Exhaustive 테스트 기법을 사용한 효율적 병렬테스팅

(An Efficient Parallel Testing using The Exhaustive Test Method)

김 우 완 †

(Wu-Woan Kim)

요 약 최근 몇 년 동안 디지털 시스템이 복잡성은 아주 빠르게 증가하고 있다. 비록 반도체 제조업자들이 제품에 대한 신뢰성을 높이려고 노력하고 있지만 어느 때에 시스템이 어딘가에서 결함이 발생할 것이라는 것을 알기는 불가능하다. 이렇듯이 회로가 복잡화함에 따라 테스트 생성(test generation)에 대한 잘 정리되어 있고 자동화된 방법이 필요하게 되었다. 하지만 현재 광범위하게 사용하고 있는 방법중 대부분은 한번에 하나씩의 패턴만을 넣어서 처리하는 방식이다. 이는 각각의 결함에 대해서 탐색하는데 많은 시간을 낭비하게 된다. 본 논문에서는 Exhaustive 방법을 사용하는 테스트 패턴 생성 방법 중에서 분할 기법을 적용하여 테스트 패턴을 생성한다. 또한 이 패턴을 이용하여 병렬로 패턴을 삽입함으로써 더욱 빠르게 결함을 발견할 수 있는 방법을 설계 및 구현한다.

키워드 : Exhaustive 테스트, 병렬테스팅, 회로분할

Abstract In recent years the complexity of digital systems has increased dramatically. Although semiconductor manufacturers try to ensure that their products are reliable, it is almost impossible not to have faults somewhere in a system at any given time. As complexity of circuits increases, the necessity of more efficient organized and automated methods for test generation is growing. But, up to now, most of popular and extensive methods for test generation may be those which sequentially produce an output for an input pattern. They inevitably require a lot of time to search each fault in a system. In this paper, corresponding test patterns are generated through the partitioning method among those based on the exhaustive method. In addition, the method, which can discovers faults faster than other ones that have been proposed ever by inserting a pattern in parallel, is designed and implemented.

Key words : Exhaustive Test, Parallel Testing, Circuit Partitioning

1. 서 론

최근 몇 년 동안 디지털 시스템이 복잡성은 아주 빠르게 증가하고 있다. 비록 반도체 제조업자들이 제품에 대한 신뢰성을 높이려고 노력하고 있지만 어느 때에 시스템이 어딘가에서 결함이 발생할 것이라는 것을 알기는 불가능하다[1]. 이렇듯이 회로가 복잡해짐에 따라 테스트 생성(test generation)에 대한 잘 정리되어 있고 자동화된 방법이 필요하게 되었다. 테스트 생성(test generation)에 대한 많은 접근들은 게이트 단계(gate level)를 묘사하는 디자인에서 탐색 방법을 기반으로 하는 방법을 사용하고 있다[2].

VLSI 회로에 제공되는 이들 방법의 높은 복잡성은 디자인에서 쉽게 생각할 수 있는 통합된 테스트 환경을 요구하게 하였고, 이를 위해 함수 단계(functional level) 테스트 생성에 관한 연구에 많은 관심을 가지게 된다[2]. 게이트 단계 기법과 광범위한 Stuck-at 결함 모델과는 달리 이들을 함수적으로 테스트 생성 방법은 물리적 결함에 적용할 수 있는 광범위한 결함 모델이 부족하다. 또한

· 본 연구는 2003학년도 경남대학교 학술논문게재연구비 지원으로 이루어 졌음.

† 정 회 원 : 경남대학교 정보통신공학부 교수
wukim@zeus.com.kyungnam.ac.kr

논문접수 : 2002년 3월 11일
심사완료 : 2003년 1월 16일

함수 테스트 생성과 결함 시뮬레이션으로부터 얻을 수 있는 테스트의 결과는 각각의 방법에 따라서 아주 다르게 나타난다는 것이다[3]. 많은 함수적인 테스트 방법 중에 Exhaustive 테스트 생성 기술은 자기 테스트 어플리케이션에 사용되는 Pseudorandom *Built-In* 패턴 생성기에 대한 대안으로 생각되어 지고 있다. 이 방법은 회로내의 모든 조합된 결함을 커버하며 완전한 테스트 집합에 대해 적은 정보를 요구한다[4,5]. 일반적으로 Exhaustive 테스트 생성 방법에서는 회로를 조각 조각으로 나누어 블럭 단위로 표현하는 분할 기법을 사용한다. 그리고 각각의 회로 조각들은 입력에 대한 k 비트 부분 공간(subspace)으로 표현되며 k 비트 부분 공간에 대해서 전부 테스트 하기 위해서는 각각의 부분공간에 대해서 2^k 개의 이진 패턴이 나오게 된다[4,6,7,8]. 모든 k 비트 패턴에 대한 2^k개의 테스트 벡터의 생성에 대한 것은 폭넓게 연구되고 있으며 많은 제안들이 나오고 있다. 하지만 이러한 테스트 생성 방법도 LSI와 VLSI칩의 복잡한 회로에 대해서는 처리하기가 어렵다. 모든 단일 Stuck-at 결함과 몇몇 복합적인 stuck-at 결함을 검출하기 위한 테스트 집합의 생성에 요구되어지는 계산 시간은 회로가 크고 복잡해질수록 엄청나게 커진다.

