하드웨어 고장 검출을 위한 행위레벨 설계에서의 테스트패턴 생성

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

High Level Test Generation in Behavioral Level Design for Hardware Faults Detection

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

Abstract

The high complexity of digital circuits has changed the digital circuits design methods from schematic-based to hardware description languages like VHDL, Verilog that make Hardware faults become more hard to detect. Thus test generation to detect hardware defects is very important part of the design. But most of the test generation methods are gate-level based.

In this paper new high-level test generation method to detect stuck-at-faults on gate level is described. This test generation method is independent of synthesis results and reduce the time and efforts for test generation

I. 서론

최근의 디지털회로 설계 방법의 경향은 스케메틱을 이용한 설계에서 VHDL[6]과 같은 하드웨어 언어를 이용한 설계로 옮겨가는 추세이다. 그 이유는 기존의 스케메틱 방법으로는 고기능 고집적의 복잡한 알고리듬을 갖는 회로를 설계하기에는 과다한 시간과 노력이 필요하므로 적합하지 않기 때문이다. 또한 회로가 복잡해지면서 회로의 제조공정상 고장 확률이 높아지게 되었으며 따라서 하드웨어 결함 검출을 위한 테스트패턴 생성이 회로설계에서 중요한 부분을 차지하게 되었다.[1][5]

기존의 테스트패턴 생성방법은 주로 게이트레벨에서 수행하거나[2][4] RTL레벨에서 테스트패턴을 생성[3]하는 것을 목적으로 하고 있다. 따라서 현재와 같은 하드웨어 언어로 설계하는 하이레벨설계 즉 행위레벨 환경에 적용하기에는 여러 가지 제약이 있다. 하이레벨설계 소스가 게이트레벨로 합성이 될 때까지는 테스트패턴을 생성할 수 없으며 합성조건에 따라서 합성 결과가 다르게 나타나기 때문에 테스트 패턴을 생성에 많은 시간과 노력이 필요하다. 따라서 하이레벨설계에서 하드웨어 결함을 검출할수 있는 테스트패턴을 생성하는 것이 필요하다.

VHDL 이나 Verilog와 같이 하드웨어설계 언어는 기존의 프로그래밍 언어와 유사하지만 하드웨어로 합성이 된다는 점이 다르다. 즉 하드웨어 표현언어에 의한 하드웨어정보를 담고 있으므로 하드웨어 결함 검출을 위한 테스트패턴 생성에 필요한 정보를 가지고 있다고 할 수 있다.

이 논문에서는 하이레벨설계와 합성된 하드웨어의 연관관계에 기초하여 하드웨어 결함을 검출할 수 있는 테스트패턴 생성 방법을 제시한다. 하드웨어 설계언어의 하나인 VHDL로 설계된 회로에서 테스트패턴을 생성한후 합성된 회로에 적용하여 결함 검출률을 확인한다.

II. VHDL설계 및 테스트패턴 생성 과정

II-1. VHDL 설계방법의 특성

VHDL을 이용한 설계는 복잡한 알고리듬을 비교적 쉽게 하드웨어로 구현할 수 있다. 이는 VHDL을 이용한 설계 방법이 기존의 프로그래밍언어 기술방법과 유사하기 때문이다. VHDL로 설계된 회로는 합성이라는 과정을 거치게 되는데 제약조건들(회로의 지연시간, 면적)에 따라 합성 결과가 다르게 나타난다. 즉 똑같은 VHDL 코드라도 합성결과는 여러 가지로 나타날 수 있다. 게이트레벨에 기초한 테스트패턴 생성 방법을 적용할 경우 합성결과에 따라 새로 패턴을 생성해야 하므로 설계단계의 시뮬레이션과 중복되게 패턴생성작업이 필요하여 시간과 비용면에서 불합리하다고 할 수 있다. 따라서 합성결과와 무관한 테스트패턴 방법이 요구된다.

II-2. VHDL 코드분석 및 그래프변환

하이레벨에서 테스트패턴 생성에 필요한 정보를 얻기 위해서 VHDL코드를 그래프로 변환할 수 있다. 그림 1은 VHDL에서 많이 사용되는 구문의 그래프 표현을 나타내고 있다. 그래프 표현에 있어서 사용되는 VHDL 구문은 크게 조건문(condition)과 기술문(statement)으로 나눌 수 있다. 그래프에서 조건문은 원으로 기술문은 사각형으로 나타내며 이들을 연결하는 edge와 edge로 연결된 path로 그래프가 구성된다. 기술문이 출력에

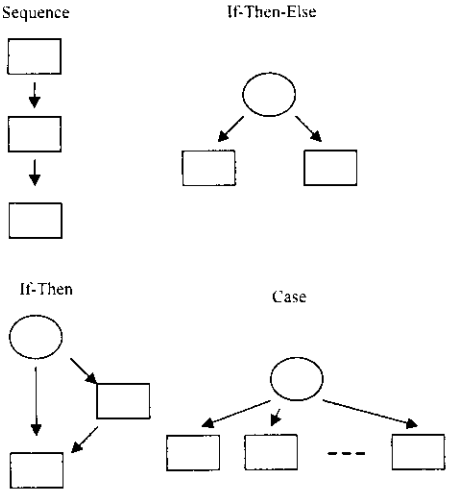

그림 1. VHDL구문의 그래프변환 예

관해 기술하고 있고 변수가 출력이 될 때 테스트패턴 생성을 위해 기술될 수 있는 경우에 따른 edge 및 기술부분이 그래프에 추가된다.(그림 2) 그래프가 완성되면 그래프를 이용하여 하드웨어 결함 검출을 위한 테스트 패턴을 생성하며 이 패턴은 합성 결과에 무관하다.

