論文2003-40SD-1-8

SoC IP 간의 효과적인 연결 테스트를 위한 알고리듬 개발

(A New Test Algorithm for Effective Interconnect Testing Among SoC IPs)

金 容 準 *, 姜 成 昊 *

(YongJoon Kim and Sungho Kang)

요 약

본 논문에서 제안하는 GNS 시퀀스는 SoC 연결 고장 테스트를 수행할 때 aliasing 고장 증후와 confounding 고장 증후를 고장 증후를 발생시키지 않는 시퀀스로 연결 고장 위치의 분석을 효과적으로 수행할 수 있다. GNS 시퀀스는 과거 보드 수준의 연결 테스트를 수행하기 위한 IEEE 1149.1 std. 와 유사한 구조로 SoC 의 연결 테스트를 수행하게 되어있는 IEEE P1500 에 적용하여 SoC 내부의 IP 상호간에 존재하는 연결 고장을 검출하고 그 위치를 분석하는데, 이때 입력되는 테스트 시퀀스의 길이가 기존 연구들에 비해 최소의 값을 가짐으로써 연결 테스트 수행 시간을 단축할 수 있는 효과적인 연결 테스트 알고리듬이다.

Abstract

Interconnect test for highly integrated environments like SoC, becomes more important as the complexity of a circuit increases. This importance is from two facts, test time and complete diagnosis. Since the interconnect test between IP's is based on the scan technology such as IEEE1149.1 and IEEE P1500, it takes long test time to apply test vectors serially through a long scan chain. Complete diagnosis is another important issue because a defect on interconnects are shown as a defect on a chip. But generally, interconnect test algorithms that need the short test time can not do complete diagnosis and algorithms that perform complete diagnosis need long test time. A new interconnect test algorithm is developed. The new algorithm can provide a complete diagnosis for all faults with shorter test length compared to the previous algorithms.

Keywords : P1500, IEEE 1149.1, Interconnect test

I. Introduction

반도체 집적 기술이 고도로 발전해 온 결과 과거 PCB 상의 기능을 하나의 칩으로 집적시킨 SoC (System on Chip) 환경으로의 변화가 진행되고 있다. SoC 환경으로의 변화는 고도의 기술을 요구하는 미래 전자산업의 핵심적인 요소로 자리잡고 있으며, 이러한 고밀도의 집적 기술에 의한 시스템 환경 하에서 이를 테스트하는 것 역시 과거의 그것과는 다른 고도의 환경을 요구하게 되었다[1,2]. 결국 집적 기술의 발달에 의해서 칩의 제작비용은 점차로 감소하고 있지만 더욱 복잡해지는 환경에 대한 테스트의 수행 비용은 오히려 상승할 것이 예상되므로 상대적인 테스트 수행 비용의 증가 문제가 필연적으로 드러나게 될 것이다[1]. 따라서 테스트 수행 비용을 줄이기 위한 테스트 수행 시간의 단축이라는 과제는 앞으로도 테스트를 수행함에 있어 주요 논점이 될 것이다. SoC 환경에서의 테스트 수행

* 正會員, 延世大學校 電氣電子工學科
(Dept. of Electrical Eng., Yonsei Univ.)

※ 이 연구는 산업자원부와 과학기술부의 시스템 IC 2010 사업의 지원으로 수행하였습니다.
接受日字 : 2002年 3月 19日, 수정완료일 : 2003年 1月 7日

은 과거 보드 수준의 IEEE 1149.1 std.[4,5]를 기반으로 한 테스트에서 P1500[3]이라는 새로운 개념을 기반으로 한다. 이는 기본적으로는 IEEE 1149.1 과 유사한 구조를 가지고 있으나 적용되는 방식에서는 차이를 보인다. 이중 SoC 내부 IP 상호간 연결의 고장 유무를 테스트하는 것은 연결의 고장이 곧 칩의 고장을 의미하는 것인 만큼 그 중요성이 더 커졌다고 할 수 있다.

P1500 에 의한 SoC 내부 회로의 연결을 테스트하는 것은 기본적으로 경계 주사 방식에서 사용된 경계 주사 셀(Boundary Scan Cell)[4]의 개념과 유사한 래퍼 셀(Wrapper Cell)[6]이라고 불리는 기본 구조를 사용하여 경계 주사 방식과 유사한 개념으로 받아들일 수 있지만 SoC 환경에서의 테스트는 보드 수준보다 훨씬 복잡한 내부 구조에 대해서 수행해야 할 여러 테스트를 한 번에 수행하므로 고도의 테스트 환경과 더욱 긴 테스트 수행 시간을 필요로 한다. 따라서 테스트 수행 시간의 단축은 테스트 길이의 단축과 깊은 관련을 맺고 있으며, 이는 연결 테스트를 수행함에 있어서도 마찬가지이다.

충실한 연결 테스트를 수행하기 위한 알고리듬이 만족시켜야 할 핵심적인 두 가지 요건은 상호 연결의 고장 유무를 검출해 내는 것과, 이 고장의 위치를 분석해 낼 수 있는 것이다. 위의 두 가지 조건을 만족시키지 않는 테스트 벡터는 완전한 테스트를 수행할 수 없으며, 위의 조건을 만족시키는 테스트 벡터가 입력된다 하더라도 테스트 수행 시간을 단축시키기 위해서는 테스트 벡터가 가능한 짧은 길이를 가지고 있어야 한다. 이중 상호 연결의 고장 유무 검출에 대한 문제는 비교적 간단히 해결할 수 있으나, 고장의 위치를 분석하는 문제는 쉽게 해결되지 않는다. 왜냐하면 전체 연결 중 여러 부분에서 고장이 발생했을 때, 이를 통해 출력되는 벡터들의 결과가 같아진다면 고장이 연관된 연결들을 구별해 낼 수 없게 되어 고장이 발생한 위치를 판단할 수 없기 때문이다. 결국 위의 문제들을 해결하는 것은 입력되는 테스트 알고리듬에 의한 것으로써, 기존에 제시된 주요한 알고리듬은 계수 시퀀스(counting sequence), 완전 독립 테스트 집합(maximal independent set), 워킹 시퀀스(walking sequence), 분할 그룹 워킹 시퀀스(split group walking sequence)등이 있다[7~9].

계수 시퀀스(counting sequence)는 모든 네트에 서로 독립적인 테스트 벡터를 통과시킨 후 이 때 나오는 결과를 관찰하여 해당 네트에 이상이 있는지를 검사하는 것이다. 따라서 전체 네트의 개수를 n이라 할 때 계수 시퀀스에서 필요한 최소한의 PTV의 개수는 $\lceil \log_2 n \rceil$ 개다. 그러나 계수 시퀀스는 고장의 검출 및 분석이라는 기본 전제를 충실히 만족시키지 못하는 불완전한 테스트 알고리듬이다.

