Functional Test Generation Guided by Steady-State Probabilities of Abstract Design

Jian Wang[12], Huawei Li[1], Tao Lv[1], Tiancheng Wang[12] and Xiaowei Li[1]
[1]State Key Laboratory of Computer Architecture,
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[2]University of Chinese Academy of Sciences, Beijing, China
{wangjian2010, lihuawei, lvtao, wangtiancheng, lxw}@ict.ac.cn

***Abstract*—This paper presents a novel method for functional test generation aiming at exploring control state space of the design. The steady-state probabilities (SP's) of the abstract design's control FSM are used to guide test generation. The SP's of the states can reflect how hard the states can be reached, and the hard-to-reach states are assigned with high priority to be exercised. Experimental results show that our method has better performance in test generation in comparison with constrained random simulation, and demonstrate that SP's provide good guidance on traversing hard-to-reach states of the design under validation.**

Keywords—functional test generation; steady-state probability; control state space exploration

I. INTRODUCTION

Simulation is the primary method for functional verification in industry nowadays. Finite state machine (FSM) state coverage is one of many coverage metrics which are used to evaluate the quality of functional test. It is non-trivial to achieve 100% state coverage for a large system because there are numerous states within it. A design can be split into control path and data path. The control path contains control logics, such as branches, conditions and explicitly coded FSMs in the design; the data path contains the data flow related logics, such as operands and results of an FPU. It is data path but not control path that contributes most to state space explosion because latches in control path have interlock mechanism while latches in data path do not. What's more, control path plays a central role in the system and more error-prone in the design. Therefore, for functional validation, instead of wasting enormous verification resources in exploring the whole state space of the design, a trade-off can be made by exploring the states of the control path, which is an abstract version of the design.

Furthermore, verification engineers usually pay more attention to a small part of logics of the control path. This part of logics usually contains some critical signals, such as forwarding control signals of pipelines of a CPU. A more compact abstraction model of the design can be obtained by reserving logics in the fan-in cones of these critical signals and removing other unrelated logics. The behaviors of the critical signals can be verified by exploring the state space of this abstract model. The scale of the FSM of this abstract model is much smaller than that of the original control path.

Several researches have presented methods for exploring the control FSM of the design under verification(DUV) [1-3]. These techniques use simple searching strategies without guidance to get traces on the control FSM. In addition, they need translations from the traces on the control FSM to the real test sequences for the DUV.

Several heuristic methods have been presented for guiding test generation to exercise certain target states. These target states are usually corner case states or states that may violate the assertions. In [4, 5], deterministic or probabilistic methods are used to search a short path from the reset state to the target states. In [6-8], some abstract-distance-guided strategies have shown success in exercising target states. These works are efficient for exercising one single target state, but it needs expert knowledge to point out which states should extra attention be paid to. Furthermore, if there are a large amount of target states, the repeated guided search processes are time-consuming.

In this paper, we present a functional test generation method for exploring the state space of the control path of the DUV. The "control path" is an abstraction model of the DUV, which is obtained by extracting all control logics or a subset of control logics(i.e. the critical logics and their fan-ins) from the DUV. During the control state space traversing, the hard-to-reach states are assigned with high priority to be exercised. This is achieved by analyzing the steady-state probability (SP), which is the probability of a random test sequence being able to exercise the state. To obtain the SP's, the control path FSM is modeled as a Discrete Time Markov Chain (DTMC). The stationary distribution (SD) of a DTMC, constituted by the SP's of states, can reflect how hard the states of the DTMC can be reached by a random test sequence. Rather than generating traces on the control FSM first and then translating the traces to real test sequences, we directly generate concrete test vectors for the DUV using the guidance of the SP's combined with constrained random simulation. We don't explicitly show which state is target state or not. Instead, we evaluate the interest that should be paid on a certain state with the SP of the state. To the best of our knowledge, this is the first work using SP's (or SD) to guide control state space exploring.

The rest of the paper is organized as follows. Section II is the overview of our method and three main steps are briefly introduced. The details of the steps are presented in Section III, IV, and V, respectively. Section VI shows the experimental results. Conclusions are presented in Section VII.

