독립고장과 양립 가능한 고장을 이용한 효율적인 테스트 패턴 압축 기법

An Efficient Algorithm for Test Pattern Compaction using Independent Faults and Compatible Faults

尹 度 鉉, 姜 成 昊, 閔 炯 福

Doe Hyun Yoon, Sungho Kang, and Hyung Bok Min

社團法人 大韓電子工學會

論文2001-38SD-2-8

독립고장과 양립 가능한 고장을 이용한 효율적인 테스트 패턴 압축 기법

(An Efficient Algorithm for Test Pattern Compaction using Independent Faults and Compatible Faults)

尹 度 鉉*, 姜 成 昊**, 閔 炯 福***

(Doe Hyun Yoon, Sungho Kang, and Hyung Bok Min)

요 약

조합회로에 대한 ATPG 알고리듬이 효율적으로 100%의 고장 검출율을 달성할 수 있게 되어 감에 따라서 고장 검출율을 그대로 유지한 상태에서 테스트 패턴을 줄이는 압축 기법의 중요성이 점차로 부각되고 있다. 본 논문에서 제시하는 알고리듬은 고장들간의 독립과 양립 관계에 기초해서, 압축된 테스트 패턴을 위해서는 양립할 수 있는 고장 집합의 크기를 크게 해야 하므로, 고장-패턴 쌍과 고장들간의 독립과 양립 관계를 이용해서 고장-패턴 쌍의 트리 구조를 생성하였다. 이 고장-패턴 트리를 바탕으로 해서 효율적으로 압축된 테스트 패턴을 생성할 수 있었고, ISCAS 85와 ISCAS 89 측정 기준 회로에 대한 결과로 제시된 알고리듬의 우수성을 검증하였다.

Abstract

As combinational ATPG algorithms achieve effectively 100% fault coverage, reducing the length of test set without loosing its fault coverage becomes a challenging work. The new approach is based on the independent and the compatible relationships between faults. For more compact test set, the size of compatible fault set must be maximized, thus this algorithm generates fault-pattern pairs, and a fault-pattern pair tree structure using the independent and the compatible relationships between faults. With the fault-pattern pair tree structure, a compact test set effectively generated. The experimental results for ISCAS 85 and 89 benchmark circuits demonstrate the effectiveness of the proposed method.

* 正會員, LG電子 Digital Media 硏究所
(LG Electronics Digital Media Lab.)
** 正會員, 延世大學校 機械電子工學部
(Dept. of Electrical and Electronic Engineering Yonsei Univ.)
*** 正會員, 成均館大學校 電氣電子컴퓨터工學部
(Dept. of Electrical and Computer Engineering Sungkyunkwan University)

※ 본 연구는 1997년도 한국학술진흥재단 대학부설연구소과제 연구비에 의하여 연구되었음.
接受日字: 1999年 10月5日, 수정완료일 :2001年1月29日

I. 서 론

일반적인 조합회로에 대해서 높은 고착 고장 검출률을 갖는 테스트 패턴을 효율적으로 생성[1-3]할 수 있게 됨으로써, 생성된 테스트 패턴의 크기를 효과적으로 줄이는 테스트 패턴 압축(compaction)이 점점 더 중요해지고 있다. 효과적으로 압축된 테스트 패턴은 테스터(ATE)의 사용 시 요구되는 테스트 패턴 저장 장소를 더 적게 사용하여 테스트 할 수 있는 점 이외에, 테스트 시에 소요되는 시간을 줄일 수 있는데, 완전 주사(full scan)를 사용할 경우에 특히 테스트에 소요되는 시간을 크게 줄일 수 있다.

일반적인 테스트 패턴 압축 기법은 압축 작업을 테스트 패턴의 생성과 병행해서 수행하는 방법(compaction during generation)과 테스트 패턴을 생성한 후에 압축 작업을 수행하는 방법(post generation compaction)으로 나누어서 생각할 수 있다.

전자의 경우는 압축 작업이 테스트 패턴의 생성과 동시에 이루어지므로 압축작업에 소요되는 시간이 크다지 크지 않다는 장점이 있지만, 생성된 모든 테스트 패턴을 작업 대상으로 할 수 없기 때문에, 패턴의 생성 순서는 고장 리스트의 순서 등에 결과가 크게 좌우되게 된다. 반면에 후자의 경우는 테스트 패턴의 생성이 완전히 끝난 후에 압축 작업을 수행하기 때문에 고장 리스트의 순서 등에 영향을 받지 않는 압축 작업을 수행할 수 있고, 따라서 전자에 비해서 더 좋은 결과를 얻을 수 있다. 그러나 알고리듬이 고려해야 할 대상이 지나치게 커지게 되어 압축 작업에 소요되는 시간이 전자에 비해서 훨씬 더 오래 걸리게 된다. 그러나 SOCRATES[4]에서 제시된 역순서 고장 시뮬레이션(reverse order fault simulation)과 같은 방법은 압축 작업을 테스트 패턴의 생성이 완전히 끝난 후에 수행하지만, 항상 최상의 결과를 보장할 수는 없지만 빠른 속도로 압축 작업을 효율적으로 수행할 수 있다.

