Optimal Test Access Mechanism (TAM) for Reducing Test Application Time of Core-Based SOCs

Jiann-Chyi Rau*, Po-Han Wu, Wnag-Tiao Huang, Chih-Lung Chien and Chien-Shiun Chen

Department of Electrical Engineering, Tamkang University, Tamsui, Taiwan 251, R.O.C.

Abstract

In this paper, we propose an algorithm based on a framework of reconfigurable multiple scan chains for system-on-chip to minimize test application time. The control signal combination causes the computing time increasing exponentially, and the algorithm we proposed introduces a heuristic control signal selecting method to solve this serious problem. We also minimize the test application time by using the balancing method to assign registers into multiple scan chains. The results show that it could significantly reduces both the test application time and the computation time.

Key Words: Test Access Mechanism (TAM), Test Application Time, Core-Based SOCs

1. Introduction

1.1 Test Challenge in SOC Designs

The more and more widening design productivity gap between VLSI system capabilities and design engineering capability, in a limited time to market scenario, has prompted many design houses to adopt a policy of design reuse at the core level [1]. The Semiconductor Industry Association's Technology Roadmap [2] predicts the percentage of reusable cores in SOC to be rising to 80% in 2006. However, with the increasing complexity and reduction of design cycle, the test application time of SOC is becoming a major bottleneck for time-to-market. It is more and more important for reusability of design to reduce the design time, but it is not enough when the verification and testing for reusable cores take up the most of design time.

1.2 TAM Architecture

There are three main kinds of test access architectures [3]: (a) Multiplexing Architecture; (b) Daisy-chain Architecture; (c) Distribution Architecture. Varma and Ahatia [4] proposed the Test Bus Architecture which combines the Multiplexing and Distribution Architectures. The modules connected to a common test bus are tested in an arbitrary but sequential order. (note: the order of test for each core may be different in practice). We call this schedule as "serial testing mode". We also show the parallel testing mode on the TestRail Architecture which presented by Marinissen et al. [5] is a combination of the Daisychain and Distribution Architectures. The advantage of the TestRail Architecture over the Test Bus Architecture is that it allows access to multiple or all wrappers simultaneously, which facilitates module-external testing.

1.3 Wrapper Architecture

A standardized, but scalable test wrapper is an integral part of the IEEE 1500 working group proposal [2]. Apart from these mandatory modes, a core test wrapper might have several optional modes, e.g., a detach mode to disconnect the core from its system chip environment and the test access mechanism, or a bypass mode for the Universal BIST Scheduler [6] and the TestRail [7] test access mechanism.

1.4 Test Wrapper Design

A scan test for a core consists of three phases: (1) scan in of the test patterns to the scan registers and ready for normal execution, (2) normal execution, and (3) capture and scan out of the responses by scan registers. We

*Corresponding author. E-mail: jcrau@ee.tku.edu.tw

define for each core *i* the number of test pattern *pi*. Let *si* be the length of the longest wrapper scan-in chain to fill all flip flops for a core *i*, and *so* is the time of the longest wrapper scan-out to scan out all flip flops.

We suppose that in each pattern exactly one time slot is used for the normal execution step; this means that the right input data has to be available at the core inputs at the moment of the normal execution step. This can be accomplished by adding scannable flip flops around the core [8]. The test time ti of core i becomes the sum of the scan-in time, the time for normal execution, and the scan-out time: $t_i = s_i \cdot p_i + p_i + s_o \cdot p_i$. In the scan test process it is common practice to use pipelining; when one pattern is scanned out, the next pattern is scanned in. This reduces the test time of a core to: $t_i = (1 + \max\{s_i, s_o\}) \cdot p_i + \min\{s_i, s_o\}$. When the term '+1' indicates that pipelining cannot be used for the scanning out the last pattern.

E. J. Marinissen presented a Rectangle Packing Model in [9]. The total TAM width was partitioned among a number of fixed-width test buses and each core was assigned to one of these TAMs. In Figure 1.1, each test of cores could be modeled as a rectangle by a fixed TAM width and the testing time. This is defined as a wrapper/TAM design problem in [10]. For different TAM widths, the same test could be modeled in different rectangles by width and the testing time. Therefore, the schedule problem would be treated as a 2D Bin-packing problem.

Test schedule is the schedule for testing a SOC. Basically, a test schedule for a SOC should maintain all processes on testing. It includes the order of the cores for testing, the width for each core, and most important is that the test schedule would show the total test time. The test schedule is based on what TAM architecture the testing process used. The total test time is one of the main factors to evaluate the testing cost. For designing a test schedule, there are three categories: (1) Serial Test Schedules, (2) Parallel Test Schedules, (3) Mixed test schedules. It was shown in [11] that for a given core, the testing time varies with TAM width as a "*staircase*" function.

1.5 Motivation

The use of cores shortens the design flow and test time for a new system and raises the competitive ability of new products through design reuse However, this design scenario in practical implementation is faced with unresolved issues: design methods for building single-chip systems, challenges in test and sign-off for these systems, and intellectual property licensing, protection, and liability. Designers need new tools and methodologies to help them to overcome these difficulties.