본 논문에서는 Exhaustive 테스트 생성 방법을 이용하되 서로 영향을 미치지 않는 회로내의 결함들 사이의 관계를 분석하여 패턴을 생성함으로써 병렬적으로 테스트할 수 있는 방법을 제안하였다. 제 2장에서는 기본적인 결함의 종류와 결함을 찾는 방법들을 소개하며 제 3장에서는 본 논문에서 사용하는 테스트 패턴 생성 및 적용 방법을 보여준다. 제 4장에서는 이를 이용한 시뮬레이션 프로그램을 구현하고 마지막 제 5장에서는 결론 및 향후 연구과제에 대해서 논한다.

2. 관련 연구

고장(failure)은 시스템에서 명시된 동작에서 벗어나는 일이 발생함을 의미하며 결함은 고장의 유무와 상관없이 물리적인 결점이 있음을 말한다[1,9]. 결함의 특징은 크게 성질(nature), 값(value), 범위(extent), 기간(duration)등으로 나눌 수 있다. 결함은 그 성질에 따라서 논리적(logical)결함과 비논리적(non-logical)결함으로 분류할 수 있다. 논리적 결함은 회로내의 어느 시점에서 논리 값이 상술된 값과 정반대가 되는 값을 발생시키는 결함을 말하며 비논리 결함은 클럭 신호, 전원 고장 등과 같은 기능 불량 등의 결함을 포함한다. 회로내의 어느 시점에서 논리 결함의 값은 논리적 값이 고정적으로 생기는지 또는 계속적으로 변하면서 생성되는지를 가리킨다. 결함의 범위는 결함의 영향이 지역적인지 또는 분산적인지를 설명하고 결함의 기간은 결함이 영구적 또는 일시적인지를 가리킨다.

2.1 부울 미분

부울 미분(Boolean Difference)의 기본 원리는 두 개의 부울 표현식에서 파생된다. 하나는 일반적인 결함이 없는 회로의 상태를 표현하고 또 다른 하나는 단일 s-a-1 또는 s-a-0 결함 상태를 가정하여서 논리적인 상태를 표현한다. 이 두 개의 표현은 Exclusive-OR로 되어 있다. 만약 결과가 1이면 결함이 있다는 것을 가리킨다[10,11].

$F(X) = F(x_1, \ldots, x_n)$를 n 변수의 논리 함수라고 하자. 만약 논리 함수의 입력 중 하나가 결함이 있고 출력은 $F(x_1, \ldots, \overline{x_i}, \ldots, x_n)$가 된다. x_i에 대해서 부울 미분 F(X)는 다음과 같이 정의할 수 있다.

$$\frac{dF(x_1, \ldots, x_i, \ldots, x_n)}{dx_i} = \frac{dF(X)}{dx_i} = F(x_1, \ldots, x_i, \ldots, x_n) \oplus F(x_1, \ldots, \overline{x_i}, \ldots, x_n)$$

함수 $dF(x)/dx_i$는 x_i에 대한 부울 미분 F(X)라고 한다. 우리는 $F(x_1, \ldots, x_i, \ldots, x_n) \neq F(x_1, \ldots, \overline{x_i}, \ldots, x_n)$일 때 $dF(X)/dx_i = 1$이고, $F(x_1, \ldots, x_i, \ldots, x_n) = F(x_1, \ldots, \overline{x_i}, \ldots, x_n)$일 때 $dF(X)/dx_i = 0$ 임을 쉽게 알 수가 있다. x_i에서 결함을 검출하기 위해서는 x_i가 $\overline{x_i}$로 변환될 때마다 입력 조합을 찾을 필요가 있다. 이때 $F(x_i, \ldots, x_i, \ldots, x_n)$는 $F(x_1, \ldots, \overline{x_i}, \ldots, x_n)$와 다를 것이다. 다시 말하면 우리가 찾고자 하는 것은 $dF(X)/dx_i = 1$과 같은 x_i에서 발생하는 각각의 결함에 대한 입력 조합을 찾는 것이다.