이러한 테스트 패턴 압축 작업은 그 결과를 서로 다른 알고리듬의 결과와 비교해 보는 방법 밖에 없게 된다. 따라서 압축된 패턴의 크기가 최소화된 패턴의 크기에 얼마나 접근했는지는 알 수가 없다. 최소화된 패턴을 구하기 위해서 MINT[5]와 같은 알고리듬이 제시되었지만 최소화된 패턴을 구하기 위해서 소요되는 시간이 지나치게 오래 걸리고 너무나 많은 메모리를 사용하기 때문에 단지 아주 크기가 작은 회로에 대해서만 결과를 구할 수 있다.

효과적으로 최소화된 테스트 패턴의 크기를 예측하기 위해서 독립 고장 집합이 제시되었다[6]. 독립 고장 집합은 이 고장 집합에 속한 어떤 두 고장에 대해서도 하나의 패턴으로 두 고장 모두를 검출할 수가 없는 고장 집합을 말한다. 따라서 독립 고장 집합에 속한 고장의 수만큼의 테스트 패턴이 있어야만 이 고장 집합에 속한 고장을 모두 검출 할 수 있고, 적어도 회로 내에서 크기가 가장 큰 독립 고장 집합-최대 독립 고장 집합에 속한 고장의 수만큼의 패턴이 필요하게 된다. 따라서 최소화된 테스트 패턴의 크기는 이 최대 독립 고장 집합의 크기보다 작을 수가 없고, 테스트 패턴을 압축한 결과가 최대 독립 고장 집합의 크기와 같다면 이 패턴이 최소 크기의 테스트 패턴인 것을 알 수 있고, 더 이상의 압축 작업은 필요 없는 것을 알 수 있다. 그러나 이 최대 독립 고장 집합을 구하는 것은 최소 크기의 테스트 패턴 생성과 마찬가지로 NP-Complete로 알려져 있으므로[7], 최대 독립 고장을 정확하게 구하기보다는 되도록 빠른 시간 안에 크기가 큰 독립 고장 집합을 구해서 압축 작업을 효율적으로 수행할 수 있도록 이용해야 한다. 본 논문에서는 독립 고장 집합을 구하고, 이 독립 고장 집합을 이용해서 같은 패턴으로 검출할 수 있는 고장들의 집합을 구하고, 이러한 고장 집합을 이용해서 효과적인 테스트 패턴 압축 작업을 수행하였다.

Ⅱ. 양립할 수 있는 고장 집합

어떤 고장들의 집합이 있을 때, 이 고장 집합에 속한 모든 고장이 하나의 테스트 패턴에 의해서 검출 될 수 있다면 이 고장 집합을 양립할 수 있는 고장 집합(compatible fault set)이라고 할 수 있다. 테스트 패턴 압축의 목표는 되도록 적은 테스트 패턴으로 고장을 검출하는 것이므로, 효과적인 압축을 수행하기 위해서는 하나의 패턴이 검출하는 고장의 수를 크게 하는 것이 유리하고, 따라서 가능한 한 크기가 큰 양립할 수 있는 고장 집합을 찾아서 이 고장 집합 내의 고장을 모두 검출할 수 있는 테스트 패턴을 찾는 것이 아주 중요하다.

때문에 본 논문에서는 양립할 수 있는 고장을 효과적으로 찾고, 그 크기를 크게 하기 위해서 다음과 같은 방법을 사용하였다. 고장-패턴 쌍(fault pattern pair)을 구성하고 이 고장-패턴 쌍은 하나의 고장과 이 고장을 검출할 수 있는 하나의 테스트 패턴으로 구성된다. 이 때 이 테스트 패턴은 X를 포함하여 다른 패턴과 합쳐질 수 있도록 한다. 이 고장-패턴 쌍은 이미 만들어진 고장-패턴 쌍과 비교하여, 실제로는 각 고장-패턴 쌍의 테스트 패턴을 비교하여 두 고장이 독립적인 관계인지, 양립 가능하지를 조사할 수 있다.

1. 두 테스트 패턴의 비교

두 테스트 패턴이 서로 합쳐질 수 있는지는 두 패턴의 각 입력 비트를 표 1을 이용하여 비교함으로써 알 수 있다.

