Automatic Functional Test Program Generation for Microprocessors

Chen-Shang Lin and Hong-Fa Ho

Department of Electrical Engineering
National Taiwan University
Taipei 10764, Taiwan

ABSTRACT

A new algorithm, O-algorithm, for automatic test program generation of microprocessors in a user environment is presented. Specifically, to eliminate the redundant tests, a weighted-digraph model is used to model the signal flow of the general microprocessors. Improved functional fault models of microprocessors are derived from Turing machine model. The O-algorithm is then constructed based on the signal flow model and functional fault models. The complexity of our algorithm is better than [6]. Moreover, the simulation had shown that the fault coverage is better than 97%.

1.INTRODUCTION

Microprocessors are extremely versatile and are hence widely used in many complex systems. However, the test of microprocessors is a nontrivial problem. Especially in the user environment, the test of microprocessors is complicated by the fact that the detail circuit information is not available. As a result, the classical gate-level test generation methods simply can not be applied in this situation. A more feasible approach is to generate test program based on the functional-level information which is available to the users.

Several algorithms [2-6] had been developed to generate test program for microprocessors in the functional level. Thatte, Brahme and Abraham [2,6] proposed a graph model for microprocessors at the register transfer level. However, the fault models were restricted to data path and its associated control function.

In this paper, a new algorithm, O-algorithm, for automatic test program generation of microprocessors is proposed. Specifically, to eliminate the redundant tests, a weighted-digraph model is used to model the signal flow of the general microprocessors. Improved functional fault models of microprocessors, such as those of register decoding function, data register function, and I/O pin function are derived from Turing machine model. These fault models cover more faults than [2,6]. The O-algorithm is then constructed based on the signal flow model and functional fault models. The complexity of our algorithm is better than [6].

In the next section, the signal flow model of microprocessors is described. The improved functional fault models are discussed in Section 3. Based on the signal flow model and fault models, a new O-algorithm for generating test program is then developed. The O-algorithm is described in Section 4 and its complexity is briefly discussed in Section 5. Then the configuration of the automatic test program system based on O-algorithm is described and the applications of the system on MCS8048 and a subset of Intel 8086 are shown in Section 6.

2.SIGNAL FLOW MODEL FOR MICROPROCESSORS

In order to test microprocessors systematically, the signal flow model for microprocessors must first be established. Let RS = {R1,R2,...,Rn} denote the set of distinct registers in a microprocessor. RS does not include on-chip memory such as RAM, cache memory or program ROM. Let IS = {I1,I2,...,Ip} denote the set of distinct instructions of a microprocessor. The signal flow of a microprocessor can be modeled as a weighted digraph G = (V,E,g), where V is the set of vertices (or nodes) comprised by the set RS , the vertice IN for input ports , and the vertice OUT for output ports; E is the set of edges in G and E = {e | e is an information flow of instruction Ii between nodes, for all Ii in IS }; and the function g in G is a weighted function from V to the set of pairs of integers which will be defined latter. In other words, the vertices in G are registers in a microprocessor and the edges (or links) stand for the information flow including data flows and/or address flow of instructions among registers.

An instruction Ii in IS is a set of paths in G. S(Ii) denotes the set of source vertice(s) of instruction Ii, and D(Ii) denotes the set of sink (destination) vertice(s) of instruction Ii. All instructions in IS are classified into three types: type -T for data Transfer instructions, type-B for Branch instructions, and type-M for data Manipulation instructions[2].

Let P(u,v) be the set of directed paths from node u to node v in G, and p(u,v) be a path in P(u,v).

DEFINITION 1: Let Ii be of type-T or type-B. The function NI(p(w,v)) is the number of distinct Ii in which at least one connected path of Ii are in p(w,v).

Notation <I1,I2,...,Ii,Ij,...,Ia> denotes that the instructions will be executed sequentially. Notation [S1|S2|...|Sm] denotes that microinstructions S1, S2,...,Sm are executed concurrently.

DEFINITION 2: Controllability of Vi is CY(Vi) = min[NI(p(IN,Vi))], for all p(IN,Vi) in P(IN,Vi).

DEFINITION 3: Observability of Vi is OY(Vi) = min[NI(p(Vi,OUT))], for all p(Vi,OUT) in P(Vi,OUT).