부울 미분에 대한 유용한 속성을 보면 다음과 같다.

$$\frac{d\overline{F(X)}}{dx_i} = \frac{dF(X)}{dx_i} : \overline{F(X)}\text{는 } F(X)\text{의 보수} \quad (1)$$

$$\frac{dF(X)}{dx_i} = \frac{d\overline{F(X)}}{dx_i} \quad (2)$$

$$\frac{d}{dx_i} \cdot \frac{dF(X)}{dx_j} = \frac{d}{dx_j} \cdot \frac{dF(X)}{dx_i} \quad (3)$$

$$\frac{d[F(X)G(X)]}{dx_i} = F(X)\frac{dG(X)}{dx_i} \oplus G(X)\frac{dF(X)}{dx_i} \oplus \frac{dF(X)}{dx_i} \cdot \frac{dG(X)}{dx_i} \quad (4)$$

$$\frac{d[F(X)+G(X)]}{dx_i} = \overline{F(X)}\frac{dG(X)}{dx_i} \oplus \overline{G(X)}\frac{dF(X)}{dx_i} \oplus \frac{dF(X)}{dx_i} \cdot \frac{dG(X)}{dx_i} \quad (5)$$

만약 $F(x_1, \ldots, x_i, \ldots, x_n) = F(x_1, \ldots, \overline{x_i}, \ldots, x_n)$ 이라면 부울 함수 F(X)는 x_i에 대해 독립적이라고 한다. 이것은 x_i에서 발생하는 결함이 마지막 출력 $F(X)$에 영향을 주지 않음을 의미하고 $dF(x)/dx_i = 0$ 임을 말한다. 다음은 식 (1)-(5)을 이용해서 더 추가되는 식들이다.

$F(X)$가 x_i에 독립이라면, $\frac{dF(X)}{dx_i} = 0$ (6)

$F(X)$가 오로지 x_i에 의존적이라면, $\frac{dF(X)}{dx_i} = 1$ (7)

$F(X)$가 x_i에 독립적이라면,

$$\frac{d[F(X)G(X)]}{dx_i} = F(X)\frac{dG(X)}{dx_i} \quad (8)$$

$F(X)$가 x_i에 독립적이라면,

$$\frac{d[F(X)+G(X)]}{dx_i} = \overline{F(X)}\frac{dG(X)}{dx_i} \quad (9)$$

위의 속성을 가지고 x_i에서 s-a-0을 체크한다면 $x_i\frac{dF(X)}{dx_i} = 1$을 만족하는 값을 말하며 x_i에서 s-a-1을 체크한다면 $\overline{x_i}\frac{dF(X)}{dx_i} = 1$을 만족하면 된다.

2.2 Exhaustive 테스팅

Exhaustive 와 Pseudoexhaustive 테스트 생성 기술은 자기 테스트 어플리케이션(self-test applicatio n)에서 사용되는 Pseudorandom Built-In 패턴 생성을 위한 중요한 대안처럼 인식되고 있다[4,6]. 이것은 회로내의 모든 조합 결함을 커버하고 완전한 테스트 결함의 특성에 대해 매우 적은 정보만을 요구한다. 함수 단계 테스트 생성(Functional-level test generation)에서는 고급 수준(higher- level)결함 모델에 대해서만 논의되어 왔고, 일반적으로 시스템에서의 내부 구조에 대한 어떠한 정보도 사용하지 않았다. 그러나 이러한 경우에 있어서는 고급 레벨의 결함 모델을 파생하는 방법에 대해서는 명확하지 않다. 많은 경우에 있어서 모듈에 대해 자세하게 알 수 없을지 몰라도 회로의 구조는 알 수 있다. VLSI 회로에 있어서 모듈은 일반적으로 정규적인 회로 내에 서로 연결되어 있다. 규칙적인 구조는 간단한 테스트에 의해서 개발할 수 있다[12,13]. 일반적으로 이러한 Pseudo-exhaustive 접근 방법에는 회로에 포함되어 있는 모듈이 철저한 테스트를 할 수 있게 하기 위해서 회로를 분할하는 방법을 사용하고 있으며, Exhaustive 테스트로부터 출력 값이 단일 신드롬(single syndrome)으로 축약할 수 있게 하기 위해서 회로를 디자인하는 방법에 대해서도 논의되고 있다[14,15,16].

3. Exhaustive 테스트 패턴을 이용한 병렬 처리

이번 장에서는 본 논문에서 제안하고 있는 패턴 생성 방법과 이를 이용하여 병렬로 테스트 패턴을 처리하는 방법에 대해 논하기로 한다. 3.1절에서는 Exhaustive 테스팅을 위해 분할을 어떻게 할 것인가에 대해서 이야기를 하고 3.2에서는 분할된 각각의 조각들에 대한 테스트 패턴을 생성하는 방법을 설명하며 마지막 3.3에서는 이 패턴을 이용하여 병렬로 테스트하는 방법에 대해서 논한다.

