論文98-35C-9-4

가중치 집합 최적화를 통한 효율적인 가중 무작위 패턴 생성

(Efficient Weighted Random Pattern Generation Using Weight Set Optimization)

李 沆 圭 *, 金 弘 植 *, 姜 成 昊 *

(Hang Kyu Lee, Hong Sik Kim, and Sungho Kang)

요 약

가중 무작위 패턴 테스트에서 적은 수의 가중 무작위 패턴을 사용하여 높은 고장 검출율을 달성하기 위해서는 최적화된 가중치 집합들을 찾아내야만 한다. 따라서 최적화된 가중치 집합을 찾아내려는 많은 연구가 행해져 왔다. 이 논문에서 결정론적인 테스트 패턴에 대한 샘플링 확률을 기반으로 하여 최적화된 가중치 집합을 효율적으로 찾는 새로운 가중치 집합 최적화 알고리듬을 제안한다. 아울러 시뮬레이션을 통해 적당한 최대해밍거리를 구하는 방법도 소개된다. ISCAS 85 벤치마크 회로에 대한 실험결과는 새로운 가중치 집합 최적화 알고리듬과 적절한 최대 해밍거리를 구하는 방법의 효율성을 뒷받침해 준다.

abstract

In weighted random pattern testing it is an important issue to find the optimal weight sets for achieving a high fault coverage using a small number of weighted random patterns. In this paper, a new weight set optimization algorithm is developed, which can generate the optimal weight sets in an efficient way using the sampling probabilities of deterministic tests patterns. In addition, the simulation based method of finding the proper maximum Hamming distance is presented. Experimental results for ISCAS 85 benchmark circuits prove the effectiveness of the new weight set optimization algorithm and the method of finding the proper maximum Hamming distance.

I. 서 론

의사 무작위 패턴 테스트는 하드웨어 오버헤드가 적기 때문에 내장된 자체 테스트 기법에서 가장 일반적으로 채택되고 있다. 그러나 무작위 패턴 저항 고장들은 의사 무작위 패턴 테스트[1,2,3,4]로는 잘 검출되지 않는다. 가중 무작위 패턴 테스트[5,6,7]는 이 문제를 극복하기 위해서 제안되었다. 이 방법은 LFSR(Linear Feedback Shift Register)에 약간의 하드웨어를 첨가함으로써 무작위 패턴 저항 고장들을 효율적으로 검출하게 되고 따라서 높은 고장 검출율을 짧은 시간 안에 달성할 수 있다.

가중 무작위 패턴 테스트는 단일 가중치 집합을 사용하거나 다중 가중치 집합을 사용한다. 단일 가중치 집합을 사용하면 하드웨어 오버헤드는 적지만 높은 고장 검출율을 빠른 시간 안에 달성하는 가중 무작위 패턴 테스트의 특성을 충분히 살릴 수 없다. 다중 가중치 집합을 사용하는 가중 무작위 패턴 테스트에서 가중치 집합의 성능에 따라 가중치 집합의 수와 가중 무작위 패턴의 길이가 좌우되므로 최적화된 가중치 집합을 구하는 것은 매우 중요한 사항이다. 또한 가중치

* 正會員, 延世大學校 電氣工學科
(Yonsei University Dept. of Electrical Eng.)

※ 본 연구는 97년도 교육부 반도체 분야 학술 연구 조성비(ISRC 97-E-5018)에 의하여 연구되었음.
接受日字:1998年5月8日, 수정완료일:1998年8月18日

집합의 성능은 가중치 집합의 수와 가중 무작위 패턴의 길이의 관점에 따라 평가되어야만 한다. 가중치 집합의 수가 증가하면 높은 고장 검출율을 달성하는데 필요한 가중 무작위 패턴의 수는 감소하게 된다. 그러나 가중치 계산 논리(logic)로 말미암아 하드웨어 오버헤드가 증가하게 된다. 반면에 가중치 집합의 수가 감소하면 하드웨어 오버헤드도 감소하지만 많은 수의 가중 무작위 패턴이 필요하게 된다. 따라서 적은 수의 가중 무작위 패턴과 적은 양의 하드웨어로 높은 고장 검출율을 달성하기 위해서는 가중치 집합 최적화가 필수적인 것이다.

효율적인 가중치 집합을 생성하기 위해 회로의 구조적 특성을 분석하여 각 입력에 적합한 가중치를 구하는 방법[6,9]이 있고 회로에 대한 결정론적인 테스트 패턴 집합을 구한 후에 이를 기반으로 가중치 집합을 구하는 방법[5,10]이 있다. 계산된 결정론적인 테스트 패턴 집합을 기반으로 가중치 집합을 구하는 경우, 기존의 방식은 ATPG 시스템을 이용하여 결정론적인 테스트 패턴 집합을 구한 후, 단순히 각 입력에 대하여 논리값 '1'의 개수 대 논리값 '1'과 논리값 '0'의 개수의 합의 비를 가중치로 계산하는 방식[1,6]을 따른다. 이 때 가중치의 성능을 향상시키기 위해 ATPG 과정에서 결정론적인 테스트 패턴에 무상관 비트의 수를 최대로 발생시키는 알고리듬[5]을 개발하기도 하였다.

