Test Generation for Speed-Independent Asynchronous Circuits with Undetectable Faults Identification

*Eunjung Oh, *Dong-Ik Lee, and **Ho-Yong Choi

*Dept. of Info. & Comm., Kwang-Ju Institute of Science & Technology, 1 Oryong-dong Buk-gu Kwang-Ju, 500-712, Korea
Tel: +82-62-970-2267, Fax: +82-62-970-2204
E-mail: {Eunjung, dilee}@csrl.kjist.ac.kr

**School of Electrical & Electronics Eng., Chungbuk National Univ., 48 Gaesin-dong Heungdeuk-gu Cheong-Ju Chungbuk, 361-763, Korea
Tel: +82-431-261-3231, Fax:+82-431-274-9614
E-mail: hychoi@cbucc.chungbuk.ac.kr

Abstract: In this paper, we propose a test pattern generation algorithm on the basis of the identification of undetectable faults for Speed-Independent(SI) asynchronous control circuits. The proposed methodology generates tests from the specification of a target circuit, which describes the behavior of the circuit in the form of Signal Transition Graph (STG). The proposed identification method uses only topological information of a target circuit and reachability information of a fault-free circuit, which is generated in the form of Binary Decision Diagram(BDD) during pre-processing.

Experimental results show that high fault coverage over single input stuck-at fault model is obtained for several synthesized SI circuits and the use of the identification process as a preprocessing decreases execution time of the proposed test generation with negligible costs.

1. Introduction

In general, testing of an asynchronous circuit is considered harder than that of synchronous circuits due to the following reasons.

- The absence of a global clock
- Redundant logic added to remove races and hazards in logic synthesis
- Large state space due to the concurrency characteristic

For the above reasons, direct adaptation of the well-developed synchronous testing methodologies to the asynchronous circuits cannot guarantee the validity and the efficiency. Furthermore, an arbitrary modification of a circuit might be hazardous in the asynchronous circuits because of those modifications would cause unexpected hazards and races.

There are two approaches to solve these complicated problems in the testing of asynchronous circuits. One is how to modify and adopt the synchronous testing methodologies for the asynchronous one[1-5]. The other is testability of an asynchronous circuit[6]. Both approaches do not consider the undetectable faults or the redundant faults of asynchronous circuits. Test generation for those faults turn out futile at last with huge amount of computation and long execution time. Without the identification of those faults in advance of the test generation, any testing methodology cannot exhibit its merit thoroughly.

In this paper, we try to lessen those difficulties by adopting followings.

1. Fundamental mode assumption[7]: We assume that tests can be applied and the results are observable on only stable states. Though normal operation of an SI circuit does not follow the assumption, this is realistic and reasonable in test generation and application phases.
2. Undetectable fault identification: The identification is performed prior to the test generation. The results are passed to the test generator. The test generator does not need to make any effort to find tests for the identified undetectable faults.
3. Test generation from the specification: We follow signal transition sequences described in a specification of a circuit. The test generator takes a prefix of the sequences as tests.

With the aboves, we have proposed an identification of undetectable faults of asynchronous circuits and have proposed a test generation method based on a specification. In this paper, we only consider a specific class of asynchronous circuits, an SI circuit with ACGpER signal network[8], which is depicted in Figure 1. The target fault model, single input stuck-at fault model, is depicted in Figure 2-(a).

2. Identification of Undetectable Fault

A fault is undetectable when the fault does not affect the behavior of a circuit under test, which is defined in the specification. Undetectable faults reveal at the end of test generation process after long execution time. Without the identification of those faults in advance of the test generation, a test generator gives useless effort to discover those faults.

Our identification method finds undetectable faults using both topological information and reachability information of the given circuit, which is a byproduct of test generation. In the identification process, all the possible faults need not to be considered due to the inherent characteristic of the ACGpER signal network. Some of faults can be excluded by simple investigation of net-list. The followings are categories of faults which are excluded in the identification process.

1. All the output stuck-at faults: SI circuits has self-checking property under the output-stuck at fault model depicted in Figure 2-(b), that is a circuit halts in the presence of any output stuck-at fault[6].

2. Input stuck-at faults equivalent to output stuck-at faults: By the simple equivalent fault collapsing analysis, input stuck-at faults equivalent to output stuck-at faults can be identified. For example, stuck-at-0(1) faults on an input of an AND(OR) gate is equivalent to stuck-at-0(1) faults on the output of the gate.

