Functional Testing of Microprocessors with Graded Fault Coverage

Rajesh Kannah[1]
ATI Research Silicon Valley Inc,
Chennai.
rkannah@ati.com

C.P. Ravikumar
Department of Electrical Engineering,
Indian Institute of Technology,
New Delhi 110016.
rkumar@ee.iitd.ernet.in

Abstract

The goal of this paper is to reduce the test application time for microprocessors. Since functional fault model is used for testing microprocessors, test development time is greatly reduced. But functional test generation leads to a large number of tests. The size of the test set is an important factor, as it determines both the storage for test instructions in the test equipment, as well as the test application time. The problem becomes still more serious when the processor is embedded as a core in a system–on–chip. Hence, in this paper, we try to address the problem of reducing the number of tests. We use the available structural information about the microprocessors to drop some of the functional tests. Some valid assumptions about the faults that are present in the microprocessor, e.g., only single stuck at faults are present, is made to reduce the number of tests. We develop fault–grading concepts and use them to reduce the number of tests. We generate tests for Intel 8086, Motorola 68000 microprocessors using functional testing procedures and reduce the number of tests using our fault grader.

1. Introduction

The rapid advances in very large scale integrated circuit technology have resulted in extremely complex processors. As the size of general and special purpose processors increase rapidly, generating high quality tests for them is becoming a serious problem in industry. Unfortunately, in addition to the complexity of microprocessors, the limited accessibility of their internal logic makes them very difficult to test [2].The classical approach of deriving tests based on the gate or flip–flop level description of microprocessors, is inadequate for several reasons. The large amount of logic, combined with the fact that the microprocessors may not be designed with DFT techniques, makes it impractical to use gate level sequential ATPG to derive tests. Physical failures in integrated circuits result in logic behavior which can not be modeled by stuck–at faults [3]. The most compelling reason is that internal details of commercial microprocessors are rarely made available to the user who must, nevertheless, attempt to derive tests for them. Another serious problem with sequential ATPG is that it takes too much time for test generation.

Thatte and Abraham [4] proposed a graph model at register transfer level to represent microprocessor and used functional level fault models for instruction level test generation. They represented the microprocessor by a set of functions such as i) register decoding function ii) instruction execution iii) data transfer and data storage function and iv) data manipulation functions. A functional fault model is then developed for each of these functions and tests are generated to detect all the faults in the fault model. Brahme and Abraham [2] takes a more detailed view of the instruction execution process and a comprehensive fault model is developed. They had shown that codewords used as data patterns bound to registers result in interesting property that all the faults in instruction execution function can be categorized into three categories. The fault model is general and can be used for any microprocessor and the process of test generation can be easily automated. Tests can be generated without the detailed knowledge of how the instruction execution and control function is implemented.

Although this functional test generation method is efficient in terms of test development time, it results in large number of tests which leads to more test application time. Large number of tests demands a high storage capacity of the testing equipment. Hence, to reduce the number of tests, we use the available structural information to drop some functional tests. We make some valid assumptions about the faults that are present in the microprocessors e.g., only single stuck at faults are present, to reduce the tests. We use

[1] Rajesh Kannah was a student in the M.Tech (Computer Technology) program at IIT Delhi when this work was carried out.

a–priori knowledge of fault occurrence also, to reduce the number of tests. We define fault grading as grading of faults according to probabilities by which they can occur in the circuit. Given the graded faults, we define graded fault coverage (GFC) as summation of probabilities of faults caught by test set to probabilities of all faults in the fault set. Our objective is to find the smallest number of tests, which gives the highest GFC, given the constraint that number of tests should not exceed a limit. As the number of tests will become prohibitive if the microprocessor is embedded in a SOC, even small amount of reduction in the number of tests, will be desirable[5]. As the industry practices are moving towards SOC, reduction of size of test set and hence testing time, becomes more relevant.

