An Effective Technique for Minimizing the Cost of Processor Software-Based Diagnosis in SoCs

P. Bernardi, E. Sánchez, M. Schillaci, G. Squillero, M. Sonza Reorda
Politecnico di Torino
Dipartimento di Automatica e Informatica
Torino, Italy
{paolo.bernardi, 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 novel cost-effective approach to the construction of diagnostic software-based test sets for microprocessors. The methodology exploits an existing post-production test set, designed for software-based self-test, and an already developed infrastructure IP to perform the diagnosis. An initial diagnostic test set is built, and then iteratively refined resorting to an evolutionary method. Experimental results are reported in the paper showing the feasibility and effectiveness of the approach for an Intel i8051 processor core.

1. Introduction

Microprocessor and microcontroller technology nowadays is virtually ubiquitous. The ever increasing usage of such devices is sustained by a high volume production. This in turn demands for a high production yield, backed by a controlled process. Fault diagnosis is an integral part of the industrial effort towards these goals.

Correct identification of the most common defective sections in a die helps to characterize the technological process, and localization of a fault allows to effectively direct physical investigation of the underlying defects. Moreover, most high-volume devices undergo several revisions. During these updates designers may decide to change some characteristics of a particular chip section in order to lower the impact of physical defects on the part (e.g., by increasing the minimum separation between active areas, or by changing the metal routing to obtain a flatter surface). The two activities of fault localization and design update are key in enhancing the final yield.

The high production volumes of many relatively simple devices call for an economically sound diagnostic methodology, since this has to be applied to a great number of faulty devices. It is therefore important to be able to devise a relatively low-cost diagnostic process.

Even more than test set construction, diagnostic set construction is a time-consuming activity. Most of the effort in diagnosis was directed towards combinational circuits, while sequential circuits received less attention, due to the widespread usage of scan-chain methodologies for test. Hard-to-test faults require a high computational effort for their coverage, but once detected they are usually easy to diagnose; easy-to-test ones, on the other hand, may be difficult to discriminate from each other and require a special effort for diagnosis. This difficulty can lead to long diagnostic tests, with correspondingly long application times and high costs.

In this paper we concentrate on a software-based diagnosis (SBD) methodology particularly suitable for microprocessor cores embedded in SoCs.

The main novelty of the proposed method is the automatic generation of a diagnostic test set using an existing post-production test set. Starting from it, we build an initial diagnostic test set which we progressively improve using an evolutionary method.

We propose to exploit an existing Infrastructure IP (IIP) [1] whose original purpose was software-based self test (SBST) [11], which is also able to provide the processor with the diagnostic test code, and to gather and store the compressed results. This solution has a very low hardware overhead, since the IIP circuitry has already been included to perform the post-production test. The advantages are numerous: the construction of the diagnostic test set exploits an existing test set, leveraging the economic benefit of reusing already performed work; the process is automated, effectively saving resources; being software-based, the diagnosis can be performed at-speed, both increasing the diagnostic capability and reducing the test application time; thanks to the reuse of the existing IIP the methodology does not require the use of high-performance (and high cost) test equipment.

As a case study we tackled SBD on a well-known microprocessor, widely used as a microcontroller core. The performed experiments allow to uniquely diagnose 61.83% of single suck-at faults, and to classify 84.30% in equivalence classes containing less than 10 faults.

The paper is organized as follows: section 2 provides a concise background in fault diagnosis for digital circuits; section 3 details the proposed approach, with a discussion of its key characteristics; section 4 presents some experimental results; finally, section 5 concludes the paper.

2. Diagnosis background

Fault diagnosis of VLSI circuits is one of the more investigated arguments in the test discipline. In the 60's,

3-9810801-0-6/DATE06 © 2006 EDAA

[2] [3] pioneered the field by introducing the first classification structures and test generation algorithms. More recently, some formal definitions aimed at unifying the fault diagnosis notation have been given in [4] [5] (i.e., the concepts of *Diagnostic Resolution*, *Diagnostic Power*, and *Equivalence Class*). These measures and concepts are currently used to characterize diagnostic test generation tools [6] [7], together with the usage of diagnostic trees [16].

