The Efficient TAM Design for Core-Based SOCs Testing

Jiann-Chyi Rau, Po-Han Wu, Chih-Lung Chien, Chien-Hsu Wu
Department of Electrical Engineering, Tamkang University
151, Ying-Chuan Rd. Tamsui,
Taipei Country, Taiwan 25137, R.O.C.
{ jcrau, phwu, clchien, cswu}@ee.tku.edu.tw

Abstract: - This paper presents a framework and an efficient method to determine SOC test schedules. We increase the test TAM widths by the framework. Our method deals with the traditional scan chains and reconfigurable multiple scan chains. Experimental results for ITC'02 SOC TEST Benchmarks show that we obtain better test application time when compared to previously published algorithms. Test access mechanism (TAM) and test schedule for System-On-chip (SOC) are challenging problems. Test schedule must be effective to minimize testing time, under the constraint of test resources. This paper presents a core section method based on generalized 2-D rectangle packing. A core cuts into many pieces and utilizes 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.

Key-Words: - SOC Testing, TAM, Testing Scheduling

1 Introduction

The heading of each section should be printed in small, 14pt, left justified, bold, Times New Roman. You must use numbers 1, 2, 3, … for the sections' numbering and not Latin numbering (I, II, III, …)
Although the SOC design also called core-based-design is still a developing technology, but a lot of researches has proposed many idea to solve the problem in the future that will occur, especially how to test the SOC chip with efficiency method. Base on this reason, we have made a research to solve it. Now, I will first introduce the concept about SOC design below.

1.1 Test challenges in Core-based 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 [3]. The Semiconductor Industry Association's Technology Roadmap [1] 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.

One of the most effective and popular techniques is Boundary Scan Test, defined as IEEE standard 1149.1 ('JTAG'). The Boundary Scan Test standard defines the additional hardware for the IC, in order to solve the board interconnect test problem. Unfortunately, the boundary scan architecture allows only one pin for test data input and another one for data output, and hence can not efficiently support multiple scan chains of the core on SOC. So for testing a SOC, JTAG may not suitable.

There still exists an important challenge. For SOC integrators, the information inside the cores may be invisible. Most of cores are encrypted Intellectual Property blocks or hard cores that the users don't have enough information about the internal part of the cores. So it is difficult and inefficient for core user to develop the test set. Core vendors have responsibility to develop the core test. The most common method is Design-For-Testability (DFT). Core vendors would offer information to SOC integrators to design test processes.

The information is such as the I/O pins, internal scan chains, test patterns, power constraints, clock timing and etc. In order to increase the interoperability of the internal test with the core, the internal test to accompany its corresponding core need to be described in accepted standard format. Such a standard format is currently being developed by IEEE P1500 [4][5] and referred to as standardization of a core test description language.

1.2 TAM architecture

The basic TAM architecture is shown in Figure 1. It can be divided into three structural elements [6]:

Source and Sink: the source generates the test pattern for the embedded cores and sink compares the obtained responses with the excepted one to know if there exists difference. The two elements can generate two test methods on-chip and off-chip by its different position. With the method on-chip, like BIST or built-in ATE, all of that have the trouble with increasing area, but in opposite way, the requirement of TAM can be loosen. With the method off-chip, the total chip pins will decide the bandwidth under testing. If we increase the TAM width, it will raise the expensive cost on hardware.

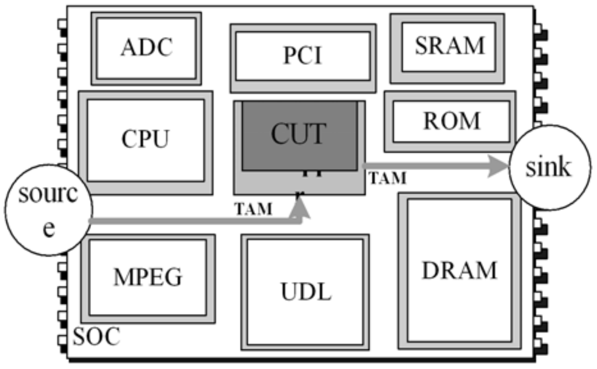

Fig. 1 The test access architecture overview

Test access mechanism (TAM): TAM is the mechanism of transporting the test data. TAM can transport the test set to the CUT and transport the response from CUT to the sink. With TAM design, there must be a trade-off between test bandwidth (capacity) and test cost. The larger TAM width can provides more bandwidth but it also increases the cost on routing. If the test source designs outside the SOC and the IC only with few pins, there is unmeaning to increase the TAM width.