In section 2, functional testing of the microprocessors is explained. In section 3, we introduce fault–grading concepts and apply that to reduce the number of tests which leads to reduced test application time. We present the use of structural information to reduce testing time. In section 4, we present the results for Intel 8086 and Motorola 68000 microprocessors. Results presented show that significant reduction in number of tests is possible using fault grading concepts. The conclusion is presented in section 5.

2. Functional Testing for Microprocessors

Following [4], a microprocessor is represented as a graph based on its architecture and instruction set. Every user–accessible register is represented as a node. Let $R = \{R_1, R_2, ..., R_n\}$ be the set of registers and $I = \{I_1, I_2, ..., I_p\}$ be the set of instructions of a sample microprocessor. R includes all the registers which can be explicitly modified by an instruction $I_i \in I$. Let $N = \{N_1, N_2, ..., N_i\}$ be the nodes of the graph. Then the nodes of the graph represent either i) a register or ii) a set of equivalent registers or iii) special nodes IN and OUT. IN and OUT nodes represent communication between microprocessor and external world. IN represents the source of all data/control inputs to microprocessor, while OUT represents the sink of all data/control outputs. The nodes are connected by directed edges. There exists a directed edge from node A to node B if and only if there exists an instruction that causes transfer of data or information from node A to node B.

Faults affecting the microprocessors are classified into the following fault classes [1,4].

1. Addressing faults affecting register decoding function.
2. Addressing function affecting instruction decoding and instruction execution function .
3. Faults in data storage function.
4. Faults in data transfer function.
5. Faults in data manipulation function.

A functional fault model is then developed for each of these functions. Fault models are general, in the sense, it does not assume any particular implementation of microprocessor. Fault model for microprocessor, allows at any given time, the presence of any number of faults but only in any one of the fault classes described above. We are allowing a very general model for microprocessors. Since we allow each function to be faulty in a very general fashion, our assumption is not as restrictive as the classical single stuck–at fault assumption. Tests are generated for each of these functional fault model using test procedures, except data manipulation function for which we are assuming tests are available.

3. Fault Grading of Functional Tests

In the previous section, we introduced the concept of functional test generation for microprocessors. When a conservative functional fault model is used for test generation, test development time is greatly reduced. But such a test generation method will yield a large number of tests. As the size of the test set is large in the case of functional test generation compared to structural test generation, we try to address the problem of reducing the number of tests. The size of the test set is an important factor, as it determines both the storage capacity for tests in the test equipment, as well as the test application time. The problem becomes still more serious when the processor is embedded as a core in a system on chip. A large test set implies that test time is large and often that the IC in question can only be tested on expensive test equipment having very large memories to store these tests. The available structural information about the microprocessors is used to drop some of the functional tests. We make some valid assumptions about the faults that are present in the microprocessor, e.g., only single stuck at faults are present, to reduce the tests. Fault–grading concepts are developed and used to reduce the number of tests.

Fault–grading is the grading of faults according to probabilities of their occurrence in the circuit under test. Given with graded faults, we define graded fault coverage (GFC) as the summation of probabilities of faults caught by test set to probabilities of all faults. Let p_f be the probability of occurrence of fault f in the circuit under test(CUT). Then we define graded fault coverage as follows,

$$\delta t_f = p_f \text{ if test vector t detects fault } f$$
$$= 0 \quad \text{otherwise}$$
$$GFC = \frac{\sum_{t \in T} \sum_{f \in F_T} \delta t_f}{\sum_{f \in F} p_f} \qquad (1)$$

where T is the test set, F_T is the set of faults caught by test set T, and F is set of modeled faults. The definition of GFC reduces to the conventional definition of fault coverage(FC), if p_f is equal to 1. Our objective is to find the smallest number of tests which give the highest GFC, given the constraint that number of tests should not exceed a limit. From the definition of graded fault coverage, it is clear that GFC ≥ FC.

