논문 2006-43SD-1-10

Rectangle Packing 방식 기반 NoC 테스트 스케쥴링

(NoC Test Scheduling Based on a Rectangle Packing Algorithm)

안 진 호*, 김 근 배*, 강 성 호**

(Jin-Ho Ahn, Gunbae Kim, and Sungho Kang)

요 약

NoC 테스트는 온칩네트워크를 TAM으로 재사용하기 때문에 SoC 구조 기반의 여러 테스트 기법을 그대로 사용할 수가 없다. 본 논문에서는 네트워크 기반 TAM의 문제점을 크게 감소시킨 새로운 형태의 NoC 테스트 플랫폼을 소개하며 이를 이용한 NoC 테스트 스케쥴링 알고리즘을 제안한다. 제안한 알고리즘은 SoC 테스트 용도로 개발된 rectangle packing 방식을 기반으로 효율적이고 체계적인 테스트 스케쥴링이 가능하게 한다. ITC'02 벤치회로를 이용한 실험 결과 제안한 방법이 기존 방법에 비해 최대 55%까지 테스트 시간을 줄일 수 있음을 확인하였다.

Abstract

An NoC (Networks-on-Chip) is an emerging design paradigm intended to cope with a future SoC containing numerous built-in cores. In an NoC, the test strategy is very significant for its practicality and feasibility. Among existing test issues, TAM architecture and test scheduling will particularly dominate the overall test performance. In this paper, we address an efficient NoC test scheduling algorithm based on a rectangle packing approach used for an SoC test. In order to adopt the rectangle packing solution as an NoC test scheduling algorithm, we design the configuration about test resources and test methods suitable for an NoC structure. Experimental results using some ITC'02 benchmark circuits show the proposed algorithm can reduce the overall test time by up to 55% in comparison with previous works.

Keywords: NoC, 테스트 스케쥴링, Rectangle Packing

I. 서 론

반도체 설계 기술이 급속하게 발전함에 따라 하나의 SoC(System-on-Chip)에 집적되는 코어의 수 역시 빠르게 증가하고 있다. 현재의 추세라면 가까운 시일 안에 수백 개의 코어를 내장한 SoC가 개발될 것으로 기대된다. 이러한 고밀도 SoC 구조에서는 내장 코어 사이의 데이터 전달 방법이 전체 SoC 성능을 결정하는 중요한 요소가 된다.

NoC(Networks-on-Chip)는 고밀도 구조에 적합한 마이크로네트워크 연결 형태를 가지는 SoC로 정의할 수 있다[1]. 마이크로네트워크란 다른 말로 온칩네트워크(On-chip Networks)라고도 하는데 컴퓨터 네트워크에서 사용되는 레이어 기반의 온칩 연결 형태를 의미한다. 온칩네트워크 하에서는 새로운 코어의 추가나 기존 코어의 삭제가 자유로우며, 코어 사이의 데이터 전송이 패킷 방식으로 이루어지기 때문에 많은 코어들의 동시 메시지 전송이 가능하다. 또한 온칩네트워크와 코어의 동작 속도가 완전히 분리되기 때문에 시스템 전체적으로는 비동기적이고 국부적으로 동기화되는 형태를 갖는다. 이것은 많은 수의 클럭이 동시에 사용되는 고밀도 SoC 구조에서는 필수적인 것이다[2].

SoC와 마찬가지로 NoC도 물리적인 결함을 검출하기 위해 테스트 과정을 거쳐야 한다[3][4]. NoC 테스트는 SoC 테스트에서 사용되는 방법과 큰 차이가 없지만 NoC의 구조적인 특징으로 인하여 SoC 테스트 알고리즘을 그대로 적용할 수는 없다. 본 논문에서는 NoC 내장 코어의 래퍼 디자인, 테스트 패턴 및 응답 입출력 구

* 학생회원, **평생회원 연세대학교 전기전자공학과
(Yonsei Univ., Electrical and Electronic Engineering)
접수일자 : 2005년11월1일 수정완료일 : 2005년1월6일

조, 테스트 패킷의 구성과 라우팅 알고리즘을 개선하여 SoC 테스트 스케쥴링 방법으로 제안되었던 기존의 rectangle packing 알고리즘을 NoC 테스트에 접목하는 새로운 형태의 NoC 테스트 스케쥴링 알고리즘을 제안한다. 제안된 알고리즘은 ITC'02 테스트 벤치마크 회로를 이용하여 기존의 방식과 비교되었으며 스케쥴링 결과나 계산 시간 모든 면에서 상당히 우수함이 증명되었다.

본 논문은 다음과 같이 구성된다. Ⅱ장에서는 SoC와 NoC 테스트 방법의 비교 및 NoC 테스트 스케쥴링을 위해 기존에 제안되었던 방식을 설명한다. Ⅲ장에서는 제안하는 스케쥴링 알고리즘의 적용을 위해 필요한 NoC 테스트 구조 및 방법에 대하여 소개하고 Ⅳ장에서 스케쥴링 과정을 자세히 소개한다. Ⅴ장에서는 실험 환경 및 결과를 언급하며, Ⅵ장에서 결론을 내며 논문을 마무리한다.