Core test wrapper: wrapper is the interface between the embedded core and SOC, it connects between the core terminals and TAM and provides the switching mechanism to the function I/O and TAM.

1.2.1. Basic TAM architecture

There are three main kinds of test access architectures [7]: (a)Multiplexing Architecture; (b)Daisy-chain Architecture; (c)Distribution Architecture.

1.2.2. Test Bus architecture

Varma and Ahatia [9] proposed the Test Bus Architecture which combines the Multiplexing and Distribution Architectures. A single test bus actives the same as the operation of Multiplexing Architecture. Modules which connected to the same test bus can be only tested sequentially. The Test Bus Architecture allows that multiple test buses exist on one SOC and operate independently just like the Distribution Architecture. So modules connected to a same test bus will be in the common detrimental conditions to make the core-external testing difficult as in the Multiplexing Architecture.

1.2.3. TestRail architecture

The TestRail Architecture presented by Marinissen et al. [10] is a combination of the Daisychain and Distribution Architectures. A single TestRail is in essence the same as what is described by the Daisychain Architecture. Modules connected to the same TestRail can be tested simultaneously, as well as sequentially. The TestRail Architecture allows for multiple TestRails on same SOC, which operate independently, as in the Distribution Architecture. 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.

TestRail Architecture supports multiple types of test schedules. It allows the modules on the common TAM operating in both serial testing mode and parallel testing mode. In parallel testing mode, the modules on TAM will be tested simultaneously.

1.3 Wrapper architecture

Good wrapper design will make the internal scan chains of core as balance as possible to reduce the core testing time. A standardized, but scalable test wrapper is an integral part of the IEEE P1500 working group proposal [11]. 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 [12] and the TestRail [13] test access mechanism.

Wrappers may need to perform test width adaptation when the TAM width is not equal to the number of core terminals. This will often be required in practice, since large cores typically have hundreds of core terminals, while the total TAM width is limited by the number of SOC pins.

1.4 Reconfigurable Core Wrappers Design

For easy testing, the providers usually use necessary Design-For-Testability (DFT) skills to the SOC chips. The integrators were allowed to insert a wrapper cell to each input and output by these DFT skills. All the internal scan chains, input ports and output ports would be assigned into new scan chains. Test vectors will be recombined with the new scan chains. Reconfigurable Multiple Scan Chains is one kind of architecture to reassign the scan chains.

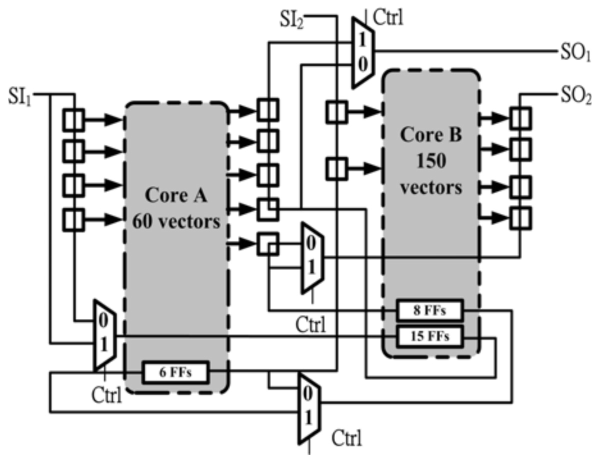

Fig. 2 An example of Reconfigurable Multiple Scan

2 Problem Formulation

2.1 Core test time

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 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 assume 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 [14]. 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:

$$t_i = s_i \cdot p_i + p_i + s_o \cdot p_i \quad \text{(Eq. 1)}$$

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:

$$t_i = (1 + \max\{s_i, s_o\}) \cdot p_i + \min\{s_i, s_o\} \quad \text{(Eq. 2)}$$

When the term ‘+1’ indicates that pipelining cannot be used for the scanning out the last pattern. The test time for a core decreases as both si and so are reduced. Therefore, the balance wrapper scan chains are important because the number of cycles to scan a test pattern to (from) a core is a function of the length of the longest wrapper scan-in (scan-out) chain.

2.2 The Popular Rectangle Packing Model

E.J. Marinissen presented a Rectangle Packing Model [15] as fixed width test buses. 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 Fig .5, 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 [16]. For different TAM widths, the same test could be modeled in different rectangles by width and the testing time. The rectangles in Fig. 3 are using the Design_wrapper algorithm [16] to model the Core 6 in SOC p93791. So based on the model, a test schedule problem would be treated as a 2D Bin-packing problem.

