Using Verification Technology for Validation Coverage Analysis and Test Generation *

Dinos Moundanos

Computer Engineering Research Center
University of Texas at Austin
ENS 424
Austin, TX 78712

Jacob A. Abraham

Computer Engineering Research Center
University of Texas at Austin
ENS 424
Austin, TX 78712

Abstract

Despite great advances in Formal Verification (FV), simulation is still the primary means for design validation. The definition of pragmatic measures for the coverage achieved and the problem of automatic test generation(ATG) are of great importance. In this paper we introduce a new set of metrics, the Event Sequence Coverage Metrics (ESCMs). Our approach is based on an automatic method to extract the control flow of a circuit which can be explored for coverage analysis and ATG. We combine FV and traditional ATPG techniques to automatically generate sequences which traverse uncovered parts of the control graph or exercise uninstantiated control event sequences.

1 Introduction

Design verification deals with checking the conformance of the design to its functional specification. It is a complex task especially in the case of modern, high performance circuits which employ a series of architectural techniques aiming at boosting performance. While much progress has been made in automating and using FV tools, a major limitation of this technology is still the size of the circuits it can handle. As a result, validation by simulation is still the primary means of checking the correctness of a design. The test vectors can be generated by random or pseudo-random (biased) test generators, constraint solvers, can be taken from typical workloads, or can be hand written by the designers. All these methods fail to provide a measurable degree of confidence that the design has been tested adequately.

Coverage is a measure of the completeness of a test suite. The ideal metric for evaluating the coverage of a test suite would be the fraction of execution paths exercised. However, finding and exercising all possible execution paths has exponential complexity. The next best measure is the fraction of reachable states or transitions that have been exercised [7]. In [7] as well as in this paper, coverage is estimated on an abstracted model, the Extracted Control Flow Machine (ECFM) model. We introduce a new set of coverage metrics, called Event Sequence Coverage Metrics (ESCMs), to complement the metrics defined in [7]. In this methodology the designer specifies interesting control event sequences that need to be exercised during simulation. These control sequences can capture the complex interaction of control events that characterize difficult corner cases in the design. The motivation behind these new metrics is that full transition coverage, even on the ECFM of a design, may be neither possible nor desirable. The former is true because the non-control parts of the design are modeled nondeterministically, and the latter is true because not all transitions may need to be exercised to fully test the functionality [4] (the concept of equivalent transitions [7] eliminates this concern). Furthermore, we provide a technique for Coverage–Directed ATG which generates test sequences that guarantee high coverage of the control behavior of the design. Additionally, for event sequences not exercised we automatically generate test vectors that would cause the machine to go through that sequence of events. Test sequence generation is performed on the ECFM model. This sequence may not be directly applicable to the original machine because of data conflicts. ATPG techniques are used to map the sequence back to the original machine.

1.1 Previous Work

Ad-hoc techniques for coverage estimation include toggle coverage on signals in the HDL program and HDL statement coverage. It is generally accepted that

*This research was supported by SRC under contract 97-DP-388 and TATP under project 003658-268 at the University of Texas at Austin

these are not accurate measures [7]. Some approaches to evaluate coverage check that all states and transitions are visited [7, 9]. Other approaches described in [5, 4] and [3] use techniques similar to the ones presented here. State machines for part of the circuit are extracted from the HDL and used as the target of test generation. The main difference between [5] and our work lies in the mapping of the tests back to the original circuit. Their approach is to artificially inject the required signal values during simulation, while we use conventional ATPG techniques. That work is extended in [4] with the concept of control events. We employ a more generic form of control events in this paper. The approach in [3] uses the counterexample facility in the SMV model checker for test generation. Mapping these tests to the actual machine depends on recognition of patterns of high level behavior and is a labor-intensive process. Finally, [2] introduces an observability–based coverage metric. The approach is based on injecting tags in variables in the HDL and observing the "activation" and "propagation" of these tags to the outputs. It is tied to the HDL syntactic style and based on the principle of activating every statement in the HDL. In [1], an extended finite state machine (EFSM) model is extracted from the behavioral description and is exhaustively traversed to generate functional tests. This approach analyzes the syntactic structure of the HDL and identifies equivalence relations among parts of the data space thus leading to a sometimes smaller FSM which is equivalent to the original FSM. However, it depends heavily on the syntactic style of the HDL. The approach in [8] introduces the concept of "design errors" to model possible faults. Coverage of the test suite is the fraction of possible design errors detected by the suite. This is a localized structural approach and does not give an indication of the extent to which the behavior of the design has been exercised.

