論文98-35C-9-5

# 기본 모드에서 동작하는 비동기 순차 회로의 시험 벡터 생성

# (Test Pattern Generation for Asynchronous Sequential Circuits Operating in Fundamental Mode)

趙 庚 衍 *, 李 宰 勳 **, 閔 炳 福 **

(Kyoung-Youn Cho, Jae Hoon Lee, and Hyoung Bok Min)

요 약

비동기 순차 회로에 대한 시험 벡터를 생성하는 문제는 매우 어려운 문제로 남아 있다. 현재까지 이 문제에 대한 알고리즘은 거의 없었다. 그리고, 기존의 접근 방식은 시험 벡터를 생성하는 동안에는 피이드백 루프를 절단하여 그 곳에 플립플롭이 있는 것처럼 가정하고 시험 벡터를 생성하는 방식이었다. 그래서, 기존의 알고리즘은 동기 순차 회로용 시험 벡터 생성 알고리즘과 매우 유사하였다. 이것은 시험 벡터를 생성할 때에는 비동기 순차회로를 동기 순차 회로로 가정하고 시험 벡터를 생성한다는 것을 의미한다. 그러므로, 생성된 시험 벡터가 비동기 순차 회로에 적용되었을 때, 대상 결함을 검출하지 못할 수도 있다는 것을 나타낸다. 본 논문에서는 비동기 순차 회로에 대한 시험 벡터를 생성할 수 있는 알고리즘을 제시하였다. 본 논문에서 제안된 알고리즘을 적용하여 생성된 시험 벡터는 임계레이스(critical race) 문제와 순환(oscillation) 문제의 발생을 최소로 하면서 비동기 순차 회로의 결함을 검출할 수 있다. 그리고, 본 논문에서 제안된 알고리즘을 적용하여 생성된 시험 벡터는 비동기 순차 회로에 대해서 대상 결함을 검출하는 것이 보장된다.

**Abstract**

Generating test patterns for asynchronous sequential circuits remains to be a very difficult problem. There are few algorithms for this problem, and previous works cut feedback loops, and insert synchronous flip-flops in the feedback loops during ATPG. The conventional algorithms are similar to the algorithms for synchronous sequential circuits. This means that the conventional algorithms generate test patterns by modeling asynchronous sequential circuits as synchronous sequential circuits. So, test patterns generated by those algorithms may not detect target faults when the test patterns are applied to the asynchronous sequential circuit under test. In this paper an algorithm is presented to generate test patterns for asynchronous sequential circuits. Test patterns generated by the algorithm can detect target faults for asynchronous sequential circuits with the minimal possibility of critical race problem and oscillation. And it is guaranteed that the test patterns generated by the algorithm will detect target faults.

## I. 서 론

현재까지 조합 회로에 대한 시험 벡터 생성에 관한 연구와 동기 순차 회로에 대한 시험 벡터 생성에 관한 연구는 많이 이루어져 왔다. 그러나, 비동기 순차 회로에 대한 시험 벡터 생성에 관한 연구는 많이 이루어지지 않았다.

비동기 순차 회로는 입력이 변화할 때 클럭 펄스를 기다리지 않고 즉시 응답을 해야만 하는 시스템, 즉, 동작 속도가 중요한 시스템에서 사용된다. 또한, 서로

* 正會員, 電子部品綜合技術硏究所 시스템IC硏究센터
(Korea Electronics Technology Institute)
** 正會員, 成均館大學校 電氣工學科
(School of Electrical and Computer Eng., Sung Kyunkwan Univ.)

※ 본 연구는 한국 학술 진흥 재단의 1997년도 대학 부설 연구소 지원 과제로 연구되었음
接受日字:1998年4月20日, 수정완료일:1998年8月19日

독립적인 내부 클럭을 가지고 있는 서로 다른 두 시스템 사이의 통신은 비동기 순차 회로로 구현되어야만 한다[1]. 이와 같이, 현재 설계되는 대부분의 시스템이 동기적인 요소 뿐만 아니라 비동기 적인 요소도 포함하고 있다는 점을 고려할 때, 비동기 순차 회로에 대한 시험 벡터 생성에 관한 연구는 매우 중요한 의미를 갖는다고 할 수 있다.

비동기 순차 회로에 대해서 시험 벡터를 생성하는 문제에 대해서는 그 동안 많은 연구가 이루어지지 않았고, 현재까지 발표된 대부분의 비동기 순차 회로용 시험 벡터 생성 알고리즘들은 시험 벡터를 생성하는 동안에는 피이드백 루프를 절단하여 그 곳에 플립플롭이 있는 것처럼 가정하고 시험 벡터를 생성하였다[2]. 그러나, 단순한 피이드백 루프와 플립플롭은 그 구조와 기능이 다르기 때문에, 기존에 발표된 알고리즘을 적용했을 경우에는 시험 벡터를 생성하였다 하더라도, 생성된 시험 벡터가 대상으로 하는 결함을 검출하지 못할 수가 있다. 이에 대한 예는 4장에 제시되었다. 본 논문에서는 이러한 기존 연구의 문제점을 해결하여, 비동기 순차 회로에 대해서 시험 벡터를 생성할 수 있는 알고리즘을 제안하였고, 그 알고리즘으로부터 비동기 순차 회로용 시험 벡터 생성기를 구현하였다. 시험 벡터 생성에 관한 기본적인 개념들은 PODEM[3], FAN[4], SOCRATES[5], ESSENTIAL[6]을 참조하였다.

## Ⅱ. 비동기 순차 회로의 구조 및 동작 원리

비동기 순차 회로에 대한 시험 벡터 생성 알고리즘을 설명하기 위해, 비동기 순차 회로의 동작 특성에 대해서 간단히 설명한다.

### 1. 기본 모드에서 동작하는 비동기 순차 회로

비동기 순차 회로가 기본 모드에서 동작한다는 것은 다음과 같은 조건을 가지고 동작한다는 것을 나타낸다.

1) 연속된 입력 벡터의 Hamming distance는 1이다.

2) 피이드백 루프에는 시간 지연 소자가 있다고 가정한다.

3) 주 입력단의 논리값은 회로가 안정된 상태에서만 변한다.