3. Faults on wires branching out to primary outputs: A fault on a wire branching out to a primary output is always detectable because value of primary outputs will change at least once.

4. Faults on input wires of C-elements: When an output of a C-element changes at least once, a fault on input wires of C-elements can be detected because the output of the C-element does not change forever.

In consequence, only faults occurred on inputs of set and reset regions, depicted in Figure 1, need to be taken into accounts in the identification and we classify these faults as potentially undetectable faults.

When a fault occurs in the ACGpER signal network, an output variable affected by the fault is uniquely determined due to the characteristic of signal network. Assume that a stuck at-*c* fault occurs on wire *x* and the function of the unique output variable is *F*. The function for the fault-free(faulty) circuit $F_g(F_f)$ of *F* can be derived by substituting *c*'(*c*) for *x*. The output of a C-element is defined as *ab*+(*a*+*b*)*c*', where *a* and *b* is inputs of a C-element and *c*' is the previous value of the C-element. Since the output of the network is the output of the C-element, the output function appears

$F_g = F(x \leftarrow c') = \mathrm{A}+\mathrm{B}\bullet F_p$ and

$F_f = F(x \leftarrow c) = \mathrm{C}+\mathrm{D}\bullet F_p$,

where F_p is previous value of *F*. A(B) and C(D) are product(sum) of two input variable of the corresponding C-element.

When $F_g \oplus F_f = 0$, the fault is redundant. Otherwise, the equation holds at least one of the following conditions

C1: $\mathrm{A} \oplus \mathrm{C} = 1$ and $\mathrm{B} = \mathrm{D} = 0$

C2: $F_p = 0$ and $\mathrm{A} \oplus \mathrm{C} = 1$

C3: $F_p = 1$ and $(\mathrm{A}+\mathrm{B}) \oplus (\mathrm{C}+\mathrm{D}) = 1$

In case of C1, the current state does not depend on the previous state. C2 and C3 are dependent on the previous state for the current state.

When a relevant variable assignment of variables consisting of A, B, C, and D are achieved without conflicts, the identification proceeds to check whether the assignment appeared in the reachable state space of specification. The reason for the reachability analysis is that our proposed test generation considers only reachable state space given by the specification. Thus, the identification can be used for redundant fault classification when the reachability analysis is omitted. Figure 3 describes the whole procedure for the identification.

3. A Testing Framework for Speed-Independent Circuits

In the following, we introduce a testing methodology for SI circuits.

1. SI circuit modeling: The proposed methodology uses the same gate modeling as that of [1]. In circuit modeling, only the specifications are taken into account rather than actual circuits considered in [1]. That is, we regard the set of reachable states of a circuit as that of the specification. Thus state oscillations and critical races, which could occur on states not defined in the specification, do not need to be considered.

2. Specified sequence generation: Sequences are obtained by traversing the given specification. Only stable states in the sequence are used for test generation, thus no test invalidation problems, which stem from unstable states and unreachable states occur. Figure 4 depicts an example of generation of the specified sequence.

3. Undetectable fault identification: All undetectable faults including redundant faults are identified by Algorithm 1in Figure 3.

4. Test generation: The specified sequences are applied sequentially to a fault-free circuit and a faulty circuit until the outputs of both circuits become different. The applied part of the sequences is the test for the fault.

Algorithm2 in Figure 5 depicts the proposed testing methodology.

4. Experimental Results

The proposed methodology has been implemented with C language and CUDD 2.1.1 package[9]. Benchmarks have been automatically synthesized by the well-known asynchronous logic synthesizer, Petrify[10]. The experiments have been performed on Sun Sparc Ultra-I. Two kinds of experiments are performed over the same set of benchmarks: (1) without the undetectable fault identification and (2) with the identification process. Results are summarized in Table 1 and Table 2, respectively.

Modeling time and ATPG time in Table 1 show the CPU time used for fault-free circuit modeling and test generation for all possible single input stuck-at faults. As shown in Table 1, high fault coverage can be achieved for most of examples without exploring the entire state space of the circuit.

As mentioned previously, the state space of an asynchronous circuit is quite different from that of a synchronous circuit. For example, the upper bound of the state space of "full" with two state holding elements is $2^{2(\text{latch})+2(\text{input})}$ for synchronous case and $2^{2(\text{input})+36(\text{internal})}$ for asynchronous case. This fact explains longer execution time of our experiments as compare to that of synchronous case for the same size of circuits.