2 The Extracted Control Flow Machine Model

The ECFM of a circuit is a model of the control flow in the design and is extracted by inspecting the finite state machine (FSM) representing the complete circuit. The difficulty in identifying the control circuitry often lies in defining the interface of the control unit with the rest of the circuit, and not in differentiating the control registers from registers holding pure data. Consequently, we abstract the data registers from the circuit and group the data into "equivalence" classes with respect to their effect on the control.

The behavior of a circuit is represented as a Mealy FSM which is a 6–tuple $(\Sigma, O, S, s^0, \Delta, \Lambda)$, where $\Sigma = \{0,1\}^n$ is the input space (n:# of input bits), $O = \{0,1\}^l$ is the output space (l:# of output bits), $S = \{0,1\}^{c+d}$ is the state space (c:# of control bits, d:# of data bits), $s^0 = \{< \vec{s_c}, \vec{s_d} >\}$ is the set of initial states, $\vec{s_c} \in \{0,1\}^c$ and $\vec{s_d} \in \{0,1\}^d$, $\Delta : \Sigma \times S \rightarrow S$ is the next–state functional vector, $\Delta = [\delta_1 ... \delta_{c+d}]$, and $\Lambda : \Sigma \times S \rightarrow O$ is the output functional vector, $\Lambda = [\lambda_1 ... \lambda_l]$. The ECFM is also represented as a Mealy type FSM $(\Sigma', O', S', s^{0'}, \Delta', \Lambda')$, where $\Sigma' = \{0,1\}^{n'+d'}$ is the input space ($n' \leq n, d' \leq d$), $O' = \{0,1\}^{l'}$ is the output space ($l' \leq l$), $S' = \{0,1\}^c$ is the state space, $s^{0'} = \{< s_c >\}$ is the set of initial states, $s_c \in S'$, $\Delta' : \Sigma' \times S' \rightarrow S'$ is the next–state functional vector, $\Delta' = [\delta_1 ... \delta_c]$, and $\Lambda' : \Sigma' \times S' \rightarrow O'$ is the output functional vector, $\Lambda' = [\lambda_1 ... \lambda_{l'}]$. The ECFM input space is smaller than $n+d$ because only those inputs and data registers that have an effect on control flow appear in the ECFM. Some outputs of the original circuit may also be dropped. In the following, we assume that $n' = n$, $d' = d$ and $l' = l$ without loss of generality. A transition in the original circuit can be represented as a 4–tuple $(\vec{i}, \vec{s_1}, \vec{s_2}, \vec{o})$ where $\vec{i} \in \Sigma$, $\vec{s_1}, \vec{s_2} \in S$ and $\vec{o} \in O$ and $\vec{o} = \Lambda(\vec{i}, \vec{s_1})$ and $\vec{s_2} = \Delta(\vec{i}, \vec{s_1})$. If we represent $\vec{s_1}$ and $\vec{s_2}$ as $\vec{s_1} =< \vec{s_{1c}}, \vec{s_{1d}} >$ where $\vec{s_{1c}} \in \{0,1\}^c$ and $\vec{s_{1d}} \in \{0,1\}^d$, $\vec{s_2} =< \vec{s_{2c}}, \vec{s_{2d}} >$ where $\vec{s_{2c}} \in \{0,1\}^c$ and $\vec{s_{2d}} \in \{0,1\}^d$, then this transition maps to the corresponding transition $(< \vec{i}, \vec{s_{1d}} >, \vec{s_{1c}}, \vec{s_{2c}}, \vec{o})$ in the ECFM. Thus, every transition in the original circuit maps to at least one transition in the ECFM. Furthermore, $\vec{o} = \Lambda'(< \vec{i}, \vec{s_{1d}} >, \vec{s_{1c}})$ and $\vec{s_{2c}} = \Delta'(< \vec{i}, \vec{s_{1d}} >, \vec{s_{1c}})$.