### 2. 비동기 순차 회로의 구조

그림 1은 비동기 순차 회로의 구조를 나타낸다. 이와 같은 비동기 순차 회로에서 피이드백 루프에는 시간 지연 소자가 존재한다고 가정한다. 그리고, 시간 지연 소자의 출력단을 PPI(Pseudo Primary Input)로 정의하고, 입력단을 PPO(Pseudo Primary Output)로 정의한다[6]. 그림 1에 주어진 고착 결함(stuck-at-fault)을 검출하기 위해서는 백트레이싱을 통해서 주 입력단 또는 PPI에 논리값을 할당하여, 결함 신호가 주 출력단에서 관찰되도록 해야한다. 이 때, 주 입력단에 할당된 논리값들은 시험 벡터로 정의하고, PPI에 할당된 논리값들은 내부 상태로 정의한다. 시험 벡터는 외부에서 인가해 줄 수 있는 논리값이고, 내부 상태에 할당된 논리값들은 한 타임 프레임 전의 주 입력단과 PPI에 논리값을 할당함으로써 만족시켜야 할 논리값들이다. 비동기 순차 회로에서 타임 프레임은 피이드백 루프에 존재한다고 가정한 시간 지연 소자를 기준으로 구분한다.

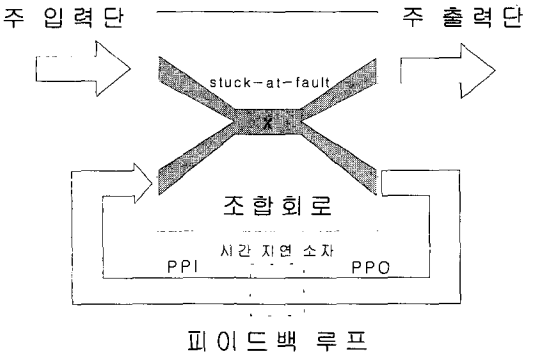


그림 1. 비동기 순차 회로의 구조
Fig. 1. Structure of asynchronous sequential circuits.

### 3. 결함의 분류

본 논문에서는 대상 결함을 5개의 범주로 분류하였다.

1) Testable 결함

Testable 결함은 시험 벡터가 생성된 결함을 말한다.

2) Redundant 결함

Redundant 결함이란 회로 자체의 구조상 결함 신호를 생성할 수 없거나, 결함 신호를 생성하더라도 결함 신호를 주 출력단까지 전파시킬 수 없는 결함이다. 즉, 회로 자체의 redundancy때문에 시험 벡터를 생성할 수 없는 결함을 말한다.

3) Aborted 결함

Aborted 결함이란 시험 벡터를 생성하는 중에 백트

랙킹 횟수가 주어진 횟수 이상이 되어 시험 벡터 생성기가 시험 벡터를 생성하는 것을 포기한 결함을 말한다.

4) Untestable 결함

Untestable 결함이란 결함 신호는 주 출력단으로 전파되었으나 내부 상태를 만족시킬 수 없는 결함을 말한다. 즉, 회로의 내부 상태 이동 문제 때문에 시험 벡터를 생성하지 못한 결함을 말한다. 이 결함은 시험 벡터를 생성할 수 없다는 것이 증명된 결함이다.

5) Uncheckable 결함

Uncheckable 결함이란 본 논문에서 제안된 동일 상태 방문 횟수 제한 때문에 시험 벡터를 생성하지 못한 결함을 말한다. 그러나, 이 결함은 다른 알고리즘을 적용하거나 동일 상태 방문 횟수 제한을 크게 하면 시험 벡터를 생성할 수도 있는 결함이다. 즉, 현재의 조건에서는 시험 벡터가 존재하는지 그렇지 않은지를 판단할 수 없는 결함을 말한다.

이것을 표로 정리하면 표 1과 같다.

표 1. 결함의 분류
Table 1. Classification of faults.

| 구분 \ 결함 | Testable 결함 | Redundant 결함 | Aborted 결함 | Untestable 결함 | Uncheckable 결함 |
|---|---|---|---|---|---|
| 시험 벡터 | 존재함 | 존재하지 않음 | 알 수 없음 | 존재하지 않음 | 알 수 없음 |
| decision tree | 존재함 | 존재하지 않음 | 존재함 | 존재하지 않음 | 존재하지 않음 |
| 판단 기준 | | 회로의 redundancy | 백트랙킹 횟수 | 상태 이동 | 동일 상태 방문 횟수 |

## Ⅲ. 시험 벡터 생성 알고리즘

비동기 순차 회로의 상태 이동에 관한 기본 개념과, 비동기 순차 회로에 대한 시험 벡터를 생성하는 알고리즘을 제시한다.

1. 상태 함수 정의

현재의 타임 프레임의 내부 상태는 전 타임 프레임의 입력 벡터와 전 타임 프레임의 내부 상태의 함수이다.

$$S^t = F( V^{t-1} , S^{t-1} ) \quad (1)$$

$S^t$ : 현재 타임 프레임의 내부 상태

$V^{t-1}$ : 전 타임 프레임의 입력 벡터

$S^{t-1}$ : 전 타임 프레임의 내부 상태

즉, $V^{t-1}$과 $S^{t-1}$이 주어지면 $S^t$는 유일하게 결정된다. 그러나, 함수 $S^t$가 주어졌을 경우, 그 상태를 만족하는 전 타임 프레임의 입력 벡터와 내부 상태의 조합은 여러 개 존재할 수 있다. 비동기 순차 회로에 대한 시험 벡터를 찾는 과정의 핵심은 주어진 내부 상태를 만족하는 전 타임 프레임의 내부 상태와 입력 벡터의 조합을 찾는 것이다.

2. 비동기 순차 회로의 시험 벡터 생성 원리

비동기 순차 회로에 대한 시험 벡터 생성 원리를 [ 정리 1 ] 에 나타내었다.

**[정리 1]** 비동기 순차 회로의 시험 벡터 생성

주 출력단으로 결함 신호를 전파하기 위한 입력 벡터 $V^0$과 내부 상태 $S^0$을 찾는다. 이제, 내부 상태 $S^0$을 만족시키기 위해

$$S^0 = F( V^{-1} , S^{-1} ) \quad (2)$$

을 만족하는 $V^{-1}$과 $S^{-1}$을 찾는다. 다시, 내부 상태 $S^{-1}$을 만족시키기 위해

$$S^{-1} = F( V^{-2} , S^{-2} ) \quad (3)$$

를 만족하는 $V^{-2}$와 $S^{-2}$를 찾는다. 이 과정을 내부 상태가 처음으로 ( $X$, $X$, $X$,…, $X$ )가 될 때까지 반복한다. 만일 $n$번째 내부 상태가 처음으로 ( $X$, $X$, $X$,…, $X$ )가 되었다면,

$$S^{-(n-1)} = F( V^{-n} , S^{-n} ) \quad (4)$$

이고, 여기서, $S^{-n}$ = ( $X$, $X$, $X$,…, $X$ ) 이다.