Example: Let f_1, f_2, f_3, f_4, f_5, f_6 be the faults in a CUT. Let p_{f1}= 0.1, p_{f2} = 0.02 , p_{f3} = 0.9, p_{f4} = 0.8, p_{f5} = 0.01, p_{f6} = 0.03. Let t_1, t_2, t_3, t_4 be the four tests. Let t_1 detect (f_1 , f_2, f_5), t_2 detect f_3, t_3 detect f_4, and t_4 detects f_6. Given that required fault coverage ≥ 90%, we require all the four tests. If the required graded fault coverage ≥ 90%, number of tests required are two (t_2, t_3). Hence, using fault grading, we reduce the number of tests by 50%.

Returning to the microprocessor example, we have used fault grading concepts to reduce the number of tests generated for testing the data storage and data transfer function. Once we come up with the functional faults, we can find the graded fault coverage for a given number of tests, we can choose a particular set of tests that maximises the graded fault coverage.

3.1. Modified Algorithm for Testing Register Decoding Function

Under a fault in register decoding function, when decoding R_i may result in one of the following i) no register is decoded ii) A set of registers in addition to or instead of R_i is accessed. The number of tests generated by original functional test procedure can be reduced by using available structural information. We use single stuck at fault assumption in microprocessors to reduce the number of tests. We present the modified register decoding algorithm. This procedure is guaranteed to detect all single stuck at faults in register deciding function.

procedure *ModifiedRegisterDecode*:
begin
Step 1: Initialize queue Q with all the registers so that R_i lies ahead of R_j if and only if $l(R_i) \leq l(R_j)$; Initialize the set A as empty.
Step 2: Delete the register R_i on top of Q and add it to set A.
Step 3: **repeat**
i) For each $R_j \in A$ s.t., HD(i,j) =1 where HD(i,j) be the hamming distance between i & j
WRITE(R_j) with data ONE
ii) WRITE(R_i) with data ZERO
iii) For each $R_j \in A$ s.t., HD(i,j) =1
READ(R_j) s.t., register R_a will be read before register R_b (R_a,$R_b \in$ A), if and only if $l(R_a) \leq l(R_b)$
iv) READ(R_i)
v) Delete R_i from front of Q and add it to set A
until Q = empty.
Step 4: Repeat Steps 1,2, and 3 with complementary data.
end

In this procedure, we limit the number of registers that can be modified, using the single stuck fault assumption. Let R_0, R_1, R_2,...,R_{15} be the registers in a microprocessor. Assume that each register R_k is given 4-bit binary code. Consider testing for register decoding function. In the original procedure when R_3 can be erroneously decoded into any of the registers in register set. In the modified algorithm, R_3 can erroneously decoded into R_1, R_2, R_7, R_{11}.

Lemma 1:

The modified test procedure for register decoding function provides a speed-up of O(n/log n) when n registers are tested.

Proof:

The original procedure will generate WRITE/READ operations on k+1 registers when R_k is added to set A. Thus, the number of WRITE/READ operations generated by this procedure is $2\sum_1^{n-1} 2(k+1) = 2(n^2+n-2)$ i.e., complexity of the test sequence is $O(n^2)$. The modified procedure will have the complexity of $2\sum_1^{n-1} 2(H(k)+1)$, where, H(k) is the number of integers which are at hamming distance of 1 from k. It is easy to see that $H(k) = \lceil \log k \rceil$. ∴The number of WRITE/READ operations generated by modified procedure is $2\sum_1^{n-1} 2(\lceil \log k \rceil + 1)$ i.e, complexity of test sequence is O(nlogn). Hence, the modified test procedure for register decoding function provides a speed-up of O(n/logn) when n registers are tested.

3.2. Modified Algorithm for Testing Instruction Execution

Addressing faults affecting the execution of an instruction I, may cause one or more of the following fault effects:
1. One or more microorders are not activated by the microinstructions of I.
2. Microorders are erroneously activated by the microinstructions of I.
3. A different set of microinstructions is activated instead of, or in addition to, the microinstructions of I.