Potential U fault and U fault in Table 2 are the number of potentially undetectable faults and identified undetectable faults, respectively. As shown in Table 2, execution time of the identification is negligible in most cases. The sixth column, Time Saving, in Table2 is achieved at the cost of the fourth column, Identification Time. Even though the benchmarks used in these experiments are relatively small, we realize that the portion of undetectable faults is not negligible in an asynchronous circuit. As circuits become lager and more complicated, this feature will be worsened. On those circumstances, the undetectable fault identification before the test generation exhibits its merits definitely.

5. Conclusions

We have proposed a testing methodology for Speed-Independent circuits with an identification of undetectable faults as a pre-processing. The proposed testing methodology generates tests based on the specification of a target circuit to exploit the normal operation of the circuit. Target faults were detected by the comparison of outputs of the fault-free and faulty circuits. For the purpose of efficient comparison, reachability information of both circuits is saved in the form of BDD and operations are conducted by BDD manipulations.

The proposed identification makes use of only topological information of the target circuit and reachability information of the fault-free circuit, which was generated during preprocessing. Thus, it can reduce the execution with negligible costs. Even though the benchmarks used in the experiments were relatively small, we showed the indispensability of the identification for efficient test generation.

Since the proposed testing methodology assumed that a specification for a target circuit was given, the set of reachable states of the circuit was restricted as that of the specification. When we consider the set of all the possible reachable state of the circuit, the identification of undetectable faults and testing methodologies might be changed. Those studies are left as future works.

Acknowledgements

This work has been supported by the KAIST/K-JIST IT-21 Initiative in BK21 of Ministry of Education.

References

[1] Oriol Roig i Mansilla, "Formal verification and testing of asynchronous circuits," PhD thesis, Universitat Politecnica de Catalunya, 1997.

[2] Marco A. Pẽna, Enric Pastor, and Jordi Cortadella, "Symbolic techniques for the automatic test pattern generation for Speed-Independent circuits," Technical Report RR-97-04, UPC/DAC, 1997.

[3] Oleg Alexandrovich Petlin, "Design for testability of asynchronous VLSI circuits," PhD thesis, Univ. of Manchester, 1996.

[4] Savita Banerjee, Srimat T. Chakradhar, and Rabindra K. Roy, "Synchronous test generation model for asynchronous circuits," In the 9th International Conference on VLSI Design, 1996.

[5] Ming-Der Shieh, "Design and synthesis of testable asynchronous sequential logic circuits," PhD thesis, Michigan State Univ., 1993.

[6] Peter A. Beerel, "CAD tools for the synthesis, verification, and testability of robust asynchronous circuits," PhD thesis, Stanford Univ., 1994.

[7] Stephen H. Unger, "Asynchronous sequential switching circuits," Wiley-Interscience, 1969.

[8] Alex Semenov, and et. al., "Partial order based approach to synthesis of speed-independent circuits," In Proc. of Async'97, 1997.

[9] Fabio Somenzi, "CUDD: CU decision diagram package release 2.1.2.," Univ. of Colorado at Boulder, 1997.

[10] Jordi Cortadella, and et. al., "Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers," In Proc. of the 11th Conf. Design of Integrated Circuits and Systems, 1997.

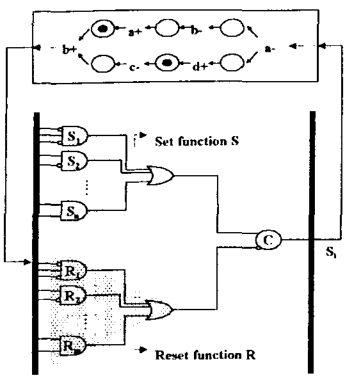

Figure 1. An ACGpER signal network

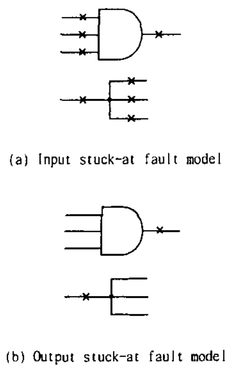
(a) Input stuck-at fault model
(b) Output stuck-at fault model

