논문 2008-45SD-2-13

SA 기법 응용 NoC 기반 SoC 테스트 시간 감소 방법

(SA-Based Test Scheduling to Reduce the Test Time of NoC-Based SoCs)

안 진 호*, 김 홍 식**, 김 현 진**, 박 영 호***, 강 성 호**

(Jin-Ho Ahn, Hong-Sik Kim, Hyunjin Kim, Youngho Park, and Sungho Kang)

요 약

본 논문에서는 NoC 기반 SoC의 테스트 시간을 감소시키기 위하여 NoC를 TAM으로 재활용하는 구조를 바탕으로 하는 새로운 형태의 스케쥴링 알고리즘을 제안한다. 제안한 방식에서는 기존 연구된 NoC 테스트 플랫폼을 사용하여 스케쥴링 문제를 rectangle packing 문제로 변환하고 이를 simulated annealing(SA) 기법을 적용하여 향상된 스케쥴링 결과를 유도한다. ITC'02 벤치회로를 이용한 실험 결과 제안한 방법이 기존 방법에 비해 최대 2.8%까지 테스트 시간을 줄일 수 있음을 확인하였다.

Abstract

In this paper, we address a novel simulated annealing(SA)-based test scheduling method for testing network-on-chip (NoC)-based systems-on-chip (SoCs), on the assumption that the test platform proposed in [1] is installed. The proposed method efficiently mixed the rectangle packing method with SA and improved the scheduling results by locally changing the test access mechanism (TAM) widths for cores and the testing orders. Experimental results using ITC'02 benchmark circuits show that the proposed algorithm can efficiently reduce the overall test time.

Keywords: NoC, 테스트 스케쥴링, Rectangle Packing, Simulated Annealing

I. 서 론

반도체 설계 및 공정 기술의 발전으로 현재의 SoC는 가까운 미래에 수백 개의 PE (processing element)들과 SE (storage element)들이 포함된 고기능, 고밀도 형태로 진보할 것으로 예상되고 있다. 이러한 형태에서는 내장된 코어 상호간의 고신뢰, 저전력, 고효율 연결 구조가 설계 용이성, 성능, 확장성, 전력 소모, 그리고 제작 비용 측면에서 중요한 제약 및 영향을 주는 요소가 되리라는 것은 자명하다. 또한 사용되는 클럭의 종류와 수가 증가하고 길어진 wire로 인한 신호 지연과 간섭으로 시스템 클럭에 의한 전체 동기화 방식이나 클럭 skew에 대한 단순화나 무시가 거의 불가능하게 되어 전체적으로는 비동기적이나 국부적으로 동기적인 모델 설정이 일반화될 것이다. 즉 단일 타이밍 기준이 없는 상태에서 SoC는 하나의 실리콘 위에 존재하는 분산형 시스템이 되는 것이다. 이러한 변화들은 지금까지의 SoC 설계 방식으로는 도저히 해결할 수 없는 많은 문제를 야기하며, 문제의 해결을 위해 새로운 형태의 SoC template로 도입된 것 중 하나가 바로 NoC(Networks-On-Chip) 기반 구조와 플랫폼이다[2]. NoC는 하나의 SoC에 내장된 코어들을 마이크로네트워크 타입으로 연결시키는 온칩(on-chip) 구조라 정의할 수 있다[3]. 즉 기존 컴퓨터 네트워크와 유사한 레이어 기반 프로토콜을 사용하여 SoC 내부 코어들이 서로 데이터를 전송하는 구조를 의미하여 온칩네트워크(on-chip networks)라고도 한다.

일반적인 SoC와 마찬가지로 NoC 기반 SoC도 물리

* 정회원, 호서대학교 전자공학과
(Hoseo Univ., Electronic Engineering)
** 정회원, 연세대학교 전기전자공학과
(Yonsei Univ., Electrical&Electronic Engineering)
*** 정회원, 한국전자통신연구원 NoC 기술팀
(ETRI, Network Tech. Lab., NoC Tech. Team)
접수일자: 2007년12월5일, 수정완료일: 2008년1월10일

