An Enhanced Technique for the Automatic Generation of Effective Diagnosis-oriented Test Programs for Processor

E. Sánchez, M. Schillaci, G. Squillero, M. Sonza Reorda
Politecnico di Torino
Dipartimento di Automatica e Informatica
Torino, Italy
{edgar.sanchez, massimiliano.schillaci, giovanni.squillero, matteo.sonzareorda}@polito.it

ABSTRACT

The ever increasing usage of microprocessor devices is sustained by a high volume production that in turn requires a high production yield, backed by a controlled process. Fault diagnosis is an integral part of the industrial effort towards these goals. This paper presents a new methodology that significantly improves over a previous work. The goal is construction of cost-effective programs sets for software-based diagnosis of microprocessors. The methodology exploits existing post-production test sets, designed for software-based self-test, and may use an already developed infrastructure IP to perform the diagnosis. Experimental results are reported in the paper comparing the new results with existing ones, and showing the effectiveness of the new approach for an Intel i8051 processor core.

1. Introduction

Nowadays industries are aggressively scaling the features in MOSFETs for leading-edge logic technology in order to sustain the historical trends of improved device performance. The 2005 International Technology Roadmap for Semiconductors (ITRS05) clearly points out that this migration of CMOS technology is severely challenging the traditional failure analysis process. The conventional approach will increasingly be too slow and difficult for routine analysis. Instead, ITRS05 maintains that software-based diagnosis (SBD) is becoming a key area, and that SBD methodologies and tools will be required to handle in the near future all major test methodologies, including scan-based and BIST-based test, functional, IDDQ and, especially important, AC (delay) test. It seems probable that the new diagnostic methodologies will be based on both test structures and product-level tests. Design for testability (DFT) techniques such as built-in self test (BIST) will be likely designed with special consideration to support the necessary data gathering [12]. Similar problems are expected to appear with mass-production devices, such as small microprocessors and microcontrollers, where the high volumes together with the tight cost bounds will exacerbate the situation.

Recently, Bernardi et al.[11] presented a methodology for minimizing the cost of microprocessor SBD in a System-on-Chip (SoC). The approach exploits an existing set of programs for post-production test converting it to a set of programs able to diagnose faults. This new set is then reduced in a process called *sifting*, and eventually enhanced with an evolutionary tool. The approach could also reuse the infrastructure IP (IIP) originally designed for the software-based self test (SBST) of the device, limiting the area overhead.

Sifting is shown able to get an 88.3% reduction of the original set, shrinking it from 1,414KB to 165KB [11]. The approach leads to rather long execution times for generating the final diagnostic set: 325 hours a SUN Blade workstation for the sifting, plus 168 hours for improving the initial diagnostic set; but leads to good results in terms of diagnostic resolution.

The cost of a diagnostic process can be roughly considered composed of two factors: the cost of generating the procedure, and the cost of applying it. The former reflects the computational resources required; the application cost is mainly influenced by the volume of data required by the diagnostic procedure, during both test set application and response analysis, by the time required for test application, and by the type of exploited automatic test equipment (ATE).

This paper presents a significant improvement over [11]. Whereas the general structure of the approach is the same, both sifting and the genetic enhancement have been completely redesigned.

The enhancements lead to an interesting reduction in computational effort, an increase in diagnostic resolution, and a dramatic reduction in the size of the final diagnostic set, thus leading to significantly smaller diagnosis time.

The paper is organized as follows: section 2 provides a concise background in fault diagnosis for digital circuits, and outlines the basic concepts on SBST; section 3 details the proposed approach, with a discussion of its new improvements; section 4 presents some experimental results; finally, section 5 concludes the paper.

2. Fault Diagnosis and SBST Background

This section summarizes some concepts about the fault diagnosis and software-based self-test topics, and formalizes the terminology adopted in this paper.

2.1 Fault diagnosis

In testing, a circuit is classified as faulty if its responses are different from the ones obtained by the good circuit; thus, the boolean comparison between the

978-3-9810801-2-4/DATE07 © 2007 EDAA

responses of the good circuit and the faulty circuit is called *syndrome*. Once a circuit has been classified as faulty, the process performed to establish the location of the circuit failure is called *fault diagnosis*, and the set of patterns required to accurately determine the location of the failure is named *diagnostic test set* (DT).