표 1. 두 패턴간의 비교
Table 1. Comparing Two Patterns.

	0	1	X
0	0	ϕ	0
1	ϕ	1	1
X	0	1	X

표 1의 각 비트간의 비교에서 ϕ이 되는 경우가 하나라도 발생한다면 두 패턴은 합쳐질 수 없는 경우이고, 두 패턴의 비교에서 ϕ이 하나도 나타나지 않는다면 두 패턴은 서로 합쳐질 수 있는 경우이다. 전자의 경우 각 패턴에 대응하는 두 고장은 독립적인 관계이고, 후자의 경우는 두 고장이 서로 양립 가능한 경우가 된다.

물론 이러한 테스트 패턴간의 비교에서 독립적이라고 판별이 나도, 실제로는 테스트 패턴을 생성할 때, 후방 추적시의 선택에 의해서 다른 패턴을 생성할 수 있으므로, 같은 고장에 대해서 다른 선택을 취한 패턴을 사용한다면 양립 가능한 경우로 나타날 수도 있다. 따라서 정확하게 두 고장이 독립 관계인지를 알려면 각 고장의 필수 할당 값(necessary assignment)만을 추출해서 이 필수 할당 값들 간에 서로 충돌이 발생하는지를 살펴보아야만 한다. 그러나 필수 할당 값만을 사용하여 두 고장이 독립 관계가 아니라고 판별이 나더라도, 이 비교는 단지 필수 할당 값만을 사용한 비교이므로, 이 두 고장이 하나의 패턴에 의해서 검출될 수 있다고 단정할 수 없게 된다. 따라서 정확하게 고장들 간의 독립 관계를 찾을 수 없다 하더라도 고장들 간의 양립 관계를 찾기 위해서는 패턴을 직접 생성한 후에 패턴간의 비교를 수행하는 것이 더 정확하다.

본 논문에서는 두 패턴간의 비교 시에 COMPACT[8]에서 제시된 방법을 이용하여 빠르게 수행할 수 있었다. 표 2에 이러한 빠른 패턴 비교를 위한 3-논리값 표시 방법을 나타내었다. 만약 depend bit 가 1 이면 value bit이 테스트 패턴 입력이 0인지 1인지를 나타내고, depend bit이 0 이면 value bit도 0으로 해서 Don't Care를 나타내게 된다.

표 2. 패턴의 표시 방법
Table 2. Pattern coding.

	0	1	X
depend	1	1	0
value	0	1	0

테스트 패턴의 각 입력을 표 2에서와 같은 방법으로 표시해서 회로의 각 입력을 depend와 value를 나타내는 두 개의 unsigned integer 형 변수로 하여 32개의 테스트 패턴을 두 unsinged integer에 표시할 수 있다. 그림 1에 이러한 방법을 이용하여 두 쌍의 32개의 테스트 패턴을 동시에 비교하는 방법을 나타내었다.

```
/* Compare Pattern1 and Pattern 2 */
intersection = Pattern1->depend & Pattern2->depend;
T1 = intersection & Pattern1->value;
T2 = intersection & Pattern2->value;
if T1 == T2
    Pattern3->depend = Pattern1->depend | Pattern2->depend;
    Pattern3->value = Pattern1->value | Pattern2->value;
```

그림 1. 두 패턴의 비교 방법
Fig. 1. Algorithm for comparing two patterns.

그림 1의 과정은 Pattern1과 Pattern2를 비교하고 만약 두 패턴이 합쳐질 수 있다면 합쳐진 패턴을 Pattern3에 저장하는 루틴이다. 두 패턴의 비교에 단지 세 번의 AND 동작이 필요하고, 두 패턴이 합쳐질 때 두 번의 OR 동작만이 필요하기 때문에 패턴의 비교가 아주 빠르게 이루어질 수 있다.

2. 고장-패턴 쌍의 트리 구조 형성

각 고장-패턴 쌍(Fault Pattern Pair: FPP)에 속한 테스트 패턴을 비교함으로써 각 고장이 독립관계인지, 양립가능한지를 판단할 수 있다. 좀 더 효과적으로 이러한 관계를 규정하기 위해서 고장-패턴 쌍을 그림 2와 같이 구성할 수 있다.

