온칩네트워크 기반 효과적인 테스트 스케쥴링 방법

A Novel Test Scheduling Algorithm using On-Chip Networks

안진호*, 문병인**, 강성호*
*연세대학교, **경북대학교
sominaby@yonsei.ac.kr, bihmoon@knu.ac.kr, shkang@yonsei.ac.kr

Abstract

It may be impractical to have TAM for test usage only in NoC because it causes enormous hardware overhead. Therefore, the reuse of on-chip networks for TAM is very attractive and logical. In network-based TAM, an effective test scheduling for built-in cores is also important to minimize the total test time.

In this paper, we propose a new efficient test scheduling algorithm for NoC based on the reuse of on-chip networks. Experimental results using some ITC '02 benchmark circuits show the proposed algorithm can reduce the test time by about 5 ~ 20% compared to previous methods.

I. Introduction

NoC (Networks on Chip) can be defined as a kind of SoC (System on a Chip) that has an interconnection structure like micro-networks. Micro-networks, "on-chip networks" in other words, are an on-chip interconnection that uses a network protocol based on communication layers.

Some test architectures incorporating NoC characteristics have been proposed recently. TAM (Test Access Mechanism) is the most activated area in those architectures. TAM is the physical mechanism connecting cores from test sources or sinks, and it determines how efficiently test stimuli and test results can be transported. In earlier TAM architectures for SoC, an on-chip test bus has been the most efficient form for TAM. However, it is not feasible for NoC since the separate test bus causes excessive hardware overhead. Thus the reuse of on-chip networks for TAM becomes inevitable. An effective test scheduling is also important to minimize the total test time because NoC generally includes hundreds of cores.

In this paper, we propose a new efficient test scheduling algorithm for NoC based on the reuse of on-chip networks. The proposed algorithm has two dominant features. One is a deflection routing of test packets to satisfy simple router operations and minimize hardware overhead of routers. The deflection routing algorithm can route packets rapidly without buffering and explicit flow controls. Furthermore, we improved the performance of the algorithm by considering core priorities. The other feature is an asynchronous test clock strategy. As networks are normally much faster than embedded cores, several cores can be tested simultaneously. The asynchronous test clock platform can enhance test parallelization more than the multi-source/sink platform described in the previous algorithms.

First, we review prior works and present the purpose of our work in section II. In section III and IV, two major features of the proposed test scheduling algorithm are presented respectively. The proposed test scheduling procedure is shown in section V with a pseudo-code. The experimental results using some ITC '02 benchmark circuits are given in section VI. Finally, this paper's conclusions are presented in section VII.

II. Related Work

The general concept of the reuse of on-chip networks for TAM is shown in [1]-[2]. Test scheduling methods in NoC can be grouped roughly into two main categories: a packet-based scheduling and a core-based one. A core-based scheduling method determines the test order of each core [3]. In this approach, the scheduler will assign each core a routing path, including an input port, an output port and corresponding channels that transport test vectors from the input to the core and the test response from the core to the output. Once the core is scheduled on this path, all resources on this path are reserved for the test of this core until the entire test is completed. However, it is impractical to use the core-based approach in a real situation because it is impossible to test cores with variable test clocks simultaneously. In other words, all NoC resources and CUTs should operate using the same clock during the test. A packet-based scheduling determines the order of generation and transmission of test packets for cores according to the priority of each core. E. Cota has proposed the test scheduling based on a packet-switching [4]. In [4], test vectors and test responses per core are represented as a set of packets to be transmitted throughout the networks, and the packets are scheduled to minimize the total test time using test parallelism. Test parallelism means that several cores are tested simultaneously through maximizing the network bandwidth. Some enhanced versions of this algorithm have been reported. One is the supplement of power constraints [5]. While a packet-based one having many merits, its experimental results have proven inferior to those of core-based one up to now.

III. Deflection Routing

A deflection routing, also known as a hot-potato routing, is based on the idea of delivering an input packet to an output channel in one cycle within a router. It assumes that each router has the equal number of input and output channels. In a deflection routing, when contention occurs and the desired output channel is not available, an input packet will pick any alternative available output channels to continue moving to the next router instead of waiting. C. Busch [6] presents proofs regarding a hot-potato routing algorithm without explicit flow control under dynamic packet injection. In the algorithm, a packet always tries to follow any good channel. A good channel means one that brings it closer to its destination. In contrary, a bad channel is one that does not. If a packet cannot advance to its destination, the packet is forced to follow some bad channel, in which case we say the packet is deflected. When two or more packets are competing in the same router for the same output channel, we say that there is a conflict. In order to resolve conflicts, Busch makes use of packet priorities. There are four priority states in the algorithm: Sleeping, Active, Excited and Running.

