A Hierarchical Test Generation for Asynchronous Circuits

Eunjung Oh[1], Soo-Hyun Kim[1], Dong-Ik Lee[1], and Ho-Yong Choi[2]

[1]Dept. of Info. & Comm., K-JIST
1 Oryong-dong, Buk-gu, Gwangju 506-712, Korea
e-mail: {eunjung, kimsh, dilee}@kjist.ac.kr

[2]School of ECE, Chungbuk Nat'l. Univ.
48 Gaeshin-dong, Cheongju 361-763, Korea
e-mail: hychoi@chungbuk.ac.kr

Abstract: In this paper, we have presented a testing method for a kind of asynchronous circuits. Target circuit model is the 3D machine that is one of the most successful implementation of extended burst-mode (XBM) machines. We present a high-level test generation method for the 3D machine using the specification of the circuit. We also present a gate-level test pattern generation method using a synchronous test pattern generator. Experimental results show that the combination of the above two methods achieves high fault coverage over 3D machines and saves test generation time.

1. Introduction

Asynchronous finite state machines (AFSMs) are one of the widely used specification method in asynchronous circuit design. One method of specifying an AFSM is using a burst-mode (BM) machine. Early researches on AFSM assumed that the environment operates in fundamental mode, that is, the environment generates a single input change and waits for the machine to stabilize before it generates the next input change [1]. Recent work on AFSM's allows the multiple-input change, called as *burst*, fundamental mode operations. BM machines were first introduced by Davis et al. [2] and formalized by Nowick and Dill [3], [4]. BM machines have been implemented using MEAT [5], the locally clocked method [4], the 3D method [6], and the UCLOCK method [7].

One important limitation of AFSMs and BM machines is the strict regulation that changes in signal values must follow a prescribed order: inputs signals changes followed by outputs change followed by state signals change. In extended burst-mode (XBM) state machines, this limitation is loosened a bit by the introduction of directed don't cares. These allow one to specify that an input change may or may not happen in a given input burst. In the 3D method [6], K. Yun introduced timing assumptions in the design, so that the design style generates optimized circuits in terms of system performance and area. The generated circuit shows higher system performance and smaller area in comparison with other classes of asynchronous circuits [8]. Despite of the outstanding features of the design style, testing problem is not addressed yet. Along with other asynchronous circuits, the XBM machine does not have a global clock. Further, the XBM machine has timing assumptions on it, thus testing of XBM machine should consider its features as well as the absence of a global clock.

In this paper, we have presented a test generation methods for a 3D machine, which is an implementation of the XBM machine. A high-level test pattern generation method is presented based on XBM specification. Also, a gate-level test generation method, using a synchronous sequential test pattern generator, is presented.

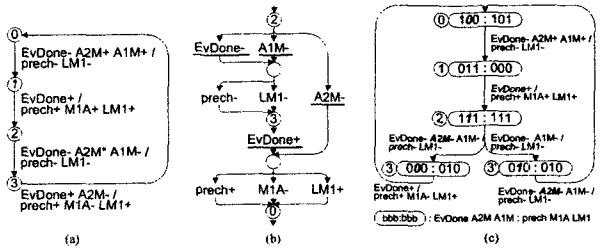

Figure 1. (a) An XBM specification (*mul1*), (b) Explicit *representation of A2M* signal transitions*, and (c) SSG of (a)

2. XBM Specification & 3D Machine

An XBM specification is represented as $G = (V, E, C, I, O, v_0, cond, in, out)$ [9]. V is a finite set of states. $E \subseteq V \times V$ is the set of state transitions. $C = \{c_1, \cdots, c_l\}$, $I = \{x_1, \cdots, x_m\}$, and $O = \{z_1, \cdots, z_n\}$ are the set of conditional inputs, edge inputs, and outputs, respectively. $v_0 \in V$ is the unique start. $cond : E \rightarrow \{0, 1, *\}^l$ and $out : V \rightarrow \{0, 1\}^n$ are the labeling function used to define values of the conditional inputs and edge inputs, respectively. $out : V \rightarrow \{0, 1\}^n$ is the labeling function used to define the values of the outputs upon entry to each state.