Ⅱ. SoC와 NoC 테스트의 비교

SoC 내장 코어의 테스트 방법은 일반적으로 TAM (Test Access Mechanism) 구조의 설계, 테스트 래퍼 디자인, 그리고 테스트 스케쥴링 등으로 분류된다. 테스트 래퍼는 SoC에 집적된 코어를 테스트하기 위하여 주변의 로직들과 격리시키고 TAM을 통해 테스트 데이터를 주고받을 수 있도록 해준다. TAM은 SoC 외부의 입출력 핀들을 통하여 내부 코어의 테스트 래퍼와 테스트 데이터를 주고받을 수 있도록 해주는 구조를 의미한다. 테스트 스케줄링이란 TAM 폭과 파워 등의 주어진 제약 조건 하에서 SoC내의 모든 코어를 테스트하는데 걸리는 시간을 최소화할 수 있는 내장된 코어의 테스트 조합을 찾는 것이다. 이러한 SoC 테스트 스케줄링과 관련된 연구는 이전부터 활발히 진행되어 왔다[5]~[12]. 그러나 지금까지 연구된 SoC 테스트 스케쥴링 알고리즘을 NoC 테스트에 바로 사용할 수는 없다. 가장 큰 이유는 바로 TAM 구조의 제한 때문이다. NoC는 SoC와 달리 내부적으로 테스트를 위한 별도의 TAM을 삽입하면 하드웨어 오버헤드가 너무 커지기 때문에 NoC의 코어 연결 구조, 즉 온칩네트워크를 TAM으로 재활용하는 방식이 필연적이다. 따라서 테스트 패턴이나 응답 역시 네트워크 상에서 패킷 형태로 움직이기 때문에 코어별로 할당되는 TAM 폭이 네트워크를 구성하는 채널 폭과 동일하게 된다. 예를 들어 온칩네트워크의 채널 폭이 32비트라면 모든 코어에 할당되는 TAM 폭 역시 32비트가 된다. 코어별로 할당된 TAM 폭이 같은 크기로 고정된 구조보다 할당 가능한 TAM 폭이 가변적인 구조의 테스트 효율성이 우수함은 쉽게 예측할 수 있다.

최근에 NoC의 비효율적인 TAM 구조로 인하여 발생하는 문제를 개선하고자 CUT(Core under Test)에서 요구되는 TAM 폭을 w', 네트워크 채널 폭을 w라 정의하고 w/w'가 정수배 n일 때, 한번에 정수 n개에 해당하는 테스트 벡터를 하나의 패킷에 보내고 CUT와 테스트 resource, 즉 테스트 source와 sink, 사이의 연속성(Pipelining)을 유지하기 위해서 CUT의 테스트 클럭을 네트워크의 테스트 패킷 전송 속도보다 n배만큼 빨리 동작하게 하는 구조를 제안하였다[13]. 테스트의 연속성이란 코어의 스캔(Scan) 데이터 입력과 동시에 출력 데이터가 발생하는 동작이 연속적으로 진행되는 것을 의미한다. 제안된 방식은 CUT의 테스트 시간이 테스트 클럭과 비례하여 감소하기 때문에 NoC 전체적인 테스트 시간을 단축시킬 수 있었다.

그러나 [13]에서 제안한 방식은 실제 적용 시 많은 문제가 발생한다. 예를 들어 CUT의 TAM 폭이 2비트이고 온칩네트워크의 채널 폭이 32비트이면 하나의 패킷으로 전송 가능한 패턴의 수는 16개이다. 따라서 테스트의 연속성을 위해서 필요한 CUT의 동작 속도는 네트워크의 속도의 16배가 되야 한다. 시스템 전체적인 파워 소모 문제나 온칩 PLL의 설계 난이도를 고려할 때 이렇듯 다양하고 빠른 테스트 클럭을 제공하기는 곤란하므로 결국 네트워크의 속도를 낮추거나 또는 동시에 전송되는 패턴의 수를 제한해야만 할 것이다. 또한 상기 구조 하에서의 테스트 스케쥴링도 TAM 폭을 고정시키고 선험적(Heuristic)인 알고리즘으로 결과를 도출하는 이전의 방식[14]을 그대로 유지하였다. [14]에서 사용한 방법은 코어 단위로 테스트 순서를 결정하는 방식이다. 먼저 우선순위가 높은 순서대로 코어를 선택하고 선택된 코어의 테스트를 위한 라우팅 경로를 설정한 후 테스트를 시작한다. 한번 설정된 코어의 라우팅 경로 및 해당 경로를 구성하는 네트워크 resource들은 진행 중인 코어의 테스트가 완료될 때까지 보존되기 때문에 같은 경로를 사용해야 하는 다른 코어들을 동시에 테스트하는 것이 불가능하다. 물론 다른 경로가 존재하는 코어들은 현재 진행 중인 코어 테스트와 병렬적으로 테스트가 가능하다. 이와 같이 일단 테스트가 시작되면 테스트가 종료될 때까지 현재 상태가 유지되는 비선점(Non-Preemptive) 방식[13][14]은 테스트되는 코어의 테스트 벡터 생성과 해당 벡터의 응답 출력까지의 연속성이 유지되기 때문에 각 코어의 테스트 시간을 최소화시킬 수 있다. 따라서 패킷 단위의 선점(Preemptive) 방식[15]~[18]보다 우수한 스케쥴링 결과를 보여준다. 그러나

라우팅 경로의 독점적 사용을 위한 구조적인 제한이 많이 수반된다.

결론적으로 임의의 TAM 폭을 코어별로 할당 가능한 SoC에 비해 온칩네트워크를 TAM으로 재사용하는 NoC는 구조적으로 기존의 SoC 테스트 스케쥴링 알고리즘을 적용할 수 없기 때문에 지금까지 별도의 선험적인 스케쥴링 알고리즘을 개발해왔다. 그러나 현재까지 발표된 동일 벤치마크 회로에 대한 스케쥴링 결과를 비교하면 SoC 구조의 스케쥴링 결과가 월등히 우수함을 알 수 있다. 따라서 NoC의 구조적인 한계로 인한 테스트 비효율성을 개선시키기 위해 NoC와 SoC의 테스트 구조를 체계적으로 접목하는 것이 필요하다.