Figure 1 shows an example of the deflection routing in this paper for a packet. A packet P generated in a source follows a good channel, and its state will be changed from Sleeping to Active in proportion to network size. In Figure 1, we assume the state change always occurs. If P in the Active state is deflected, P is changed into the Excited. The state of P in Excited will be Running if P can move closer to its destination by one step. P in the Running will be routed in its home-run path. If P in the Excited or the Running state is deflected, its state will be changed into the Active.

An NoC test scheduling should be comparable for as many NoC structures as possible. Therefore, the minimal additional logic and simple control for the test are quite helpful to make the scheduling algorithm robust in whatever NoC will be used. Busch's algorithm

needs not to have any buffer in a router, and not to require any flow control. Thus the algorithm can be implemented regardless of the router structure and the routing algorithm used for data communication between cores.

While most routing procedures implemented in the proposed test scheduling algorithm are similar to Busch's, test packets for each core can be assigned with a different priority state according to the core priority when the packets are generated in a test source. For example, a test packet that belongs to the core having the highest priority can start at a higher priority state than Sleeping. The priority of a core is determined by its test time. However, the determination of its start priority by core priorities has been somewhat heuristic until now. Currently, we assign the Active state to test packets of the core whose test time is over than a tenth of the total sum of test time of all embedded cores. This heuristic approach can reduce the total test time in parallel with a sorting of the cores in decreasing order of test time as introduced in previous studies.

Fig. 1. Deflection Routing without Flow Control

IV. Asynchronous Clock Platform

The test vectors and responses are transmitted via on-chip networks. We cannot fully take advantage of the merits of on-chip networks if we use the network as a dedicated routing path for a core test. On-chip networks are generally much faster than testing speeds of embedded cores. Therefore, several cores can be tested at the same time. A NI (Network Interface) can compensate for the difference of operation clock frequency between the network and a core using buffers. In this approach, we need not to use the multi-source/sink platform described in the previous works. The difference of clock speed between the network and the core roughly corresponds to the number of test sources and sinks. For example, if on-chip networks operate two times faster than a tested core, it has the same effect as if there are two test sources and two sinks. However, an asynchronous test clock platform shows better results than multi-sources and sinks under the same condition since the asynchronous platform can fully schedule all cores by packets. The concept of an asynchronous test clock platform is presented in Figure 2. In Figure 2, R denotes a router, C is a core, and P is a test packet. The number attached to R, C, and P indicates their identity. If $R0$ operates by CLK_{R0} period, an input test packet for $C3$ at $R0$ is processed, and transmitted to a next router $R3$ for every CLK_{R0} time step. If CLK_{R0} is three times faster than CLK_{C3}, $R0$ can process test packets for $C1$ and $C2$, $P1$ and $P2$ in the Figure, while maintaining a pipelining of test vector of $C3$. At the present, Figure 2 only illustrates the idea. In the proposed algorithm, test packets for the same core are generated continuously in a test source until one test pattern is all generated if uninterrupted by higher priority cores.

Fig. 2. Concept of Asynchronous Test Clock Platform

V. Test Scheduling Procedure

In this study, we used 2-D mesh topology with channels set to be 32bit wide. A mesh structure is more practical and widespread application for NoC. Each node in a mesh is connected to its four neighbors via a bi-directional channel. A test source can inject packets at a rate of one packet per network time step, and a test sink can absorb packets at the same rate as the test source. Each packet contains a header including its destination, priority, and packet id indicating the position of the packet within a test pattern. A router will receive a packet, and decide where to go with it using the proposed routing rules based on the packet's destination.

Before the proposed test scheduling algorithm starts, we should set the coordinates of routers, the priority of cores, the number of test patterns per core, the number of test packets per test pattern, the position of cores within topology, the position of a test source and sink, and the operation clock period of networks and cores. For convenience, we assumed all cores have the same test clock, and the network speed is determined by multiples of the test clock. In the case of the priority of cores, a core with the longest test time has the highest priority. A test source and sink are directly connected to a router located in the boundary since we considered an external ATE case. For example, we illustrate the experimental conditions for d695 in Figure 3 and Table 1. Numbers at routers in Figure 3 are the coordinates of routers.

