조합논리회로를 위한 효율적인 테스트 컴팩션 알고리즘

(Efficient Test Compaction Algorithms for Combinational Logic Circuits)

김 윤 홍 †

(Yun-Hong Kim)

요 약 본 논문에서는 조합논리회로의 테스트 컴팩션을 위한 두 가지 효율적인 알고리즘을 제안한다. 제안된 알고리즘들은 각각 동적인 컴팩션 기법과 정적인 컴팩션 기법을 사용하고 있으며, 실험을 위해 기존의 ATPG시스템인 ATALANTA에 통합 구현하였다. ISCAS85와 ISCAS89(완전스캔 버전) 벤치마크 회로에 대한 실험에서 본 시스템은 기존에 발표된 다른 컴팩션 알고리즘에 비하여 보다 작은 테스트 집합을 보다 빠르게 생성하였으며, 실험 결과를 통하여 제안된 알고리즘들의 유효성을 입증할 수가 있었다.

Abstract This paper proposes two efficient test compaction algorithms for combinational circuits. These algorithms use a dynamic compaction technique and a static compaction technique, respectively, and are incorporated into the existing high-performance ATPG system, called ATALANTA, for experiments. The integrated system generated smaller test sets than the other conventional algorithms did for the benchmark circuits from ISCAS85 and ISCAS89(the full scan version) in less CPU times. Experimental results show the effectiveness for the two proposed algorithms.

1. 서 론

디지털 회로를 위한 자동 테스트패턴 생성 시스템(Automatic Test Pattern Generation System, 이하 ATPG)에 관한 연구는 지난 수 십년 동안 광범위하게 이루어졌다. 이 분야에서의 주된 연구는 모든 고장을 검출하는 전체 테스트패턴 집합(이하 테스트 집합)을 어떻게 효율적으로 생성할 것인가에 집중되었다.[1]-[4] 그러나 테스트 인가(test application)에 따른 비용과 시간은 테스트 집합 크기에 비례하여 증가한다. 특히 스캔 설계된 회로의 경우에는 테스트 인가 시간이 스캔 플립플롭 수와 테스트 집합 크기의 곱에 비례하여 증가하기 때문에, 컴팩트한 테스트 집합의 생성은 더욱 중요해지고 있다. 따라서 테스트 집합 크기를 줄이기 위한 추가적인 처리를 하더라도 경제적인 칩 테스트을 위해서 컴팩트한 테스트 집합이 매우 바람직하다. 더욱이 테스트 집합 크기가 줄어들면 자동 테스트 장비(ATE)의 버퍼 크기도 작아지게 되고, 또한 칩에 내장되는 테스트 하드웨어의 부담도 줄어들게 된다.

Irredundant한 조합논리회로에서 모든 단일 stuck-at고장을 검출하는 최소 테스트 집합을 계산하는 문제의 복잡도는 NP-hard하다[5]고 이미 증명되었기 때문에, 테스트 집합의 컴팩션은 쉬운 작업이 아니다. 지금까지 서로 다른 휴리스틱들을 이용한 몇 가지 테스트 컴팩션 방법들이 제안되었는데[6]-[12], 이런 방법들은 크게 정적인 방법과 동적인 방법으로 나눌 수 있다.

동적인 컴팩션 방법은 테스트패턴이 생성되는 동안에 적용되는 방법으로서, 각 테스트패턴이 보다 많은 고장을 검출하도록 하여 전체 테스트패턴 수를 줄이려는 것이다. 여기에는 random fill[9], maximal compaction[10], rotating backtrace[10] 등의 컴팩션 방법들이

· 본 논문은 상명대학교 '99년도 교내연구비의 지원을 받아 연구되었음.

† 정 회 원 : 상명대학교 컴퓨터정보통신공학부 교수

yhkim@smuc.sangmyung.ac.kr

논문접수 : 2000년 8월 10일

심사완료 : 2001년 1월 26일

속한다. 동적인 컴팩션 방법의 경우에 전체 실행시간은 일반적으로 대상고장의 수에 따라 증가한다. 정적인 컴팩션 방법은 테스트 집합이 우선 생성된 후에 이를 압축하기 위해 사용하는 방법으로서, 실행시간은 일반적으로 초기 테스트 집합의 크기에 비례하여 증가한다. 여기에는 reverse order fault simulation[4], essential fault pruning[8], ROTCO[11], double detection[12], two-by- one[12], two-by-three[12] 등의 기법들이 속한다.