Two faults f_1 and f_2 are *structurally equivalent* if and only if no test set could be able to produce a different response for f_1 and f_2. on the other hand, the *diagnostic fault equivalence* could be defined in the following terms: two faults f_1 and f_2 belonging to a fault list F can be classified as *diagnostically equivalent* (or *undistinguishable*) for a given diagnostic set D if they produce the same syndrome during the application of D. Thus, structurally equivalence implies diagnostic equivalence but not the opposite. Structural equivalence is exploited in testing as well as in diagnosis trying to reduce the fault list size, decreasing fault simulation times.

A widely studied diagnosis topic is the generation of an effective DT; state-of-the-art DT generation techniques employ ATPG tools [1], [3], random [10] and evolutionary [4] pattern generation or functional approaches [9], [11]. The effectiveness of a DT is usually assessed using measurements related to *diagnostically equivalent fault classes*. For a given DT, an equivalent fault class (ec_i) is a subset of the fault universe including undistinguishable faults [1], [2]; clearly, structurally equivalent faults will always belong to the same *ec*, no matter the considered DT. Every individual ec_i is completely disjoint from the others, and their union is the fault universe itself. In simple words, a DT is most effective if it is able to split the fault universe in the biggest possible number of *ec*'s as small as possible.

Let us define *D(n)* as the fraction of faults that are classified into equivalence classes of cardinality less than or equal to *n* by the used DT.

The ability of a generated DT is usually measured by means of its *diagnostic power*, defined as the fraction of all faults completely distinguished from all other faults or belonging to fault classes ec_i of size 1 (i.e., *D(*1*)*), or its *diagnostic expectation*, that is a simple average of *ec* sizes [5].

It is also useful to define *D(10)* as the fraction of faults that can be considered *correctly classified*, because the exact analysis of equivalence between faults cannot be performed for medium or large sequential circuits.

Information to built diagnostic structures is usually gathered following two classification approaches:

- a *coarse classification*, obtained by processing only the pass/fail information related to each test pattern belonging to *DT*
- a *fine classification*, performed using the whole faulty circuit syndromes and consists in building an *output-based* diagnostic tree for each equivalent fault class to be further divided.

The coarse classification process requires fast fault simulations producing only a go/nogo information for each simulated fault, reducing the process accuracy; on the other side, fine classification process execute slow fault simulations improving the process precision.

2.2 Software-based testing techniques

Traditionally, parametric testing of processor cores has been applied using an ATE [6]. However, technological progress is pushing up the complexity and operating frequencies of low-end microprocessor cores. Thus, even though ATE effectiveness on applying parametric test is unquestionable, the costs for an ATE able to run at-speed functional tests are becoming prohibitive [8]. To overcome these problems, industries are trying to reduce the use of expensive ATEs. One interesting strategy is to perform microprocessor testing resorting to the so called Software-Based Self-Test (SBST) [7], where the test set consists in a set of assembly programs and does not rely on any special test point to force values or observe behaviors during test application. Such programs are loaded in an internal memory resorting to the available resources, and then executed in the core. The processor is labeled as good or faulty depending on the results produced by the test programs. A minimum effort is thus needed to extract test results.

SBD has several advantages with respect to the use of hardware-based methods: the diagnostic process can be performed at-speed without using costly equipment; in case scan chains are used there is no dependence on their exact structure, so the approach works even if the scan chains don't; as any software method, SBD is intrinsically technology-independent. Finally, the method benefits from a high flexibility, since the test program can be easily adapted to new needs. On the other side, SBD has a lower control and observation abilities upon the internal nodes of the circuit; the method is still not fully mature, so it requires significant effort to generate the diagnostic program set.

Regarding SBD, a deterministic approach has been proposed in [9], where the authors exploit SBST to tackle diagnosis of the 2k-gate processor called PARWAN. They proposed the following:

- A great number of short test programs are generated in order to partition the fault universe in as many subspaces as possible
- Each program presents a reduced set of instructions to isolate faults related to different processor functional parts
- Multiple copies of the same program are created, each propagating errors on different observable points in order to distinguish the faults affecting the processor outputs
- At the end of the test set creation a binary tree is built for use in the actual diagnosis process.