3.1 Exhaustive 테스트를 위한 분할

Exhaustive 와 Pseudoexhaustive 테스트 생성 기술은 자기 테스트 어플리케이션(self-test application)에서 사용되는 Pseudorandom Built-In 패턴 생성을 위한 중요한 대안처럼 인식되고 있다. 이것은 회로내의 모든 조합 결함을 커버하고 완전한 테스트 결합의 특성에 대해 매우 적은 정보만을 요구한다. 비록 n개의 입력을 갖는 조합 회로일지라도 Exhaustive 테스트를 위해서는 2^n개의 테스트 패턴이 요구된다. 그러므로 n이 엄청나게 커진다면 실행이 불가능하게 된다. 많은 경우에 있어서 테스트하고 있는 회로의 출력은 오로지 첫 번째 입력들의 부분 집합(subset)에 의존적이다. 일반적으로 Pseudoexhaustive 접근 방식은 분할 기법(partitioning technique)을 가지고 회로를 분할하여서 각각을 회로의 조각으로 간주하여 테스트한다.

여기서 사용하는 방법도 병렬 처리를 위하여 하나의 결함이 다른 결함에 영향을 미치지 않는 상호배제적인 방법을 가지고 회로를 분할하였다. 본 논문에서는 가장 기본적인 방법인 게이트 하나 하나를 분할하는 방법을 사용하였다. 이는 분할 기법에 있어서 가장 기본적인 방식이므로 이를 택하였다.

예로 사용할 회로는 그림 1에서와 같이 병렬 로드와 동기 클리어, 인크리먼트 기능을 가진 4비트 이진 레지스터이다. 이 회로는 점선 부분에서의 일부분이 계속적으로 반복되는 모습을 보여주고 있다. 이에 대해서 패턴을 구해 보면 역시 유사하게 나오는 것을 볼 수 있다. 그러므로 여기서는 점선 부분만을 가지고 테스트 벡터를 생성하고 테스트할 것이다. 점선 부분을 분리하여 우리가 테스트할 회로를 만들어 보면 그림 2에서 보는 것과 같다. 먼저 각각의 게이트를 기준으로 분할을 하므로 각각의 게이트를 종류별로 나누어 보면 먼저 인버터(inverter)와 2 입력 AND게이트 , 3 입력 AND게이트 , 2 입력 OR게이트 , 3 입력 OR게이트가 있다.

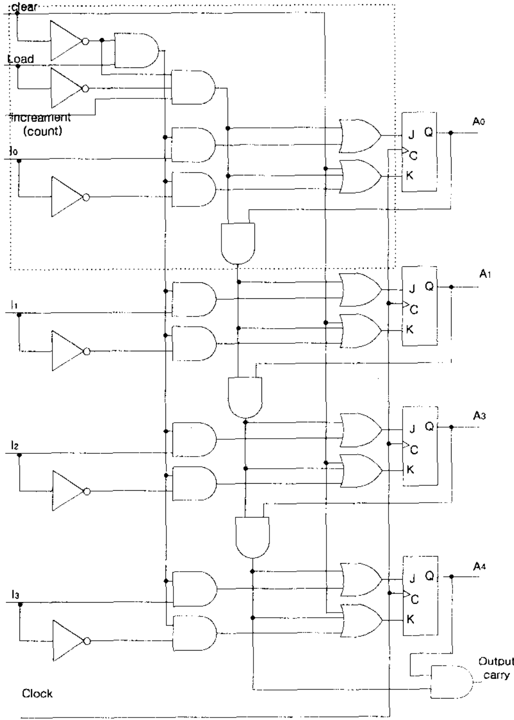

그림 1 4 비트 이진 레지스터

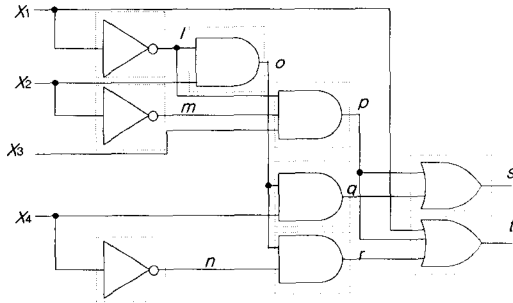

그림 2 분할의 예

분할의 장점은 모든 테스트 패턴을 처리하는데 있어서 중간위치에서 테스트 할 수 있어서 테스트 시간을 줄일 수 있다는 장점과 각각의 분할된 영역에서 보면 상호간의 결함이 영향을 미치지 않는 부분이 생긴다는 것이다. 이로 인해서 병렬로 테스트 패턴을 입력할 수 있고 중간결과를 가지고 다음 처리에도 사용할 수 있으므로 상당한 효율성을 가진다. 하지만 테스트 포인트가 많아지면 테스트할 위치가 많아지므로 회로가 복잡해 질 수 있다. 그러므로 분할은 어느 정도의 크기를 가지고 분할을 할 것인가 하는 것도 많은 연구 과제중 하나이다. 본 논문에서는 분할에 있어서 가장 기본적인 방법인 게이트 단위로 분할을 하였다. 그림 2의 회로에서 보여주는 것과 같이 게이트 단위로 분할할 경우 4번의 패스만을 거치면 전체회로의 결함을 계산할 수 있음을 알 수 있다. 간단한 예를 들면, 먼저 첫 번째로 패스가 짧은 것은 l, m, n 이다. 이 위치에서는 x_1, x_2, x_4가 어떠한 결함이 발생하더라도 서로에게 영향을 미치지 않는다. x_1이 s-a-1이라고 하자. 이 값은 m, n에 어떠한 영향도 미치지 않는다. 그러므로 이 3개의 결함을 동시에 구할 수가 있다. 또한 이 위치에서 결함이 없다면 이 값을 가지고 다음 결함을 찾을 수 있는 부분은 o, p가 있다. 이 위치에서는 또한 x_3, l, m위치의 결함을 찾을 수가 있다. 이러한 방식을 이용하여 여러개의 결함을 병렬적으로 찾아 나갈 수 있다.