2. The Proposed TAM Architecture for Optimal Testing Scheduling

2.1 Optimal TestRail Scheduling Algorithm

Problem description: Given a set of cores *C* and total available SOC TAM width W_{max}, determine the TestRail Architecture *R* for an optimal test schedule of each core C_i ($1 \leqq i \leqq K$) under the constraint of W_{max}, such that the SOC total testing time *T* is minimum and no test resource conflict happen.

The total testing time *T* for TestRail Architecture is determined by the maximum of testing time of the individual TestRails. The algorithm OTR that we proposed in this paper focuses on the sequential testing of TestRail Architecture. In the case of sequential testing, all other cores connected to the same TestRail are set in bypass mode when a core is being tested.

We can simply determine the testing time T(*r*) for a TestRail *r* with width w by the help of the procedure TestTime(*r*, *w*). The procedure TestTime(*r*, *w*) uses a procedure Wrapper_design(*c*, *w*) which can calculate the test time for each core on the TestRail *r*, such that we only accumulate the individual test time to obtain the total test result for a TestRail *r*. The algorithm OTR that we proposed is composed of four main steps as shown in Figure 2.1.

2.2 Initial Solution

We use a factor W_{in} to set the size of the TAM widths in advance. Our initial solution would start with a spe-

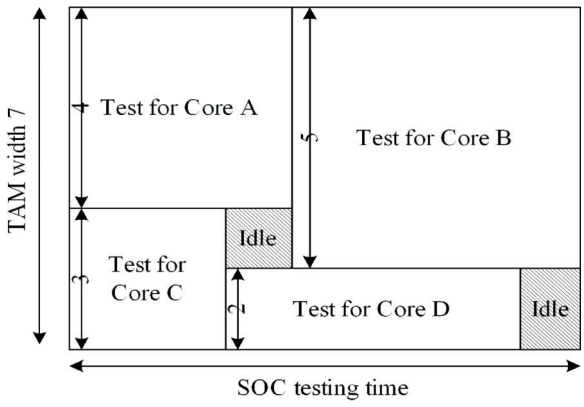

Figure 1.1. The example of test schedule.

Algorithm 1 [OTR]
1 Initial_Solution
2 Compress_Increase
3 Combine
4 Remove

Figure 2.1. The algorithm OTR.

cific number of TAMs in the first place, so that we can assign a particular arrange of the W_{in} value.

In step 1, we assign a parameter W_{in} to limit the maximum width. We select the largest core to be our candidate before we placed it into the TAM every time. 'The largest" means that the core has the maximum testing time on the constraint of width W_{in}. Every time when we place the core into a new TAM, the value of W_{in} will be updated to the width of this TAM if the remaining unused width is larger than the width of this TAM.

On the other word, if the remaining width is smaller than the width of this TAM, the value of W_{in} will be updated to the remaining width. The step will execute repeatedly until the remaining width is equal to zero. In Figure 2.2(a), after the $Core_1$ placed into the Core 1, the value of W_{in} would be updated to min(W_avail, P_1) and we find the large candidate core which would be placed in next time by the constraint of W_{in}. In Figure 2.2(b), we chose $Core_2$ to place the Core 2.

In step 2, if we still have unassigned cores left. We still select the largest core from the remaining cores, but the value of W_{in} will be updated to the width of the TAM which has the minimum testing time. Then we place it into the TAM which the testing time of TAM is minimum after the core was placed. We executed the step 2 iteratively until each core had got assigned.

2.3 Compress and Increase

We will try to optimize the total testing time of a given TestRail Architecture. It is an iterative procedure which consists of two steps. In step 1, we find the TAM which has the minimum testing time and compress its width to increase its testing time until the TAM does not have the minimum testing time any more.

In step 2, we take these widths to add the width of TAM which has the maximum testing time to decrease its testing time until it is not the maximum TAM any more or it already have get no benefit for reduction of total testing time. In Figure 2.3, the widths which are free up from TAM_4 would be used for adding the width of TAM_1 to decrease the overall testing times. The procedure ends if the TAM with the shortest testing time can not decrease its widths.

2.4 Combine

The procedure iteratively tries to combine two TAMs to free up the TAM width, such that the width can be used for the reduction of the total testing time. In step 1, the procedure tries to free up the TAM width by combining

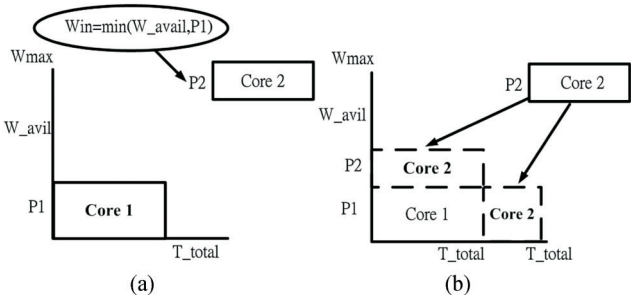

Figure 2.2. An example for Initial_Solution.