가중 무작위 패턴의 성능을 향상시키기 위해 여러 개의 가중치 집합을 사용하기도 한다. 이러한 다중 가중치 집합에 기초한 가중 무작위 패턴의 생성의 경우[5,6,8,10], 결정론적 테스트 패턴들의 집합을 테스트 패턴들간의 해밍거리(Hamming distance)에 따라 여러 개의 부분 집합으로 분할하는 방식[5,6,10]을 일반적으로 채택한다. 이때 가중치 집합의 수와 가중 무작위 패턴의 길이는 최대 해밍거리 값에 막대한 영향을 받는다. 최대 해밍거리 값이 증가하면 가중치 집합의 수는 감소하지만 가중 무작위 패턴의 길이는 증가하는 경향이 있다. 반면에 최대 해밍거리 값이 감소하면 가중 무작위 패턴의 길이는 감소하지만 가중치 집합의 수가 증가하는 경향이 있다. 따라서 최대 해밍거리는 생성된 가중치 집합이 가중 무작위 패턴의 길이와 가중치 집합의 수라는 관점에서 모두 효율적일 수 있도록 최적화되어야만 할 것이다. 그러나 지금까지의 연구 발표에서는 최대 해밍거리의 최적값을 합리적으로 구하는 타당한 방법이 제안되지 않았다. 대부분 직관적인 방식에 의존하여 최대 해밍거리를 정할 뿐이다.

우리는 미리 계산된 결정론적 테스트 패턴들을 이용해 가중치 집합을 최적화하는 방법을 개발하였다. 우리는 기존의 방식대로 결정론적인 테스트 패턴을 기반으로 가중치를 구한 후 그 가중치에 대한 결정론적인 테스트 패턴들의 샘플링확률을 구한다. 이 때, 낮은 샘플링확률을 갖는 테스트 패턴들이 존재하는데 이 낮은 샘플링확률들이 상승하도록 가중치를 보정한다. 이러한 과정을 반복하여 최적화된 가중치 집합을 구한다. 또한 최대 해밍거리의 최적값을 효율적인 방법으로 구하는 방법을 제안한다. 우선 정량적인 실험분석을 기반으로 유도해낸 공식을 통하여 최대 해밍거리의 초기값을 구하고 시뮬레이션을 통해 초기값을 보정하여 최적화된 최대 해밍거리를 구한다.

즉, 이 논문에서는 새로운 가중치 집합의 최적화 알고리듬이 제안되고 적당한 최대 해밍거리를 효율적으로 구하는 방법이 소개된다.

Ⅱ. 가중 무작위 패턴 생성의 분류

가중 무작위 패턴을 생성하는 방법들은 가중치 집합의 수에 따라 두 가지로 분류될 수 있다. 하나는 단일 가중치 집합에 기초한 방법[1]이고 또 다른 하나는 다중 가중치 집합에 기초한 방법[5,6,8]이다. 첫 번째 방법의 하드웨어 오버헤드는 매우 작지만 짧은 시간 동안에 높은 고장 검출율을 달성할 수 없다. 반면 두 번째 방법의 경우 높은 고장 검출율을 적은 수의 가중 무작위 패턴으로 달성할 수 있다. 특정한 구조의 회로에 대해서 고장 검출율을 어느 정도 수준까지 높이기 위해서는 다중 가중치 집합을 사용해야 한다.

$t_j = (t_j[1], t_j[2], \cdots, t_j[m])$을 하나의 결정론적인 테스트 패턴이라고 하고 $W = (w_1, w_2, \cdots, w_m)$을 m개의 주입력을 갖는 회로에 대하여 생성된 하나의 가중치 집합이라고 하자. 하나의 결정론적인 테스트 패턴 t_j의 샘플링 확률 P_j는 다음 식(1)에 의해 계산할 수 있다.

$$P_j = \Pi_{i=0,\, t_j[i] \neq x}^{m} \{(w_i \times t_j[i]) + (1 - w_i) \times (1 - t_j[i])\} \quad (1)$$

다중 가중치 집합이 필요한 예를 그림 1에 나타내었다. 그림 1의 회로는 n개의 신호선이 AND 게이트와 OR 게이트로 동시에 입력되는 형태이다. 우선

AND 게이트의 출력단 O_1의 논리값 0 고착고장과 OR 게이트의 출력단 O_2의 논리값 1 고착고장을 동시에 검출하려고 한다고 가정하자. O_1의 논리값 0 고착고장과 O_2의 논리값 1 고착고장의 테스트 패턴을 각각 t_1, t_2라고 하면 $t_1 = (1,1,\cdots,1)$, $t_2 = (0,0,\cdots,0)$이 된다. t_1과 t_2를 이용해서 가중치 집합 W를 구해보면 $W = (0.5,0.5,\cdots,0.5)$ 가 된다.

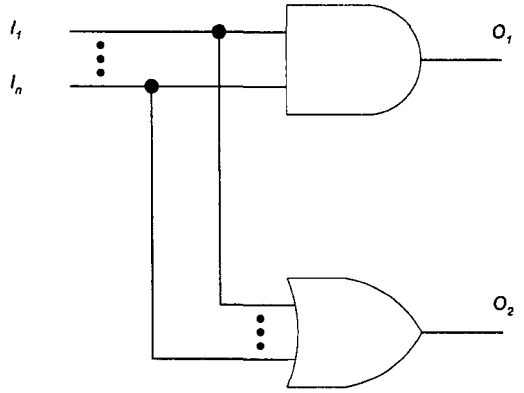


그림 1. 다중 가중치 집합이 필요한 예
Fig. 1. Example requiring multiple weight sets.