3.2 부울 미분을 이용한 테스트 벡터 생성

부울 미분의 기본 원리는 두 개의 부울 표현식에서 파생된다. 하나는 일반적인 결함이 없는 회로의 상태를 표현하고 또 다른 하나는 단일 s-a-1 또는 s-a-0 결함 상태를 가정하여서 논리적인 상태를 표현한다. 이 두 개의 표현은 Exclusive-OR로 되어 있다. 만약 결과가 1이면 결함이 있다는 것을 가리킨다. 그림 2의 예를 가지고 먼저 생각해 보자. 만약 x_1에서 s-a-0가 있다고 가정하고 부울 미분을 이용해서 벡터 테이블을 만들어 본다. 출력 S를 가지고 생각하면 부울 미분은 다음과 같다.

$$F(S) = \overline{x_1}\,\overline{x_2}x_3 + \overline{x_1}x_2x_4$$

$$\frac{dF(S)}{dx_1} = \frac{d(\overline{x_1}\,\overline{x_2}x_3 + \overline{x_1}x_2x_4)}{dx_1}$$

$$= \frac{d(\overline{x_1}(\overline{x_2}x_3 + x_2x_4))}{dx_1}$$

$$= \overline{x_1}\frac{d(\overline{x_2}x_3 + x_2x_4)}{dx_1} \oplus (\overline{x_2}x_3 + x_2x_4)\frac{d\overline{x_1}}{dx_1}$$

$$\oplus \frac{d(\overline{x_2}x_3 + x_2x_4)}{dx_1}\frac{d\overline{x_1}}{dx_1} \quad (\text{식 4 이용})$$

$$= (\overline{x_2}x_3 + x_2x_4)$$

위의 결과를 이용해서 s-a-0을 구하기 위해서는 $x_1 \frac{dF(S)}{dx_1} = 1$ 이 되어야 한다. 그러므로 x_1는 1 이고 x_2가 1 이면 x_3의 값은 이 벡터에 아무런 영향이 주지 않으므로 1과 0 둘 모두 올 수 있으며, x_4는 1 이 되어야 한다. 또는 x_2가 0 이면 x_3는 1이 되어야 하고, x_4는 어떠한 값이 와도 상관이 없다. 그러므로 테스트 벡터는 다음과 같이 {1011},{1011},{1101},{1111} 4개가 올 수가

있다. 그리고 이 회로에 대한 모든 테스트 벡터는 상기의 방법으로 구할 수 있다. 그러나 우리는 이 모든 것을 구하는 것이 아니라 각각의 게이트에 대해서 테스트 벡터를 구하고 나머지는 이를 이용해서 테스트 벡터를 자동적으로 구할 것이다. 앞에서도 언급했듯이 우리가 구할 테스트 벡터는 인버터와 두 입력 AND게이트 3입력 AND게이트와 2 입력 OR게이트 3 입력 OR게이트이며, 이에 대한 각각의 테스트 벡터는 표 1과 같다.

표 1 게이트 별 stuck-at 테스트 벡터

		s-a-0	s-a-1
인버터	x_1	{1}	{0}
2입력 AND	x_1	{1,1}	{0,1}
	x_2	{1,1}	{1,0}
3입력 AND	x_1	{1,1,1}	{0,1,1}
	x_2	{1,1,1}	{1,0,1}
	x_3	{1,1,1}	{1,1,0}
2입력 OR	x_1	{1,0}	{0,0}
	x_2	{0.1}	{0,0}
3입력 OR	x_1	{1,0,0}	{0,0,0}
	x_2	{0,1,0}	{0,0,0}
	x_3	{0,0,1}	{0,0,0}

표 1에는 각각의 게이트에 대한 s-a-0 와 s-a-1이 나와 있다. 이제 이 값을 이용하여 테스트를 생성할 것이다.

3.3 Exhaustive 테스트 패턴 생성

그림 2 회로에서와 같이 분할은 게이트단위로 분할로 실행하였다. 그러므로 각각의 검출 포인트는 $l, m, n, o, p, q, r, s, t$ 가 된다. 분할이 되어 있으므로 서로 영향을 받지 않는 결함에 대해서는 병렬로 동시에 값을 넣을 수가 있다.

