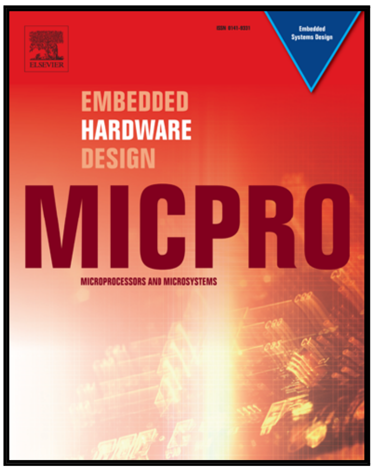


Improving the efficiency of functional verification based on test prioritization

Shupeng Wang , Kai Huang

PII:	S0141-9331(15)00200-8
DOI:	10.1016/j.micpro.2015.12.001
Reference:	MICPRO 2319
To appear in:	*Microprocessors and Microsystems*
Received date:	31 July 2015
Revised date:	25 November 2015
Accepted date:	14 December 2015

Please cite this article as: Shupeng Wang , Kai Huang , Improving the efficiency of functional verification based on test prioritization, *Microprocessors and Microsystems* (2015), doi: 10.1016/j.micpro.2015.12.001

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

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

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.

Key words: Functional verification; simulation-based verification; coverage metrics; test prioritization; k-means algorithm

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 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 tests, which can improve the coverage metrics evidently, can be simulated earlier in the simulation process.

3. Test encoding method

It is required that test encoding methods can be used to indicate the feature of tests not only in the simulation environment for verifying the single-core processor, but also in the simulation environment for verifying the multi-core processor. As the information of a test is complicated, unstructured and abstract, it needs to be represented in an organized way as defined by a feature vector. For a test, its feature set mainly comes from three information sources [36]: the first is the configuration of a set of the control registers that have a significant effect on the system status, such as L1 control register and status register. The configuration of these registers usually brings many kinds of scenarios and cover large verification coverage space. The second is the information and characteristic of every instruction, such as op-code and the associated address which is the address of the data accessed by the instruction. The third is data dependency and address dependency in a sequence of instructions, and the distance between the dependent instructions which may create scenarios of interest. For instance, the overlapping of load instructions and store instructions may lead to many different scenarios in multi-core processor systems. As the feature set only depends on Instruction Set Architecture (ISA), the feature set can be reused for generations of functional verification compatible the ISA and the development of the feature set can be seen one-cost.

Based on the feature set, each single instruction can be converted into a 5-element feature vector. All the configurations of the control registers are encoded and measured by the first element of the feature vector. All op-codes in the instruction set are encoded and the one included by this instruction is indicated by the second element of the feature vector. Then all the system address domains are encoded and the address domain of the associated address in the instruction is measured by the third element. Further, the fourth element of the feature vector is set to indicate whether this instruction depends on the instructions ahead of it, which is called the dependency relation for short. The final element of the feature vector indicates the distance between this instruction and its nearest dependent instruction that is called the dependent distance for short.

Fig. 2 illustrates a simple example of test encoding process. The test includes ten instructions and our goal is to covert each instruction into a 5-element feature vector. As shown in Fig. 2, the third instruction is to modify the system state from *SLEEP* (low power state) to *ACTIVE* (normal state), hence the first element of the third feature vector is set to 1. The fourth instruction of these two tests is to load data from the register *r2* to the register *r3*. The second element of the fourth feature vector is set to 2 to indicate the op-code of the fourth instruction is *LDW*. As the address domain of its associated address is cacheable and shareable, the third element of the fourth feature vector is set to 2. The fourth instruction depends on the second one, hence the fourth element of the fourth feature vector is set to 1 and its final element is set to 2 to indicate the dependent distance between these two dependent instructions.

4. Data point mapping based on estimated coverage

In this section, we introduce data point mapping method based on the coverage of tests to indicate functional tests in the simulation environment with a set of data points. As these functional tests are un-simulated and their true coverage is not yet known, we require a method to estimate coverage for an un-simulated test. Fig. 3 shows the process of obtaining a coverage database and the coverage estimation of these functional tests based on the coverage database. For each single instruction in Fig. 3, *Inst* indicates its op-code, while op_1 and op_2 indicate the first operand and the second operand separately. As shown in Fig. 3a, the coverage database is based on the feature vector. For many instances in the database, they come from the simulation of a single-instruction test. However, there is no information about dependencies and this may have a strong effect on the accuracy of the estimated coverage of the un-simulated tests. Thus we summarize ten of the most common sequences, including *ST* after *LRW*, *LD* after *LRW*, *ST* after *LD*, *ST* after *ST*, *LD* after *ST*, *LD* after *LD*, *ST* after *ADD* after *LD*, *ST* after *RSUB* after *LD*, *ST* after *MULT* after *LD*, and *BT* after *COMP* after *LD*. The dependencies in a simulation consist of static dependencies and dynamic dependencies. The static dependencies are included in the test and can be analyzed statically, and the dynamic dependencies take place during the simulation of tests. In addition, most of the static dependencies are related to the presented ten samples and in this paper we try to estimate the coverage of tests statically before the simulation, hence we estimate the coverage of most of the un-simulated sequences consisting of two or three continuous dependent instructions based on these ten samples. And the estimated coverage of the un-simulated sequences consisting of more than three continuous dependent instructions can be obtained by dividing into a set of the proposed samples. A number of single-instruction tests and the above-mentioned multiple-instruction tests are simulated to obtain the coverage database. For every configuration of the control registers, at least ten instances are needed to be included in the coverage database. For each combination of the op-codes in the instruction set and the address domains, the coverage database needs to include at least ten instances. Assume that the number of the system configurations, the number of the op-codes in the instruction set, the address domain and the dependency relation are denoted by *CON*, *OPC*, and *AD* respectively, it is necessary for the coverage database to include at least 10 * (*CON* + *OPC* * *AD*). For each of the ten above-mentioned sequences, the coverage database also needs to include at least ten instances. As shown in Fig. 3b, then for a given un-simulated test consisting of a sequence of instructions, for each instruction *I* we retrieve the coverage from the coverage database based on the feature vector instance that is closest to the feature vector of the instruction *I*. The closeness is decided based on the Hamming-distance calculation between the dependent distances of the instruction *I* and the dependent instance stored in the database. Then the corresponding coverage is retrieved and used for the estimated coverage of the instruction *I*. For instance, for the first instruction in an un-simulated test as shown in Fig. 3b, as the closest feature vector to it is feature vector *z*, its estimated coverage would be the true coverage of the feature vector *z*. And the third instruction depends on the second instruction, and the closest feature vector set consisting of two dependent feature vectors is the feature vector set consisting of feature vector *x* and feature vector *y*, thus the coverage of the two-instruction sequence consisting of the second instruction and the third instruction in the un-simulated test can be estimated to be coverage *y*.

