조합회로에 대한 게이트 지연 검사 패턴 생성기의 속도 향상에 관한 연구

박 승용, 김 규철
단국대학교 컴퓨터공학과

A Study on Speed Improvement of Gate Delay Test Generator for Combinational Circuits

Seungyong Park, Kyuchull Kim
Dept. of Computer Engineering, Dankook Univ.

Abstract

Fault dropping is a very important part of test generation process. It is used to reduce test generation time. Test generation systems use fault simulation for the purpose of fault dropping by identifying detectable faults with generated test patterns. Two kinds of delay fault model is used in practice, path delay fault model and gate delay fault model. In this paper we propose an efficient method for gate delay test generation which shares second test vector.

1. 서론

검사 패턴 생성(test pattern generation)과 고장 시뮬레이션(fault simulation)을 위한 고장 모델(fault model)에는 여러가지가 있다. 회로 내의 어떤 라인(line)이 입력 값에 무관하게 고정된 값을 가지면 그 라인이 고착고장(stuck-at fault)을 가지고 있다고 한다. 디지털 시스템의 복잡도가 증가하면서 기존의 고착고장을 검출하는 검사만으로는 시스템의 올바른 동작을 보장할 수 없게 되었다. 어떤 회로가 정상 속도의 시스템 클럭에서는 오동작(malfunction)을 보이다가 시스템 클럭의 속도를 낮추었을 때 정상적인 동작을 보이면 그 회로는 지연고장(delay fault)을 가지고 있다고 말한다[1].

조합회로에서 입력 패턴의 공급과 출력들의 검사를 위해 주입력과 주출력들에 래치(Latch)를 연결하였을 때, 지연고장이 존재하지 않음을 보장하기 위해서는 회로내의 주입력에서 주출력까지의 모든 경로들이 실제 동작 클럭내에 신호를 전파시켜야 한다[2]-[4]. 게이트 지연 고장을 경로를 통해 검사할 때, 고장 크기가 경로의 슬랙(slack of path), 즉 시스템 클럭 간격과 고장이 없는 상태에서 경로의 전파지연의 차보다 더 클 경우에 경로내에 지연고장이 존재한다. 따라서 크기가 작은 지연 고장도 검출할 수 있게 하기 위해서는 경로의 슬랙이 작아야 한다. 이러한 이유로 지연고장은 고장 지점을 통과하는 가장 긴 경로를 통해서 검출하려는 연구가 선행되었다[6].

게이트 지연고장 검사를 위와 같이 행할 경우에 검사의 신뢰성을 높일 수 있으나, 검사의 속도는 느리다. 본 논문에서는 고착고장 테스트 생성 기법을 이용한 테스트 생성기와 고장 시뮬레이션을 사용하지 않는 추가 검출고장 제거(fault dropping)방법을 이용하여 검사의 속도를 높이는데 역점을 두었다.

본 논문의 구성은 다음과 같다. 2 장에서는 경로지연 고장과 게이트지연 고장에 대한 내용을 서술한다. 3 장에서는 본 논문의 전반적인 검사 패턴 생성 알고리즘을 소개한다. 4 장에서는 추가 검출 고장 제거에 사용된 알고리즘을 소개하고, 5 장에서는 실험결과 및 결과값을 고찰한다. 6 장에서는 결론을 서술한다.

2. 경로 지연 고장과 게이트 지연 고장

지연 고장 모델에는 게이트 지연 고장 모델[1],[4],[5]과 경로 지연 고장 모델[2],[3]이 있다. 게이트 지연 고장들은 회로 내 각 게이트의 입력단이나 출력단에 논리값의 완상승(slow-to-rise) 고장이나 완하강(slow-to-fall) 고장으로 나타난다. 경로 지연 고장이란 지정된 경로를 통한 신호의 전파가 일정한 시간 내에 이루어지지 않는 고장을 말한다.

경로 지연 고장 검사 기법은 회로 내의 모든 경로들이 다 검사되었을 경우에 회로 자체에 지연 고장이 없다고 신뢰할 수 있다는 장점이 있다. 그러나 회로 내 경로의 수는 회로 내 게이트 수에 지수적으로 증가하기 때문에 모든 경로를 검사한다는 것은 불가능하다.

