An Improved SoC Test Scheduling Method Based on Simulated Annealing Algorithm

Jingjing Zheng¹, Zhihang Shen¹, Huaien Gao¹, Bianna Chen¹, Weida Zheng¹ and Xiaoming Xiong¹,²
¹GuangDong University of Technology, China
²Guangzhou National Integrated Circuited Design Industrialization Center for Modern Service Industry
dayzjj1989123@163.com

Abstract. In this paper, we propose an improved SoC test scheduling method based on simulated annealing algorithm (SA). It is our first to disorganize IP core assignment for each TAM to produce a new solution for SA, allocate TAM width for each TAM using greedy algorithm and calculate corresponding testing time. And accepting the core assignment according to the principle of simulated annealing algorithm and finally attain the optimum solution. Simultaneously, we run the test scheduling experiment with the international reference circuits provided by International Test Conference 2002 (ITC’02) and the result shows that our algorithm is superior to the conventional integer linear programming algorithm (ILP), simulated annealing algorithm (SA) and genetic algorithm (GA). When TAM width reaches to 48, 56 and 64, the testing time based on our algorithm is lesser than the classic methods and the optimization rates are 30.74%, 3.32%, 16.13% respectively. Moreover, the testing time based on our algorithm is very close to that of improved genetic algorithm (IGA), which is state-of-the-art at present.

1. Introduction
In recent years, higher requirements of size, power consumption, cost, and development cycle on integrated circuit chip have been put forward in various fields such as consumer electronic, and military electronic systems. It is vital that various kinds of functional modules in system are integrated into one chip so the system on chip (SoC) can be built by reusing Intellectual Property (IP) [1]. The number of data ports of IP cores in SoC are different. Therefore, it becomes increasingly difficult to test each IP core when the test access mechanism (TAM) width is limited. The purpose of SoC test scheduling is to shorten the testing time by assigning IP cores to each TAM reasonably [2]. However, test scheduling, which is a Polynomial non-deterministic (NP) problem [3], has reached a bottleneck with the development of the SoC and has aroused widespread concern in the academic and industrial circles. Many academic organizations and international organizations solved this problem so far: the Test Technology Committee of IEEE’s computer society set up the P1500 IEEE standard. It became a formal standard in 2005 for a unified IP test interface between the IP core provider and the user [4]; V. Iyengar et al [5] regarded the test scheduling as a ILP (Integer Linear Programming) problem to find an optimal method for SoC test; J. Yang et al [6] combined the representation of closed graph transfer and simulated annealing algorithm to optimize the test scheduling; J. Jia et al [3] described the feasible solution set to test scheduling problem as a population based on the adaptability of genetic
algorithm and provided an approximately optimal solution set according to the principle of survival of the fittest evolution; X. Cui et al. [7] proposed a SoC test scheduling method based on ant colony algorithm with the peak temperature constraint. It is obvious that the test time is optimized under the conditions of chip’s thermal safety.

In this paper, an improved SoC test scheduling method based on simulated annealing algorithm (SA) by optimizing IP core assignment for each TAM is proposed. Experimental results show that this algorithm is superior to conventional integer linear programming algorithm (ILP) [5], simulated annealing algorithm (SA) [6] and genetic algorithm (GA) [3] in shortening testing time. In addition, the testing time based on our algorithm is very close to that of improved genetic algorithm (IGA) [8], which is state-of-the-art at present.

The remainder of this paper is organized as follows. Section 2 reviews traditional partition problem of SoC testing optimization proposed by V. Iyengar. Section 3 provides a different partition problem from the traditional one and it is solved by an improved simulated annealing algorithm. Experimental results on benchmark circuits are then shown in Section 4. Finally, Section 5 concludes this paper.