When the estimated coverage of tests has been obtained, we can transform the representation of the coverage by each test to a binary string by using the binary numeric values 1 and 0 to represent covered/not covered coverage, such as the lines of code and the functional points. Table 1 shows an illustrative example of the data mapping process. As shown in Table 1, the embedded system is described in Verilog HDL with 9 lines of code and the line coverage is selected to evaluate the verification procedure. The estimated line coverage of T_1 and T_2 is obtained based on the coverage database. T_1 does not cover the eighth and the ninth lines, meanwhile T_2 does not cover the sixth and the seventh lines. Therefore, the binary representation string for T_1 is “111111100”, and consequently the binary representation string for T_2 is “111110011”. Then these two tests can be indicated by two data points embedded in the nine-dimensional space based on their estimated coverage.

5. Clustering approach

5.1. The standard k-means algorithm

Having created data points used to indicate the functional tests, now we apply the clustering approach to them. The k-means algorithm is one of the most popular and simple clustering algorithms and it is effective in producing clusters for many practical applications. The function of k-means algorithm is grouping *n* data points into *k* disjointed clusters so that the data points in the same cluster are similar, yet the data points belonging to different clusters differ. The standard k-means algorithm consists of two separate phases: the first phase is to select *k* centroids randomly, one for each cluster. The next phase is to calculate the distance between each data point and all the centroids, and associate the data point to the nearest centroid. At this moment, the new centroids are needed to be recalculated, as the inclusion of the new data points may lead to a change in the cluster centroids. Once *k* new centroids are found, a loop is generated to create a new building between data points and the new nearest new centroid. In this way, the *k* centroids may change their locations in a step by step manner. These two phases are repeated until the centroids do not move anymore. This signifies the cluster membership stabilizes. The standard k-means algorithm is outlined below as Algorithm 1.

```
Algorithm 1: The standard  k-means algorithm
Input:
  $D = \{d_1, d_2, \cdots\cdots, d_n\}$  // set of n data points
  k // Number of desired clusters
Steps:
  1.  Select k data-items randomly as initial centroids;
  2.  Repeat
      2.1  Associate each data item $d_i$ its nearest centroid;
      2.2  Calculate the new mean of each cluster;
      Until cluster membership stabilizes.
Output:
  A set of k clusters.
```

The standard k-means algorithm has three shortcomings, including

1. The number of desired clusters, *k*, is defined by users.
2. The initial centroids are selected randomly.
3. The standard algorithm has to calculate the distance from each data point to each centroid when it executes the iteration each time. In this way, its complexity and computation time increases obviously.

5.2. Enhanced k-means algorithms

For these three above-mentioned shortcomings, the definition of the value of *k* and the selection of the initial centroids are critical for final quality of clustering, while the third shortcoming has a strong influence on computation time. As the computation time of the k-means algorithm is much shorter than the simulation time of tests, the third shortcoming of the standard k-means algorithm is not crucial in our application. However, the final quality of clustering has a strong effect on the test prioritization, hence the first two shortcomings are vital in our application. This motivates us to find some methods to overcome the first two shortcomings and improve the efficiency of the test prioritization.

For the first shortcoming of the standard algorithm, Ordered Multiple Runs of k-means (OMRk) [37] is widely used to compute the value of *k*. This method executes k-means repeatedly for an increasing number of clusters (*k*). For each value of *k*, the partitions achieved by the k-means are assessed with the Simplified Silhouette [37] which indicates how compact and separate the clusters of a given partition are. A common practice consists of assuming $k_{max} = \sqrt{n}$ [38], where *n* is the number of the data points.

OMRk is a simple and efficient method to compute the value of *k* and it runs k-means algorithm several times for each *k* from 2 through $\sqrt{n}$, where *n* is the number of data points. After k-means is executed, *n* data points are partitioned into *k* clusters and the corresponding tests are classified into *k* test sets. The Simplified Silhouette is also used to indicate the clustering efficiency. In the proposed method, the similarity between two data points is indicated by the similarity between the coverage of their corresponding tests. As these tests are all un-simulated, their coverage is estimated. Assume that there are two un-simulated tests t_1 and t_2 and their estimated coverage are $C(t_1)$ and $C(t_2)$ respectively, the similarity between them is defined as

$$\frac{|\ C(t_1) \cap C(t_2)\ |}{|\ C(t_1) \cup C(t_2)\ |} \tag{1}$$

Assume that the test t_i belongs to test set *M*, the similarity between t_i and the centroid of test set *M* is denoted by $m(t_i)$. The similarity between t_i and the centroid of another test set *N* is denoted by $q(i, N)$. After computing $q(i, N)$ for the other test sets that are not tests set M, the highest one is retained and termed $n(t_i)$, i.e., $n(t_i) = \max q(i, N)$. This value represents the coverage similarity between t_i and its nearest neighboring test set. The Simplified Silhouette $S(t_i)$ is defined as:

$$S(t_i) = \frac{m(t_i) - n(t_i)}{\max\{m(t_i), n(t_i)\}} \tag{2}$$

The higher the value of the Simplified Silhouette is, the more compact the clusters of the given partition are. Therefore, the partition with the highest value of the Simplified Silhouette is chosen among the obtained solutions.

For the second shortcoming of the standard algorithm, we propose Algorithm 2 to find the initial centroids with better accuracy. The detail of Algorithm 2 is listed as follows.

```
Algorithm 2: Finding the initial centroids
Input:
    $D = \{d_1, d_2, \cdots\cdots, d_n\}$  // Set of $n$ data points
    $k$ // Number of desired clusters
Steps:
    1. Set $m = 1$;
    2. Compute the distance between each data point and all other data points in the set $D$;
    3. Find the farthest pair of data points from the set $D$ which contains these two data points $P_m$ and $P_{m+1}$, Delete these data points from the set $D$;
    4. Find the data point in $D$ that is closest to the data point $P_m$. Delete it from $D$ and make use of this point and $P_m$ to form a data-point set $A_m$ ($1 \leq m \leq k$). At the same time, find the data point that is closest to the data point $P_{m+1}$, delete it from $D$ and form a data-point set $A_{m+1}$;
    5. Find the data point in $D$ that is closest to the da-
```

```
        ta-point set $A_m$, add it to $A_m$ and delete it from $D$.
        Meanwhile, find the data point in $D$ that is closest
        to the data-point set $A_{m+1}$, add it to $A_{m+1}$ and de-
        lete it from $D$;
    6.  Repeat step 5 until the number of data points in
        $A_m$ and $A_{m+1}$ reaches $n/k$;
    7.  If $m < k - 2$
            Set $m$ equals to $m + 2$, find another pair of
            data points from $D$ between which the dis-
            tance is the farthest, Go to step 3;
        Else
            If $m = k - 2$, use all the data points in $D$ to
            form data-point set $A_k$;
    8.  For each data-point set $A_m$ ($1 \le m \le k$), find the
        arithmetic mean of the vectors of data points in
        $A_m$, these means will be the initial centroids.
Output:
    A set of $k$ initial centroids.
```