어떤 경우의 지연 고장들은 다른 지연 고장의 영향으로 경로지연 고장검사로는 발견되지 않을 수도 있다. 예를 들면, 그림 1 에서 경로 b-f-g-i-k 가 주입력단 b 의 신호에 대한 하강 천이(falling transition)를 검사하는 경로라고 하자. 경로 지연 고장 모델에서 b 의 신호 하강 천이를 검사 경로를 통해 전파하기 위해서는 a 와 c 는 초기값과 최종값 모두 논리값 1 을 가져야 한다. 이렇게 a 와 c 에 할당된 논리값은 노드 j 에 하강 천이

를 발생시킨다. 이때 노드 j의 천이가 노드 i의 하강 천이가 노드 k로 전파되는 것을 방해할 수 있다. 즉 경로 b-f-g-i-k 는 경로 지연 고장 모델에서는 완전히 (robustly) 검사될 수 없다. 게이트 지연 고장 모델에서는 노드 b의 완하강 고장을 b-f-g-i-k 를 통해서 검출해 낼 수 있다.

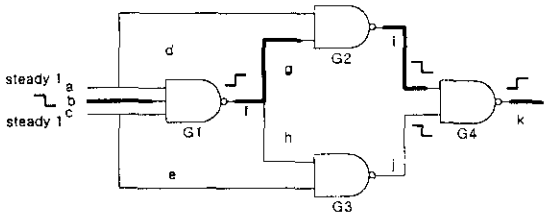

그림 1. 경로 지연 고장 모델의 한계

경로 지연 검사법은 많은 해결해야 할 어려운 부분이 있다. 본 논문에서 제안한 지연 고장 검사 패턴 생성 기법은 게이트 지연 고장 모델을 사용하며, 기존의 고착 고장 테스트 생성기법을 이용하였다. 다음 절에서는 지연 고장 검사 패턴 생성 알고리즘을 설명한다.

3. 게이트 지연 고장 검출 패턴 생성 알고리즘

게이트 지연고장을 검출하는 패턴은 서로 다른 두개의 주입력 벡터로 구성된다. 본 논문에서는 기존의 고착고장을 검출하는 알고리즘 중 PODEM 알고리즘이 사용한 5-value를 사용한다[7]. 5-value 는 1, 0, *X*, *D*, $\overline{D}$ 로 구성되는데, 1 은 논리값 1 을, 0 은 논리값 0 을 나타내고 *X* 는 미지값(unknown value)을 나타낸다. *D* 는 정상 회로에서는 논리값 1 을 가지고, 고장 회로에서는 논리값 0 을 갖는 논리값이다. $\overline{D}$ 는 정상 회로에서는 논리값 0 을 가지고, 고장 회로에서는 논리값 1 을 갖는 논리값이다. 게이트 지연 고장의 테스트를 생성하려면 먼저 고장 지점을 천이 이전의 값으로 초기화하는 패턴 T_1을 생성하고, 천이를 유발시키고, 고장의 영향을 주출력으로 전파시키는 T_2를 생성한다.

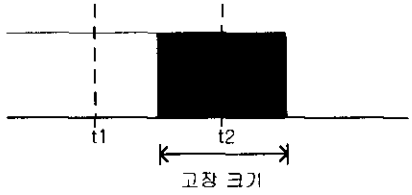

그림 2. 노드 b의 논리값 변화

	첫번째 값	두번째 값
지연고장이 없을 때	1	0
지연고장이 있을 때	1	1
고착고장으로의 모델링	1	$\overline{D}$

그림 3. 노드 b의 논리값 변화표

생성된 T1 과 T2 가 고장 지점의 게이트 지연 고장을 검사하는 패턴이다.