적인 결함을 검출하기 위해 테스트 과정을 거쳐야 한다. 그러나 NoC 기반 SoC는 고밀도 구조를 목표로 하는 형태이므로 기존 SoC 테스트에서 사용하는 테스트 로직을 그대로 변형없이 사용하는 것은 매우 비현실적이다. 이러한 문제를 해결하기 위해 우리는 이전 연구를 통하여 NoC를 네트워크 기반 TAM으로 재활용한 rectangle packing 방식 테스트 스케쥴링 알고리즘을 제안하였다[1]. 상기 논문에서는 내장 코어의 테스트를 위한 테스트 벡터 및 응답 입출력 구조, 테스트 패킷의 구성과 라우팅 알고리즘을 개선하여 SoC 테스트 스케쥴링 방법으로 제안되었던 기존의 rectangle packing 알고리즘[4]을 NoC 기반 SoC 테스트에 접목한 새로운 형태의 스케쥴링 알고리즘을 소개하였다. 그러나 rectangle packing 방식에서는 테스트되는 코어 별로 할당되는 초기 TAM, preferred TAM,의 폭에 따라 테스트 스케쥴링 효과가 크게 좌우된다. Preferred TAM 폭이란 최대 TAM 폭을 할당하는 경우와 유사한 테스트 효과를 제공하는 사전 계산된 TAM의 폭을 의미한다. 이에 본 논문에서는 이러한 preferred TAM 폭에 의한 스케쥴링 결과의 변화 정도를 줄이고 또한 개선시키기 위하여 새로운 simulated annealing(SA) 기반 rectangle packing 방법을 제안한다. 제안한 방법은 초기 preferred TAM 폭 및 테스트되는 코어들의 순서를 점진적으로 변화시키면서 최적화된 테스트 스케쥴링 결과를 구하는 방식이다. 결과적으로 제안한 방식은 테스트 파이프라이닝은 유지하면서 테스트 병렬성을 극대화하여 개선된 스케쥴링 효과를 얻을 수 있다.

본 논문은 다음과 같이 구성된다. 먼저 Ⅱ장에서는 NoC 기반 테스트 스케쥴링을 위해 기존에 제안되었던 방식을 설명한다. Ⅲ장에서는 제안하는 스케쥴링 알고리즘의 적용을 위해 필요한 rectangle packing 방식의 테스트 스케쥴링 방법에 대하여 소개하고 Ⅳ장에서 SA 기반의 테스트 스케쥴링 과정을 자세히 소개한다. Ⅴ장에서는 실험 환경 및 결과를 언급하며, Ⅵ장에서 결론을 내며 논문을 마무리한다.

Ⅱ. 기존 연구의 정리

NoC 기반 SoC 테스트를 위하여 먼저 테스트 데이터를 전송할 NoC에 대한 테스트가 선행되어야 한다[5]. NoC 테스트에 관련된 주요 이슈 및 방법론은 참고문헌 [6]에 소개되었다. NoC 테스트가 완료되면 비로소 NoC를 TAM 으로 활용하여 내장 코어에 대한 테스트가 시작될 수 있다. 내장 코어의 테스트 방법은 일반적으로 TAM 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 스케쥴링이란 TAM 폭과 파워 등의 주어진 제약 조건 하에서 SoC내의 모든 코어를 테스트하는데 걸리는 시간을 최소화할 수 있는 내장된 코어의 테스트 조합을 찾는 것이다. 그러나 지금까지 연구된 일반적인 SoC 테스트 스케쥴링 알고리즘을 NoC 기반 SoC 테스트에 바로 사용할 수는 없다. 가장 큰 이유는 바로 TAM 구조의 제한 때문이다. NoC를 온칩연결 구조로 하는 SoC는 버스 방식의 SoC와 달리 내부적으로 테스트를 위한 별도의 TAM을 삽입하면 하드웨어 오버헤드가 너무 커지기 때문에 NoC를 TAM으로 재활용하는 방식이 필연적이다. 따라서 테스트 패턴이나 응답 역시 네트워크상에서 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.

NoC의 네트워크를 사용하여 내장 코어에 대한 테스트를 스케쥴링하는 방법은 크게 2가지 접근 방법으로 분류할 수 있다. 첫 번째 방법은 패킷 기반 방식이다. 패킷 기반 스케쥴링은 내장 코어의 우선 순위를 참조하여 각 코어를 위한 테스트 패킷의 생성 및 전송 순서를 결정하는 것이며 제공되는 NoC의 대역폭을 최대한 사용할 수 있도록 테스트 패킷의 병렬성을 증가시키는 형태로 개발된다[7]. 테스트의 병렬성이란 동시에 테스트되는 코어의 수를 증가시키는 것을 의미한다. 코어 기반 스케쥴링은 테스트되는 코어의 순서를 결정하는 것이며 테스트가 시작된 코어는 모든 테스트가 끝날 때까지 필요한 테스트 리소스 및 테스트 패킷의 라우팅 경로에 있는 모든 NoC 구성 요소를 독점적으로 사용할 수 있는 권리가 주어진다[8~9]. 즉 테스트 스케쥴러는 코어의 테스트를 위한 패킷의 라우팅 경로를 할당하고 해당 경로에 있는 NoC 라우터의 입출력 포트 및 연결 채널을 테스트 중인 코어를 위해서 선점하고 필요한 테스트 벡