As the probability for partitioning the data points with long distance between each other into the same cluster is low after executing k-means algorithm, we set them in the different data-point sets at first. First we find the farthest pair of data points in the whole set D. Then the data point, that is closest to one of these two selected data points, is discovered to form data-point set A_m. Similarly, the data point, that is closest to another selected data point, is discovered to form data-points set A_{m+1}. Then, these four data points are deleted from D. Then the data points closest to data-points set A_m are found and added to A_m. Then they are deleted from D. At the same time, the data points which should belong to A_{m+1} are found and deleted from D. This process is repeated until the number of data points in A_m and A_{m+1} reaches the threshold. If m is less than k-2, go back to find another pair of data points from D with the farthest distance and repeat the above steps. If m equals to k-2, the data points in D can used to form the k_{th} data-points set A_k

exactly and the initial centroids are obtained by averaging all the vectors in each data-point set. The proposed method can improve the accuracy of k-means algorithm and reduce its execution time at the same time.

For the third shortcoming of the standard algorithm, during the iteration, the data point may get redistributed to different clusters and the centroids are recalculated by taking the mean of the values of its data points. However, it is not necessary to calculate that distance each time which takes up a lot of execution time especially for large-capacity databases. We select the improved method proposed in [39] to act as Algorithm 3 to solve this problem in this paper. The main idea of this algorithm is setting two simple structures to keep track of the distance between each data point and the centroid of its present nearest cluster. The labels of clusters data points belong to and the distance between data points and the nearest cluster during the each iteration are saved in these two structures which can be used in the next iteration. The distance between this data point and the new centroid is calculated and if this distance is smaller than or equals to the retained distance, this data point keeps staying in the original cluster and there is no need to calculate the distance between this data point and other k-1 centroids. This method makes the improved algorithm faster than the standard k-means algorithm.

In the standard k-means algorithm, the initial centroids are selected quite randomly. As a result, the centroids are recalculated many times before cluster membership stabilizes. Since complete redistribution of data points takes place according to the centroids, this procedure takes time $O(nkl)$, where n is the number of data points, k is the number of desired clusters and l is the number of iterations. Since OMRk runs k-means t times for each k from 2 through $\sqrt{n}$, its overall computational cost is estimated as:

$$O\left(t\cdot n\cdot l\cdot\left(2+3+\cdots+\sqrt{n}\right)\right)\Rightarrow O\left(t\cdot n^2\cdot l\right) \tag{3}$$

The overall time complexity of OMRk can be written for short as $O(n^2)$, since t and l are much less than n. Similarly, the proposed coverage-based method used to compute the value of k runs k-means for each k from 2 through $\sqrt{n}$, its overall time complexity is also $O(n^2)$. Algorithm 2 requires a time complexity of $O(n^2)$ for finding the initial centroids, since the maximum time required here is for computing the distance between each data points and all other data-points in the set D. To assign data points to clusters, Algorithm 3 requires $O(nk)$. Here, some data points remain in its cluster, this requires $O(1)$. However, the other data points move to other clusters based on their relative distance from the new centroid and the old centroid, this requires $O(k)$. Assuming that half of the data points move from their clusters, this requires $O(nk/2)$. As a result, the total time complexity of Algorithm 3 is $O(nk)$.

To overcome the shortcomings of the standard k-means algorithm, we propose three enhanced k-means algorithms to cluster functional tests. In the first enhanced k-means algorithm, Algorithm 2 is used to determine the initial centroids and Algorithm 3 is used to assign every data point to the appropriate cluster. Meanwhile the value of k of this algorithm is still defined by users. Since the time complexity of Algorithm 2 and Algorithm 3 is $O(n^2)$ and $O(nk)$ respectively, where k is much less than n, the total time complexity of the first enhanced k-means algorithm is $O(n^2)$. In the second enhanced k-means algorithm, Algorithm 2 and Algorithm 3 are used to improve the accuracy and speed, and the value of k is computed with OMRk. Since the time complexity of OMRk is $O(n^2)$, the total time complexity of the second enhanced k-means algorithm is also $O(n^2)$. In the third enhanced k-means algorithm, Algorithm 2 and Algorithm 3 are applied to improve the accuracy and speed, while the value of k is computed with the proposed coverage-based method. Since the total time complexity of the proposed coverage-based method is also $O(n^2)$, the total time complexity of the third enhanced k-means algorithm is $O(n^2)$. In other words, the total time complexity of the proposed enhanced k-means algorithms is all $O(n^2)$. Since k and l are less than n, and the total time complexity of the three enhanced k-means algorithms is more than that of the standard k-means algorithm. Since the computation time of the k-means algorithm is much shorter than the simulation time of functional tests, the final quality of clustering is much more important than the computation time for the test prioritization approach. And the proposed enhanced k-means algorithms are all more accurate than the standard k-means algorithm, thus the proposed enhanced k-means algorithms are more efficient than the standard k-means algorithm for the test prioritization.

6. Prioritization

When the data points have been clustered, the tests are accordingly classified into a set of test sets where the functional tests within the same test set have more chances to cover the similar coverage space than those in the other test sets. Then we number the test sets according to the generated sequence and apply the prioritization technique to them. Using the estimated coverage, we can reorder tests in each test set according to the total number of lines of code or functional points covered by them. To obtain a complete set of reordered tests across test sets, we visit each test set using a round robin method. Starting from the first test set, we pick the first test in the set, add it to the prioritized list of test (initially an empty list), and remove the added test from the test set. Then we move the next test set, and repeat the same process until all the tests have been added to the prioritized list. In cases where a test set runs out of tests due to the fact that the number of tests varies with each test set, we skip that set and move to the next one. For instance, suppose there are four test sets consisting of twelve functional tests (T_1, T_2, …, T_{12}) in the simulation environment.

Using our prioritization technique, these twelve tests can be reordered and the prioritized list is [T_1, T_3, T_8, T_{12}, T_2, T_4, T_5, T_{11}, T_{10}, T_6, T_9, T_7], as show in Fig. 4.

7. Experimental results

We selected the CK810MP of Hangzhou C-SKY Microsystems Co., Ltd., to evaluate the feasibility of the proposed method. As shown in Fig. 5, CK810MP system consists of several modified CK810 processors, interconnection, and memory. CK810 is a high-performance 32-bit embedded processor based on CSKY v2 instruction set and its Load Store Unit (LSU) is modified to support cache coherence according to the specification. A number of CK810 processors are connected by a bus-based interconnection that is responsible for maintaining cache coherence and dealing with requests to memory. The data channel and instruction channel are separate to increase bandwidth. Finally, an efficient symmetric multiprocessor, CK810MP, is obtained with the addition of memory. We made extensive experiments with a CK810 quad-core processor system, as the quad-core processor is the mainstream of the embedded systems currently, such as mobile phones and personal computers. In addition, the quad-core processor can meet the performance requirement of most of embedded applications and it is a good trade-off between performance and power. The CSKY v2 instruction set consists of 183 op-codes, and it has about 63 configurations of the control registers and its memory is divided into four domains. In addition, C++ was selected to act as our program language.