Definition: Two transitions $(\vec{i}, \vec{s_1}, \vec{s_2}, \vec{o})$ and $(\vec{j}, \vec{s_3}, \vec{s_4}, \vec{p})$ in the ECFM of a circuit are **equivalent** iff $\vec{s_1} = \vec{s_3}$ and $\vec{s_2} = \vec{s_4}$ and $\vec{o} = \vec{p}$.

Lemma: The equivalence partitions on the transitions of the ECFM of a circuit define corresponding equivalence partitions on the transitions of the original circuit. □

Theorem: The process of grouping equivalent transitions in the ECFM of a circuit partitions the state space of the original circuit in terms of its effect on the control flow of the circuit. □

It should be noted that the reachable state space in the ECFM does not directly correspond to the reachable control states in the actual machine. However, it is a close approximation because, in general, data assumes any value.

3 Functional Coverage Metrics

3.1 Previously Defined Coverage Metrics

The two metrics that we use , a state coverage metric (SCM) and a more accurate transition coverage metric (TCM), are given as follows.

$$SCM = \frac{\#\ of\ ECFM\ states\ visited}{\#\ of\ reachable\ ECFM\ states} \qquad (1)$$

$$TCM = \frac{\#\ of\ ECFM\ transitions\ traversed}{\#\ of\ reachable\ ECFM\ transitions} \quad (2)$$

Full transition coverage may not be possible because some transitions in the ECFM may require data values that are not possible in the original machine. Hence, it is reasonable to be satisfied with a relatively high value of TCM. Second, the user has some control over the ECFM generation and can iterate over the extraction process by changing the designation of registers as control or data.

3.2 Extending the Coverage Metrics

We propose a new approach which is called Event Sequence Coverage. By an event sequence we mean a series of events that have to take place in a specific order and according to some specific timing requirements. We define two types of event sequences: "good" sequences indicating the presence of desirable behavior (liveness requirement) and "bad" sequences monitored to ensure the absence of undesirable behavior (safety requirement). We define two additional coverage metrics.

$$ESCM_{incl} = \frac{\#\ of\ good\ Event\ Sequences\ covered}{\#\ of\ good\ Event\ Sequences\ monitored} \quad (3)$$

$$ESCM_{excl} = \frac{\#\ of\ bad\ Event\ Sequences\ covered}{\#\ of\ bad\ Event\ Sequences\ monitored} \quad (4)$$

The Test Plan goals are specified by the user in a specialized Language, and are translated into a special form of State Machines, the Event Sequence Finite State Machines (ESFSMs), which are characterized by the presence of non deterministic states. We take advantage of non determinism to describe the timing requirements of the event sequences. The Language is described in the following.

```
INCLUDE | EXCLUDE
TE <triggering_event> IMPLIES
CA <consequent_action>
```

The Triggering Event is simply described as a series of Boolean expressions which involve signals and Boolean connectives. The Consequent Action is described in the following form:

```
[Comb_Op]
{<set_of_signals>}{<tw1>,...,<twN>}
[{<set_of_signals>}{<tw1>,...,<twN>},...]
[FOLLOWED]
[{<set_of_signals>}{<tw1>,...,<twN>},...]
```

where Comb_Op can be AND, OR, MUTEX (mutual exclusion). A timing window (TW) $[t_1, t_2]$ specifies when and for how long a signal is to be asserted, where t_1, t_2 are natural constants with $t_1 \leq t_2$. The start point is relative to the Triggering Event (TE), infinity is equivalent to the end of simulation and endpoints are inclusive. The following list describes some of the different forms of Timing Windows.