이 때, 시험 벡터는 $V^{-n}$, $V^{-(n-1)}$, …, $V^{-1}$, $V^0$ 이 된다.

만일, 타임 프레임의 제한 조건 내에서 내부 상태 ( $X$, $X$, $X$,…, $X$ )에 도달하지 못하면 시험 벡터를 생성할 수 없게된다.

**[증명]**

[ 정리 1 ] 의 과정을 역순으로 고찰한다. 먼저 주어진 비동기 순차 회로의 주 입력단에 $V^{-n}$의 시험 벡터를 인가하면 현재의 내부 상태에 관계없이 다음 내부 상태는 $S^{-(n-1)}$이 된다. 이제, 현재의 내부 상태는

$S^{-(n-1)}$이 되었고, 이 상태에서 주 입력단에 $V^{-(n-1)}$의 시험 벡터를 인가하면 다음 내부 상태는 $S^{-(n-2)}$가 된다. 이 과정을 계속해서 반복하여 현재의 내부 상태가 $S^0$이 되었다고 하자. 이 상태에서 주 입력단에 $V^0$을 인가하면 결함 신호가 주 출력단으로 전파되고, 목표로 하는 결함은 검출된다. 그러므로, 시험 벡터는 $V^{-n}$, $V^{-(n-1)}$, ⋯, $V^{-1}$, $V^0$이 된다. ■

[정리 1]의 과정을 상태 다이어그램으로 나타내면 그림 2와 같다.

그림 2는 시험 벡터에 따른 비동기 순차 회로의 내부 상태의 이동을 나타낸다. 내부 상태 중 F는 결함 신호가 주 출력단으로 전파된 상태를 나타낸다. 그리고, 점선으로 표현된 경로는 주어진 내부 상태를 만족시키는 다른 입력 벡터와 내부 상태의 조합이 존재할 수 있음을 나타내는데, 시험 벡터 생성기는 실선으로 표시된 역 경로를 따라서 상태 이동을 하면서 시험 벡터를 생성했다는 것을 나타낸다.

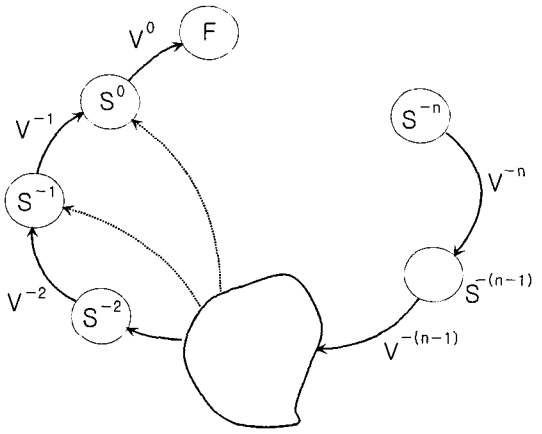


그림 2. 시험 벡터 생성 과정을 설명하는 상태 다이어그램

Fig. 2. State diagram explaining test pattern generation.

3. 최종 상태

[정리 1]에서 내부 상태가 ( $X$, $X$, $X$,⋯, $X$ )가 될 때까지 내부 상태를 만족시켰는데, 그 이유는 현재 할당되어 있는 비동기 순차 회로의 내부 상태에 관계없이 주 입력단에 할당되는 시험 벡터만으로 비동기 순차 회로를 원하는 상태로 이동 시키기 위해서이다. 이것을 [정리 2]에서 제시하였다.

**[정리 2]** 첫번째 적용될 시험 벡터

첫번째 적용될 시험 벡터 $V$는 한 내부 상태 $S$에 대하여 식(5)를 만족해야 한다.

$$S = F(V, (X, X, X, \cdots, X)) \quad (5)$$

여기서, 내부 상태 ( $X$, $X$, $X$,⋯, $X$ )를 최종 상태라고 정의한다.

**[증명]**

비동기 순차 회로가 turn-on되었을 때, 비동기 순차 회로의 내부 상태 변수에 할당될 논리값은 알 수 없고, 또한, 그 논리값들이 항상 일정한 논리값이라고 보장할 수도 없다. 그러므로, 내부 상태와 관계없이 주 입력단에 할당된 논리값으로만 내부 상태를 결정할 수 있어야 한다. 식 (5)에서 내부 상태 ( $X$, $X$, $X$,⋯, $X$ )는 회로의 내부 상태가 어떠한 상태에 있든지 관계없다는 것을 나타낸다. 즉, 식 (5)의 의미는 회로의 내부 상태와 관계없이 주 입력단에 시험 벡터 $V$가 인가되면 회로의 내부 상태는 내부 상태 $S$가 된다는 것을 나타낸다. ■

4. 피이드백 루프를 통한 논리값의 전파

시험 벡터를 생성하는 중에 백트레이싱을 통해서 논리값이 할당된 PPI에 피이드백 루프를 통해서 논리값이 전파될 수가 있다. 이 논리값은 논리값 X일 수도 있고 X가 아닌 다른 논리값일 수도 있는데, 각각의 경우는 다음과 같다.

**[Heuristic 1]** 피이드백 루프를 통한 X가 아닌 논리값의 전파

백트레이싱을 통해서 논리값이 할당된 PPI에 피이드백 루프를 통해서 X가 아닌 논리값이 전파되면, 피이드백 루프를 통해서 계속해서 임플리케이션(implication)을 수행한다.