Ⅲ. NoC 테스트 플랫폼

1. NoC 기본

NoC는 온칩네트워크와 내장 코어들로 구성되며, 온칩네트워크는 라우터와 라우팅 채널, 그리고 네트워크 연결부(NI: Network Interface)로 구성된다. 라우터는 라우팅 채널과 채널을 연결해주며 입력된 데이터의 목적지에 따라 어느 출력 포트로 데이터를 보낼 것인지를 라우팅 알고리즘에 따라 결정한다. 라우팅 채널은 물리적인 데이터 연결 통로이며 NI는 내장 코어와 라우터를 연결해준다. 온칩네트워크의 구성 방법에 따라 다양한 형태의 NoC 구조가 만들어지는데 일반적으로 네트워크 토폴로지(Topology), 프로토콜(Protocol), 라우터의 구조 및 동작 방법 등을 기준으로 분류된다. 본 논문에서는 가장 일반적인 2차원 메쉬(Mesh)를 기본 토폴로지로 하고 XY routing, worm-hole switching, 그리고 credit-based flow control을 사용한다고 가정한다. 또한 모든 라우터에는 입력 포트에 flit 버퍼가 있고 flit 하나의 크기는 채널 폭과 동일하다. 채널 폭은 16 혹은 32비트의 2가지 경우로 한정한다. 그림 1에서 상기 NoC 구조로 회로를 구성한 예를 보여준다. 내장 코어들은 헤더(Header)와 페이로드(Payload), 그리고 트레일러(Trailer)로 구성되는 패킷을 통해 데이터를 주고받는다. 테스트 벡터와 응답 역시 패킷 단위로 전송된다.

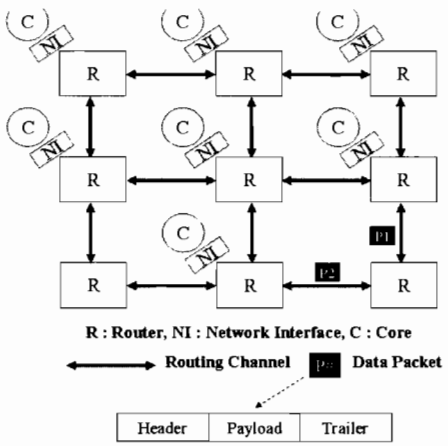


그림 1. NoC의 구성 예
Fig. 1. An Example of NoC Structure.

지금까지의 NoC 테스트 구조는 여러 개의 테스트 source와 sink가 내장 코어를 통해 네트워크에 연결되는 멀티 입출력 구조를 기본 형태로 한다[13]~[17]. 그러나 [3]과 [18]에서 제안한 방식과 같이 라우터에 직접 테스트 source와 sink를 연결하는 방식이 보다 더 현실적이며 이러한 형태는 현재 널리 사용되는 AMBA 버스를 이용한 ARM의 테스트 인터페이스에서도 쉽게 그 효용성을 확인할 수 있다[19]. 따라서 본 논문에서는 멀티 코어 입출력 구조 대신 라우터에 직접 연결되는 테스트 패턴 입력 및 응답 출력 구조를 갖는다. 멀티 입출력 구조에서 사용되는 입출력 포트 수는 테스트 패킷의 전달 속도를 통해서 조절될 수 있다. 즉 입출력 포트 수가 각각 3개인 경우는 네트워크 동작 속도가 코어의 테스트 속도보다 3배 빠른 경우와 동일하다고 볼 수 있다. 물론 3개의 패턴이 동시에 전달되는 멀티 입출력 모드와 패턴 3개를 차례대로 빠르게 전달하는 모드는 직/병렬 전송 방식 차이로 인한 알고리즘의 수정을 유발하지만 다음에서 설명할 TAM 폭의 가변성과 테스트 resource와 CUT의 연속성 유지 측면에서 직렬 전송 방식이 보다 더 유리함을 알 수 있다.

2. TAM 폭의 가변성

NoC 테스트 스케쥴링을 SoC 구조를 기반으로 하는 기존 알고리즘에 접목하기 위해서는 일단 코어에 할당되는 TAM 크기를 다양하게 해야 한다.

본 논문에서는 [13]에서 제안한 방법과 같이 하나의 패킷에 여러 개의 테스트 패턴을 보내는 방법을 적용한다. 그러나 2장에서 언급한 바와 같이 테스트 클럭 가변으로 인한 문제를 해결하고자 시분할 패킷 전송 방식을 도입한다. 시분할 패킷 전송이란 모든 코어의 테스트 클럭을 동일하게 설정하고 패킷 하나에 포함된 패턴 수만큼 시간적인 차이를 두고 패킷을 전송하는 방법이다. 즉 한번에 많은 수의 테스트 패턴을 보내고 다음 패턴 전송 시까지 남는 시간동안 다른 코어의 테스트 패턴을 전송하면 CUT의 연속성을 유지하면서 테스트 병렬성을 높일 수 있어 우수한 스케쥴링 결과를 얻을 수 있다. 이 방식은 [18]에서 사용한 패킷 전송 방식과 유사하나 [18]에서는 코어의 우선순위에 따라 패킷 전송 순서를