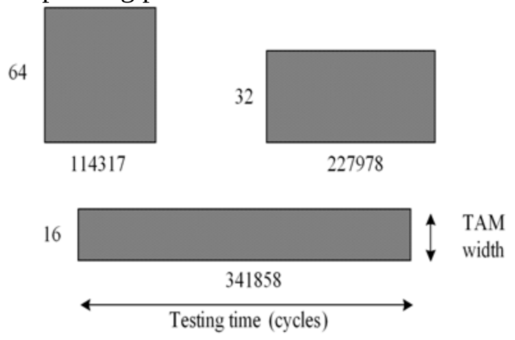

Fig. 3 Example rectangles for Core 6 in SOC p93791

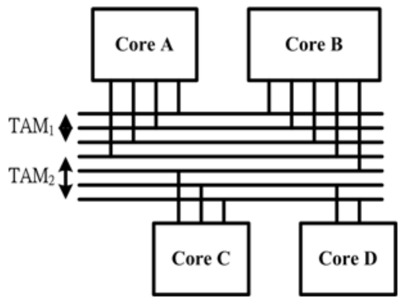

Fig. 4 TAM design using generalized rectangle packing

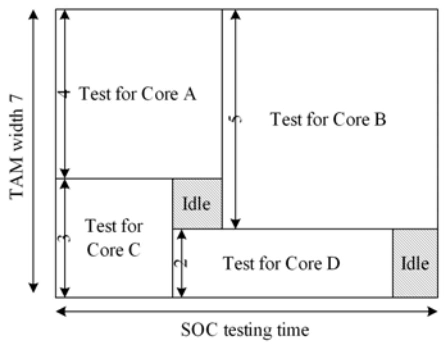

Fig. 5 The test schedule of fig.4

2.3 The General Test Schedule

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.

2.4 Using dynamic TAM width to test a core

The past test scheduling that the partition of TAM width is fixed. The test scheduling will have a lot of idle time. An example is showed in Fig. 6 there are three idle time times existing in test scheduling. The more idle time will enable a result of the test scheduling worse.

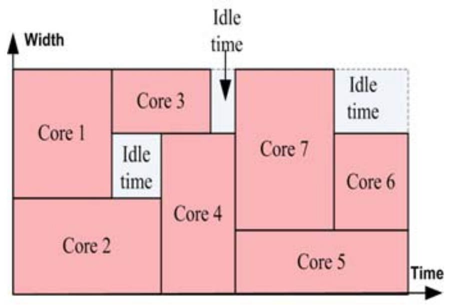

Fig. 6 SOC test scheduling

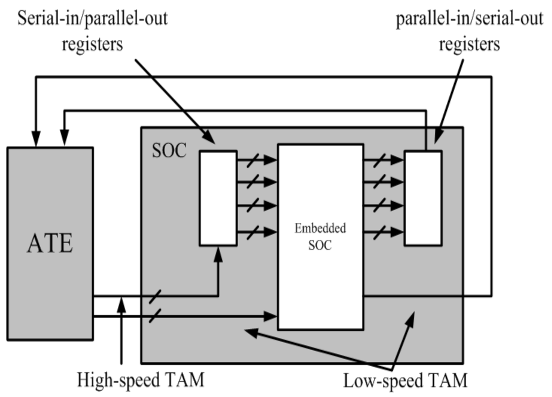
Serial-in/parallel-out registers
parallel-in/serial-out registers
SOC
ATE
Embedded SOC
High-speed TAM
Low-speed TAM

Fig. 7 Virtual TAM architecture

2.5 Virtual TAM

Sehgal [25] proposed an ideal to increase the test TAM width by using virtual TAM. We usually assume that the automatic test equipment (ATE) operates at the same frequency as the SOC core's internal scan chains frequency. In fact, the ATE operating frequency is higher than SOC core's internal scan chains frequency. Sehgal uses the characteristic of such frequency to increase the test width. Fig. 7 is the virtual TAM architecture. Quasem & Gupta (2004) proposed a framework of Reconfigurable Multiple Scan Chains. The frame can reduce the test application time, but it requires a lot of control signals. We proposed an algorithm to combine those methods and virtual enable signals. The attempt to minimize the test application time was a success.

We propose two methods for solving the problem of test scheduling. The first method is called stairway scheduling witch cut each core to become many piece among SOC. The second is using virtual TAM to control signal.