터 패킷 및 테스트 응답 패킷을 송수신한다. 코어 기반 스케쥴링은 코어의 스캔(Scan) 데이터 입력과 동시에 출력 데이터가 발생하는 동작이 연속적으로 진행되기 때문에 코어 별 테스트 시간을 최소화할 수 있다. 따라서 코어 기반 방식이 패킷 기반 방식에 비해 전체적인 테스트 시간 감소 효과가 크게 나타나고 있다. 그리고 NoC의 유휴 채널 활용도를 증가시켜 전체 테스트 시간을 줄이는 방법도 제안되었다[9]. 그러나 제안된 방식은 테스트 패킷의 연속성을 유지하기 위해 복잡한 온칩 PLL 제작이 필요한 단점을 가지고 있다.

앞서 언급한 기존 접근 방식과 달리 일반적인 SoC 테스트 스케쥴링 방법을 NoC 플랫폼에 접목시킨 새로운 방식이 [1]에서 소개되었다. 제안된 방식에서는 먼저 테스트 패킷의 구조 및 라우팅 방법 등을 포함한 NoC 기반 SoC의 테스트 플랫폼을 개발하였다. 상기 플랫폼을 사용하면 기존 SoC 테스트 스케쥴링 방식 중 하나인 rectangle packing 아이디어를 적용할 수 있다. Rectangle packing은 전형적인 NP-hard 문제로서 제한된 높이의 상자 속에 다양한 크기의 rectangle로 표현되는 item들을 상자의 폭이 최소값을 가지도록 적재하는 방법을 가리킨다. 최적값을 위해서는 세부 item들이 조합되어 적재될 수 있는 경우의 수를 모두 고려해야 하지만 item의 수가 증가할수록 경우의 수를 전부 따지는 것은 불가능하다. 따라서 일정 시간 내 구할 수 있는 값 중 가장 좋은 값을 문제의 해결안으로 채택해야 한다. 이에 일정 시간 내 탐색되고 고려할 수 있는 여러 경우의 수를 다양화하고 효과적으로 검색하는 방법이 필요하다.

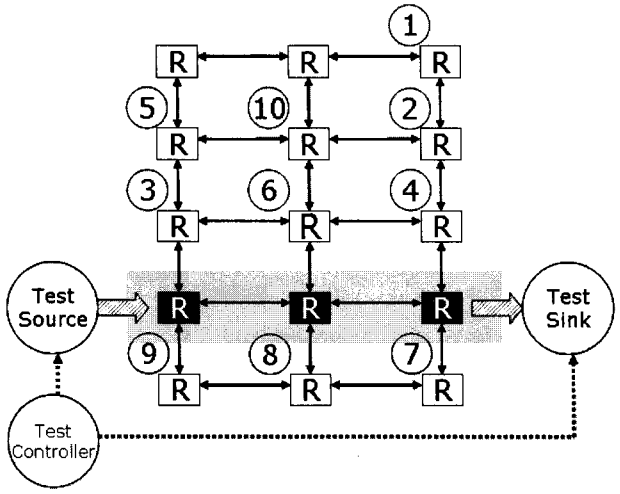


그림 1. NoC 기반 d695 회로의 테스트 구성
Fig. 1. Test Configuration of an NoC-Based d695 Circuit.

Ⅲ. Rectangle Packing 기반 테스트 스케쥴링

본 연구에서 사용되는 NoC의 주요 특징을 정리하면 다음과 같다.

- 2-D Mesh 토폴로지
- XY 라우팅(or Dimension-ordered 라우팅)
- Wormhole 스위칭
- 채널 폭: 16 또는 32비트

그림 1에서는 NoC를 기반으로 구성된 d695 벤치마트 회로의 구조 예를 보여준다. 그림 1에서 원형 내 숫자는 d695 회로의 코어 번호를 의미하며 R은 라우터를 가리킨다. 또한 라우터에 직접 연결되는 테스트 패턴 단일 입력 및 응답 출력 구조를 갖는다. 멀티 입출력 구조에서 사용되는 입출력 포트 수는 단일 입출력 구조의 테스트 패킷 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 TAM 폭의 가변성과 테스트 리소스와 CUT(Core Under Test)와의 연속성 유지 측면에서는 직렬 전송 방식이 보다 더 유리하다.