식 (1)을 이용해서 t_1과 t_2에 대한 샘플링 확률 P_1과 P_2를 각각 구해보면 $P_1 = 0.5^n$, $P_2 = (1-0.5)^n$이 된다. 즉 P_1과 P_2 모두 0.5^{-n}이 된다. 샘플링 확률 P_1과 P_2의 값은 매우 낮아서 가중치 집합 W를 사용하여 t_1과 t_2를 샘플링하기 위해서는 엄청나게 많은 수의 가중 무작위 테스트 패턴을 가해야만 함을 알 수 있다. 따라서 t_1과 t_2를 같은 테스트 패턴 집합에 동시에 포함시켜 가중치를 계산하는 것은 불합리함을 알 수 있다. t_1에 대한 가중치 집합과 t_2에 대한 가중치 집합을 따로 구하면 훨씬 적은 수의 가중 무작위 테스트 패턴으로 O_1의 논리값 0 고착고장과 O_2의 논리값 1 고착고장을 검출할 수 있다. 여기서 우리는 t_1과 t_2 사이에는 같은 비트의 논리값이 서로 다른 충돌(conflict)이 너무 많아서 하나의 테스트 패턴 집합으로 묶일 수 없다는 것을 알 수 있다. 즉 두 패턴 사이의 해밍거리가 너무 크기 때문에 양질의 가중치 집합을 생성해 낼 수 없는 것이다.

예에서 보았듯이 특정한 회로에 대해서 단일 가중치 집합만을 사용해서는 높은 고장 검출율을 달성할 수 없다. 다중 가중치 집합에 기초한 방법의 단점은 하드웨어 오버헤드가 크다는 것이다.

가중치 계산의 소스에 따라 또 다른 분류법이 존재한다. 하나는 구조적 분석에 의한 방법[6,9]이고 또 다른 하나는 결정론적인 테스트 패턴을 기초로 한 방법[5,10]이다. 전자의 경우 자동 테스트 패턴 생성(Automatic Test Pattern Generation)과정에서 사용될 수 있다는 장점이 있다. 그러나 충분한 고장 검출율을 적절한 시간 내에 달성할 수 없다는 단점이 있다. 후자의 경우 자동 테스트 패턴 생성 시스템에 사용될 수는 없지만 높은 고장 검출율을 효율적으로 달성할 수 있다.

우리는 높은 고장 검출율을 달성하기 위해서 다중 가중치 집합과 결정론적인 테스트 패턴 집합을 기초로 한 방법을 채택했다. 이 논문은 적당한 하드웨어를 사용하여 적절한 시간 내에 높은 고장 검출율을 달성하는 것을 목표로 한다.

Ⅲ. 샘플링 확률

하나의 결정론적인 테스트 패턴 t_j가 하나의 가중 무작위 패턴과 같을 확률을 P_j라 하자. N개의 가중 무작위 패턴을 가한 후에 한 개의 결정론적인 테스트 패턴이 샘플링되지 않을 확률은 $(1-P_j)^N$이 된다. N개의 가중 무작위 패턴에 의해 t_j가 샘플링될 확률 P_{Nj}를 구하면

$$P_{Nj} = 1-(1-P_j)^N \tag{2}$$

이 된다.

식(2)의 양변에 로그를 취한 후 N에 대해 정리하면

$$N = \frac{\log(1-P_{Nj})}{\log(1-P_j)} \tag{3}$$

이 된다.

식(3)으로부터 결정론적인 테스트 패턴 집합을 샘플링하는데 필요한 가중 무작위 패턴의 수는 매우 낮은 샘플링 확률을 갖는 결정론적인 테스트 패턴에 의해 주로 결정지어 진다는 사실을 알 수 있다. 다시 말해서 매우 낮은 샘플링 확률을 갖는 결정론적인 테스트 패턴을 샘플링하기 위해서 가중 무작위 패턴 생성 시스템은 많은 수의 가중치 테스트 패턴을 소모하게 되는 것이다. 그래서 우리는 이러한 낮은 샘플링 확률들을 어느 정도 수준까지 향상시켜서 그러한 샘플링 확률을 갖는 결정론적인 테스트 패턴들이 쉽게 샘플링되

게 하는 새로운 가중치 집합 최적화 알고리듬을 제안하고자 한다.

압축(compaction)을 수행하지 않은 결정론적인 테스트 패턴 내에는 상당수의 무상관 비트가 존재하게 된다. 우리는 이 무상관 비트를 낮은 샘플링 확률들이 증가되도록 편향시켜서 최적화된 가중치 집합을 찾고자 한다. 낮은 샘플링 확률을 갖는 결정론적인 테스트 패턴이 증가함에 따라 높은 샘플링 확률을 갖는 결정론적인 테스트 패턴들이 감소하는 것은 당연한 일이다. 그러나 높은 샘플링 확률을 갖는 결정론적인 테스트 패턴들은 낮은 샘플링 확률을 갖는 결정론적인 테스트 패턴을 샘플링하기 위해서 가하는 가중 무작위 패턴에 의해 샘플링될 가능성이 크다. 따라서 매우 낮은 샘플링 확률들을 상당한 수준까지 향상시킬 수 있다면 높은 고장 검출율을 달성하는데 필요한 가중 무작위 패턴의 길이는 매우 짧아질 것이다. 이렇게 편향된 테스트 패턴 집합을 사용해서 생성한 가중치 집합들은 매우 적은 수의 가중 무작위 테스트 패턴으로 목표로 삼았던 결정론적인 테스트 패턴을 모두 샘플링할 수 있다.

