Design of Dynamically Assignmentable TAM Width for Testing Core-Based SOCs

Jiann-Chyi Rau
Department of Electrical Engineering, Tamkang University
151, Ying-Chuan Rd. Tamsui, Taipei Hsien 251, Taiwan, R.O.C
jcrau@ee.tku.edu.tw

Chien-Shiun Chen, and Po-Han Wu
Department of Electrical Engineering, Tamkang University
151, Ying-Chuan Rd. Tamsui, Taipei Hsien 251, Taiwan, R.O.C
{cschen, phwu@ee.tku.edu.tw}

***Abstract*—Test access mechanism (TAM) and testing schedule for System-On-Chip (SOC) are challenging problems. Testing schedule must be effective to minimize testing time, under the constraint of test resources. This paper we presents a new method based on genealized rectangle packing, as two-dimensional packing . A core cuts into many pieces and utilize the design of reconfigurable core wrappers, and is dynamic to change the width of the TAM executing the core test. Therefore, a core can utilize different TAM width to complete test.**

***Keywords*—SOC Testing, TAM, Testing Scheduling**

I. INTRODUCTION

The chip of nowadays is more and more complex. Chips often include a lot of systems; we call System-On-Chip (SOC). Therefore, how to testing SOC is very important problem. For testing a SOC, we need to build a test wrapper for each core, internal scan registers within each core, and a test access mechanism (TAM) [6]. The test wrapper is comprised of a standard cell at each core input and output that enables isolation of the core from the SOC for testing independently. The core providers design the internal scan registers for the necessary Design-For-Testablity (DFT). The TAM is a mechanism to transport test data (test patterns as well as responses). The test control signals between SOC pins and core I/O and internal scan chains. A SOC requires above-mentioned element just accomplish the testing, and an efficiency of testing depend on the test application time. The scan-based testing methodology needs high-test application time. Because scan requires test data to be shifted in and out by one or more scan chains. The recent approaches to minimize test application time include [7], [8], and [9].

Our idea is based on a reconfigurable core wrappers [1] and rectangle packing [2]. We improve the configuration of reconfigurable core wrappers in the hardware. In Our algorithm, we use the method of [2] to get the initial information.

This paper is organized as follows. In Section 2 we introduce the reconfigurable core wrappers and propose the improvement configuration. The general rectangle-packing problem introduce in Section 3. Simultaneously, in Section 3 we illustrate our idea and an algorithm. In Section 4 are the experimental results and Section 5 is conclusion.

II. THE RECONFIGURABLE CORE WRAPPERS

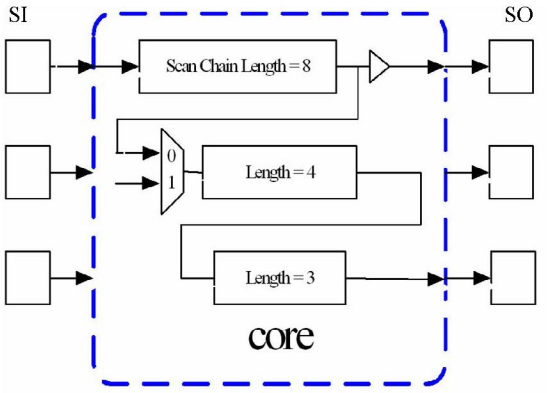
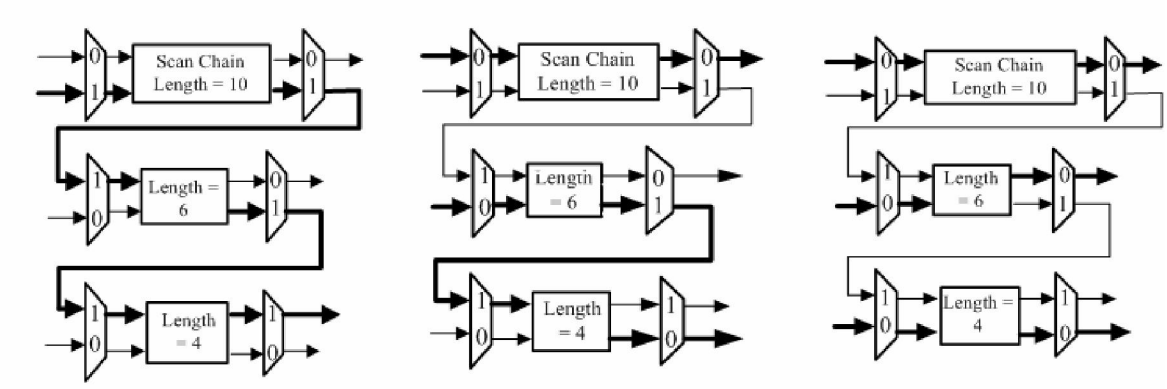