**[이유]**

[Heuristic 1]은 동기 순차 회로의 경우와 다른 점인데, 동기 순차 회로에서는 클럭 펄스가 발생하지 않는 어느 한 타임 프레임 동안에는 플립플롭을 통해서 논리값이 전파되지 않는다. 그러나, 비동기 순차 회로는 피이드백 루프를 통해서 논리값의 변화가 즉시 전파된다. 그러므로 [Heuristic 1]은 비동기 순차 회로의 피이드백 루프의 특성을 적용하기 위한 방법이다. ■

**[Heuristic 2]** 피이드백 루프를 통한 논리값 X의 전파

백트레이싱을 통해서 논리값이 할당된 PPI에 피이드백 루프를 통해서 논리값 X가 전파되면, 그 논리값 X를 피이드백 루프를 통해서 더 이상 전파시키지 않는다. 그리고, 다음의 목표는 백트레이싱을 통해서 PPI에 할당되었던 논리값을 그 PPI에 대응하는 PPO에 할당하는 것이 되고 이 목표를 가지고 백트레이싱을 수행한다.

**[ 이유 ]**

백트레이싱을 통해서 PPI에 논리값이 할당되었다면, 그 논리값은 결함 신호를 전파하거나 내부 상태를 만족시키기 위해서 반드시 필요한 논리값이다. 만일, 백트레이싱을 통해서 논리값이 할당된 PPI에 논리값 X를 할당하고 임플리케이션을 수행하게 되면, 다음 백트레이싱의 결과는 같은 PPI에 같은 논리값을 할당하는 것이 될 수 있다. 이렇게 되면 이 PPI에는 백트레이싱을 통해서 할당된 논리값과 논리값 X가 교대로 무한히 반복하여 할당될 수 있다. 그러므로, 이와 같은 상황을 피하기 위해서 [ Heuristic 2 ] 를 적용한다. ■

5. 백트랙킹 방법

시험 벡터 생성 중에 백트레이싱을 통해서 할당된 논리값이 잘못된 결정이었다는 것이 판명되면, 백트랙킹을 수행하는데 백트랙킹할 논리값은 ESSENTIAL [6] 에서 제안된 방법을 적용하여 결정한다. 조합 회로의 경우에 백트랙킹을 수행할 때에는 단순히 전에 할당했던 논리값을 새로운 논리값으로 바꾸고 임플리케이션을 수행하면 된다. 그 이유는 회로에 메모리 소자가 없기 때문이다. 그러나, 비동기 순차 회로에는 메모리 소자(피이드백 루프)가 있어서 잘못 할당되었던 논리값의 영향이 메모리 소자에 저장될 수 있다. 이러한 현상을 방지하기 위해 [ Heuristic 3 ] 을 적용하여 백트랙킹을 수행한다.

**[ Heuristic 3 ]** 백트랙킹 방법

1. 백트랙킹할 논리값을 할당한다.
2. 백트레이싱을 통해 논리값이 할당되었던 주 입력단과 PPI에 현재 할당되어 있는 논리값을 저장한다.
3. 모든 신호선에 논리값 X를 할당한다.
4. 2번에서 저장했던 논리값을 대응하는 신호선에 할당하고 임플리케이션을 수행한다. 이때 PPI에 할당될 논리값을 먼저 할당한다.

**[ 이유 ]**

[ Heuristic 3 ] 을 적용하면 잘못 할당되었던 논리값에 의한 영향을 회로로부터 제거하여 신호선에 잘못된 논리값이 할당되는 것을 방지해준다. 그리고, PPI에 할당될 논리값을 먼저 할당하는 이유는 PPI에 할당될 논리값은 전 타임 프레임에서 할당될 논리값이기 때문이다. ■

6. 동일 상태 방문 횟수 제한

시험 벡터를 생성하는 과정에서 시험 벡터 생성 알고리즘이 무한 루프에 빠질 가능성이 있다. 이것을 그림 3을 예로 하여 설명한다. 결함 신호를 주 출력단으로 전파하기 위하여 State 0으로 이동했고, State 0을 만족 시키기 위하여 State 1의 상태로 이동했다고 하자. 이때, State 1을 만족시키기 위해서 State 2로 이동했고, 다시 State 2를 만족시키기 위해서 State 1로 이동했다고 하자. 본 논문에서 제안된 알고리즘은 *deterministic* 알고리즘이기 때문에 State 1을 만족시키기 위해서 또 다시 State 2로 이동하게 된다. 즉, 시험 벡터 생성기는 State 1과 State 2를 무한히 반복하게 되어 무한 루프에 빠지게 된다. 이와 같은 현상을 방지하기 위하여 동일 상태는 한번만 방문하도록 한다면, State 1을 두번째 방문할 때 백트랙킹하여 State 3으로 이동하여 ( 1, 0, 0 ), ( 0, 1, 1 ), ( 1, 1, 1 )의 시험 벡터를 생성하게 된다. 그러나, 첫번째 시험 벡터와 두번째 시험 벡터의 Hamming distance는 3이기 때문에 기본 모드에서 적용 가능한 시험 벡터가 아니다. 그러므로, 시험 벡터 생성기는 시험 벡터를 생성하지 못하게 된다. 그러나, 만일 동일 상태를 두 번 방문할 수 있도록 허용하면서 시험 벡터를 생성하면, 시험 벡터 생성기는 State 0, State 1, State 2, State 1, State 3의 순서로 내부 상태를 이동하여, ( 1, 0, 0 ), ( 1, 0, 1 ), ( 0, 0, 1 ), ( 0, 1, 1 ), ( 1, 1, 1 )의 시험 벡터를 생성하게된다. 시험 벡터의 논리값 X를 논리값 0으로 바꾼 것은 다음 절( Ⅲ. 7 절 )에서 설명한다. 즉, 동일 상태 방문 횟수를 제한하지 않으면 시험 벡터 생성기가 무한 루프에 빠지게 된다. 그리고, 동일 상태 방문 횟수의 제한을 작게 하면 시험 벡터가 존재함에도 불구하고, 시험 벡터를 생성하지 못할 수 있고, 동일 상태 방문 횟수의 제한을 크게 하면 루프를 많이 돌기 때문에 CPU 타임이 증가하고, 생성된 시험 벡터의 타임 프레임 수가 커질 수 있다. 이와 같은 문제점을 해결하기 위해서, 시험 벡터를 생성할 때에는 [ Heuristic 4 ] 를 적용한다.

