코아 테스트 스케듈링에 관한 연구

최동춘[0], 민형복, 김인수
성균관 대학교
{dcchoi[0], hbmin, iskim}@stella.skku.ac.kr

A Study of Core Test Scheduling for SOC

Dongchoon Choi[0], Hyoung Bok Min, Insoo Kim
Sungkyunkwan University

요 약

본 논문은 SOC 내에 존재하는 코아들을 테스트하는 과정에서 개별 코아들의 테스트 조건을 기반으로 한 스케듈링을 통해 최적의 Testing time을 구하는 연구이다. SOC 내에 존재하는 코아들은 주어지는 TAM(Test Access Mechanism) Width에 따라 각 코아들의 Width가 달라지고, 최대 Width에서 최소 Width(1)까지 각 Width 별로 Testing time을 계산할 수 있다. 코아들의 각 Width 별 Testing time을 기존의 Rectangle Packing Algorithm을 수정, 보완하여 효율적으로 구성한 수정 Rectangle Packing Algorithm에 적용하여 최적의 Testing time을 구하는 것이 본 논문의 목적이다.

1. 서 론

코아 테스트는 SOC(System-on-Chip) 내에 존재하는 IP 코아들을 테스트하는 일련의 방법들을 말한다. 코아 테스트에 있어서 우리가 얻고자 하는 것들은 Testing time 감소, Area 감소, Power Consumption 등이다.

코아를 테스트하기 위한 테스트 설계의 기본 구조를 살펴보면, 크게 세 가지로 분류할 수 있다. Test Wrapper, TAM(Test Access Mechanism), Test Control Mechanism 이 세 가지인데, 이 중에서 Test Wrapper와 TAM은 Testing time과 깊은 관련이 있고 이것이 최근의 주된 연구 주제가 되고 있다[1][2][3][4].

또한, Testing time의 감소를 중심으로 한 주제인 테스트 스케듈링에 대한 연구도 활발히 진행되고 있다. SOC 내에 존재하는 코아들을 하나의 작은 Rectangle로 표현하고, 이를 주어진 조건 내에서 최적으로 정렬하여 Testing time을 산출해내는 방법이 최근 연구되고 있다. 이는 Rectangle Packing Algorithm[3][4]이라는 테스트 스케듈링 알고리즘으로 테스트 스케듈링에 관련된 본 논문의 기반이 되고 있는 스케듈링 알고리즘이다.

Rectangle Packing Algorithm을 기반으로 테스트 스케듈링 알고리즘을 구현한 기존 논문을 수정하고, 새로운 개념과 함수를 추가하여 기존보다 효율적인 알고리즘을 제안할 것이다.

2. 코아 테스트

코아 테스트는 SOC에 내재되어 있는 독립적인 개체들을 테스트하기 위한 일련의 방법들을 말한다. 코아 테스트를 진행하는데 있어서 다음의 Testing time, Area, Power Consumption 등의 요소들은 매우 중요하다. 이 요소들 중 관심을 끄는 분야는 Testing time 감소에 관한 연구이다.

코아를 테스트하기 위한 테스트 설계의 기본 구조를 살펴보면 크게 세 가지(Test Wrapper, TAM, Test Control Mechanism)로 나눌 수 있다[5][6].

본 논문은 Testing time 감소를 목적으로 하기 때문에 Test Control Mechanism에 대해서는 언급하지 않을 것이다.

코아 테스트에 대해 알아야 할 기본적인 구조들을 다음에 간략하게 설명하였다.

- Test Wrapper : Test Wrapper[1][2][5][6]는 코아를 감싸며 TAM과 테스트하고자하는 코아 사이의 Interface 역할을 하는 Logic이다. 주로 Test mode 설정과 Test Width 조정 등 테스트 환경의 변화에 맞게 테스트를 원활히 진행되도록 보조한다.
- Test Control Mechanism : Test Control Mechanism[1][2][5][6]은 전반적인 Test Operation에 관련하여 필요한 Control Signal들을 생산하여 효율적인 Test Flow를 유지하는 일을 한다.
- Test Access Mechanism : TAM[1][2][5][6]은 SOC의 Primary I/O와 Wrapper 사이에 존재하며, Test Data(Test Vector, Response, Control Signal etc.)들의 이동을 책임지고 있다.