Figure 1. The Reconfigurable Core Wrappers Design

In [1] Koranne describes the design of reconfigurable core wrappers. These kinds of wrappers enable dynamic change width of the TAM to execute a core test. This manner is to put multiplexers on the head of certain scan chain. The connection of core internal scan chain will allow arbitrarily changing. Therefore, we can utilize different test resource (the different width of the TAM) to test the core. An example of scan chain configuration is shown in Figure 1. The core consists of three internal scan chains of length 8, 4, and 3, respectively. A multiplexer before the head of the L4 is controlled by the control signal. Then the control signal is selected as a zero, the TAM of 1 bit width which length equals 15. Then, the TAM of width 2 bits which length equals 8 and 7, respectively. The control signal is selected as a one.

Now, we propose an improvement configuration. We put the multiplexer before the head and after the tail of all scan chains. This method enables more easier to change the width of the TAM. Why we use this configuration? Because we want to utilize different TAM widths at the different time to execute a core testing. Therefore, the current time changes to follow the testing schedule, Then the TAM widths are going to change to follow the current time. So, the core can use dissimilar test resource to complete testing. An example is shown in Figure 2.

1-4244-0387-1/06/$20.00 ©2006 IEEE



(a) TAM Width = 1 (b) TAM Width = 2 (c) TAM Width = 3

Figure 2. To add multiplexer before the head and after the tail of all scan chains (a) TAM Width = 1, (b) TAM Width = 2, (c) TAM Width = 3

(a), (b), and (c). In Figure 2 (a), the basic configuration consists of three internal scan chains of length 10, 6, 4, respectively. All scan chains include two multiplexers that the one before the head other after the tail. Then all multiplexers that control signal are selected as a one, the TAM of 1bit width which length equals 20. In Figure 2 (b), the TAM of width 2 bits which length equals 10, 10, respectively. The multiplexer before the head and after the tail of L10 is selected as a zero. This scan chain is the first independent TAM. The L6 and L4 are connected with a second TAM. The control signal before the head of L6 is selected as a zero. The control signal after the last of L6 is selected as a one and the control signal before the head of L4 is selected as a one too. The control signal after the tail of L4 is selected as a zero. In Figure 2 (c), TAM widths are 3 bits. All multiplexers are selected as a zero.

III. THE RECTANGLE PACKING PROBLEM AND TEST SCHEDULING

To execute the testing schedule must find an initial rectangle of the core before. The rectangle of the core includes two parts. The one part represents the test resource (the number of TAM widths). Other one represents the testing time under the test resource. How to find the initial rectangle of a core? We need to compute the optimal test resource for the core, and to find the testing time under this optimal test resource. Then, we can execute the testing schedule.

In the past, testing schedule which the partition of TAM width is fixed. So, Testing schedule exist a lot of idle time.