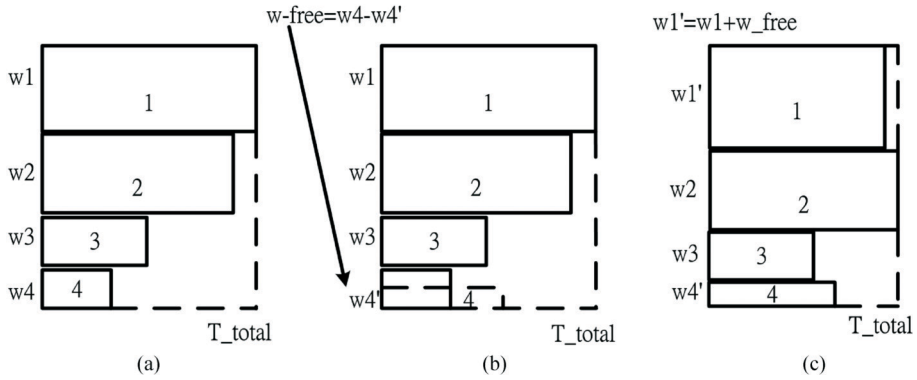

Figure 2.3. An example for procedure Compress_Increase.

two non-bottleneck TAMs. The two TAMs would be merged are determined to which the time is the shortest after they merged each other. The width of new merged TAM is which width is larger between them. In step 2, the freed up widths are distributed over all TAMs. We add the width of TAM with the longest testing time until it is not the longest one or it can not decrease the testing time any more. The procedure ends if all TAMs have been merged into one single TAM, or when no TAM can be found such that the testing time does not exceed the current overall testing time. For Figure 2.4, if we can merge TAM 3 into TAM 2, we will get the freed-up TAM width w3.

2.5 Remove

Moving the core from the TAM with the longest testing time to another TAM to reduce the total testing time. We select the core with the smallest testing time in the TAM with the longest testing time, and try to move the core to the other TAM without exceeding the overall testing time. If there are over than one TAM which the module can be moved in, we select the TAM which the testing time is minimum after the core was moved in. We don't directly move the core into the TAM with the shortest time because we also think about the idle time for our test schedule result. We would not only try to minimize the total testing time but also reduce the idle time in our result. In Figure 2.5, the core 4 with the shortest time which is removed to another TAM to reduce the total testing time. The procedure is repeated until the TAM with the longest testing time contains only one module, or the module with the shortest testing time can not be moved in any TAM to get the reduction of total testing time.

2.6 Experimental Results

We used four SOCs from the new set of ITC'02 SOC Test Benchmarks [12]. The number in each SOC name is a reflection of its test complexity. The first letter *d* means that the SOC comes from Duke University and *p* means industrial Philips Company. Table 1 present results of testing time in clock cycles achieved by the four optimization methods for a range of W values from 16 to 64. We considered our possible values of the parameters B in the range of $3 \leqq B \leqq 5$. The value of B means that the number of TAMs in our initial solution at least so that we can decide the value of W_{in} in the range of

$$\left\lfloor W_{\max} / B_{\max} \right\rfloor \leq Win \leq \left\lceil W_{\max} / B_{\min} \right\rceil \tag{1}$$

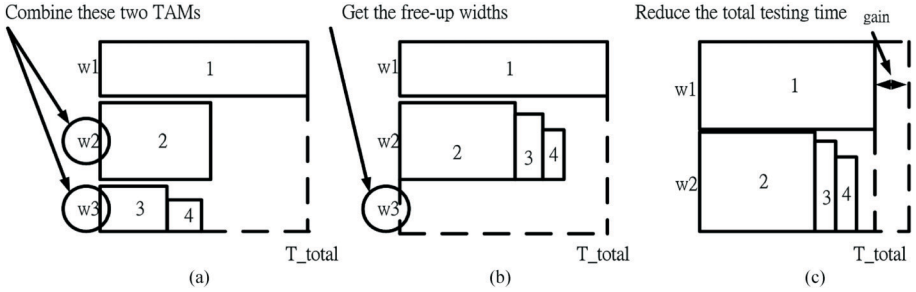

Figure 2.4. The operation of procedure combine.

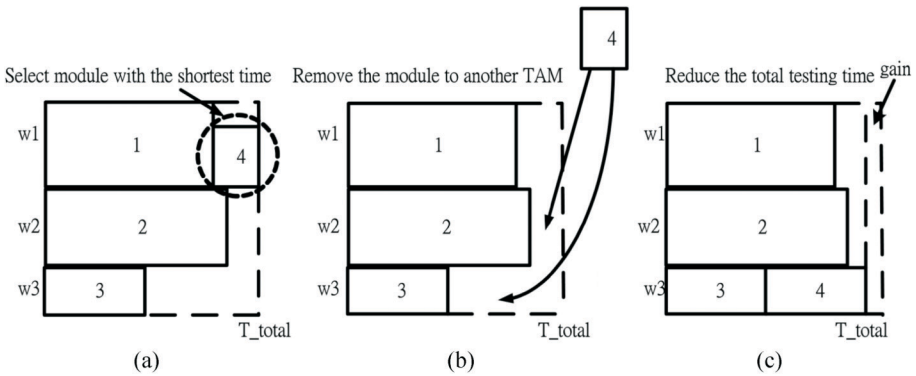

Figure 2.5. The operation of procedure remove.

