# 상위 수준 설계에서의 테스트패턴 생성

김종현, 박승규, 김동욱
광운대학교 전자재료공학과
139-701 서울시 노원구 월계동 447-1
e-mail : saltlite@explore.kwangwoon.ac.kr

# High Level Test Generation

Jonghyeon Kim, Seungkyu Park Dongwook Kim
Dept. of Electronic Materials Eng. Kwangwoon Univ.
447-1 Wolgye-Dong Nowon-Ku, 139-701, Seoul, Korea

## Abstract

**IC testing plays a very important role in IC manufacturing process. Modern complex ASIC chips making it difficult for gate level and RLT level test generation techniques to generate good test vector in resonable time.**

**In this paper we proposed new test pattern generation method in VHDL description to detect manufacturing faults. This method based on software testing can easily generate test vector and independent to synthesis result.**

## I. 서 론

반도체 제조시 공정상의 결함을 검출하는 테스트 과정은 시스템 설계 및 반도체 설계과정에서 매우 중요한 부분을 차지하고 있다. 시스템상에서 IC결함에 의한 고장을 검출 및 수리하는데는 많은 시간과 비용이 요구되며 따라서 결함이 있는 IC들은 시스템 상이 아닌 IC 제조시에 판별되어야 한다. 따라서 제조공정시의 결함을 검출 할 수 있도록 많은 연구가 진행 되었다.[1][6][9]

현재까지의 하드웨어 고장검출을 위한 테스트 패턴 생성방법은 주로 게이트레벨[4][9] 및 RTL레벨[6]에서 이루어졌다. 게이트 레벨에서의 테스트 패턴 생성 방법은 높은 고장 검출율을 얻을 수 있으나 회로의 크기가 커질수록 테스트 패턴을 생성하기 위해서는 더욱 많은 시간이 요구된다. 또한 게이트 레벨에서는 대상 회로가 복잡해질수록 효과적인 테스트 패턴 생성이 어려워지며 패턴을 생성 할 수 없는 경우도 증가하게 되어 테스트 생성의 비용 및 시간의 증가를 가져온다. 이는 전체적인 디자인 시간 및 비용의 증가를 가져오게 된다.

현대의 디지털 VLSI는 공정 기술 및 설계 의 발달로 인해 고집적화 및 여러 기능을 복합적으로 수행할 수 있도록 고기능화 되고 있다. 일반적으로 칩이 고기능화, 고집적화 될수록 공정시의 고장확률은 증가하게 되며 테스트 패턴 생성은 더욱 힘들어 진다. 따라서 빠르고 쉽게 테스트 패턴을 생성할 수 있는 새로운 방법이 요구되고 있다.[2][3][4]

디지털 회로 설계 방법은 기존의 스케메틱 베이스에서 HDL[7]베이스로 옮겨 가고 있는 추세이며 이는 회로가 복잡해짐에 따라 기존의 스케메틱 방법으로는 효율적인 설계가 힘들기 때문이다. HDL을 이용하면 복잡한 회로 및 알고리듬을 쉽게 구현 할 수 있으며 회로의 재사용이 매우 용이하다. HDL을 이용하여 회로 설계시 합성이라는 과정을 거치게된다. 이때 합성 툴(tool) 및 합성 조건 등에 따라 합성의 결과는 달라지게 되며 이에 따른 제조 공정도 달라지게 된다. 따라서 합성할 때마다 하드웨어 고장 검출을 위한 테스트 패턴을 재 생성해야 한다.

현재 널리 사용되고 있는 HDL인 VHDL은 회로의 동작에 관해 기술을 하고 있으므로 VHDL 상에서 테스트 패턴을 생성한다면 보다 빠르고 쉽게 테스트 패턴을 생성 할 수 있을 것으로 생각된다. 또한 제조공정과 무관한 테스트 패턴을 생성함으로써 합성시에 다시 패턴을 생성 할 필요가 없으며 따라서 패턴의 재사용이 가능하다. 따라서 본 논문에서는 이러한 VHDL[5][7]

의 장점과 VHDL과 프로그래밍 언어의 유사점을 이용하여 이전의 테스트패턴 생성 방법의 단점을 보완하고 앞으로 다가올 설계환경에 맞는 새로운 테스트패턴 생성 방법에 대해 연구하였다.

## II. 소프트웨어 테스트

소프트웨어 테스트는 코딩이 의도하는데로 됐는지 검사하는 것으로써 크게 black box 테스트와 white box 테스트로 나누어 진다.