Figure 1 shows an XBM specification. Signals which are not enclosed in angle brackets and ending with +/- are *terminating edge signals*. The signals enclosed in angle brackets are *conditionals*, which are level signals whose values are sampled when all of the terminating edges associated with them have occurred. A state transition occurs only if all of the conditions are met and all the terminating edges have appeared. A signal ending with an asterisk (*) is a *directed don't care*. If a is a directed don't care, there must be a sequence of state transitions in the machine labeled with $a*$. If a state transition is labeled with $a*$, the following state transitions in the machine must be labeled with $a*$ or with $a+$ or $a-$ (the terminating edge for the directed don't care). A directed don't care may change at most once during a sequence of state transitions it labels.

A 3D machine is a particular implementation style for XBM machines. As shown in Figure 2-(a), a 3D AFSM is defined as a 4-tuple (X, Y, Z, δ) [9]. X/Y is a non-empty set of primary input/output symbols. Z is a (possibly empty) set of internal state variable symbols. $\delta : X \times Y \times Z \rightarrow Y \times Z$ is a next-state function. The hardware implementation of a 3D machine is a hazard-

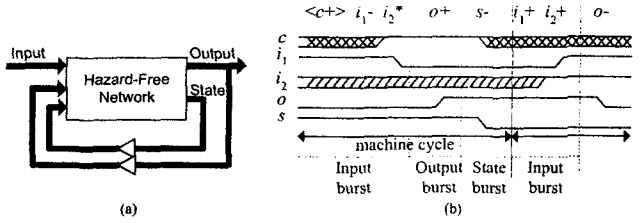

Figure 2. (a) The 3D machine and (b) Machine cycle of the 3D machine

free network which implements the next-state function, with the outputs of the network fed back as inputs to the network. Yun et al. presented two ways of implementing the next-state function: two-level SOP and generalized C-element in [9]. In this paper, we consider the two-level SOP implementation only.

There are three types of machine cycle in a 3D machine [9]. In this paper, we consider only Type II machine cycle, which is the most frequently used. A 3D machine with Type II machine cycle operates following sequences: an input burst followed by an output burst followed by a state burst. Figure 2-(b) depicts an example of operation of a 3D machine with 2 input signals (i_1, i_2), a output signal (o), and a state signal (c). The conditional signal c should be stabilized before the input burst $(i_1 - i_2*)$ occurs. After the input burst occurs, an output burst $(o+)$ followed by a state burst $(s-)$ occurs.

3. A High-Level Test Generation for 3D Machines

In this section, we present a high-level test generation using an XBM specification. A high-level fault model, State Transition Fault (STF) is defined based on a Stable State Graph (SSG) of an XBM specification. The SSG is a variation of state graph with extra state transitions caused by directed don't care signals. The test generation for STFs is also presented.

3.1 State Transition Fault

An STF is defined based on an XBM specification given as a specification of a 3D machine. We said that a fault exists in a state transition when a state transition of an XBM specification does not conducted correctly. Since a BM machine has the assumption of the fundamental mode of operation, each burst specified in an XBM specification is assumed to occur on a stable state. Thus, testing could be done using stable states: a test pattern has applied on stable states and the results have been observed on stable states.

An example of an XBM specification with a directed don't care signal ($A2M$) is depicted in Figure 1-(a). Figure 1-(b) shows the explicit representation of $A2M*$. Underlined signals are input signals. In a state, an output burst is generated after the occurrence of an input burst. The generated output burst is fed back to generate a state burst. The input burst should hold its values until the generation of the output burst is completed. Before each burst occurs, the machine should be in stable state. These timing assumptions are guaranteed by the delays on feed-back lines. To show this step-wise operation, empty circles are inserted between an input burst and an output burst. Each circle in Figure 1-(b) indicates a stable state.