3 Proposed stairway scheduling

3.1 Concept of stairway scheduling

We propose a new test scheduling method. This method considers that testing of the core is cut many pieces and use different width of TAM to test. Then, the test scheduling of the core will become a form of stairway. We call stairway scheduling. Such, we can reduce the test application time of the core. In this paper we assume that all cores in the SOC can test at the same time, and we want to insert the core testing in the idle time. Then, the core must change the TAM width that to satisfy of TAM width of the current idle time. Such the core testing can use different TAM width to complete.

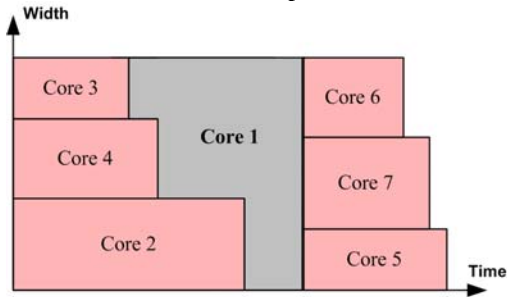

Fig. 8 SOC test of stairway scheduling

3.2 Proposed algorithm

We partition into two parts. One part describes basic core testing to complete the scheduling. This part not dynamically changes the TAM width to test the core. The design follow is shown in Fig. 9. Then, the width of TAM must have enough numbers to schedule the core at the current time. Until not exist any core enough to satisfy the TAM width at the current time. Then, we assign remnant TAM width to the largest core of scheduling. At this time, doesn't have any width of TAM enough to schedule any core. The total TAM width is zero at current time.

Another part describes dynamic testing of a core. That is called stairway scheduling. In this part, we can utilize the TAM width at the current time to test the core. Therefore, the core is completed testing through different testing resource (TAM). The scheduling is shown in Fig. 8 before.

In Fig. 10 we show the design follow of the stairway scheduling. The total TAM width is zero at current time. We move the current time to the next time and free the TAM width to test the core. This action is hold until to finish the testing of the core. Then, the TAM width equals the total width, we execute basic core testing (In Fig. 9)

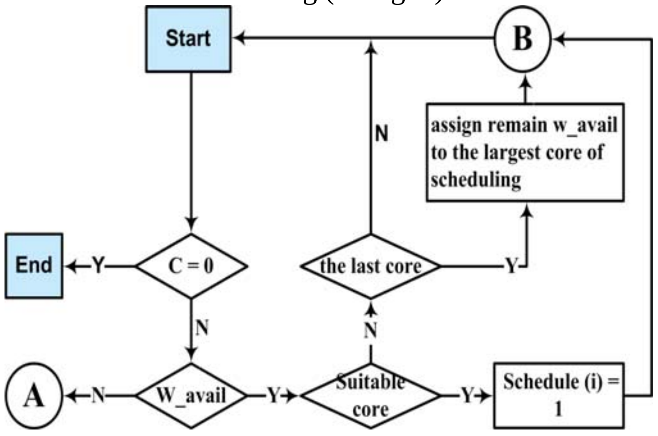

Fig. 9 The design follows of basic test scheduling (no stairway scheduling)

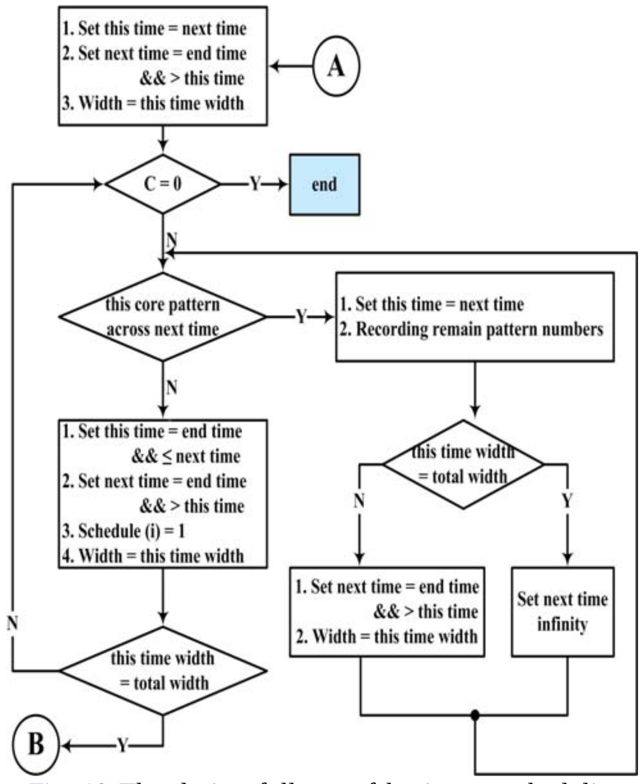
1. Set this time = next time
2. Set next time = end time && > this time
3. Width = this time width
A
C = 0
Y
end
N
this core pattern across next time
Y
1. Set this time = next time
2. Recording remain pattern numbers
N
1. Set this time = end time && ≤ next time
2. Set next time = end time && > this time
3. Schedule (i) = 1
4. Width = this time width
this time width = total width
N
Y
1. Set next time = end time && > this time
2. Width = this time width
Set next time infinity
N
this time width = total width
B
Y