완전 독립 테스트 집합(maximal independent test set)은 네트의 개수를 n이라고 할 때 $\lceil \log_2(n+2) \rceil \le p \le n$으로 정해지는 p의 수만큼의 테스트 길이를 가지도록 하면서 일정한 개수의 "1" 값을 배열하는 시퀀스이다. 고장 위치의 분석에 대한 문제는 공정 기술의 차이에 따라서 극히 제한된 수의 네트에 대해서만 고려하는 것이 가능하다는 것을 보임으로써 해결을 시도한다[4]. 하지만 이는 고장 위치의 분석에 대한 일반적인 해결안은 제시하지 못한다.

워킹 원 시퀀스(walking one sequence)는 모든 고장을 검출해 내면서 동시에 고장 위치의 분석과 관련된 문제도 완벽하게 해결한다. 하지만 워킹 원 시퀀스를 가할 때 필요한 테스트 패턴의 길이가 네트의 수와 같은 n개로써 대단히 긴 테스트 수행시간이 필요하다는 치명적인 단점이 있다.

분할 그룹 워킹 시퀀스(split group walking sequence)[8]는 네트를 그룹별로 나누어 워킹 시퀀스를 응용하여 대입함으로써 워킹 시퀀스보다 적은 개수의 테스트 벡터로 같은 효과를 제공하는 방법이다. 이 방식은 전체 네트의 수를 n이라고 할 때 n에 의해 정해지는 $k(=\lceil \sqrt{n} \rceil)$만큼의 네트를 가진 N_g개의 그룹들로 구성된다. 그 결과는 워킹 원 시퀀스와 마찬가지로 모든 고장의 검출과 위치 분석이 가능하다. 이 때 필요한 테스트 패턴의 길이는 두 가지 경우로 나뉘어 지는데 우선 마지막 그룹에 속한 네트의 수가 k와 같을 때는 $4k-1$개가 필요하고, 마지막 그룹에 속한 네트의 수가 k와 같지 않을 때는 $4k-2$개의 PTV가 필요하게 된다. 그러나 분할 그룹 워킹 시퀀스는 여전히 긴 테스트 길이를 가지는 알고리듬이다.

본 논문에서 제시하는 알고리듬은 상호연결 테스트 수행 시 고장의 유무를 검출해 내면서 고장의 위치 역시 완전히 분석할 수 있고, 기존에 제시된 고장의 유무 검출과 위치 분석이 가능한 알고리듬들에 비해 테스트 수행시간에 핵심적인 영향을 끼치는 테스트 패턴의 길이가 훨씬 짧다. 이에 더해서 본 논문이 제시하는 테스트 패턴은 일정한 규칙성을 가지고 구성되어 있으므로

하드웨어 구현이 편리할 것이다.

본 논문의 구성은 다음과 같다. 첫째로 상호연결 테스트를 이해하는데 필요한 용어를 정의함으로써 테스트를 수행할 때 관건이 되는 고장 모델에 대해 이야기한다. 둘째로 본 논문이 제시하는 GNS 시퀀스라는 새로운 알고리듬을 구성하는 방법을 설명하고 이것이 고장 위치를 분석하는데 관건이 되는 confounding 고장 증후를 발생하지 않음을 증명함으로써 GNS 시퀀스가 가지는 특성을 논한다. 마지막으로 GNS 시퀀스와 기존의 알고리듬의 고장 검출 및 고장 위치 분석 능력과 테스트 벡터의 길이를 비교함으로써 GNS 시퀀스가 가지는 효과를 설명한다.

Ⅱ. 용어 정의

Soc 환경은 <그림 1>과 같이 하나의 칩 안에 다양한 기능을 수행하는 IP들이 포함되어 있으며, 각 IP들은 다양한 연결 회로 및 버스 구조를 이용하여 연결되어 있다. IP들간의 연결 테스트는 IP들의 동작과는 무관하게 래퍼 셀들 사이의 연결만을 통해서 이루어진다. 이 때 래퍼 셀에 인가될 테스트 알고리듬의 예를 표 1에 나타내었다. 이 예를 통해 설명될 본 논문에서 사용되는 용어들은 다음과 같다.

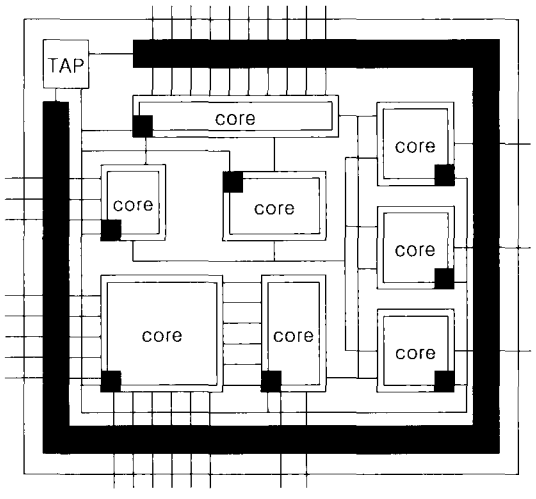


그림 1. SoC 구조
Fig. 1. SoC structure.

■ 테스트 패턴 : 테스트를 수행하기 위해 입력되는 벡터의 집합. 테스트 시퀀스라고도 한다. <표 1>의 경우는 워킹 원 시퀀스의 예를 보인 것이다.

■ PTV(Parallel Test Vector) : 테스트 패턴 가운데 전체 네트에 동시에 입력되는 벡터이다. 따라서 PTV의 개수는 테스트 수행시간을 결정짓는 중요한 요소이다. <표 1>에서 전체 네트에는 "1000000"이라는 PTV 1이 동시에 인가되는 것이다.

■ STV(Sequential Test Vector) : 테스트 패턴 가운데 한 네트에 순차적으로 입력되는 벡터이다. 따라서 STV의 길이는 PTV의 개수와 같다. <표 1>에서 n_2라는 하나의 네트에는 "0100000"이라는 STV 2 가 순차적으로 입력된다.

■ SRV(Sequential Response Vector) : 한 네트에 연속적으로 가해진 STV에 대해 출력되는 출력 값이다. 이때 결과로 나오는 SRV를 관찰함으로서 해당하는 네트에 이상이 있는지의 여부를 확인한다.

■ 테스트 길이(test length) : 테스트 패턴의 길이를 말하며 따라서 테스트 길이가 짧을수록 테스트 수행시간이 짧아진다.

■ 자가 분석(self-diagnosis) : 가해지는 테스트 패턴에 대한 SRV를 관찰하는 것만으로 회로의 고장 여부를 판단 할 수 있을 때 자가 분석적이라고 한다.

■ 단락 고장(short fault) : 단락은 둘이나 그 이상의 네트들 사이에서 일어나며 OR-단락 고장(OR-short fault)과 AND-단락 고장(AND-short fault) 그리고 상대적 드라이버 단락(weak short)으로 구분한다. 네트가 논리값 "1"에 의해서 조정될 때 결과값이 각 네트의 OR의 논리값으로 나타나는 것이 OR-단락 고장이고, 네트가 논리값 "0"에 의해서 조정될 때 결과값이 각 네

표 1. 테스트 벡터
Table 1. test vector set.