결정하는 preemptive 방식이지만 제안한 방식은 CUT 별 패킷 전송이 서로 겹치지 않게 미리 설정하는 non-preemptive 방식이다. 그림 2에서 간단한 예를 나타내었다. NoC 채널 폭이 32비트이고 현재 테스트 중인 코어 3개에 대하여 각각 16비트, 8비트, 8비트씩 TAM이 할당되어 있다. 모든 CUT와 네트워크의 동작 속도가 동일하다고 가정할 때 먼저 CUT_1을 위하여 테스트 패턴 2개가 들어있는 32비트 패킷 P1을 보낸다. 그리고 다음 클럭에 CUT_2용 테스트 패턴 4개가 들어있는 패킷 P2를 보낸다. 그 후 다시 CUT_1을 위한 테스트 패킷을 보내야하는데 이것은 패킷 1개당 포함된 패턴의 수가 CUT_2나 CUT_3는 4개인 반면에 CUT_1은 2개밖에 안되므로 테스트 연속성을 위해서 다른 CUT에 비해 2배의 속도로 패킷을 보내야 하기 때문이다. 다음 클럭에는 CUT_3용 패킷 P3을 보낸다. 3개의 CUT 중 어느 1개의 테스트가 완료되는 시점까지 상기 과정은 반복된다. 만약 CUT_1의 테스트가 제일 먼저 끝나게 되면 CUT_2와 CUT_3의 패킷 전송 시점은 그대로 유지한 채 CUT_1의 패킷 전송 시점에 다른 코어의 테스트 패킷을 전송하면 된다. 단 시분할 방식으로 다양한 크기의 TAM 폭을 갖는 코어들을 혼합하여 테스트 할 경우 전송 시점의 주기 계산이 상당히 복잡해진다. 따라서 본 논문에서는 주기 계산의 용이함과 채널 폭의 효율적 사용을 위해 TAM 크기를 2의 지수배, 즉 1, 2, 4 ... , 2^k (단 $2^k \leq W$)의 크기로 제한한다. 여기서 W는 TAM 폭의 최대 값, 즉 네트워크의 채널 폭을 의미한다.

시분할 패킷 전송 방식은 온칩네트워크나 코어의 테스트 속도에 의한 영향이 거의 없다. 예를 들어 네트워크의 속도가 코어 속도보다 n배 빠른 테스트 구조에서는 n개의 W비트 TAM이 병렬로 존재한다고 가정하고 각각 스케쥴링을 하면 된다. 그러나 하나의 코어를 테스트하기 위하여 2개 이상의 TAM을 동시에 사용하는 방식은 국부적으로 전송 시점의 주기 계산이 틀어지

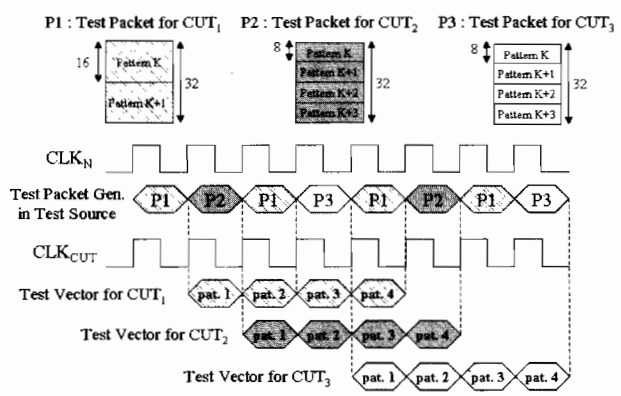


그림 2. 시분할 패킷 전송
Fig. 2. Time-Division Packet Transmission.

기 때문에 전송 시점의 완벽한 주기성을 위해서 현재까지는 고려하지 않았지만 라우터와 NI에 약간의 버퍼 추가만으로 국부적인 주기 이탈 문제는 쉽게 해결할 수 있으므로 이를 이용한 추가적인 테스트 시간 감소가 가능하다. 또한 코어의 테스트 클럭이 감소하거나 증가하는 경우는 시분할 전송 시점의 증가 혹은 감소로 처리할 수 있다. 예를 들어 테스트 클럭이 2배 증가하면 전송 주기를 2배로 감소시키고, 클럭이 2배 감소하면 주기를 2배로 증가시키면 된다. 이것은 스케쥴링 알고리즘에서 할당된 TAM 크기가 2배로 증가하거나 감소하는 것으로 처리하면 손쉽게 구현 가능하다.

3. 테스트 Resource와 CUT의 연속성 유지

테스트 연속성은 테스트 시간과 시분할 전송 시점의 정확한 계산을 위해서 반드시 풀어야할 문제이다. 특히 전송 중인 패킷들이 상호 충돌하지 않도록 하는 효과적인 라우팅 알고리즘이 필요하다. 서로 다른 코어의 테스트 패킷들이 같은 경로를 사용하지 않도록 미리 결정하는 이전의 방식에서는 시분할 패킷 전송으로 인한 테스트 병렬성 향상 효과가 크게 감소한다. 따라서 테스트 패킷의 라우팅 특성을 고려한 새로운 형태의 라우팅 방법을 도입하였다.

NoC의 테스트 구조를 보면 테스트 source와 sink의 수는 각 1개인 반면 테스트 되는 코어의 수는 상당히 많다. 즉 테스트 source에서 CUT까지는 one-to-many 형태의 라우팅 특성을 가지며, 반대로 CUT에서 테스트 sink까지는 many-to-one의 라우팅 특성을 가진다. One-to-many 형태의 라우팅에서 one에 해당하는 노드가 시분할적으로 패킷을 전송하고 라우팅 과정에서 패킷의 역행이 없다면 라우팅 경로가 중복되더라도 패킷 충돌이 발생하지 않는다. 따라서 테스트 source에서 CUT까지는 기본 라우팅 알고리즘인 XY routing을 사용한다. XY routing이란 목적지 노드로 이동할 때 먼저 X축 방향으로 이동한 다음 Y축 방향으로 이동하는 방식으로 dimension order routing이라고도 한다. 그러나 CUT에서 테스트 sink까지는 상기와 같은 방식을 사용할 수가 없다. 다양한 위치의 CUT에서 임의 시점에 발생하는 패킷들을 XY routing 방식을 사용하여 서로 충돌없이 동일한 목적지까지 전송 가능하도록 스케쥴링 하는 것은 불가능하기 때문이다. 이와 같은 문제를 해결하기 위해서 우리는 global combining 방식[20]을 사용하였다. Global combining이란 CUT별 응답 패킷을 생성할 때 테스트 패턴용 패킷처럼 여러 데이터를 채널 폭 크기만큼 그룹으로 묶어서 전송하는 것이 아니라 패턴