The algorithm begins with the injection of test packets of the highest priority core by the length of a test pattern. If it is done, a test source generates the test packets of the next priority core in order. However, the test source will hold the packet generation for a core whenever no output is available. Moreover, the test source immediately stops the current packet generation, and generates the test packets of the core having higher priority than the current one if the core is ready to receive the next packets. Incoming packets at routers will take priority over those generated/absorbed by the test source, sink, and cores. The time required to test a core is defined as follows.

$$T_{in} = (T_{injection} + T_{router} * N_{router} + T_{NI}) / S_{network} \quad (1)$$

$$T_{out} = (T_{absorption} + T_{router} * N_{router} + T_{NI}) / S_{network} \quad (2)$$

$$T_{packet} = T_{in} + T_{out},\ T_{pattern} = \Sigma T_{packet},\ T_{core} = \Sigma T_{pattern} \quad (3)$$

where T_{in} means the time to transmit a test vector packet from a test source to a core. Tout means the time to receive a test response packet from the core to a test sink. $T_{injection}$ is the time for injecting a packet in the test source, and $T_{absorption}$ is for absorbing in the test sink. T_{router} is the time for routing a packet in a router, and N_{router} is the number of routers in the routing path. T_{NI} is the time to process a packet in NI. $S_{network}$ is the relative clock speed of on-chip networks compared to a test clock, and is restricted to an integer value. In Equation (3), T_{packet} denotes the delivery time of one test packet from a test source to a test sink, $T_{pattern}$ denotes the time of one test pattern, and T_{core} is the final result that represents the total test time of a core. We assume $T_{injection}$, $T_{absorption}$, T_{router}, and T_{NI} are all 1. The pseudo-code of the algorithm is shown in Figure 4. The total test time of NoC equals to the summation of T_{core} of all embedded cores.

Fig 3. NoC Topology in d695

Table 1. Test Packets for d695

Core	# of test patterns	# of packets per test pattern	Test time	Priority
1	12	1	25	9
2	73	7	588	8
3	75	32	2507	6
4	105	54	5829	2
5	110	55	6206	1
6	234	41	9869	0
7	95	34	3359	5
8	97	46	4605	3
9	12	64	836	7
10	68	55	3863	4

```
program NoC_test_schedule
  set mesh size, the position of embedded cores;
  set the priority of cores by their test time;
  set test information of each core;
  set the position of a test source and sink;
  set the on-chip network speed;
begin
  until(all cores are tested) {
    repeat network speed {
      Router_operation in all routers;
      Test_source_operation;
      Test_sink_operation;
    }
    test operation in all cores;
  }
end
program Router_operation
begin
  while(there is an input packet) {
    case(packet.destination)
      when  'here'  :
        if(dest. is core and NI input port is idle)
          transfer packet into NI;
        else if(dest. is sink and sink input port is idle)
          transfer packet into sink;
        else
          deflects to random output port;
      when  'not here'  :
        deflection routing by the packet destination;
  }
end
program Test_source_operation
begin
  if(there is a free port) {
    fetch a test packet from queue of a test source;
    deflection routing by the packet destination;
    generate a next test packet and queuing;
  }
end
program Test_sink_operation
begin
if(there is a input packet) {
    absorb a test packet;
    update test information;
    analyze the test response;
  }
 end
```

Fig 4. Pseudo-Code of NoC Test Scheduling

VI. Experimental Results

The proposed algorithm is evaluated by the test application time through C-level simulations. The algorithm takes a minute in a d695, and is not over 10 minutes even in a p93791 on a Sun Microsystems 1.2-GHz UltraSPARC® III. Though there are some variation of simulation results according to the position of a test source, a sink, and cores, the extent of variation due to it is not significant. Table 2 displays the simulation results of the proposed algorithm compared to previous results for four different ITC '02 benchmark circuits.