This technique is based on the processor functional characteristics instead of a pure structural analysis. Anyway, the effort required to generate a test set following these guidelines is not trivial, and grows with the complexity of the considered processor.

3. Proposed Approach

Herein, the evolution of the automatic methodology able to generate a suitable diagnostic set of programs starting from an initial test set built for post-production testing is presented. The previously devised method described in [11] is outlined; and then, the improved methodology is described.

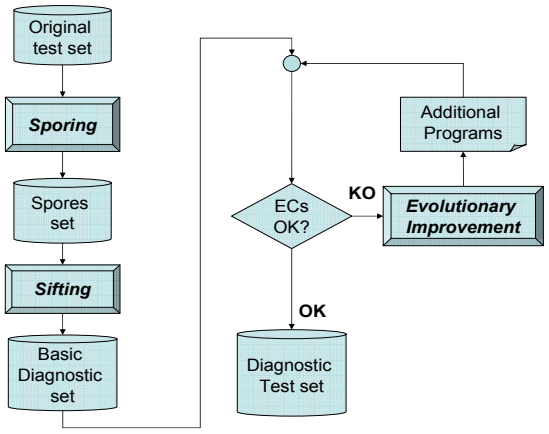

Figure 1: Methodology workflow

3.1 Base methodology

The method presented in [11] starts the generation of a set of test programs suitable for diagnosis from a post-production test set that is automatically divided in small programs, called *spores*; then, a heuristic process, called *sifting*, selects a subset of such small programs generating the initial diagnostic set. Finally, relying on an evolutionary tool called *μGP*, new programs are included into the diagnostic set improving the former results.

The workflow is divided in the following three steps:

- *Sporing*: the initial test set of programs is split up, generating a vast set of small programs or spores.
- *Static sifting*: following a static analysis, only the most promising programs are kept in the test set.
- *Evolutionary improvement*: resorting to an automatic tool, the diagnostic ability of the test set is improved.

Figure 1 illustrates the workflow of the proposed methodology; further details about this method can be found in [11]. A brief outline of the steps that have been improved in the new method is provided below.

3.1.1 Sporing

The execution of a program is simulated, and for every instruction execution a small program is generated; this program first sets the processor state to the one reached immediately before the considered instruction, then executes the instruction, and finally propagates the results to some accessible output, such as I/O ports.

3.1.2 Static sifting

As detailed in [11], the goal of sifting is to obtain a minimal diagnostic set without losing diagnostic capability with respect to the complete set of spores. To achieve this, first of all, every spore is assigned a fitness value, then the whole set is sorted in decreasing order; and finally, starting from the top of this list, spores are kept until their cumulative fault coverage equals that of the complete set; the following are discarded as redundant. Figure 2 reports the pseudo-code of the sifting process.

```
foreach (s in SporeSet)
  evaluateSporeFitness(s);
sortSporeSet();
T=faultCoverage(SporeSet);
foreach (s in SporeSet)
  B:=B+s;
  exit if (faultCoverage(B)=T)
```

Figure 2: Static sifting pseudo-code

The fitness values are computed based on the concept of *fault density* that is the number of spores able to detect a fault. Every spore is assigned a fitness value $f_s(d_F, NF_s)$:

$$f_s = \left(\sum_F \frac{1}{d_F} \right) \cdot \frac{1}{NF_s} \qquad (1)$$

where F is the fault index over the covered faults, d_F is the corresponding fault density and NF_s is the number of faults covered by the spore. The value of f_s ranges from 0 to 1 and the higher its value the higher the diagnostic capability of the spore.

3.1.3 Evolutionary improvement

The sporing process may generate a vast number of code fragments, all of which have to be fault simulated for evaluation. To save time fault dropping is employed; in this way, however, only a pass/fail information can be generated. The processor to be diagnosed has been equipped with an IIP containing a MISR to collect information from the I/O ports: this allows a finer classification of faults. The MISR signature is collected only at the end of the fault simulation.

The sifted spore set is then fault simulated and the information given by the MISR signatures is gathered. The signatures, 24 bit wide, contain more information than a simple pass/fail indicator, thus allowing to classify covered faults in different sets: this makes a *fine classification* possible.