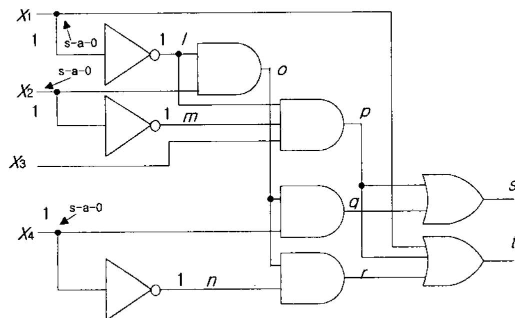

그림 3 다중 stuck-at의 예

예를 들어 가장 짧은 패스를 선택해 보자. 가장 짧은 패스는 l, m, n이 된다. 이 포인트에 입력은 각각 x_1, x_2와 그리고 x_4이다. 또한 3개의 값은 서로에게 영향을 미치지 않는다. 그러므로 3개의 결함을 구할 수 있다. 그림 3은 이러한 것을 보여주는 예이다. 만약 벡터로 $x_1x_2x_3x_4$ ={11-1} 값이 주어졌고 l, m, n이 각각의 값에 1의 값이 나왔다면 모두가 s-a-0라는 것을 알 수가 있다. 이러한 점을 이용했을 때 다중 결함에 대해서도 처리가 가능할 수가 있다. 또한 이 값이 이상이 없을 경우 이 값을 이용하여 다음 값인 o, p값을 이용하여 l, m의 위치에 대한 결함을 찾을 수 있다. 이렇게 함으로써 빠르게 결함을 찾을 수 있다. 우리는 여기에 필요한 테스트 패턴을 기본적인 게이트의 결함을 가지고 벡터를 만들고 나서 이 값을 이용하여 가장 가까운 패스내의 게이트와 결함이 가능한 입력 값을 결합하여 가능한 결함 벡터를 구하고 이 값을 이용하여 테스트한다.

3.4 병렬 처리

지금까지 그림 2 와 3에서와 같이 병렬 처리를 위한 테스트 패턴을 만드는 방법에 대해서 설명했다. 지금부터 이를 병렬로 처리하는 방법에 대해서 논하기로 한다. 먼저 각 포인트 위치에서 값을 검출한다. 검출한 값과 에러가 없는 회로에서의 값과 비교 서로 다르면 결함이 있는 것으로 간주한다. 이 회로는 간단히 XOR 회로에 의해서 구현이 될 수 있다. 먼저 1차 적으로 가장 가까운 패스를 찾고 이에 대한 값을 가져온다. 그리고 이 값과 에러가 없는 회로와 XOR게이트를 이용하여 비교하고 만약 정상이면 그 다음 짧은 패스를 찾는다. 그리고 이에 대한 값의 포인트 위치의 값과 에러가 없는 회로의 값과 비교한다. 이 값은 이전에 검사한 패스에 대해서는 모두 이상이 없다는 것이므로 이 값을 이용하여 계산할 수 가 있다. 계속 적으로 이러한 과정을 거쳐서 계산 해 나갈 것이다.

4. Exhaustive 테스트 패턴을 이용한 병렬 테스트 구현

이 장은 지금까지의 이론을 가지고 실제로 테스트 패턴을 생성하고 처리하는 프로세스를 구현하는 부분이다. 일반 PC를 이용하여 시스템을 구현하였으며, 구현 환경은 표 2와 같다.

먼저 3장에서 보여주었던 회로에서 분할된 게이트의 패턴을 구한 후, 이 패턴을 이용하여 패턴을 생성하는 부분에 대해 4.1에서 설명할 것이다. 이 처리가 끝난 후 각각의 위치에 대한 가상적인 stuck-at 결함을 만들고 이

를 찾는 프로시저 구현 부분을 4.2에서 논한다.

표 2 시스템 구현 환경

PC 환경	내용
CPU	333MHz
메모리	64M
운영체제	Windows98
구현언어	Visual C++

4.1 패턴 생성 프로시져 구현

그림 4에서 입력 패턴은 이 프로그램이 수행과 동시에 본 회로에서 각각의 인버터, 두 입력 AND게이트, 3 입력 AND게이트, 2 입력 OR게이트 , 3 입력 OR게이트에 대한 각각의 결함에 대한 패턴이 자동으로 생성되고 나서 각각의 패턴을 이용하여 왼쪽위치에 해당 결함을 찾을 수 있는 최적의 벡터를 표시해 준다. 이를 찾는 루틴은 앞에서도 언급하였듯이 각각의 최단 패스를 먼저 설정하고 이 패스의 입력을 본다. 이 입력이 결함이 생길 수 있는 패턴을 게이트별 결함 패턴에서 가져와서 이를 이용하여 입력 단에 입력시키고 여기에 최대한 최적화된 패턴을 구한다.