2. The traditional partition problem of SoC testing optimization

Problem definition: a testing structure based on wrapper is proposed in IEEE 1500 standard. The wrapper is an interface of combining cores with TAMs, which provides core’s normal function and utilizes TAM to test cores and accesses data. In order to effectively solve the combinatorial optimization between the wrapper and TAM, V. Iyengar divided this problem into four sub problems: $P_W$, $P_{AW}$, $P_{NAW}$, and $P_{NP,AW}$ depending on combinatorial optimization [5], and it can be seen clearly that the previous problem is sub problem of the next one.

Table 1  Four questions proposed by V. Iyengar

| Question   | Description                                                                 |
|------------|-----------------------------------------------------------------------------|
| $P_W$      | Design a wrapper for a given core, such that the core testing is minimized. |
| $P_{AW}$   | Determine (i) an assignment of cores to TAMs of given widths and (ii) a wrapper design for each core, such that SOC testing time is minimized. (Item (ii) equals $P_W$.) |
| $P_{NAW}$  | Determine (i) a partition of the total TAM width among the given number of TAMs, (ii) an assignment of cores to the TAMs, and (iii) a wrapper design for each core, such that SOC testing time is minimized. (Items (ii) and (iii) together equal $P_{AW}$.) |
| $P_{NP,AW}$| Determine (i) the number of TAMs for the SOC, (ii) a partition of the total TAM width among the TAMs, (iii) an assignment of cores to TAMs, and (iv) a wrapper design for each core, such that SOC testing time is minimized. (Items (ii), (iii) and (iv) together equal $P_{NAW}$.) |

Generally speaking, the $P_W$ can be solved well by BFD(best fit decreasing) heuristic algorithm [5]. For problem $P_{AW}$ and $P_{NAW}$, they can be converted into ILP problem or solved by heuristic algorithms (e.g, simulated annealing algorithm, genetic algorithm, ant colony algorithm). However, These algorithms have shortcomings respectively, i.e. the testing time is too long or the algorithm is too complex. For $P_{NP,AW}$, we generally adopt the method of exhaustion [9]. To improve previous effort, this paper study the four problems differing from V. Iyengar’s angle.

3. Proposed Algorithm

We notice that, given a fixed core assignment for each TAM, it is easy to determine close-to-optimal TAM widths for each TAM with heuristic algorithms [9], and the problem is transformed into how to search a best core assignment for each TAM. Therefore, the previous four problems proposed by V. Iyengar are rewritten as follows:
① $P_W^*$: Given a core, design a wrapper such that the core testing time is minimized.
② $P_{AW}^*$: Given total TAM width $W$, $N_C$ cores in a SoC, the number of TAMs $B$ and a core assignment for each core, determine TAM widths for each TAM such that the SoC testing time is minimized.
③ $P_{NAW}^*$: Given total TAM width $W$, $N_C$ cores in a SoC, the number of TAMs $B$, determine (i) a core assignment for each TAM and (ii) TAM widths for each TAM, such that SoC testing time is minimized.
④ $P_{PNAW}^*$: Given total TAM width $W$ and $N_C$ cores in a SoC, determine (i) the number of TAMs $B$, (ii) a core assignment for each TAM and (iii) TAM widths for each TAM, such that SoC testing time is minimized.

Similarly, the four rewritten problems are progressive, where the former is a sub-problem of the latter. $P_W^*$ is the same as that V. Iyengar defined, which can be solved well with BFD heuristic algorithm [5]. We leave this untouched and denote the solution to $P_W^*$ as Design_wrapper().

3.1 Solution to $P_{AW}^*$ based on greedy algorithm

We define the function solving $P_{AW}^*$ as allocate_TAM_Width(). As this function will be called repeatedly when solving $P_{NAW}^*$, the running time of allocate_TAM_Width() needs to be very short and we propose a solution based on greedy algorithm, shown as follows. Firstly, we assign one-bit wire to each TAM and calculate the testing time of each TAM using Design_wrapper(). The TAM with the longest testing time is recorded as bottleneck_TAM. And then increasing the width of bottleneck_TAM by one until the total width is run out.