An example is shown in Figure 3. There are three idle times in the testing schedule. More idle times cause testing schedule bed. Therefore, we propose a new testing schedule method. We call stairway scheduling. This method considers that the core cuts into many pieces and tests by different width of TAM. This paper we assume that all cores in the SOC can test at the same time. Then, we insert the core testing into the idle time. Therefore, the core must change the TAM width that to satisfy the idle time. Such the core testing can complete different TAM width. An example is shown in Figure 4. When the Core 4 completes the testing and the TAM width cannot schedule any core at the current time. The next scheduling of the core (Core 3) will use the TAM width of Core 4. Then the current time moves to the next time (the Core 1 end time) and the Core 3 testing use the TAM width (Core 4 + Core 1) at the current time. These actions complete until the pattern of Core 3 to finish shifting, and the Core 3 end it's testing.

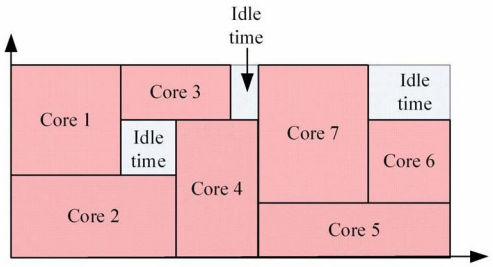


Figure 3. The SOC testing schedule (In the past method)

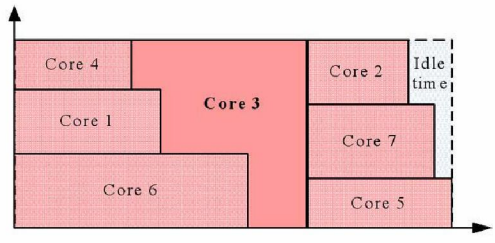


Figure 4. The SOC testing schedule (Our method)

In Figure 5, we detail the algorithm that we have developed to solve the problem of test scheduling. In our algorithm utilize the method [2] to find the initial rectangle of a core, and to assign the idle width of the TAM to the longest core at the

current time. We elaborate on each step of the algorithm in the following paragraphs.

Procedure *SOC_schedule_optimizer*

```
1 Compute collection of core rectangles using a design of wrapper;
2 Initialize(C, d, p);
3 Set w_avail = W_max; this_w = W_max; this_time = 0; next_time = 0;
4 While C ≠ ∅ {
5   If w_avail > 0 AND this_w = W_max {
6     If Core i ∈ C can be found, such that
        Width (i) ≤ w_avail AND T_i(width (i)) is maximum; {
      Schedule (i) = 1; }
7     Else Core i∈ C can be found AND C − {i } ≠ ∅ {
        In schedule structure, find a longest end_time (i); AND
        Assign remain w_avail to this core; }}
8   Else {
      Set this_time = next_time;
      In schedule structure, find the end_time (i) > this_time AND
        end_time (i) is minimum;
      Set next_time = end_time (i) ;
      Set w_avail = this_time width AND this_w = w_avail;
9     Calculate T_ic (width_c (i)) under current w_avail;
10    If T_ic (width_c (i)) > next_time {
11      While a Core pattern ≠ ∅ {
12        Calculate pattern numbers, such that
            T_ip (width_ip (i)) will across next_time
              under current w_avail;
          Records remain pattern numbers;
          Set this_time = next_time;
          Set w_avail = this_time width AND this_w = w_avail;
13        If this_time width = W_max {
            Set next_tim = ∞ ;}
14        Else {
            In schedule structure, find the end_time (i) > this_time
              AND end_time (i) is minimum;
            Set next_time = end_time (i) ; }}
          Schedule (i) = 1; }
15    Else {
        Set next_time = this_time + T_ic (width_c (i));
        Set w_avail = this_time width AND this_w = w_avail;
      Schedule (i) = 1; }}}
16  Return Schedule:
```

Figure 5. Algorithm for solve the problem of test scheduling

A. *The Structure of the core*

We use two structures, which the one is core_initial and the other is core_sedule to store the TAM width, the pattern numbers, and testing time values, respectively. The Structure for the core of the SOC is presented in Figure 6. This data structure update with the begin times and the end times for each core.

B. *The Initial rectangle of the core*

In Line 1, we use the method [2] to compute the collection of the initial rectangles. The Line 3 set initial values of w_avail, this_w, this_time, and next_time. (In this paper, Wmax is chosen to be 64.)