3. 기존 Rectangle Packing Algorithm

3장에서는 기존 Rectangle Packing Algorithm의 특징에 대한 설명과 기존 알고리즘의 취약점에 대해 간략하게 언급하도록 하겠다.

3.1. 기존 Rectangle Packing Algorithm의 특징

SOC 내의 각 코아들을 가로는 Testing time, 세로는 TAM Width의 조건으로 직사각형의 모양을 만들어서 주어진 Maximum Width 내에서 최적 정렬하여 최소의 Testing time을 얻어내는 방법이 Rectangle Packing Algorithm이다[1][2]. 기존 논문의 스케듈링 알고리즘을 이후부터 Rectangle Packing Algorithm이라 하겠다.

$1 \le i \le N$, R_i 개의 코아들이 있고, Maximum Width(W)는 64로 가정한다. 코아들은 각 Width별, $1 \le Width \le 64$, 로 Testing time을 가진다. 이후부터 이를 TTT(Test Time Table)라 부르도록 하겠다. TTT(Ti(i))을 Initialize Function[2]에 적용하면 각 코아별로 최적 Width(Wp(i))를 얻을 수 있고, 이것으로 각 코아별 최적 Testing time(T(Wp(i))를 구하게 된다. 이 데이터를 Rectangle Packing Algorithm에 적용하여 SOC 전체의 최적 Testing time을 구한다.

[그림 1]은 SOC 내에 존재하는 총 8개의 코아들이 Rectangle Packing Algorithm에 따라 정렬된 모습을 보여주고 있다. 여기서 idle_time은 조건에 맞는 코아가 없어서 버려진 공간을 말한다.

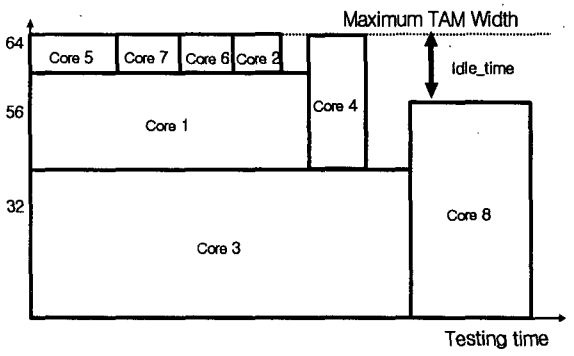

그림 1. Rectangle Packing Algorithm의 예제

3.2. 기존 알고리즘의 취약점

각 코아별로 TTT(Test Time Table)을 작성하고, 이를 Initialize Function을 거쳐 각 코아별 최적 Width(Wp)를 구하고, Rectangle Packing Algorithm에 적용하였다. 여기서 Initialize Function은 실험에 의한 값을 가지고 각 코아별로 TTT의 모든 값들을 비교하여 Wp를 구하는 과정이다.

TTT를 막대그래프로 표현해보면 계단과 같은 모양을 하고 있다. 즉, 작은 Width의 감소로 Testing time이 크게 증가하는 상황이 발생할 수 있다.

코아 스케듈링 알고리즘에 세부적으로 정렬 조건을 둔다면 더욱 효과적인 스케듈링 알고리즘이 될 것이다. 또한, 코아들을 내부 스캔 체인의 존재 여부에 따라 두 그룹으로 분류하여 적용 방법을 새로운 정렬 조건으로 하여 수정 스케듈링 알고리즘 구현에 사용할 것이다.

자세한 내용은 다음 장에서 설명하도록 하겠다.

4. 수정 Rectangle Packing Algorithm

3장에서 Rectangle Packing Algorithm에 대한 취약점들에 대해 언급하였다.

이번 장에서는 3장에서 언급한 취약점들을 수정하고 새로운 방법들을 추가하여 수정 Rectangle Packing Algorithm을 제안할 것이다.