- $[t_1..t_2]$: signal has to be asserted from t_1 to t_2 continuously.
- $[t_1*]$: signal has to be asserted at least once from t_1 to ∞.
- $[t_1, t_2]$: signal has to be asserted at least once from t_1 to t_2.
- $[t_1 * ..t_2]$: signal has to be asserted sometime between t_1 and t_2 and remain asserted until t_2.

Figures 1 and 2 illustrate the translation of two event sequences into ESFSMs (u stands for unknown). The first sequence indicates that signal q will be asserted 1 cycle after p is asserted and will remain asserted for up to 3 cycles.

```
INCLUDE TE {p} CA {q}{[1..*4]}
```

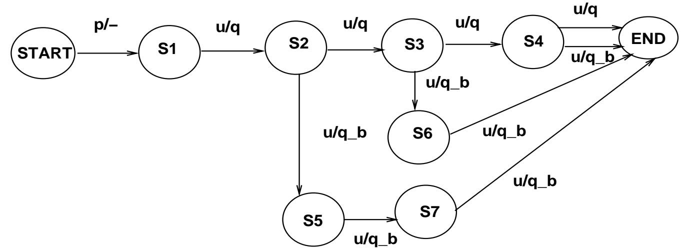

Figure 1: An FSM for Event Sequence 1

The second event sequence indicates that signal q will eventually be asserted 2 cycles after p is asserted.

```
INCLUDE TE {p} CA {q}{[2*]}
```

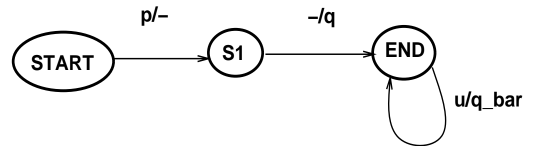

Figure 2: An FSM for Event Sequence 2

The timing window in the second event sequence has the semantics of eventuality, and so it cannot be represented as an FSM. So we internally complement the event sequence and translate this new sequence into an ESFSM. We then check for absence (presence) of bad (good) behavior depending on whether the original event sequence was a "good" ("bad") sequence. Our Event Sequence coverage system is depicted in Figure 3. First the ECFM of the design is extracted and given as input, along with the test suite, to the core of the system which consists of a 2-value logic simulator and a symbolic engine based on BDDs. Event Sequences are

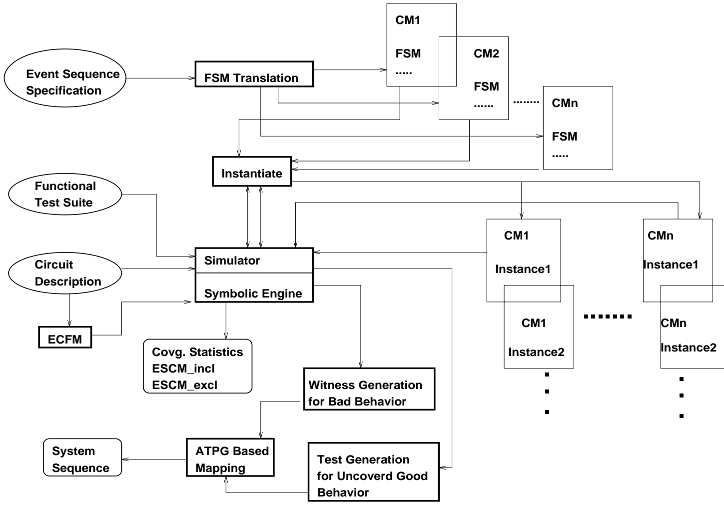

Figure 3: Event Coverage System

translated into ESFSMs and for each one of them a Coverage Monitor (CM) is created. CMs contain both an explicit and a symbolic description of the ESFSM. When the simulation starts for each occurrence of a Triggering Event a Coverage Monitor is instantiated. A test generation system is provided for the cases where a "good" event sequence is not observed, and a witness generation system is provided for the cases where a "bad" event sequence is observed during simulation.