그림 1 에서 노드 b에 완하강 고장을 검출하는 패턴을 생성해 보자. 검사 패턴을 검출하려면 노드 b에 논리값 1 을 할당한 뒤, 0 으로 변화 시킨 후 그 고장의 영향이 출력 k에 나타나는지를 확인하면 된다. 이를 위해 먼저 노드 b를 논리값 1 로 초기화한다. 다음으로 노드 b를 논리값 0 으로 변화시켜 값의 천이(transition)를 일으킨다. 여기에서 지연 고장은 활성화 되었다. 이제 지연고장의 영향을 f 로 전파시키기 위해 노드 a와 노드 b에 비제어값(non-controlling value)인 1 을 할당한다. f에 전파된 고장의 영향은 다시 g와 h를 거쳐 i와 j로 전파되며, 결국 주출력 k에 나타난다. 즉 두개의 입력 패턴 (X, 1, X), (1, 0, 1)을 연속적으로 가할 때, 지연 고장이 없다면 주출력 f의 값은 1 이 되고, b에 완하강 고장이 있을 경우에 출력값은 0 이 된다. 그림 2 는 노드 b의 논리값 변화를 설명한다. 시간 t1 에서 b의 논리값은 1 로 초기화 되어있다. 시간 t2 에서 고장이 없는 회로에서는 논리값이 0 으로 천이 되어야 하는데, b의 완하강 지연 고장으로 인해 논리값이 변함없이 1 로 고착되어 있다. 위에서 얻어낸 검사 패턴을 다시 고찰해 보자. 첫번째 패턴 (X, 1, X)는 고장 지점 b의 논리값을 초기화 시키는 패턴이다. 두번째 패턴 (1, 0, 1)은 b의 지연고장을 활성화시키고 주출력으로 전파하는 패턴이다. 또 패턴 (1, 0, 1)은 b의 1 고착 고장을 생성하는 패턴임을 역시 알 수 있다. 그러므로 게이트 지연 고장 검사 패턴은 고장 지점을 초기화하는 패턴과 고장 지점의 초기화된 값에 대한 고착 고장을 검출하는 패턴의 쌍으로 이루어 진다. 그림 3 은 b의 완하강 고장 시 논리값을 고착고장 모델에서의 논리값으로 변환을 나타낸다.

```
01. while(fault list is not empty)
02. {
03.     Get a fault from undetected fault list.
04.     GenT1(fault);
05.     if(T1 generated)
06.     {
07.         GenT2(fault);
08.         if(T2 generated)
09.         {
10.             Gen_another_Test();
11.             test has successfully generated.
12.         }
13.         else
14.             test generation failed.
15.     }
16.     else
17.         test generation failed.
18. }
```

그림 4. 검사 패턴 검출 알고리즘

그림 4 는 본문이 제안하는 테스트를 생성하는 알고리즘이다. 네번째 줄은 첫번째 검사 패턴인 초기화를 위한 패턴을 생성하는 함수를 호출하고, 성공적으로 첫번째 패턴이 생성되면, 일곱번째 줄에서 고착 고장 검사 패턴, 즉 두번째 검사 패턴을 생성한다 이 알고리즘으로 PODEM 을 사용하였다. 성공적으로 수행되

면, 열번째 줄에서 추가 검사 패턴 생성 알고리즘을 수행한다. 추가 검사 패턴 생성 알고리즘은 4 장에서 자세히 논의 하기로 한다.

보다 빠른 검사 패턴 생성을 위해서 고장이 활성화되고 고장의 영향이 전파되는 회로의 경로는 짧을수록 좋다. 따라서 본 논문이 제안하는 알고리즘은 가장 짧은 경로를 통해 고장을 활성화시킨다. 하지만 고장을 주출력으로 전파시킬 때는 무조건 짧은 경로가 좋지는 않다. 짧은 경로를 택한 경우에는 검사 패턴 생성이 쉬운 장점이 있지만, 추가적으로 검출할 수 있는 고장의 수는 줄어들 것이다. 반대로 긴 경로를 택한 경우에는 검사 패턴 생성은 힘들지만 더 많은 고장들을 발견하게 될 것이다.