	PTV 1	PTV 2	PTV 3	PTV 4	PTV 5	PTV 6	PTV 7	
n_1	1	0	0	0	0	0	0	STV 1
n_2	0	1	0	0	0	0	0	STV 2
n_3	0	0	1	0	0	0	0	STV 3
n_4	0	0	0	1	0	0	0	STV 4
n_5	0	0	0	0	1	0	0	STV 5
n_6	0	0	0	0	0	1	0	STV 6
n_7	0	0	0	0	0	0	1	STV 7

트의 AND의 논리값으로 나타나는 것이 AND-단락 고장이다. 상대적 드라이버 단락은 단락된 네트 중 한 네트가 우월할 때 SRV가 지배적인 네트의 SRV에 따른다. 즉 상대적 드라이버 단락은 공정에 따라서 AND-단락 고장이나 OR-단락 고장으로 모델링 할 수 있다.

■ Aliasing 고장 증후 : 고장난 네트들의 SRV가 정상 동작하는 SRV와 같은 경우로써, <그림 2>의 n_2와 n_5 사이에서 OR-단락 고장이 발생한 경우의 n_2와 n_5의 SRV와 정상 동작하는 n_3의 SRV는 "101"로서 같아지게 된다. 이 경우 고장 위치를 분석할 수 없다.

■ Confounding 고장 증후 : 가해진 테스트 패턴에 대해서 고장난 네트들의 SRV가 독립적으로 고장난 다른 네트들의 SRV와 같은 경우로서, <그림 2>의 n_1와 n_7 사이에서 OR-단락 고장이 발생한 경우와 n_3과 n_8이 OR-단락 고장을 일으킨 경우의 SRV는 "111"로 같아지게 된다. 따라서 고장 위치를 분석하는 것은 불가능하다.

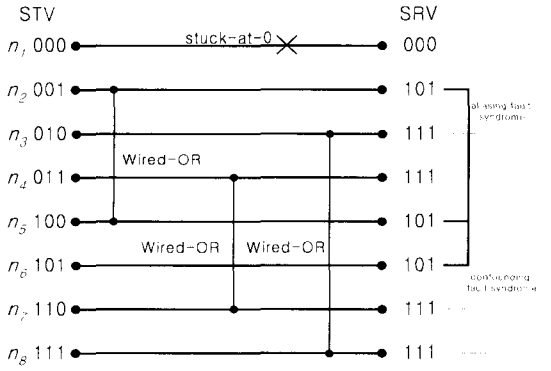


그림 2. 고장 모델의 예
Fig. 2. Example of fault model.

■ 독립적(independent) : OR-단락 고장의 경우 한 네트에서 "1" 값이 포함되는 PTV의 모든 위치에 대해서 다른 네트가 모두 "1" 값을 가지고 있지 않은 경우 이 두 네트는 독립적이라고 한다. AND-단락의 경우 논리값 "0"에 대해 같이 적용한다. 즉 <그림 2>에서 n_2는 첫 번째 비트에만 "1" 값을 가지고 있고, n_5는 세 번째 비트에서 "1" 값을 가지므로 n_2와 n_5는 서로 독립적이다.

■ 다중 네트 고장은 2-네트 고장의 합집합이고, 모든 다중 네트 고장을 테스트하는 것은 테스트 벡터 개수의 지수 적인 증가를 가져오므로 고장을 고려하는 것은 2-네트 고장에 대한 것으로만 제한한다[7]. 또한 한 네트에 대한 다중 고장은 고려되지 않는다. 왜냐하면 다중 고장을 고려하여도 검출되는 고장은 다중 고장 중 한가지의 고장만이 확실히 검출되기 때문이다.

Ⅲ. GNS 시퀀스

1. 네트의 그룹별 구성

GNS 시퀀스를 설명하기 위해서는 그 구성의 기초가 되는 몇 가지 정의가 필요하다. GNS 시퀀스는 테스트 패턴을 인가하는 전체 네트를 몇 개의 작은 단위로 구분한 후 이 단위별로 각각 구별되는 테스트 패턴을 가함으로써 고장의 검출 및 분석과 관련된 문제들을 해결한다. 이 때 일정 단위별로 가해지는 테스트 패턴이 효과적으로 동작하도록 다음과 같이 전체 네트를 여러 개의 소그룹으로 나누어준다.

표 2. 네트의 구성
Table. 2. net composition for GNS sequence.

네트	그룹	네트
n_1	G_1	N_{11}
n_2		N_{12}
n_3		N_{13}
n_4	G_2	N_{21}
n_5		N_{22}
n_6		N_{23}
n_7	G_3	N_{31}
n_8		N_{32}
n_9		N_{33}

<표 2>는 왼쪽의 네트로 표시한 전체 네트에 대해서 GNS 시퀀스에서 사용하는 새로운 네트의 구성을 나타낸 것이다. 여기에서 전체 네트는 3개의 네트당 하나의 그룹으로 묶여서 전체 9개의 네트가 각각 3개의 네트로 구성된 3개의 그룹으로 새롭게 나누어진 것을 확인할 수 있다. 이때 표에서 보이는 그룹과 새로운 네트의 구성은 다음과 같은 방식으로 이루어진다. 우선 전체 네트의 수가 n 개라고 하면, 이들은 n_1, n_2, n_3, · · ·와 같은 순서로 표시할 수 있는데, 이 때 n 에 따라 결정

되는 k 라는 수를 다음과 같이 정의한다.

· $k = \lceil \sqrt{n} \rceil$

이 때 k 는 n 의 제곱근에 가까운 값으로써, 이는 효과적으로 테스트 벡터를 가하여 짧은 테스트 길이를 가지기 위해 정한 것이다. 즉 GNS 시퀀스의 특성에 의한 것으로써 자세한 설명은 뒤에서 이루어질 것이다. 그리고 이 k 에 의해서 전체 네트를 k 개의 네트를 포함한 여러 개의 그룹으로 G_1, G_2, G_3, · · ·와 같이 나타낸다. 즉 하나의 그룹은 k 개만큼의 네트를 포함하고 있고, 이때 전체 그룹의 개수는 아래와 같이 결정된다.

· $num(G) = k-1$ ($(k-1)^2 < n \le k^2$ k)

· $num(G) = k$ ($k^2-k < n \le k^2$)

이는 전체 네트의 수에 따라 결정되는데, k 개의 네트를 포함하는 그룹이 k 개 존재한다는 것은 전체 네트의 수가 n 개일 때, $k^2-k < n \le k^2$ 인 경우를 의미한다. 즉 $n=k^2-k$ 인 경우는 k 개만큼의 네트를 포함하는 그룹이 k 개 존재하도록 배열할 때 마지막 그룹의 내부에 0개의 네트가 포함되는 경우로써 전체 그룹의 숫자는 하나가 줄어든 $k-1$ 개가 되는 것이다. 또한 네트의 수가 k^2보다 커지거나 $(k-1)^2$보다 적어지면 위에서 정의한 $k=\lceil \sqrt{n} \rceil$ 의 범위를 벗어나므로, k는 새로운 값으로 정해진다. 결국 그룹은 G_1, G_2, G_3, · · · G_k(or G_{k-1})와 같이 정해진다. 마지막으로 각 그룹의 내부에 존재하는 k 개의 네트들을 순서대로 N_1, N_2, N_3, · · · N_k 와 같이 표현한다. 결국 전체 네트에 대해서 그룹 및 그룹 내부의 네트를 표현한 것인데, 이 두 가지를 한번에 표현하면 다음과 같다.