II. METHOD OVERVIEW

This work focuses on functional test generation for exploring the control space of the DUV. SP's of the DTMC

This work was supported in part by National Natural Science Foundation of China (NSFC) under grant No. (61176040, 61221062), and in part by National Basic Research Program of China (973) under grant No. 2011CB302501.
978-3-9815370-2-4/DATE14/©2014 EDAA

modeling the control FSM is used to guide the exploring process. As shown in Fig. 1, our method contains three main steps:

1.Extraction of the control FSM from the abstract DUV. As shown in Fig. 1 (a), the DUV can be split into control path and data path, while the control path is in the dashed rectangle. The control FSM is generated from the control path of the DUV.

2.SPs calculation for the DTMC modeling the control FSM. As shown in Fig. 1 (b), the digital beside each node is the steady-state probability of each state. A low SP means that it is not easy for a random test sequence to drive the FSM to this state.

3.Functional test generation guided by SP's. Concrete test vectors are generated to drive different traces to traverse all the abstract states. During traversing, a concrete state is exercised with high preference if the corresponding abstract state of this concrete state has a low SP. Fig.1 (c) shows an example of an abstract trace corresponding to a concrete trace. The trace starts from the reset state, and at each clock cycle, a concrete test vector is generated to drive the trace going deep in the FSM. As CYCLE2 shows, of the two next states of state B, the state with the smaller SP (i.e., 0.21) is more likely to be exercised using the proposed strategies.

The three steps will be described in detail in the next three sections.

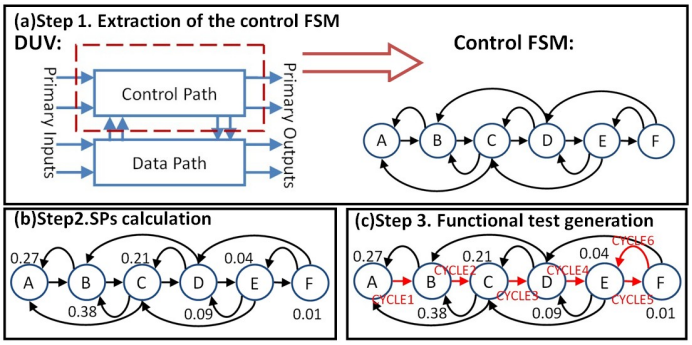

Fig. 1. Method overview: 3 steps

III. CONTROL FSM EXTRACTION

There are two main steps for getting the control FSM of the DUV. Step 1 is to get an abstract design containing only control latches and related signals. Step 2 is to translate the abstract design to an state transition graph(STG).

In step 1, the control path latches are distinguished from the date path latches by annotation(like in[1-3]). We believe it is easy for designers to distinguish the control path latches from the data path latches. So we don't focus on how to distinguish them automatically. Instead, we make annotations in the code, thus the latches annotated are considered as the control path latches while the rest ones are the data path latches. Then the signals which are not the fan-ins of the control path latches are removed from the design. The data path latches are considered as the primary inputs of the abstract design. Consequently, the fan-ins of the data path latches are also removed .

In step 2, the abstract design is translated to the control FSM which is represented as an STG. We use VIS[9] to get the BLIF representation of the abstract design, and then use MVSIS[10] to translate the BLIF file to an STG which is an explicit representation of the control FSM. Each abstract state corresponds to a set of concrete states.

The explicit representation of the control FSM will hinder the scalability to large designs. It is possible to implement the proposed method on an implicitly represented FSM, such as BDDs, which is not the focus of this paper.

IV. STATIONARY DISTRIBUTION CALCULATION

A. Modeling

As mentioned before, the control FSM can be modeled as a DTMC. Then a state in the FSM corresponds to a state in the DTMC, and the probability of the input space attached on a transition of the FSM becomes the probability of a corresponding transition in the DTMC. The simulation of a random test sequence of N clock cycles can be modeled as a random walk which starts from the initial state and contains N steps on the DTMC.

B. Stationary Distribution and Steady-state probability

Given a DTMC with (n+1) states including the initial state, suppose p_{ij} is the probability of that state j will be the next state if state i is the current state, we have the 1-step transition probability matrix $\mathbf{P}$:

$$\mathbf{P}=\begin{bmatrix} p_{00} & \cdots & p_{0n} \\ \vdots & \ddots & \vdots \\ p_{n0} & \cdots & p_{nn} \end{bmatrix} \quad (1)$$

where $p_{ij}=|E_{ij}|/(\sum_{k=0}^{n}|E_{ik}|)$, $|E_{ij}|$ is the number of test vectors which can drive a transition from state i to state j.