1. Allocate one bit width to every TAM;
2. Repeat when the sum of TAM width less than $W$;
3. Calculate the testing time of each TAM $T(i)$ ( $i=1,2,...,B$ ) using Design_wrapper();
4. Record the TAM whose testing time is the longest as bottleneck_TAM;
5. Increase the width of bottleneck_TAM by one;
6. Output TAM width allocation;

To demonstrate the effectiveness of the proposed greedy algorithm, we compare it with the enumerate algorithm. The experiment is based on ITC’02 benchmark SoCs: d695 [10,11]. To simplify the experiment, and without loss of generality, we set the number of TAM to 3. Given total TAM width $W$ and a core assignment, then we can attain the corresponding results shown as table 2, where the solution $\{(1,6,8),(2,3,4,7),(5,9,10)\}$ means core 1, core 6 and core 8 are assigned to TAM 1 and so on. The table shows that the testing time based on the proposed algorithm and the enumerate algorithm are the same, demonstrating the effectiveness of our proposed algorithm.

| Total TAM width $W$ | Core assignment | The result based on the proposed algorithm | The result based on the enumerate algorithm |
|---------------------|-----------------|---------------------------------------------|-------------------------------------------|
| 16                  | $\{(1,6,8),(2,3,4,7),(5,9,10)\}$ | 42568                                       | 42568                                     |
| 24                  | $\{(1,2,10),(3,4,8),(5,6,7,9)\}$ | 28289                                       | 28289                                     |
| 32                  | $\{(1,6,7,10),(2,5),(3,4,8,9)\}$ | 21566                                       | 21566                                     |
| 40                  | $\{(1,5,10),(2,6,7,9),(3,4,8)\}$ | 17901                                       | 17901                                     |
| 48                  | $\{(1,2,6,7,9),(3,4,8),(5,10)\}$ | 16984                                       | 16984                                     |
| 56                  | $\{(1,3,4,8),(2,5,7,10),(6,9)\}$ | 13637                                       | 13637                                     |
| 64                  | $\{(1,2,6,9),(3,4,8),(5,7,10)\}$ | 12941                                       | 12941                                     |
3.2 Solution to $P_{NW}^*$ and $P_{NNAW}^*$ based on SA

The name and inspiration of simulated annealing algorithm come from annealing in metallurgy. First the solid is heated hot enough. As the temperature of the solid goes up, the thermodynamic free energy of the solid increases and inner particles becomes disordered. Then the solid cools down slowly and meanwhile the inner particles become ordered and stay equilibrium state at every temperature. When the temperature decreases to the minimum, the thermodynamic free energy becomes the minimum [12, 13]. In our proposed SA, the testing time represents the thermodynamic free energy $E$ and the parameter $t$ represents the temperature $T$. The algorithm flow chart is shown as figure 1.

![Figure 1. Flow chart of SA.](image)

The main flow of the proposed algorithm is shown as follow. In the beginning, we set the minimum number of TAMs ($TAM_{Num_{min}}$) to 1 and the maximum number of TAMs ($TAM_{Num_{max}}$) to $\min \{N_C, W\}$. Then given number of TAMs, beginning with a random initial core assignment, and keep on searching for its neighbor solutions. Once a feasible solution is obtained, TAM width allocation is performed by allocate_TAM_Width() and the testing time are computed. The new solution will be accepted or abandoned according to the principle of SA. Finally, the best solution is provided during the stochastic search process.