$T = \{t_1, t_2, \cdots, t_n\}$을 하나의 테스트 패턴 집합이라고 하고 $t_j[i]$를 하나의 테스트 패턴 t_j의 i번째 비트라고 가정하자. 이 때 i번째 비트의 가중치는 다음 식에 의해 구할 수 있다.

$$w_i = \frac{|\{t_j \in T | t_j[i] = 1\}|}{|\{t_j \in T | t_j[i] \neq X\}|} \quad (4)$$

Ⅳ. 가중치 집합 최적화

그림 2에 가중치 집합 최적화 알고리듬을 나타내었다. 가장 먼저 하나의 결정론적인 테스트 패턴 집합이 ATPG 시스템에 의해 구해진다. 그리고 결정론적인 테스트 패턴들간의 해밍거리를 고려하여 구한 테스트 패턴 집합을 여러 개의 부분 집합으로 분할한다. 이 때 가장 크기가 큰 부분 집합만이 가중치 집합을 구할 때 사용된다. 만약에 두 개의 패턴이 너무 큰 해밍거리를 갖는다면 두 개의 패턴 사이에 너무 많은 충돌이 존재함을 의미하므로 하나의 테스트 패턴 부분집합에 포함시킬 수 없다. 하나의 결정론적인 테스트 패턴과 테스트 패턴 집합에 속하는 나머지 테스트 패턴들간의 해밍거리를 모두 구한다. 구해진 모든 해밍거리가 최대 해밍거리(maximum Hamming distance)보다 작을 때에만 이 테스트 패턴은 그 테스트 패턴 집합에 포함될 수 있다. 이 때 가중 무작위 테스트 패턴 생성에서 최대 해밍거리는 가중치 집합의 성능을 결정하는데 있어서 매우 중요한 요소이다. 가중 무작위 테스트 패턴의 길이와 가중치 집합의 수는 최대 해밍거리에 의해 크게 영향을 받는다. 가중 무작위 패턴의 길이의 경우는 최대 해밍거리에 비례하는 경향이 있다. 하지만 작은 값의 최대 해밍거리는 너무 많은 수의 가중치 집합을 생성하게 된다. 따라서 최대 해밍거리 역시 생성된 가중치 집합이 가중 무작위 패턴의 길이와 가중치 집합의 수라는 관점에서 모두 효율적일 수 있도록 최적화되어야만 할 것이다. 우리는 최적화된 최대 해밍거리를 효율적으로 찾는 새로운 방법을 제안한다.

```
set_optimization_wrpg()
{
  while ( fault_coverage < target_fault_coverage )
  {
    deterministic_testgen();        // 결정론적인 테스트 패턴을 ATPG를 통해 구함

    divide_test_set();              // 구해진 테스트 집합을 해밍거리에 따라 여러
                                    개의 부분 집합으로 분리한 후 가장 큰 부분
                                    집합을 선택

    generate_weight_set();          // 가장 큰 부분 집합을 이용하여 가중치 집합을
                                    식 (4)에 의해 구함

    calculate_sampling_prob();      // 샘플링 확률을 계산한 후 가장 낮은 샘플링
                                    확률을 갖는 두 개의 테스트 패턴을 구함

    do
    {
      find_modi_bits();             // 가장 낮은 샘플링 확률을 갖는 두 개의 패턴을
                                    이용하여 수정될 비트의 위치를 결정
      bias_X_value();               // 가장 낮은 샘플링 확률값이 증가하도록 무상관
                                    비트를 편향시킴
      regenerate_weight_set();      // 편향된 결정론적인 테스트 패턴을 이용해 새로운
                                    가중치 집합을 식 (4)에 의해 구함
      calculate_sampling_prob();    // 편향되지 않은 결정론적인 테스트 패턴에
                                    대한 샘플링 확률을 생성

    }while ( Pnew > Pold)           // 새로운 가장 낮은 샘플링 확률이 이전의 가장 낮은
                                    샘플링 확률보다 높으면 반복, 그렇지 않으면 중지

    fault_coverage = wrpg_fault_simulation();  // 최적화된 가중치 집합을 이용하여
                                               가중치 무작위 패턴 생성과 고장
                                               시뮬레이션을 수행
  }
}
```

그림 2. 가중치 집합 최적화 알고리듬
Fig. 2. Optimization weight sets and fault simulation.

미리 계산된 결정론적인 테스트 패턴 집합으로부터 분리되어진 부분 집합 중에서 가장 큰 것에 대해 식 (4)를 이용하여 가중치 집합을 생성한 후 각각의 결정

론적인 테스트 패턴의 샘플링 확률을 식(1)에 의해 구한다.