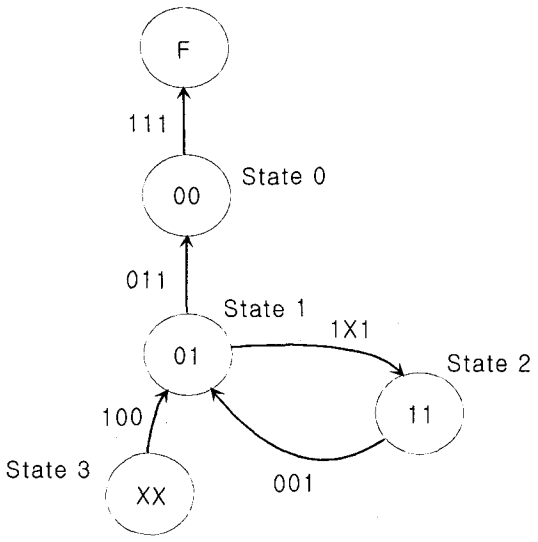


그림 3. 동일 상태 방문 횟수 제한
Fig. 3. Limit of the times to visit equal state.

**[Heuristic 4]** 반복 기법( Iterative method )

최대 방문 횟수는 선택할 수 있도록 하고, 동일 상태 방문 횟수 제한을 1부터 최대 방문 횟수까지 1씩 증가시키면서 시험 벡터를 생성한다. 만일, 도중에 시험 벡터가 생성되면 다음 결함에 대한 시험 벡터를 생성한다. 이 방법을 반복 기법이라고 정의한다.

**[이유]**

반복 기법을 적용하면 시험 벡터를 생성하기 위해 소요되는 CPU 타임은 증가하지만 시험 벡터의 타임 프레임수를 최소로 하면서 결함 시험도를 높일 수 있다. ■

7. 시험 벡터의 기본 모드 적용 가능성 판단 및 수정

생성된 시험 벡터가 기본 모드에서 적용 가능하도록 하기 위해서, 시험 벡터를 생성하면서 시험 벡터의 기본 모드 적용 가능성을 판단하여야 하고, 생성된 시험 벡터에서 논리값 X를 수정하여야 한다.

1) 기본 모드 적용 가능성 판단

시험 벡터가 기본 모드에서 적용 가능한지를 판단하기 위해서는 [ 알고리즘 1 ] 을 적용한다. 이 때, 논리값 X를 실제로 바꾸는 것이 아니라, 확인만 한다.

**[알고리즘 1]**

① 어떤 입력의 논리값이 모든 타임 프레임에 대해서 논리값 X인 경우는 그 논리값 X에 논리값 0(또는 논리값 1)을 할당한다.

② 전 타임 프레임(다음 타임 프레임)의 시험 벡터와 Hamming distance가 1이라면 현재의 타임 프레임의 시험 벡터에 존재하는 논리값 X에는 전 타임 프레임(다음 타임 프레임)의 시험 벡터의 논리값을 할당한다. 만일, 이 과정에서 구해진 Hamming distance가 1보다 크면, 주어진 시험 벡터는 기본 모드에서 적용 가능한 시험 벡터가 아니다.

③ 양쪽 타임 프레임의 논리값이 같으면 그 사이에 존재하는 논리값 X는 변화시킬 필요가 없다.

이 과정을 통해서 논리값 X를 수정한 후, 남아 있는 논리값 X에 대해서 할당 가능한 모든 논리값의 조합을 할당하면서 기본 모드에서 적용 가능한지를 확인한다. 하나의 조합에 대해서라도 기본 모드에 적용 가능하다면, 주어진 시험 벡터는 기본 모드에 적용 가능하다.

2) 시험 벡터 수정

생성된 시험 벡터를 기본 모드에서 동작하는 비동기 순차 회로에 적용하기 위해서는 시험 벡터에 존재하는 논리값 X를 논리값 0 또는 논리값 1로 수정해야 하는데, [ 알고리즘 1 ] 을 적용하여 시험 벡터를 수정한다. 이 때는 논리값 X를 실제로 바꾼다.

8. 시험 벡터 출력

[ 알고리즘 1 ] 을 적용하여 시험 벡터를 수정하면 연속된 시험 벡터가 동일 할 수 있다. 만일, 연속된 시험 벡터가 동일하면 한 시험 벡터만 시험 벡터로 받아들인다. 그 이유는 연속된 입력 벡터가 동일하면, 비동기 순차 회로에서는 아무 의미도 없기 때문이다. 이것은 비동기 순차 회로가 기본 모드에서 동작한다는 것을 전제로 하기 때문에 가능하다. 이것은 동기 순차 회로와 다른 점인데, 동기 순차 회로에서는 연속된 시험 벡터가 동일하더라도 모두 시험 벡터로 받아들여야 한다. 그 이유는 동기 순차 회로에서는 시험 벡터가 클럭 펄스와 관련이 있기 때문이다.

9. 시험 벡터 생성 알고리즘

atpg()는 전체 시험 벡터 생성 함수이고, Justify()는 내부 상태를 만족시키는 함수 이다.

```
    atpg()
    begin
1     if( error_at_PO() and is_cur_state_justified() )
2       if( check_status() == SUCCESS )
3         return SUCCESS;
4       else
```

```
5         return Justify( 1 );
6       end if
7     end if
8     if( test_not_possible() == YES )
9       return FAILURE;
10    end if
11    objective = Objective()
12    data = Backtrace( objective )
13    if( Imply( data ) != FAILURE )
14      if( atpg() == SUCCESS )
15        return SUCCESS;
16      end if
17    data = next_sequence_value();
18    if( Imply( data ) != FAILURE )
19      if( atpg() == SUCCESS )
20        return SUCCESS;
21      end if
22    end if
23    data = next_sequence_value();
24     if( Signal line of data is not reachable from
      target fault )
25      Imply( data );
26        return FAILURE;
27    end if
28    if( Imply( data ) != FAILURE )
29      if( atpg() == SUCCESS )
30        return SUCCESS;
31      end if
32    end if
33    data = next_sequence_value();
34    if( Imply( data ) != FAILURE )
35      if( result == SUCCESS )
36        return SUCCESS;
37      end if
38    end if
39    data = next_sequence_value();
40    Imply( data )
41    return FAILURE
    end

    Justify( frame )     /* Justify Internal state. */
    begin
1     if( check_different_bit() == FAILURE )
2       return FAILURE;
3     end if
4     objective = J_objective( &flag );
5     switch( flag )
6       case SUCCESS :
7         if( check_Hamming_distance() == FAILURE )
8           return FAILURE;
9         else
10          return SUCCESS;
11      end if
12    case FAILURE :
13      return FAILURE;
14    case CONTINUE :
15      if( check_state() == FAILURE )
16        return FAILURE;
17      end if
18      if( check_Hamming_distance() == FAILURE )
19        return FAILURE;
20      end if
21      switch( Justify( frame+1 ) )
22        case SUCCESS :
23          return SUCCESS;
24        case FAILURE :
25          return FAILURE;
26      end case
27  end case
28  data = Backtrace( objective );
29  if( Imply() != FAILURE )
30    if( Justify( frame ) == SUCCESS )
31      return SUCCESS;
32    end if
33  end if
34  data = next_sequence_value();
35  if( Imply() != FAILURE )
36    if( Justify( frame ) == SUCCESS )
37      return SUCCESS;
38    end if
39  end if
40  data = next_sequence_value();
41   if( Signal line of data is not reachable from
    target fault )
42    Imply( data );
43    return FAILURE;
44  end if
45  if( Imply( data ) != FAILURE )
46    if( Justify( frame ) == SUCCESS )
47      return SUCCESS;
48    end if
49  end if
50  data = next_sequence_value();
50  if( Imply( data ) != FAILURE )
51    if( Justify( frame ) == SUCCESS )
52      return SUCCESS;
53    end if
54  end if
55  data = next_sequence_value();
56  Imply( data )
57  return FAILURE
  end
```