본 논문에서는 테스트 컴팩션을 위한 새로운 동적인 컴팩션 알고리즘과 정적인 컴팩션 알고리즘을 제안한다. 새로운 동적인 컴팩션 알고리즘으로서 제안되는 중복패턴 제거 알고리즘은 테스트패턴 생성 과정 중에 발생되는 중복패턴(redundant test pattern)을 확인하기 위해 고장 시뮬레이션(fault simulation)을 수행하고 이로부터 얻어지는 정보를 이용하여 동적으로 테스트 집합에서 중복패턴을 제거한다. 새로운 정적인 컴팩션 알고리즘인 주고장 축소 알고리즘은 초기 테스트 집합을 두 부분으로 분할하고 테스트패턴 당 고장검출율이 낮은 부분에 대해서만 컴팩션 과정을 적용하는데, 각 테스트패턴에 대한 주고장(essential fault)을 하나씩 줄여나가도록 하여 최종적으로 해당 테스트패턴을 중복패턴으로 만들어 제거하는 방법이다. 제안된 테스트 컴팩션 알고리즘들은 Virginia Polytechnic Institute & State University의 하동삼 교수 연구실에서 개발한 ATPG시스템인 ATALANTA[13]에 통합하여 구현되었으며, ISCAS85[14]와 ISCAS89[15] 완전스캔 버전의 벤치마크회로에 대한 실험을 통하여 제안된 알고리즘의 성능이 우수함을 확인하였다.

본 논문의 구성은, 우선 2절에서 테스트 컴팩션의 기본 개념과 용어에 대하여 소개하고, 3절에서는 중복패턴 제거 알고리즘에 대하여 자세히 설명한다. 주고장 축소 알고리즘은 4절에서 제시되고, 구현된 테스트 컴팩션 시스템에 대한 실험결과는 5절에서 다루게 되며, 마지막으로 6절에서 결론을 내린다.

2. 테스트 컴팩션의 기본 개념 및 용어 정의

본 절에서는 테스트 컴팩션의 기본 개념을 설명하고 본 논문에서 사용할 용어를 정의한다.

일반적인 테스트패턴 생성 과정에서는, 고장집합 중에서 현재까지 검출되지 않은 고장을 하나 선택한 후 이 고장에 대한 테스트패턴을 생성한다. 이때 생성된 테스트패턴은 대상 고장을 검출하기 위해 최소한의 주입력(primary input)들만이 결정된 상태이며, 나머지 주입력들은 값이 결정되지 않은 X상태이다.

따라서 통상적으로 ATPG는 테스트패턴 생성의 부담을 덜기 위해 X값을 임의로 결정한 후, 해당 테스트패턴에 의해 같이 검출될 수 있는 고장들을 확인하기 위해 고장 시뮬레이션을 수행한다. 이 과정을 모든 고장이 검출될 때까지 또는 원하는 고장검출율(fault coverage)이 만족될 때까지 반복함으로써 전체 테스트 집합이 만들어진다.

그러나 ATPG의 실행시간 관점이 아니라 테스트 집합의 크기라는 관점에서 본다면, 위와 같은 과정은 개선되어야 한다. 특히 반도체 회로의 VLSI화에 따라 칩 제조 비용 중에서 테스트 비용이 상당 부분을 차지하고 있는 상황에서, 테스트 집합의 크기가 커지면 테스트 인가 비용이 증가되고 결국 테스트 비용의 증가로 이어지게 된다.

ATPG가 생성한 테스트 집합의 크기가 비교적 큰 근본적인 이유는 다음과 같다. ATPG는 기 생성된 테스트패턴들과의 관계를 고려하지 않고 미검출된 고장만을 고려하여 한 방향으로 진행되기 때문에 전체적인 테스트패턴 수가 증가하게 되는 것이다. 한 테스트패턴이 생성된 시점에서는 그 패턴이 최선일 수 있지만, 이후에 생성된 테스트패턴들의 검출 고장들과의 중복 범위에 따라 그 패턴이 불필요할 수도 있고 또는 그 패턴을 변경시켜 다른 몇 개의 패턴들을 대치시킬 수도 있는 것이다.

그림 1은 테스트패턴과 검출 고장 범위의 관계를 도식적으로 표현한 것이다. 고장 f1을 검출하는 테스트패턴으로 초기에 생성된 1X1X001은 고장 f1이외에도 상당히 많은 고장을 검출할 수 있는 가능성을 갖고 있지만, X를 각각 1로 결정하여 {1010001, 1011001, 1110001, 1111001}중에서 최종 테스트패턴으로 1110001을 선택하면서 이들 중 일부분의 고장만을 검출할 수 밖에 없다.

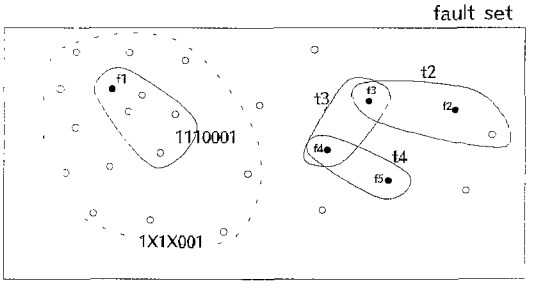

그림 1 테스트패턴과 검출고장과의 관계

따라서 매번 생성되는 테스트패턴마다 X를 효과적으로 결정하여 고장들이 중복 검출되는 것을 최대한 억제한다면 전체 테스트패턴 수는 크게 감소된다.