In the first experiment, four databases from UCI repository of machine learning databases were selected to test the accuracy and speed of the first enhanced k-means algorithm. These data sets were given as input to the standard k-means algorithm and the first enhanced k-means algorithm. The execution time and accuracy of these two clustering algorithms with the same value of *k*, taken as 3, are shown in Table 2. The accuracy is achieved by comparing the clustering obtained by these two clustering algorithms with the pre-determined clusters already available in the DCI data sets. The results in Table 2 show that the first enhanced k-means algorithm is more accurate and more efficient than the standard k-means algorithm. This means that Algorithm 2 and Algorithm 3 can improve the accuracy and speed of clustering. As Algorithm 2 and Algorithm 3 are used in all the enhanced k-means algorithms, these three enhanced k-means algorithms are all more accurate and efficient than the standard k-means algorithm.

In our simulation experiment, there were 2000 tests each with 80 to 100 instructions. All these tests were first converted into a set of feature vectors and the line coverage was used to evaluate the process of functional verification. Thus, we obtained estimated line coverage of these tests with the coverage database and converted them into a set of data points embedded in the multi-dimensional space based on their estimated line coverage. It took about 170 hours using the constrained random tests generation framework to meet the requirement of the coverage database. However this creation effort can be seen as one-time cost. To give an idea on the accuracy of the coverage estimation of the un-simulated tests, Fig. 6 shows the result based on these 2000 tests used in the experiment. The x-axis represents the accuracy indicated in terms of the percentage of overlap between the estimated coverage and the true simulation coverage of a test. The average estimation accuracy is about 81%. We will show that this accuracy is sufficient for the test prioritization approach to be effective in the later experiment. The estimation accuracy can be improved through adding more instances to the required situations. However, it may take a long time to add these instances.

Bus Interface Unit (BIU) and Load Store Unit (LSU) are two of the most complex units in the CK810MP system. The RTL design of BIU is described in Verilog HDL with about 5320 lines of code, and the RTL design of LSU is described in Verilog HDL with around 7790 lines of code. In our experiments, we focused on the line coverage of BIU and LSU. In the second experiment, all the 2000 tests in the simulation environment were simulated in the random order and the line coverage covered by some tests may be the same as that covered by the simulated tests. Thus, these tests cannot improve the line coverage and cause significant coverage jump. As shown in Fig. 7, the line coverage curves improve hardly in a time frame, the line coverage of BIU and LSU reaches 75% with about 800 functional tests and 1050 functional tests respectively. Finally, the line coverage of BIU reaches 95.9% and the line coverage of LSU reaches 87.3%. Since it takes more than two days to simulate these 2000 tests, the efficiency of functional verification is unacceptable and it is necessary to take some measures.

In the third experiment, we reordered the functional tests in the simulation environment with the proposed test prioritization approach based on the first enhanced k-means algorithm and the standard k-means algorithm respectively. Both of the values of *k* of these two k-means algorithms were set to 23 that was about $\sqrt{n}/2$, where *n* equaled to 2000. Then we simulated these reordered tests in order and the simulation results are shown in Fig. 8. As shown in Fig. 8a, with the test prioritization approach based on the standard k-means algorithm, the line coverage of BIU reaches 75% with about 280 functional tests, which achieves a 65% saving in simulation time. And the line coverage of LSU reaches 75% with about 340 functional tests, obtaining a 67% saving in simulation time. Finally, the line coverage of BIU reaches 88.9% and the line coverage of LSU reaches 78.5% with about 426 functional tests. This simulation result means that more than two days of simulation time can be reduced to about nine hours with the test prioritization approach based on the standard k-means algorithm. Fig. 8b shows the simulation result with the test prioritization approach based on the first enhanced k-means algorithm. As shown in Fig. 8b, the line coverage of BIU reaches 75% with about 210 functional tests, which achieves a 73% saving in simulation time. And the line coverage of LSU reaches 75% with about 265 functional tests, obtaining a 74% saving in simulation time. Finally, the line coverage of BIU reaches 90.3% and the line coverage of LSU reaches 82.5% with about 343 functional tests. This means that two days of simulation time can be reduced to about eight hours. The simulation results shown in Fig. 7 and Fig. 8 mean that the test prioritization approach based on the k-means algorithm can help to reduce the simulation time and improve the efficiency of functional verification. As the Algorithm 2 is used to determine the initial centroids and assign every data point to the appropriate cluster with Algorithm 3 in the first enhanced k-means algorithm, the first enhanced k-means algorithm is more accurate than the standard k-means algorithm. The simulation results also mean that the first enhanced k-means algorithm is more efficient than the standard k-means algorithm for the test prioritization approach. However, the value of *k* is defined by us in these two k-means algorithms, the clustering efficiency of these two k-means algorithm is low and the low clustering efficiency has an effect on the effectiveness of the test prioritization.

In our fourth experiment, the test prioritization approach based on the second enhanced k-means algorithm was applied to reorder the functional tests, and then the reordered tests were simulated in order. The OMRk was used to compute the value of *k* and the highest value of the Simplified Silhouette was obtained when the value of *k* was set to 15. Fig. 9 shows the simulation results with the test prioritization approach based on the second enhanced k-means algorithm. As shown in Fig. 9, with the test prioritization approach, the line coverage of BIU reaches 75% with about 140 functional tests, which achieves an 82% saving in simulation time. In addition, the line coverage of LSU reaches 75% with about 240 functional tests, obtaining a 77% saving in simulation time. Finally, the line coverage of BIU and LSU reaches 91.8% and 84.1% respectively with about 266 functional tests. The simulation results mean that more two days of simulation time can be reduced to about five hours with the test prioritization approach based on the second enhanced k-means algorithm. The value of *k* selected by OMRk is more accurate the value used in the third experiment. However, the OMRk is coverage-independent, thus it is difficult to estimate the final simulation result based on the OMRk before simulation. The clustering efficiency of the second enhanced k-means algorithm is higher than that of the first enhanced k-means algorithm, however it still needs to be improved further.