Table 1. Experimental result for SOC benchmarks

SOC	Wmax	ILP. [13]	GRP [14]	Cluster [15]	OTR (Our)
d695	16	42,568	44,545	44,330	42,694
	24	28,292	31,569	30,021	30,018
	32	21,566	23,306	23,488	22,357
	40	17,901	18,837	19,034	17,681
	48	16,975	16,984	16,194	16,145
	56	13,207	14,974	13,479	12,941
	64	12,941	11,984	11,033	11,035
p22810	16	462,210	489,192	--	443,813
	24	361,571	330,016	--	303,064
	32	312,569	245,718	259,975	232,049
	40	278,359	199,558	206,205	194,193
	48	268,472	173,705	173,705	164,680
	56	266,800	157,159	146,390	145,417
	64	260,638	142,342	133,587	133,587
p34392	16	998,733	1,053,491	--	1,033,214
	24	720,858	759,427	876,529	698,657
	32	591,027	544,579	585,309	584,524
	40	544,579	544,579	544,579	544,579
	48	544,579	544,579	544,579	544,579
	56	544,579	544,579	544,579	544,579
	64	544,579	544,579	544,579	544,579
p93791	16	1,771,720	1,932,331	--	1,789,381
	24	1,187,990	1,310,841	--	1,190,788
	32	887,751	988,039	947,111	914,865
	40	698,583	794,027	816,972	732,407
	48	599,373	669,196	677,707	608,612
	56	514,688	568,436	542,445	527,127
	64	460,328	517,958	467,680	467,091

and tabulated our best results. Although our testing time for p93791 are higher than the testing times obtained using the method in [13], but we only need the less computing time to calculate the result.

3. Reconfigurable Multiple Scan-Chains for Reducing Test Application Time of SOCs

3.1 The Outline of Our Method

In this section, we propose another novel method to solve SOC scheduling problem. This method based on the Reconfigurable Multiple Scan Chains (RMSC) architecture [16]. Basically, our method could be divided into two parts, Control Signal Selection and Registers Assignment, in Figure 3.1 shows the basic flow of our method.

3.2 Module of Reconfigurable Multiple Scan Chain

The cores are prepared with internal scan chains and test vectors for each core. Reconfigurable Multiple Scan Chains (RMSC) [16] is one kind of architectures to construct the scan chains for TAM. Our algorithm is based on this architecture. We determine the order of registers in the scan-chains by the types of registers in the Registers Assignment. The definations are as fellowing:

n: the number of cores in the SOC

C: denotes the set of cores. $C = (C_1, C_2, \ldots, C_n)$. Notice that the cores were ordered in terms of strictly increasing test lengths. In Parallel Test Schedule, both two cores have the same start_time and end_time, we treat the two cores into a big single core for processing.

L: denotes the set of test lengths of set *C*. $L = (L_1, L_2, \ldots, L_n)$. The order is the same as set *C*. In overlapped test application scheme, the test for a SOC consists of a se-

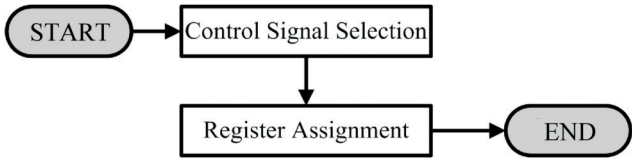

Figure 3.1. The basic flow.

quence of test sessions. Based on definitions, if there are n cores in core set C, there are n test sessions. The SOC test schedule could be (TS_1, TS_2, …, TS_n).

The example in Figure 3.2 is the modified RMSC architecture. The test sessions in Figure 3.2(a) could be showed as Figure 3.2(b). Two cores are two test sessions. The number of patterns for TS_i is $L_i - L_{i\text{-}1}$. In the sequence of test sessions, we could know the processes of test clearly by the name of test sessions.

Chain Cycles (CC_i) denotes the chain cycles under the test session TS_i which is the minimum number of clocks required to shift in or shift out the test data. The scan chains are reconfigurable, each CC_i for TS_i may not be the same.

In Figure 3.3(a) shows two scan chains of Figure 3.2(a) under test session 1. For a test vector, the bottleneck is the shift-in cycles for Scan Chain 1. At the end of TS_1, the control signal $Ctrl_1$ is activated. All the scan chains are reconfigured with $Ctrl_1$. This operation is shown in Figure 3.3(b). In Figure 3.3(c), there shows the scan chains with the activated $Ctrl_1$. All registers of Core A are bypassed CC_2 for TS_2 is 7.

In other words, the control signals and the MUXs would bypass the cores. The timing for activating the control signals should be defined as following:

(1) $Ctrl_i$ is activated at the end of TS_i only.