· N_{11}, N_{12}, N_{13}, · · ·, N_{21}, N_{22}, N_{23}, · · ·, $N_{k(k-1)}$, $N_{k(k-1)}$, N_{kk}

즉, N_{11}은 첫 번째 그룹에 속한 첫 번째 네트를 의미하는 것이고, N_{kk}는 k 번째 그룹에 속한 k 번째 네트를 의미하는 것이다. 이것을 <표 2>에서의 예로 살펴보면, 전체 네트의 개수는 9개이므로 $k=3$ 이고, 따라서 한 그룹 내부에는 3개씩의 네트가 포함된다. 또한 $n-9=k^2$ 이므로 그룹의 개수는 $k=3$ 개다. 결국 네트가 9개인 회로에 대해서는 3개의 네트로 구성된 3개의 그룹에 GNS 시퀀스를 가함으로써 연결 테스트를 수행해 낼 수 있다.

2. 워킹 시퀀스

GNS 시퀀스는 그룹별 워킹 시퀀스(group walking sequence), 네트별 워킹 시퀀스(net walking sequence), 네트별 이동 워킹 시퀀스(shifted net walking sequence)의 세 부분의 워킹 원 시퀀스로 이루어지며 이들을 각각 W_g, W_n, W_s로 표현한다. 즉 전체 네트는 k개 또는 $k-1$개의 그룹으로 나누어진 채로 테스트 패턴을 입력받고, 이때 입력되는 테스트 패턴의 각 STV는 위의 세 가지 워킹 시퀀스로 구성되는 것이다. 각 워킹 원 시퀀스는 기존의 워킹 시퀀스를 변형시킨 형태를 가지고 있으며, 이들을 각 그룹별로 다르게 변형시켜 가함으로써 테스트 패턴이 최소의 테스트 길이를 가지도록 한다.

그룹별 워킹 시퀀스 W_g는 하나의 그룹내부에 동일한 순서의 비트에 "1" 값을 인가하고 이를 제외한 나머지 비트에는 모두 "0" 값을 인가한다. 그리고 그룹의 순서에 따라 "1" 비트가 입력되는 비트의 순서를 한 비트씩 이동시킨다. 예를 들어 <표 3>에서 보면 첫 번째 그룹에 속한 네 개의 네트들에 대해서는 모두 첫 번째 비트에 "1" 값을 입력하고 나머지 비트에는 모두 "0" 값을 입력한다. 그리고 두 번째 그룹에는 "1" 비트가 입력되는 순서의 비트를 이동하여 두 번째 비트에는 모두 "1" 값을 입력하고 나머지 비트에는 모두 "0" 값을 입력한다. 따라서 네 개의 그룹으로 구성된 W_g에 대해서 "1" 값을 순차적으로 입력할 비트는 그룹의 개수와 같은 4개가 존재하게 된다. 즉 W_g는 그룹 단위로 워킹 원 시퀀스를 가해준 것이므로 이에 따른 W_g의 PTV의 개수는 네트의 수에 따라 달라지는 그룹의 개수 k 또는 $k-1$과 같게 되는 것이다. W_g를 가하는 목적은 각 그룹들 간에는 서로 독립적이면서 동일 그룹에는 같은 PTV를 가함으로써 입력되는 테스트 벡터를 그룹 단위로 나누어주기 위한 것이다.

네트별 워킹 시퀀스 W_n은 모든 그룹에 동일하게 워킹 원 시퀀스를 입력한다. 즉 하나의 그룹 내부의 각 네트에 대해 독립적인 비트에 대해서 "1" 값을 하나씩 인가하고 나머지 비트에 대해서는 모두 "0" 값을 인가한다. 따라서 <표 3>에서 보는 바와 같이 모든 그룹은 동일한 형태의 워킹 원 시퀀스가 인가되는 것이다. 이 때 인가되는 테스트 길이는 하나의 그룹에 속한 네트의 수만큼의 독립적인 비트에 대해 "1" 값이 입력될 수 있도록 4개가 되어야 한다. 따라서 PTV의 개수는 각 그룹의 네트의 수 k와 같다. W_n을 가하는 목적은 그룹별로 워킹 원 시퀀스를 가함으로써 앞서 W_g에서 동일하게 분류된 각 그룹 내부의 네트들을 구별하기 위한 것이다. 따라서 W_g와 W_n에 가한 테스트 벡터만으로도

완전히 독립적인 시퀀스를 가할 수 있게 되었으나, 이는 여전히 confounding 고장 증후를 일으킬 가능성이 있으므로 네트별 이동 워킹 시퀀스 W_s를 가함으로써 이 문제를 해결한다.

표 3. GNS 시퀀스(n=15, k=4)
Table 3. GNS sequence(n=15, k=4).

네트	W_g				W_n				W_s			
N_{11}	0	0	0	1	0	0	0	1	0	0	0	1
N_{12}	0	0	0	1	0	0	1	0	0	0	1	0
N_{13}	0	0	0	1	0	1	0	0	0	1	0	0
N_{14}	0	0	0	1	1	0	0	0	1	0	0	0
N_{21}	0	0	1	0	0	0	0	1	0	0	1	0
N_{22}	0	0	1	0	0	0	1	0	0	1	0	0
N_{23}	0	0	1	0	0	1	0	0	1	0	0	0
N_{24}	0	0	1	0	1	0	0	0	0	0	0	1
N_{31}	0	1	0	0	0	0	0	1	0	1	0	0
N_{32}	0	1	0	0	0	0	1	0	1	0	0	0
N_{33}	0	1	0	0	0	1	0	0	0	0	0	1
N_{34}	0	1	0	0	1	0	0	0	0	0	1	0
N_{41}	1	0	0	0	0	0	0	1	1	0	0	0
N_{42}	1	0	0	0	0	0	1	0	0	0	0	1
N_{43}	1	0	0	0	0	1	0	0	0	0	1	0

네트별 이동 워킹 시퀀스 W_s는 첫 번째 그룹에 W_n에서와 같이 워킹 원 시퀀스를 가하고, 그룹의 순서 및 각 그룹내의 네트의 순서에 따라 입력된 "1" 값을 한 비트씩 이동한다. 즉 N_{11}에서 첫 번째 비트에 "1" 값이 들어간 경우 N_{21}에는 두 번째 비트에 "1" 값이 들어간다. 이러한 방식으로 "1" 비트를 이동시키면 k번째 비트인 최상위 비트에 "1" 값을 가지는 네트는 "1" 값의 위치를 이동할 비트가 없게 된다. 이 경우 하나의 그룹 내부에 존재하는 네트의 개수 k에 대해서는 서로 독립적인 "1" 비트를 인가할 수 있는 k개 만큼의 테스트 길이가 존재하므로 최상위 비트인 k번째 비트에 "1" 값을 가지는 네트가 속한 그룹 다음 그룹의 동일한 순서의 네트에 위치하는 "1" 값을 다시 최하위 비트인 첫 번째 비트로 이동시킴으로서 비트 이동에 따르는 테스트 벡터 길이의 증가요인을 제거한다. 즉 <표 3>에서 W_s의 경우 N_{11}은 네 번째 비트에서 "1" 값을 가지고 있으며, 이에 따라 N_{21}은 다시 첫 번째 비트에서 "1" 값을 가지게 된다. 결국 W_s에 입력된 테스트 벡터를 살펴보면 순서를 고려한 워킹 원 시퀀스를 가한 형태가 된다. 따라서 Ws의 PTV의 개수 역시 각 그룹에 속한 네트의 수인 k이다.