1. $TAM_{Num_{min}} = 1$, $TAM_{Num_{max}} = \min \{N_C, W\}$;
2. for $TAM_{Num} = TAM_{Num_{min}}$ to $TAM_{Num_{max}}$
3. create a random initial core assignment $X_{new}$;
4. call allocate_TAM_Width() to perform TAM width allocation and calculate initial testing time $E_{new}$
5. $E_{best} = E_{new}$, $E_{current} = E_{new}$
6. set initial temperature $T = T_0$
7. while ($T >$ end temperature $T_{end}$)
   // Run a few iterations at same $T$
8. for each iteration
9. randomly move to reach a new core assignment $X_{\text{new}}$;
10. call allocate_TAM_Width () to perform TAM width allocation and calculate new testing time $E_{\text{new}}$;
11. if $E_{\text{new}} < E_{\text{current}}$ or $e^{AE/T} > \text{rand}()$ then
12. $E_{\text{current}} = E_{\text{new}}$
13. update core assignment solution;
14. if $E_{\text{new}} < E_{\text{best}}$ then
15. $E_{\text{best}} = E_{\text{new}}$
else
16. record the best solution
else
17. restore old solution
18. decrease temperature $T$ at a certain rate $\alpha$
19. output the best solution;

3.3 Core assignment based on SA
To avoid repetitiveness of assignment and enhance code efficiency, we have two rules in generating a new random solution:
1. TAM without cores is not allowed;
2. The smallest core index assigned to TAM i is smaller than TAM j, provided i is smaller than j. With the rule, the solution space shrinks to $\frac{1}{B!}$, where $B$ is the number of TAMs;

Following the aforementioned rule, a new solution in SA can be provided by reassignment, which means picking one core from a random set and putting it into another random set. We define this move as M1. To prove the completeness of the above M1, we have a lemma as follow:

**Lemma:** Given an arbitrary solution $A_1, A_2, ..., A_B$, we are able to reach any other solution $A_1^*, A_2^*, ..., A_B^*$ after running M1 finite times.

**Proof**

1. First we use M1 repeatedly to select the cores of $A_1^*$ from $A_2, ..., A_B$ and put them into $A_1$ until all the cores of $A_1^*$ are in $A_1$. (Once a set is empty, we can select a core which does not belong to $A_1^*$ from other sets and put it into the set)
2. Similarly, we are able to reach that all the cores of $A_i^*$ are in $A_i$. ($i = 1, 2, ..., B$)

We assume that the number of cores in $A_1^*$ is $\text{Core\_Num}_1^*$, the number of cores in $A_i$ after finite M1 is $\text{Core\_Num}_i$. And the total number of cores is $N_C$, that is $\sum_1^B \text{Core\_Num}_i^* = N_C$. Based on step 2, we have $A_i^* \subseteq A_i$, that is $\text{Core\_Num}_i \geq \text{Core\_Num}_i^*$ ($i = 1, 2, ..., B$). After limiting M1, the total number of cores does not change. Consequently, $\sum_1^B \text{Core\_Num}_i = \sum_1^B \text{Core\_Num}_i^* = N_C$. The above equation is true only when $\text{Core\_Num}_i = \text{Core\_Num}_i^*$. Therefore, we are able to reach that $A_i = A_i^*(i = 1, 2, ..., B)$.

4. Experimental result
The Comparison among the proposed algorithm and the traditional ILP algorithm [5], simulated annealing algorithm, and genetic algorithm was taken using the ITC’02 benchmark SoC: d695.

Results are shown in table 3 and figure 2. Table 3 shows the testing time of each algorithm when the total TAM width is 16, 24, 32, 40, 48, 56, 64 respectively. By contrast, the testing time by our algorithm is basically less than other classic and state of the art methods. Moreover, Figure 3 shows that the advantage is more obvious when TAM width reaches larger. When TAM width reaches 48,56 and 64, the testing time by our algorithm is less than that by others and the optimization rate are 30.74%, 3.32% and 16.13% respectively.
Table 3. The test result of conventional algorithms and ours

| benchmark SoC | algorithm | Total TAM width |
|---------------|-----------|-----------------|
|               |           | 16   | 24   | 32   | 40   | 48   | 56   | 64   |
| D695          | ILP       | 42952| 28327| 21423| 17210| 16975| 16516| 15694|
|               | SA        | 42361| 28638| 21967| 17414| 15141| 12692| 11242|
|               | GA        | 41949| 28290| 21329| 17084| 16975| 14610| 12960|
|               | Our algorithm | 42268| 28289| 21437| 17523| 14310| 12462| 10869|