한편, 고장을 고려하는 순서에 따라서 불필요한 테스트패턴이 발생할 수도 있다. 예를 들어, 그림 1에서 고장 f2를 검출하기 위해서 테스트패턴 t2(f2와 f3를 검출)를 우선 생성한 후, 고장 f4를 검출하는 테스트패턴 t3(f3과 f4를 검출)가 생성된 상황을 가정해 보자. 다음 대상고장으로 f5를 선택하여 테스트패턴 t4(f4와 f5를 검출)를 생성했다면, 결과적으로 f3과 f4는 t2와 t4에 의해서 검출되기 때문에 t3는 새로운 고장 검출에 전혀 기여를 하지 않는 불필요한 테스트패턴일 뿐이다. 이런 테스트패턴은 reverse order fault simulation[4]을 통해서도 제거되지 않는다.

다음은 본 논문에서 사용될 용어를 정의한다. 주어진 테스트 집합 중에 한 테스트패턴이 어느 다른 테스트패턴에 의해서도 검출되지 않는 고장을 적어도 하나 이상 검출한다면, 그 테스트패턴을 *주 패턴*이라고 한다. 그리고 주 패턴에 의해서만 검출되는 고장을 *주 고장*이라고 한다. 따라서 주 패턴은 적어도 하나 이상의 주 고장을 검출한다. 주어진 테스트 집합 중에 한 테스트패턴이 어떠한 주 고장도 검출하고 있지 않다면, 즉 그 패턴에 의해서 검출되는 모든 고장들이 다른 테스트패턴에 의해서 검출된다면, 그 테스트패턴을 *중복 패턴*이라고 한다. 두개의 고장이 동일한 테스트패턴에 의해서 검출될 수 있다면, 이런 고장들을 *호환적*이라 하고, 그렇지 않으면 *비호환적*이라고 한다.

3. 동적인 테스트 컴팩션

자동 테스트패턴 생성 중에 초기의 테스트패턴에 의해 검출된 몇몇 고장들은 나중에 생성된 테스트패턴에 의해 우연히 검출될 수 있다. 따라서 ATPG과정 중에 테스트패턴이 점차 많이 생성됨에 따라 중복 패턴이 발생할 확률도 점점 높아질 수 있다. 본 논문에서는, 테스트패턴 생성 과정 중에 이런 중복 패턴을 확인하고 테스트 집합에서 그때그때 이를 제거하는 중복 패턴 제거 알고리즘을 제안한다.

그림 2에서 기술된 중복패턴 제거 알고리즘은 우선 고장목록에 있는 모든 고장에 대해 고장시뮬레이션을 수행하고, 각 테스트패턴에 의해 검출되는 고장, 각 패턴의 주 고장의 수, 각 고장이 검출된 회수 등을 기록한다. 테스트패턴 생성 과정 중에 패턴의 주 고장 수가 0이 되면, 즉 그 패턴이 중복패턴이 되면, 테스트 집합으로부터 해당 테스트패턴을 제거한다.

예를 들어 그림 1에서 테스트패턴 t_2, t_3, t_4가 각각 검출하는 고장들은 표 1과 같다. 표 1에서 t_2는 f_2를 대상

```
void RemoveRedundantPattern(pattern t)
{
   검출가능한 모든 고장에 대해 테스트패턴 t를 고장시뮬레이션한다;
   for (각 고장 fi, fi {검출된 고장 목록}) {
      ++CountOfDetection[fi];          /* 고장 fi의 검출 회수 */
      if (CountOfDetection[fi] == 1) {
         ++NumOfEssentialFault[t];  /* t의 주 고장 수*/
      } else if (CountOfDetection[fi] == 2) {
fi를 검출하는 테스트패턴 tj (tj t)를 검색한다;
--NumOfEssentialFault[tj];
if (NumOfEssentialFault[tj] == 0) {    /* tj 가 중복 패턴일 경우 */
   테스트 집합에서 tj를 제거한다;
   for (tj에 의해 검출되는 각 고장 fk) {
      --CountOfDetection[fk];
      if (CountOfDetection[fk] == 1) {
fk를 검출하는 테스트패턴 tk(tk tj)를 검색한다;
++NumOfEssentialFault[tk];
                }
             }
          }
       }
    }
```

그림 2 중복 패턴 제거 알고리즘

으로 생성되었고, t_3는 f_4를, t_4는 f_5를 대상으로 생성되었다. 그러나 t_2, t_3, t_4까지 생성된 후에 살펴보면, t_3가 검출하는 f_3, f_4는 t_2와 t_4에 의해서 모두 검출되는 고장이기 때문에 t_3는 불필요한 중복 패턴이 된다. Reverse order fault simulation[4]를 수행하여도 이 경우의 t_3는 제거되지 않는다. 본 논문에서 제안하는 중복 패턴 제거 알고리즘을 적용하면, t_3가 중복 패턴이라는 것이 확인되고 테스트 집합에서 제거된다.