10. 피이드백 루프를 찾는 알고리즘

피이드백 루프는 게이트를 노드(node)로, 연결선을 에지(edge)로 하여, 주입력단으로부터 주출력단까지 DFS( Depth First Search )를 하면서, 주출력단으로 연결된 경로와 이미 방문한 노드로만 연결된 경로를 찾는다. 이때 이미 방문한 노드로만 연결된 에지 중에서 가장 주입력단에 가까운 에지를 피이드백 루프로 정한다.

```
initialize the labels of all the edges to 0;
find_feedback( pseudo_PI );

find_feedback( v )
begin
    for each fanout vertex( w ) of v
        cur_label = the label of the edge between the
        vertex v and w;
        if( cur_label == 1 )
            label = 0;
            return;
        else if( cur_label == 2 )
            label = 2;
            return;
        else
            set the label of the edge between the
            vertex v and w to 1;
            find_feedback( w );
            set the label of the edge between the
            vertex v and w to label;
        end if
    end for
    if( one of the labels of the outgoing edge of v is
    2 or
       v has no outgoing edge )
        label = 2;
        return;
    end if
end
```

## Ⅳ. 시험 벡터 생성 시뮬레이션

비동기 순차회로에 대해서 시험 벡터를 생성하기 위해서 기존에 해온 접근 방식과 본 논문에서 제안한 방식을 그림 4 와 그림 5를 예로 하여 설명한다.

먼저 기존의 접근 방식부터 설명한다. 그림 4 (a) 회로에 대해서 시험 벡터를 생성한다고 하자. 회로에 주어진 대상 결함을 검출하기 위해서 피이드백 루프에는 논리값 0이 할당 되어야하고, 주 입력단에는 시험 벡터 ( $X$, 1 ) 이 할당되어야 한다. 이때 회로는 그림 4 (b)와 같고, 결함은 검출된다. 이제 내부 상태를 만족시키기 위하여 피이드백 루프에 할당된 논리값 0으로 백트레이싱을 수행하면 주 입력단에 시험 벡터 ( $X$, 0 )이 할당됨을 찾을 수 있다. 이때 회로에는 그림 4 (c)와 같이 논리값이 할당된다. 즉, 기존의 접근 방식으로 생성된 시험 벡터는( 1, 0 ),( 1, 1 )이 된다. 논리값 $X$는 논리값 1로 치환하였다. 이제 생성된 시험 벡터를 회로에 적용한다. 시험 벡터 ( 1, 0 ) 을 인가하면 회로에는 그림 4 (d)와 같이 논리값이 할당되고, 다음 시험 벡터 ( 1, 1 )을 인가하면 그림 4 (e)와 같이 논리값이 할당된다. 이때 피이드백 루프를 통해서 논리값이 전파되면 회로에는 그림 4 (f)와 같이 논리값이 할당된다. 즉, 기존의 접근 방식으로 생성된 시험 벡터를 적용하면 회로는 그림 4 (e) 상태와 그림 4 (f) 상태를 무한히 반복하여 오실레이션이 발생된다.

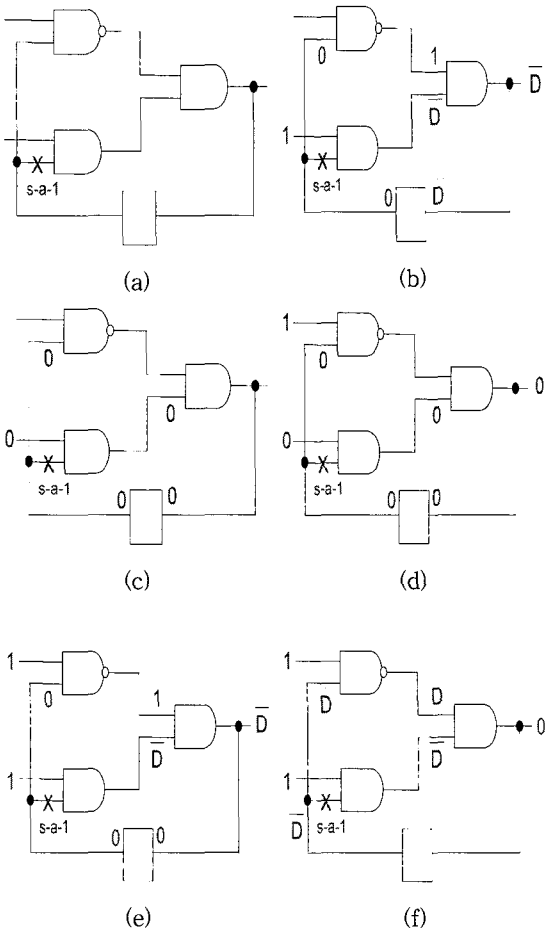


그림 4. 기존의 시험 벡터 생성 예

Fig. 4. An example of traditional test vector generation.

