA)* , vol. 17, pp. 97-107, April 2001

[13] N. Kranitis, A. Paschalis, D. Gizopoulos, Y. Zorian, "Effective Software Self-Test Methodology for Processor Cores", *in Proc. of the IEEE Design Automation & Test in Europe Conference (DATE) 2002*, pp. 592-597

[14] N. Kranitis, D. Gizopoulos, A. Paschalis, Y. Zorian, "Instruction-Based Self-Testing of Processor Cores", *in Proc. of the IEEE VLSI Test Symposium (VTS) 2002*, pp. 223-228.

[15] N. Kranitis, A. Paschalis, D. Gizopoulos, Y. Zorian, "Instruction-Based Self-Testing of Processor Cores", *in Journal of Electronic Testing: Theory and Applications (JETTA)*, no 19, pp.103-112, 2003 (Special Issue on 20th IEEE VLSI Test Symposium 2002)

[16] N. Kranitis, G. Xenoulis, D. Gizopoulos, A. Paschalis, Y. Zorian, "Low-Cost Software-Based Self-Testing of RISC Processor Cores", *in Proc. of the IEEE Design Automation & Test in Europe Conference (DATE)*, 2003 pp. 714-719 *(Best Paper Award Nomination)*

[17] N. Kranitis, G. Xenoulis, D. Gizopoulos, A. Paschalis, Y. Zorian, "Software-Based Self-Testing of Large Register Banks in RISC Processor Cores", *in Proc. of the 4th IEEE Latin-American Test Workshop (LATW)*, January 2003

[18] N. Kranitis, G. Xenoulis, D. Gizopoulos, A. Paschalis, Y. Zorian, "Low-cost software-based self-testing of RISC processor cores", *IEE Proceedings - Computers and Digital Techniques*, *Invited Paper, Special Issue on the IEEE Design, Automation and Test in Europe Conference (DATE'03)*, vol. 150 , no. 5 , pp. 355-360, Sept. 2003

[19] N. Kranitis, G. Xenoulis, A. Paschalis, D. Gizopoulos, Y. Zorian, "Application and Analysis of RT-Level Software-Based Self-Testing for Embedded Processor Cores", *in Proc. of the IEEE Intl.Test Conference (ITC) 2003* , pp. 431-440.

[20] N. Kranitis, A. Paschalis, D. Gizopoulos, G. Xenoulis, "Software-Based Self-Testing of Embedded Processors", *IEEE Transactions on Computers*. Vol. 54, No.4, pp. 461-475, April 2005

[21] A. Krstic, L. Chen, W.C. Lai, K.T. Cheng, S. Dey, "Embedded Software-Based Self-Test for Programmable Core-Based Designs", *IEEE Design & Test of Computers*, July-August 2002, pp. 18-26.

[22] A. Paschalis, D. Gizopoulos, N. Kranitis, M. Psarakis, Y. Zorian, "An Effective BIST Architecture for Fast Multiplier Cores", *in Proc. of the IEEE Design Automation & Test in Europe Conference (DATE) 1999*, pp.117-121

[23] A. Paschalis, N. Kranitis, D. Gizopoulos, M. Psarakis, Y. Zorian, "Effective Deterministic Arithmetic BIST Architecture for Embedded Processor Cores", *in Proc. of the 4th IEEE International Workshop on Testing Embedded Core-based System-Chips, (TECS'00)*, May 2000

[24] A. Paschalis, D. Gizopoulos, N. Kranitis, M. Psarakis, Y. Zorian, "Deterministic Software-Based Self-Testing of Embedded Processor Cores", *in Proc. of the IEEE Design Automation & Test in Europe Conference (DATE) 2001*, pp. 92-96

[25] P. Parvathala, K. Maneparambil, W. Lindsay, "FRITS – A Microprocessor Functional BIST Method", *in Proc. of the IEEE Intl.Test Conference (ITC) 2002*, pp. 590-598.

[26] J. Pihl, E. Sand, Arithmetic Module Generator, http://www.fysel.ntnu.no/modgen

[27] Plasma CPU Model. http://www.opencores.org/projects/mips

[28] J. Shen, J. Abraham, "Native Mode Functional Test Generation for Microprocessors with Applications to Self-Test and Design Validation", *in Proc. of the IEEE Intl. Test Conference (ITC) 1998*, pp. 990-999.

[29] S.M. Thatte, J.A. Abraham, "Test Generation for Microprocessors", *IEEE Transactions on Computers*, Vol C-29, No.6, pp. 429-441, June 1980.