(2) Once a control signal is activated, it remains active until the last test session. The cores bypassed by control signals means no need to feed with patterns.

(3) Once $Ctrl_i$ is activated, it is possible to bypass the register in core C_1, C_2,..., C_i. If the number of control signals is a constraint, this definition is necessary for the test schedule, which would be explain later (In 4.3). Based on the information of CC and set L, we can estimate the total test cycle of the test by the following equation:

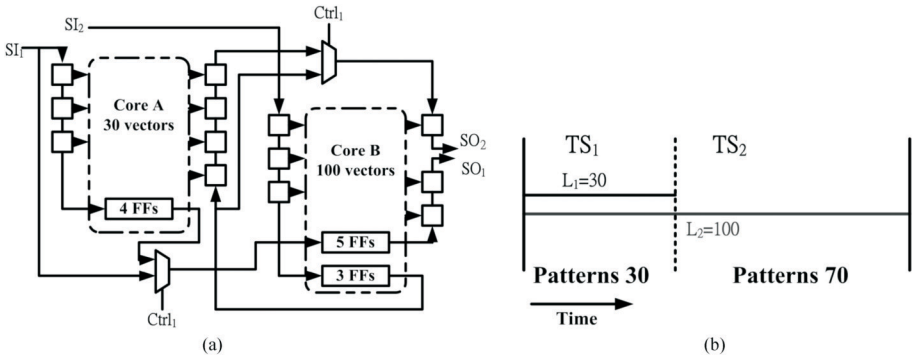

Figure 3.2. Example of (a) reconfigurable multiple scan chain design, (b) test sessions.

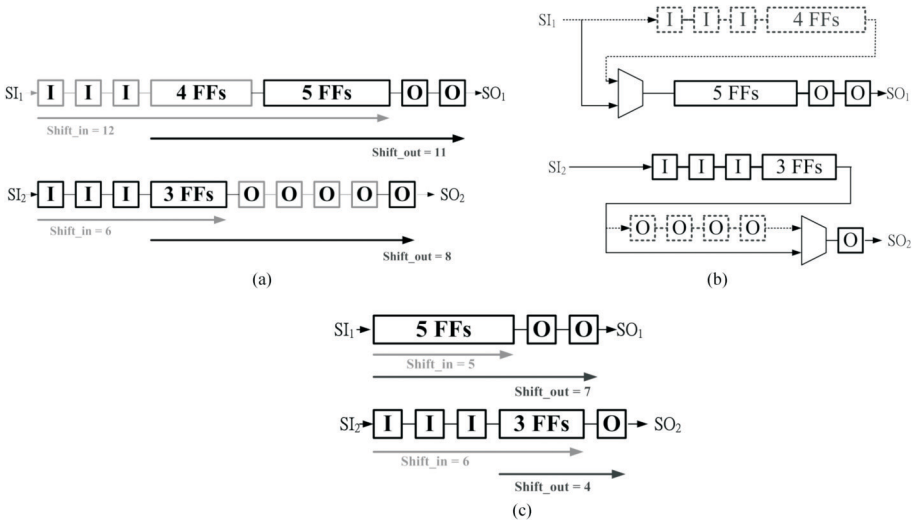

Figure 3.3. Example of (a) Scan Chains for test session 1, (b) the registers bypassed by $Ctrl_1$, (c) Scan Chain for test session 2.

$$\tau = [\sum_{i=1}^{n} (L_i - L_{i-1}) * (CC_i + 1)] + CC_1, \text{ where } L_0 = 0 \quad (2)$$

τ denotes the total test time(clocks). The summation is from the test session 1 to the test session n. The $L_i - L_{i\text{-}1}$ part denotes the number of test patterns for each test session. The $CC_i + 1$ part denotes the Chain Cycles for each test session and one capture cycle. Finally, CC_1 is the initial shifting cycles.

Since the addition of 1 to CC_i in each term of the summation merely adds a constant term equal to L_n and the last term, CC_1, is typically negligible compared to other terms, the objective function we try to minimize is given by the following expression:

$$\delta = \sum_{i=1}^{n} (L_i - L_{i-1}) * CC_i, \text{ where } L_0 = 0 \quad (3)$$

For the minimizing test time, there should be $n - 1$ control signals for n test sessions. If there may not have control signals activated at the end of some test sessions TS_i, then CC_i is the same with previous test session $TS_{i\text{-}1}$, as the denotation $CC_{i\text{-}1}$.

3.3 Algorithm of Control Signal Selection

We use the Control Signal Selected Table (CSST) for recording which control signals. For n test sessions, there need $n - 1$ control signals only. So the *CSST* should be $CSST = (CS_1, CS_2, ..., CS_{n\text{-}1})$. Before the Selection process, all CS_n would be initialized as 0. 0 denote not selected.

At first, we initialize the $CSST = (0, 0, 0)$. Then we build the $1 \times n$ matrix of TSP, n means the number of cores. Each element in TSP represents the number of test patterns for each test session and is computed by $L_i - L_{i\text{-}1}$, where $L_0 = 0$. For example, if $L = (15, 35, 53)$, we build TSP = [15 20 8 10].