표 1 그림1의 테스트패턴에 대한 검출고장

테스트패턴	검출 고장 목록
t2	f2, f3
t3	f3, f4
t4	f4, f5

중복 패턴 제거 알고리즘에 의해 생성된 테스트 집합은 중복 패턴을 제거하는 처리 순서의 차이 때문에 double detection 알고리즘[12]의 결과와 약간 다를 수 있지만, 성능면에서는 비슷하다. 그러나 본 논문에서 제시하는 컴팩션 과정은 중복 패턴 제거 알고리즘에 의해서만 이루어지는 것은 아니며, 중복 패턴 제거 알고리즘 수행 중에 추출되는 정보, 즉 각 테스트패턴에 의해 검출되는 고장 집합, 각 고장이 현재의 테스트집합에 의해 검출되는 회수 등의 정보들은 다음 단계에서 수행되는 주 고장 축소 알고리즘에 의해 유용하게 활용된다. 따라서 중복패턴 제거 알고리즘 자체의 성능은 기존의 알고리즘과 큰 차이가 없더라도, 중복패턴 제거 알고리즘과 주 고장 축소 알고리즘이 서로 결합된 전체 컴팩션 시스템의 성능은 크게 개선된다.

4. 정적인 테스트 컴팩션

정적인 테스트 컴팩션은 초기 테스트 집합이 우선 생성된 후에, 각 테스트패턴과 이에 대한 주 고장의 관계를 고려하여 주고장들을 가능한 많이 검출하도록 테스트패턴들을 조정함으로써 전체 테스트패턴 수를 줄이는 방식이다.

그림 3은 컴팩션 과정을 통하여 테스트 집합의 각 테스트패턴에 의해 검출되는 주 고장 수의 분포가 일반적으로 어떻게 변화되는지를 보여준다. 일반 ATPG에 의해 생성된 초기 테스트 집합(t_1부터 t_m까지 m개의 테스트패턴)에서는 각 테스트패턴에 의해 검출되는 주 고장 수가 비교적 고르고 넓게 분포되어 있다. 그러나 각 테스트패턴 당 검출되는 주 고장의 수가 많아지도록 기존의 테스트패턴들을 결합, 변경 또는 삭제를 하면, 보다 적은 수의 테스트패턴들(그림 3에서 $n < m$)에 그 분포가 집중되도록 할 수 있다. 바로 이 과정이 컴팩션, 특히 정적인 컴팩션에 의해 수행된다.

정적인 컴팩션은 생성된 각 테스트패턴의 주 고장을 확인하고 이를 다른 테스트패턴으로 옮기기 위해 테스트 집합의 모든 테스트패턴들에 대해 다양한 반복적인 처리를 해야 하는 상당히 많은 계산이 요구되는 알고리즘이다. 따라서 테스트 집합이 크면, 그 크기에 비례하여 상당한 처리시간이 소요된다.

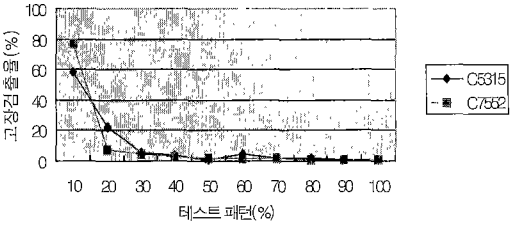

그림 4 두 벤치마크회로에 대한 고장검출율 커브

그림 4는 ISCAS85 벤치마크 회로[14] 중에서 C5315와 C7552에 대한 테스트패턴을 ATALANTA[13]를 통해 생성(백트랙 제한회수 10, reverse order fault simulation사용)하면서 테스트패턴 수의 증가에 따른 고장검출율의 변화를 보여준다. 초기 20%의 테스트패턴이 전체 고장의 80%이상을 검출하고 있다. 이로부터 알 수 있는 사실은 초기 20%의 테스트패턴들은 이미 상당 수의 고장들을 검출하고 있기 때문에, 컴팩션을

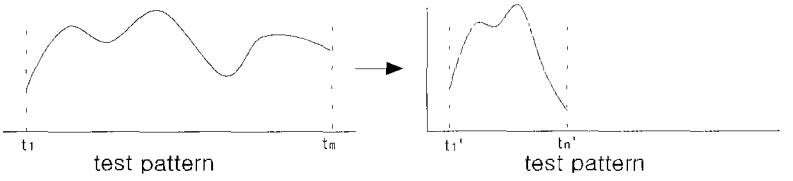

그림 3 컴팩션 과정에 의한 주 고장 수의 분포 변화

위해 다시 고려하는 것은 비효율적이며, 굳이 한다 하여도 큰 효과를 기대하기가 어렵다는 것이다. 나머지 80%의 테스트패턴들은 전체 고장의 불과 20%만을 검출하고 있기 때문에 바로 이 테스트패턴들이 컴팩션을 위해 집중적으로 고려되어야 할 부분이다. 기존의 정적인 컴팩션 알고리즘들은 모든 테스트패턴을 대상으로 하고 있기 때문에 테스트 집합 크기가 감소되는 것에 비해 많은 CPU시간을 사용한다.