모든 결정론적인 패턴의 샘플링 확률을 구한 후 가장 낮은 샘플링 확률을 갖는 두 개의 테스트 패턴을 찾는다. 이 두 개의 패턴에 대하여 같은 구체적 논리값을 갖는 비트들을 구한다. 가중치 계산에 이용되는 결정론적인 테스트 패턴 집합에 포함되어 있는 패턴들이 이 비트들에 대하여 무상관 비트이면 이 것들의 일부를 두 개의 가장 낮은 샘플링 확률을 갖는 결정론적인 테스트 패턴들이 공통으로 갖는 논리값으로 편향시킨다. 편향된 결정론적인 테스트 패턴들을 이용해 가중치 집합을 새롭게 계산하고 편향되기 전의 원래 패턴들에 대한 샘플링확률을 새로운 가중치를 이용하여 구한다. 그리고 나서 가장 낮은 샘플링 확률을 갖는 두 개의 테스트 패턴을 다시 구한다. 새롭게 생성된 최저 샘플링확률 P_{new}가 이전의 최저 샘플링확률 P_{old} 보다 크면 이 과정은 반복되고 그렇지 않으면 중단된다.

테스트 패턴 집합에 표 1에 나타난 t_1, t_2, t_3, t_4의 4개의 패턴이 존재한다고 가정하자. 식(4)에 의해 구해진 가중치는 표 1의 마지막 줄에 나타내었다. 식(1)을 이용해 각 테스트 패턴에 대한 샘플링 확률을 구해보면 표 2와 같다. t_1과 t_3이 가장 낮은 샘플링 확률을 가지므로 두 패턴에 대해 같은 구체적인 논리값을 갖는 비트를 구한다. 표 3에서 나타난 것처럼 t_1과 t_3은 1, 4 비트에서 각각 동일한 논리값 1, 1을 갖는다. 이 때 비트 6도 같은 논리값은 갖지만 무상관 비트이므로 편향시킬 목록에서 제외한다. t_1과 t_3을 제외한 테스트 패턴에 대해 1, 4 비트에 편향시킬 수 있는 무상관 비트가 존재하는지 조사한다. 표 4에 나타난 것처럼 편향시킬 수 있는 무상관 비트는 상당수 존재한다. 하지만 이 모든 무상관 비트를 편향시키면 편향의 정도가 너무 심해져서 매우 낮은 샘플링 확률을 갖는 테스트 패턴이 파생될 것이다. 따라서 그 중에 일부분만을 골라 선택적으로 편향시킨다. 여기에서는 t_4의 첫 번째 비트의 논리값 X만 논리값 1로 편향시킨다.

표 5와 표 6에 나타낸 것처럼 편향되어진 테스트 패턴에 대해 새로운 가중치를 구하고 이 가중치를 이용해 샘플링 확률을 새로 구한다. 여기서 편향하기 전의 최저 샘플링 확률은 1/27이었으나 편향되고 난 후의 최저 샘플링 확률은 1/24이 됨을 알 수 있다. 최저 샘플링 확률이 증가되었으므로 이 과정은 반복된다.

표 1. 무상관 비트의 편향 전의 가중치
Table 1. Weights before biasing don't care bits.

	1	2	3	4	5	6
t_1	1	0	1	1	0	X
t_2	0	X	X	X	0	1
t_3	1	1	0	1	1	X
t_4	X	0	X	1	X	X
가중치	2/3	1/3	1/2	1	1/3	1

표 2. 상관 비트 편향 전의 샘플링 확률
Table 2. Sampling probabilities before biasing don't care bits.

	t_1	t_2	t_3	t_4
샘플링 확률	4/27	2/9	1/27	2/3

표 3. 편향할 비트와 논리값
Table 3. The bits and logic values to be biased.

	1	2	3	4	5	6
t_1	1	0	1	1	0	X
t_3	1	1	0	1	1	X
편향 비트	1	·	·	1	·	·

표 4. 편향시킬 수 있는 무상관 비트
Table 4. Don't care bits able to be biased.

	1	2	3	4	5	6
편향 비트	1	·	·	1	·	·
t_2	0	X	X	X(1)	1	1
t_4	X(1)	0	X	1	X	X

표 5. 무상관 비트의 편향 후의 가중치
Table 5. Weights after biasing don't care bits.

	1	2	3	4	5	6
t_1	1	0	1	1	0	X
t_2	0	X	X	X	0	1
t_3	1	1	0	1	1	X
t_4	1(X)	0	X	1	X	X
가중치	3/4	1/3	1/2	1	1/3	1

표 6. 무상관 비트 편향 후의 샘플링 확률
Table 6. Sampling probabilities after biasing don't care bits.

	t_1	t_2	t_3	t_4
샘플링 확률	1/6	1/6	1/24	1/2

가중치 집합의 최적화 과정이 끝나면 최적화된 가중치 집합을 이용하여 가중 무작위 패턴이 생성과 고장 시뮬레이션이 수행된다. 만족할 만한 고장 검출율이 달성될 때까지 이전의 전과정이 반복된다.

V. 해밍거리(Hamming distance)

앞에서 보았듯이 해밍거리가 큰 테스트들을 같은 테스트 패턴 집합에 함께 포함시켜 가중치를 계산하면 양질의 가중치 집합을 생성할 수 없다. 따라서 최대 해밍거리를 정해서 테스트 패턴 집합에 속한 임의의 패턴들과의 해밍거리가 최대 해밍거리보다 작은 테스트 패턴만 그 테스트 패턴 집합에 포함시킨다. 최대 해밍거리의 값이 변함에 따라 가중 무작위 패턴 생성 및 고장 시뮬레이션의 실험결과가 많은 영향을 받게 된다. 따라서 그림 3에 최대 해밍거리의 최적값을 효율적으로 찾아내는 새로운 알고리듬을 나타내었다.