NoC 기반 SoC 스케쥴링은 [1]에서 제안한 테스트 플랫폼을 사용하면 rectangle packing 문제로 정의될 수 있다. 테스트 시간은 TAM 폭에 따라 계단형으로 변화하고 코어의 테스트는 할당된 TAM 크기가 높이를 가

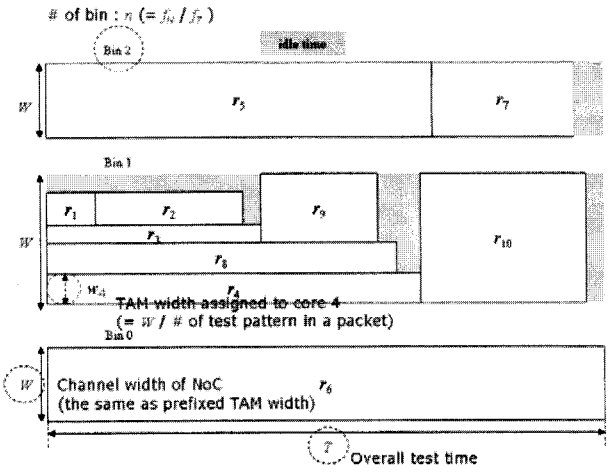


그림 2 Rectangle Packing 방식 테스트 스케쥴링 예
Fig. 2. Rectangle Packing Example.

리키고, 할당된 TAM 크기에 따라 결정되는 테스트 시간이 폭으로 표현되는 하나의 사각형으로 표현될 수 있다. 따라서 하나의 코어 테스트는 할당 가능한 TAM 크기의 전체 갯수에 해당하는 사각형의 집합에 해당한다. Rectangle packing 기반 테스트 스케쥴링이란 결국 각 코어별로 사각형의 집합 중 어느 하나를 선택하고 이것을 높이가 고정된 상자 속에 차례대로 적재하며 결과적으로 적재한 상자의 폭이 전체 SoC의 테스트 시간에 해당하므로 이 폭을 최소화시키는 것이 최종 목표인 과정으로 정의될 수 있다. 단 테스트 플랫폼 구조의 특징상 할당 가능한 TAM 폭은 K+1개, $2^K \leq W$(W: NoC 채널 폭의 크기)로 제한된다.

지금까지 설명한 rectangle packing 방식의 테스트 스케쥴링 예를 그림 2에서 나타내었다. 그림 2의 경우는 채널 폭인 W인 NoC로 구성된 어떤 SoC가 10개의 코어 $c_i(1 \leq i \leq 10)$로 이루어져 있으며 모든 코어의 테스트 속도, f_T,는 동일하고 NoC의 동작 속도, f_N,는 코어의 테스트 속도에 비해 3배 빠르다고 가정하였다. 이 문제에 대한 테스트 스케쥴링은 내장 코어 c_i의 테스트 집합 R_i, 할당 가능한 TAM 폭과 이에 대한 테스트 시간의 조합으로 구성되는 사각형의 집합, 중에서 하나의 사각형 r_i를 선택하여 높이가 W인 상자 3개, f_N과 f_T의 속도비,에 차례대로 높이가 W를 초과하지 않도록 적재하고 모든 코어를 삽입한 이후 3개의 상자 중 가장 폭이 긴 상자의 길이를 구하는 것이다. 따라서 그림 2에서 T가 결국 최종적으로 계산된 상기 SoC의 테스트 시간이다.

Ⅳ. SA 기반 테스트 스케쥴링 알고리즘

Simulated annealing은 널리 알려진 approximation 알고리즘으로서 초기 해(initial solution)를 중심으로 좀 더 향상된 결과를 얻기 위해 주변 해를 점진적으로 검색하며 local optimal한 해를 구하는 방식이다. 비교적 구현이 쉬운 반면 초기 해의 성능에 따라 최종 결과의 우수성이 결정되는 문제가 있다. SA 알고리즘은 초기 해를 결정하는 것에서 시작한다. 초기 해를 임의로 설정할 경우 전체 해집합의 검색 범위를 크게 증가시켜야 하기 때문에 알고리즘의 계산 시간이 상당히 증가한다. 따라서 일반적으로 초기 해는 선험적인 방법이나 사전에 검증된 해를 중심으로 결정한다. 초기 해가 결정되면 먼저 이 값을 현재 해((current best solution)로 설정한다. 그리고 현재 해를 중심으로 하는 이웃 해(neighboring solution)를 결정한다. 만약 이웃 해가 현재 해보다 더 좋은 결과를 보여준다면 상기 이웃 해가 새로운 현재 해로 설정된다. 만약 그 반대로 이웃 해가 현재 해보다 좋지 않다면 상기 이웃 해는 확률적으로 현재 해를 대체하게 된다. 확률 값은 다음과 같이 주어진다.