앞에서 k값을 $\lceil \sqrt{n} \rceil$ 와 같이 정의한 것은 전체 그룹의 개수 k에 대해서 W_s에서 정해지는 PTV의 개수를 네트의 수와 같은 k개로 맞추기 위함이다. 왜냐하면 Ws에 속한 각 그룹의 동일한 순서의 네트는 그룹의 개수만큼 "1" 비트의 위치를 이동하게 된다. 그런데 이 때 "1" 비트를 이동하는 횟수가 그룹의 개수보다 커지게 되면 결국 "1" 비트는 이전의 그룹에서 존재했던 비트로 돌아오게 된다. 이 경우는 confounding 고장 증후의 발생 가능성을 배재할 수 없게 되어 고장 위치를 분석할 수 없게 된다. 따라서 W_s에서의 PTV의 개수는 최소한 그룹의 개수보다는 커야하며, 이 때 앞에서 정의한 테스트 길이인 k값은 이 조건을 충분히 만족시킨다. 즉 최적의 PTV의 개수를 얻기 위해서는 그룹의 개수와 각 그룹이 포함하는 네트의 수가 같은 경우이다. 따라서 k를 네트의 개수에 대한 $\lceil \sqrt{n} \rceil$ 로 정한 것이다.

AND-단락 고장에 대해서는 위와 같은 형태의 워킹-0시퀀스를 가함으로써 같은 결과를 얻을 수 있다.

3. 테스트 길이

테스트 길이는 테스트 알고리듬의 성능을 평가하는 중대한 기준이 된다. 테스트 수행 시간의 증가는 곧 테스트 비용의 증가를 야기하는 것이므로 가능하면 짧은 테스트 길이가 요구되는 것이다. GNS 시퀀스에서의 테스트 길이는 각 워킹 원 시퀀스 별로 다음과 같다. 우선 W_g는 각 그룹별로 워킹 원 시퀀스를 가한 것이므로 그룹의 개수와 같은 테스트 길이를 가진다. 그룹의 개수는 네트의 수에 따라서 k(또는 k-1)개가 된다. W_n은 그 자체로 워킹 원 시퀀스의 형태를 지니고, 이는 각 그룹별로 구성되므로 테스트 길이는 한 그룹내의 네트의 개수인 k와 같다. W_s 역시 구성 형태가 다를 뿐 테스트 길이는 W_n와 같으므로 GNS 시퀀스의 테스트 길이는 위의 세 부분의 테스트 길이를 합한 $3k$(또는 $3k$-1)가 된다.

4. 고장의 검출 및 분석

GNS 시퀀스에서 하나의 STV는 기본적으로 W_g, W_n, W_s의 각 워킹 원 시퀀스에 대해서 하나의 "1" 값을 가지고 있으므로 하나의 STV는 3개의 "1" 값을 가지며, 나머지 비트는 모두 "0" 값을 가지게 된다. 따라서 "1" 값이나 "0" 값 만으로만 이루어진 STV는 존재

하지 않고, 이를 통해 모든 고착 고장에 대한 검출이 가능함을 알 수 있다. 따라서 모든 네트에 대해 SRV를 관찰하는 것만으로 고착 고장의 검출 및 위치에 대한 분석 역시 가능하다.

또한 단락 고장에 대해서는 서로 독립적인 STV에 대해 동일한 SRV들이 존재하는 경우 단락 고장의 존재를 쉽게 확인할 수 있다. 그러나 이에 대한 분석은 앞에서 살펴본 aliasing 고장 증후와 confounding 고장 증후가 발생하지 않아야 보장된다. 이중 aliasing 고장 증후에 대한 문제는 쉽게 해결된다. GNS 시퀀스는 각 워킹 시퀀스가 각 STV마다 하나의 "1" 값을 포함하고 있으므로 각 네트에 인가되는 STV는 모두 3개의 "1" 값을 가지고 있다. 이것이 aliasing 고장 증후의 제거를 보장한다. 왜냐하면 GNS 시퀀스에서 단락 고장이 일어난 네트들의 경우 고장이 일어나지 않은 네트의 SRV와 단락 고장이 일어난 네트들의 SRV는 동일한 개수의 "1" 값을 가질 수 없기 때문이다. 즉 인가되는 모든 STV들이 독립적이므로 단락 고장이 일어난 네트들은 적어도 3개보다는 많은 "1" 값으로 구성된 SRV를 가질 것이며, 고장이 일어나지 않은 네트들은 STV와 같은 결과를 가지므로 3개의 "1" 값을 가지는 SRV를 가질 것이다. 따라서 단락 고장이 일어난 네트들과 고장이 일어나지 않은 네트들이 같은 SRV를 가질 수 있는 가능성이 배제되어 aliasing 고장 증후 발생의 가능성 역시 제거된다.

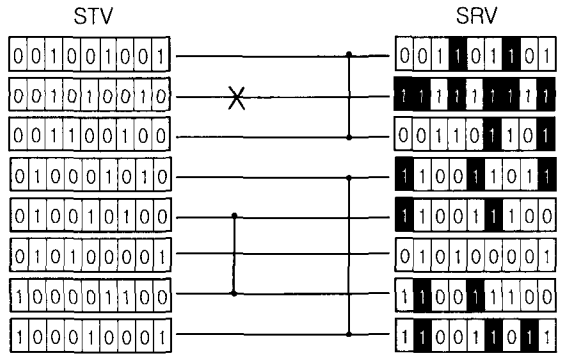


그림 3. GNS 시퀀스의 고장 검출
Fig. 3. Example of fault detection.

<그림 3>은 GNS 시퀀스의 고장 검출의 예를 보인 것이다. 그림에서 가로로 표시된 선이 회로의 연결일 때 X로 표시된 것은 고착 고장을 나타내고, 세로로 연결된 선은 해당 네트들 사이의 단락 고장을 표시한다. 두 번째 네트의 경우 고착 고장은 고착-1 고장이라고 가정했을 때, 전체 비트가 "1" 값으로만 구성된 SRV가 고착-1 고장의 존재를 검출해 내고 그 위치 역시 분석해 낸다. 또한 단락 고장의 경우 서로 동일한 SRV를 가지는 네트들의 쌍을 통해 단락 고장의 존재 여부를 확인할 수 있다. aliasing 고장 증후의 경우 검은색으로 표시된 비트가 각종 고장에 의해 STV에서 변형된 SRV의 비트를 나타내는데, 고장이 일어나지 않은 6번째 네트의 경우 "1" 비트가 3개 존재하는데 반해, 기타 단락 고장이 일어난 1, 3, 4, 5, 7, 8번째 네트의 경우 이 3개의 "1" 값 이외에 네트별로 <그림 3>에서 검은색으로 표시된 다양한 개수의 "1" 값이 추가되어 있으므로 고장이 일어난 네트와 고장이 일어나지 않은 네트가 동일한 SRV를 가질 가능성은 완전히 배제된다. 따라서 aliasing 고장 증후는 발생하지 않는다.