고장지점으로부터 주입력까지 이어지는 가장 짧은 경로와 고장지점으로부터 주출력까지 이어지는 가장 긴 경로를 찾기 위해서 모든 게이트들을 두번 레벨화(levelize) 하였다. 먼저 주출력에서부터 주입력으로의 깊이 우선 탐색(depth first search) 방법을 사용하여 각 게이트들의 레벨을 정한다. 이어 주입력에서부터 주출력으로의 깊이 우선 탐색 방법을 사용하여 각 게이트들의 역레벨을 정한다. 고장 게이트로부터 주입력에 이를 때까지 각 게이트의 팬인(fanin) 게이트들 중에서 레벨 값이 가장 작은 게이트를 선택하면 가능한 가장 짧은 활성화 경로를 구할 수 있다. 또 전파 경로를 구할 때는 고장 게이트로부터 주출력에 이를 때까지 각 게이트의 팬아웃(fanout) 게이트들 중에서 역레벨 값이 가장 큰 게이트를 선택해 나가면 가장 긴 전파 경로를 구할 수 있다.

두가지 방법으로 검사 패턴을 생성할 때 그 결과와 효율을 5 장에서 살펴보기로 한다.

4. Fault Dropping 알고리즘.

고착고장 테스트를 생성할 때는 흔히 고장 시뮬레이션을 통하여 추가 검출 고장 제거(fault dropping)를 수행한다. 동시적 고장 시뮬레이션[8]이나 HySim[9], PROOFS[10]와 같은 빠른 고착 고장 시뮬레이션을 위한 방법들이 많이 연구되어 왔으나 이러한 방법들은 지연고장 시뮬레이션에는 적합하지 않다. 고착고장에서의 고장 시뮬레이션을 행하는 데는 0, 1, X 세개의 값만 필요하다. 지연고장의 테스트는 두개의 입력 패턴으로 구성되므로 0, 1, 그리고 미지의 값(unknown value)의 조합인 9 개의 값이 필요하다. 이러한 이유로 앞에서 거론된 알고리즘들은 게이트 지연 고장 모델에서 검사패턴을 생성할 때 추가 검출 고장을 제거하는데 사용하기에는 적합하지 않다.

그림 5 는 그림 4 의 10 번째 줄에 나와 있는 추가 검사 패턴 알고리즘인 Gen_another_Test()를 나타낸다.

```
01. Make Sensitized Path(D propagation path).
02. While(fault list in sensitized path is not empty)
03. {
04.     drop_fault_using_fault_simulation()
05.     GenT1(faults in );
06. }
```

그림 5. 추가 검사 패턴 생성 알고리즘

알고리즘의 첫번째 줄은 고착 고장 테스트 생성으로 만들어진 두번째 패턴에 의한 각 게이트의 값들을 고찰하여 고장의 영향이 전파된 경로를 찾는 과정이다. 이어 네번째 줄에서 고장 시뮬레이션에 의한 추가 검출 고장 제거를 행하며, 다섯번째 줄에서 활성화 경로 내의 제거되지 않은 고장에 대해 첫번째 초기화 패턴만을 제거하는 알고리즘을 수행한다. 초기화 패턴이 만들어 지면 경로내 고장 리스트가 빌 때 까지 네번째 줄과 다섯번째 줄을 반복한다.

앞의 그림 1 에서 b 에 1 고착 고장이 있다고 가정하자. 패턴 (1, 0, 1)은 경로 b-f-g-i-k 와 b-f-h-j-k 를 통해 b 에 존재하는 1 고착 고장을 검사할 수 있다. 이 경우 각 노드의 논리값을 살펴보면 b, i, j 는 논리값 D 를 가지고 f, g, h, k 는 논리값 $\overline{D}$ 를 가진다. 즉, 주어진 패턴은 노드 b, i, j 에 존재하는 1 고착고장을 검출할 수 있고, 노드 f, g, h, k 에 존재하는 0 고착고장을 검출할 수 있다. 이러한 결과는 패턴 (1, 0, 1)을 그림 1 의 회로에 대해 고장 시뮬레이션을 행하여 얻을 수 있다. 결과적으로 노드 b 에 존재하는 1 고착고장을 검출하여 시간을 많이 소모하는 검사 패턴 생성을 하지 않고도 추가로 6 개의 고장을 검사할 수 있음을 알게 되었다. 지연 고장의 경우, b 의 완하강 고장을 검사하려면, (X, 1, X), (1, 0, 1) 패턴이 필요하다. 하지만 이 경우 고장 시뮬레이션으로는 추가로 f, g, h 의 완상승 고장만이 발견된다. 결국 지연 고장 모델하에서는 고장 시뮬레이션이 고착 고장 모델하에서의 고장 시뮬레이션보다 효율이 좋지 않다. 본 논문에서는 고착 고장 검사 생성기를 사용하여 지연 고장 검사 패턴을 생성한다. 이 경우 두번째 벡터는 고장의 전파경로 상에 있는 게이트들이 공유하게 된다. 즉 두번째 패턴으로 노드가 D 값을 가지게 되면, 0 으로 초기화하는 첫번째 패턴만 찾아내면 노드에 존재하는 완상승 고장을 검출할 수 있다. 반대로 노드가 $\overline{D}$ 값을 가지면, 노드를 1 로 초기화하는 첫번째 패턴만 찾아내면 노드에 존재하는 완하강 고장을 검출할 수 있다. 초기화 패턴을 생성하는 작업은 고착 고장을 검출하는 패턴을 만드는 작업보다 훨씬 시간이 적게 걸린다. 따라서 빠른 시간 안에 지연 고장 검사 패턴을 생성해 낼 수 있다.