After the fine classification has been performed using the sifted diagnostic set several large equivalence classes remain. Each of these is then targeted for splitting: an evolutionary tool, called μGP, is used to automatically

generate a program whose goal is to split that class, and whose fitness is computed accordingly.

3.2 Improved methodology

The new generation methodology is based on the existing workflow, but the individual steps have been redesigned to improve results. The starting point, an already existing test program, is the same: the main justification for its use is not just technical but also economic and it hasn't changed over time. The sporing process also remains identical, since its purpose is to decompose an existing program into small fragments corresponding, as closely as possible, to single instruction executions. As the base concept of a spore has not changed, so has not the process for their generation. Subsequent steps, however, have been refined.

3.2.1 Dynamic sifting

The new sifting process is as follows: the fitness of every spore is evaluated; then the spore with the highest fitness is considered for inclusion in the sifted set. If the total number of equivalence classes increases then the spore is included, otherwise it is discarded, and the one with the next highest fitness is analyzed.

Every time a spore is included in the sifted set the fitness evaluation has to be repeated, since some faults may be uniquely diagnosed. Taking into account these faults is useless, so they are eliminated from the fault list when computing densities.

The new dynamic sifting pseudo-code is shown in figure 3.

```
newClasses:=0;
B:=0;
do
  equivalenceClasses:=newClasses;
  foreach (s in SporeSet)
    evaluateSporeFitness(s);
  sortSporeSet();
  foreach (s in SporeSet)
    newClasses=evalClasses(B+s);
    if (newClasses>equivalenceClasses)
      B:=B+s;
      break;
while (newClasses>equivalenceClasses);
```

Figure 3: Dynamic sifting pseudo-code

Having produced a pass/fail information, every spore has the ability to partition the entire fault set into two disjoint sets. When multiple spores (multiple programs, indeed) are considered, all these partitions overlap, possibly producing further fragmentation of the fault set. An increase in equivalence class number directly corresponds to a greater fragmentation. Given an existing partition of the fault set the additional information given by a spore cannot move a fault from a subset (not even that of uncovered faults) to an already existing subset. Exactly like glass splinters belonging to a slab, fault sets can only be broken up, not merged. Unlike glass splinters, equivalence classes can only be divided up to individual faults, and, more important, they are much more difficult to aim at for splitting.

The advantage of dynamically re-computing fitness is that several spores with similar (static) fitness may help isolate the same subset of faults. Once the first one is included in the sifted set the others become nearly useless, but a static fitness does not reflect this fact. Worse, with a static fitness some of those may be included as well in the sifted set but only give a very small contribution to diagnosis. If many such contributions accumulate they may mask the existence of a few spores able to more effectively split the set. Experimental results show that this is indeed the case.

3.2.2 New evolutionary improvement

The new sifting process leads to vastly improved results with respect to the previous one: the same diagnostic power is obtained with just a fraction of the spores. Since spores are very short programs they are relatively easy to fault simulate. One drawback of the initial spore set is that, although very large, it does not include all opcodes and all operand combinations, even for opcodes present in the set. This is a consequence of their generation process: it starts from an existing set of test programs, and can only extract information contained therein, but does not generate more.

This leads to a simple idea for evolutionary improvement of the spore set: generate more spores by mutation of existing ones. Every spore has a fixed structure, composed of initialization of the processor state, execution of a single instruction, called target instruction, and observation of the results. The mutation is not intended to change this scheme, but rather to work within it. The performed modifications are of two kinds: small changes of the operands related to the target instruction, and arbitrary changes of executed operation. The reason for this different treatment of operands and opcodes is that the cardinality of the opcode set is much lower than the cardinality of the operands space.