C. *A first movement of scheduling the core*

In Line 5 is a general action to schedule the core. The w_avail is not equal the Wmax and the TAM width of the initial rectangle satisfy this w_avail such we schedule this core.

D. *Assigning the idle TAM width to cores*

In Line 7, we use the method [2] to assign the remained TAM width to the longest core at the current time (this_time). An example is shown in Figure 7. Therefore, the TAM width (w_avail) equals the total (Wmax) width at the current time (this_time).

E. *Using the action of stairway schedule the core*

In Line8, we set the this_time to the next_time and release the TAM width at the current time (this_time). In Line 9, the testing time Tic (widthc (i)) of the core compute under current TAM width. In Line 10, Line 11, and Line 12, we will calculate whether the testing time Tip (widthip (i)) not across next_tme and compute the remnant pattern numbers. In Line 13, when the Width of the TAM equals the total TAM width (Wmax), we will set next_tme infinite. We schedule the core testing, when all patterns are finished to shift. In Line 15, the testing time Tip (widthip (i)) not across next_tme, we schedule this core testing.

Structure *core_initail*

```
1. num (i) /* the number of a core */
2. end_time (i) /* the initial time of a core */
3. width (i) /* the initial width of a core */
4. remain_p (i); /* the remain pattern numbers of a core */
5. Schedule (i)/* boolean indicates test for Core i has completed */
```

Structure *core_sedule*

```
1. num (i) /* the number of a core */
2. this_time (i) /* the scheduling begin time of a core */
3. end_time (i) /* the scheduling finish time of a core */
4.Schedule (i) /* boolean indicates test for Core i has completed */
```

Figure 6. The Structure of the core

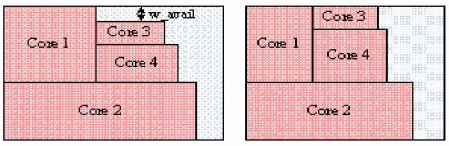


Figure 7. Assigning remain TAM width

F. *The idle time of the stairway scheduling*

An ideal result of the stairway scheduling will not have occurred the idle time. But Tip (widthip (i)) is not completely to match the next time. A little idle time will exist in the scheduling of the core testing. An example is shown in Figure 8 there are two idle times in the Core 3 testing. But idle times are small. An experimental result proves the times of idle will not affect the total time of testing.

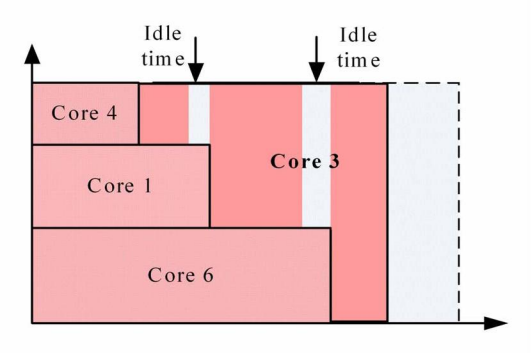


Figure 8. The idle time of the stairway scheduling

IV. EXPERIMENTAL RESULTS

In this section, we show the experimental result in Table1, Table2, Table3, Table4, respectively. The algorithms were implemented using C++ language. We used four SOCs from the new set of ITC'02 SOC Test Benchmarks [5]: d695, p22810, p34392, and p93791and compare our results with three previously published approaches: (1) the Test Bus Architecture optimization method base on ILP and exhaustive enumeration in [4], (2) the rectangle packing co-optimization in [2], (3) the wrapper/TAM co-optimization in [3]. We use the TAM width 16, 24, 36, 48, 56, 64, to test every benchmark. Experimental results show our method get the better results than others method.