$$p = e^{-(\Delta C/T)} \tag{1}$$

확률 값 계산식 (1)에서 ΔC는 현재 해와 이웃 해의 비용 차이(cost difference)를 의미하며 T는 현재 온도를 의미한다. 온도 파라미터는 이웃 해를 현재 해로 설정하는 비율을 제어하는 역할을 한다. 즉 온도가 높으면 높을수록 이웃 해가 현재 해가 될 확률은 증가하는 것이다. SA 알고리즘에서는 초기 온도 파라미터를 설정하고 알고리즘이 진행되는 동안 점차로 온도를 낮추는 방식을 취하는데 이러한 이유는 알고리즘의 초기에는 이웃 해를 보다 자주 현재 해로 설정하도록 하여 해 집합 내 검색 범위를 보다 충분히 확보하고 알고리즘의 반복 횟수가 증가할수록 검색 범위 내 최적값으로 수렴할 수 있도록 하려는 것이다.

본 연구에서는 전체 테스트 스케쥴링 과정 중에서 코어 별 테스트 집합 R_i에서 r_i를 선택할 때 SA 방식을 적용하였다. 전체 테스트 스케쥴링 과정을 SA로 진행할 경우 계산 시간 상 알고리즘의 반복 횟수를 크게 제한할 수 밖에 없고 결과적으로 스케쥴링 결과가 만족스럽지 못했기 때문이다. 그리고 preferred TAM 폭을 이용한 rectangle packing 결과를 초기 해 S_{init}로 결정하고 이를 기준으로 이웃 해를 선정하였다. 사용한 cost function은 코어의 테스트 시간이다. 현재 해를 기준으로 이웃 해를 도출하는 과정 또한 알고리즘의 성능을 좌우하는 중요 요소이다. 이웃 해를 구하는 방법은 문헌마다 그리고 적용하고자 하는 분야마다 매우 다양하고 대부분 선험적인 방식을 이용한다. 본 연구에서 사용한 방식은 참고문헌 [10]에서 사용한 방법을 기본으로 하여 rectangle packing 방식에 맞게 변형하였고 일부 조건을 추가하였다. 이웃 해 S_n은 현재 해 S_{cur}의 임의 변경으로 생성하였으며 다음과 같은 네 가지 변경 방식을 사용하였다.

A. 가장 폭이 긴 상자를 선택하여 그 상자에 담겨있는 사각형 중 하나를 골라서 그 높이와 폭을 변경한다

B. 전체 상자 중 두 개의 상자를 선택하여 각 상자에서 하나의 사각형을 임의로 고르고 그 두 개의 사각형을 상호 교환한다

C. 전체 상자 중 임의로 하나를 선택하여 그 상자에 담겨있는 사각형 중 하나를 골라서 그 높이와 폭을 변경한다

D. 전체 상자 중 임의로 하나를 선택하여 그 상자에 담겨있는 사각형 중 하나를 골라서 그 클럭 모드를 변경한다

상기 4가지 방식은 알고리즘 진행 과정에서 임의로 선정되는데 전체적인 선택 확률은 균일하게 하였다. 그리고 상기 방식으로 선정된 이웃 해 S_n을 이용하여 테스트 스케쥴링 결과 $T(S_n)$를 구할 때는 사용한 이웃 해 유도 방식에 따라 달리 하는데 A와 B 방식으로 이웃 해를 구한 경우 수정되거나 교환된 사각형을 가지고 있는 상자에 대해서만 스케쥴링을 다시 하였고 C방식의 경우에는 수정된 사각형을 가지고 모든 상자에 대해서 다시 스케쥴링하여 그 결과를 비교하였다. 그 이유는 C 방식과 A방식을 차별화시키기 위함이다. 그리고 D 방식은 단일 클럭 모드가 아닌 멀티 클럭 모드 1에서 코어의 클럭 모드를 변경시키는 경우에만 적용하였고 C 방식과 마찬가지로 전체 상자에 대하여 다시 스케쥴링하였다. 멀티 클럭 모드 1은 코어의 테스트 클럭을 x1 또는 x2 모드로 고정하는 모드이다. 멀티 클럭 모드 2에서 사용하지 않은 이유는 rectangle packing 과정에서 클럭 모드가 적응적으로 변화하기 때문에 강제 설정한 클럭 모드가 유지되지 않아 클럭 모드 변경의 실효성이 없기 때문이다. 멀티 클럭 모드 및 종류에 대한 자세한 설명은 참고문헌 [1]을 참조바란다.