본 논문에서는 초기 테스트 집합을 두 부분으로 분할한다. 첫 부분은 테스트패턴 당 고장검출율이 상당히 높기 때문에 정적인 컴팩션 알고리즘을 적용하지 않는다. 그러나 두번째 부분은 패턴 당 고장검출율이 낮기 때문에, 즉 각 테스트패턴이 검출하는 고장 수가 적기 때문에, 정적인 컴팩션 알고리즘을 적용하면 효율적으로 큰 컴팩션 효과를 얻을 수 있다.

테스트 집합의 분할 기준은 원하는 실행시간과 결과 수준에 따라 유동적일 수 있다. 처리시간이 증가하더라도 보다 좋은 컴팩션 결과을 원한다면, 분할을 전혀 하지 않거나 첫 분할부분을 아주 작게 할 수 있다. 만약 처리시간을 가능한 줄이면서 적절한 수준의 컴팩션 결과를 원한다면, 첫 분할부분을 크게 잡을 수도 있다. 따라서 테스트 집합의 분할 기준은 실행시간과 컴팩션 결과 간의 tradeoff를 고려하여 경험적으로 결정하여 사용할 수 있다.

본 논문의 테스트 컴팩션 시스템에서 사용하는 테스트 집합 분할 기준은 전체 고장의 80%까지 검출하는 테스트패턴들을 기준으로 한다. 바꾸어 말하면, 전체 고장의 20%를 검출하는 테스트패턴들만을 대상으로 본 논문에서 제안하는 정적인 테스트 컴팩션 알고리즘을 적용한다.

한 테스트패턴의 주 고장을 하나 제거할 때마다 주 고장 수가 하나씩 감소하게 된다. 따라서 모든 주 고장을 제거하게 되면 해당 테스트패턴은 중복 패턴이 되어, 결국 테스트 집합에서 이를 삭제할 수 있다.

그림 5에 제시된 주고장 축소 알고리즘은 초기 테스트 집합이 생성되면 각 테스트패턴의 주 고장을 가능한 많이 제거하여 테스트 집합을 보다 더 컴팩션하도록 반복적으로 수행된다. 주고장 축소 알고리즘은 복수 대상 테스트 생성(multiple target test generation) 과정[8]을 사용하여 주어진 고장들을 모두 검출하는 테스트패턴을 생성한다. 복수 대상 테스트 생성 과정은 주어진 모든 고장들을 검출하기 위해 각 고장마다 검출을 위해

```
void ReduceEssentialFault(테스트집합 T_sub, 반복회수 NumPass, 최대시도회수 LimitOfFailure)
{
int i;
for (i = 0; i < NumPass; i++) {
  for (T_sub에 속하는 각 테스트패턴 t_j) {
    FlagAllRemoved = TRUE;
    NumOfFailure = 0;
    for (t_j에 의해 검출되는 각 고장 f_k) {
      for (각 테스트패턴 t_i, 여기서 i > j) {
        FlagRemoved = MultipleTargetTestGeneration(t_i, t_j, f_k); /* t_new 패턴 생성 */
        if (FlagRemoved == TRUE) break;
      }
      if (FlagRemoved == TRUE) {
        t_i 을 t_new로 교체;
        t_new에 대해 고장 시뮬레이션을 수행;
      } else {
        ++NumOfFailure;
        FlagAllRemoved = FALSE;
        if (NumOfFailure == LimitOfFailure) break;
      }
    }
    if (FlagAllRemoved == TRUE)
      T_sub에서 t_j를 제거;
  }
}
}
```

그림 5 주고장 축소 알고리즘

만족되어야 하는 필수할당 신호선을 우선 파악하고 이들 간의 충돌은 없는지 사전에 확인함으로써 테스트패턴의 존재 여부를 빠른 시간 내에 알 수 있다.

주고장 축소 알고리즘은 Two-By-One(TBO)[12]과 Essential Fault Pruning(EFP)알고리즘[8]을 개선하였다. 초기 테스트 집합이 주어지면, TBO는 두 테스트패턴을 새로운 한 패턴으로 교체하여 테스트 집합 크기를 줄이려고 한다. 이 과정은 두 패턴의 주 고장 뿐만 아니라 두 패턴에 의해서만 검출되는 고장들을 모두 검출하는 새로운 테스트패턴을 찾는 과정을 통해 이루어진다. 이런 테스트패턴을 찾아내는 것이 불가능하여도, 두 테스트 패턴을 제거하는 것은 가능하기도 하다. 3개의 테스트패턴을 2개의 새로운 패턴으로 교체시키는 Three-By-Two(TBT)[12] 알고리즘에 의해 이루어질 수 있기 때문이다. 그러나 최악의 경우에 TBO알고리즘은 모든 가능한 컴팩션을 위해 $O(V^2)$개의 패턴 쌍을 점검해야 하며, TBT 알고리즘은 $O(V^3)$개의 패턴 쌍을 점검해야 한다. 따라서 이 알고리즘들은 지나치게 계산 비용이 많이 들게 된다.