We can get a distribution $\boldsymbol{\pi}=[\pi_0,\pi_1,\pi_2,\ldots\pi_i,\ldots,\pi_n]$ on the state space S. $\boldsymbol{\pi}$ is a stationary distribution (SD) [11], if

$$\boldsymbol{\pi}=\boldsymbol{\pi}\mathbf{P}, \text{ i.e. for } \forall j\in S,\ \pi_j=\sum_{i\in S}\pi_i p_{ij}. \quad (2)$$

The steady-state probability (SP) of state i is π_i of the SD $\boldsymbol{\pi}$.

The SP of a certain state is actually the probability of the state being reached by all states of the state space, as shown in (2). A random walk on a DTMC converges to the SD [11]. So the SP can reflect how hard a state of the FSM modeled by the DTMC can be reached by a random test sequence.

We can get an approximate SD by iterative calculation. Given an arbitrary probability distribution $\mathbf{q}=[q_0,q_1,\ldots,q_i,\ldots,q_n]$, an approximate $\boldsymbol{\pi}^*$ can be calculated using the following equation:

$$\boldsymbol{\pi}^*=\mathbf{q}\mathbf{P}^t \quad (3)$$

It can be proved that $\boldsymbol{\pi}^*$ converges to the accurate SD at an exponential rate [12].

V. FUNCTIONAL TEST GENERATION

The principle of the SP's guiding strategy is that a state whose corresponding abstract state has a lower SP will have a

higher priority to be exercised in each simulation step, i.e. the hard-to-reach states are preferred to be exercised. During simulation, concrete test vectors are generated to drive different traces on the concrete design. Each trace, starting from the reset state, can cover some control FSM states(abstract states) corresponding to the concrete states in this trace. The simulation stops when all the abstract states are covered.

Among the input signals of the original design, some are input signals of the control path, such as α in Fig.2. These input signals can be driven directly and are set to the required values of the transition condition(e.g. "00" of the transition condition (00 X1)). The rest inputs of the design (such as β) are driven by constrained random vectors. The values of the control FSM latches(which constitute a subset of the original design latches), are read each simulation cycle and the current state of the abstract model is identified. A state is selected as the next state using the guidance of SP's and the test vectors which can drive the transition from the current state to the selected state is generated. This process is repeated until no new state could be exercised, after which a new trace is initiated from the reset state.

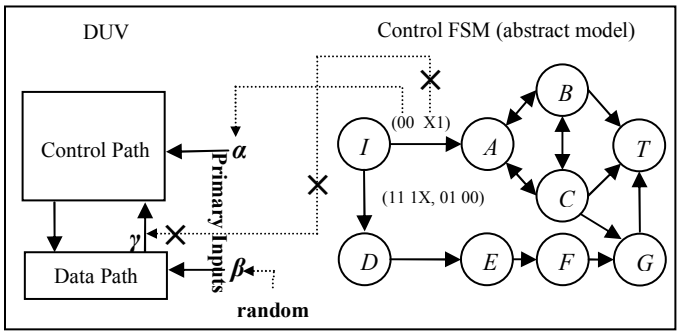

Fig. 2. An example of test generation process

In each simulation cycle, the next state is probabilistically selected from the feasible subsequent states of the current state. The probability of one feasible state i being selected form k feasible states, denoted as pe_i, is evaluated by the SP's of their corresponding abstract states, which is calculated in (4).

$$pe_i = SP_i^{-1} / (\sum_j SP_j^{-1}) \quad (4)$$

As implied in (4), states with lower SP's are more likely to be selected.

Apart from the feasible states, there may be states to which no test vector can drive the transition from the current state. The reason is that the input signals of the abstract model, as shown in Fig.2, contains both the primary inputs (see α) and the signals from the data path (see γ). The primary inputs can be driven directly while the signals from the data path, which are usually fan-outs of the data path latches, cannot be driven directly. For instance, the condition of the transition from I to A requires that signals from the data path are set to "X1"(see the last two bits of the vector (00 X1)). But this condition may conflicts with the status of the data path in this simulation step, resulting that A cannot be reached.

During simulation, as the status of the data path is constantly changing, an unfeasible transition in a certain simulation cycle may become feasible in another cycle or another trace. However, if the condition for a transition has never been satisfied, this transition is probably a dead-end transition, caused by the discrepancies between the abstract model and the original design. Dead-end transitions are not deterministically identified in this work. Instead, if the condition of a transition has conflicted with the data path status for N times before it is first met, this transition is indentified as a dead-end transition. The threshold N is set as 10 times the expectation of the number of times the condition should be first met. For example, N for the transition I to A ("X1") is 20 (i.e., $10*2^1$). The traces following up will dodge these dead-end transitions.