*Init_Hdis, Test_length, Num_inputs*를 각각 최대 해밍거리의 초기값, 결정론적 테스트 패턴의 길이, 회로의 주입력의 수라고 두자. 여러 가지의 실험을 통해서 최대 해밍거리의 최적값은 결정론적 테스트 패턴의 거리와 회로의 주입력의 수에 밀접한 상관관계가 있음을 알 수 있었다. 그래서 우리는 다음 식에 따라 최대 해밍거리의 초기값을 구한다.

$$Init\_Hdis = CN \times Num\_inputs + CL \times Test\_length \quad (5)$$

*CN*과 *CL*은 각각 주입력의 수와 결정론적 테스트 패턴 길이에 대한 상수값이고 이것은 실험적 결과 분석을 통해서 정해진다. 최대 해밍거리의 초기값을 이용해 가중 무작위 패턴 생성과 고장 시뮬레이션을 수행한 후에 최대 해밍거리를 1만큼씩 증가시키면서 가중 무작위 패턴 생성과 고장 시뮬레이션을 *CI*번 반복한다. *CI*번만큼의 고장 시뮬레이션 중 가중 무작위 패턴의 길이와 가중치 집합의 수의 관점에서 성능 향상을 보이는 결과가 있으면 이전의 모든 과정이 반복된다. 때때로 가중치 집합의 수는 증가하고 가중 무작위 패턴의 길이는 감소하거나 반대로 가중치 집합의 수는 감소하고 가중 무작위 패턴의 길이는 증가하는 경우가 있다. 이전까지의 가중치 무작위 생성과 고장 시뮬레이션의 결과 중 가장 좋은 성능을 나타내는 경우의 가중 무작위 패턴의 수가 *l*이고 가중치 집합의 수가 *s*라고 하면 하나의 가중치 집합의 감소는 *l*/*s*개의 무작위 패턴 길이의 감소와 같은 성능 향상으로 평가하기로 했다. 예를 들어 *l*이 8000이고 *s*가 4인 경우에 한 개의 가중치 집합이 증가하고 2500개의 가중 무작위 패턴이 감소하면 *l*/*s*인 2000보다 많은 수의 가중 무작위 패턴이 감소했으므로 성능이 향상된 것으로 판별한다.

```
find_max_Hdis()
{
  Init_Hdis = CN*Num_inputs + CL*Test_length;   // 식 (5)를 이용하여 최대해밍거리의 초기값 구함
  max_Hdis = Init_Hdis;                         // 최대해밍거리를 초기화시킴
  do
  {
    for ( i=0; i<CI ;i++ )                      // 최대 해밍거리를 증가시키면서 CI번 반복
      {
        weight_set_optimization_wrpg();  // 그림 2의 가중치 집합 최적화와 가중 무작위 패턴
                                         생성을 수행 후 결과 저장
        max_Hdis++;                             // 최대해밍거리를 증가시킴
      }

   if (is_not_first_iteration)   // while 루프에서 첫번째 수행이 아니면 이전의 결과를 저장
      old_inc_performance = new_inc_performance;

   new_inc_performance = evaluate_result();   // 가중치 집합 최적화와 가중 무작위 패턴 생성의
                                              결과 평가

  } while ( new_inc_performance > old_inc_performance ) // 성능이 향상되는 동안 반복

  max_Hdis = Init_Hdis-1;                               // 최대해밍거리를 다시 초기화시킴
  do
  {
      for ( i=0; i<CI ;i++ )                    // 최대 해밍거리를 감소시키면서 CI번 반복
      {
        weight_set_optimization_wrpg();  // 그림 2의 가중치 집합 최적화와 가중 무작위 패턴
                                         생성을 수행 후 결과 저장
        max_Hdis--;                             // 최대해밍거리를 감소시킴
      }

  if (is_first_iteration)          // while 루프에서 첫 번째 수행이면 old_dec_performance에 앞의
                                   최대 해밍거리 증가 루프에서의 결과값인 old_inc_performance를
                                   입력
      old_dec_performance = old_inc_performance;
  else                             // while 루프에서 첫 번째 수행이 아니면 이전 루프의 결과를 저장
      old_dec_performance = new_dec_performance;

  new_performance = evaluate_result();      // 가중치 집합 최적화와 가중 무작위 패턴
                                            생성의 결과 평가

  }while ( new_dec_performance > old_dec_performance ) // 성능이 향상되는 동안 반복
}
```

그림 3. 최대 해밍거리의 최적값을 찾는 알고리듬
Fig. 3. Algorithm of finding maximum Hamming distance.

이 과정이 끝나면 최대 해밍거리를 식(5)에 의해 구한 값보다 1 작은 값으로 초기화시킨 후 1씩 감소시키면서 똑같은 과정을 반복한다. 이러한 방법을 통해 최대 해밍거리의 최적값을 적당한 시간 내에 효율적으로 찾아낼 수 있다.