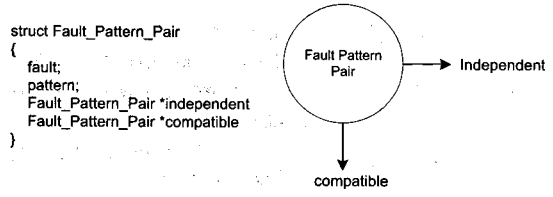

그림 2. 고장-패턴 쌍 (a) 구조적 표현 (b) 그래프 형태
Fig. 2. Fault-Pattern Pair. (a) structural expression (b) graphical expression

각 고장-패턴 쌍은 고장과 해당 고장을 검출할 수 있는 테스트 패턴을 포함하고, 이 고장-패턴 쌍과 비교

해서 독립 관계인 고장-패턴 쌍과 양립 가능한 관계에 있는 고장-패턴 쌍을 가리키는 포인터를 가진다. 이러한 고장-패턴 쌍을 비교하고 그 관계를 연결하면 그림 3과 같은 트리 구조를 형성할 수 있다.

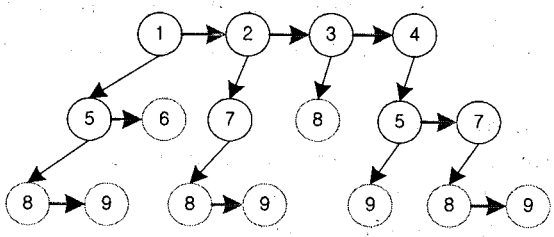

그림 3. 고장-패턴 쌍 트리 구성의 예
Fig. 3. Example of constructing Fault-Pattern Pair Tree.

이 그림 3에서 보이는 트리 구조에서 고장 1, 2, 3, 4는 서로 독립 관계에 있고, 1, 5, 8은 양립 가능한 고장 집합이다. 고장 8의 경우는 고장 5와도 양립가능하고, 동시에 고장 3, 7과도 동시에 양립 가능한 고장이다. 이러한 트리 구조로부터 다음과 같은 양립 가능한 고장 집합을 얻을 수 있다.

{1, 5, 8}, {1, 5, 9}, {1, 6}, {2, 7, 8}, {2, 7, 9}, {3, 8}, {4, 5, 9}, {4, 7, 8}, {4, 7, 9}

같은 고장이 여러 고장 집합에 나타나는 것은 양립 가능한 고장 집합을 구성할 수 있는 여러 가지 경우가 존재하기 때문이다.

실제로는 고장-패턴 쌍의 트리를 형성할 때는 두 고장이 양립 가능할 경우 새로 추가되는 고장-패턴 쌍에는 새로 합쳐진 패턴을 저장해서 전체 고장-패턴 쌍의 트리 구조를 구성한 후에 일일이 고장 집합을 구성하지 않아도 트리의 단말에 있는 고장-패턴 쌍의 테스트 패턴을 이용할 수 있도록 구성하였다.

새로이 생성된 고장-패턴 쌍을 고장-패턴 쌍 트리에 추가하는 방법을 그림 4에 나타내었다. 기존의 고장-패턴 쌍 트리가 있을 때에, 트리의 제일 왼쪽 첫 번째 노드부터 이 알고리듬을 적용할 수 있다. 어떤 노드(this_node)가 있고, 이 노드의 패턴이 T1일 때, T1과 새로이 생성된 패턴 T2를 비교하여 양립할 수 있다면 새로운 합쳐진 패턴 T3를 생성하게 된다. 만약 this_node의 양립 노드가 NULL이라면 패턴 T3를 가지는 새로운 고장-패턴 쌍을 this_node의 양립 노드로 추가하고, NULL이 아니라면 양립 노드를 새로운 this_node로 하고 패턴 T3를 T2로 하여 재귀적으로 compare_and_merge 알고리듬을 호출하게 된다. 그리고 나서는 독립 관계에 있는 노드를 향해서 계속적으로 재귀적 호출을 시도한다. 만약 독립 관계에 있는 노드가 NULL이라면 flag를 검사해서 이 노드가 지금까지의 다른 노드들과 양립 관계에 있었던 적이 있었는지를 검사하고 만약 어떠한 노드와도 양립 관계에 놓이지 않았다면 새로운 독립 관계의 노드를 추가하게 된다.

```
compare_and_merge(this_node, pattern T2, flag)
{
    memory allocation for pattern T3;
    compare this_node's pattern(T1) and T2;
    if compatible
    {
        flag = MERGED;
        if this_node->compatible is NULL
            make a new FPP node which is a this_node's compatible;
        else
            compare_and_merge (this_node->compatible, T3, NOT_MERGED);
        free memory for T3;
        if this_node->independent is not NULL;
            compare_and_merge(this_node->independent, T2, flag);
        else if flag is NOT_MERGED
            make a new FPP node which is this_node's independent;
    }
}
```

그림 4. compare_and_merge 알고리듬
Fig. 4. compare_and_merge algorithm.