In practical terms, an equivalence class (EC) is a set of faults exactly causing the same faulty behavior for each applied pattern; diagnostic test generation aims at determining a pattern set able to partition the circuit faults in a set corresponding to ECs as small as possible. Up to now, the methods to determine ECs and to generate suitable patterns can be classified in *structural* and *functional* [8] techniques, both analyzing the circuit structure. These inspections permit to identify the possible logic value allowing the separation of two potentially equivalent faults by propagating different response values on the observation points.

With respect to diagnostic techniques based on pattern generation, software-based diagnosis of processors presents additional problems:

- There is usually a common part of logic excited each time one instruction is executed, since test programs rely on instructions rather than on test patterns.
- A test program suited to excite a specific module of the processor could also cover a wide number of faults not belonging to the pinpointed part, and therefore offer a very reduced classification ability.
- Many of the internal processor circuit elements cannot be accessed directly using a specific instruction, thus resulting hardly diagnosable.

In [9], Chen and Dey tackled software-based diagnosis for 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.

Differently from [9], the purpose of our paper is to define a workflow for the automatic generation of a diagnostic test set starting from an existing post-production test set. Also, our approach uses an *n*-ary tree for fault classification, increasing the diagnostic capability of the set.

3. Proposed approach

The software-based diagnosis (SBD) methodology discussed herein is an automated method able to generate a suitable diagnostic set of programs starting from an initial test set built for post-production testing. Additionally, it exploits an evolutionary approach to improve the final results. Figure 1 graphically describes the workflow of the method.

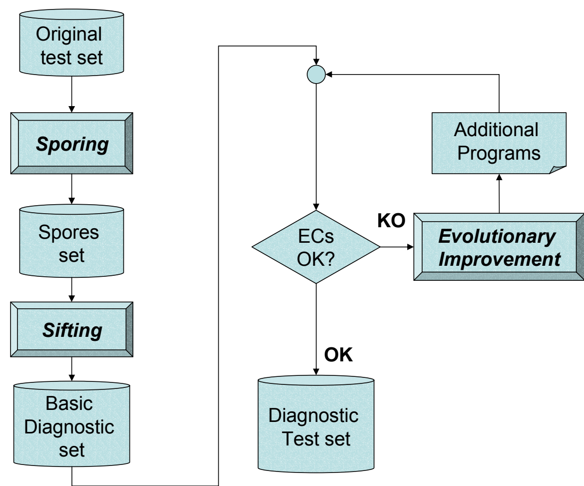

Figure 1: Method Workflow

In synthesis, the workflow is divided in the following steps:

- *Sporing*: the initial test set of programs is split up, generating a vast set of small programs.
- *Sifting*: following a heuristic 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.

The following sections will better detail all the above steps. Moreover, although it cannot be considered part of the algorithm, the evaluation of the diagnostic capability of the generated test set (*diagnostic assessment*) is fundamental and will be described as well.

3.1 Sporing

A test set of programs for post-production testing is usually devised to cumulatively cover the highest possible number of faults. Each program is written with a specific target, for example to cover the faults belonging to a functional module of the processor. A conventional set of

programs could be generated by using different approaches: by hand following some deterministic method (as in [11]); exploiting the test engineer expertise by writing test programs to cover corner cases; using automatic approaches (as in [13] or [14]); or even exploiting a random generation and compaction. In any case, the diagnostic set construction method presented here is independent on the origin of the initial test set of programs.

As mentioned before, the original test set is supposed to guarantee a high FC% of the processor but is likely unsuitable for diagnostic purposes, because its only goal is to cover faults, not distinguish them. A typical post-production test program is normally written with a definite set of goals, mainly compactness, short application time and high fault coverage. For pattern-based diagnosis it has been demonstrated that it is better to use many test sets each of which covers few faults [15]; likewise, for software-based diagnosis it is useful to have many very small programs that cover the smallest possible fault set.