VI. 실험결과

새로운 가중치 집합 최적화 알고리듬을 이용한 가중 무작위 패턴 생성 시스템을 구현하였고 ISCAS 85 벤치마크 회로에 대해서 고장 시뮬레이션을 수행하였다. 의사 무작위 패턴에 의해 고장이 쉽게 검출되는 회로의 경우 가중 무작위 테스트를 적용할 필요가 없으므로 이런 회로에 대한 실험결과는 커다란 의미를 부여하지 못한다.

새로운 가중치 집합 최적화 알고리듬을 채택한 경우와 그렇지 않은 경우에 대한 최저 샘플링확률의 비교를 표 7에 나타내었다. '최적화'는 가중치 집합 최적화 알고리듬을 채택한 경우를 나타내고 '정상'은 채택하지 않은 경우를 의미한다. 예상했던 대로 c499를 제외한 모든 경우에 최저 샘플링확률은 상당히 증가했다. c499의 경우 결정론적인 테스트 패턴에 무상관 비트가 거의 존재하지 않는다. 따라서 우리의 가중치 최적화 알고리듬을 적용시킬 수가 없었다.

표 7. 조합회로들의 최저 샘플링 확률
Table 7. The lowest sampling probabilities of combinational circuits.

회로	정상	최적화
c432	3.89×10^{-7}	1.28×10^{-6}
c499	6.914×10^{-7}	6.914×10^{-7}
c880	1.47×10^{-6}	3.39×10^{-6}
c1355	2.88×10^{-12}	2.90×10^{-12}
c1908	2.73×10^{-10}	4.45×10^{-10}
c2670	6.00×10^{-15}	6.53×10^{-13}
c3540	1.41×10^{-9}	2.15×10^{-9}
c5315	6.80×10^{-11}	1.29×10^{-10}
c6288	3.25×10^{-13}	1.90×10^{-12}
c7552	2.02×10^{-54}	2.13×10^{-54}

ISCAS 85 벤치마크 회로에 대해서 가중 무작위 패턴 생성과 고장 시뮬레이션이 수행되었다. 표 8에 의사 무작위 테스트 방법, X_test[5] 그리고 가중치 집합 최적화 알고리듬을 채택한 새로운 방법의 성능비교를 나타내었다. 패턴 생성 과정에서 k개의 연속적인 패턴이 새로운 고장을 하나도 검출해 내지 못하면 패턴 생성은 중단된다. 의사 무작위 패턴 테스트의 경우는 패턴 생성이 완전히 중지되고 가중 무작위 테스트는 새로운 가중치 집합에 대한 가중 무작위 패턴 생성이 시작된다. 표 8는 k값이 1024인 경우이다. k의 값이 커지면 가중치 집합의 수는 감소하지만 가중 무작위 패턴의 길이는 증가하게 된다. 반대로 k의 값이 작아지면 가중 무작위 패턴의 길이는 감소하지만 가중치 집합의 수는 경향이 있다. 따라서 적절한 k값을 선택해야 최적화된 가중치 집합을 구할 수 있다.

표 8. 고장 시뮬레이션 결과
Table 8. Results of fault simulation.

회로	의사 무작위			X_test			최적화		
	가중치 집합의 수	패턴 수	고장 검출율 [%]	가중치 집합의 수	패턴 수	고장 검출율 [%]	가중치 집합의 수	패턴 수	고장 검출율 [%]
c432	1	832	99.8	1	1512	100	1	1408	100
c499	1	800	100	1	1251	100	1	1216	100
c880	1	4032	99.9	1	796	100	1	768	100
c1355	1	2625	100	2	4397	100	1	3455	100
c1908	1	6624	99.9	3	5452	100	2	5120	100
c2670	1	2752	88.1	3	9289	100	3	4352	100
c3540	1	3232	99.7	3	3575	100	2	3488	100
c5315	1	2784	100	1	2138	100	1	2624	100
c6288	1	160	100	1	74	100	1	64	100
c7552	1	6144	95.9	6	17046	100	6	8960	100

c432, c499, c880, c1355, c1908, c5315, c6288의 경우 의사 무작위 테스트로도 높은 고장검출율을 적은 수의 테스트 패턴으로 구할 수 있다. 즉 이러한 회로들은 의사 무작위 패턴 저항 고장들을 거의 가지고 있지 않다. 그러나 c2670, c3540, c7552의 경우 의사 무작위 테스트 방법으로는 충분한 고장 검출율을 달성할 수 없음을 알 수 있다.

c5315를 제외한 모든 회로에 대해 가중치 집합 최적화 알고리듬을 채택한 경우가 그렇지 않은 경우 보다 훨씬 좋은 성능을 보여 주었다. 즉 같거나 적은 수의 가중치 집합에 대해서 훨씬 적은 수의 가중 무작위 테스트 패턴으로 같은 고장을 달성할 수 있는 것이다. c5315의 경우 의사 무작위 테스트 패턴만으로 쉽게

테스트되기 때문에 가중치 최적화 알고리듬의 영향이 반영되기 힘들다. 따라서 예상되었던 성능향상이 이루어지지 않았다고 생각된다.

표 9. 최대 해밍거리의 초기값과 최적값의 비교

Table 9. The initial and optimal value of maximum Hamming distance.