4 Test Generation

4.1 Coverage Directed Test Generation

In this case SCM or TCM drive the test generation process. Test generation is terminated if a user–specified level of coverage is achieved or if a user-specified time limit is reached. The objective is to come up with a set of transition sequences starting from the initial state whose union will include all transitions in the ECFM graph. The traversal of the graph initially proceeds in a depth–first manner, selecting untraversed transition groups out of each visited state until no more untraversed groups exist. We mark control states out of which untraversed transition groups exist. These states are placed in a set and graded based on two criteria: the number of untraversed transition groups out of them and the number of unvisited next states that they have. We give higher priority to the second factor and break ties based on the first factor. When we reach a state where no more choices exist we go back to the set of marked states and pick one state based on the discussion above. This introduces a breadth–first flavor in our search and allows us to visit a bigger part of the ECFM state graph given the time and memory limits. The input vectors to the ECFM may consist of both Primary Inputs (PIs) and data registers. This means that the transition sequences generated in the way described above cannot be directly used for simulation of the original unit under test. The data values need to be justified, and if that is not possible the part of the transition sequence after the "unjustifiable" vector is dropped. There are several ways in which the justification can be achieved. For example one can use high level functional information to put the right values into the data registers at the right time.

4.2 Test Generation for Event Sequences

Definition: State s_1 in FSM M_1 is compatible with state s_2 in FSM M_2 iff every input sequence that is applicable to M_1 in state s_1 is also applicable to M_2 in state s_2 and causes the two machines to produce the same output sequence when applied to M_1 in initial state s_1 and M_2 in initial state s_2.

Definition: State s_1 in ESFSM M_1 is loosely compatible with state s_2 in FSM M_2 if s_1 is compatible with s_2 for all input sequences that contain one and only one applicable input vector for each non deterministic state and all applicable input vectors for each deterministic state [6].

The problem of proving the loose compatibility of the ESFSM with the design ECFM is formulated as a fixed point problem of discovering the set of states in the design that are loosely compatible with the start state of the ESFSM. The predicate γ encodes the set of pairs of compatible states in the ESFSM and the ECFM of the design:

$$\gamma(\vec{x}, \vec{q}_a, \vec{q}_b) \;=\; \bigwedge_{i=1}^{l} \left(\lambda_{a_i}(\vec{x}, \vec{q}_a) \odot \lambda_{b_i}(\vec{x}, \vec{q}_b)\right) \qquad (5)$$

where subscript a is used for the ESFSM and b is used for the ECFM of the design and $\odot$ stands for the XNOR operator.

To find the set of loosely $k+1$-compatible state pairs LC_{k+1} from the set of loosely k-compatible pairs LC_k, we find the **inverse image** and **pre-image** of the set LC_k, *i.e.*, the pairs of states which must end up in loosely k-compatible states in one transition, and the pairs of states which could end up in loosely k-compatible states in one transition. We further ensure that the transitions to LC_k from its pre-image satisfy the predicate γ.

$$LC_0 \;=\; 1 \qquad (6)$$

$$\begin{aligned} LC_{k+1}(\vec{q}_a, \vec{q}_b) &= LC_k(\vec{q}_a, \vec{q}_b) \\ \bigwedge &[(\neg ND(\vec{q}_a) \wedge \forall \vec{x} : LC_k(\vec{Q}_a, \vec{Q}_b)) \\ \bigvee &(ND(\vec{q}_a) \wedge \exists \vec{x} : (\gamma \wedge LC_k(\vec{Q}_a, \vec{Q}_b)))] \end{aligned} \qquad (7)$$

where the predicate ND represents the non–deterministic states in the ESFSM. The set of loosely compatible state pairs LC is obtained when the fixed-point is reached.

$$LC(\vec{q}_a, \vec{q}_b) \;=\; LC_k(\vec{q}_a, \vec{q}_b) \qquad if\ LC_{k+1} = LC_k \quad (8)$$