Finally, the third enhanced k-means algorithm was implemented to improve effectiveness of the test prioritization. Fig. 10 shows the simulation result with the test prioritization approach based on the third enhanced k-means algorithm where *k* was set to 26. As shown in Fig. 10, with the test prioritization based on the third enhanced k-means algorithm, the line coverage of BIU reaches 75% with about 133 functional tests, which achieves an 84% saving in simulation time. And the line coverage of LSU reaches 75% with about 160 functional tests, obtaining an 85% saving in simulation time. Finally, the line coverage of BIU and LSU reaches 94% and 85.4% respectively, which is almost the same as the coverage obtained by simulating all the two thousand functional tests randomly. Meanwhile, this similar line coverage is achieved only with about 208 functional tests, obtaining a 90% saving in simulation time. This means more two days of simulation time can be reduced to about four hours with the test prioritization approach based on the third enhanced k-means algorithm, while maintaining the similar coverage. As the value of *k* is computed with the proposed coverage-based method in the third enhanced k-means algorithm, the efficiency of the third enhanced k-means algorithm for the test prioritization approach is the maximal in all the k-means algorithms proposed in this paper.

8. Conclusions

In this paper, we propose a simple and accurate test encoding method to convert tests into a set of feature vectors, and then a coverage database based on feature vector can be obtained by simulating a set of simple functional tests. Using the coverage database, we can obtain the estimated coverage of un-simulated tests and then all the functional tests in a simulation environment can be indicated by data points embedded in multi-dimensional space based on their estimated coverage. The test prioritization provides a way to simulate more important tests earlier in the simulation process so that we can obtain the high coverage level earlier in the functional verification process. The clustering approach has proved to be a good chance to improve the effectiveness of the test prioritization. The k-means algorithm is one of the most common clustering algorithms, however it has some shortcomings which have an effect on the effectiveness of the test prioritization. We propose three enhanced k-means algorithms to overcome these shortcomings and improve the effectiveness of the test prioritization. Finally, all the functional tests can be reordered with the test prioritization approach based on the enhanced k-means algorithms. The proposed test prioritization approach has been applied in the functional verification process of a high-performance 32-bit quad-processor system. The experimental results show that, the test prioritization approach can help to improve the efficiency of functional verification and the enhanced k-means algorithms are more accurate than the standard k-means algorithm for the test prioritization approach. In comparison with simulating all the tests randomly, the efficiency of functional verification with the test prioritization approach based on the k-means algorithm improves evidently, especially based on the third enhanced k-means algorithm. The line coverage achieved with test prioritization approach based on the third enhanced k-means algorithm is almost the same as that achieved by simulating all the 2000 tests randomly without test prioritization approach. Nevertheless, there is a 90% saving in simulation time. The experimental results demonstrate the benefits of the proposed test prioritization method in terms of both effectiveness and efficiency.

The main limitation of the proposed method is the value of *k*, the number of desired clusters, may be not the most perfect value. Evolving some statistical methods to compute the more accurate *k* is suggested for future research. And the possibility to involve human decision-makers in the process of test prioritization could be one point of investigation.

Acknowledgments

This work is supported by the National Science Foundation of China under Grant No. 61100074, National Science and Technology Major Project of China under Grant No. 2012ZX01039-004 and Fundamental Research Funds for the Central Universities under Grant No. 2013QNA5008. In addition, we would like to thank Xiaomeng Zhang and Tianyi Xie for proof reading the article.

References

[1] A. Samarah, A. Habibi, S. Tahar, N. Kharma, Automated Coverage Directed Test Generation Using a Cell-Based Genetic Algorithm, in: Eleventh Annual IEEE International High-Level Design Validation and Test Workshop, 2006, pp.19-26. [doi:10.1109/HLDVT.2006.319996]

[2] S. Tasiran, K. Keutzer, Coverage metrics for functional validation of hardware designs, IEEE Design & Test 18(4) (2001) 36-45. [dio: 10.1109/54.936247]

[3] O. Guzey, L. -C. Wang, Coverage-directed test generation through automatic constraint extraction, in: IEEE International High Level Design Validation and Test Workshop, 2007, pp. 151-158. [doi:10.1109/HLDVT.2007.4392805]

[4] C. Pacheco, S. K. Lahiri, M. D. Ernst, T. Ball, Feedback-Directed Random Test Generation, in: 29th International Conference on Software Engineering, 2007, pp. 75-84. [do i: 10.1109/ICSE.2007.37]

[5] M. Braun, S. Fine, A. Ziv, Enhancing the efficiency of Bayesian network based coverage directed test generation, in: the IEEE International Workshop on High-Level Design Validation and Test, 2004, pp. 75-80. [dio: 10.1109/HLDVT.2004.1431241]

[6] S. Fine, A. Freund, I. Jaeger, Y. Mansour, Y. Naveh, A. Ziv, Harnessing Machine Learning to Improve the Success Rate of Stimuli Generation, IEEE Transactions on Computers 55(11) (2006) 1344-1355. [dio: 10.1109/TC.2006.183]

[7] D. Baras, L. Fournier, A. Ziv, Automatic Boosting of Cross-Product Coverage Using Bayesian Networks, Hardware and Software: Verification and Testing, in: 4th International Haifa Verification Conference, 2008, pp. 53-67. [dio: 10.1007/978-3-642-01702-5_10]

[8] I. Wagner, V. Bertacco, T. Austin, StressTest: an automatic approach to test generation via activity monitors, in: the 42nd annual Design Automation Conference, 2005, pp. 783-788. [dio: 10.1109/DAC.2005.193922]

[9] I. Wagner, V. Bertacco, T. Austin, Microprocessor Verification via Feedback-Adjusted Markov Models, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 26(6) (2007) 1126-1138. [dio: 10.1109/TCAD.2006.884494]

[10] A. Habibi, S. Tahar, A. Samarah, L. Donglin, O. Ait Mohamed, Efficient assertion based verification using TLM, in: Design, Automation and Test in Europe, 2006, pp. 1-6. [dio: 10.1109/DATE.2006.244005]

[11] H. Shen, W. Wei, Y. Chen, B. Chen, Q. Guo, Coverage Directed Test Generation: Godson Experience, in: 17th Asian Test Symposium, 2008, pp. 321-326. [dio: 10.1109/ATS.2008.42]

[12] K. Eder, P. Flach, H.-W. Hsueh, Towards Automating Simulation-Based Design Verification Using ILP, in: 16th International Conference, 2006, pp. 154-168. [dio: 10.1007/978-3-540-73847-3_20]

[13] Y. Katz, M. Rinmon, A. Ziv, G. Shaked, Learning microarchitectural behaviors to improve stimuli generation quality, in: the 48th Design Automation Conference (DAC), 2011, pp. 848-853. [dio: 10.1145/2024724.2024914]

[14] M.P.J. George, O.A. Mohamed, A coverage driven test generation methodology using consistency algorithm, in: the 5th Asia Symposium on Quality Electronic Design (ASQED), 2013, pp. 27-32. [dio: 10.1109/ASQED.2013.6643559]

[15] P. H. Chang, D. Drnabac, L. -C. Wang, Online selection of effective functional test programs based on novelty detection, in: The International Conference on Computer-Aided Design (ICCAD), 2010, pp. 762-769. [doi: 10.1109/IC CAD.2010.5653868]