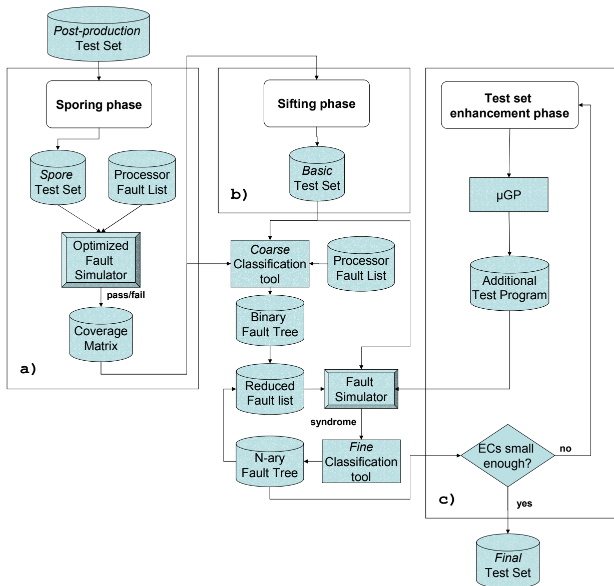

Figure 2: proposed workflow

Usually the high fault coverage is obtained by the stimulation of the functional modules of the processor with a great amount of input data, generated in a looping section of the test program. Many faults, especially in arithmetic units, need specific bit patterns on the execution unit inputs to be covered. If we could write a program that delivers only those data needed to detect a specific fault, and not others, that program would exhibit a very low fault covering ability, but a fairly good diagnostic capacity.

The basic idea is therefore to obtain a number of very small programs, each one covering a small number of faults (ideally the smallest possible number, indeed) not covered by other programs in the same set.

The *spores* are very small programs that put the processor in a specific state in order to control some specific part of the processor, execute a target instruction and propagate the results to the primary outputs. Each spore represents a completely independent program, able to excite some processor function, observe the results, and possibly signal fault occurrence. It is worth noting that the spores are not written from scratch, but generated by automatically breaking an existing test set in very small fragments, thus fully exploiting its covering ability.

The *sporing* process is based on an ad hoc instruction set simulator able to trace the execution data flow of each instruction of the test program. Its goal is the generation of independent and small programs able to exactly replicate the behavior of the processor while executing a target instruction. Clearly, each program belonging to the initial test set is fragmented in a huge number of spores. More details can be found in [10].

A first fault simulation is required to analyze the generated spores. This preliminary fault simulation is aimed at determining the fault coverage figure for each generated *spore*. Figure 2.a refers to this process: the result of this step is a *coverage matrix* storing for each *spore* the information about covered faults.

3.2 Sifting

At this point we are left with an inordinate number of spores, of the order of tens of thousands. It would be impractical and wasteful to simply apply all these programs to a faulty processor to diagnose it, so an effort is appropriate to reduce this diagnostic set. This goal is obtained by a *sifting* process, detailed below.

First of all, the fault covering ability of each spore is considered in the context of the entire diagnostic set; the important thing for a spore is not covering a large number of faults, but covering faults which are not detected by other spores: all the spores able to do this have to be retained in the final diagnostic set.

Every fault is detected by a certain number of spores, depending on whether it is easy-to-detect or a hard-to-detect. This leads to the concept of *density* of the fault, that is, the number of spores able to detect it.

The diagnostic capability of a spore is then evaluated with respect to the whole set, using the density concept. Every spore is assigned a preliminary fitness value $f_s(d_F, NF_s)$:

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

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. The spores are then sorted by fitness value in decreasing order. In this way the spores

with higher diagnostic capability are preferred for inclusion in the final diagnostic set. It is possible to note that the highest fitness value equals 1, and it is assigned to one spore covering only one fault of density 1. Finally, starting from the top of this list, only the spores that cover faults not detected by the previous ones are kept, while the others are discarded as redundant. These test programs compose the *initial* test set.

3.3 Diagnostic assessment

Immediately after the sifting phase, the diagnostic ability of the selected *initial* test set is evaluated. This computation has been divided in two steps, graphically shown in figure 2.b. Similarly to the process formalized in [9], a first *coarse* classification is based on the construction of a compact diagnostic tree obtained by processing only the pass/fail information related to each test program included in the initial test set; this information is simply extracted from the coverage matrix. The *binary* tree structure is shown in figure 3.a.

A *fine* classification is then performed for all the equivalence classes (ECs) isolated by the coarse classification and still composed of more than one fault. This second classification is done by using the faulty circuit responses on primary outputs (also called *syndromes*) to build in parallel an *n*-ary tree for each EC to be further divided. The *n*-ary tree structure is in figure 3.b. To reduce the impact of this second fault simulation process aimed at retrieving the faulty syndromes, an incremental approach has been used whereas:

- the *n*-ary tree structure is updated at the end of the fault-simulation of each test program in the initial test set and faults included in ECs of size 1 definitively dropped out from the fault list
- faults not yet classified and not covered by the next test program to be fault simulated are temporarily dropped out from the fault list as their syndrome is not useful for classification.