4. Confounding 고장 증후

고장 위치의 분석을 위한 핵심적인 문제가 되는 confounding 고장 증후의 제거는 GNS 시퀀스 내부의 각 워킹 원 시퀀스를 차례로 살펴봄으로써 확인이 가능하며, 이를 확인하기 위해 다음과 같이 confounding 고장 증후 발생이 가능한 경우를 줄여 나간다.

우선 하나의 그룹에 대해 워킹 원 시퀀스의 형태를 취하고 있는 W_n은 동일한 그룹내부에 대해서는 완벽한 형태의 워킹 원 시퀀스를 지원하므로 confounding 고장 증후가 발생하지 않는다. 워킹 원 시퀀스는 모든 STV가 서로 독립적인 결과를 가지도록 하나의 비트에만 "1" 값을 인가하기 때문에 단락 고장이 일어난다 해도 특정 비트에서 "1" 값을 포함하는 SRV는 2개 이상이 될 수 없다. 따라서 confounding 고장 증후는 발생할 수 없는 것이다. confounding 고장 증후는 서로 다른 그룹들이 연관된 형태로만 일어날 수 있다.

또한 동일한 그룹 내부의 동일한 비트에 "1" 값을 입력한 그룹별 워킹 시퀀스인 W_g에 의해서, confounding 고장 증후는 두 개의 그룹 사이에서만 일어난다는 것을 알 수 있다. 즉 앞에서 2 네트 사이의 고장만을 가정했으므로, 서로 다른 두 그룹 내부에 존재하는 네트들 사이에서 단락 고장이 일어난 경우 이에 대한 SRV가 다른 단락 고장이 발생한 SRV와 같은 결과를 가지려면, 이 네트들은 앞에서 단락이 일어난 두 그룹 내부의 네트들이어야만 한다. 즉 <표 3>의 첫 번째 그룹에서 W_g는 "0001"이라는 시퀀스를 가지고 두 번째 그룹은 "0010"이라는 시퀀스를 가지므로 이 두 그룹에 속한

네트들 사이에서 단락 고장이 발생했다면 이들에 대한 SRV는 "0011"이 될 것이다. 그런데 또 다른 단락 고장이 "0011"이라는 SRV를 가지기 위해서는 그 각각의 네트들이 첫 번째 그룹과 두 번째 그룹에 각각 포함되어야만 하는 것이다.

이때 W_n에 대해서 다시 한번 살펴보면, 위에서 confounding 고장 증후와 관련된 네 개의 네트는 그룹 내에서 동일한 순서에 존재함을 알 수 있다. 즉 표 3에서 N_{11}과 N_{22}가 단락된 경우 W_n의 SRV는 "0011"이 된다. 그런데 이 때 첫 번째 그룹과 두 번째 그룹 내에 속한 네트들 중에서 단락이 되었을 때 "0011"이라는 SRV를 가질 수 있는 경우는 N_{12}와 N_{21}이 단락된 경우 뿐이다. 이는 W_n이 각 그룹별로 완전한 형태의 워킹 원 시퀀스를 나타내고 있는 것이 기인한다.

결국 위의 내용을 정리하면, GNS 시퀀스에서 confounding 고장 증후가 발생할 수 있는 경우는 관련된 두 개의 그룹 내부의 동일한 순서에 위치한 각 2개의 네트들이 서로 순서를 바꾸어 단락되었을 때 뿐이다. 이후의 증명은 수식을 통하여 W_s를 살펴봄으로써 가능하며, 여기까지의 결과를 정리하면 N_{ip}와 N_{iq}, 그리고 N_{jp}와 N_{jq}사이에서만 confounding이 일어날 수 있다는 것이다(단, $p<q\leq k$이고 $i<j\leq k$). 그러나 이때 N_{ip}-N_{iq}간의 OR-단락과 N_{jp}-N_{jq}간의 OR-단락은 동일 그룹 내에서의 단락이므로 W_g에 의해서 구분이 가능하고, N_{ip}-N_{jq}간의 OR-단락과 N_{ip}-N_{jq}간의 OR-단락은 각 그룹에서 동일한 순서의 네트끼리의 단락이므로 W_n에 의해서 confounding 고장 증후가 발생하지 않으므로 결국 confounding 고장 증후가 발생할 수 있는 경우는 N_{ip}-N_{jq}간의 OR-단락과 N_{ip}-N_{jq} 간의 OR-단락 사이에서 뿐이다.

이때 W_s에서의 모든 STV는 각각 1개의 "1" 값을 가지고 있으며 이때 N_{ip}가 W_s에서 가지는 "1" 값의 위치 $X(0<X\leq k)$는

$$X = i+p-1 \ (i+p-1\leq k) \text{ -------①}$$
$$X = i+p-1-k \ (i+p-1>k) \text{ -----②}$$

의 두 가지 경우로 표현된다. 즉 "1" 값의 위치는 그룹과 네트의 순서에 따라서 결정되는 것이다. 이때 전자는 "1" 값이 하위 비트에서 상위 비트로 이동한 경우이고, 후자는 "1" 값이 최상위 비트에서 이동해서 다시 최하위 비트로 넘어간 이후의 위치를 나타낸다.

예를 들면 <표 3>에서 k=4이고 N_{31}의 경우를 보면 i=3이고 p=1이므로 X=3+1-1=3이다. 따라서 "1" 값은 세 번째 비트에 존재하게 된다. 또 같은 경우 N_{33}인 경우는 i=3이고 p=3이므로 $i+p-1$=5이므로 k(=4)보다 큰 값을 가지게 되고 따라서 이 경우에는 $X=i+p-1-k=1$로써 "1" 값은 첫 번째 비트에 존재한다. 따라서 Ws에서 이동되는 비트에 의해서 "1" 값의 위치는 위의 두 가지 경우로 나타나므로 confounding 고장 증후는 두 그룹 사이의 공통적인 네트들이(N_{ip}, N_{iq}, N_{jp}, N_{jq}) <표 4>와 같이 조합된 경우에 대해서만 발생할 수 있다(단, $p<q\leq k$이고 $i<j\leq k$).

표 4. W_s의 STV에서의 "1" 값의 위치(X)
Table 4. "1" bit position in a STV of W_s.