1. Build a initial test schedule(S_{init}) by the rectangle packing method proposed in [1].
2. Set the initial temperature(T_{init}), the number of iteration(N_{iter}), stopping temperature(T_{stop}), and temperature reduction ratio(K).
3. Let the current solution be $S_{cur} = S_{init}$ and the current temperature be $T = T_{init}$
4. While (T is higher than T_{stop}) {
   For i from 1 to N_{iter} loop {
     Generate a neighboring solution(S_n)
     Get a test schedule by S_n, $T(S_n)$
     Compute ΔC, the difference between $T(S_n)$ and $T(S_{cur})$
     If $\Delta C < 0$ then $S_{cur} = S_n$
     Else {
      Generate one random value q, $0 < q < 1$
      If $q < \exp(-\Delta C/T)$ then $S_{cur} = S_n$
     }
   }
   Set new temperature $T = K*T$;
 }

그림 3. SA 기반 테스트 스케쥴링 과정
Fig. 3. SA-Based Test Scheduling Procedure.

SA 기반 테스트 스케쥴링의 전체 과정은 그림 3에서 나타내었다. 그림 3에서 사용된 주요 파라미터에 대한 정의는 다음과 같다.

- T_{init}: 온도 파라미터의 초기 값
- N_{iter}: 현재 온도 기준 이웃 해 구하는 횟수
- T_{stop}: 온도 파라미터의 최종 값
- K: 온도 파라미터의 감쇠 비율

V. 성능 평가 및 실험 결과

제안한 알고리즘은 C 시뮬레이션을 통하여 검증하였으며 모든 실험은 Sun UltraSPARC III 1.2-GHz 프로세서 시스템에서 진행되었다. 검증을 위해서 사용한 회로는 ITC'02 테스트 벤치마크 중 d695, g1023, p22810, 그리고 p93791 4개의 회로를 대상으로 하였다. 최종 결과 도출을 위해 사용한 SA의 주요 파라미터 값은 다음과 같다.

- T_{init}: 4000
- N_{iter}: $100*N_{core}$(내장 코어의 총 수)
- T_{stop}: 0.01
- K: 0.98

상기 값은 참고문헌 [10]에서 사용한 값을 참고하여 계산 시간과 최종 결과의 성능을 고려하여 결정하였다.

실험 조건은 모든 코어의 테스트 속도는 동일하다는 가정 하에서 NoC의 채널 폭이 32비트와 16비트 2가지 경우에 대하여 진행되었다. 단일 클럭 모드에서의 실험 결과는 표 1에서 나타내었다. 표 1 중 첫 번째 열은 벤치마크 회로의 이름, 두 번째 열은 테스트 환경과 관련된 내용으로서 n은 NoC 동작 속도와 코어 테스트 속도의 차를 가리킨다. 즉 $n = f_N/f_T$이며 n은 정수 값이다. 모든 실험 결과는 [1]의 결과와 비교하여 그 성능을 평가하였다. 표 1에서 best value는 SA 스케쥴러가 생성한 local optimal value이며, converged value는 스케쥴러가 선택한 최종 값이다. Local optimal value가 converged value와 항상 일치하지 않는 이유는 SA 스케쥴러는 온도 파라미터 T와 반복 파라미터 N_{iter}에 의하여 사전 정해진 횟수를 반복해야 하므로 중간에 구한 local optimal value가 열화된 값으로 대체되는 경우가 발생하기 때문이다.

표 1. 단일 클럭 모드에서의 테스트 스케쥴링 실험 결과
Table 1. Test Scheduling Results in Single Test Clock Mode.

SoC Name	n	W = 32				W = 16			
		[1]	SA		Reduction Ratio	[1]	SA		Reduction Ratio
			Converged	Best			Converged	Best	
d695	2	13732	12192	11978	12.8	23193	25473	21437	7.6
	3	9869	9869	9869	0.0	16197	16197	15124	6.6
	4	9869	9869	9869	0.0	12192	12192	12192	0.0
g1023	2	14794	14794	14794	0.0	17798	18574	17798	0.0
	3	14794	14794	14794	0.0	14794	14832	14794	0.0
	4	14794	14794	14794	0.0	14794	14794	14794	0.0
p22810	2	138990	137460	132111	4.9	249164	248382	248382	0.3
	3	102965	102965	102965	0.0	177009	173705	166184	6.1
	4	102965	102965	102965	0.0	145417	147389	145417	0.0
p93791	2	470011	457228	455918	3.0	932380	895255	890035	4.5
	3	341858	341858	341858	0.0	623715	602361	599301	3.9
	4	259263	258822	258822	0.2	470011	477191	457614	2.6

표 2. 멀티 클럭 모드 1에서의 테스트 스케쥴링 실험 결과
Table 2. Test Scheduling Results in Multi-Test Clock Mode 1.