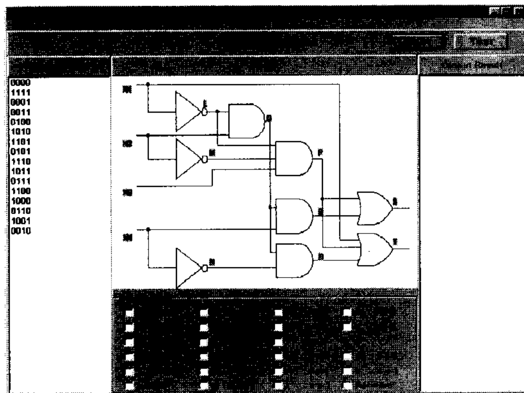

그림 4 Exhaustive 입력 패턴 생성

예를 들면 l, m, n 위치에서 각각의 패턴을 입력받아 3개가 동시에 생길 수 있는 패턴을 찾아낸다. 이 경우 패턴은 s-a-0에 대해서는 1101과 1111이 나올 것이다. 즉, 이 패턴이 입력 패턴으로 선택되게 된다. 그리고 다음 최단 패스에 위치한 o, p위치에서 또한 패턴을 찾게 된다. 둘 다 동시에 발생할 수 있는 패턴을 찾는다. 이렇게 해서 각각의 위치에서 동시에 찾을 수 있는 패턴을 구하여 입력 패턴 부분에 입력된다.

4.2 패턴 테스트 프로시져

4.1에서 모든 결함을 발견할 수 있는 패턴을 구했다. 이를 이용하여 여기서는 각각의 결함에 대해서 테스트하는 프로시저를 구현할 것이다. 그림 5는 이러한 테스트 패턴을 처리한 결과를 보여준다. 왼쪽은 4.1절에서 나온 입력 패턴이고 중간에 있는 부분은 가상적으로 선택할 수 있는 stuck-at 결함이다. 만약 여기서 체크 버튼 중 "Random"을 선택하면 그림 5에 보여지는 선택할 수 있는 결함 중에 하나를 랜덤하게 선택하여 테스트하게 된다.

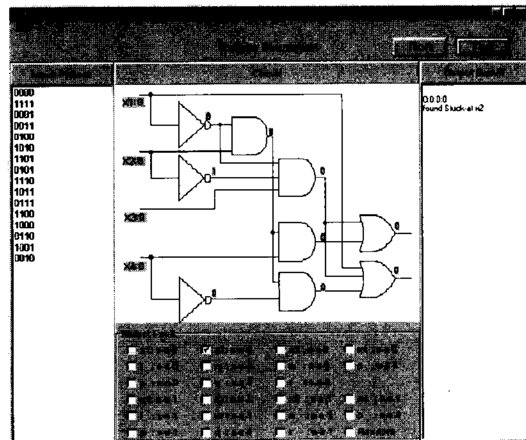

그림 5 테스트 프로시저 실행

테스트 결과는 오른쪽 화면에 보여주게 되며 어느 위치에 어떠한 결함이 발생했는지를 보여준다.

5. 결론 및 향후 연구 과제

본 논문에서 우리는 Exhaustive 테스팅 기법 중에 분할 방법을 사용하여 테스트 패턴을 생성하는 방법을 보여주었고 이를 이용하여 병렬적으로 처리할 수 있는 방법을 보았다. 본 논문에서 사용하는 방법은 병렬로 테스트 패턴을 적용할 수 있으므로 Exhaustive 테스트의 단점인 테스트 수행시간을 줄일 수 있는 장점이 있다. 수행시간은, 순수하게 Test Pattern을 적용하여 에러를 검출하는데 필요한 경우를 전제하면, 전통적인 D-Algorithm에 의해 발생된 Test Pattern을 순차적으로 입력단자에 입력시킬 때 보다 병렬 비트수 (본 논문에서는 8 비트) 배, 즉 8배가 빠르다. 표 3은 ISCAS '85의 각 회로에 대한 KSIM의 수행시간[17]과 본 논문에서 제시한 방법의 수행시간 간의 비교표이다. KSIM은 다양한 Fault-Toleance와 Testing Mechanism들의 평가하기 위한 Fault 시뮬레이터이다. 이 시뮬레이터의 설계목적이

ISCAS-85 회로, 즉, 순수하게 조합회로의 Netlist를 위한 것이기 때문에 피드백되거나, 순차회로용 시뮬레이터로는 제약이 있다. 본 논문에서 언급하고 있는 회로는 이 시뮬레이터의 목적에 부합하는 조합회로로 국한시키기 때문에 [17]의 수치와는 단지 테스트 패턴을 Serial하게 입력하는가, 혹은 Parallel하게 입력하는가에 따른 차이만을 고려한다. 그러나, 회로를 너무 작게 분할할 경우 체크 포인트(check point)와 테스트 패턴 입력 위치가 많아지는 단점이 있고, 이 에 따라 하드웨어적인 Redundancy 문제가 발생할 수 있다. 또한 이 체크 포인트를 작게 하기 위해서 분할을 크게 하였을 경우 각각의 조각들에 대한 패턴이 많아지는 단점을 가지고 있다. 그러므로 효율적인 분할은 테스트의 효율을 올리는데 아주 큰 역할을 한다.