EFP는 각 테스트패턴에 대한 주 고장들을 하나씩 제거해 나감으로써 테스트 집합 크기를 줄이려는 알고리즘이다. 한 테스트패턴에 대한 주 고장이 모두 제거되면, 이 패턴은 중복패턴이 되어 테스트 집합에서 제거될 수 있다. 단 한 개의 패턴만을 교체하여 주 고장들을 한번에 제거하려는 TBO는 EFP알고리즘의 특별한 경우로 볼 수 있다. EFP는 TBO의 이런 제한을 완화하여 테스트 집합에 있는 하나 이상의 테스트패턴을 교체하여 주 고장을 제거할 수 있도록 함으로써, TBO보다 훨씬 좋은 성능을 갖는다. 최악의 경우에 EFP는 O(VE)개의 고장집합에 대해 테스트패턴을 생성해야 할 수도 있다. 여기서 E는 주 고장 수이고, V는 초기 테스트 집합의 테스트패턴 수이다. 거의 모든 경우에 E는 V보다 크기 때문에, EFP알고리즘은 TBO보다 훨씬 많은 계산 비용이 든다.

테스트 집합을 컴팩션하는 문제는 그림 3에서 봤던 것 처럼 테스트 집합의 주 고장들을 중복 패턴의 수가 최대화 되도록 주어진 테스트패턴들에 재분배 시키는 문제로 볼 수 있다. 따라서 고려해야 할 탐색공간은 주어진 테스트패턴에 주 고장들을 재배분 시킬 수 있는 모든 가능한 경우이다. TBO나 EFP는 탐색공간에 대한 광역적인 시각을 갖지 못하고 주 고장을 한꺼번에 검출하도록 한번에 한 테스트패턴을 제거하는 것에만 집중함으로써 부분적인 그리디(greedy) 탐색을 할 뿐이다. 한 테스트패턴이 중복 패턴이 될 경우에만 해당 주 고장들이 제거되며, 그렇지 않으면 주 고장들이 그대로 남아있게 된다. 따라서 이러한 제약 때문에 TBO나 EFP는 전체 탐색공간 중의 일부분만을 탐색할 뿐이다.

주고장 축소 알고리즘은 탐색공간에 대한 광역적인 시각을 갖는다. 즉, 중복패턴의 수가 최대화 되도록 주어진 테스트패턴에 주 고장을 배분함으로써 TBO와 EFP를 개선하였다. 따라서 어떤 테스트패턴이 중복패턴이 되지 않는다 하여도, 본 알고리즘은 가능한 한 많은 주 고장을 제거하려고 함으로써 주 고장 수를 우선 감소시킨다.

단번에 주 고장을 모두 제거하는 것은 불가능할 수도 있지만, 여러 테스트패턴에 걸쳐서 가능한 조금씩이라도 주고장을 지속적으로 제거해 나감으로써 해당 테스트패턴이 중복패턴이 될 확률을 높인다.

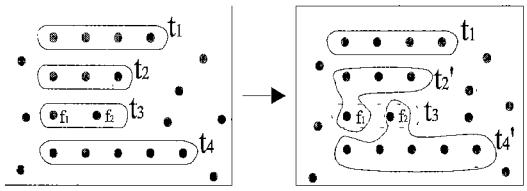

그림 6 주고장 축소 알고리즘의 주고장 제거

그림 6은 주고장 축소 알고리즘에 의해 테스트패턴 t_3가 어떻게 중복패턴이 되어 제거되는지를 보여준다. t_3가 컴팩션을 위해 고려 중인 테스트패턴이라고 가정하자. 테스트패턴 t_1, t_2, t_4중 어느 패턴을 변경시켜도 t_3의 주고장 f_1, f_2를 한번에 검출하는 것이 불가능하다면 기존의 TBO와 EFP는 t_3를 제거할 수 없다. 그러나 주고장 축소 알고리즘은 한번에 제거하는 것이 불가능하면, 몇 번의 반복을 통하여 f_1을 검출하도록 t_2를 t_2'로 변경하고 f_2를 검출하도록 t_4를 t_4'로 변경하여 결국 t_3의 주고장들을 모두 제거할 수 있다. 따라서 t_3는 중복패턴이 되어 제거될 수 있다.

5. 실험결과

본 논문에서 제안된 테스트 컴팩션 알고리즘들은 Virginia Polytechnic Institute & State University의 하동삼 교수 연구실에서 개발한 ATPG시스템인 ATALANTA [13] v2.0에 통합하여 구현되었다. 구현된 테스트 컴팩션 시스템의 전체 흐름도는 그림 7과 같다. 실험을 위한 벤치마크회로에는 ISCAS85[14]와 ISCAS89[15] 완전스캔 버전을 사용하였다.