Testing methods using stable states of asynchronous circuits are presented in [10], [11]. These methods generate test patterns based on state graph of a Speed-Independent (SI) circuit. The test generations are performed on only stable states. Both of the previous approaches targets SI circuits with Signal Transition Graph (STG) specifications. If a given AFSM could be represented as a state graph, of which states are binary encoded, the method presented in [10] could be applicable. For an AFSM with directed don't care signal, i.e., XBM specification, the reachable states by a directed don't care signal are not unique one. Thus, we should consider additional state transitions caused by directed don't care signals as well as state transitions in the original XBM specification.

3.2 Stable State Graph of XBM Specification

An STF is defined based on state transitions of an XBM specification. Considering state transitions caused by directed don't care signal, we introduce an SSG of an XBM specification. The concept of stable states of an AFSM is the same as that of an STG. The difference is the behavior of the directed don't care signals. As shown in Figure 1-(b), the $A2M$ signal can fall at any point between states 2 and 3 and between 3 and 0. The binary encoded state vector representation, SSG, is shown in Figure 1-(c). state 3 in Figure 1-(a) can be represented two binary encoded states, state 3 and state 3', in Figure 1-(c). In the view of testing, both of state transitions, ⟨ state 2, state 3⟩ and ⟨ state 2, state 3'⟩ should be checked. We define an explicit representation of an XBM specification with regard to directed don't care signals. Following is the formal definition of an explicit XBM specification.

Definition 1 (Explicit XBM specification) An explicit XBM specification, $G^E = (V^E, E^E, I, O, v_0, cond^E, in^E, out)$ of an XBM specification G is a directed graph. V^E is $V \cup V(E)$; $V(E)$ is the set of the duplicated states of destination states of the state transition having don't care signals. $E^E \subseteq V^E \times V^E$ is the set of state transitions. $cond^E = E^E \rightarrow \{0,1\}^l$ and $in^E : V^E \rightarrow \{0,1\}^m$ are the labeling function used to define the values of the conditional inputs and the edge inputs, respectively. I, O, v_0, and out are same as definitions of G.

The main difference of the newly defined representation is that the transitions caused by directed don't care signals are represented in the same form of other signals. In other words, edges with $a*$ is replaced by several edges with $\{a+, a-\}$ and several destination states. Based on the explicit XBM specification, an SSG can be defined. Following is the formal definition of an SSG.

Definition 2 (Stable State Graph (SSG)) Let $J = \{j_1, j_2, \cdots, j_{l+m+n}\}$ and $J_I = \{j_{l+1}, j_{l+2}, \cdots, j_{l+m}\}$, $J_I \subseteq J$ and $0 \leq I \leq l + m + n$, be the set of signals and edge input signals of $G^E = (V^E, E^E, I, O, v_0, cond^E, in^E, out)$, respectively. An SSG on J, $\Phi_J = (S, T, \delta, s_0)$, is defined as follows;

- S is the binary encoded representation of V^E.
- $s_0 \in S$ is the initial state.
- $T = J_I \times \{+, -\}$ is the set of edge input signal transitions.
- $\delta : S \times T \rightarrow S$ is a partial function having the property that:

  $\forall s, s' \in S, \forall t \in T$, such that $\delta(s,t) = s'$,
  if $t = j+$, then $s(j) = 0$ and $s'(j) = 1$,

Table 1. State transitions, tests, and outputs

State transition	Test	Output
$\langle s0 \rightarrow s1 \rangle$	100-011	000
$\langle s1 \rightarrow s2 \rangle$	100-011-111	111
$\langle s2 \rightarrow s3 \rangle$	100-011-111-000	010
$\langle s2 \rightarrow s3' \rangle$	100-011-111-010	010
$\langle s3 \rightarrow s0 \rangle$	100-011-111-000-100	101
$\langle s3' \rightarrow s0 \rangle$	100-011-111-010-100	101

if $t = j-$, then $s(j) = 1$ and $s'(j) = 0$.

Based on the above definition, an SSG can be derived for a given XBM specification. Figure 1-(c) is an SSG of Figure 1-(a). Once, an SSG is given, STFs can be defined on the transitions of the SSG. Following is the formal definition of an STF.