This fault model is general, as it allows for partial execution of instructions and for execution of "new" instructions, not present in the instruction set of the microprocessor. Missing microorders are generally easy to detect, as any instruction that does not activate all its

microorders will produce incorrect result. To detect the execution of additional microorders, it is essential that the effect of fault shows up as erroneous data in the internal registers. We tackle the problem by associating codewords with each register (except program counter). The property of codewords should satisfy that any single microorder operating on codewords should either produce an noncodeword, or load a register R_i with a codeword cw_j of a different register.

We again use single stuck fault model to reduce the number of tests. We preserve the tests for READ , WRITE sequences generated by *ReadType0* , *ReadType1* , *ReadType2*, *WriteTest* procedures. We make an attempt to drop some of the tests for instruction execution function generated by *InstructionTest* procedure. Instead of reading all the registers before and after executing the instruction, we will read a particular set of registers that will be modified, because of faults in the execution process of the instruction.

procedure *ModifyInstructionTest*
begin
for every instruction I
A={∅}; B={R_0, R_1, R_{n-1}), where n be the number of registers in the processor
begin
From the opcode of the instruction , find the instructions which are at hamming distance (HD)=1 from I and find the registers they will affect. Add these registers in a set A.
for every $R_i \in B$
WRITE (R_i) with cw_i execute I
for every $R_i \in A$
READ(R_i)
B=A;
end
end

Assume there are m instructions, n be the number of registers. The procedure in section 2.4.2.3 generate nm WRITE, nm READ, and m execute instructions. Whereas modified procedure generate $\sum H(k)$ WRITE, $\sum H(k)$ READ, and m execute instructions. Hence, the modified test procedure for instruction execution function provides a speed-up of $(2nm+m)/(2(\sum H(k)+m))$ when m instructions are tested.

3.3. Modified Test for Data Storage and Data Transfer

In this section, we present a method to reduce the number of tests generated for data transfer function. We grade the bridge faults between two lines in a storage element and data transfer path. We use the *geometric* probability distribution function to model this problem. The probability of a bridge fault between i^{th} and j^{th} lines is taken to be $1/2^{|i-j|}$, when $i \neq j$. We take the probability of faults between two *immediate* neighbors to be 0.5. Let I_1, I_2, I_3 , , I_k be a sequence of instructions such that $E(I_1)$, $E(I_2)$,$E(I_3)$, . ,$E(I_k)$ form a directed path from the IN node to the OUT node in the graph model of the microprocessor. Such a sequence is said to be IN/OUT transfer.

We have developed a test procedure that grades the faults in $T(I_1)$, $T(I_2)$, $T(I_3)$ ... $T(I_k)$ and in registers $D(I_1)$, $D(I_2)$, $D(I_3)$,...,$D(I_{k-1})$, where $T(I_k)$ is the transfer path associated with the k^{th} instruction and $D(I_k)$ is the destination register of the instruction I_k . Let each transfer path be w lines wide. A test for the transfer paths and the registers involved in an IN/OUT transfer consists of repeating the IN/OUT transfer for different interconnect test patterns. In the modified procedure, we first find the probability of each fault for the considered set of faults. Then, we find the set of faults that each interconnect test pattern will catch. We finally find the particular set of interconnect test patterns that maximize the graded fault coverage.

procedure *ModifyDataTransfer*
begin
Step 1: Find the probability of each fault in the fault set.
Step 2: Determine the faults detected by each interconnect test pattern.
Step 3: Find the graded fault coverage for different intersconnect test combinations using eqn. 1.
Step 4: Choose the combination of interconnect patterns to be written in internal registers such that graded fault coverage is maximum and number of tests are minimum. Exhaustive algorithm is used to solve. The number of interconnect test patterns are $2(\lceil \log n \rceil +1)$. ∴ Even exhaustive algorithm will need O(n) time.
Step 5: Execute the each IN/OUT transfer with interconnect test patterns obtained from Step 4.
end

4. Results

In this section, results obtained from the functional test generation procedures for Intel 8086, Motorola 68000 microprocessors are presented. We also have generated the reduced test set using modified test generation algorithms. We have used fault grading concepts and the available structural information to reduce the size of the test set. Results for testing READ,WRITE and Data Transfer from register to alu are not presented. For them, we preserve the tests generated by original functional testing procedure.

