Microprocessors and Microsystems 41 (2016) 1–11

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Improving the efficiency of functional verification based on test prioritization

Shupeng Wang, Kai Huang[*]

College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou 310027, China

ARTICLE INFO

Keywords:
Functional verification
Simulation-based verification
Coverage metrics
Test prioritization
k-means algorithm

ABSTRACT

Functional verification has become the key bottleneck that delays time-to-market during the embedded system design process. And simulation-based verification is the mainstream practice in functional verification due to its flexibility and scalability. In practice, the success of the simulation-based verification highly depends on the quality of functional tests in use which is usually evaluated by coverage metrics. Since test prioritization can provide a way to simulate the more important tests which can improve the coverage metrics evidently earlier, we propose a test prioritization approach based on the clustering algorithm to obtain a high coverage level earlier in the simulation process. The *k*-means algorithm, which is one of the most popular clustering algorithms and usually used for the test prioritization, has some shortcomings which have an effect on the effectiveness of test prioritization. Thus we propose three enhanced *k*-means algorithms to overcome these shortcomings and improve the effectiveness of the test prioritization. Then the functional tests in the simulation environment can be ordered with the test prioritization based on the enhanced *k*-means algorithms. Finally, the more important tests, which can improve the coverage metrics evidently, can be selected and simulated early within the limited simulation time. Experimental results show that the enhanced *k*-means algorithms are more accurate and efficient than the standard *k*-means algorithm for the test prioritization, especially the third enhanced *k*-means algorithm. In comparison with simulating all the tests randomly, the more important tests, which are selected with the test prioritization based on the third enhanced *k*-means algorithm, achieve almost the same coverage metrics in a shorter time, which achieves a 90% simulation time saving.

1. Introduction

Functional verification has become the bottleneck in the system design development cycle due to the increasing complexity of hardware design and the never ending pressure of shorter time-to-market in recent years [1]. Since simulation-based verification can locate the errors rapidly and it is not limited by the size of embedded systems, it has been the most used method for functional verification. In a practical simulation process, a test generator produces extensive functional tests which are usually assembly programs. Then these functional tests are then fed in parallel to a design under test (DUT) and its reference model to check the functional correctness of the embedded systems. In a state-of-the-art verification flow, a test plan is first created, specifying the tasks of the DUT to be verified. The completeness of these tasks is usually measured by coverage metrics [2] including code coverage and functional coverage. Code coverage, such as branch, condition and line coverage, is obtained by evaluating the hardware code execution. Meanwhile, functional coverage is obtained by evaluating function points which are the combination of the characteristics of the DUT and a series of considerable events that must be verified. The coverage metrics can locate the functions that have not been verified and evaluate the progress of functional verification. The simulation-based verification is very effective to ensure the integrity and functional correctness of embedded systems with the help of coverage metrics, however its success in terms of both simulation cost spent and finial verification quality achieved, highly depends on the quality of the functional tests in the simulation environment. The major drawback of mainstream simulation is the difficulty of producing and selecting the functional tests that can lead to the desired coverage level in a short time. Typically, functional tests are divided into three categories, direct tests manually written by designers, random tests created by test generators randomly, and coverage-directed tests generated dynamically based on an algorithm to achieve higher coverage metrics [3,4].

Coverage-directed test generation (CDTG) techniques dynamically analyze coverage results and automatically update the test generation process (usually by changing constraints of the

[*] Corresponding author. Tel.: +86 13958081772.
E-mail addresses: wangsp@vlsi.zju.edu.cn (S. Wang), huangk@vlsi.zju.edu.cn (K. Huang).

http://dx.doi.org/10.1016/j.micpro.2015.12.001
0141-9331/© 2015 Elsevier B.V. All rights reserved.

generator) according to the simulation results and in this way improve the efficiency of functional verification. CDTG techniques employ a variety of learning techniques such as Bayesian Networks [5–7], Markov Models [8,9], Genetic Algorithms [1,10,11] and Inductive Logic Programming [12]. Katz et al. proposed to learn test knowledge from micro-architectural behavior and embed the knowledge into test generator to produce more effective tests [13]. In [14], the authors proposed a methodology based on consistency algorithm to attain faster coverage. While many promising techniques have been proposed, CDTG remains an on-going and active research area [15].