Definition 3 (State Transition Fault (STF)) A single STF f is defined on a state transition δ of Φ_J. When a fault-free state transition $\delta_g(s,t) = s'_g$ is corrupted by f, a faulty state transition $\delta_f(s,t) = s'_f$, $s'_{INg} = s'_{INf}$ and $s'_{OUTg} \neq s'_{OUTf}$, i.e., input signals are same and output signals are different, is generated.

3.3 Test Generation for State Transition Fault

The test generation for STFs is conducted on an SSG. By the definition of an SSG, the number of STFs is the number of the state transitions of an SSG. From a given XBM specification, an SSG could be constructed. With the SSG, an STF fault list is generated. For each STF, a test pattern, which detects the fault, is generated. The two fundamental steps in generating a test for a fault are 1) activation of the fault, and 2) propagation of the resulting error to primary outputs (POs). A state of an SSG consists of input and output signals. Assume that an STF f occurs on a state transition $\langle s_1, t, s_2 \rangle$ and the faulty transition results in reaching a state s_3, $s_2 \neq s_3$. In such a case, s_2 and s_3 have different values only in non-input variables. If the fault effect appears on output variables among non-input variables, the fault is detected. Otherwise the fault effects should be propagated on other states, s_4, on which the fault effects appear. Thus the application of an input sequence from a given initial state r to $s_2(s_4)$ makes it possible to check the existence of f. The input sequence is a test for the STF f. Therefore the test generation problem can be reduced to the problem finding the shortest path from r to $s_2(s_4)$. Finding the path can be performed a depth-first traversal of an SSG. Table 1 lists state transitions of Figure 1-(c), test, and the expected output.

4. A Test Generation for 3D machines using Synchronous TPG

In this section, we present a gate-level test pattern generation for 3D machines using an existing synchronous sequential TPG. The motivation of this approach is 1) there does not exist a test pattern generator for 3D machines, and 2) a 3D machine can be modified to a synchronous machine by introducing a simple assumption.

4.1 Testing of 3D Machines

Machine cycle of a traditional synchronous circuit is same as the period of a global clock. Synchronous TPGs has been developed based on the above fact. If the delays appeared in Figure 2-(a) could be replaced by state-holding elements, a 3D machine can be represented as same configuration of a synchronous FSM. For the state holding element, consider a flip-flop. The clock period of a flip-flop is set long enough, that is the value is larger than the maximum delay of a 3D machine. With this modification, a synchronous TPG could be applied to a 3D machine.

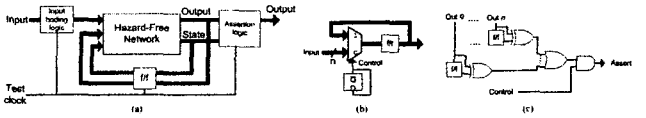

Figure 3. (a) A fault simulation model of 3D machine, (b) An example of input holding logic, and (c) An example of assertion logic

However, when a synchronous TPG is applied to a 3D machine, the generated test pattern should be validated. The reason is that a synchronous TPG conducted by the defined clock period, while a machine cycle of a 3D machine consists with 2 clock cycles as depicted in Figure 2-(b). To preserve the behavior of a 3D machine, a test pattern should satisfy the following requirements.

R1: a test pattern should hold its value throughout a machine cycle.

R2: a test pattern should generate output burst followed by a state burst.

The most relevant method for generating a test pattern, which satisfies the above requirements, is test generation presented in Section 3. Since an XBM specification includes all the requirements implicitly, the generated high-level test pattern could be applied to a 3D machine based on machine cycle of a 3D machine. Despite of the incompatibility of machine cycle of a synchronous TPG and a 3D machine, we consider to use an existing TPG. In a situation that a dedicated TPG for a 3D machine does not exist, this could be the second choice. The generated test patterns by a synchronous TPG are a psudo-test for a 3D machine. Thus the generated test patterns should be simulated to check its validity. The test patterns, which are validated by a simulation, do not guarantee the shortest one. The efficiency is not major concern in case that a small portion of fault lists is considered in the test generation.