To further verify the validity of the proposed method, a comparison with the state-of-the-art improved genetic algorithm was taken [8]. With the same experimental setting, the comparing results is shown in Figure 3. The results shows that the testing time by our method is very close to that by IGA algorithm. This difference becomes smaller when the total TAM width increases. When the TAM width reaches to 48, 56 and 64, the testing time by our algorithm is 14310/12462/10869 and that by IGA is 14310/12540/10571, that is, the rate of optimization is 0/0.62% / -2.82%, respectively.

Figure 2. The comparison result with traditional algorithms

Figure 3. The comparison result with IGA
5. Conclusion
This paper proposed an improved simulated annealing algorithm to solve the SoC test scheduling by searching an optimal core assignment to each TAM. In term of core assignment based on SA, to prove the completeness of the M1, we defined a lemma and given it proof. The experimental result shows much better performance comparing with the classic ILP/SA/GA algorithm and is very close to that of state-of-the-art IGA. Our further work can be done by implementing this algorithm in considering constrains of the power consumption.

Reference
[1] C. P. Yue, D. Luo, G. Zhu, Y. Wang, K. Q. Maqbool and Z. Li, Recent developments in transceiver SoC design for next generation optical networks[C], 2015 Asia-Pacific Microwave Conference (APMC), Nanjing, 2015, pp. 1-3.
[2] Lu Xi and Mo Tingting. Parallel Test Task Search Genetic Algorithm[J]. Microelectronics & Computer, 2015,03:146-150.
[3] Lei Jia and Fang Gang. An Test Scheduling Method of SoC based on Genetic algorithm[J]. Chinese Journal of Scientific Instrument, 2007 28 (4) :15—18.
[4] IEEE Std 1500-2005.IEEE 1500 Standard For Embedded Core Test[S].
[5] V. Iyengar, K. Chakrabarty and E. J. Marinissen, Test wrapper and test access mechanism co-optimization for system-on-chip[C], Test Conference, 2001. Proceedings. International, Baltimore, MD, 2001, pp. 1023-1032.
[6] Yang Jun and Luo Lan. SoC test schedule based on TCG and simulated annealing algorithm[J]. Journal of Circuits and Systems, 2006,05:37--43.
[7] Cui Xiaole, Xiong Zhitian, Cheng Wei and Li Chongren. Thermal-aware SoC test scheduling method based on ant colony optimization[J]. Chinese Journal of Scientific Instrument, 2014, 04:948-953.
[8] Liu Wei and Li Bin. An Improved Test Scheduling Method of SoC Based on Genetic Algorithm. Radio Communications Technology, 2016,42( 2) : 37−40.
[9] Li Jiang, L. Huang and Q. Xu, Test architecture design and optimization for three-dimensional SoCs[C], 2009 Design, Automation & Test in Europe Conference & Exhibition, Nice, 2009, pp. 220-225.
[10] E. J. Marinissen, V. Iyengar and K. Chakrabarty, A set of benchmarks for modular testing of SOCs[C], Test Conference, 2002. Proceedings. International, 2002, pp.519-528.
[11] H. Zhang and V. D. Agrawal, SoC TAM Design to Minimize Test Application Time[C], Test Workshop (NATW), 2015 IEEE 24th North Atlantic, Johnson City, NY, 2015, pp. 55-60.
[12] Xie Yun. Principle and Realization of the Simulated Annealing Algorithm, Numerical Mathematics A Journal of Chinese Universities. 1999,03:212-218.
[13] W. Wu, L. Li and X. Yao, Improved simulated annealing algorithm for task allocation in real-time distributed systems[C], Signal Processing, Communications and Computing (ICSPCC), 2014 IEEE International Conference on, Guilin, 2014, pp. 50-54.