[16] M. Behm, J. Ludden, Y. Lichtenstein, M. Rimon, M. Vinov, Industrial experience with test generation languages for processor verification, in: Design Automation Conference, 2004, pp. 36-40. [doi: 10.1145/996566.996578]

[17] R. Emek, I. Jaeger, Y. Naveh, G. Bergman, G. Aloni, Y. Katz, et al., X-Gen: a random test-case generator for systems and SoCs, in: Seventh IEEE International High-Level Design Validation and Test Workshop, 2002, pp. 145-150. [doi:10.1109/HLDVT.2002.1224444]

[18] Z. Li, M. Harman, R. M. Hierons, Search Algorithms for Regression Test Case Prioritization, IEEE Transactions on Software Engineering 33(4) (2007) 225-237. [dio: 10.1109/TSE.2007.38]

[19] C. Hettiarachchi, H. Do, B. Choi, Effective Regression Testing Using Requirements and Risks, in: the Eighth International Conference on Software Security and Reliability, 2004, pp. 157-166. [dio: 10.1109/SERE.2014.29]

[20] S. Elbaum, A. G. Malishevsky, G. Rothermel, Test case prioritization: A family of empirical studies, IEEE Transactions on Software Engineering 28(2) (2002) 159-182. [dio: 10.1109/32.988497]

[21] G. Rothermel, R. Untch, C. Chu, M. J. Harrold, Prioritizing test cases for regression testing, IEEE Transactions on Software Engineering 27(10) (2002) 929-948. [dio: 10.1109/32.962562]

[22] J. Kim, A. Porter, A history-based test prioritization technique for regression testing in resource constrained environments, in: the 24rd

International Conference on Software Engineering, 2002, pp. 119-129.

[23] D. Jeffrey, N. Gupta, Test case prioritization using relevant slices, in: the 30th Annual International Computer Software and Applications Conference, 2006, pp. 411-420. [dio: 10.1109/COMPSAC.2006.80]

[24] M. Sherriff, M. Lake, L. Williams, Prioritization of regression tests using singular value decomposition with empirical change records, in: the 18th IEEE International Symposium on Software Reliability, 2007, pp. 81-90. [dio: 10.1109/ISSRE.2007.25]

[25] S. Mirarab, L. Tahvildari, A prioritization approach for software test cases on Baysian Networks, in: Fundamental Approaches to Software Engineering, 2007, pp. 276-290. [dio: 10.1007/978-3-540-71289-3_22]

[26] A. Malishevsky, G. Rothermel, S. Elbaum, Modeling the cost-benefits tradeoffs for regression testing techniques, in: International Conference on Software Maintenance, 2002, pp. 204-213. [dio: 10.1109/ICSM.2002.1167767]

[27] H. Do, S. Mirarab, L. Tahvildari, G. Rothermel, The effects of time constraints on test case prioritization: A series of controlled experiments, IEEE Transactions on Software Engineering 36(5) (2010) 593-617. [dio: 10.1109/TSE.2010.58]

[28] X. Qu, M. Cohen, G. Rothermel, Configuration-aware regression testing: An empirical study of sampling and prioritization, in: the 31st International Conference on Software Engineering - Companion Volume, 2009, pp. 375-378. [dio: 10.1109/ICSE-COMPANION.2009.5071025]

[29] S. Yoo, M. Harman, P. Tonella, A. Susi, Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge, in: the eighteenth international symposium on Software testing and analysis, 2009, pp. 201-212. [dio: 10.1145/1572272.1572296]

[30] R. Carlson, D. Hyunsook, A. Denton, A clustering approach to improving test case prioritization: An industrial case study, in: the 27th IEEE International Conference on Software Maintenance (ICSM), 2011, pp. 382-391. [dio: 10.1109/ICSM.2011.6080805]

[31] D. Leon, A. Podgurski, A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases, in: the International Symposium on Software Reliability Engineering, 2003, pp. 442-453. [dio: 10.1109/ISSRE.2003.1251065]

[32] M. H. Dunham, Data Mining-Introductory and Advanced Concepts, Pearson Education, India, 2006.

[33] R.V. Singh, M. P. S. Bhatia, Data clustering with modified k-means algorithm, in: 2011 International Conference on Recent Trends in Information Technology (ICRTIT), 2011, pp. 717-721. [dio: 10.1109/ICRTIT.2011.5972376]

[34] D.X. Jiang, C. Tong, A. D. Zhang, Cluster Analysis for Gene Expression Data, IEEE Transactions on Data and Knowledge Engineering 16(11) (2004) 1370-1386. [doi:10. 1109/TKDE.2004.68]

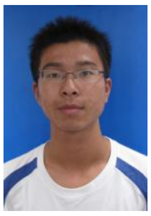
[35] M. Polczynski, M. Polczynski, Using the k-Means Clustering Algorithm to Classify Features for Choropleth Maps, Cartographica: The International Journal for Geographic Information and Geovisualization 49(1) (2014) pp. 69-75. [dio: 10.1353/car.2014.0007]

[36] W. Chen, L.C. Wang, J. Bhadra, M. Abadir, Simulation knowledge extraction and reuse in constrained random processor verification, in: the 50th ACM / EDAC / IEEE Design Automation Conference (DAC), 2013. pp. 1-6.

[37] M. C. Naldi, R. J. G. B. Campello, E. R. Hruschka, A. C. P. L. F. Carvalho, Efficiency issues of evolutionary k-means, Applied Soft Computing, 11(2) (2011) 1938-1952. [dio: 10.1016/j.asoc.2010.06.010]

[38] M.C. Naldi, A. Fontana, R. J. G. B. Campello, Comparsion among methods for k estimation in k-means, in: Ninth International Conference on Intelligent Systems Design and Applications, 2009, pp. 1106-1013. [dio: 10.1109/is da.2009.78]

[39] N. Shi, X. Liu, Y. Guan, Research on k-means Clustering Algorithm: An Improved k-means Clustering Algorithm, in: the Third International Symposium on Intelligent Information Technology and Security Informatics (IITSI), 2010, pp. 63-67. [dio: 10.1109/IITSI.2010.74]



Shupeng Wang was born in Hebei, China, in 1990. He received the BE degree in electronic science and technology from Zhejiang University, Hangzhou, China, in 2012. Currently, he is a PHD student at Zhejiang University. His research interests include multiprocessor system design and verification.

Kai Huang was born in Jiangxi, China, in 1980. He received BSEE from Nanchang University, China, in 2002. Then he obtained PhD in Engineering Circuit and System from Zhejiang University, China in 2008. From May 2006 to August 2006, he worked as a short-term visitor in TIMA laboratory, France. From 2009 to 2011, he worked as post-doc, research assistant in institute of VLSI design, Zhejiang University. In 2010, he also worked as collaborative expert in Verimag Laboratory, France. Currently, he is an associate professor, Department of Information Science & Electronic Engineering, Zhejiang University. His research interests include embedded multiprocessor and SoC system-level design methodology and platform.