5. 실험 및 결과

이번 절에서는 4 가지 지연 고장 검사 패턴 생성 알고리즘을 표를 통해서 서로 비교하기로 한다. 알고리즘들은 C++로 구현되었으며, 프로그램은 64 M 메모리를 가진 Pentium 100 PC 에서 실행되었다. 먼저 TGen1 은 추가 검출 고장 제거 과정에서 고장 시뮬레이션 만을 사용한 결과이다. TGen2 는 추가 검출 고장 제거 과정에서 고장 시뮬레이션을 행하지 않고, 전파경로 내의 패턴들에 대해 초기화 패턴만을 생성한 알고리즘이며, 짧은 경로를 택하여 고장의 영향을 전파하였다. TGen3 는 알고리즘 TGen2 와 같지만, 긴 경로를 택하여 고장의 영향을 전파한 점만 다르다. TGen4 는 TGen1 과 TGen3 의 방법을 절충하였다. 생성된 테스트를 고장 시뮬레이션으로 추가 검출 고장을 제거하고, 전파 경로 내에 제거 되지 않고 남아있는 고장에 대해

서만 초기화 패턴을 생성한 방법이다.

ckt name	# of faults	# of aborts	# of redund.	# of dropped	Time (sec.)	fault coverage
C432	864	11	1	480	37.5	98.7
C499	998	0	8	712	24.3	100
C880	942	0	0	530	17.1	100
C1355	1574	4	2	1066	78.2	99.7
C1908	1879	13	0	1216	57.6	99.3
C2670	2747	135	25	1457	223.0	95.0
C3540	3428	162	82	1906	393.0	95.2
C5315	5350	7	9	3434	150.9	99.9
C7552	7750	273	65	4594	811.6	96.5

표 1. TGen1 의 결과

ckt name	# of faults	# of aborts	# of redund.	# of dropped	time (sec.)	fault coverage
C432	864	8	1	512	36.3	99.1
C499	998	0	8	728	22.1	100
C880	942	0	0	687	2.8	100
C1355	1574	4	2	1213	29.5	99.7
C1908	1879	11	0	1486	22.6	99.4
C2670	2747	116	29	1805	142.5	95.7
C3540	3428	172	87	2358	288.4	94.9
C5315	5350	10	7	4142	77.9	99.8
C7552	7750	226	74	5701	593.9	97.1

표 2. TGen2 의 결과

ckt name	# of faults	# of aborts	# of redund	# of dropped	time (sec.)	fault coverage
C432	864	9	1	504	37.8	99.0
C499	998	0	8	732	24.5	100
C880	942	0	0	682	4.0	100
C1355	1574	0	2	1284	25.8	100
C1908	1879	11	0	1493	24.0	99.4
C2670	2747	129	25	1783	161.4	95.3
C3540	3428	165	82	2362	283.2	95.1
C5315	5350	5	7	4143	76.4	99.9
C7552	7750	176	65	5734	545	97.7