SoC Name	n	W = 32				W = 16			
		[1]	SA		Reduction Ratio	[1]	SA		Reduction Ratio
			Converged	Best			Converged	Best	
d695	2	11978	12192	11978	0.0	21571	23488	21223	1.6
	3	9869	11978	9869	0.0	15196	15100	14918	1.8
	4	9869	9869	9869	0.0	12192	12192	12192	0.0
g1023	2	9822	9822	9677	1.5	17461	16486	16174	7.4
	3	7550	7550	7477	1.0	11462	11784	11451	0.1
	4	7397	7397	7397	0.0	10248	10248	10248	0.0
p22810	2	127480	130082	122997	3.5	248382	248382	248382	0.0
	3	93102	86852	86852	6.7	170999	173705	164268	3.9
	4	72981	72981	72981	0.0	145417	145417	145417	0.0
p93791	2	470011	455934	455918	3.0	932380	895255	890035	4.5
	3	341858	341858	338827	0.9	623715	602361	599301	3.9
	4	259263	271050	258822	0.2	470011	477191	457614	2.6

표 3. 멀티 클럭 모드 2에서의 테스트 스케쥴링 실험 결과
Table 3. Test Scheduling Results in Multi-Test Clock Mode 2.

SoC Name	n	W = 32				W = 16			
		[1]	SA		Reduction Ratio	[1]	SA		Reduction Ratio
			Converged	Best			Converged	Best	
d695	2	11013	12081	10718	2.7	21460	23370	20951	2.4
	3	8082	7586	7562	6.4	15100	15472	14757	2.3
	4	6096	6096	6096	0.0	12081	12081	12081	0.0
g1023	2	8899	9735	8899	0.0	17056	16328	15818	7.3
	3	7397	7416	7397	0.0	11756	11270	11270	4.1
	4	7397	7397	7397	0.0	9735	9735	9735	0.0
p22810	2	124463	124581	124190	0.2	227127	220143	219392	3.4
	3	91409	86852	83145	9.0	152873	148270	147115	3.8
	4	72981	72708	72708	0.4	139320	139320	139320	0.0
p93791	2	457274	445942	445358	2.6	912193	888057	881008	3.4
	3	328248	307072	299913	8.6	608504	589313	588407	3.3
	4	238078	232633	231189	2.9	468846	487140	449540	4.1

표 1의 결과를 보면 4개의 실험 회로 모두 기존 방식보다 제안된 방식이 상당히 우수함을 알 수 있다. W=32일 때는 평균 테스트 시간 감소 비율은 1.7%, W=16에서는 2.6% 정도에 이르렀다. g1023의 경우 코어 중 하나의 테스트 시간이 전체 시스템 테스트 시간을 결정하는 구조이기 때문에 추가 감소의 여지가 없어 성

능 개선이 이루어지지 않았다. 여기서 감소 비율은 비교 대상인 [1]의 결과와 SA의 best value 사이의 차이를 기준으로 계산되었다. NoC 채널 폭이 16일 때 감소 비율이 증가한 이유는 코어 테스트 집합 R_i가 더 작기 때문에 그만큼 해집합 검색 범위가 감소하기 때문이다.

표 2와 3에서는 멀티 클럭 모드에서의 스케줄링 결과를 보여준다. 본 연구에서 사용한 멀티 클럭은 f_T와 $2f_T$ 2가지 경우이고 멀티 클럭 모드 1에서는 테스트 시간이 큰 코어에는 $2f_T$, 그 외의 것은 f_T를 사전에 결정하여 할당하는 모드이고 멀티 클럭 모드 2는 모든 코어는 기본적으로 f_T로 시작하고 rectangle packing 과정에서 적응적으로 일부 코어에 $2f_T$를 할당하는 방식이다. 표 2와 3의 결과를 분석하면 W=32인 경우 평균 테스트 시간은 각각 1.4%와 2.7%, W=16인 경우에는 2.2%와 2.8% 정도 감소하였다. 결과적으로 회로의 크기나 채널 폭, 그리고 네트워크 속도 등 다양한 조건 하에서 SA 기반 알고리즘의 스케줄링 성능이 우수하였다. 그러나 계산 시간 측면을 보면 기존 방식은 단 한 번의 스케줄링으로 끝나기 때문에 모든 회로에서 수 초만에 최종 결과가 도출되었지만 점진적으로 결과를 개선하는 SA 방식은 반복 계산으로 인하여 d695나 g1023의 경우 10-30초 정도, p22810이나 p93791과 같이 내장 코어의 수가 많은 회로에서는 최대 5분 정도의 시간이 소요되었다.