4.2 A Fault Simulation for 3D Machines

Generated test patterns should be simulated to check whether the test patterns violate the operation of a 3D machine or not. Figure 3 depicts a simulation model of a 3D machine. The modified parts are follows:

- State-holding elements: delays are replaced by flip-flops. The clock period of a flip-flop is set to the value is larger than the maximum delay of a 3D machine.
- Input holding logic: input holding logic preserves an input through 2 clock cycles. As depicted in Figure 3-(b), a new input could be entered the circuits when *Control* signal is 0. When *Control* is 1, the previous input, which is stored in filp-flops, is used and the current input is discarded. The *Control* signal has value 0 and 1 repeatedly by the clock.
- Assertion logic: a machine cycle of the modified 3D machine is delimited by the *Control* signal. An output burst should occur during *Control* is 0 and a state burst during *Control* is 1. Figure 3-(c) depicts and example of assertion logic for output burst.

Input holding logic and assertion logic are designed to check the two requirements in Section 4.1 respectively.

Table 2. Experimental results

Example				ex1				ex2			ex3					
											STF model		HITEC			Total
Circuit	I / O	# of gate	# of s.a.f	# of tr.	# of E-tr.	# of test	Fault Cov.	CPU (Sec.)	# of test	Fault Cov.	#of fault	#of test	#of fault	#of test	CPU (Sec.)	Fault Cov.
MUL1	3/3	24	258	4	4	9*2	0.5969	2.633	61*2	0.8721	154	9*2	71	49*2	2.60	0.8721
MUL2	3/3	8	90	3	4	7*2	0.6333	0.05	15*2	1	57	7*2	33	13*2	0.083	1
ALU1	3/5	24	268	9	9	18*2	0.7239	0.25	62*2	0.9366	194	18*2	57	53*2	0.217	0.9366
pscsi-ircv	4/3	16	196	7	7	11*2	0.6531	0.117	40*2	1	128	11*2	68	23*2	0.066	1
pscsi-trcv	4/3	11	150	7	7	8*2	0.7067	0.117	28*2	0.9733	106	8*2	40	20*2	0.083	0.9733
pscsi-trcv-bm	4/4	19	276	9	9	22*2	0.7609	0.133	43*2	1	210	22*2	66	23*2	0.084	1
scsi-isend-csm	5/4	19	278	9	9	12*2	0.6475	0.15	59*2	0.9712	180	12*2	90	40*2	0.100	0.9712
scsi-trcv-csm	5/4	19	280	9	9	12*2	0.7571	0.133	56*2	0.9679	212	12*2	59	35*2	0.117	0.9679

5. Experimental Results

We have experimented three kinds of test generation: *ex1*) a high-level test generation based on XBM specifications, *ex2*) a gate-level test generation by a synchronous test pattern generator, and *ex3*) a combined test generation using *ex1*) and *ex2*). Experiments are conducted on a Sun Sparc Ultra-I with 128 megabytes of RAM. An asynchronous logic synthesizer, 3D [9], is used to generate a synthesized 3D machine, in the form of gate-level netlist, from an XBM specification. We have chosen HITEC [12] as a synchronous test pattern generator. The first experiment is conducted as following step:

1. Generate STF list based on an XBM specification.
2. Generate tests for STFs.
3. Apply the generated test vectors to synthesized 3D machines with single stuck-at faults.
4. Evaluate fault coverage and execution time.

Table 2 shows the results of the experiment. *Circuit* is the name of benchmarks. I/O is the number of input/output variables. $\#ofgate$ and $\#ofs.a.f$ are the number of gates and stuck-at faults of the synthesized gate-level circuits, respectively. $\#oftr.$ and $\#ofE-tr.$ are the number of the state transitions in an XBM specification and an SSG, respectively. The number of STFs is same as $\#ofE-tr.$ $\#oftest$ is the length of the generated test patterns. $FaultCov.$ is the stuck-at fault coverage of the high-level tests and obtained by a fault simulator, PROOF [12]. For fault simulation, delays of a 3D machine are replaced with D flip-flops. The generated test patterns are applied for a machine cycle, that is 2 clock cycles. Thus figures of $\#oftest$ column are represented as *The number of test pattern* $*$ 2. The average of stuck-at fault coverage with the generated high-level test patterns is about 0.6849.