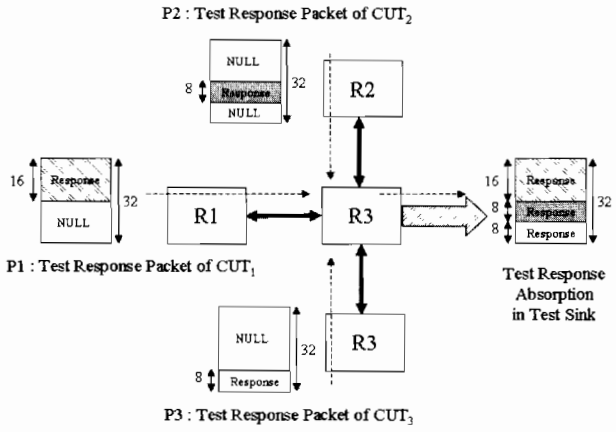


그림 3. 테스트 응답 패킷 라우팅
Fig. 3. Test Response Packet Routing.

하나에 해당하는 응답 값만 바로 전송하고 라우터에서는 입력되는 패킷들의 유효 데이터 영역을 통합하여 하나의 패킷으로 재생성해서 전송하는 방식이다. 그림 3에서 예를 나타내었다. 할당된 TAM 폭이 16비트인 CUT_1과 8비트인 CUT_2와 CUT_3가 테스트 응답 패킷을 생성할 때 CUT_1은 전체 32비트의 패킷 중 상위 16비트에 응답 값을 보내고 CUT2와 CUT3은 하위 16비트 중 각 8비트씩 분리하여 데이터를 보낸다. 페이로드 영역중 유효 데이터 영역에 대한 정보는 패킷 헤더에 추가하면 된다. 라우터 R3에서는 동시에 입력된 패킷 P1, P2, P3의 페이로드 영역을 combining하고 헤더의 유효 데이터 위치 및 기타 정보를 통합, 수정한 후 하나의 패킷으로 변환, 테스트 sink로 출력한다. 패킷별 유효 데이터의 위치가 중복되는 방식일 경우에도 이에 상응하는 정보만 헤더에 포함시키면 처리 가능하다. 제안한 라우팅 방식은 동시에 여러 개의 테스트 응답 패킷이 하나의 라우터에 입력되더라도 동시에 테스트 되는 CUT들의 응답 크기의 합은 최대 채널 크기이므로 충돌로 인한 시간

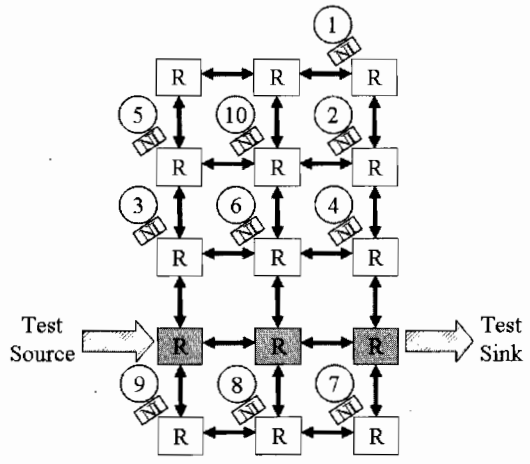


그림 4. NoC 기반 d695 회로의 테스트 구조
Fig. 4. Test Platform of d695 based on NoC

적인 지연 없이 패킷 전송이 가능하며 XY routing을 사용해도 테스트 연속성을 보장할 수 있다. 또한 라우터 내에서 combining을 위한 추가 하드웨어나 설계 복잡도는 그리 늘어나지 않을 것으로 판단된다.

그러나 테스트 source와 CUT, 그리고 테스트 sink의 위치에 따라 테스트 패턴 패킷과 응답 패킷의 충돌이 발생할 수 있다. 이러한 문제는 충돌이 일어나지 않도록 테스트 source와 sink의 위치를 효과적으로 정하면 해결될 수도 있지만 NoC 구조에서 코어의 추가 및 삭제가 자주 발생한다는 가정 하에 CUT들의 위치에 따른 영향을 받지 않도록 사전에 결정하는 것이 필요하다. 이것은 테스트 source와 sink를 같은 행 또는 열에 배치하고 해당 행 또는 열에는 코어를 배치하지 않으면 쉽게 구현될 수 있다. 이로 인해 추가되는 네트워크 resource는 전체 NoC 크기에 비교하면 극히 미미할 것이다. 지금까지 설명한 내용을 바탕으로 NoC 기반 d695 회로를 테스트하기 위한 전체 구조를 그림 4에서 나타내었다.

Ⅳ. 테스트 스케쥴링 알고리즘

3장에서 설명한 테스트 구조 및 방법을 적용하면 NoC 테스트에 기존 SoC 테스트 스케쥴링 알고리즘을 쉽게 접목할 수 있다. 본 논문에서는 지금까지 제안된 여러 SoC 스케쥴링 알고리즘 중 [6]에서 제안한 rectangle packing 방식을 적용한다. 코어 테스트는 할당된 TAM을 높이로 하고 이 때 소요되는 테스트 시간을 너비로 하는 사각형으로 표현할 수 있다. Rectangle packing 방식이란 이러한 사각형들을 TAM의 최대 크기를 제한 높이로 하는 박스 안에 박스의 폭이 최소화되도록 채워 넣는 방법을 의미한다.