네트	X					
	case i	case ii	case iii	case iv	case v	case vi
N_{ip}	①	①	①	①	1	②
N_{iq}	①	①	①	②	2	②
N_{jp}	①	①	②	①	2	②
N_{jq}	①	②	②	②	2	②

이 중 표에 나타나지 않은 경우는 동일 그룹 내에 존재하는 q번째 네트보다 앞서 있는 p번째 네트에 ②의 형태로 "1" 값이 나타나고, q번째 네트에는 ①의 형태가 나타나는 경우다. 그러나 "1" 값의 이동 횟수는 그룹의 수 k(또는 k-1)를 넘을 수 없고 "1" 값이 이동할 수 있는 비트 수인 PTV의 길이는 k이므로 이는 실제로 일어나지 않는 경우이다. 따라서 나머지 경우에서만 confounding 고장 증후의 발생이 가능하다. 이때 실제로 confounding 고장 증후가 발생하는 경우는 앞서 서술한 대로 N_{ip}와 N_{jq}, 그리고 N_{iq}와 N_{jp}가 각각 단락된 경우이고, 이때 N_{ip}와 N_{jq}, 그리고 N_{iq}와 N_{jp}는 같은 위치에서 "1" 값을 가진다. 즉 X값이 같은 경우에 한해서 confounding 고장 증후는 발생하는 것이다.

따라서 이들 사이에서 confounding 고장 증후가 발생하는 경우를 고려하면 그 결과는 다음과 같다.

위에서 기본 가정은 $p<q\leq k$ 와 $i<j\leq k$ 이다. 따라서 <표 5>의 결과에 나타난 바와 같이 위의 모든 경우는

표 5. confounding 고장 증후 발생 경우의 제거
Table 5. Elimination of confounding fault syndrome.

case	"1"값의 위치	필요 조건	결과
case i	N_{ip}=①N_{iq}=①N_{jp}=①N_{jq}=①	$i+p-1=j+p-1$ & $i+q-1=j+q-1$ ➡ $i=j$	불능
case ii	N_{ip}=①N_{iq}=①N_{jp}=①N_{jq}=②	$i+p-1=j+p-1$ & $i+q-1=j+q-1-k$ ➡ $i=j$ & $i=j-k$	불능
case iii	N_{ip}=①N_{iq}=①N_{jp}=②N_{jq}=②	$i+p-1=j+p-1-k$ & $i+q-1=j+q-1-k$ ➡ $i=j-k$	불능
case iv	N_{ip}=①N_{iq}=②N_{jp}=①N_{jq}=②	$i+p-1=j+p-1$ & $i+q-1-k=j+q-1-k$ ➡ $i=j$	불능
case v	N_{ip}=①N_{iq}=②N_{jp}=②N_{jq}=②	$i+p-1=j+p-1-k$ & $i+q-1-k=j+q-1-k$ ➡ $i=j-k$ & $i=j$	불능
case vi	N_{ip}=②N_{iq}=②N_{jp}=②N_{jq}=②	$i+p-1-k=j+p-1-k$ & $i+q-1-k=j+q-1-k$ ➡ $i=j$	불능

일어나지 않고, 이를 통해 GNS 시퀀스는 confounding 고장 증후를 발생시키지 않음을 알 수 있다.

Ⅳ. 성능 평가

이번 장에서는 앞에서 살펴본 기존의 알고리듬들과 GNS 시퀀스를 테스트 수행 시 핵심적인 조건이 되는 고장의 검출 및 분석의 가능성과 입력되는 테스트 벡터의 길이를 통해 비교한다. 이는 연결 테스트로서의 기능을 완벽하게 수행해 내면서 동시에 가능한 짧은 테스트 수행 시간을 요구하는 테스트 알고리듬의 특성에 따른 비교이다.

먼저 고장의 검출 및 고장 위치의 분석의 가능성을 보면, 앞서 언급했던 기존의 알고리듬들 중에서 계수 시퀀스나 완전 독립 테스트 집합은 고착 고장의 문제는 해결할 수 있지만, 이를 제외한 단락 고장과 관련된 aliasing 고장 증후 및 confounding 고장 증후의 발생과 관련된 문제는 전혀 해결하지 못한다. 물론 이들은 테스트 길이가 대단히 짧다는 큰 장점이 있지만, 기본적으로 완전한 테스트를 수행하지 못하는 테스트 알고리듬이므로 테스트 길이의 논의에서는 제외한다. 나머지 기존의 알고리듬들인 워킹 원 시퀀스 및 분할 그룹 워킹 시퀀스, 그리고 GNS 시퀀스는 앞서 나열한 고장과 관련된 모든 문제를 해결한다. 워킹 원 시퀀스는 명확하게 모든 문제를 해결하고 있으며, 분할 그룹 워킹 원 시퀀스와 GNS 시퀀스는 모두 워킹 원 시퀀스를 변형하여 문제를 해결하고 있다. 결국 이들은 자가 분석(self-diagnosis)이 가능한 테스트 알고리듬이다. 자가 분석이란 인가된 테스트 패턴에 대한 SRV만을 관찰함으로써 회로의 고장 여부를 확인할 수 있는 것이다. 앞서 설명한 aliasing 고장 증후 및 confounding 고장 증후는 이러한 자가 분석을 불가능하게 만드는 요소들이고, 따라서 이 문제를 해결한 테스트 알고리듬만이 자가 분석이 가능한 것이다. 결국 앞서 제시한 바와 같이 연결상의 고장 유무를 완전하게 검출하고 분석할 수 있는 테스트 알고리듬에 대해서 테스트 길이를 비교해 본 결과는 <표 6>과 같다. 각 테스트 알고리듬에 대해 표시된 수는 전체 네트 수에 대해 연결 테스트 수행에 필요한 테스트 길이를 나타내고, 백분율로 표시된 부분은 워킹 시퀀스를 100%로 놓고, 이에 대한 다른 테스트 시퀀스를 비교해서 백분율로 나타낸 것이다. 전체 네트의 수를 n이라고 할 때 각각의 테스트 벡터의 길이는 워킹 시퀀스의 경우 존재하는 네트 수와 같은 n이고, 분할 그룹 워킹 시퀀스는 $4k-1$(또는 $4k-2$)이다. 또한 GNS 시퀀스는 앞서 설명한 바와 같이 $3k$(또는 $3k-1$)이다. 이때 분할 그룹 워킹 시퀀스에서 적용되는 $k=\lceil\sqrt{n}\rceil$로써 GNS 시퀀스에서의 k와 같은 크기이다. <표 6>에서 보는 바와 같이 자가 분석이 가능한 워킹 시퀀스, 분할 그룹 워킹 시퀀스, GNS 시퀀스의 테스트 길이를 비교해 보면 워킹 시퀀스는 나머지 두 테스트 알고리듬에 비해서 훨씬 긴 테스트 길이를 가지고 있다. 물론 전체 네트가 10인 아주 작은 연결의 경우는 오히려 워킹 시퀀스가 더 짧은 테스트 길이를 갖기만, 고집적화된 SoC 환경에서 이는 의미가 없다고 할 수 있다.

표 6. 각 알고리듬의 테스트 길이
Table 6. Test lengths of each test algorithm.