The second experiment is a gate-level test generation for 3D machines using HITEC. For synthesized 3D machines, HITEC generates psedo-test patterns. To simulate the generated patterns, each example is modified according to the simulation model presented in Section 4.2 *HITEC* column shows the test generation time ($CPU(Sec.)$), the number of patterns ($\#oftest$), and stuck-at fault coverage ($FaultCov.$). The generated test patterns show high fault coverage, about 0.9651 on average.

The third experiment is a combined test generation. For the uncovered faults in the first experiment, HITEC generates pseudo-test patterns. The generated tests by HITEC are also fault simulated. The column $\#offaults$ are covered faults by each method. From the results, the test generation time and the number of test are reduced compare to the second experiment. Higher fault coverage is obtained compare to the first experiment.

6. Conclusions

We have presented a testing method for a 3D machine, which is a successful implementation of an XBM machine. We have presented a high-level test generation method for the 3D machine using the specification of the circuit. A gate-level test pattern generation method is also presented, which uses a synchronous test pattern generator, HITEC. Experimental results showed that the combination of the above two methods achieves high fault coverage over 3D machines and saves the test generation time.

Acknowledgement

This work was supported in part by grant No. R02-2000-00254 from the Basic Research Program of the Korea Science & Engineering Foundation and by the KAIST/K-JIST IT-21 Initiative in BK21 of Minisry of Education and by Korea.

References

[1] S. H. Unger, *Asynchronous Sequential Switching Circuits*, Wiley-Interscience, 1969.
[2] A. Davis, B. Coates, and K. Stevens, "The post office experience: designing a large asynchronous chip," Proc. Hawaii Int'l. Conf. System Sciences, Vol. I, pp. 409-418, 1993.
[3] S. M. Nowick and D. L. Dill, "Synthesis of asynchronous state machines using a local clock," Proc. ICCD, pp. 192-197, 1991.
[4] S. M. Nowick, *Automatic Synthesis of Burst-Mode Asynchronous Controllers*, Ph.D. Thesis, Stanford Univ., Dept. of Computer Science, 1993.
[5] A. Davis, B. Coates, and K. Stevens, "Automatic synthesis of fast compact asynchronous control circuits," in *Asynchronous Design Methodologies*, S. Furber and M. Edwards, Eds. Vol. A-28 of IFIP Transactions. Elseviser Science Publishers, 1993.
[6] K. Y. Yun, *Synthesis of Asynchronous Controllers for Heterogeneous Systems*, Ph.D. Thesis, Standford Univ., 1994.
[7] S. M. Nowick and B. Coates, "UCLOCK: automated design of high-performance asynchronous state machines," Proc. ICCD, pp.434-441, 1994.
[8] K. Y. Yun, P. A. Beerel, V. Vakilotojar, A. E. Dooply, and J. Arceo, "The design and verification of a high-performance low-control-overhead asynchronous differential equation solver," *IEEE Trans. VLSI Syst.*, Vol.6, pp.643-655, 1998.
[9] K. Y. Yun and D. L. Dill, "Automatic synthesis of extended burst-mode circuits. I. (Specification and hazard-free implementations)," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol. 18, Issue. 2, pp.101-117, 1999.
[10] E. Oh, S.-H. Kim, D.-I. Lee, H.-Y. Choi, "High-level test generation for asynchronous circuits from signal transition graph," Proc. Workshop on Synthesis and System Integration of Mixed Technologies, pp.159-166, 2001.
[11] O. Roig, J. Cortadella, M. Pẽna, and E. Pastor, "Automatic test generation of synchronous test patterns for asynchronous circuits," Proc. DAC, pp.620-625, 1997.
[12] HITEC/PROOFS User's Manual, Center for Reliable & High-Performance Computing, Univ. of Illinois, 1996.