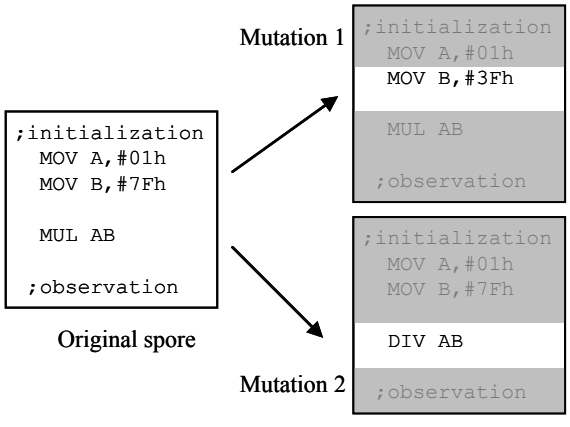

Figure 4: Two possible mutations of a spore

Figure 4 exemplifies two possible mutations of the original spore reported. In the first case, a slight variation is undergone by the *B* accumulator, toggling a single bit of its initial value; in the second case, the target instruction is mutated becoming a *DIV* instruction.

The evolutionary process starts from the sifted set and generates one new spore from a randomly chosen one. This is first compared with the existing ones to discover if it is identical to one of them: if it is, there is no need to evaluate it. To evaluate the spore a fault simulation is performed, collecting the signature for each fault, after which the equivalence class number is recomputed. If the new spore is able to split some fault subset then it is retained in the final set, together with simulation data; otherwise it is still retained as an existing (but useless) spore, and its simulation data are discarded. In this way it is ensured that every generated spore is only evaluated once.

4. Experimental evaluations

To experimentally demonstrate the new improvements of the enhanced method, the same case study presented in [11] was tackled, allowing us to perform a systematic comparison at each step of the proposed workflow. All the experiments were performed in a SUN Blade processor-based workstation.

As in [11], the considered microcontroller is an Intel i8051 microcontroller, supposed to be embedded into a SoC. The synthesized microcontroller, obtained using a generic home-developed library, contains 37,417 equivalent gates, and the collapsed fault list counts 12,642 faults.

The whole generating process was started from a post-production test set composed of 8 test programs written by hand, and reaching a fault coverage of about 92% on the collapsed list.

The *sporing* step generates about 60k test programs containing a few instructions that initialize the microprocessor, compute the target instruction, and propagate the results to save the syndrome; this set of programs is called spore set. Further details about sporing can be found in [13]. The spores set was fault simulated gathering only a pass/fail information in order to speed up the initial phase of the process. At this point of the process, the sifting is performed producing the initial test set; and finally, the evolutionary improvement left us with the final test set.

Table 1 summarizes the results obtained exploiting the new approach. In the columns, the main characteristics of the different sets of programs collected through the whole generation process are reported.

It could be noticed that not only the total application time of the diagnostic set does not grow, but it reduces by a factor of approximately four.

	Post-production test set	*Initial* test set	*Final* test set
Programs [#]	8	449	852
Test set size [KB]	4	13.20	18.26
Total Clock Cycles	1,003K	146.39K	239.85K
D(1) [%]	11.56	47.31	70.61
D(10) [%]	32.90	69.2	83.41

Table 1: The equivalence class summary for the analyzed processor core

Table 2 shows the overall comparison in the final results between the former methodology and the new one.

	[11]	*New method*
Programs [#]	7,266	852
Test set size [KB]	177	18.47
Total Clock Cycles	2,000.0K	246.7K
D(1) [%]	61.39	70.6
D(10) [%]	84.30	84.31

Table 2: Methodologies comparison

The new results compare very favorably with the old ones: the diagnostic power increases by about 9% and the percentage of correctly classified faults remains almost the same. These results are obtained with a diagnostic set much smaller than previously, both in terms of program number, memory occupation and application time. All of these figures decrease by about ten times.

Table 3 presents a comparison in terms of required time to perform both strategies.

	[11]	*New Method*
Sporing	0.3h	0.3h
Coarse FS	225h	225h
Sifting	0.5h	2.24h
Fine FS	100h	6h
Evolutionary improvement	168h	129h
TOTAL	493.8h	362.54h

Table 3: Time comparison

The total generation time decreases by about 30%: since both the sporing process and the fault simulation needed for the coarse classification (Coarse FS) remain unchanged, so do the related times; the sifting process is slower, but amounts to a small fraction of the total; since the initial test set is much smaller than previously, the fault simulation performed for the fine classification (Fine FS) is much faster; finally, the times needed for evolutionary improvement are roughly similar. It should be noticed that a trade-off exists between the duration of the evolutionary improvement and the quality of the obtained results.