표 3 ISCAS 85 benchmark들에 대한 KSIM의 수행시간 대 제시된 방법의 수행시간

Netlist	KSIM (seconds)	Proposed Testing (seconds)
c17	0.03	0.015
c432	0.5	0.25
c499	1.3	0.65
c880	2.4	1.2
c1355	4.3	2.15
c1908	14.46	7.23
c2670	41.0	20.5
c3540	72.8	36.4
c5315	189.6	94.8
c6288	185.2	92.6
c7552	495.3	247.65

본 논문에서는 이러한 분할부분에서 전체에 게이트 단위로 분할을 하였으므로 아직 효율성은 떨어진다. 하지만 향후 이 부분을 개선하면 더욱 더 빠른 테스트가 가능할 것이다.

참 고 문 헌

[1] Parag K. Lala, *Fault Tolerant and fault testable hardware design* , Prentice Hall International, New York, New York, 1985.

[2] Grabriel M. Seiberman, and Ilan Spillinger, "Functional Fault Simulation as a Guide for Biased-Random Test Pattern Generation," IEEE Transactions on Computer, VOL. 40, NO. 1 January 1991.

[3] S.P. Tomas, and J.P. Shen, "A survey of: functional level testing and testability measures," Res. Rep. CMUCAD-83-18, Carnegi-Mellon Univ., Pittsburgh, PA, 1983.

[4] Janusz Rajski, and Jerzy Tyszer, "Recursive Pseudoexhaustive Test Pattern Generation," IEEE Transactions on Computer, VOL. 42, NO. 12, December 1993.

[5] A.K. Das, A. Sanyal, and P. Pal Chaudhuri, "On characterization of cellular automata with matrix algebra," Inform. Sci., vol. 65, pp 251-277, June 1992.

[6] M. Abramovici, M.A. Breuer, and A.D. Friedman, *Digital Systems Testing and Testable Design.* Computer Science Press, New York, New York, 1990.

[7] E.J. McCluskey, "Verification testing - A pseudoexhaustive test technique," IEEE Trans. Comput. Vol. C-33, pp. 541-546, June 1984.

[8] G.E. Sobelman, and C.H. Chen, "An efficient approach to pseudoexhaustive test generation for BIST design," in Proc. ICCD, 1989, pp. 576-579.

[9] Anderson, Tand P. Lee, *Fault-tolerance. principle and practice*, Prentice-Hall International, New York, New York, 1981.

[10] Dhiraj K. Pradahan, *Fault-Tolerant Computing*, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[11] Sellers, F.F, MoYo Hsiao and C.L. Bearnson, "Analyzing errors with the Boolean difference," IEEE Trans. Comput., pp. 676-683 July 1968.

[12] K.P. Parker and E.J. McCluskey, "Probablistic Treatment of General Combinational Networks," IEEE Trans. Computers, Vol. 24, No.6, pp668-670, June 1975.

[13] Dhiraj K. Pradahan, *Fault-Tolerant Computer System Design*, Prentice-Hall International, New York, New York, 1993.

[14] M.H. Konijnenburg, A.J. Van De Goor, and J.Th. Van der Linden, "Circuit Partitioned Automatic Test Pattern Generation Constrained by Three-State Buses and Ristrictors," 5th Asian Test Symposium(ATS '96), pp29-33, Nov. 1996.

[15] Douglas Chang, and malgorzata Marek-Sadowska, "Partitioning Sequential Circuits on Dynamically Reconfigurable FPGAs," IEEE Transactions on Computers, pp565-578, June 1999.

[16] D.J.Huang, and A.B. Kahng, "When Clusters Meet Partitions: New Density-Based Methods for Circuit Decomposition," Proc. of EDTC, pp60-64, March 1995.

[17] Kristian Wiklund, "A gate Level Fault Simulation Toolkit," Charmers University of Technonology, Gothenburg, Sweden, Tech. Report 00-17, 2001.

김 우 완
1982년 경북대학교 전자공학과 졸업(공학사). 1990년 Texas A&M 대학교 전기전자공학과 졸업(공학석사). 1995년 Texas A&M 대학교 전기전자공학과 졸업(공학박사). 1996년~현재 경남대학교 정보통신공학부 부교수. 관심분야는 컴퓨터 네트워크, 모바일 커뮤니케이션, 멀티미디어 통신