Ⅲ. 양립 가능한 고장 집합을 이용한 제안된 알고리듬

본 논문에서는 2절에서 설명한 방식으로 구한 고장-패턴 쌍의 트리 구조를 이용하여 새로운 테스트 패턴 압축 기법을 구현하였는데, 그 과정은 그림 5와 같다. 먼저 회로의 독립 고장 집합을 계산하고, 이중에서 고장의 수가 가장 많은 최대 고장 집합 (Maximum Independent Fault Set)을 찾는다. 최대 고장 집합에 속하는 고장들 중에서 고장-패턴 쌍을 구하고, 그 다른 독립 고장 집합에 속하는 고장들에 대해서도 테스트 패턴을 생성해서 앞서 설명한 compare_and_merge 알고리듬을 이용해서 고장-패턴 쌍 트리를 생성한다. 생성된 고장-패턴 쌍 트리에서 양립 노드가 NULL인 테스트 패턴 만을 추출한 후에 이 테스트 패턴들의 남아 있는 Don't care 비트들을 무작위 패턴으로 채운 후에 최종적인 테스트 패턴을 추출한다.

여기서 독립 고장 집합을 구한 방법은 COMPACTEST[9]에서 제시된 최대 비분기 영역을 이용한 수정된 Labeling 알고리듬을 이용하였다. 물론 이 방법으로 구한 최대 독립 고장은 그다지 크기가 크기 않지만 단지 한 번의 전방향 탐색만으로 비분기 영역 내에서의 독립 고장 집합을 구할 수 있으므로, 아주 빠르게 독립 고장 집합을 구할 수 있다. 이 독립 고장 집합의 크기 순서대로 고장-패턴 쌍의 트리를 구성하면 효율적으로 양립 가능한 고장 집합을 구할 수 있다. 이 트리 구조에서 어떤 노드의 양립(compatible) 노드는 항상 바로 위의 노드의 패턴을 포함하고 따라서 바로 위 노드의 고장을 검출 할 수 있게 된다. 따라서 이 고장 트리에서 가장 밑에 위치하는 고장-패턴 쌍, 양립 노드가 NULL인 고장-패턴 쌍의 패턴들이 가장 크기가 큰 양립 가능한 고장 집합에 대한 테스트 패턴이 된다.

```
compaction algorithm
{
    Independent_Fault_Set_calculation();
    for faults in Maximum Independent Fault Set
    {
        T2 = pattern_generation( );
        if FPP is NULL
            make_a_new_FPP( );
        else
            compare_and_merge(FPP, pattern T2, NOT_MERGED);
    }
    for 2nd IFS to last IFS
    {
        for faults in current IFS
        {
            T2 = pattern_generation();
            compare_and_merge(FPP, pattern T2, NOT_MERGED);
        }
    }
    select_terminal_node_pattern_from_FPP_tree_structure();
    fill_random_value_on_unspecified_bits_of_patterns();
    extract_minimal_test_set();
}
```

그림 5. 제안된 압축 알고리듬
Fig. 5. Proposed compaction algorithm.

예를 들어 그림 6의 c17 회로에서는 다음과 같은 4개의 고장을 포함하는 최대 독립 고장 집합을 구할 수 있다. p/q는 p노드의 stuck-at-q 고장을 표시한다.

{1/0, 17/1, 1/1, 4/1}

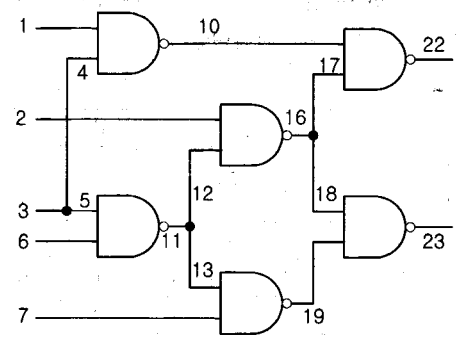

그림 6. c17 회로
Fig. 6. c17 circuit.

이 최대 독립 고장에 속한 고장들은 서로 독립 관계이므로 그림 7에서 볼 수 있는 고장-패턴 쌍의 트리를 구성할 수 있다. 양립(compatible)이나 독립(independent) 관계를 표시하지 않은 노드는 양립이나 독립 노드로 NULL을 갖는 경우이다.

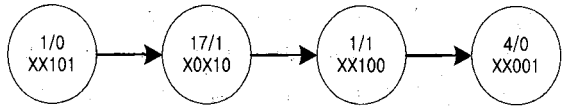

그림 7. 최대 독립 고장 집합
Fig. 7. Maximum Independent Fault Set.