For the sake of comparison, two additional experiments were performed in order to assess the goodness of the new algorithm improvements. Thus,

mixed experiments were launched using in the first case the old sifting or static sifting, and the new evolutionary improvement method; and in the second case the new sifting or dynamic sifting, and the old evolutionary improvement tool. Table 4 presents the obtained values mixing old and new approaches. The results are reported in terms of the diagnostic capability reached by the different sets of diagnostic programs.

	D(1) [%]	D(10) [%]
Static sifting alone	35.70	58.02
Static sifting + old EI	61.39	84.30
Static sifting + new EI	60.37	69.42
Dynamic sifting alone	47.31	69.20
Dynamic sifting + old EI	51.50	73.71
Dynamic sifting + new EI	**70.61**	**83.41**

Table 4: Mixed experiments

This table clearly shows that both modifications of the original methodology improve the quality of the final diagnostic set.

5. Conclusions and future work

In this paper, an improved methodology for the generation of software-based diagnostic sets for microprocessors has been presented.

The new improvements of the presented methodology allowed to achieve a substantial reduction in the computational effort required to automatically generate a diagnostic set of programs with respect to the method previously presented. Moreover, since the number of programs and the application time of the diagnostic set are also reduced, the effective time required to perform processor diagnosis is successfully reduced.

The reported results experimentally demonstrate that SBD techniques are becoming effective and economical solutions for the current diagnosis requirements.

6. Acknowledgements

The authors wish to thank Danilo Ravotto for implementing the new sifting and evolutionary tool, and performing experiments for the new methodology.

This work was partially supported by the Italian Ministry for University through the PRIN04 project "Tecniche per la progettazione di circuiti e sistemi elettronici digitali innovativi ad alta disponibilità e affidabilità".

7. References

[1] A.Veneris, R.Chang, M.S.Abadir, M.Amiri, "Fault equivalence and diagnostic test generation using ATPG", IEEE International Symposium on Circuits and Systems, Volume 5, 23-26 May 2004, Page(s): V-221 - V-224

[2] T.Bartenstein, "Fault distinguishing pattern generation", IEEE International Test Conference, 3-5 Oct. 2000, Page(s):820 – 828

[3] T.Gruning, U.Mahlstedt, H.Koopmeiners, "DIATEST: a fast diagnostic test pattern generator for combinational circuits", IEEE International Conference on Computer-Aided Design, 1991, Page(s):194 – 197

[4] P.Camurati, A.Lioy, P.Prinetto, M.Sonza Reorda, "Diagnosis oriented test pattern generation", IEEE European Design Automation Conference, 1990, Page(s):470 – 474

[5] P.G.Ryan, W.K.Fuchs, I.Pomeranz, "Fault dictionary compression and equivalence class computation for sequential circuits", IEEE International Conference on Computer-Aided Design, 1993, Page(s):508 – 511

[6] V.Agrawal, M. Bushnell, "Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits", Kluwer Academic Publishers, 2000

[7] L.Chen, S.Dey, "Software Based Self Test methodology using a embedded Soft-ware Tester", IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems, 2001, pp 369-380

[8] Semiconductor Industry Association (2002) International Technology Roadmap for Semiconductors 2002 Update, http://www.semichips.org/pre_stat.cfm

[9] L.Chen, S.Dey, "Software-based diagnosis for processors", IEEE/ACM Design Automation Conference, 2002, Page(s): 259-262

[10] R.C.Aitken, V.K.Agarwal, "A Diagnosis method using pseudo-random vectors without intermediate signatures", IEEE International Conference on Computer-Aided Design, 1989, Page(s): 574-580

[11] P.Bernardi, E.Sanchez, M.Schillaci, G.Squillero, M.Sonza Reorda, "An Effective Technique for Minimizing the Cost of Processor Software-Based Diagnosis in SoCs", IEEE Conference on Design, Automation and Test in Europe, 2006, Page(s): 412-417

[12] International Technology Roadmap for Semiconductors – ITRS 2005 Edition, http://www.itrs.net/Links/2005ITRS/Home2005.htm

[13] E.Sánchez, M.Sonza Reorda, G.Squillero, "On the transformation of Manufacturing Test Sets into On-Line Test Sets for Microprocessors", IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2005, pp. 494-504