The set of ECFM states compatible with the start state of the ESFSM $\vec{q}_{a_{start}}$ has characteristic function given by

$$LC(\vec{q}_a, \vec{q}_b)_{\vec{q}_a = \vec{q}_{a_{start}}} \quad (9)$$

In the case of good behavior not covered we check the compatibility of the start state of the ESFSM with the ECFM of the design. If the set of loosely compatible states is non empty we form the product machine and traverse the state space from initial states. The traversal is done by picking new states via compatible transitions. The procedure terminates either when a final state is picked as next state or when a previously picked transition is chosen again, since in this case we have entered a loop. In the case of bad behavior observed we want to generate a witness sequence. We mark the state of the ECFM at which the Excluded Behavior is observed and we then perform backward traversal using pre–image computations guided by the ESFSM representing the excluded behavior.

4.3 Justification and Mapping Back

As was mentioned earlier, when performing coverage–directed test generation and test generation for event sequences, the test sequence generated on the ECFM is not directly applicable to the original circuit. This sequence has to be expanded and mapped back to the original circuit. First, for inputs that have been abstracted away in the ECFM model, random values have to be provided. Second, all data conflicts in the generated sequence must be resolved by justifying the values of those PI's which are data registers in the original circuit. We propose the use of ATPG techniques for performing both justification and mapping back. In general, we have a pair of partially or completely specified states and need to generate a sequence that will bring the original machine from the first to the second state. Each of these states consists of a control part retained in the ECFM and a data part which is captured in the transitions of the ECFM. In this case we take the justification sequence generated by the ATPG tool for the second state and look for a subsequence that involves the first state. This subsequence is then incorporated in our test. If a justification sequence cannot be produced then the corresponding vectors are dropped from our test. Additionally we "cache" justification sequences so that if the same state needs to be justified again we do not have to go through the sequence expansion process again.

5 Experimental Results

We applied our methodology to two microprocessors (see Table 1). The first (last) four columns provide information about the original circuit (its ECFM) Column 8 gives the extraction time. The Viper microprocessor is a 32-bit microprocessor with four general purpose registers, two ALU registers along with a memory address register and an instruction register. The gl85 circuit is a model of the 8085 microprocessor. This model uses 8–bit input and output buses in the place of the 8–bit bidirectional address–data bus. Operation of the gl85 proceeds under the control of two state machines. Table 2 presents our validation coverage analysis results. Column 2 gives the number of test vectors applied, columns 3 and 4 give the SCM and TCM and column 5 gives the time to compute the metrics in minutes. Take the Viper as an example. We evaluate the functional coverage on the ECFM of the Viper which has 32 states out of which 17 are reachable from the initial state. Our system also identified instances of unexercised behavior. For example the circuit does not enter the halt state with the comparison flag bit set, which implies that an instruction causing the machine to enter the halt state was not tested after an instruction causing the flag to be set. For the gl85, 7296 states out of the possible 16384 states in the ECFM are reachable. Table 2 correlates the stuck-at fault coverage and the functional coverage. We see that high stuck–at coverage does not guarantee high functional coverage. Table 3 presents the results on Coverage–Directed Test Generation for the two microprocessors. Our objective is to get 100% state coverage, or terminate the process when a timeout limit of 200000 seconds is reached. The 1st column gives the number of vectors generated, the 2nd column gives the number of

Table 1: Circuit Statistics

Circuit	Original			ECFM			
	PIs	POs	FFs	PIs	POs	FFs	Extr.
viper	33	53	251	75	1	5	8.8s
gl85	17	27	256	44	11	14	8.5s

Table 2: Functional Coverage Results

Circ	Vec	SA Cov(%)	SCM (%)	TCM (%)	Time (m)
viper	5959	91.46	100	91.46	8.84
gl85	24307	79.78	93.81	29.15	37.63

Table 3: Functional Test Generation Results

Circ.	#Vec.	#Just.	Time(s)	SA Covg.(%)	
				All	Control
viper	19813	1856	9867	81.78	94.17
gl85	57307	2472	200000	53.10	72.98

times justification was necessary, the 3d column gives the overall time required, and the last two columns give the stuck-at fault coverage that the generated sequences achieve, for the complete fault list or the control fault list (faults present in the ECFM). Although stuck-at fault coverage is not necessarily an accurate measure of the quality of a functional test sequence, we believe that it is a good indication of its effectiveness. For the viper the objective of 100% state coverage was achieved while for gl85 the process timed out. Both sequences are quite lengthy, which is a common characteristic of functional test generation. However, this approach produces a test sequence with much better coverage compared to directly applying ATPG on the circuits.