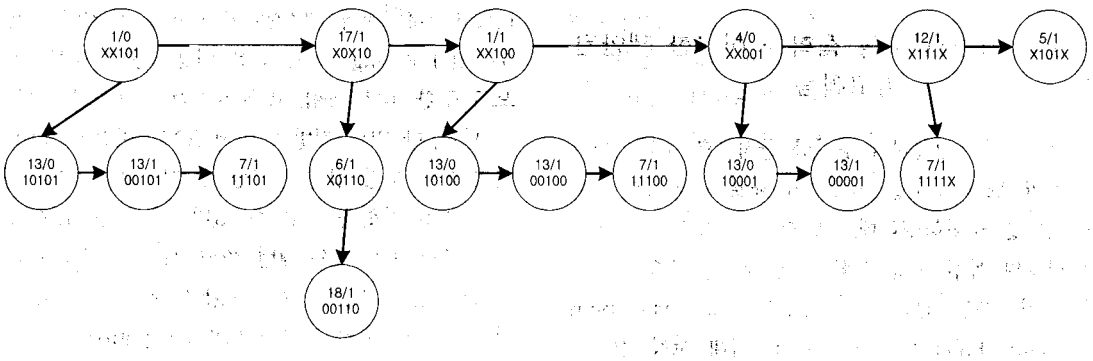

그림 8. 최종적인 고장-패턴 쌍 트리 구조
Fig. 8. Constructed Fault-Pattern Pair Tree structure.

여기에 나머지 독립 고장 집합에 속한 고장들에 각각 대해서 테스트 패턴을 생성하고 compare_and_merge 루틴을 수행하면 그림 8과 같은 최종적인 고장-패턴 쌍의 트리를 구성할 수 있다.

이 고장-패턴 쌍 트리에서 가장 크기가 큰 양립 가능한 고장 집합에 대한 테스트 패턴들, 즉 트리에서 양립 노드가 NULL인 노드의 패턴들을 선택할 수 있고 이 패턴들을 그림 9에 나타내었다.

입력	1	2	3	6	7
패턴 1	1	0	1	0	1
패턴 2	0	0	1	0	1
패턴 3	1	1	1	0	1
패턴 4	0	0	1	1	0
패턴 5	1	0	1	0	0
패턴 6	0	0	1	0	0
패턴 7	1	1	1	0	0
패턴 8	1	0	0	0	1
패턴 9	0	0	0	0	1
패턴 10	1	1	1	1	X
패턴 11	X	1	0	1	X

그림 9. 선택된 테스트 패턴
Fig. 9. Selected Test Patterns.

이렇게 선택한 테스트 패턴은 모두 최대 크기의 양립 가능한 고장 집합에 대한 테스트 패턴이므로 패턴 하나가 검출할 수 있는 고장의 수가 많고, 따라서 이 패턴들을 적절히 선택해서 불필요한 패턴을 제거한다면 아주 적은 크기로 회로의 모든 고장을 검출할 수 있는 패턴을 구할 수 있다.

효과적으로 최소의 패턴을 선택하기 위해서 Double Detection[9]에서 제시된 방법으로 필수적인 테스트 패턴을 먼저 선택하고 나머지 테스트 패턴들에 대해서 각 패턴이 검출하는 고장의 수를 세어서, 가장 많은 수의 고장을 검출하는 테스트 패턴을 우선적으로 선택함으로써 테스트 패턴의 수를 효과적으로 줄일 수 있다. 또한 이 작업을 계속적으로 반복하여 다른 패턴에서 이미 검출하는 고장만을 검출하는 필요 없는 테스트 패턴이 하나도 없고, 모든 테스트 패턴이 다른 어떠한 패턴에도 검출되지 않는 고장을 적어도 하나 이상 검출하는 단계까지 테스트 패턴을 줄일 수 있다.

이러한 방식으로 압축된 테스트 패턴을 구하는 과정은 회로의 크기가 상대적으로 작을 경우는 상당히 효율적으로 동작할 수 있지만 회로의 크기가 아주 커서 고장-패턴 쌍의 트리가 지나치게 크게 구성되는 경우나 회로의 특성상 각 고장에 대한 테스트 패턴이 X값을 거의 포함하지 않고 모든 주 입력단의 값이 0이나 1로 확정되어진 형태로 나타날 경우에는, 지나치게 오랜 시간이 필요하거나 구해진 패턴이 오히려 압축된 형태가 되지 못할 수가 있다. 따라서 실제의 구현에서는 이러한 경우에서 생길 수 있는 단점을 보완하기 위해서 그림 5에서 제안된 알고리듬을 수행하기 이전에 의사 무작위 패턴을 생성하여 전체 고장 리스트 중 절반의 고장을 검출할 때까지 한꺼번에 32개씩의 의사 무작위 패턴을 미리 생성하였다.