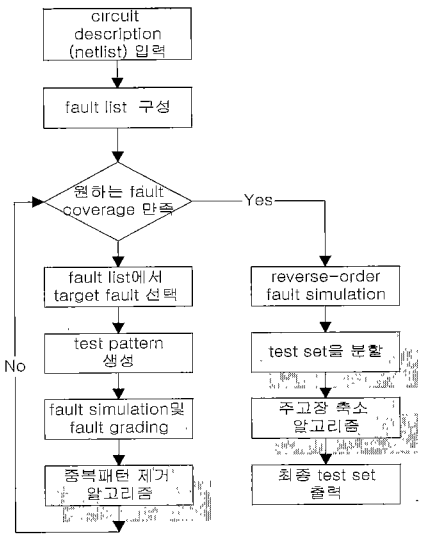

그림 7 본 시스템의 전체 흐름도

본 시스템은 Sun Ultra1/SunOS5.6 (256 MB RAM) 상에서 C언어로 구현되었다. 제안된 테스트 컴팩션 알고리즘은 이전에 발표된 것들 중 가장 좋은 결과를 갖는 테스트 컴팩션 알고리즘인 COMPACTEST[10,12]와 TSC(Test Set Compaction)[8]에 대해 실험결과가 비교된다. 실험 결과는 표 2에 나타내었다. 표 2에서 LB열은 각 회로에 대한 최소 테스트 집합 크기에 대해 현재까지 알려진 가장 큰 Lower Bound를 나타내며, 표 2의 값들 중에서 지금까지 알려진 가장 작은 테스트 집합 크기에는 표시를 하였다. 모든 실험에서 본 시스템은 백트랙 회수 제한을 10으로 하였다. 본 시스템의 CPU 실행시간은 고장 시뮬레이션과 초기 테스트 집합 생성 시간을 포함하고 있다. 본 시스템에 의해 생성된 모든 테스트 집합은 100%의 test efficiency(검출가능한 고장에 대한 고장검출율)를 갖는다. COMPACTEST와 TSC에 대한 성능 결과는 [12]와 [8] 논문에서 제시된 결과값들이다. ISCAS89 벤치마크 회로(완전스캔 버전)에 대한 TSC의 실험결과는 보고되지 않았다.

표 2의 실험결과를 보면, 본 시스템을 통하여 얻어진 모든 테스트 집합은 그 크기가 기존에 발표된 결과와 같거나 더 작다는 사실을 알 수 있다. 제시된 벤치마크 회로 중에서 8개 회로에 대해서는 본 시스템에 의해 생성된 테스트 집합의 크기가 Lower bound와 동일하였다. COMPACTEST, TSC와 본 시스템이 생성한 테스트 집합 크기를 비교해 보기 위해 각 벤치마크 회로에 대한 평균 테스트집합 크기 감소비율을 구해보면, COMPACTEST에 대해서는 약 4.9%, TSC에 대해서는 약 4.4% 정도이다. C880의 경우, COMPACTEST에 비해서 약 19%정도가 감소되었고, S38417의 경우에는 16.5% 정도가 감소되었다.

본 시스템에서 테스트 컴팩션에 소요된 CPU 실행시간은 표 2의 오른쪽 열에 제시되었다. COMPACTEST,

표 2 벤치마크회로에 대한 실험 결과

회 로	테스트 집합 크기				CPU 실행 시간(초)
	LB	COMPACTEST	TSC	본 시스템	본 시스템
C432	27	29	29	27√	5.0
C499	52	52√	53	52√	13.9
C880	13	21	18	17	16.4
C1355	84	84√	86	84√	23.5
C1908	106	106√	106√	106√	63.1
C2670	44	45	44√	44√	58.6
C3540	80	91	90	85	244.0
C5315	37	44	46	41	212.3
C6288	6	14	14	13	52.5
C7552	65	80	76	73	635.8
S444	24	24√	NA	24√	0.7
S526	49	50	NA	49√	2.4
S820	93	94	NA	94	22.2
S1238	121	124	NA	122	55.0
S13207	233	235	NA	233√	942.7
S38417	62	85	NA	71	9418.9

TSC와 본 시스템의 구현환경이 달라서 CPU실행시간의 향상 정도에 대한 객관적인 비교가 어렵기 때문에 표 2에서 COMPACTEST와 TSC의 CPU실행시간에 대한 부분은 생략하였다. 본 시스템은 컴팩션을 위해서 전체 테스트 집합을 둘로 분할하여 처리함으로써 처리속도를 개선하여 더 큰 규모의 회로에 대해서도 비교적 작은 시간 내에 컴팩션을 수행할 수 있었다.