네트수 / 알고리듬	10	100	500	1000	5000	10000
워킹 시퀀스	10	100	500	1000	5000	10000
	100%	100%	100%	100%	100%	100%
분할 그룹 워킹 시퀀스	12	39	91	124	283	399
	120%	39%	18.2%	12.4%	5.66%	3.99%
GNS 시퀀스	11	30	68	96	212	300
	110%	30%	13.6%	9.6%	4.24%	3%

또한 워킹 시퀀스와 나머지 두 테스트 알고리듬의 테스트 길이는 전체 네트의 수가 증가할수록 그 차이가 상대적으로 더 커진다. 또한 분할 그룹 워킹 원 시퀀스와 GNS 시퀀스를 비교해 보았을 때, 백분율만으로 보면 별 차이가 없는 듯 하지만, 테스트 길이 즉 필요한 PTV의 개수를 비교하면 상당한 차이가 있음을 확인할 수 있다. 따라서 GNS 시퀀스는 나머지 두 가지 알고리듬에 비해 훨씬 짧은 테스트 길이만으로 연결 테스트를 수행할 수 있는 것이다.

V. 결 론

SoC 환경은 고도의 집적 기술을 기반으로 하여 칩의 크기와 전력 소비를 모두 줄여 나가는 최첨단 기술의 집약체이다. 따라서 이를 테스트하는 문제 역시 새로운 기준을 필요로 하고 있으며, 이중 내부 IP들 간의 연결을 테스트하는 문제 역시 오로지 보드상의 연결만을 바라보던 과거와는 달리 칩 내부의 연결이 더욱 복잡해지고 이러한 연결의 고장이 곧 칩의 고장으로 이어지는 SoC 환경에서 그 중요성이 증대되었다고 할 수 있다. 이러한 연결 테스트를 수행하기 위한 알고리듬은 고장의 유무를 검출해내고 그 고장의 위치를 분석해낼 수 있는 성능을 충실하게 수행하면서 동시에 가능한 짧은 테스트 길이를 가지도록 설계되어야 한다. 왜냐하면 짧은 테스트 길이는 단지 테스트 수행 시간의 단축만을 의미하는 것이 아닌 테스트 수행 비용의 감소를 의미하는 것이기 때문이다. 연결 테스트와 관련된 모든 문제를 완벽하게 해결해 내는 알고리듬인 워킹 시퀀스의 치명적인 단점이 여기에 있다. 이에 더해서 테스트 알고리듬은 결국 하드웨어로 구현하는 것이 목적이므로 하드웨어 설계가 편리한 규칙성 있는 구조를 가지는 것이 좋다.

본 논문에서 제안하는 GNS 시퀀스는 위에서 제시한 조건들을 효과적으로 수행해 내는 알고리듬으로써, 모든 네트를 규칙적인 형태의 그룹으로 나누어 테스트 벡터를 가해준다. 이때 가하는 테스트 벡터는 세 부분의 변형된 워킹 원 시퀀스의 형태를 가진 그룹별 워킹 시퀀스, 네트별 워킹 시퀀스, 네트별 이동 워킹 시퀀스로 구성되어 있다. 이 세 부분의 워킹 시퀀스는 각각의 STV에 대해서 워킹 원 시퀀스를 가지고 있으므로, 입력되는 STV들은 각 시퀀스마다 하나씩의 "1" 값을 가지고 있게되어 한 네트에 입력되는 전체 STV는 각각 세 개씩의 "1" 값이 서로 독립적으로 분포되도록 설계되어 있다. 따라서 고착 고장 및 단락 고장을 모두 검출해 낼 수 있으며, 고장 위치의 분석과 관련된 aliasing 고장 증후와 confounding 고장 증후를 발생시키지 않음으로써 성능 면에서 고장의 검출 및 위치 분석이 완벽하게 가능한 워킹 시퀀스와 동일한 결과를 얻어낼 수 있다. 이에 더해서 GNS 시퀀스는 테스트 길이가 워킹 원 시퀀스를 기준으로 볼 때 네트 수의 증가에 비례해서 상대적으로 더욱 크게 감소함으로써 테스트 수행시간을 크게 감소시킬 수 있다. 결국 GNS 시퀀스는 복잡한 여러 가지의 테스트를 동시에 수행하는 SoC 환경에서 연결 테스트를 수행함에 있어 고장의 검출 및 분석을 충실히 수행해 내면서도 테스트 수행 시간을 크게 단축시키는 알고리듬으로써 사용될 수 있을 것으로 기대된다.

참 고 문 헌

[1] Y. Zorian, S. Dey, M. J. Rodgers, "Test of Future System-on-Chips", Proc. IEEE/ACM International Conference, pp. 392~398, 2000.

[2] C. Su, W. Tseng, "Configuration Free SoC Interconnect BIST Methodology", Proc. International Test Conference, pp. 1033~1038, 2001.

[3] B. I. Dervisoglu, "A Unified DFT Architecture for use with IEEE 1149.1 and VSIA/IEEE P1500 Compliant Test Access Controllers", Proc. Design Automation Conference, pp. 53~

58, 2001.

[4] IEEE. S. 1149.1-1990, "IEEE Standard Test Access Port and Boundary Scan Atchitecture", IEEE, 1989.

[5] K. P. Parker, "The Boundary-Scan Handbook", Kluwer Academic Publishers, 1992, pp. 1~41, 118~129.

[6] E. J. Marinissen, S. K. Goel, M. Lousberg, "Wrapper Design for Embedded Core Test", Proc. International Test Conference, pp. 911~920, 2001.

[7] H. Bleeker, P. V. D. Eijnden, F .D. Jong, "Boundary-Scan Test : A practical Approach," Kluwer Academic Publishers, 1993, pp.123~172.

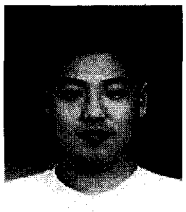
[8] 김현진, 신종철, 강성호, "회로 기판상의 연결 테스트에 대한 분할 그룹 워킹 시퀀스," Trans. KIEEE. Vol.47. No.12, pp. 2251~2257, 1998

[9] W. K. Kautz, "Testing of Faults in Wiring Interconnects," IEEE Trans. Computers, Vol. C-23, No. 4, April, pp. 358~363, 1974.

[10] Jung-Chen Lieu and Melvin A. Breuer, "Maximal Diagnosis for Wiring Networks," Proc. International Test Conference, pp.96~105, 1991.

[11] W-t. Cheng, J. L. Lewandowski, and E. Wu, "Diagnosis for Wiring Interconnect," Proc. International Test Conference, pp 565~571, 1990.

저 자 소 개



金 容 準(正會員)

2002년 2월 연세대학교 전기공학과 졸업. 현재 연세대학교 전기전자공학과 석사과정

姜 成 昊(正會員)

1986년 2월 서울대 공대 제어계측공학과 졸업. 1988년 5월 The University of Texas at Austin 전기 및 컴퓨터공학과 졸업(석사). 1992년 5월 The University of Texas at Austin 전기 및 컴퓨터공학과 졸업(공박). 미국 Schlumberger 연구원. Motorola 선임 연구원. 현재 연세대학교 공과대학 전기전자공학과 부교수