For example, if there exist three instructions, Ix:"MOV Rx<-Ry" , Iy:"MOV Ry<-#d", and Iz:"PUSH Rx" only, then <Iy,Ix> is the only way to control the state (or value) of register Rx. Hence P(IN,Rx) = { <Iy,Ix> }, and <Iy,Ix> is the only element in

CH2540-3/88/0000/0605$01.00 © 1988 IEEE

P(IN,Rx). Thus p(IN,Rx) = <Iy,Ix>, and NI(p(IN,Rx)) = 2. Consequently, the controllability of Rx, CY(Rx), is 2. On the other hand, if Iz = "PUSH Rx" is the only instruction that can be used to observe the value of vertice Rx. Then P(Rx,OUT) = {"PUSH Rx"} and NI(p(Rx,OUT)) = 1. Thus the observability of Rx, OY(Rx), is equal to one.

DEFINITION 4: Dom_CY(x) (Dom_OY(x)) is the set of vertices in V such that the controllability (observability) of each vertice in the set is exactly x.

From the definitions of Dom_CY() and Dom_OY(), G can be levelized based on the CY and OY values respectively. The levelized G based on CY is called G_cy, and that based on OY is G_oy. The graph G_cy and G_oy, are termed as *onionskin graphs*. Each Dom_OY() or Dom_CY() is a skin of the onions. The outermost skin of G_cy is the set of vertices with CY = 0 and the outermost skin of G_oy is the set of vertices with OY = 0.

The reason of associating both CY and OY values with each register is that these values may be different which affects the test sequence. Take the segment register ES of Intel 8086 as an example, CY(ES)=2 and OY(ES)=1. The controlling and observing the register must be considered separately.

With the levelized graphs, the properties of READ() and WRITE() can be discussed. Let Vi be an element of RS, WRITE(Vi) be an instruction which loads the register Vi with a value from input port and READ(Vi) be an instruction which moves the value of register Vi to the output port.

PROPERTY 1: For a vertice Vi with CY(Vi) greater than or equal to 1, WRITE(Vi) must be by way of one or more vertices in one or more Dom_CY(CY(Vj)), where CY(Vj) < CY(Vi).

PROPERTY 2: For a vertice Vi with OY(Vi) greater than or equal to 1, READ(Vi) must be by way of one or more vertices in one or more Dom_OY(OY(Vj)), where OY(Vj) < OY(Vi).

The above two properties will be useful to schedule the test sequence.

3. FUNCTIONAL LEVEL FAULT MODELS

3.1. Fault Model for Microprocessors

A Turing machine model is employed to describe the faulty situations of microprocessors. Let the microprocessor be a Turing machine[9] of the quadruple (K,Σ,δ,s), where K is a finite set of states, excluding the halt state h; Σ is a set of signals which include instructions, input signals, and output signals; s is the initial state which is an element of K; δ is a function from $K \times \Sigma$ to $(K \cup \{h\}) \times (\Sigma \cup \{L,R\})$. K contains the states of memory elements (registers). The machine takes a varying number of clock cycles to perform the function δ.

A configuration of Turing machine $M = (K,\Sigma,\delta,s)$ is a member of $(K\cup\{h\}) \times \Sigma^* \times \Sigma \times (\Sigma^*(\Sigma-\{\#\})\cup\{e\})$, where the '*' denotes that its front item repeats zero, one, or more than one times, '#' is the blank symbol, and 'e' is the end marker. The first part of configuration represents the current state of microprocessor. The second part of configuration stands for the track of machine behavior, where the track includes the executed instruction sequence, corresponding pin action sequences, and the number of clock cycles used by each input signals. The third part is the current input signal. The current input signal can be either an instruction or input signal(s). The last part stands for the future input signals.

Let (q1,w1,a1,u1) and (q2,w2,a2,u2) be two configurations of a Turing machine. Then (q1,w1,a1,u1) ->m (q2,w2,a2,u2) is the yield in one step, if and only if, for some a2 in $\Sigma \cup \{L,R\}$, δ(q1,a1) = (q2,a2), where L is left and R is right. The yields in more than one steps are denoted as: (q1,w1,a1,u1) ->* (q2,w2,a2,u2).

DEFINITION 5: Let Turing machine M be (K,Σ,δ,s). Assume that the given configuration is (q1,w1,a1,u1), and the fault-free yield in one step is (q2,w2,a2,u2). i.e. (q1,w1,a1,u1) ->m (q2,w2,a2,u2). If the configuration of yield in one step is (q1,w1,a1,u1) ->m (q,w,a,u) with q != q2 ("!=" stands for "is not equal to"), w != w2, a != a2, or u != u2, then the machine M is said to be faulty.