TABLE I. EXPERIMENTAL RESULTS FOR SOC D695

W	LB	ILP[4]	Rect. pack.[2]	Par_eval[3]	Our method
16	41231	42568	44545	42644	43023
24	27487	28292	31569	30032	28878
32	20615	21566	23306	22268	22234
40	16492	17901	18837	18448	17848
48	13743	16975	16984	15300	15864
56	11780	13207	14974	12941	13722
64	10307	12941	11033	12941	12279

TABLE II. EXPERIMENTAL RESULTS FOR SOC P22810

W	LB	ILP[4]	Rect. pack.[2]	Par_eval[3]	Our method
16	412538	462210	489192	468011	499625
24	275025	361571	330016	313607	373043
32	206269	312569	312662	246332	242292
40	165015	278359	278360	232049	212241
48	137512	278359	268474	232049	178297
56	117868	268472	266800	153990	138824
64	103134	260638	260638	153990	129609

TABLE III. EXPERIMENTAL RESULTS FOR SOC P34392

W	LB	ILP[4]	Rect. pack.[2]	Par_eval[3]	Our method
16	936881	998733	1053491	1033210	1051402
24	624587	720858	759427	882182	715732
32	544579	591027	544579	663193	544579
40	544579	544579	544579	544579	544579
48	544579	544579	544579	544579	544579
56	544579	544579	544579	544579	544579
64	544579	544579	544579	544579	544579

TABLE IV. EXPERIMENTAL RESULTS FOR SOC P93791

W	LB	ILP[4]	Rect. pack.[2]	Par_eval[3]	Our method
16	1707095	1771720	1932331	1786200	1763635
24	1138063	1187990	1310841	1209420	1268411
32	853547	887751	988039	894342	871332
40	682838	698583	794027	741965	696894
48	569031	599373	669196	599373	603498
56	487741	514688	568436	514688	514141
64	426773	460328	517958	473997	464305

V. CONCLUSION

In this paper, we propose an idea of section a core to test. The core can split to complete a testing, but in the hardware requires adding extra cost. Therefore, an engineer must consider how to find a balance at test application time and hardware.

REFERENCES

[1] S. Koranne. "Design of Reconfigurable Core Wrappers for Embedded Core Based SOC Test". In Proc. of ISQED, pages 106— 111, March 2002

[2] V. Iyengar, K. Chakrabarty, and E. J. Marinissen. "On using rectangle packing for SOC wrappers/TAM co-optimization". In Proc. VLSI Test Symp., 2002, pp. 253—258.

[3] V. Iyengar, K. Chakrabarty, and E. J. Marinissen, "Efficient Wrapper/TAM Co-Optimization for Large SOCs". In Proc. Design, Automation, and Test in Europe (DATE), pages 491-498, March 2002.

[4] K. Chakrabarty, "Design of System-on-a-chip Test Access Architecture Using Integer Linear Programming". In Proceedings IEEE VLSI Test Symposium (VTS), 2001, pp 127-134.

[5] E.J. Marinissen, V. Iyengar and K. Chakrabarty, "A Set of Benchmarks for Modular Testing of SOCs ". In Proceedings IEEE International Test Conf. (ITC), Baltimore, MD, Oct. 2002.

[6] Yervant Zorian, Erik J. Marinissen, and Sujit Dey, "Testing Embedded-Core Based System Chips". In Proceedings IEEE International Test Conference, pp 130-134, 1998.

[7] Vikram Iyegnar, Krishnendu Chakrabarty, and Erik Jan Marinissen, "Test Wrapper and Test Access Mechanism Co-Optimization for System-on-Chip". In Proceedings IEEE International Test Conference, pp 1023-1032, 2001.

[8] Vikram Iyegnar, Krishnendu Chakrabarty, and Erik Jan Marinissen, "Test Access Mechanism Optimization, Test Scheduling, and Tester Data Volume Reduction for System-on-Chip". IEEE Transaction on Computers, 52(12), pp. 1619-1631, 2003.

[9] Chih-pin Su, and Cheng-wen Wu, "A Graph-Based Approach to Power-Constrained SOC Test Scheduling". Journal of Electronic Testing: Theory and Application 20, 45-60, 2004.