다음은 본 논문에서 제안한 방법으로 시험 벡터를 생성한다. 먼저 대상 결함을 주 출력단에서 검출하기 위하여, 피이드백 루프에는 논리값 0 이 할당되어야 하고, 주 입력단에는 시험 벡터 ( $X$ , 1 )이 할당되어야 한다. 이때 회로에는 그림 5. (a) 와 같이 논리값이 할당된다. 그런데, 피이드백 루프를 통해서 논리값 $X$ 가 아닌 논리값이 전파되었기 때문에 계속해서 피이드백 루프를 통해서 시뮬레이션을 수행한다. 그러면 그림 5. (b)와 같이 논리값이 할당된다. 이 때는 피이드백 루프를 통해서 논리값 $X$가 전파되었기 때문에 더 이상 시뮬레이션을 수행하지 않고, 피이드백 루프에 할당된 논리값으로 백트레이싱을 수행한다. 이 때 주 입력단에는 시험 벡터 ( 1 , 1 )이 할당되어야 한다는 것을 찾을 수 있다. 그러나, 이 때 그림 4 (e)와 그림 4 (f) 회로가 교대로 반복하여 오실레이션을 발생했다는 것을 판단할 수 있다.

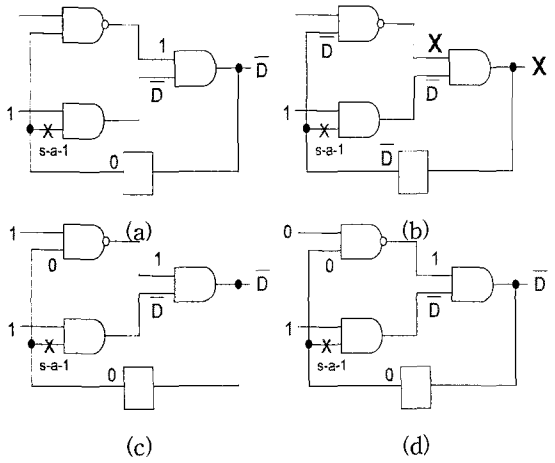


그림 5. 본 논문에서 제안된 시험 벡터 생성 예
Fig. 5. An example of test vector generation presented in this paper

그러므로 백트랙킹을 수행하여 주 입력단에 시험 벡터 ( 0 , 1 )을 할당한다.

내부 상태 만족은 기존의 방법과 같은 방법으로 수행되어 시험 벡터 ( $X$ , 0 )을 찾게된다. 이때 시험 벡터는 ( 0 , 0 ), ( 0 , 1 )이 된다. 그리고, 이 시험 벡터는 오실레이션을 발생시키지 않으면서 대상 결함을 검출한다. 그림 4 (a) 회로의 예와 같이 기존의 접근 방법으로 시험 벡터를 생성하면 시험 벡터 생성기가 시험 벡터를 생성했다 하더라도, 실제 비동기 순차 회로에 적용했을 때 오실레이션을 발생시켜 대상 결함을 검출하지 못할 수가 있다. 그러나, 본 논문에서 제안된 방법으로 시험 벡터를 생성하면 생성된 시험 벡터는 대상 결함을 검출하는 것이 보장된다.

물론 기존의 접근 방법에서 논리값 $X$를 논리값 0으로 치환했다면, 시험 벡터는 본 논문에서 제안된 방법으로 생성한 시험 벡터와 같아진다. 그러나, 이것은 우연한 상황에 의해서 시험 벡터가 대상 결함을 검출할 수도 있고 그렇지 못할 수도 있다는 것을 나타낸다. 본 논문에서 제안된 방법은 이와 같이 시험 벡터에 존재하는 논리값 $X$도 체계적인 방법으로 논리값 0 또는 논리값 1로 치환한다.

## V. 실험 결과

실험은 TTL 74 시리즈 중에서 비동기 순차 회로에 대해서 실험하였다. 각각의 회로의 내부 구조는 [7] 을 참고하여 ISCAS'89 벤치마크[8] 형식으로 구성하였다. 벤치마크 회로에 대한 간단한 설명은 표 2에 제시하였다. 표 2에 제시된 벤치마크 회로에 대한 실험 결과를 표 3에 제시하였다.

실험은 RAM이 64M byte인 Ultra spark에서 수행하였고, 한 결함에 대한 백트랙킹 횟수는 500회로 제한하였다. 동일 상태 방문 횟수는 5회로 제한하였고, 반복 기법을 적용하였다. 결함에 대한 설명은 Ⅳ절에서 설명하였고, 결함 시험도는 식 (7)을 적용하여 구하였다.

$$fault\ coverage = \frac{\#\ detected\ faults}{\#\ total\ faults} \times 100\ [\%] \quad (7)$$

표 2. 실험용 회로의 설명
Table 2. Explanation of experimental circuits.

| 회로 이름 | 피이드백 루프 수 | 기능 | 상용 칩 이름 |
|---|---|---|---|
| neffwr | 1 | J-K Negative Edge-triggered Flip Flop With Rest | 74LS73AP |
| dwcp | 3 | D flip flop With Clear and Preset | 74S74 |
| 3to8dec | 3 | 3-line TO 8-line DECoder/demultiplexer with address latch | 74LS137P |
| 2by4pbm | 5 | 2-bit-BY-4-bit Parallel Binary Multiplier | SN74LS261 |
| 4bcpr | 4 | 4-Bit Cascadable Priority Register | SN74278 |
| 4blwc | 4 | 4-Bit Latch With Clear | SN74116 |
| pulsesyn | 4 | PULSE SYNchronizers/drivers | SN74120 |

표 3. 실험 결과
Table 3. Experimental results.

| 회로 이름 | 전체 결함수 | red. faults | untest. faults | uncheck. faults | abort. faults | backt-rackings | time frames | CPU Time [sec] | fault coverage [%] |
|---|---|---|---|---|---|---|---|---|---|
| neffwr | 35 | 1 | 0 | 0 | 24 | 51428 | 15 | 93.63 | 28.57 |
| dwcp | 38 | 1 | 1 | 0 | 1 | 3090 | 63 | 5.15 | 92.11 |
| 3to8dec | 96 | 0 | 0 | 0 | 1 | 4533 | 127 | 20.33 | 98.96 |
| 2by4pbm | 159 | 0 | 0 | 0 | 2 | 6925 | 186 | 55.03 | 98.74 |
| 4bcpr | 89 | 0 | 0 | 0 | 1 | 3763 | 144 | 11.55 | 98.88 |
| 4bIwc | 86 | 0 | 3 | 0 | 5 | 14026 | 145 | 62.48 | 90.70 |
| pulsesyn | 41 | 2 | 0 | 0 | 21 | 53967 | 42 | 98.52 | 43.90 |

대부분의 비동기 순차 회로에 대해서 높은 결함 시험도를 나타내었지만, neffwr과 pulsesyn에 대해서는 aborted 결함이 많이 발생하였다. 이것은 두 회로에 대해서 본 논문에서 제안된 Heuristic 이 효율적으로 적용되지 못했다는 것을 나타낸다. 벤치마크 회로 중에서 2by4pbm에 대한 ISCAS'89 벤치마크 format netlist를 부록에 수록하였다.