The machine M is faulty if and only if the function δ of M is faulty, except two faulty functions cancel each other out. The fault models of microprocessors are constructed based on δ function. The fault effects are observed from Σ. The fault which can not be observable is called a redundant fault. The redundant fault is not considered hereafter.

The function δ of M can be divided into six parts: instruction execution function δi, data register decoding function δd, data transfer function δt, data register function δr, data manipulation function δm, pin function δp. Hence δ can be expressed as follows: $\delta = \delta i \times \delta d \times \delta t \times \delta r \times \delta m \times \delta p$

If one or more parts of δ described above are faulty, then δ is faulty. For convenience, δo will stand for the fault-free parts of δ function.

3.2. Instruction Execution Process

In a microprogrammed microprocessor, instruction execution process can be represented as follows: Ii = < M1,M2,...,Mi,...,Mn > and Mi = [Si1 | Si2 | ... | Sij | ... | Sik], where Mi is the microinstruction, Sij is the microorder[2], n is the length of microprogram for Ii, k is the length of control word for M. On the other hand, if microprocessor is hardwired, an instruction can be viewed as: Ii = [S1 | S2 | ... | Sk].

An instruction sensitive fault is a permanent fault that appears with a given instruction Ii, Ii in IS. A fault sensitive instruction is an instruction associated with a given fault(s). Let C1 = (q1,w1,a1,u1) , C2 = (q2,w2,a2,u2), and C1 ->m C2, so that δ(q1,a1) = (q2,a2). Assume δi in δ of microprocessor is faulty, δi becomes f_δi and f_δi != δi. Then δ(q1,a1) is equal to either

(q2,a2) if a1 is not the fault sensitive instruction or

(q3,a3) if a1 is the fault sensitive instruction,

where q3 != q2 and/or a3 != a2. Therefore, any fault in δi must be an instruction sensitive fault.

For Ii in IS, and F(Ii) denotes faulty instruction. Then F(Ii) = Ii + π_a - π_m +ε; where π_m is the subset of Ii which is inactive in the faulty instruction, but should be active in the fault-free instruction; π_a is the subset of {IS - Ii} which is active in addition to the fault-free instruction; and ε is the set of faults that is not in IS. π_a, π_m, and ε are instruction sensitive faults of Ii. If $\pi_a = \pi_m$ = NOP (No operation) and ε is an empty set then F(Ii) = Ii, Ii is fault-free.

THEOREM 1: Let the test sequence be <...Ii,Ij,Ik,...>, and Ij be the fault sensitive instruction of a fault Fx. Assume that the instructions before Ij is fault-free. Then the instruction sensitive fault Fx is undetectable, if Fx is an additional operation fault consisting of the subset of the previous instructions which last changed those registers affected by Fx[11].

3.3.Data Path Units

The data paths of the general microprocessors consist of various functional units. Here only the improved fault models of the following two functions are described: the register decoding function and the data register function. The other functions of data path can be found in [11].

Let the function of register decoding be a one-to-one and onto function fd : Rsi -> Rss, where Rsi is the ordered set of input-selected registers, Rss is the ordered set of selected registers defined by fd. Note that both Rsi and Rss are the power sets of RS. Let δ(q1,a1) = δd X δo(q1,a1) and f_δd denote the faulty δd.

Assume that Rss = { rs1,rs2,...,rsn }, and Rsi = { ri1,ri2,...,rin }. We can discuss as follows: CASE 1: Rss = Rsi, which means that rs1 = ri1, rs2 = ri2, ..., rsn = rin; CASE 2: Rss != Rsi, which means that either Rss and Rsi are different or elements in both Rss and Rsi are the same but differ in order. CASE 1 is the fault-free case, the latter situation of CASE 2 is called the pseudo fault, and the first situation of CASE 2 is the faulty case. Then the following results can be shown[11].

THEOREM 2: A fault of fd is undetectable if and only if the faulty fd remains a one-to-one and onto function.

COROLLARY 1: Pseudo fault is undetectable.

COROLLARY 2: A fault of fd is detectable if and only if fd is not a one-to-one function or not an onto function.

4.TEST GENERATION PROCEDURES

Based on the above discussions, a new algorithm for automatic test program generation of microprocessors can be developed. The new algorithm arranges the test sequence according the onion graphs described in Section 2. Hence our algorithm is called O-algorithm.