VI. EXPERIMENTS

In our experiments, the control path is abstracted from the DUV manually. The SD calculation and the test generation algorithm are implemented in C++ language. The randomization of the data path inputs is implemented in Systemverilog. All experiments are run on an Intel Dual-Core 2.60GHz Linux machine with 2GB memory. Experiments are conducted on: (1) two ITC99 benchmark circuits, b10 and b12; (2) mips16, a 16-bit MIPS core from opencores [13]; (3)mips789, a 32-bit MIPS core from opencores[13].

A. Comparisons with constrained random test generation

To validate the efficiency of our method in test generation for covering all control FSM states, state coverage of the control FSM are made between our method and the constrained random test generation method, as shown in Fig.3. The constraints used in our method and the constrained random method are only for ensuring legal inputs based on the functional specification. It takes too many clock cycles of simulation for the constrained random method to reach 100% state coverage. For comparison, we evaluate the first 500,000 cycles of simulation. Using our method, all reachable control FSM states are traversed for b10, b12 and mips16 within 250,000 clock cycles. While using the constrained random method in 500,000 cycles of simulation, 98.54% state coverage is reached for b10, 5.69% for b12 and 56.95% for mips16. For Mips32, 100% control FSM coverage is achieved in 2,250,000 clock cycles using our method, while only 10.7% control FSM coverage is obtained using the constrained random method.

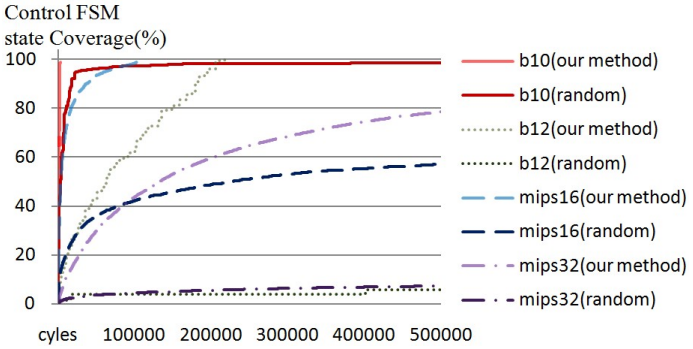

Fig. 3. Comparison of Control FSM state coverage

B. The effect of SP's guidance

To evaluate the efficiency of the proposed SP's guiding strategy, we implemented two non-SP-guidance methods, named DFS_trace and BFS_trace. The DFS_trace method

adopts the test generation process presented in Section V without SP's guidance (by setting the same SP for each state), which is essentially depth-first search (DFS). BFS, identified by breadth-first search, is the search algorithm for state reachability analysis and counter-example trace generation of most BDD based model checking tools, such as VIS[10]. In the BFS_trace method, SP's guidance is also turned off, traces are generated when breadth-first traversing all states and each trace terminates when it reaches a leaf state. Some comparisons between the non-SP-guidance methods and the proposed method with SP's guidance are made. 505 hardest-to-reach states which have lowest SP's are picked from b12 and simulated with the three methods for 5 times using different random seeds. The average number of clock cycles T for each state being exercised for the first time and the variance of T are shown in Fig. 4. For most states, the average number of clock cycles needed for the state to be exercised with our method using SP's guidance is less than that of not using SP's guidance. It implies that one effect of the SP's guiding strategy is to make the hard-to-reach states to be accessed more easily. The variance of T with our method using guidance is much less than that of not using SP's guidance. It implies that another effect of the SP's guiding strategy is to make the performance of the test generation more stable for exercising hard-to-reach states.

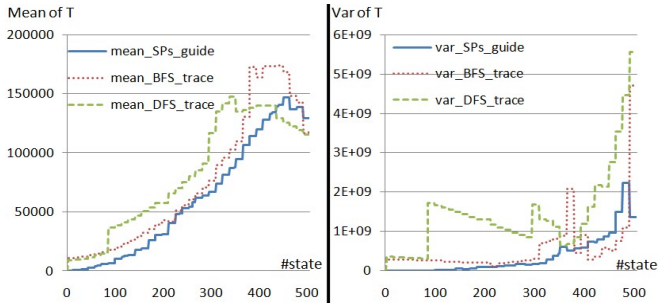

Fig. 4. Comparison of mean of T and variance of T between simulations with SP's guidance and those without SP's guidance