4.1. Initialize Function의 교체

Initialize Function을 통해 구해진 각 코아별 최적 Width(Wp)는 Width의 감소에 따른 Testing time 증가라는 단점이 있다. 본 논문에서는 Initialize Function으로 인하여 발생할 수 있는 Testing time 증가에 관한 문제를 해결하기 위해 가장 작은 Testing time을 갖는 Width를 최적 Width(Wp)로 선택하는 Optimal_Width Function을 새로 작성하여 Initialize Function과 교체할 것을 제안한다.

[그림 2]를 보면, TAM Width가 44-64까지 주어질 경우는 Wp가 44'가 되고, Wp가 24-43까지 주어질 경우에는 Wp가 '24'가 된다.

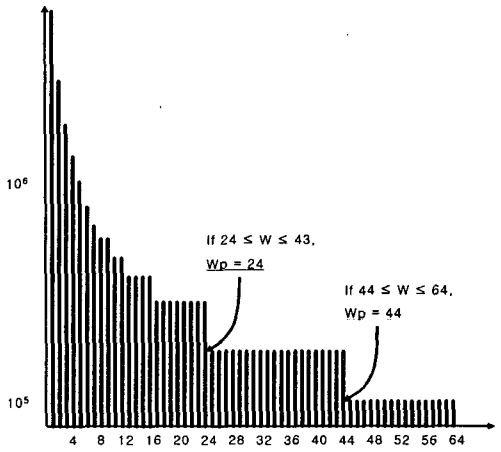

그림 2. SOC p93791 코아 6의 Test Time Table을 기반으로 한 그래프

Function의 교체와 다음 절에서 새롭게 제안하게 될 정렬 조건의 수정 및 추가가 서로 조화를 이루어서 Testing time을 감소하는데 큰 역할을 할 것이다.

4.2.정렬 조건의 수정 및 추가

기존 논문의 Rectangle Packing Algorithm에서는 코아들의 정렬 조건에 대한 부분이 크게 세 부분으로 나누어져 있다. 이 세 가지 조건을 제외하고는 정렬 조건들을 찾아보기 힘들다.

새롭게 제안하는 정렬 조건은 SOC 내의 전체 코아들을 내부 스캔 체인의 존재 여부에 따라 두 그룹으로 분류하여 사용하는 것이다. 여기에 기존 정렬 조건과의 차별성은 Width를 중심으로 코아가 선택, 정렬된다는 것이다.

기존의 알고리즘에서는 코아의 최적 Width를 중심으로 정렬이 이루어졌지만 새롭게 제안하는 알고리즘은 avail_width(코아 정렬에 관련된 Width 변수)를 중심으로 코아가 선택되고 정렬되어진다는 것이다. avail_width의 변화에 따라 적합한 코아들을 선택하는데 처음 조건은 내부 스캔 체인의 존재 여부에 따라 결정된다. 내부 스캔 체인이 존재하는 코아의 경우 Wp가 avail_width보다 클 경우에 avail_width를 Wp로 재 선택하여 다른 코아들과 경쟁하여 정렬 순서와 위치가 결정된다.

avail_width가 '0'이 될 때까지 이러한 과정이 반복된다. 여기서 내부 스캔 체인이 없는 코아들은 avail_width를 Wp로 한 Testing time을 선택하여 경쟁하게 된다.

여기에 제한 조건이 한 가지 추가되는데, 내부 스캔 체인이 존재하는 코아의 경우 avail_width가 '1'일 경우 기존의 Wp를 그대로 가지고 경쟁을 한다. 내부 스캔 체인이 존재하는 경우 Width가 '1'이 되면 Testing time이 극단적으로 높아지기 때문에 이를 피하도록 한다. avail_time이 '0'이 되면 next_time이

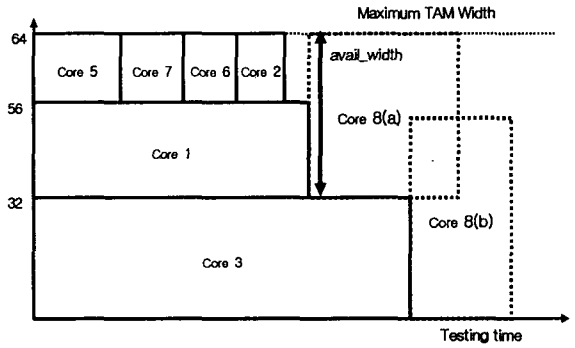

그림 3. 수정 Rectangle Packing Algorithm의 예제

라는 조건 변수를 기준으로 코아들을 다시 정렬하게 된다. 마지막으로 정렬하는 코아는 Wp가 avail_time보다 작을 경우 avail_time을 모두 사용하여 최적의 Testing time을 얻도록 정렬한다.