This process allows the generation of a compact fault dictionary composed of

- pass/fail sequence leading to a *Coarse* EC
- the set of discriminating syndromes for each *Fine* EC.

3.4 Evolutionary improvement

At this phase of the workflow, there is still a set of unsatisfactorily large ECs. Thus, an effort is appropriate to partition these large classes.

To improve the diagnostic ability of the test set we resort to an evolutionary tool called µGP [12]. It is an evolutionary approach to generic optimization problems with a focus on the generation of test programs for microprocessors. It is based on an evolutionary core, an instruction library that allows targeting a specific microprocessor and an external evaluator to provide the core with the necessary feedback.

The evolutionary approach receives the information about the large ECs and generates new assembly programs able to split them. The fitness function provided as feedback to the evolutionary core is the size of the largest EC found using the fine classification. During this phase, the ability of a generated program in dividing the large EC is directly obtained by analyzing the faulty syndrome.

The fine classification tree is directly updated after each program generation. This third part of the diagnostic evaluation process is in figure 2.c.

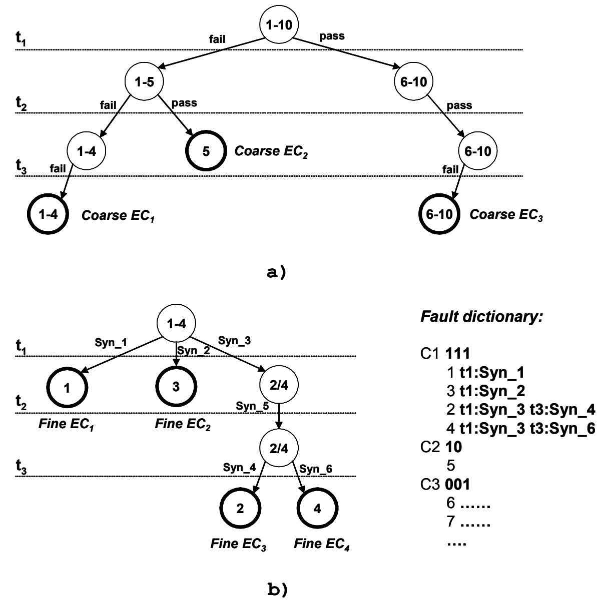

Figure 3: two steps, *coarse* and *fine*, of the diagnostic tree construction and the resulting fault dictionary

This process, using an *n*-ary diagnostic tree instead of a binary one, is totally new with respect to the work reported in [9]. The time it takes can be traded with the quality of the obtained results.

4. Experimental evaluations

The feasibility and the effectiveness of the proposed approach for the software-based diagnosis of processor cores have been proved considering as a case of study an Intel i8051 microcontroller, supposed to be embedded into a SoC.

One of the major problems in testing a SoC is the reduced accessibility of each single embedded core: to improve the observability of the i8051 to be diagnosed, we resorted to the solution proposed in [1] to support the software-based self-test procedure of processor cores. The i8051 core is complemented with a 24 bit multiple input signature register (MISR) connected to its parallel output ports. The faulty signature, or syndrome, stored by the MISR can be read at the end of each test program execution. The synthesized microcontroller, obtained using a generic home-developed library, contains 37,417

equivalent gates, and the collapsed fault list counts 12,642 faults.

Let $D(n)$ be the percentage of faults that are classified into a class of cardinality less than or equal to n by the diagnostic test set. $D(1)$ is the percentage of faults that are uniquely classified; $D(10)$ is the percentage 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.

We started from a post-production test set composed of 8 test programs written by a skilled test engineer, reaching a fault coverage of about 92% on the collapsed list. 7 out of 8 programs aim at covering the ALU faults and are generated by following the deterministic approach described in [11]; the remaining program has been written resorting to the same technique, but it aims at the coverage of those faults in the decoding and control units still not covered. The *sporing* process generates about 60k test programs and the average number of instruction for these programs is 7: each program includes an initial and a final part, required to obtain the syndrome saving. More details about the program structure and *sporing* can be found in [10].

The fault simulation process required to proceed to the *sifting* phase has been done exploiting an in-house developed tool and required about 75 hours on 3 SUN Blade processor-based workstations.