Table captions

Table 1 An illustrative example of data point mapping

Table 2 Performance comparison of two algorithms for different datasets

Tab. 1

		T_1	T_2
1	module muxtwo (out, a, b, sel);	√	√
2	input a, b, sel;	√	√
3	output out;	√	√
4	reg out;	√	√
5	always @ (sel or a or b)	√	√
6	if(!sel)	√	×
7	out = a;	√	×
8	else	×	√
9	out = b;	×	√

Tab. 2

Dataset	Standard k-means algorithm		First enhanced k-means algorithm	
	Execution Time (ms)	Accuracy (%)	Execution Time (ms)	Accuracy (%)
E-Coli	65	79.2	70	82.4
Thyroid	60	75.3	50	85.3
Iris	59	84.6	53	93.7
Glass	74	79.5	82	92.1

Figure captions

Fig. 1 Overview of the proposed test prioritization technique

Fig. 2 An illustrative example of test encoding method

Fig. 3 Coverage estimation of un-simulated tests (a) To obtain the coverage database; (b) Coverage estimation

Fig. 4 An illustrative example of test prioritization

Fig. 5 Architecture of CK810MP system

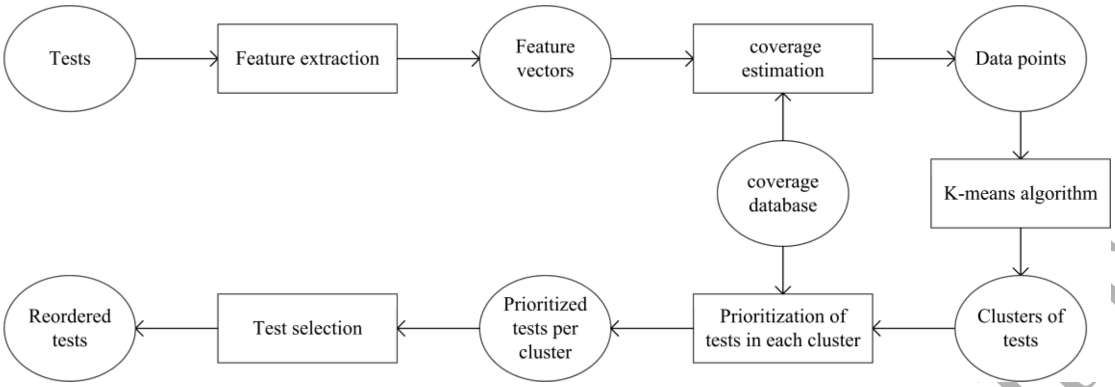
Fig. 6 Estimation accuracy

Fig. 7 Line coverage improvement by simulating tests randomly

Fig. 8 Line coverage improvement with the standard k-means algorithm (a), the first enhanced k-means algorithm (b)

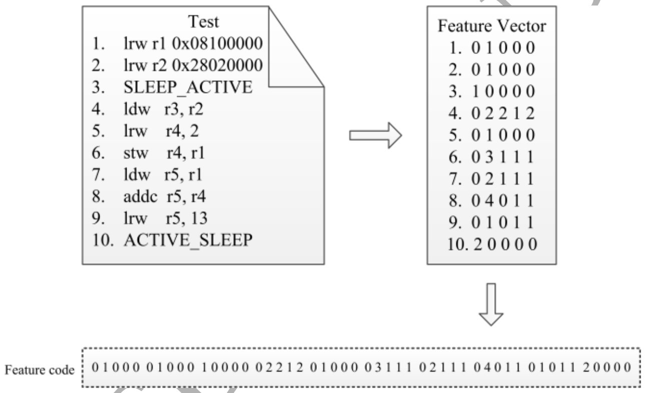
Fig. 9 Line coverage improvement with the second enhanced k-means algorithm

Fig. 10 Line coverage improvement with the third enhanced k-means algorithm



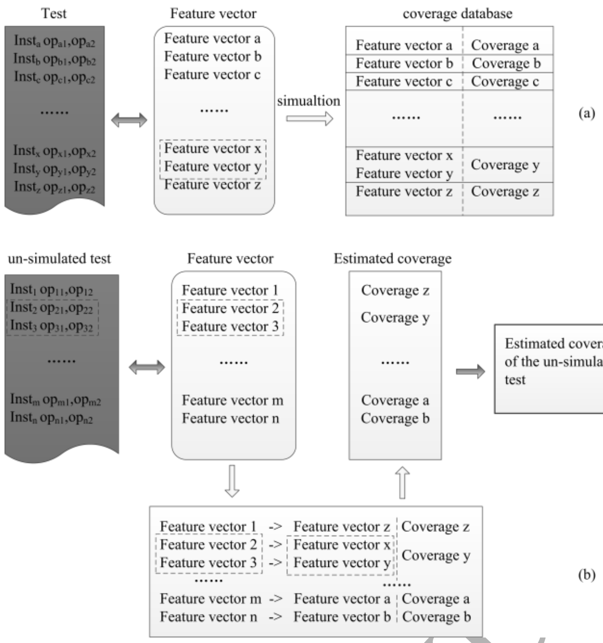
Tests
Feature extraction
Feature vectors
coverage estimation
Data points
coverage database
K-means algorithm
Reordered tests
Test selection
Prioritized tests per cluster
Prioritization of tests in each cluster
Clusters of tests

Fig. 1



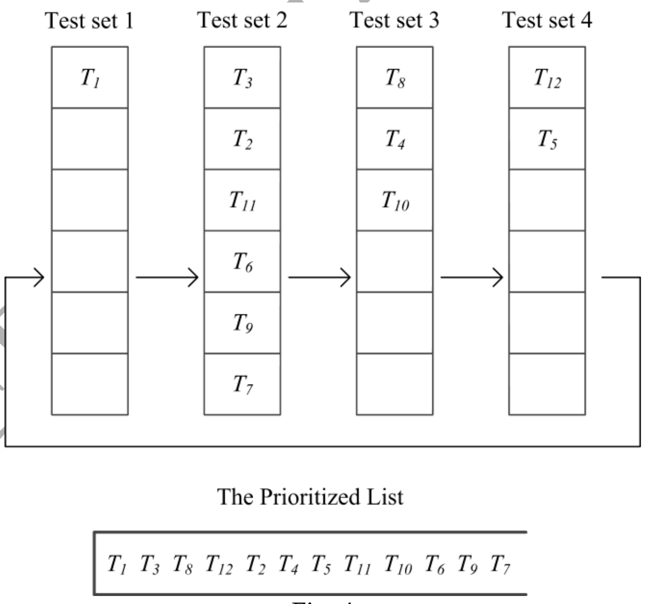
Test
1. lrw r1 0x08100000
2. lrw r2 0x28020000
3. SLEEP_ACTIVE
4. ldw r3, r2
5. lrw r4, 2
6. stw r4, r1
7. ldw r5, r1
8. addc r5, r4
9. lrw r5, 13
10. ACTIVE_SLEEP
Feature Vector
1. 0 1 0 0 0
2. 0 1 0 0 0
3. 1 0 0 0 0
4. 0 2 2 1 2
5. 0 1 0 0 0
6. 0 3 1 1 1
7. 0 2 1 1 1
8. 0 4 0 1 1
9. 0 1 0 1 1
10. 2 0 0 0 0
Feature code
0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 2 2 1 2 0 1 0 0 0 0 3 1 1 1 0 2 1 1 1 0 4 0 1 1 0 1 0 1 1 2 0 0 0 0