회로	주입력 수	테스트 수	초기값	최적값
c432	36	152	7	10
c499	41	95	6	9
c880	60	366	13	15
c1355	41	127	7	15
c1908	33	129	7	14
c2670	233	720	38	36
c3540	50	710	19	20
c5315	178	1380	45	40
c6288	32	170	7	9
c7552	207	933	39	41

표 9에 식(5)에 의해 구해진 최대 해밍거리의 초기값과 실제 구해진 최대 해밍거리의 최적값을 나타내었다. 대부분의 경우에 최대 해밍거리의 초기값과 실제로 구해진 최대 해밍거리의 최적값 사이의 차이는 아주 미미함을 알 수 있다. 이것은 이 논문에서 제안된 최대 해밍거리 최적값을 찾는 알고리듬의 효율성을 증명해 준다.

Ⅶ. 결 론

가중 무작위 패턴 테스트에서 가중치 집합의 성능은 테스트 시간과 하드웨어 오버헤드에 상당한 영향을 미치는 매우 중요한 요소이다. 우리는 최적화된 가중치 집합을 효율적으로 생성할 수 있는 알고리듬을 개발하였다. 가중치 집합의 성능을 결정하는 중요한 요소인 최대 해밍거리의 최적화된 값을 빠른 시간 내에 찾아내는 방법을 고안하여 가중치 집합 최적화 알고리듬에 적용하였다. 벤치마크 회로에 대한 실험 결과들은 새로운 가중치 집합 최적화 알고리듬이 고성능의 가중치 집합을 생성한다는 사실을 입증하여 주었다. 즉 같은 수의 가중치 집합에 대하여 훨씬 적은 수의 가중 무작위 패턴을 가하여 같은 고장 검출율을 달성할 수 있는 것이다. 따라서 이 가중치 집합 최적화 알고리듬을 가중 무작위 패턴 테스트에 적용하면 그렇지 않은 경우에 대해 동일한 하드웨어 오버헤드로 보다 짧은 시간 내에 높은 고장 검출율을 달성할 수 있다.

참 고 문 헌

[1] J. Hartmann and G. Kemnitz, "How to Do Weighted Random Testing for BIST" in Proc. of International Conference on Computer Aided Design, 1993, pp. 568-571.

[2] C-A. Chen and S. K. Gupta, "A Methodology to Design Efficient BIST Test Pattern Generation System" in Proc. of International Test Conference, 1994, pp. 491-500.

[3] R. Kapur, S. Patil, T. J. Snethen and T. W. Williams, "Design of an Efficient Weighted Random Pattern Generation System" in Proc. of International Test Conference, 1994, pp. 491-500.

[4] D. J. Neebel and C. R. kime, "In-homogeneous Cellular Automata for Weighted Random Pattern Generation" in Proc. of International Test Conference, 1993, pp. 1013-1021.

[5] B. Reeb and H-J. Wunderlich, "Deterministic Pattern Generation for Weighted Random Pattern Testing" in Proc. of European Design and Test Conference, 1996, pp. 30-36.

[6] M. Bershteyn, "Calculation of Multiple Sets of Weighted Random Testing" in Proc. of International Test Conference, 1993, pp. 1031-1040.

[7] M. A. Miranda and C. A. Lopez-Barrio, "Generation of Optimized Single Distributions of Weights for Random Built-In Self-Test" in Proc. of International Test Conference, 1993, pp. 1023-1030.

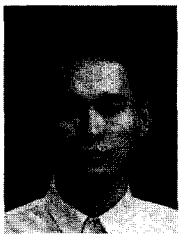
[8] D. J. Neebel and C. R. Kime, "Multiple Weighted Cellular Automata" in Proc. of VLSI Test symp., 1994, pp. 81-86.

[9] R. Lisanke, F. Brglez, A. J. Degeus and

D. Gregory, "Testability Driven Random Test Pattern Generation" in Trans. on Computer Aided Design, 1987, pp. 1082-1087.

[10] F. Muradali, V. K. Agarwal and B. Nadeau-Dostie, "A New Procedure for Weighted Random Built-In Self-Test" in Proc. of International Test Conference, 1990, pp. 660-669.

저 자 소 개



李 沆 圭(正會員)

1997년 2월 연세대학교 전기공학과 졸업(공학사). 현재 연세대학교 대학원 전기공하과 석사과정. 주관심분야는 테스팅, DFT와 VLSI & CAD 등임

金 弘 植(正會員)

1997년 2월 연세대학교 전기공학과 졸업(공학사). 현재 연세대학교 대학원 전기공하과 석사과정. 주관심분야는 테스팅, DFT와 VLSI & CAD 등임

姜 成 昊(正會員)

1986년 2월 서울대학교 제어계측공학과 졸업(공학사). 1988년 5월 The Univ. of Texas at Austin. Electrical and Computer Eng. 졸업(공학 석사). 1992년 5월 The Univ. of Texas at Austin Electrical and Computer Eng. 졸업(공학 박사). 1989년 11월 ~ 1992년 8월 미국 Schlumberger Inc. Research Scientist. 1992년 9월 ~ 1992년 10월 미국 The Univ. of Texas at Austin Post Doctoral Fellow. 1992년 8월 ~ 1994년 6월 미국 Motorola Inc. Senior Staff Engineer. 1994년 9월 ~ 현재 연세대학교 전기공학과 조교수