Table 4: Control Event Sequence Coverage Results

Event Sequence Monitoring	Type	Instances
Instruction Decoding	Comparison	335
	Boolean	478
	Arithmetic	93
	Call	45
	Memory	54
	I/O	87
Illegal Instructions		310
Overflow Conditions	Addition	12
	Subtraction	11
	Left Shifting	2

We then utilized the sequence of 5959 vectors on the viper to monitor for specific event sequences. The whole process takes 18 minutes. We are basically monitoring for good behavior and we are using 32 event sequences (27 monitoring instruction decoding, 3 monitoring overflow conditions, 1 monitoring illegal instructions). The results are given in Table 4. We generated an event sequence which has the setting of register B as the triggering event and the raising of the STOP flag(which signifies entering the HALT state) within 2 to 6 cycles as the consequent action. We utilized the algorithms described in section 4.2 for event sequence test generation and generated a sequence of two instructions that will cause the STOP flag to be set and the machine to enter the HALT state immediately after that. The first is a comparison instruction and the second is an illegal instruction. This took 17.1 seconds. We then expanded this sequence by justifying the values that needed to be loaded to registers R and M, the registers used by the comparison instruction. This was done in 16 seconds. All experiments were run on an UltraSparc II with one GB of RAM.

6 Conclusions

In this report we have considered design validation by addressing two issues associated with it. We extended the previously introduced state and transition coverage metrics with additional metrics targeting control event sequences. Secondly we provided two techniques in automatic test generation. Our future work involves the application of these techniques to larger more complicated examples (with emphasis on pipelined designs) and the utilization of more efficient and scalable techniques for mapping back to the original machine. Finally we would also like to investigate the possibility of automating the process of discovering the event control sequences that need to be exercised.

References

[1] K. Cheng and A. Krishnakumar. "Automatic Functional Test Generation Using the Extended Finite State Machine Model". *Proc. 30th DAC*, pages 86–91, 1993.

[2] S. Devadas, A. Ghosh, and K. Keutzer. "An Observability–Based Code Coverage Metric for Functional Simulation". *Proc. ICCAD*, pages 418,425, 1996.

[3] D. Geist, M. Farkas, A. Landver, Y. Lichtenstein, S. Ur, and Y. Wolfsthal. "Coverage–Directed Test Generation Using Symbolic Techniques". *Proc. Formal Methods in CAD*, 1996.

[4] R. Ho and M. Horowitz. "Validation Coverage Analysis for Complex Digital Designs". *Proc. ICCAD*, pages 146–151, 1996.

[5] R. Ho, C. Yang, M. Horowitz, and D. Dill. "Architecture Validation for Processors". *Proc. 22nd International Symposium on Computer Architecture*, pages 404–413, 1995.

[6] Y.V. Hoskote. "Formal Techniques for Verification of Synchronous Sequential Circuits". *Ph.D. Dissertation, ECE Dept., UT Austin*, 1995.

[7] Y.V. Hoskote, D. Moundanos, and J.A. Abraham. "Automatic Extraction of the Control Flow Machine and Application to Evaluating Coverage of Verification Vectors". *Proc. ICCD*, pages 532–537, 1995.

[8] S. Kang and S. Szygenda. "Design Validation: Comparing Theoretical and Empirical Results of Design Error Modeling". *IEEE Design and Test*, pages 18–26, 1994.

[9] D. Moundanos, J.A. Abraham, and Y.V. Hoskote. "A Unified Framework for Design Validation and Manufacturing Test". *Proc. ITC*, pages 875–884, 1996.