표 3. TGen3 의 결과

ckt name	# of faults	# of aborts	# of redund.	# of dropped	Time (sec.)	fault coverage
C432	864	10	1	476	33.2	98.8
C499	998	0	8	687	19.7	100
C880	942	0	0	682	3.2	100
C1355	1574	0	2	1284	18.4	100
C1908	1879	11	0	1493	24.0	99.4
C2670	2747	130	26	1776	133.4	95.2
C3540	3428	163	82	2362	235.9	95.1
C5315	5350	5	7	4143	69.8	99.9
C7552	7750	176	65	5721	453.6	97.7

표 4. TGen4 의 결과

TGen1 의 성능이 가장 좋지 않다. 게이트 지연 검사 패턴에 고장 시뮬레이션을 이용한 추가 고장 검출 개수가 TGen2 나 TGen3 의 방법에 의한 개수 보다 상대적으로 적다. 따라서 생성해야 할 검사패턴의 수가 더 많아지기 때문이다. TGen2 와 TGen3 의 비교에서 TGen2 는 고장에 대한 검사 패턴을 빨리 생성할 수 있으나, 한 패턴에 대해 추가 검출되는 고장의 수는 작고, TGen3 는 검사 패턴 생성은 느리지만 상대적으로 추가 검출 고자의 수가 많다. 결과적으로 걸린 시간은 거의 비슷하지만 큰 회로일수록 TGen3 가 속도면에서 약간 우수함을 볼 수 있다. TGen4 는 거의 모든 회로에 대해 가장 우수한 성능을 보인다.

6. 결론

실험결과 고장 시뮬레이션을 이용한 추가 검출 고장 제거법은 속도는 빠르지만 많은 고장을 추가로 제거하지 못한다. 본 논문에서 제안된 두번째 검사 패턴을 공유하는 방법은 전파 경로상의 게이트에 존재하는 지연 고장의 검사 패턴을 빠르게 생성한다. 따라서 기존의 추가 검출 고장 제거에 사용된 고장 시뮬레이션을 행한 후, 전파 경로상의 제거되지 않는 지연 고장에 대해 초기화 패턴만을 생성해 내는 본문의 알고리즘을 적용하면 검사 패턴 생성 시간을 단축할 수 있다.

7. 참고 문헌

[1] E. P. Hsieh, R. A. Rasmussen, L. J. Vidunas, and W.T. Davis, "Delay Test Generation", Proc. 14th Design Automation Conf., pp.486-491, June 1977.

[2] G. L. Smith, " Model for Delay Faults Based upon Path", Proc. Int. Test Conf., pp.342-349, 1985.

[3] C. J. Lin and S. M. Reddy, " On Delay Fault Testing in Logic Circuits", IEEE Trans. On CAD, pp. 694-703, September 1987.

[4] E. S. Park and M. R. Mercer, " Robust and Nonrobust Tests for Path Delay Faults in a Combinational Circuit", Proc. Int. Test Conf., pp. 1027-1034, 1987.

[5] Chin Jen Lin and Sudhakar M. Reddy, " On Delay Fault Testing in Logic Circuits", IEEE Transactions on computer-aided design, Vol. CAD-6, N0. 5, pp. 694-703, September 1987

[6] E. S. Park and M. R. Mercer, " An Efficient Delay Test Generation System for Combinational Logic Circuits", 27th ACM/IEEE Design Automation Conf., pp.486-491, June 1990.

[7] P. Goel, " An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits", IEEE Trans. On Comput., Vol.30, No3, pp. 215-222, 1981.

[8] E. G. Ulrich and T. Baker, "The Concurrent Simulation of Nearly Identical Digital Networks", Design Automation Workshop, Vol. 6, pp. 145-150, April 1973.

[9] K. Kim and K. K. Saluja, "HYSIM: Hybrid Fault simulation for Synchronous Sequential Ciruits", VLSI Design, An International Journal of Custom-Chip Design, Simulation and Testing, Vol. 4, No. 3, pp. 181-197, July 1996.

[10]W. T. Cheng and J. H. Patel, "PROOFS: A Super Fast Fault Simulator for Sequential Circuits", Journal of Electronic Testing: Theory and Applications, pp. 7-13, 1990.