4.1. Tests for Intel 8086

In table 1, we have shown the results for cases which have GFC ≥ 90%. Total number of tests required to test READ, WRITE, and Data transfer from register to ALU =29143. For a given graded fault coverage, the total number of tests for the microprocessor can be reduced. For example, if the required graded fault coverage is greater than 94%, the total number of tests without fault grading(FG) = 40539 and number of tests with fault grading = 32091 i.e., we reduced the size of the test set by 21%.

Table 1. Results for Intel 8086 microprocessor

Testing function	*With out FG*	*With FG*	*GFC (%)*	*Reduction (%)*
Register decode	532	232	100	56.30
Instruction execution	2890	910	100	68.51
Data transfer	500	300	94.41	40.00
Data transfer	500	400	99.77	20.00
ALU to register	17090	10254	94.41	40.00
ALU to register	17090	13672	99.77	20.00

Table 2. Results for Motorola 68000 microprocessor

Testing function	*With out FG*	*With FG*	*GFC (%)*	*Reduction (%)*
Register decode	716	212	100	70.30
Data transfer	450	225	93.83	50.00
Data transfer	450	300	99.68	33.33
Data transfer	450	375	99.99	16.66
ALU to register	45048	22704	93.83	50.00
ALU to register	45048	30272	99.68	33.33
ALU to register	45048	37840	99.99	16.66

4.2. Tests for Motorola 68000

In table 2, we have shown the results for cases which have GFC ≥ 90%. Total number of tests required to test READ, WRITE, Instruction execution, and Data transfer from register to alu =54667.If the required graded fault coverage is greater than 93%, the size of the test set is reduced by 31% with the use of fault grading.

5. Conclusions

In this paper, functional test generation algorithms for microprocessors are presented. We developed algorithms to reduce testing time of microprocessors using fault grading and the available structural information. The microprocessor is modeled as a graph and represented by a set of functions i) register decoding ii) instruction decoding and instruction execution function iii) data storage and data transfer function and iv) data manipulation function . Tests for faults in the above functions is generated except iv. The complexity of test sequence for register decoding function is reduced from $O(n^2)$ to O(n logn) using single stuck at fault assumption. The number of test vectors for instruction decoding and instruction execution function is reduced using single stuck at fault assumption. The size of the test vector set for data storage and data transfer function is reduced using fault grading. Tests for Intel 8086 and Motorola MC68000 microprocessors are generated using functional testing procedures. We reduced the number of tests generated by the functional testing using our fault grader. We obtained 21% reduction in the size of test set for Intel 8086 and 31% reduction for Motorola 68000.

References:

[1] Abramovici M., Breuer M.A., and Friedman A.D., "Digital systems testing and testable design", Jaico Publishing house, First edition ,1997.

[2] Brahme D., and Abraham J.A., "Functional testing of microprocessors," IEEE transactions on Computers, vol c-33, pp 475 -485, June 1984.

[3] Galiay J., Crozet Y., and Vergniault M., "Physical versus logical fault models in MOS LSI circuits", IEEE transactions on computers, vol c-29 ,pp 527 -531, June 1980.

[4] Thatte S.M.,and Abraham J.A., "Test generation of microprocessors", IEEE transactions on computers, vol c-29 ,pp 429 -441, June 1980 .

[5] Zorian Y., Marinnisen E.K., and Dey S., "Testing embedded core-based system chips", IEEE Computer, pp. 52-59 , June 1999.