In practice, random tests as well as direct tests are mostly used to ensure the functional correctness of embedded systems. Direct tests are applied to cover corner cases and important features of the DUT. Unfortunately, they are often not portable, even across multiple proliferations of the same hardware design, and must be virtually recreated from scratch each time [9]. Random tests are generated by a test generator based on the test templates provided by designers which define the structure of the desired test, along with primitives to control the randomization of the related data, such as op-codes, register operands, and memory addresses [16,17]. Random testing is a long-standing approach used to locate design errors, where thousands of random tests can be created by the test generator based on the test templates easily and quickly. In comparison with direct tests, random tests are completed with minimal manual involvement. However, simulating all the random tests in a simulation environment takes a long time and it is unnecessary, as many tests can only cover the similar coverage space and helpless to improve the coverage metrics evidently. Meanwhile it is unaffordable to simulate all the generated random tests due to time and computational power constraints. One way to help this situation is to apply test prioritization that reorders tests and identifies more important tests that are likely lead to the high coverage level. Then these important tests can be simulated early within the limited simulation time. With this approach, the chances to achieve the desired coverage level in a short time can be increased and the efficiency of functional verification can be improved.

To date, various test prioritization techniques [18–22] have been proposed and empirically studied. Most of researchers used code coverage information to implement prioritization techniques, and recent prioritization techniques used other types of code information, such as slices [23], change history [24], or code modification information and fault proneness of code [25]. Further, numerous empirical studies showed that prioritization techniques that use source code information can improve the effectiveness of testing [26–28]. And the researchers found that a clustering approach could help improve the effectiveness of prioritization [29,30]. Leon and Podgurski proposed a test prioritization technique incorporating sampling methods that select tests from clusters that are formed based on distributions of test execution profiles [31]. Their technique utilizes clustering approach in test prioritization, but they simply apply random selection of tests from clusters for prioritization. In contrast, in [29], the authors reduced the required number of pair-wise comparisons significantly by clustering tests. In [30], the authors implemented new prioritization techniques that incorporate a clustering approach and utilize code coverage, code complexity, and history data on real faults. Though the above-mentioned test prioritization techniques are very efficient in testing, to the best of our knowledge, these test prioritization techniques are all used in the software testing rather than the functional verification of embedded systems. We conjecture that the test prioritization technique, which utilizes the clustering approach, can also play a role in the functional verification of embedded systems. If this conjecture is correct, we could manage the functional verification of embedded systems more efficiently with the test prioritization technique that can utilize clustering approaches. For instance, if we do not have enough time to simulate all the tests in a simulation environment, by simulating a limited number of tests from each test set, we still could have a better chance to obtain a higher coverage level than otherwise. In this paper, we investigate whether this conjecture is correct and use the k-means algorithm [32,33] to improve the test prioritization technique in the simulation-based verification. The *k*-means is one of the simplest but most popular unsupervised learning algorithms that has been extensively used to cluster data points. The algorithm is easy to implement and apply even on large data sets and therefore the *k*-mean clustering technique has been successfully applied in various areas [34,35], ranging from statistics, data mining to general information technology. The *k*-means algorithm can simplify the test prioritization process by dividing tests into a set of test sets that have similar coverage space. However, the standard *k*-means algorithm has some shortcomings. We make several attempts to overcome these shortcomings and improve the efficiency of the *k*-means algorithm and the effectiveness of the test prioritization.

The rest of the paper is organized as below. Section 2 describes the proposed test prioritization technique that incorporates a clustering approach briefly. Section 3 shows the proposed test encoding method to convert tests into a set of feature vectors. Section 4 introduces the coverage estimation flow of un-simulated tests based on a coverage database and then functional tests in a simulation environment can be indicated by a set of data points based on their estimated coverage. Section 5 describes the clustering approach with the enhanced *k*-means algorithms and the test prioritization process based on the estimated coverage of tests is shown in Section 6. Section 7 presents the experimental results on a high-performance 32-bit quad-processor system. Finally, the main conclusions are summarized in the final section.

2. Overview of the proposed approach

As shown in Fig. 1, the basic steps of the proposed approach are following:

1. The first step is to direct the test generator to produce a large number of single-instruction tests and common used sequences called database tests. Then these database tests are indicated by a set of feature vectors with the proposed test encoding approach.
2. The next step is to simulate these database tests and obtain their true coverage. Then we can build a coverage database based on their true coverage and feature vectors. This coverage database can be used to estimate the coverage of un-simulated tests.
3. The third step is to direct the test generator to produce n functional tests to be the total test set $T = \{t_1, \ldots, t_n\}$. Then these functional tests are converted into a set of feature vectors and estimated their coverage with the coverage database. Based on their estimated coverage, these functional tests can be indicated with a set of data points embedded in multi-dimensional space.
4. The next step is to partition these data points into k clusters with the proposed enhanced k-means algorithms. These functional tests are classified into k test sets accordingly based on the correspondence between the tests and the data points.
5. The fifth step is the prioritization of functional tests in each test set based on the estimated coverage. In this way, the functional tests in each test set are prioritized.
6. The final step is to visit each test set with the round robin method and reorder these functional tests. Then these reordered tests are simulated in order and the more important