Black box 테스트는 코딩의 내용은 검사하지 않고 소프트웨어가 원하는 동작을 올바르게 수행하는지를 검사하는 것으로써 특정한 입력값에 대한 출력값을 확인하여 소프트웨어의 오류를 찾아낸다. 반면에 white box 테스트는 코딩자체가 테스트의 목적이 되는 것으로써 data flow testing, path testing 등이 white box 테스트에 속한다. 즉, coding 내용을 특정한 방법에 의해 확인함으로써 coding의 오류를 직접 찾아낸다. 소프트웨어 테스트의 목적이 소프트웨어의 코딩오류를 찾아내는 것이므로 VHDL에 적용하여 하드웨어 오류를 검사하는 테스트패턴을 생성하는 것이 적절하지 못할 수 있다. 따라서 본 연구에서는 소프트웨어 테스트를 기초로 하여 하드웨어와의 연관성을 고려하여 테스트패턴을 생성한다. 즉 공정상의 결함에 의해 올바른 동작이 일어나지 않는것을 소프트웨어 코딩의 오류에 의해 소프트웨어가 올바르게 동작하지 않는것에 연관시켜서 소프트웨어 테스트에 기초한 하드웨어 고장을 검출할 수 있는 테스트 패턴을 생성하는 방법에 대해 연구하였다.

## III. 테스트 패턴 생성

### 3.1 상위수준 테스트 패턴 생성

VHDL상에서 테스트 패턴을 생성할 경우 다음과 같은 장점이 있다.

#### 3.1.1 패턴의 재사용

VHDL을 이용하여 회로를 설계할 경우 VHDL 특성상 재사용의 확률이 매우 크며 또한 여러 CAD툴에서 사용될 수 도 있다. 재사용시에 합성과정을 거치게 되는데 이때 합성결과가 여러 가지로 나타날 수 있다. 따라서 합성결과에 따라 테스트 패턴의 재생성을 필요로 하게 된다. 합성 결과에 무관한 테스트패턴을 VHDL상에서 생성할 수 있다면 테스트 패턴을 한번만 생성하면 되므로 테스트 비용과 시간이 절약된다.

#### 3.1.2 패턴생성의 용이성

게이트 레벨이나 RTL레벨에서 테스트 패턴 생성등 기존의 테스트패턴 생성 방법은 높은 고장검출율을 얻을 수 있다. 하지만 회로의 크기가 커지고 회로의 복잡도가 증가할수록 기존의 방법에 의한 테스트 패턴 생성 방법은 생성시간 및 비용의 증가를 가져오게 된다. 이는 기존의 테스트패턴 생성방법은 회로의종류 및 functional 정보가 반영되지 못하고 오직 패턴생성 알고리듬에 의존하기 때문이다. 반면에 VHDL에서 패턴을 생성할 경우 VHDL자체가 회로의 여러 정보들을 포함하고 있으므로 functional 정보 및 회로의 여러 정보등을 이용할 수가 있다. 따라서 회로의 크기가 커지고 복잡도가 증가해도 보다 쉽게 테스트패턴을 생성할 수가 있으며 순차회로와 같이 기존의 방법이 쉽게 패턴을 생성 할 수 없었던 회로도 쉽게 테스트패턴을 생성 할 수가 있다.

### 3.2 VHDL 코드의 그래프 변환

VHDL상에서 테스트패턴을 생성하기 위해서 VHDL 구문을 그래프로 변환한다. VHDL의 구문은 크게 조건문(condition)과 기술문(statement)으로 나눌수 있다. 즉 if-then, loop, case 문등은 조건문이며 출력 및 변수에 대한 값의 대입은 기술문에 속한다. 그림 1은 VHDL로 설계한 4×1 MUX를 보여주고 있으며 그림 2는 그림 1에 대한 그래프변환 결과를 나타내고 있다. 그래프는 노드와 edge 그리고 path로 구성된다.

### 3.3 고장 모델

본 연구의 목표는 VHDL상에서 생성한 테스트 패턴을 합성된 게이트레벨에 적용하여 stuck-at-fault를 검출하는 것이다. 따라서 게이트레벨 상에서의 고장 모델은 single stuck-at-fault로 하며 패턴생성을 위한 그래프상에서는 기술문과 조건문을 연결하는 edge를 고장모델로 삼는다. 즉, 게이트상의 stuck-at-fault에 의해 올바른 데이터값이 출력까지 전달되지 못함을 그래프 상의 edge의 고장에 의해 올바른 값이 그래프 상에서 전달되지 못함으로 가정하고 테스트 패턴을 생성한다.