먼저 테스트 스케쥴링에 앞서 래퍼 디자인을 해야 한다. 래퍼 디자인이란 코어의 테스트 시간 최소화를 위해 테스트 래퍼에 스캔 체인 및 입출력 포트를 효율적으로 할당하는 것이며 일종의 NP-hard 문제이다. 본 논문에서는 래퍼 디자인을 위해서 [12]에서 제안한 one-element exchange 방법을 사용하여 코어별로 TAM 폭과 이에 대한 테스트 시간으로 표현되는 사각형 집합을 구한다. 그러나 전체 TAM 폭을 모두 고려하지 않고 3장에서 언급한 바와 같이 2의 지수 값 중에서 pareto-optimal point에 해당하는 값만 스케쥴링의 대상으로 한다. 상기 과정을 거쳐 구한 코어별 사각형 집합을 이용하여 rectangle packing 방식의 스케쥴링을 하게 되며 그 내용은 다음과 같다.

- 데이터 구조

먼저 코어별로 그림 5와 같은 데이터 구조를 설정한다. 테이터 구조는 선호 TAM 크기(W_{prefer}), 최종 결정된 TAM 크기($W_{assigned}$), 테스트의 시작과 끝 시간, 스케쥴링 여부, 테스트 완료 여부, 사용한 TAM 위치로 구성된다. W_{prefer}는 할당 가능한 TAM 폭 중 테스트 시간 대비 상대적으로 효율적인 TAM 폭을 의미하며 $W_{assigned}$는 스케쥴링 결과 최종 할당된 TAM 폭이다.

- W_{prefer}의 결정

Wprefer는 코어별 테스트 파라미터에 따라 다양한 결과가 도출될 수 있다. 본 논문에서 Wprefer를 얻기 위해 사용한 방법은 TAM 폭이 1일 때의 사각형 면적과 TAM 폭이 2의 지수 값으로 증가할 때의 사각형 면적을 순서대로 비교하여 그 차이가 TAM 폭이 1일 때의 사각형 면적의 p%를 초과하면 바로 이전의 TAM 폭을Wprefer로 설정하였다. 이것은 예를 들어 TAM 폭이 2배로 커지면 테스트 시간은 2배로 감소하는 것이 이상적이나 실제로는 그에 미치지 못하므로 TAM 폭의 증가가 테스트 시간에 미치는 영향이 미미해지면 더 이상의 TAM을 할당하지 않는 방식이다. 코어별로 최적의 p값이 전부 다르므로 알고리즘 상에서는 10%부터 50%까지5% 간격으로 대입하여 가장 스케쥴링 결과가 좋은 값을 최종 선택하였다. 또한 Wprefer를 기준으로 모든 코어의 테스트 시간의 합을 구했을 때 특정 코어의 테스트 시간이 차지하는 비중이 상당히 큰 경우 해당 코어의 테스트 시간을 최소화시키기 위하여 할당 가능한 최대 TAM 폭을 배정하면 전체 테스트 시간을 감소시킬 수 있다. 알고리즘에서는 전체 테스트 시간 중 q%를 기준으로 해당 코어들을 선택하였고 q값은 10%부터 30%까지의 범위 내에서 5% 단위로 가변하였다. 상기 p와 q의 범위는 가장 좋은 스케쥴링 결과가 나왔을 때의 p와 q값을 통계적으로 분석하여 결정되었다.

- TAM폭의 할당

NoC 채널의 크기가 W이고 네트워크의 동작 속도가

Data Structure of a Core	
1. W_{prefer}	// Preferred TAM width for core
2. $W_{assigned}$	// TAM assigned to core
3. Begin_time	// Test start time
4. End_time	// Test end time
5. Scheduled	// Indicates core has been scheduled
6. Completed	// Indicates test for core has completed
7. Bin_pos	// TAM position that core is placed

그림 5. 코어의 데이터 구조
Fig. 5. Data Structure of a Core.

- TAM폭의 할당

코어 테스트 속도에 n배인 구조에서 진행되는 테스트 스케쥴링은 높이가 W인 상자 n개에 사각형을 채워서 n개 중 가장 긴 상자의 폭을 최소화하는 과정으로 표현할 수 있다. 각 상자별로 아직 스케쥴링 안된 코어 가운데 W_{prefer} 대비 가장 긴 테스트 시간을 갖는 코어의 사각형을 먼저 배치하고 idle time이 발생하면 이를 채우는 rectangle packing 과정은 [6]과 동일하다. 단 W_{prefer}를 결정하는 방식은 [6]의 내용과 다르며 본 논문에서 사용한 방식은 앞에서 설명하였다. 그리고 스케쥴링 시 단일 상자 내에서는 비연속적 TAM 핀 할당[10]이 가능하지만 2개 이상의 상자를 포함하는 비연속적 핀 할당은 2개 이상의 TAM을 사용하는 경우와 동일하므로 3장에서 언급한 바와 같이 현재는 적용하지 않는다. 전체 스케쥴링 과정의 의사 코드는 그림 6과 같으며 그림 7에서 W가 32이고 n이 3일 때 p22810 스케쥴링 결과 예를 나타내었다.

V. 성능 평가 및 실험 결과

모든 실험은 Sun UltraSPARC III 1.2-GHz 프로세서 시스템에서 진행되었으며 ITC'02 테스트 벤치마크[21] 중 d695, g1023, p22810, 그리고 p93791 4개의 회로를 대상으로 실험하였다. 4개의 회로 모두 계산 시간은 1초를 넘지 않았다.