In addition, we use the metric $v(c)$ defined below to evaluate the effect of state exercise.

$$v(c) = \sum_{i=1}^{k} \ln\left(\frac{1}{SPr_i}\right),$$

$$SPr_i = \frac{SP_i}{mean\_SP},$$

where k is the number of already traversed states by c cycles of simulation, SPr_i is the relative steady probability of each already traversed state, SP_i is the steady probability of each traversed state, and $mean\_SP$ is the mean of SP's of all states. $1/SPr_i$ can reflect how hard a state can be reached, in consequence, the more hard-to-reach states in already traversed states and the harder these states to reach, the larger will $v(c)$ be. Comparisons between DFS_trace, BFS_trace and the SP's guiding strategy are shown in Fig. 5. The comparisons are conducted on b12 for all the states. The horizontal axis represents the number of simulation cycles and the vertical axis represents $v(c)$. The comparisons show that $v(c)$ with SP's guiding strategy is larger than that with DFS_trace during the simulation and is larger than that with BFS_trace during the early half stage of simulation, which means the SP's guiding strategy leads the verification effort to exercising more valuable states earlier. When verification resources are limited and 100% state coverage cannot be reached, more hard-to-reach states will be exercised in the early stage of verification using the SP's guiding strategy.

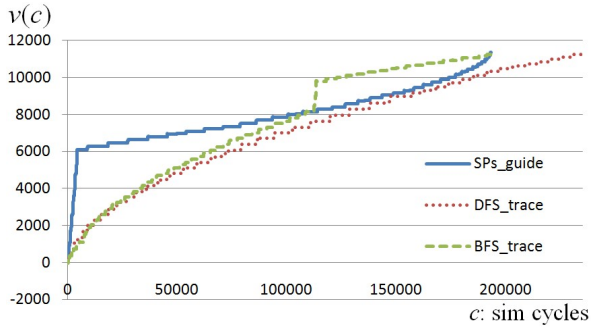

Fig. 5. Comparions of the effect of state exercise

VII. Conclusion

A novel method for functional test generation aiming at exploring control space of the design is presented in this paper. The information of steady-state probabilities of the abstract design's control FSM is used to guide test generation. The guiding strategy plays an important role in traversing hard-to-reach states of the DUV. Experimental results show our method exhibits significantly better performance than the constrained random simulation method in exploring control state space, and is effective in exercising hard-to-reach states.

References

[1] R. C. Ho, C. Han Yang, M. A. Horowitz, and D. L. Dill, "Architecture validation for processors," *in Proc. Int. Symp. Comput. Architecture*, 1995, pp. 404-413.

[2] D. Geist, M. Farkas, A. Landver, Y. Lichtenstein, S. Ur, and Y. Wolfsthal, "Coverage-directed test generation using symbolic techniques." *in Proc. Int. Conf. on Formal Methods in Comput.-Aided Des.*, 1996, pp. 143-158

[3] D. Moundanos, J. A. Abraham, and Y. V. Hoskote, "Abstraction techniques for validation coverage analysis and test generation," *IEEE Trans. Comput.*, vol. 47, pp. 2-14, 1998.

[4] A. Kuehlmann, K. L. McMillan, and R. K. Brayton, "Probabilistic state space search," *in Digest of Technical Papers of Int. Conf. on Comput.-Aided Des.*, 1999, pp. 574-579.

[5] C. H. Yang and D. L. Dill, "Validation with guided search of the state space," *in Proc. Des. Automation Conf.*, 1998, pp. 599-604.

[6] I. Wagner, V. Bertacco, and T. Austin, "StressTest: an automatic approach to test generation via activity monitors," *in Proc. Des. Automation Conf.*, 2005, pp. 783-788.

[7] M. Li and M. S. Hsiao, "An ant colony optimization technique for abstraction-guided state justification," *in Proc. Int. Test Conf.*, 2009, Paper 16.2.

[8] T. Zhang, T. Lv, and X. Li, "An abstraction-guided simulation approach using Markov models for microprocessor verification," *Des., Automation & Test in Europe Conf. & Exhibition*, 2010, pp. 484-489.

[9] http://vlsi.colorado.edu/~vis/.

[10] http://embedded.eecs.berkeley.edu/mvsis/.

[11] M. Mitzenmacher and E. Upfal. (2005). *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge: Cambridge University Press.

[12] R. Motwani and P. Raghavan. (1995). *Randomized Algorithms*. Cambridge: Cambridge University Press.

[13] http://opencores.org/project.