Next for Line 03, we build another $n \times 1$ matrix of CSC. Each element represents the minimum shift cycle for the single core with the TAM width w. Notice that the TAM width w is the same as the total TAM width. For above example, we could build $CSC = [15\ 20\ 8\ 10]^T$. Line 04, the Matrix M is an $n \times n$ matrix multiplied by CSC and TSP. In the matrix M, each element means the cycles for the test session. Then we choosing $t + p$ control signals. t is the constraint of number of control signals and p is the parameter for increasing the accuracy. The array S is selecting control signals, which element represents the reducing cycles as the corresponding control signal is chose, as shown in Figure 3.5. The array $S = (S_1, S_2, ..., S_{n\text{-}1})$ is calculated by the following equation:

$$S_i = \sum_{j=1}^{i} \sum_{k=i+1}^{n} M_{(j,k)} \quad (4)$$

For the above example, $S = (456, 360, 360)$. Then we find the maximum number $S_m = 456$ which $m = 1$. So we set $CSST_1 = 1$ as choosing the $Ctrl_1$. After every control signal is choosing, the matrix M should be updated. The cycles would be changed corresponding the control signals. So a control signal is chose should be responded on matrix M^1 in Figure 3.6 (a). Although we selected the $Ctrl_1$ by S_1, we set the part of summation in S_1 to 0 as the updating.

After updating the matrix M, the loop would select next control signal until reaching $t+p$. In Figure 3.6 (b), $S = (0, 144, 240)$, so we set $CSST_3 = 1$, then control signal selection is completed.

The selected t control signals may be not the best solu-

Algorithm of Control Signal Selection

```
01 Initialize CSST;
02 Build matrix TSP;
03 Build matrix CSC;
04 Build matrix M by TSP x CSC;
05 for (i = 0; i < (t + p); i++){
06     Compute array S;
07     Find the maximum S_m in S, set CSST_m = 1;
08     Update M;
09 }
```

Figure 3.4. Algorithm of control signal selection.

$$M = \begin{array}{c|cccc} & TS_1 & TS_2 & TS_3 & TS_4 \\ Core_1 & 180 & 240 & 96 & 120 \\ Core_2 & 120 & 160 & 64 & 80 \\ Core_3 & 240 & 320 & 128 & 160 \\ Core_4 & 300 & 400 & 160 & 200 \end{array} \quad \begin{array}{l} \rightarrow S_1 \\ \rightarrow S_2 \end{array}$$

Figure 3.5. S1 and S2 on matrix M.

$$M^1 = \begin{array}{c|cccc} & TS_1 & TS_2 & TS_3 & TS_4 \\ Core_1 & 180 & 0 & 0 & 0 \\ Core_2 & 120 & 160 & 64 & 80 \\ Core_3 & 240 & 320 & 128 & 160 \\ Core_4 & 300 & 400 & 160 & 200 \end{array} \quad \text{(a)}$$

$$M^2 = \begin{array}{c|cccc} & TS_1 & TS_2 & TS_3 & TS_4 \\ Core_1 & 180 & 0 & 0 & 0 \\ Core_2 & 120 & 160 & 64 & 0 \\ Core_3 & 240 & 320 & 128 & 0 \\ Core_4 & 300 & 400 & 160 & 200 \end{array} \quad \text{(b)}$$

Figure 3.6. (a) Updating matrix M^1 (b) Next updating mxtrix M^2.

tion. So we set a parameter p to increase the accuracy. Although the constraint number of control signals is t, we use the control signal selection process to select p additional control signals. Then we take the $t + p$ control signals as a new control single space. Based on the new space, we would try fewer choices to reduce the computing time.

3.4 The Order for Registers Assignment

We could classify wrappers into three categories. (1) *Pure drivers*: wrappers for primary input terminals (PI). During testing mode are feeding test patterns to CUT only. (2) *Pure receivers*: wrappers for primary output terminals (PO). During testing mode are receiving test response from CUT only. (3) *Driver-receivers*: wrappers for primary bi-direction terminals. During testing mode are able to feed and receive test data (test patterns and response).

Figure 3.7 shows that the order of registers would influence the chain cycles. *O* denotes the wrapper for PO. *I* denote the wrapper for PI. *B* denotes the wrapper for the bi-direction terminal. In Figure 3.7(a), if we arrange the registers in the order from input such as: *pure drivers*, *driver-receivers*, and *pure receivers*. Then the chain cycles could be minimized in Figure 3.7 (b).