Fig. 2



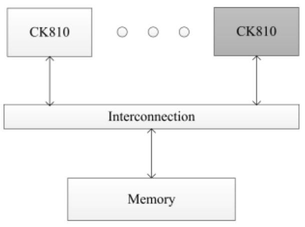
Test
Feature vector
coverage database
simualtion
(a)
un-simulated test
Feature vector
Estimated coverage
Estimated coverage of the un-simulated test
(b)

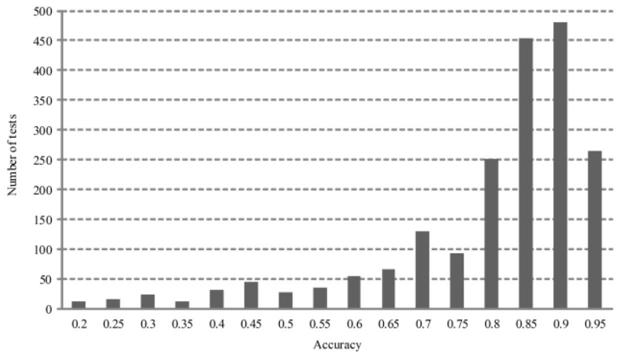
Fig. 3

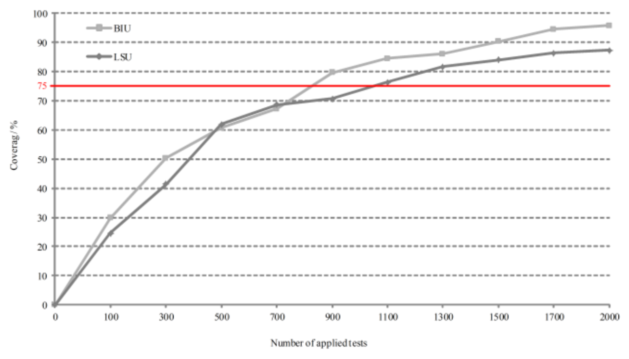


Test set 1
Test set 2
Test set 3
Test set 4
The Prioritized List
T1 T3 T8 T12 T2 T4 T5 T11 T10 T6 T9 T7

Fig. 4

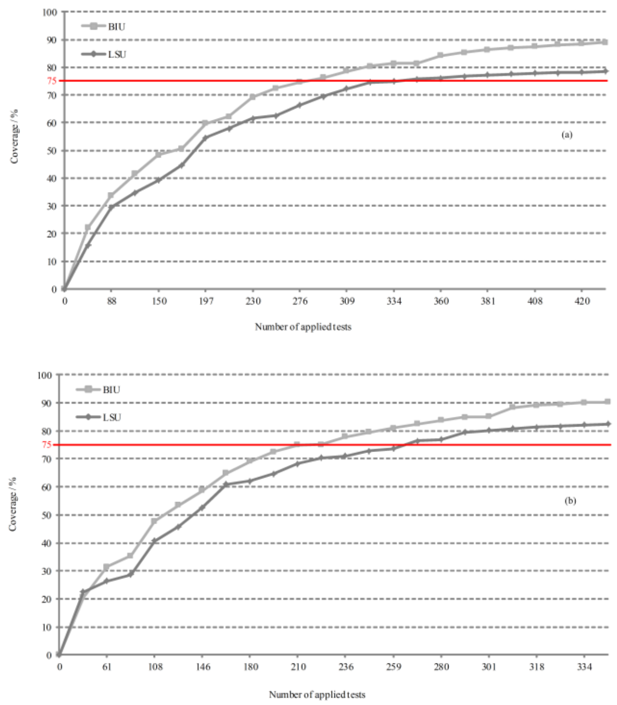

CK810
CK810
Interconnection
Memory
Fig. 5


Number of tests
Accuracy
0
50
100
150
200
250
300
350
400
450
500
0.2
0.25
0.3
0.35
0.4
0.45
0.5
0.55
0.6
0.65
0.7
0.75
0.8
0.85
0.9
0.95
Fig. 6



BIU
LSU
75
Coverag / %
Number of applied tests

Fig. 7



BIU
LSU
75
Coverage / %
(a)
Number of applied tests
BIU
LSU
75
Coverage / %
(b)
Number of applied tests

Fig. 8

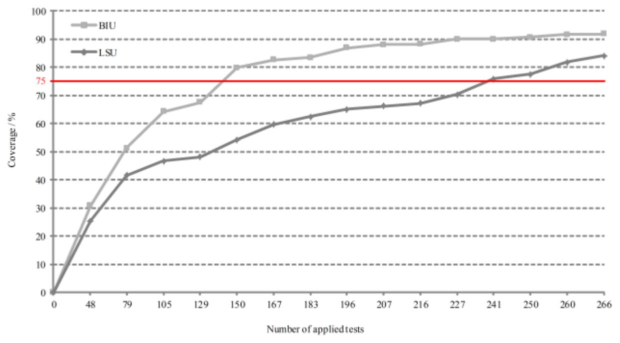

BIU
LSU
Coverage / %
Number of applied tests
0
10
20
30
40
50
60
70
75
80
90
100
0
48
79
105
129
150
167
183
196
207
216
227
241
250
260
266

Fig. 9

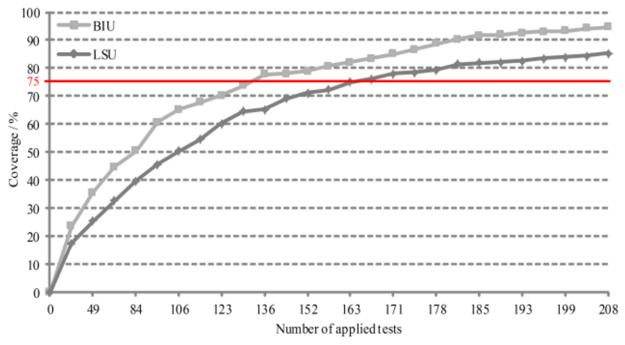

BIU
LSU
Coverage / %
Number of applied tests
0
10
20
30
40
50
60
70
75
80
90
100
0
49
84
106
123
136
152
163
171
178
185
193
199
208

Fig. 10