Table 2. Test Scheduling Results

Circuit Name	# of Ports or Relative Clock Speed	Results in (3)	Results in (4)	Proposed
d695	2/2 or 2	18869	26012	20075
	3/3 or 3	13412	20753	12759
	4/4 or 4	10705	14785	10774
	- or 5	N.A	N.A	10088
	- or 6	N.A	N.A	10035
g1023	2/2 or 2	25062	31898	28616
	3/3 or 3	17925	22648	18434
	4/4 or 4	16489	18851	16168
	- or 5	N.A	N.A	15800
	- or 6	N.A	N.A	15360
p22810	2/2 or 2	271384	315708	312296
	3/3 or 3	180905	222432	208595
	4/4 or 4	150921	170999	162661
	- or 5	N.A	N.A	132982
	- or 6	N.A	N.A	115409
p93791	4/4 or 4	333091	435787	342819
	- or 5	N.A	N.A	285793
	- or 6	N.A	N.A	248108

Results in (3) based on core□based scheduling is superior to those of the proposed in some cases. However, as mentioned in section II, the core-based approach is so theoretical and impractical that it is not suitable for a real situation. We expected the results of the proposed algorithm would exceed those of (4) if we increase the test clock frequency of networks and cores with high priority. As compared with results in (4) based on packet-based scheduling similar to the proposed one, the test scheduling results using the proposed idea is quite noticeable in all cases. Furthermore, we can see from the results that the proposed algorithm is more effective under conditions of

large circuits and high speed networks. This is very encouraging for the practicality and feasibility of the proposed algorithm.

In Table 3, the analysis results of the routing algorithm used in the proposed algorithm are discussed. The number of hops indicates the average number of router that a test packet visits from a test source to a test sink. The number of deflections indicates how often a test packet is deflected on the way from a test source to a test sink. Generally, the number of hops is increased as circuit size grows since a greater percentage of packets have changed to higher states; thus, conflict within a router increases. However, the increment of network speed can reduce the confliction because the packet absorption rate increases linearly with respect to the network speed. Therefore, the average number of hops and deflections gradually decreases as the network clock speed increases.

Table 3. Statistics of Deflection Routing

Circuit Name	Mesh Size	Relative Clock Speed	# of Hops	# of Deflections
d695	3 * 4	2	10.4	5.1
		3	7.1	2.7
		4	8.8	4.1
		5	7.1	2.8
		6	8.0	3.5
g1023	4 * 4	2	11.5	5.3
		3	10.6	4.7
		4	10.3	4.5
		5	8.9	3.5
		6	9.6	4.1
p22810	6 * 6	2	26.6	15.8
		3	24.8	14.8
		4	25.1	15.2
		5	22.2	13.2
		6	20.4	11.9
p93791	6 * 6	4	24.2	14.5
		5	21.6	12.8
		6	21.6	12.9

VII. Conclusion

In this paper, we propose a new efficient test scheduling algorithm for NoC based on the reuse of on-chip networks. The proposed algorithm adopts a deflection routing without a flow control to minimize test hardware overhead, and extend the routing algorithm to consider the core priority. Moreover, we develop an asynchronous test clock platform. The asynchronous test clock platform enhances the test parallelization more than the multi-source and sink platform described in the previous studies, thus improves test-scheduling results. Experimental results using some ITC '02 benchmark circuits show that the proposed algorithm provides superior results in spite of low hardware overhead. We expect the proposed test scheduling algorithm will be widely applicable due to its feasibility and practicality.

References

[1] B. Vermeulen, J. Dielissen, K. Goossen, and C. Ciordas, "Bringing Communication Networks on a Chip: Test and Verification Implications," IEEE Communications Magazine, pp. 74-81, 2003

[2] M. Nahvi and A. Ivanov, "Indirect Test Architecture for SoC Testing," IEEE Trans. on CAD, pp. 1128-1142, 2004

[3] C. Liu, E. Cota, H. Sharif, and D. K. Pradhan, "Test Scheduling for Network-on-Chip with BIST and Precedence Constraints," Proc. IEEE ITC, pp. 1369-1378, 2004

[4] E. Cota et al. "The Impact of NoC Reuse on the Testing of Core-based Systems," Proc. IEEE VTS, pp. 128-133, 2003

[5] E. Cota, L. Carro, F. Wagner, and M. Lubaszewski, "Power-Aware NoC Reuse on the Testing of Core-Based Systems," Proc. IEEE ITC, pp. 612-621, 2003

[6] C. Busch, M. Herlihy, and R. Wattenhofer, "Routing without Flow Control," Proc. the 13th ACM Symposium on Parallel Algorithms and Architectures, pp. 11-20, 2001