## VI. 결 론

본 논문에서는 기본 모드에서 동작하는 비동기 순차 회로를 시험할 수 있는 시험 벡터 생성 알고리즘을 제안하였다. 본 논문에서 제안된 알고리즘은 비동기 순차 회로에 임계레이스 문제와 순환 문제가 발생하는 것을 최소로 하는 시험 벡터를 생성한다. 임계레이스 문제를 최소화한 것은 연속된 시험 벡터의 Hamming distance를 1로 유지함으로써 가능했고, 순환 문제를 최소화한 것은 순환을 발생시키는 입력 벡터를 시험 벡터로 받아들이지 않음으로써 가능했다. 또한, 기존의 접근 방식에서는 피이드백 루프에 플립플롭이 있는 것처럼 가정하고 시험 벡터를 생성했는데, 본 논문에서는 피이드백 루프의 특성을 그대로 살리면서 시험 벡터를 생성했기 때문에, 시험 벡터가 대상 결함을 검출할 가능성이 매우 높아졌다.

즉, 본 논문에서 제안된 방법은 연속된 시험 벡터의 Hamming distance를 1로 유지하고, 주 입력단의 논리값이 변했을 때 일시적으로 unstable한 논리값이 여러번 피이드백 루프를 통해서 전파되는 것도 고려하여 시험 벡터를 생성하였다. 그렇기 때문에 기존의 방법에서와 같이 생성된 시험 벡터가 대상 결함을 검출하지 못하는 문제점을 해결하였다.

## 부 록

Benchmarck 회로 중에서 2by4pbm에 대한 netlist

```
INPUT(B0)
INPUT(B1)
INPUT(B2)
INPUT(B3)
INPUT(B4)
INPUT(M0)
INPUT(M1)
INPUT(M2)
INPUT(G)

OUTPUT(O0)
OUTPUT(O1)
OUTPUT(O2)
OUTPUT(O3)
OUTPUT(O_BAR4)

G1 = NOT(B0)
G2 = NOT(B1)
G3 = NOT(B2)
G4 = NOT(B3)
G5 = NOT(B4)
G6 = AND(M0, M1)
G7 = NOR(M0, M1)
G8 = NOR(M2, G9)
G9 = NOT(G)

G10 = NOT(G1)
G11 = NOT(G2)
G12 = NOT(G3)
G13 = NOT(G4)
G14 = NOT(G5)
G15 = NOR(G6, G7)
G16 = NOR(G8, G9)

G17 = AND(G10, G6, G8)
G18 = AND(G1, G7, G16)
G19 = AND(G11, G15, G8)
G20 = AND(G2, G15, G16)
G21 = AND(G9, G42)
G22 = AND(G11, G6, G8)
G23 = AND(G2, G7, G16)
G24 = AND(G12, G15, G8)
G25 = AND(G3, G15, G16)
G26 = AND(G9, G43)
G27 = AND(G12, G6, G8)
G28 = AND(G3, G7, G16)
G29 = AND(G13, G15, G8)
G30 = AND(G4, G15, G16)
G31 = AND(G9, G44)
G32 = AND(G13, G8, G6)
```

```
G33 = AND(G4, G7, G16)
G34 = AND(G14, G15, G8)
G35 = AND(G15, G5, G16)
G36 = AND(G9, G45)
G37 = AND(G14, G6, G8)
G38 = AND(G5, G7, G16)
G39 = AND(G14, G15, G8)
G40 = AND(G15, G5, G16)
G41 = AND(G9, G47)

G42 = OR(G17, G18, G19, G20, G21)
G43 = OR(G22, G23, G24, G25, G26)
G44 = OR(G27, G28, G29, G30, G31)
G45 = OR(G32, G33, G34, G35, G36)
G46 = NOR(G37, G38, G39, G40, G41)
G47 = NOT(G46)

O0 = BUFF(G42)
O1 = BUFF(G43)
O2 = BUFF(G44)
O3 = BUFF(G45)
O_BAR4 = BUFF(G46)
```

## 참 고 문 헌


[1] M.MORRIS MANO, *Digital Design*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1991.

[2] Oriol Roig et al., "Automatic Generation of Synchronous Test Patterns for Asynchronous Circuits," in *Proc. Design Automation Conference* pp. 620-625, 1997.

[3] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," in *Proc. IEEE Trans. Comput.*, vol. C-30, pp. 215-222, Mar. 1981.

[4] H. Fujiwara and T. Shimono, "On the Acceleration of Test Generation Algorithms," in *Proc. IEEE Trans. Comput.*, vol. C-32, pp. 1137-1144, Dec. 1983.

[5] M. H. Schulz et al., "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System," in *Proc. IEEE Trans. Computer-Aided Design*, vol. 7, pp. 126-137, Jan. 1988.

[6] M. H. Schulz et al., "ESSENTIAL: An Efficient Self-Learning Test Pattern Generation Algorithm For Sequential Circuits," in *Proc. IEEE Int. Test Conf.*, pp. 28-37, 1989.

[7] *The Bipolar Digital Integrated Circuits Data Book*, TEXAS INSTRUMENTS, 1985.

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


## 저 자 소 개


**閔 炯 福(正會員)** 第 35 卷 C編 第 4號 參照
현재 성균관대학교 전기 전자 및 컴퓨터공학부 부교수

**李 宰 勳(正會員)** 第 35卷 C編 第 4號 參照
현재 성균관대학교 전기공학과 박사과정

**趙 庚 衍(正會員)**
1970년 8월 11일생. 1995년 2월 성균관대학교 전기공학과(학사). 1995년 1월 ~ 1995년 7월 한국전력공사. 1998년 2월 성균관대학교 전기공학과(석사). 1998년 3월 ~ 현재 전자부품종합기술연구소 연구원. 주관심분야는 ATPG, DFT, RF 통신