의사 무작위 패턴을 생성할 때, 더 많은 고장을 의사 무작위 패턴을 사용하여 검출하려고 하면, 고장-패턴 쌍을 이용한 트리의 크기가 줄어들게 되어서 이 과정이 더 빠르게 되므로 전체적으로 걸리는 시간이 줄어

표 3. ISCAS 85회로에 대한 결과 비교
Table 3. Compaction result comparisons for ISCAS 85 benchmark circuits.

회로	테스트 패턴의 수				수행 시간(초)			
	[12]	[13]	[9]	새 알고리듬	[12]	[13]	[9]	새 알고리듬
c17	6	5	-	4	0	0	-	0
c95	16	14	-	12	0	0	-	0
c432	55	54	38	44	0	6	4.1	1
c499	53	56	52	52	0	11	3.4	0
c880	83	63	26	44	0	3	8.0	6
c1355	86	85	84	86	1	36	15	3
c1908	137	153	113	124	2	46	40	8
c2670	142	107	51	77	11	34	91	119
c3540	197	184	97	141	10	123	204	72
c5315	157	188	49	81	5	66	244	750
c6288	30	-	18	33	8	-	203	26
c7552	276	239	84	167	39	188	490	776

표 4. ISCAS 89 회로에 대한 결과 비교
Table 4. Compaction result comparisons for ISCAS 89 benchmark circuits.

회로	테스트 패턴의 수			수행시간(초)		
	[12]	[9]	새 알고리듬	[12]	[9]	새 알고리듬
s208	42	27	29	0	0.9	0
s298	38	23	30	0	1.4	0
s344	28	15	19	0	1.4	0
s349	29	13	19	0	1.4	0
s382	41	25	30	0	1.8	0
s386	83	64	70	0	3.1	0
s400	41	24	33	0	1.9	0
s420	74	43	51	1	2.9	0
s444	38	24	28	0	2.3	0
s510	67	55	59	0	5.8	0
s526	79	51	63	0	4.2	0
s641	74	24	42	1	3.1	0
s713	74	24	41	1	4.1	1
s820	136	95	110	1	14	1
s832	134	96	109	2	16	1
s838	143	75	80	4	11	24
s953	116	79	83	2	16	2
s1196	163	117	143	1	27	4
s1238	170	129	147	2	34	5
s1423	90	34	44	1	21	48
s1488	152	102	116	1	36	4
s1494	150	101	115	1	36	4

들게 된다. 그러나 고장-패턴 쌍 트리가 더 적은 수의 고장만을 고려하므로 최종적인 테스트 패턴의 크기가 더 커지게 된다. 즉 의사 무작위 패턴을 얼마만큼 이용할 것인 가로 전체 알고리듬에 걸리는 시간과 최종적인 고장의 질을 결정할 수 있다.

Ⅳ. 제안된 알고리듬의 구현 및 검증

2절과 3절에 제시된 알고리듬은 Sun UltraSparc-1 Workstation에서 C언어로 구현되었고 표 3과 4에서 ISCAS 85 측정 기준 회로[10]와 ISCAS 89 측정 기준

회로[11]의 완전 스캔된 회로에 대한 결과를 나타내었고 기반이 된 ATPG 알고리듬[12]과 [13]의 결과, 그리고 [9]의 결과와 비교하였다. '새 알고리듬'이라고 표시된 열이 본 논문의 결과이고 이 결과와 [12]의 결과, 그리고 [9]의 결과는 모든 회로에 대하여 무해 고장을 제외한 모든 테스트 생성이 가능한 고장에 대해서 100% 고장 검출율을 갖는다. 반면에 [13]의 결과는 모든 회로에 대하여 100% 고장 검출율을 갖지는 못한다. [13]의 결과는 c6288에 대한 결과와 ISCAS 89 측정 기준 회로에 대한 결과가 없기 때문에 단지 ISCAS 85 회로의 결과만 비교할 수 있었다.

이 결과 비교에서 볼 수 있듯이, c6288에 대한 결과를 제외하고는 제안되 알고리듬이 [12], [13]의 결과보다 훨씬 더 적은 테스트 패턴을 얻을 수 있음을 알 수 있다. [9]의 결과와 비교하면 제안된 알고리듬이 [9]의 결과 보다 더 많은 테스트 패턴을 생성하는 것을 알 수 있다. 그러나 [9]의 알고리듬은 수행시간이 제안된 알고리듬에 비해서 너무나 오래 걸리는 반면에, 제안된 알고리듬은 모든 경우에 그렇지는 못하지만 [9]의 결과와 상당히 근접한 결과를 보이면서 수행시간은 크게 줄어들어 있음을 알 수 있다.