[그림 3]에서 코아 8의 (a)는 수정 알고리즘에 의한 것이고, (b)는 기존 알고리즘에 의해 정렬된 것이다. [그림 1]과 [그림 3]을 비교해보면 기존 알고리즘에서는 avail_width에 알맞게 정렬 가능한 코아 4를 선택하였고, 수정 알고리즘에서는 코아 8을 정렬 조건에 따라 먼저 선택하였다. 코아 8을 먼저 선택함으로써, 전체 Testing time이 감소하였다.

5. 실험 결과

본 실험은 수정 Rectangle Packing Algorithm의 증명을 위해 기존 논문에서 사용했던 ITC'02 SOC Test Benchmarks 중에서 p34392 그리고 p93791 두 가지 SOC를 기초로 하였다.

실험 환경은 Sun O.S 5.7, UltraSparc II 450Mhz, Memory 4G인 Unix Machine에서 실행되었다.

감사의 글

본 연구는 IDEC으로부터 CAD Tool을 지원받았으며, 한국과학재단 목적기초연구[R01-2000-000-00247-0(2002)]에 따른 연구비에 의하여 수행되었다.

표 1. SOC p34392와 p93791의 실험 결과

width	SOC p34392			SOC p93791		
	기존 방법 (cc)	새로운 방법 (cc)	T(%)	기존 방법 (cc)	새로운 방법 (cc)	T(%)
16	1023820	975741	-4.70	1851135	2033744	+9.86
24	759427	658734	-13.26	1248795	1310480	+4.94
32	544595	543105	-0.27	975016	977693	+0.27
40	544595	543105	-0.27	794020	768110	-3.26
48	544595	543105	-0.27	627934	645260	+2.76
56	544595	543105	-0.27	568436	565836	-0.46
64	544595	543105	-0.27	511286	489043	-4.35

참고 문헌

[1] Iyengar, V., Chakrabarty, K., Marinissen, E.J., "Test Wrapper and Test Access Mechanism Co-Optimization for System-on-Chip", Test Conference, pp. 1023-1032, 2001.

[2] Iyengar, V., Chakrabarty, K., Marinissen, E.J., "On using Rectangle Packing for SOC wrapper/TAM Co-Optimization", VLSI Test Symposium, 2002. (VTS 2002). Proceedings 20th IEEE, pp. 253-258, 2002.

[3] Y.Zorian, E. J. Marinissen and S. Dey., "Testing embedded-core-based system chips", Proc. Int. Test Conf., pp. 130-143, 1998.

[4] Vikram Iyengar, Krishnendu Chakrabarty and Erik Jan Marinissen., "Efficient Wrapper/TAM Co-Optimization for Large SOCs.", Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings, pp. 491-498, 2002.

[5] P1500 Scalable Architecture Task Force Members, "Preliminary Outline of the IEEE P1500 Scalable Architecture for Testing Embedded Cores", IEEE P1500 General Working Group.

[6] IEEE P1500 General Working Group Website, "IEEE P1500 Standards For Embedded Core Test", http://grouper.ieee.org/groups/1500/

[7] M. Abramovici, M. A. Breuer and D. Friedman, "Digital Systems Testing and Testable Design", Computer Science Press, 1990.

[8] Alexander Miczo, "Digital Logic Testing and Simulation", John Wiley & Sons, 1986

[9] Eichelberger, E. B., T. W. Williams, " A Logic Design Structure for LSI Testability", Proc. 14th Design Automation Conf., pp. 462-468, June 1977.