The basic operations of testing a given circuit are to control and to observe its states. For microprocessors, these two operations are WRITE() and READ(), respectively. Therefore, in the testing of microprocessors, it is necessary to be certain the validity of WRITE() and READ() before other functions can be tested with these two basic operations. To validate WRITE() and READ() without redundancy, the levelization of onion graphs is employed in our algorithm.

From the above discussions, O-algorithm is divided into nine procedures as shown in Fig.1. The first procedure is the onionning procedure. Onionning procedure calculates the CY value and OY value for each vertice in the directed graph G. Meanwhile, the instruction WRITE() and READ() are found and recorded. All of these results, CY value, OY value, READ(), and WRITE(), will be used by subsequent procedures. The second procedure of O-algorithm is to generate test program for testing WRITE() and READ() in a sequence based on CY and OY values.

The first step is to verify WRITE(Dom_CY(1)) by a set of core instruction {Iv}, which consists of "MOV R,#data", "CMP Ri,Rj", and "BEQ" or equivalent instructions. Procedure 2A is designed to test the missing operation of {Iv} in Dom_CY(). All instructions in {Iv}} should be tested to assure there is no missing operation. There are, however, other additional operation faults. Procedure 2B is designed to generate test program which can detect part of additional operation fault of {Iv}, when Procedure 2A has completed successfully. If the microprocessor is tested through Procedure 2A and 2B without any detectable fault, the instructions in {Iv} are fault-free in Dom_CY(1). Base on this result, further testing can be carried out.

Then procedures can be developed to generate test for all WRITE() operations. Procedure 2C to Procedure 2F are designed to detect faults of WRITE() for all vertices from CY=2 to maximum CY value. In these steps, READ() has not yet been tested, thus these vertices are observed by using the instructions "CMP" and "BEQ" in {Iv}, which have been tested during Procedure 2A and 2B.

Next step of Procedure 2 is to generate test for READ() with the verfied WRITE(). Procedure 2G and 2H are designed to detect faults of READ() for all vertices from OY=1 to maximum OY value. After the successful test of Procedure 2G and 2H, READ() is fault-free. Procedure 2I and 2J are designed to detect additional operation fault of "CMP" and "BEQ" in all vertices except those in Dom_CY(1).

After the testing of Procedure 2, READ() and WRITE() are fault-free. Thus test can be designed to examine the functions of microprocessor by using WRITE() to control the states first, followed by activating the function under test, and then using READ() to observe the states of vertices. The remaining procedures of O-algorithm are to generate test program for various functions of microprocessors with the verified WRITE() and READ(). These procedures can be found in [11] and will not be discussed in this paper.

5.COMPLEXITY OF O-ALGORITHM

The complexity of the O-algorithm will be summarized. First let us define the following notations:

Nr : total number of distinct registers in microprocessor,

Nis : total number of instructions in IS,

BWD : bit width of a data word in microprocessor.

All of the above parameters can be obtained directly from the specification of microprocessor.

The comparison of complexity between [6] and O-algorithm is shown in Table 1. The first term in the complexity is for the test generation of data register function and data transfer function. It can be seen that O-algorithm is more efficient than [6] in this aspect. This is because of the elimination of redundant testing of linked faults[6] in the test of READ(). The second term in the complexity is for the test generation of all instructions of a microprocessor. Two algorithms have the same complexity in this aspect.

Algorithms	Complexity
Brahme & Abraham(1984)[6]	Nr^4*log(BWD) + Nr*Nis
O-algorithm [11]	Nr^2*log(BWD) + Nr*Nis

Table 1. Comparison of Complexity

6.SYSTEM OVERVIEW AND RESULTS OF SIMULATION

The software system had been implemented based on O-algorithm. The system was implemented in C language under UNIX operation system on VAX-8200. The system is composed of around 10000 lines of C codes, which consists of three parts. They are the test program generator "GEN" which accepts the specifications such as instruction set of a microprocessor and generates test program in object code according to O-algorithm, a functional fault simulator "SIM" which simulates the behavior of the target

microprocessor, and comparison & statistics utilities, COM&STAT.

The Intel 8086 microprocessor was chosen as the experimental vehicle because of its popularity and the completeness of its instruction set. The maximum CY value of Intel 8086 is two, and the maximum OY value is also two. Because the full instruction set of Intel 8086 is too large for functional test program generation and functional fault simulation, a subset of instructions were selected. These selected instructions include the essential instructions such as "MOV Ri,#data", "CMP Ri,Rj", "BEQ", all move instructions with various addressing modes, and integer arithmetic/logic instructions.