As the figure shown in Figure 3.8, SI denotes the registers for shift in. SO denotes the registers for shift out. Figure 3.8 shows the testing of three test patterns. For reducing the testing time, the lengths of the scan chains should be as balanced as possible. During the test process, the registers of the cores in the later test sessions would be bypassed later. In other words, the registers would be existed in the scan chain longer. So we assign the registers of the later test sessions at first. By earlier

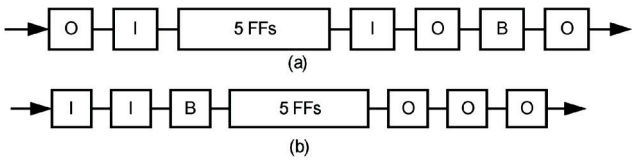

Figure 3.7. Examples for the order of register in single scan-chain (a) Random, (b) Specified.

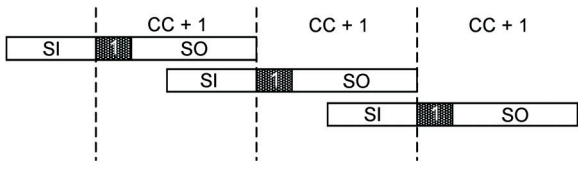

Figure 3.8. Advantage of the specified order.

assigned, the scan chains could be easier to be balanced. Based on the reasons above, the order of registers for Registers Assignment is based on the term of decreasing n of test sessions. The test sessions order set for registers assignment is $S = (TS_n, TS_{n-1},..., TS_1)$.

3.5 The Definitions of Blocks

There may be not enough control signals for all test sessions. So t control signals would divide test sessions into $t + 1$ block. In Figure 3.9, two control signals divide test sessions into three blocks. For the test sessions of the same block, the registers assignment should be considered together since they are facing the same scan chains. So we can treat the cores in the same block as a single big core and assign the registers to the minimum shift cycles.

3.6 Algorithm of Registers Assignment

First we obtain all combinations of the new control signal space as choices. Then TS are ordered in terms of decreasing n as the order for the registers assignment. Lines 06 to Line 08 are assigning the registers to TAM for the minimum shift cycles. Line 09 and Line 10 are computing the test cycles, the test time calculation is based on blocks. After Line 08 the scan-chains was setup for all test sessions in the same block. In other words, the *CCs* for each test session in the same block are the same. We could just sum the number of test patterns of each test session in the block and multiply by *CC* as the test cycles of the block. Although the scan chains would be assigned registers by next block, so the total test time is the summation of the blocks one by one.

After Line 11, the TAM would be compared and replaced if the choice is the batter solution. BestAns contents the best choice and test cycles that the loop had done before. After every choice had been tried, it would be compared with the BestAns by the total test cycles. Then the loop would continue for next choice until all choices had been tried. At last, the choice and the total test cycles would be the solution and be returned.

3.7 Experimental Results

We show experimental results of our proposed met-

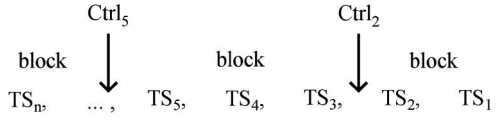

Figure 3.9. Example of blocks.

hod. Even though these SOCs originally contained multiple levels of design hierarchy, we have assumed in our experiments that all cores have the same hierarchy level.

In Table 2, the number after the SOC names represents the number of cores each SOC included. *W* denotes the TAM width and PINs denotes the total pins for the test scheduling comparing to *W*. The value *t* is the number of control signals that is used and cycles represents the test application time for the choice of *t* and *SCs*. In Table 2, our method has better test time under the less PINs. With the less TAM width, our method could save the control signals.

4. Conclusion

In this paper, we have proposed two different methods to solve SOC testing schedule problem. The OTR algorithm optimizes TestRail Architectures with respect to required ATE vector memory depth and test application time. Our algorithm also takes both wrapper and TAM design into account and require a negligible amount of computing time. The testing time results of The OTR algorithm are comparable or better than those previously published.

Another one, we have proposed an effective and efficient algorithm based on based on the framework of Reconfigurable Multiple Scan Chains to solve core-based SOC schedule problem. In this algorithm, the computing time is decreased by the Control Signal Selection. The

Algoruthm of Registers Assignment

```
01 Obtain all choices form new control signals space;
02 Order TS in the terms of decreasing n;
03 for every choice {
04     divide the TS into t+1 blocks by the choice;
05     for every block (in the order above) {
06         sort the internal scan chains of the cores in the
               block in decreasing order;
07         assign the inter scan chains to TAM in the order above;
08         assign Bidirs, Inputs and Outputs to TAM;
09         cycles = calculate the test cycles of the block;
10         TotalCycles = TotalCycles + cycles;
11     }
12     if (TotalCycle < BestAns. cycles
13         copy and replace current TAM content, TotalCycles and
               choice to BestAns;
14     Clear TAM, TotalCycles;
15 }
16 Return BestAns;
```

Figure 3.10. Algorithm of registers assignment.