	Post-production test set	*Initial* test set	*Final* test set
Programs [#]	8	7,231	7,266
Test set size [KB]	4	165	177
Max Clock Cycles	1.00M	1.93M	2.02M
$D(1)$ [%]	11.56	35.70	61.93
$D(10)$ [%]	32.90	58.02	84.30

TABLE I: the equivalence class summary for the analyzed processor core

The sifting phase, including the fault simulation, required about 100 hours on the same hardware. And the test set obtained processing coverage matrix is composed of 7,231 test programs (about 12% of the original set).

Table I shows the main characteristics of the three test sets: post-production; initial (sifted); final (enhanced). The rows reports: the number of test programs in the test set; the test set size; the maximum length of a test program in clock cycles; the result of the diagnostic assessment D(1) and D(10). Table II, on the other hand, details the size of the equivalence classes for the three test sets.

It is interesting to note that the initial test set demonstrates a $D(10)$ equal to 58% of the processor faults. The biggest class, composed of 3,755 faults, is the one including those faults detected by all the test programs of the test set. Apart from this class, the greatest class has size 84.

The diagnostic enhancement of the test set eventually consisted in the generation of 35 new test programs. Each generation process is started by evolving an initial population of 20 test programs. The biggest class obtained by this enhancement process has size 1,092.

If the diagnostic test set is considered as a monolithic block to be entirely uploaded and executed on the Automatic Test Equipment (ATE), it is composed of all 7,266 programs and requires 2,020,656 clock cycles. However, if an intelligent/interactive ATE enabling the diagnostic process to be stopped as soon as the fault (or the faults) responsible for the wrong behavior has been individuated, the diagnostic process can be reduced to the execution of an average of 3,762 programs (corresponding to 900,858 clock cycles).

EC size	*Post-production* test set [# fault]	*Sifted* test set [# fault]	*Enhanced* test set [# fault]
1	1,334	4,120	7,148
2	802	872	1,106
3	543	522	546
4	360	264	336
5	255	150	155
6	138	150	180
7	112	154	91
8	80	224	72
9	63	81	36
10	110	160	60
11 – 100	2,335	1,090	720
>100	5,410	3,755	1,092

TABLE II: equivalence class summary for the analyzed processor core

The figures 4 to 6 graphically show fault classification obtained at the end of each phase.

5. Conclusions and future work

A novel automated approach for the generation of software-based diagnostic sets for microprocessors has been presented. It exploits an existing post-production software-based test set and uses the existing Infrastructure IP designed for applying it, thus it is cost-effective and can be seamlessly fit into an existing design flow.

The reported results clearly show the effectiveness of the method on a widely used microprocessor core, thus highlighting the industrial relevance of the approach.

Work is currently under way to improve the workflow, with particular effort upon the reduction of the computational effort required.

6. References

[1] P. Bernardi, M. Rebaudengo, M. Sonza Reorda, “Using Infrastructure IPs to support SW-based Self-Test of Processor Cores”, IEEE International Workshop on Microprocessor Test and Verification, 2004, pp. 22-27

[2] H. Y. Chang, “A Distiguishability criterion to selecting efficient diagnostic tests”, Proc. Spring Joint Computer Conference, 1968, pp. 529-534,

[3] A. D. Friedman, “Fault Detection in Redundant Circuits”, IEEE Trans. Computers, vol. EC-16, February 1967, pp. 99-100

[4] P. Camurati, D. Medina. P. Prinetto, M. Sonza Reorda: “A diagnostic test pattern generation algorithm”, IEEE International Test Conference, 1990, pp. 52-58

[5] T. Griining, U. Mahlstedt, H. Koopmeiners: “DI- ATEST: a fast diagnostic test pattern generator for combinational circuits”, ICCAD-91: IEEE International Conference on Computer Aided Design, Santa Clara, CA (USA), November 1991, pp. 194-197

[6] A. Veneris, R. Chang, M.S. Abadir, M. Amiri, “Fault equivalence and diagnostic test generation using ATPG” IEEE ISCAS 2004, Vol.5, pp. V-221 - V-224

[7] S. Chakravarty, “A Sampling Technique for Diagnostic Fault Simulation”, IEEE International VLSI Test Symposium, 1996, pp. 192-197

[8] N. Jha and S. Gupta, “Testing of Digital Systems”, Cambridge University Press, 2003

[9] Li Chen, S. Dey, “Software-based diagnosis for processors”, Design Automation Conference, 2002, pp. 259-262