Fig. 10 The design follows of basic test scheduling (stairway scheduling)

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 remnant 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;
```

Fig. 11 Proposed procedures for test scheduling

In Fig. 11, we detail the algorithm that we have developed to solve the problem of test scheduling. In our algorithm utilize the method [14] to find the initial rectangle of a core then to schedule the testing. We elaborate on each step of the algorithm in the following paragraphs.

3.3 Our algorithm explain

We use two data structures, which the one is core_initial and the core_sedule to store the TAM width, the pattern numbers, and testing time values, respectively. The structure for the core of SOC is presented in Fig. 12. This data structure update with the begin times and finish times for each core.

In Line 1, we use the method [14] 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). 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. In Line 7, we use the method [14] to assign the remained TAM width to the longest core at the current time (this_time). An example is shown in Fig. 13. Therefore, the TAM width (w_avail) equals the total (Wmax) width at the current time (this_time).

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 */

Fig. 12 Data structures of 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.

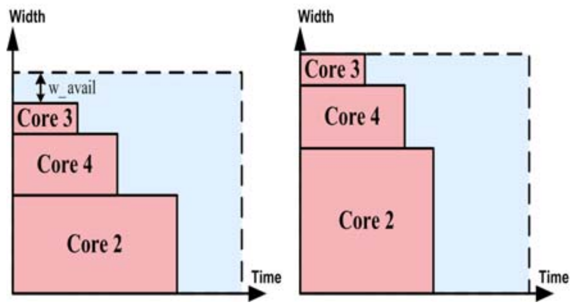

Fig. 13 Assign the remnant TAM width to the longest core

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 Fig. 14 there are two idle times in the Core 1 testing. But idle times are small. An experimental result proves the times of idle will not affect the total time of testing.

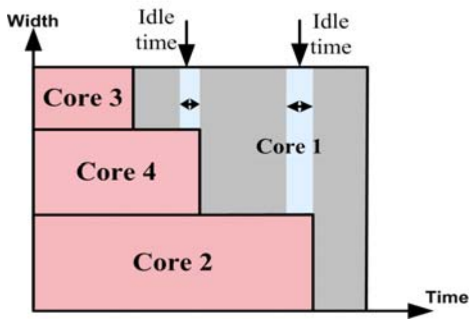

Fig. 14 the idle time of the stairway scheduling

4 Virtual Enable Signal

4.1 Concept of Virtual Enable

As mentioned in last section, the disadvantage of the virtual TAM method will limit the results. We consider that virtual TAM is a good method. Therefore the method which is the most beneficial is adopted. A method to enable signal and making the test TAM width increate was used. It shows the architecture of virtual enable signal in Fig. 15.

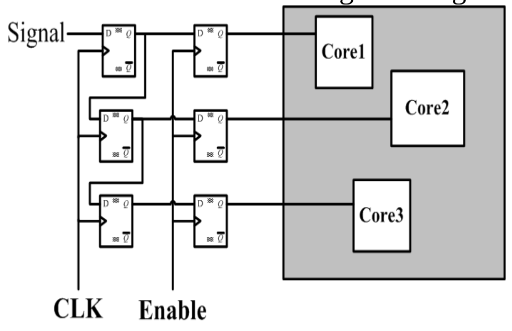

Fig. 15 Virtual control signals

In the figure, the Enable signal hold in low at start. Until the left Flip-flops holding the correct value, the Enable signal transfer from low to high. When Enable signal is high, the left Flip-flop's values shift to the right Flip-flops. Each module will need a control signal to change from normal mode to shift mode or form shift mode to normal node at mixed-architecture. Virtual TAM was used to reduce the pin's consumption and increase the test TAM width. The other benefit is that the bandwidth decreasing with control signal affects less to total test application time. The test time formula was modified in order to increase the amount of test TAM width.