Table 2. Comparison with other works

SOC	W	PINs	ILP [13]	GRP [8]	Cluster [17]	TR [18]	Proposed		
							t	SCs	cycles
d695 (10)	16	42	42644	43713	44330	44307	1	20	44689
							6	18	36122
							8	16	41528
	32	74	22268	23021	23488	21518	1	36	26548
							6	34	24697
							8	32	27767
p22810 (29)	16	61	468011	452639	(N/A)	458068	5	28	606795
							11	25	325837
							27	16	456963
	32	93	246322	246150	259975	222471	5	44	542203
							9	42	244989
							11	41	251277
							27	32	343044
p34392 (20)	16	52	1033210	1023820	(N/A)	1010821	1	25	908814
							4	24	841720
							17	16	1075617
	32	84	591027	544579	585309	551778	6	39	646062
							12	36	616186
							17	32	698426
p93791 (33)	16	65	1786200	1851135	(N/A)	1791638	3	31	969757
							5	30	962566
							12	26	1079224
							24	16	1711254
	32	97	894342	975016	(N/A)	912233	5	46	606060
							7	45	658576
							24	32	1121699

Registers Assignment is simplified by the blocks divided by the control signals and a specified registers order. The algorithm is performed well for the SOC with a larger number of cores embedded and tested by few pins.

References

[1] Keating, M. and Bricaud, P., *Reuse Methodology Manual for System-on-Chip Designs*, Kluwer Academic Publishers (1998).

[2] IEEE P1500 Web Site, http://grouper.ieee.org/groups/1500/.

[3] Aerts, J. and Marinissen, E. J., "Scan Chain Design for Test Time Reduction in Core-Based ICs," In *Proceedings IEEE International Test Conference*, pp. 448–457 (1998).

[4] Varma, P. and Bhatia, S., "A Structured Test Re-Use Methodology for Core-Based System Chips," In *Proceedings IEEE International Test Conference*, pp. 294–302, Washington, DC (1998).

[5] Marinissen, E. J., Arendsen, R., Bos, G., Dingemanse, H., Lousberg, M. and Wouters, C., "A Structured and Scalable Mechanism for Test Access to Embedded Reusable Cores," In *Proceedings IEEE International Test Conference*, pp. 284–293 (1998).

[6] Zorian,Y., "A Distributed BIST Control Scheme for Complex VLSI Devices," *In Proceedings IEEE VLSI Test Symposium*, pp. 6–11 (1993).

[7] Bleeker, H., van den Dijnden, P. and de Jong, F., "BoundaryScan Test-A Practical Approach. Kluwer Academic Publishers," Dordrecht, Netherlands (1993).

[8] Iyengar, V., Chakrabarty, K. and Marinissen, E. J., "On Using Rectangle Packing for SOC Wrapper/TAM Co-Optimization," In *Proceedings IEEE VLSI Test Symposium*, pp. 253–258 (2002).

[9] Iyengar, V., Chakrabarty, K. and Marinissen, E. J., "Test Wrapper and Test Access Mechanism Co-Optimization for System-on-Chip," In *Proceedings IEEE Internal Test Conference*, pp. 1023–1032 (2002).

[10] Iyegnar, V., Chakrabarty, K. and Marinissen, E. J., "Test Access Mechanism Optimization, Test Scheduling, and Tester Data Volume Reduction for System-on-Chip," In *Proceedings IEEE Transaction on Computers*, pp. 1619–1631 (2003).

[11] Goel, S. K. and Marinissen, E. J., "Effective and Efficient Test Architecture Design for SOCs," In *Proceedings IEEE International Test Conference*, pp. 529–538 (2002).

[12] Larrson, E. and Peng, Z., "An Integrated Framework for the Design and Optimization of SOC Test Solution," In *Proceedings Date, Automation and Test in Europe Conference and Exhibition 2001*, pp. 138–144 (2001).

[13] Sehgal, A. and Chakrabarty, K., "Efficient Modular Testing of SOCs Using Dual-Speed TAM Architectures," In *Proceedings Date, Automation and Test in Europe Conference and Exhibition*, pp. 422–427 (2004).

[14] Xu, Q. and Nicolici, N., "Multi-Frequency Test Access Mechanism Design for Modular SOC Testing," In *Proceedings 13th Asian Test Symposium*, pp. 2–7 (2004).

[15] Chakrabarty, K., Iyengar, V. and Krasniewski, M. D., "Test Planning for Modular Testing of Hierarchical SOCs," In *Proceedings IEEE Transactions on Computer-Aided Design of integrated Circuits and Systems*, pp. 435–448 (2005).

[16] Chakrabarty, K., Iyengar, V., Krasniewski, M. D. and Kumar, G. N., "Design and Optimization of Multi-Level TAM Architectures for Hierarchical SOCs," In *Proceedings 21st VLSI Test Symposium*, pp. 299–304 (2003).

[17] Goel, S. K. and Marinissen, E. J., "Cluster-Based Test Architecture Design for System-on-Chip," In *Proceedings IEEE VLSI Test Symposium (VTS)*, pp. 259–264 (2002).

[18] Goel, S. K. and Marinissen, E. J., "A Test Time Reduction Algorithm for Test Architecture Design for Core-Based System Chips," *Journal of Electronic Testing: Theory and Applications*, pp. 425–435 (2003).

Manuscript Received: Sep. 15, 2008
Accepted: Jun. 6, 2009