실험 환경은 [14], [16]과 마찬가지로 모든 코어의 테스트 속도는 동일하다는 가정 하에서 NoC의 채널 폭이 32비트와 16비트 2가지 경우에 대하여 진행되었다. 또한 기존 방법은 모두 멀티 입출력 구조이고 제안된 방법

```
Procedure of Test Scheduling
1. Design wrapper of cores ;
2. Calculate W_prefer of cores;
3. Set W_avail[n] = W; cur_time[n] = 0; next_time[n] = 0;
   // n means # of TAM
4. until p > 50% {
5.    until q > 30% {
6.       until all cores are tested {
            Select TAM position, i, that has the lowest cur_time;
            Rectangle packing process (cur_time[i], next_time[i], i);
            // refer to [6] about rectangle packing details
         }
         If max. cur_time is less than current Best_test_time,
         Update Best_test_time;
         Increase q by 5%,
      }
      Increase p by 5%;
   }
7. return Best_test_time;
```

그림 6. 테스트 스케쥴링 과정
Fig. 6. Test Scheduling Procedure.

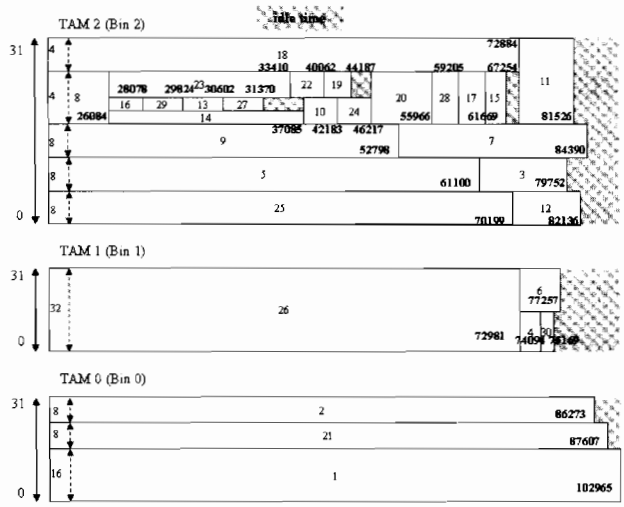


그림 7. p22810의 스케쥴링 결과 (W=32, n=3)
Fig. 7. Scheduling Result of p22810 (W=32, n=3).

은 단일 입출력 구조이므로 멀티 모드의 패턴 입출력 포트 수에 비례하여 제안된 알고리즘에서는 네트워크의 속도를 증가시켰다. 예를 들어 입출력이 각각 2/2 모드는 네트워크의 패킷 전송 속도가 코어 테스트 속도에 2배인 경우와 비교하였다.

표 1에서 제안된 알고리즘의 스케쥴링 결과와 이전 논문에서 발표된 내용을 비교하였다. 표 1의 결과를 보면 4개의 실험 회로 모두 기존 방식보다 제안된 방식이 상당히 우수함을 알 수 있다. 특히 같은 비선점 방식인 [14]의 실험 결과와 비교하여도 테스트 시간이 많이 감소되었음을 확인할 수 있다. 실험 결과 중 네트워크의 속도가 증가해도 테스트 시간이 변하지 않는 이유는 해당 벤치 회로 중 특정 코어의 테스트 시간이 최대 TAM 폭을 할당해도 더 이상 감소하지 않기 때문이다. 예를 들어 g1023의 코어 3의 테스트 시간 최소 값은 14794이고 g1023 회로의 테스트 시간은 이 값이 bottleneck이 되어 더 이상 감소할 수 없다. 그림 7에서 나타낸 p22810의 코어 1도 같은 경우이다. 회로의 크기나 채널 폭, 그리고 네트워크 속도 등 다양한 조건 하에의 실험 결과는 제안한 알고리즘이 스케쥴링 결과나 계산 시간 모든 면에서 기존 방식보다 우월함을 보여준다.

VI. 결 론

본 논문에서는 SoC 테스트 스케쥴링 알고리즘을 NoC 구조에 적용하는 새로운 형태의 NoC 테스트 스케쥴링 알고리즘을 제안했다. SoC 구조 기반 알고리즘인 rectangle packing 방식으로 NoC 테스트 스케쥴링을 하기 위해 시분할 테스트 패킷 전송과 combining 라우팅을 도입하여 코어별 할당 가능한 TAM 폭과 테스트 속도의 제한을 감소시켜 NoC의 구조적인 문제를 해결하였다. 실험 결과를 통해서 우리는 제안한 방식이 대단히 효율적이고 체계적임을 확인할 수 있었다.

참 고 문 헌

[1] L. Benini and G. D. Micheli, "Networks on Chips: A New SoC Paradigm," IEEE Computer, Vol 35, pp. 70-78, Jan. 2002.
[2] P. Guerrier and A. Greiner, "A Generic Architecture for On-Chip Packet-Switched Interconnections," Proc. DATE, pp. 250-256, Mar. 2000.
[3] B. Vermeulen, J. Dielissen, K. Goossens, and C. Ciordas, "Bringing Communication Networks on a Chip: Test and Verification Implications," IEEE

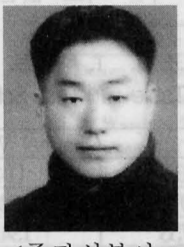
표 1. 테스트 스케쥴링 실험 결과
Table 1. Test Scheduling Results.