## IV. 테스트 패턴 생성 알고리듬

테스트패턴 생성의 기본 알고리듬은 그림 3과 같다. 즉 먼저VHDL을 그래프로 변환한 후 테스트 대상 edge의 집합을 만든다. 이때 집합 중에서 같은 테스트

```
Library IEEE;
use IEEE.std_logic_1164.all;

entity mux4_1 is
port( sel:in        std_logic(1 downto 0);
      a,b,c,d:in    std_logic;
      y:out         std_logic);
end mux4_1;

architecture logic of mux4_1 is
begin
process(sel,a,b,c,d)
begin
   if(sel(1)='0') then
      if(sel(0)='0' then
         y<=a;
      else
         y<=b;
      end if;
   else
      if(sel(0)='0') then
         y<=c;
      else
         y<=d;
      end if;
   end if;
end process;
end logic;
```

그림 1. 4-1 MUX

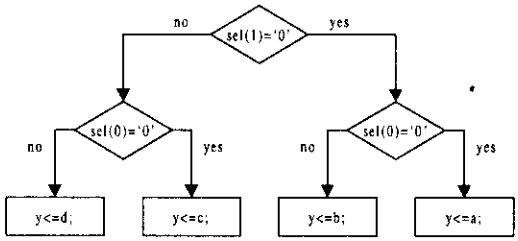


그림 2. 그림1의 그래프 변환

패턴으로 검사가 가능한 중복 edge가 생기는데 중복되는 edge를 edge집합에서 제거한다. Edge집합 중에서 대상 edge를 선별한후 대상 edge를 검사하도록 테스트 패턴을 생성 한다. 이때 시작점에서부터 대상 edge까지 그래프 상에서 가장 단거리 path를 통과하도록 조건문 및 기술문 값을 조절한다. 이후 대상 edge에서 가장 단거리의 path를 통해서 그래프의 끝점까지 도달하도록 조건문 및 기술문의 값을 조절한다. 이때 그래프상의 path의 길이는 기술문의 개수에 따라 정해진다. 즉 path상에 있는 기술문의 개수가 그 path의 길이가 된다. 그래프 상에서 패턴 생성시 최단 거리를 통하는 이유는 loop와 같은 구문에서도 VHDL구문의 특별한 변형 없이 테스트패턴 생성이 가능하고 되도록 작은수의 기술문을 통과하도록 하여 오류데이터가 중간에 소실되는 경우를 막기 위해서 이다. 또한 자동으로 테스트패턴생성이 용이하다.