[10] 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

[11] N. Kranitis, G. Xenoulis, D. Gizopoulos, A. Paschalis, Y. Zorian, “Low-cost Software-Based Self-Testing of RISC Processor Cores” IEE Proceedings of Computers and Digital Techniques, 2003, Volume 150, Issue 5, pp. 355-360

[12] G. Squillero, “MicroGP – An Evolutionary Assembly Program Generator” Genetic Programming and Evolvable Machines. Springer Science + Business Media B.V. ISSN: 1389-2576

[13] F. Corno, E. Sánchez, M. Sonza Reorda, G. Squillero “Automatic Test Program Generation - a Case Study” IEEE Design & Test, Special issue on Benchmarking for Design and Test, 2004, Volume 21, Issue 2, pp. 102-109

[14] P. Parvathala, K. Maneparambil, W. Lindsay, “FRITS - a microprocessor functional BIST method”, IEEE International Test Conference, 1990, pp. 52-58

[15] I. Pomeranz, S. M. Reddy, “A diagnostic test generation procedure based on test elimination by vector omission for synchronous sequential circuits”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, May 2000, Volume 19, Issue 5, pp. 589-600

[16] V. Boppana, I. Hartanto, W.K. Fuchs, “Full fault dictionary storage based on labeled tree encoding”, IEEE VLSI Test Symposium, 1996, pp.174-179.

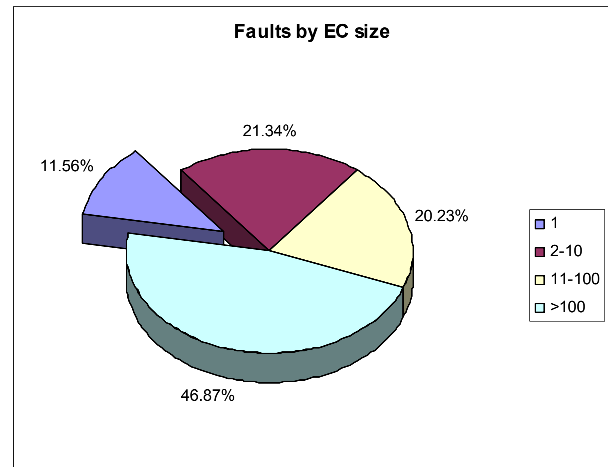

Figure 4: *Post-production* test set diagnostic properties

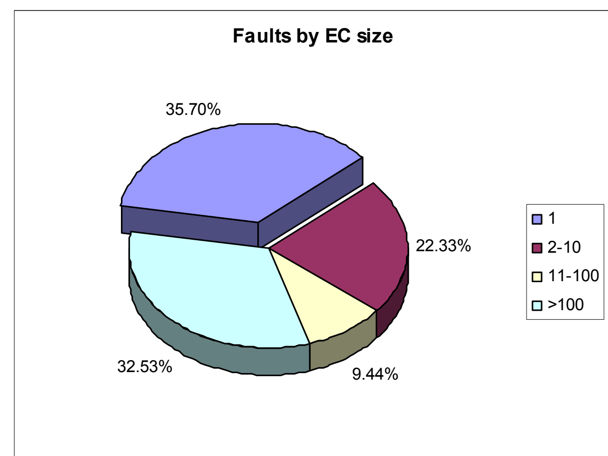

Figure 5: *Initial* test set diagnostic properties

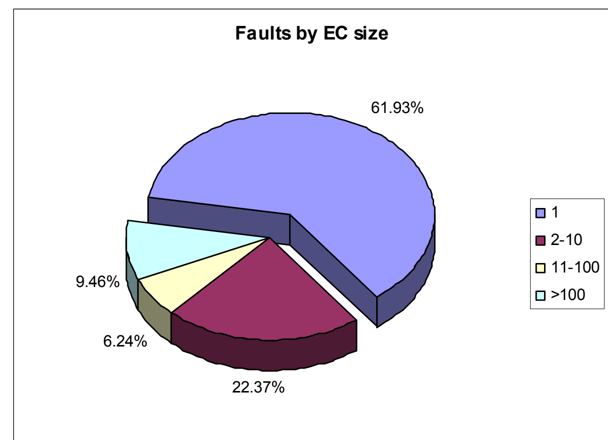

Figure 6: *Final* test set diagnostic properties