Circuit Name	Channel Speed / Core Speed	Channel Width = 32			Channel Width = 16	
		[16]	[14]	Proposed	[14]	Proposed
d695	2/2 or 2	26012	18869	13732	26200	23193
	3/3 or 3	20753	13412	9869	17807	16197
	4/4 or 4	14785	10705	9869	16197	12192
g1023	2/2 or 2	31898	25062	14794	28635	17798
	3/3 or 3	22648	17925	14794	19620	14794
	4/4 or 4	18851	16489	14794	17925	14794
p22810	2/2 or 2	315708	271384	138990	331672	249164
	3/3 or 3	222432	180905	102965	221134	177009
	4/4 or 4	170999	150921	102965	166800	145417
p93791	2/2 or 2	-	611991	470011	935987	932380
	3/3 or 3	-	482352	341858	734390	623715
	4/4 or 4	435787	333091	261082	502876	470011

Communications Magazine, Vol 41, pp. 74-81, Sep. 2003.
[4] M. Nahvi and A. Ivanov, "Indirect Test Architecture for SoC Testing," IEEE Trans. on CAD, Vol. 23, No. 7, pp. 1128-1142, July 2004.
[5] K. Chakrabarty, "Test Scheduling for Core-Based Systems Using Mixed-Integer Linear Programming," IEEE Trans. on CAD, pp. 1163-1174, Oct. 2000.
[6] V. Iyengar, K. Chakrabarty, and E. J. Marinissen, "On using Rectangle Packing for SOC Wrapper/TAM Co-Optimization," Proc. VTS, pp.253-258, 2002.
[7] W. Zou, S .R. Reddy, I. Pomeranz, and Y. Huang, "SOC Test Scheduling Using Simulated Annealing," Proc. VTS, pp. 325-330, April 2003.
[8] E. Larsson and Z. Peng, "A Reconfigurable Power-Conscious Core Wrapper and its Application to SOC Test Scheduling," Proc. ITC, pp. 1135-1144, Sep. 2003.
[9] E. Larsson and Z. Peng, "A Reconfigurable Power-Conscious Core Wrapper and its Application to SOC Test Scheduling," Proc. ITC, pp. 1135-1144, Sep. 2003.
[10] Y. Xia, M. Chrzanowska-Jeske, B. Wang, and M. Jeske, "Using a Distributed Rectangle Bin-Packing Approach for Core-based SoC Test Scheduling with Power Constraints," Proc. ICCAD, pp. 100-105, Nov. 2003
[11] A. Sehgal and K. Chakrabarty, "Efficient Modular Testing of SOCs Using Dual-Speed TAM Architectures," Proc. DATE, pp 422-427, Feb. 2004.
[12] J. Im, S. Chun, G. Kim, J. Ahn, and S. Kang, "RAIN(RAndom INsertion) Scheduling Algorithm for SoC Test," Proc. ATS, pp 242-247, Nov. 2004.
[13] C. Liu, V. Iyengar, J. Shi, and E. Cota, "Power-Aware Test Scheduling in Network-on-Chip Using Variable-Rate On-Chip Clocking," Proc. VTS, pp. 349-354, May 2005.
[14] C. Liu, E. Cota, H. Sharif, and D. K. Pradhan, "Test Scheduling for Network-on-Chip with BIST and Precedence Constraints," Proc. ITC, pp. 1369-1378, Oct. 2004.
[15] E. Cota, M. Kreutz, C. A. Zeferino, L. Carro, M. Lubaszewski, and A. Susin, "The Impact of NoC Reuse on the Testing of Core-based Systems," Proc. VTS, pp. 128-133, April 2003.
[16] E. Cota, L. Carro, F. Wagner, and M. Lubaszewski, "Power-Aware NoC Reuse on the Testing of Core-Based Systems," Proc. ITC, Vol. 1, pp. 612-621, Sep. 2003.
[17] A. M. Amory, E. Cota, M. Lubaszewski, F. G. Moraes, "Reducing Test Time with Processor Reuse in Network-on-Chip Based System," Proc. the 17th Symposium on Integrated Circuits and Systems Design, pp. 111-116, Sep. 2004.
[18] J. Ahn, B. I. Moon, and S. Kang, "A Practical Test Scheduling using Network-Based TAM in Network on Chip Architecture," LNCS, Vol. 3740, pp. 614-624, Oct. 2005.
[19] ARM IHI 0011A, AMBA (Rev 2) Specification, ARM Limited, 1999.
[20] J. Duato, Interconnection Networks: An Engineering Approach, Morgan Kaufmann Publishers, San Francisco, CA, USA, 2003.
[21] E. J. Marinissen, V. Iyengar and K. Chakrabarty, ITC'02 SoC Test Benchmarks, http://www.hitech-projects.com/itc02socbenchm

저 자 소 개

안 진 호 (정회원)
1995년 연세대학교 전기공학과 학사
1997년 연세대학교 전기공학과 석사
2002년 LG전자 DTV연구소 선임연구원
2006년 현재 연세대학교 전기전자공학과 박사 과정
<주관심분야 : SoC 설계 및 응용, 테스트>



김 근 배 (학생회원)
2003년 연세대학교 전기공학과 학사
2004년 연세대학교 전기전자공학과 석사
2006년 현재 연세대학교 전기전자공학과 박사 과정
<주관심분야 : On-line Test, Memory Test>

강 성 호 (평생회원)
1986년 서울대학교 제어계측공학과 학사
1988년 The University of Texas, Austin 전기 및 컴퓨터 공학과 석사
1992년 The University of Texas,Austin 전기 및 컴퓨터공학과 박사
1992년 미국 Schlumberger 연구원.
1994년 Motorola 선임 연구원
2006년 현재 연세대학교 전기전자공학과 교수
<주관심분야 : SoC 설계 및 SoC 테스트>