6. 결 론

본 논문에서는 테스트 컴팩션을 위한 새로운 동적인 컴팩션 알고리즘과 정적인 컴팩션 알고리즘을 제안하였다. 새로운 동적인 컴팩션 알고리즘으로서 제안된 중복패턴 제거 알고리즘은 테스트패턴 생성 과정 중에 발생되는 중복패턴을 확인하기 위해 고장 시뮬레이션을 수행하고 이로부터 얻어지는 정보를 이용하여 동적으로 테스트 집합에서 중복패턴을 제거한다. 새로운 정적인 컴팩션 알고리즘인 주고장 축소 알고리즘은 초기 테스트 집합을 두 부분으로 분할하고 테스트패턴당 고장검출율이 낮은 부분에 대해서만 컴팩션이 적용되는데, 각 테스트패턴에 대한 주고장을 하나씩 줄여나가도록 하여 최종적으로 해당 테스트패턴을 중복패턴으로 만들어 제거하는 방법이다.

제안된 테스트 컴팩션 알고리즘들은 그 성능을 확인하기 위해 ATPG시스템에 통합하여 구현되었으며, 벤치마크회로에 대한 성능실험을 통하여 제안된 알고리즘이 기존에 발표된 알고리즘에 비해서 테스트 집합 크기 면에서 그 성능이 우수함을 확인하였다.

참 고 문 헌

[1] J. P. Roth, "Diagnosis of automata failures: A calculus and a method," IBM J. Res. Develop., vol. 10, pp.278-291, July 1966.

[2] P. Goel, "Implicit enumeration algorithm to generate Tests for Combinational Logic Circuits," IEEE Trans. Comput., vol. C-30, pp.215-222, Mar. 1981.

[3] H. Fujiwara and T. Shimono, "On the acceleration of test generation algorithms," IEEE Trans. Comput., vol. C-32, no.12, pp.1137-1144, Dec. 1983.

[4] M. Schulz, E. Trischler, and T. Sarfert, "SOCRATES: A highly efficient automatic test pattern generation system," IEEE Trans. Computer-Aided Design., vol. 7, pp.126-137, Jan. 1988.

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

[6] G. Tromp, "Minimal test sets for combinational circuits," in Proc. IEEE Int. Test Conf., pp.204-209 Oct.1991.

[7] Y. Matsunaga, "MINT-An exact algorithm for finding minimum test sets," IEICE Trans. Fundamentals, vol. E76-A, pp.1652-1658, Oct. 1993.

[8] J. Chang and C. Lin, "Test Set Compaction for Combinational Circuits," IEEE Trans. Comput., vol. 14, no.11, pp.1370-1378, Nov. 1995.

[9] P. Goel and B. C. Rosales, "Test generation & dynamic compaction of tests," in Dig. Papers Test Conf., pp.189-192, Oct. 1979.

[10] I. Pomeranz, L. N. Reddy, and S. M. Reddy, "COMPACTEST: A method to generate compact test sets for combinational circuits," in Proc. IEEE Int. Test Conf., pp.194-203, Oct. 1991.

[11] L. N. Reddy, I. Pomeranz, and S. M. Reddy, "ROTCO: A reverse order test compaction technique," in Proc. 1992 Euro-ASIC Conf., pp.189-194, 1992

[12] S. Kajihara, I. Pomeranz, K. Kinoshita and S. M. Reddy, "Cost-Effective of Minimal Test sets for Stuck-at Faults in Combinational Logic Circuits," IEEE Trans. on Computer-Aided Design, pp.1496-1504, Dec. 1995.

[13] H. K. Lee and D. S. Ha, "On the Generation of Test Patterns for Combinational Circuits," Technical Report No. 12_93, Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University.

[14] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinatorial benchmark circuits and a target translator in Fortran," in Proc. Int. Symp. Circuits Syst., June 1985.

[15] F. Brglez, D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," in Proc. the Int. Symp. on Circuits and Systems, pp.1929-1934, May 1989.

[16] S. B. Akers, C. Joseph, and B. Krishnamurthy, "On the Role of Independent Fault Sets in the Generation of Minimal Test Sets," in Proc. IEEE Int. Test conf., pp.204-209, Oct. 1991.

[17] M. S. Hsiao, E. M. Rudnick, and J. H. Patel, "Fast algorithms for static compaction of sequential circuit test vectors," in Proc. IEEE VLSI Test Symp., pp.188-195, Apr. 1997.

[18] R. Guo, I. Pomeranz, and S. M. Reddy, "Procedures for static compaction of test sequences for synchronous sequential circuits based on vector restoration," in Proc. Design,

Automation, and Test in Europe(DATE) Conf., pp.583-587, Feb. 1998.

김 윤 홍
1986년 한양대학교 전자공학과 학사. 1988년 한양대학교 전자공학과 석사. 1992년 한양대학교 전자공학과 박사. 1993년 ~ 현재 상명대학교 천안캠퍼스 컴퓨터·정보·통신 공학부 컴퓨터시스템공학전공 부교수. 관심분야는 VLSI CAD, Test Generation, Testable Design, Logic Synthesis