V. 결 론

본 논문에서는 테스트 패턴의 압축 기법에 관해서 연구하였고, 회로 내의 고착 고장들 간의 관계를 독립 관계와 양립 가능한 관계로 구별하였다. 각 고장간의 관계를 판별하는 방법을 제시하였고, 적은 수의 테스트 패턴으로 고장을 검출하기 위해서는 양립 가능한 고장들의 집합을 구성하는 것이 효율적임을 보였다. 효율적으로 양립 가능한 고장들의 집합을 구성하기 위해서 고장-패턴 쌍을 만들고, 이 고장-패턴 쌍들 간의 관계가 독립인지, 양립 가능한지를 기준으로 고장-패턴 쌍의 트리를 형성할 수 있었다. 생성된 고장-패턴 쌍의 트리로부터 양립 가능한 고장 집합을 구하고 각 고장 집합에 대한 테스트 패턴을 찾을 수 있었고, 이러한 패턴으로부터 적은 수의 테스트 패턴을 추출할 수 있었다. ISCAS 85와 완전 스캔이 삽입된 ISCAS 89 측정 기준회로에 대한 실험 결과를 구하였고, 테스트 패턴 생성의 기반이 된 ATPG 결과와 기존의 테스트 패턴 압축 기법의 결과의 비교를 통하여 본 논문에서 제시된 알고리듬이 빠른 수행 시간 안에 상당히 크기가 적은 테스트 패턴을 생성함을 보였다. 또한 본 알고리듬에서 의사 무작위 패턴의 사용에 의해서 수행 시간을 단축시킬 수 있었는데, 이 의사 무작위 패턴을 더 많이 사용할수록 수행 시간은 줄어들었지만 결과는 조금씩 더 커지는 것을 알 수 있었다. 따라서 좀 더 효율적인 테스트 패턴 압축 알고리듬을 구현하기 위해서는 각 회로에 알맞게 목표 고장을 설정하고 그에 맞추어서 의사 무작위 패턴을 생성할 수 있는 알고리듬이 필요하다.

참 고 문 헌

[1] J. P. Roth, "Diagnosis of Automata Failures: A Calculus and a Method," IBM Journal of Research and Development, vol. 10, pp. 278-291, July 1966.

[2] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," IEEE Trans. on Computer, vol. C-30, pp. 215-222, Mar. 1981.

[3] H. Fujiwara, and T. Shimono, "On the Acceleration of Test Generation Algorithms," IEEE Trans. on Computer, vol. C-32, pp. 1137-1144, Dec. 1983.

[4] M. Schultz, E. Trischler, and T. Sarfert, "SOCRATES: A Highly Efficient Automatic Test Pattern Generatio System," IEEE Trans. on CAD, pp. 126-137, Jan. 1988.

[5] Y. Matsunaga, "MINT-An exact algorithm for finding minimum test sets," IEICE Trans. Fundamentals, vol. E-76-A, pp. 1652-1658, Oct. 1993.

[6] S. B. Akers, C. Joseph, and B. Krishnamurthy, "On the role of independent fault sets in the generation of minimal test sets," Proc. of International Test Conference, Sept. 1987, pp. 1100-1107.

[7] B. Krishnamurthy, and S. B. Akers, "On the complexity of estimating the size of test set,' IEEE Trans. on Computer, vol. C-33, no. 8, pp. 750-753, Aug. 1984.

[8] B. Ayari, and B. Kaminska, "A new dynamic

test vector compaction for automatic test generation," IEEE Trans. on CAD, vol. C-13, no. 3, pp. 353-358, Mar. 1994.

[9] S. Kajihara, I. Pomeranz, K. Kinoshita, and S. M. Reddy, "Cost-effective generation of minimal test sets for stuck-at faults in combinational logic circuits," Proc. of 30th Design Atumation Conference, pp. 102-106, June 1993.

[10] F. Brglez, and H. Fujiwara, "A Neural Netlist of 10 Combinational Benchmark Designs and a Special Translator in Fortran," Proc. of International Symposium on Circuits and Systems, June 1985.

[11] F. Brglez, D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," Proc. of International Symposium on Circuits and Systems, May 1989.

[12] Sang Yoon Han, Sungho Kang, "Efficient Redundancy Identification for Test Pattern Generation," Proc. of IEEE International ASIC conference, pp 52-56, Sep. 1997. pp 52-56, September 1997.

[13] 임동욱, 민형복, "검사 신호에 대한 저비용 압축," 정보과학회 논문지(A), 제 25권, 제 11호, Nov. 1998

저 자 소 개

尹 度 鉉(正會員) 第 37卷 SD編 12月號 參照

姜 成 昊(正會員) 第 37卷 SD編 12月號 參照

閔 炯 福(正會員) 第 36卷 SD編 9月號 參照