Figure 2. Fault model

Algorithm 1

Input : a target circuit C, the set of reachable states according to the specification R(C), and FaultList
Output : the set of undetectable fault list, Undetect

```
UndetectFaultIdentification(C, R(C), FaultList)
begin
    Undetect = ∅
    for every x stuck-at c in FaultList
        F_o = the function of the uniquely determined output variable
when x stuck-at c fault occur;
        Fg = F_o (x←c')= A + B·F;
        Ff = F_o (x←c) = C+D·F;

C1:     if (A⊕C =1 && B=D=0)
            Assignments = ConflictCheck(A⊕C =1);
            if Assignments ≠ FALSE
                Traverse(R(C), Assignments);
            else put the fault into Undetect; break;
            endif
C1':    else if (A⊕C =1 && B=D≠0)
            Assignments = ConflictCheck(A⊕C =1 && B=D=0);
            if Assignments ≠ FALSE
                Traverse(R(C), Assignments);
            else goto C2;
            endif
        endif
C2:     Assignments = ConflictCheck(F'=0 && (A+B)⊕(C+D) = 1)
            if Assignments ≠ FALSE
                Traverse(R(C), Assignments);
            else goto C3:
            endif
C3:     Assignments = ConflictCheck(F'=1 && (A+B)⊕(C+D) = 1)
            if Assignments ≠ FALSE
                Traverse(R(C), Assignments);
            else put the fault into Undetect; break;
            endif
    endfor
    return Undetect;
end
```

Figure 3. Algorithm 1

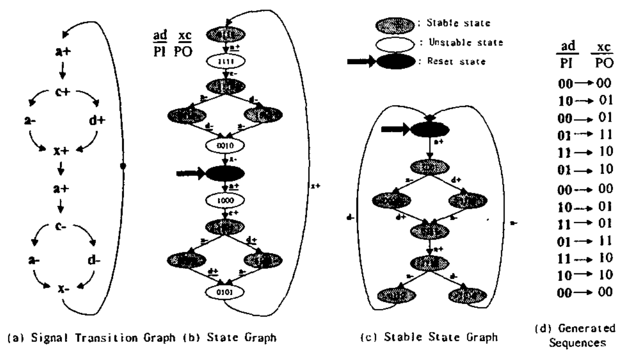

Figure 4. An example of the specified sequence generation

Algorithm 2

Input : a target circuit C and a specification STG for C
Output : Test[fault][test] which contains test for the relevant fault

```
ATPG(C, STG)
begin
    StateGraph = GenerateStateGraph(STG);
    StableGraph = GenerateStableGraph(StateGraph);
    Sequences[i] = GenerateSequences(StableGraph);
    i = LengthofSequences;
    FaultList = All possible fault;
    UndetectableFaultList = ∅
    Test[fault][test] = ∅
    GoodCircuit[i] = SymbolicTraversal(C,Sequences);
    PotentialUndetect = PotentialUndetectAnalysis(C);
    Undetect = UndetectFaultIdetification(C ,GoodCircuit[],
PotentialUndetect);

    for(every fault f in (FaultList-Undetect))
        C_f = FaultInsertion(C,f);
        i=0;
        do
            FaultyCircuit = NextStableState(C_f, Sequences[i]);
            If (i == LengthofSequences)
                put the fault f into UndetectableFaultList
                break;
            else
                Test[f][i] = Sequences[i];
            endif;
            i++;
        while(GoodCircuit[i] ≠ FaultyCircuit)
    endfor
    return Test[][]
end
```

Figure 5. Algorithm2

Table 1. Experimental results 1:
Test generation without undetectable fault identification

Example	Total Fault	Fault Cov.(%)	Modeling Time (Sec.)	ATPG Time (Sec.)	# of Seq.	# of Test
chu172	24	100	0.04	[illegible]	8	67
full	36	97.2	1.06	[illegible]	15	87
hazard	40	86.75	0.92	[illegible]	13	138
vbe5b	44	93.18	2.58	[illegible]	36	145
chu150	48	95.83	2.69	[illegible]	13	145
martin	48	91.67	4.07	[illegible]	13	133
chu133	50	88	7.27	[illegible]	37	269
converta	58	98.28	7.10	[illegible]	13	147

Table 2. Experimental results 2:
Test generation with undetectable fault identification

Example	Potential U fault	U Fault	Identification Time (Sec.)	ATPG Time (Sec.)	Time Saving (%)
chu172	4	0	0.11	[illegible]	-15.9
full	8	1	0.14	[illegible]	14.9
hazard	8	2	0.17	[illegible]	17.2
vbe5b	9	3	0.21	[illegible]	43.8
chu150	10	2	0.21	[illegible]	25.3
martin	10	4	0.16	[illegible]	23.5
chu133	10	6	0.26	[illegible]	44.1
converta	14	1	0.28	[illegible]	9.7