A total of 9332 fault units including single and multiple faults were simulated in functional level. For single faults, 8132 fault units were randomly selected, and for multiple faults, 1200 fault units were selected among which 1106 multiple faults were selected from data register and data transfer paths, 30 were selected from data manipulation function and register decoding function, and 64 were selected from data path and instruction function. The overall distribution of faults is as follow: approximate 80 percent from data register and data transfer paths, 7.4 percent from register decoding function, and 12.6 percent from all others.

Among the selected 9332 fault units, 9063 fault units of them were detected by the functional test program generated by O-algorithm. Thus the test program detected 97.11 percent of all selected fault units.

The gate-level fault simulation had also been carried out for MCS8048 to validate the effectiveness of O-algorithm which is based on functional-level models. A total of 100 faults of data path were simulated. And all of them were detected by our algorithm. The reason that only 100 gate-level faults were simulated is that the gate-level fault simulation for a microprocessor is extremely time-consuming.

7.CONCLUSIONS

To solve the problem of testing microprocessor in a user environment, a new algorithm, O-algorithm, for automatic test program generation of microprocessors had been presented. Specifically, to eliminate the redundant tests, a weighted- digraph model had been used to model the signal flow of the general microprocessors. Improved functional fault models of microprocessors, such as those of the register decoding function, data register function, and I/O pin function, had been derived from Turing machine model. The O-algorithm had been then constructed based on the signal flow model and functional fault models. The complexity of our algorithm had been shown to be better than [6]. The simulation had also demonstrated that our algorithm had achieved 97% estimated fault coverage.

REFERENCES

[1] C. Timoc, F. Stott, L.Hess, "A Novel Approach to Test Generation for VLSI", *The Proceedings of COMPCON S'82*, 1982, pp.78-86.

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

[3] Tonysheng Lin, Stephen Y.H.Su, "The S-Algorithm:A Promising Solution for Systematic Functional Test Generation," *IEEE Transactions on Computer-Aided-Design*, Vol.CAD-4, No.3, July 1985.

[4] C. Bellon, R.Velazco, "Hardware and Software Tools for Microprocessor Functional Test", *IEEE International Test Conference*, 1984, pp.804-810.

[5] C.Timoc, F.Stott, K.Wickman, and L. Hess, "Adaptive self-test for a microprocessor", *IEEE International Test Conference*, 1983, pp.701-703.

[6] Dhananjay Brahme, Jacob A. Abraham,"Functional Testing of Microprocessors," *IEEE Transactions on Computers*, Vol.C-33,No.6, June 1984, pp.475-485.

[7] Kyushik Son and James Y.O. Fong, "Automatic Behavioral Test Generation", *IEEE Test Conf.* 1982, pp.161-165.

[8] M.S.Abadir, H.K.Reghbati, "Functional Test Generation for LSI Circuits Described *by Binary Decision Diagrams," International Test Conference* 1985, pp.483-492.

[9] Harry R. Lewis, Christos H. Papadimitrion, *"Elements of the Theory of Computation"*, 1981, Prentice-Hall Inc. p52.

[10] C. Bellon, A.Liothin, S.Sadier, G.Saucier, R.Velazco, F.Grillot, M.Issenman, "Automatic Generation of Microprocessor Test Program", *IEEE Design Automation Conference*, 1982,pp.566-573

[11] Hong-Fa Ho, *"Automatic Functional Test Program Generation for Microprocessors"*, Master Thesis, Institute of Electrical Engineering, National Taiwan University, June 1987.


```
INPUT:        a weighted digraph G with the specification of
              microprocessor.
OUTPUT:       functional test program for microprocessor.
begin
Procedure 1 :Onionning, Construct G_cy and G_oy.
Procedure 2 : Generate test program to test {Iv},
              WRITE(RS), and READ(RS).
Procedure 3 : Generate test program to test register decoding
              function.
Procedure 4 : Generate test program to test data transfer
              function and data register  function.
Procedure 5: Generate test program to test data
              manipulation function.
Procedure 6 : Generate test program to test addressing mode
              function.
Procedure 7: Generate test program to test input pin
              functions.
Procedure 8 :Generate test program to test all instructions.
Procedure 9 :Generate test program to test program counter
              and address bus functions.
end
```

Fig.1. O-algorithm