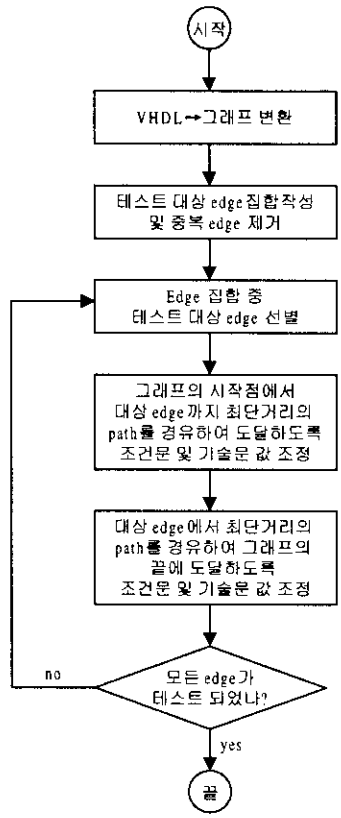


그림 3. 패턴생성 알고리듬

## V. 시뮬레이션

VHDL을 이용하여 설계된 회로에서 생성한 테스트패턴을 게이트레벨로 합성된 회로에 적용하여 고장 검출율을 계산하였다.(표 1) 그림 4는 VHDL을 이용하여 설계한 relational operator이며 그림 5는 그래프 변환 결과를 타나낸다. 그림 5에서 11개의 edge중 중복되는 edge를 제거한 테스트 대상 edge는 모두 5개이며 모두 8개의 패턴이 생성되었다. 그림 5에서 생성된 테스트 패턴을 게이트레벨로 합성된 회로(그림 6)에 적용하여 102개의 stuck-at-fault중 99개를 검출하여 97%의 고장 검출율을 얻었다. 또한 합성조건을 달리 하여 합성한 회로에 같은 패턴을 적용하여 100%의 고장 검출율을 얻을 수 있었다. 즉 VHDL상에서 합성 결과에 무관한 테스트 패턴을 생성 할 수가 있었다. 표 1에서 보듯이 다른 회로에서도 비교적 높은 고장 검출율을 얻을 수 있었다.

```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity relational_operators is
port( A,B:in          std_logic_vector(2 downto 0);
      Y1,Y2,Y3:out    std_logic;
      Y4:out          std_logic);
end relational_operators;

architecture logic of relational_operators is
begin
process(a,b)
begin
    Y1<=A<B;
    Y2<=A<=B;
    Y3<=A>B;
    if (A>=B) then
        Y4<='1';
    else
        Y4<='0';
    end if;
end process;
end logic;
```

그림 4. Relational Operator

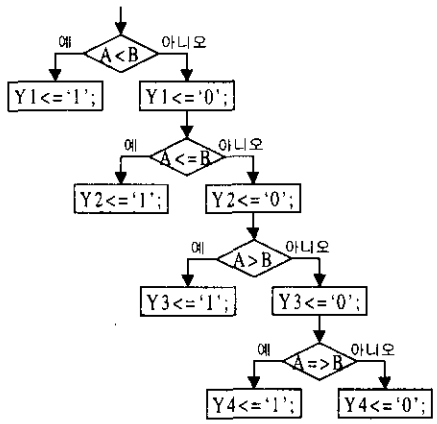


그림 5. 그래프 변환

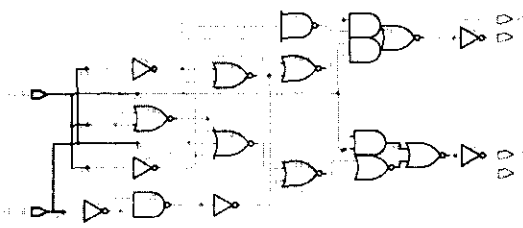

그림 6. 그림 4의 합성된 결과

| 회로 | 게이트수 | 노드수 | PI/PO | 패턴수 | 고장 검출율 |
|---|---|---|---|---|---|
| Relational Operator | 19 | 51 | 6/4 | 8 | 97% |
| Relational Operator 2 | 30 | 80 | 6/4 | 8 | 100% |
| 4Bit 비교기 | 29 | 93 | 8/3 | 10 | 92% |
| 8-3 encoder | 18 | 62 | 8/3 | 8 | 91% |

표 1 시뮬레이션 결과

## VI. 결론

본 논문에서는 기존의 테스트패턴 생성 방법의 단점을 보완하고 새로운 디자인 환경에 적합한 새로운 테스트패턴 생성 방법을 제안하였다. 제안된 방법은 기존의 방법보다 빠르고 쉽게 테스트패턴을 생성하였으며 또한 합성 결과에 무관하므로 기존의 방법에 비해 테스트패턴 생성 비용 및 시간을 줄일 수 있을 것으로 사료된다.

## Reference

[1] Hideo Fujiwara, *Logic Testing and Design for Testablity*, MIT Press, 1985

[2] Manzer Masud and Maddumage Karunarate, "Test Generation based on Synthesizable VHDL Description", Euro-DAC, September, 1994

[3] James R. Armstrong, "Hierarchical Test Generation : Where We Are, And Where We Should Be Going", Euro-DAC, September, 1994

[4] Mark Willard Johnson, *HIGH LEVEL TEST GENRATION USING SOFTWARE TESTING METRICS*, M.S. Thesis, University of Illinois ad Urbana-Champaign, 1994

[5] I.S 1076-1993, *IEEE Standard VHDL Language Reference Manual*. IEEE 1993

[6] A. Magdolen, J. Bexakova, E. Gramatova, M. Fischerove, "REGGEN-Test Pattern Generation on Register Transfer Level", Euro-DAC, September, 1994

[7] Douglas J. Smith, *HDL Chip Design*, Doone Publications, 1996

[8] Paul C. Jorgensen, *Software Testing*, CRC Press, 1995

[9] Abramovici, Breuer and Friedman, *Digital Systems Testing and Testable Design*, IEEE, 1990