VI. 결 론

본 논문에서는 SA 기법을 응용한 새로운 형태의 NoC 기반 SoC 테스트 스케줄링 알고리즘을 제안하였다. 제안된 방식에서는 먼저 테스트 스케줄링 문제를 rectangle packing 문제로 변환하고 이를 SA 알고리즘을 사용하여 해집합의 검색 범위를 확장, 보다 최적화된 스케줄링 결과를 유도하였다. 실험 결과를 통해서 우리는 제안한 방식이 다양한 조건 하에서 대단히 효율적이고 체계적임을 확인할 수 있었다.

참 고 문 헌

[1] J.-H. Ahn and S. Kang, "Test Scheduling of NoC-based SoCs Using Multiple Test Clocks," ETRI Journal, Vol. 28, No. 4, pp. 475-485, Aug. 2006.

[2] L. Benini and G. D. Micheli, "Networks on Chips: A New SoC Paradigm," IEEE Computer, Vol. 35, pp. 70-78, Jan. 2002.

[3] A. Ivanov and G. D. Micheli, "The Network-on-Chip Paradigm in Practice and Research," IEEE Design&Test of Computers, pp. 399-403, Sep.-Oct. 2005.

[4] V. Iyengar, K. Chakrabarty, and E. J. Marinissen, "On Using Rectangle Packing for SOC Wrapper/TAM Co-Optimization," Proc. VTS, pp.253-258, 2002.

[5] B. Vermeulen, J. Dielissen, K. Goossens, and C. Ciordas, "Bringing Communication Networks on a Chip: Test and Verification Implications," IEEE Communications Magazine, Vol. 41, pp. 74-81, Sep. 2003.

[6] P. P. Pande, G. D. Micheli, C. Grecu, A. Ivanov, and R. Saleh, "Design, Synthesis, and Test of Networks on Chips," IEEE Design&Test of Computers, pp. 404-413, Sep.-Oct. 2005.

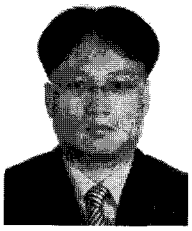
[7] E. Cota, L. Carro, F. Wagner, and M. Lubaszewski, "Power-Aware NoC Reuse on the Testing of Core-Based Systems," Proc. ITC, Vol. 1, pp. 612-621, Sep. 2003.

[8] 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.

[9] 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.

[10] W. Zou, S. M. Reddy, I. Pomeranz, and Y. Huang, "SOC Test Scheduling Using Simulated Annealing," Proc. VTS, pp. 325-330, 2003.

저 자 소 개

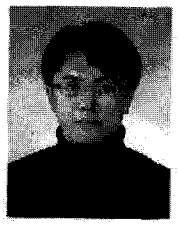


안 진 호(정회원)

1995년 연세대학교 전기공학과 학사 졸업.
1997년 연세대학교 전기공학과 석사 졸업.
2002년 LG전자 DTV연구소 주임연구원.
2006년 연세대학교 전기전자공학과 박사 졸업.
2008년 현재 호서대학교 전자공학과 전임강사.
<주관심분야 : SoC 설계 및 응용, 테스트>

김 현 진(학생회원)

1997년 연세대학교 전기공학과 학사 졸업.
1999년 연세대학교 전기 및 컴퓨터공학과 석사 졸업.
2005년 삼성전기 중앙연구소 선임연구원
2008년 현재 연세대학교 전기전자공학과 박사과정
<주관심분야 : SoC 설계 및 응용, CAD>



김 홍 식(정회원)

1997년 연세대학교 전기공학과 학사 졸업.
1999년 연세대학교 전기 및 컴퓨터공학과 석사 졸업.
2004년 연세대학교 전기전자 공학과 박사 졸업.
2005년 Virginia 공대 박사후 연구원.
2006년 삼성전자 시스템 LSI 사업부 책임연구원
2008년 현재 연세대학교 TMS 사업단 연구교수.
<주관심분야 : SoC 설계 및 테스트>

박 영 호(정회원)

1985년 대전산업대학교 전자계산학과 졸업.
2008년 현재 한국전자통신연구원 NoC 기술팀 책임연구원.
<주관심분야 : SoC 설계, 컴퓨터 네트워크>

강 성 호(평생회원)

1986년 서울대학교 제어계측공학과 학사 졸업.
1988년 The University of Texas, Austin 전기 및 컴퓨터 공학과 석사 졸업.
1992년 The University of Texas, Austin 전기 및 컴퓨터 공학과 박사 졸업.
1992년 미국 Schlumberger 연구원.
1994년 Motorola 선임 연구원.
2008년 현재 연세대학교 전기전자공학과 교수.
<주관심분야 : SoC 설계 및 SoC 테스트 >