II-3. 하이레벨 고장 모델

테스트패턴 생성을 위한 그래프상의 고장 모델은 edge의 고장으로 한정한다. 즉 데이터가 흐르는 path상에 가상적인 고장이 발생하였다고 가정한 후 고장을 검출하기 위하여 그 path를 검사하는 테스트패턴을 형성한다. 따라서 여기서 가정한 path상의 고장은 물리적 고장이나 코드자체의 오류가 아닌 data가 흐르는 path상의 결함이다. 이 path를 테스트하기 위해 패턴을 형성하여 path에 data신호가 전달되게 함으로써 게이트레벨로 합성된 회로에서 data의 흐름을 방해하는 하드웨어 고장의 영향이 출력측으로 전달되록하여 하드웨어 결함을 검출하게 된다.

II-4. 패턴생성

고장 edge를 검사하기 위해 조건문과 기술문의 값들을 결정하여 테스트패턴을 형성한다. VHDL 예문과 그에 따른 그래프모델은 그림 3과 그림4에 나타나 있다. 그림에서 보듯이 모두 6개의 edge가 있으며 따라서 6개의 edge 고장이 있을 수 있다. 하지만 최상위 2개의 edge를 검사하는 것은 하위 4개의 edge를 검사하는 것에 포함되므로 최상위2개의 edge를 검사하기 위한 테스트패턴은 생성하지 않는다. 따라서 4개의 edge를 검사하기 위해 조건문과 기술문의 값들을 최상위부터 결정해 나가며, 값들이 모두 결정된 후 역으로 추적해 가면서 최종 테스트 패턴을 생성하게 된다.

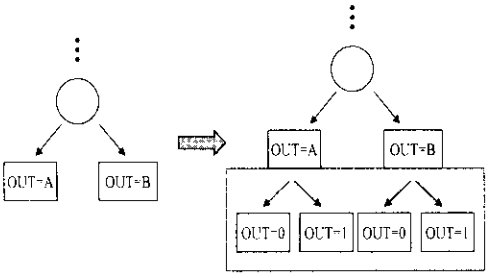

그림 2. 그래프에 추가된 edge 및 기술문

```
entity mux is
port(   sel     :in     std_logic;
        a,b     :in     std_logic;
        o       :out    std_logic);
end mux;

architecture logic of mux is
begin
if sel='0' then
        o<=a;
else
        o<=b;
end if;
end process;
end logic;
```

그림 3. VHDL 예제

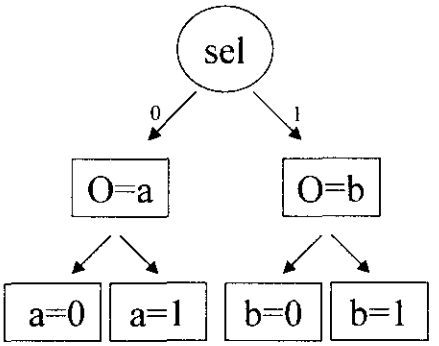

그림 6. 그림 3에 대한 변환된 그래프

III. 패턴생성 알고리듬

패턴생성 알고리듬은 그림 5와 같다. 우선 VHDL 코드를 그래프로 변환한다. 변환된 그래프의 모든 edge 중에서 패턴생성을 위해 검사해야 할 대상 edge를 결정한 후 입력에서부터 조건문 및 기술문의 값들을 결정해 나가며 출력에 도달할 때까지 이 과정을 반복한다. 출력에 도달 하면 역으로 검사했던 edge를 입력까지 찾아가며 생성된 패턴값들을 조합하여 테스트 패턴을 생성하게 된다. 이에 대한 예제는 그림 6과 그림 7에 나타내었다.

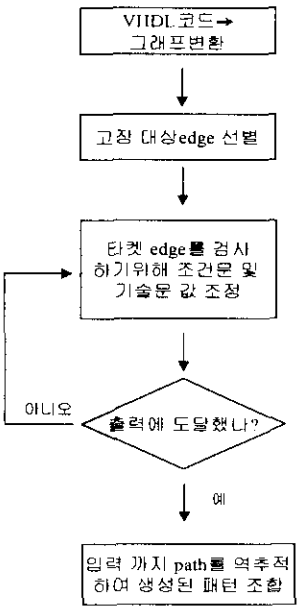

그림 7. 패턴생성 알고리듬

```
library IEEE;
use IEEE.STD_LOGIC_1164.all, IEEE.NUMERIC_STD.all;
entity ADD_DEC_CASE is
port(Address:in integer range 0 to 15;
    AddDec_0to3,AddDec_8to11,AddDec_12to15:out std_logic;
    AddDec_4to7:out unsigned(3 downto 0));
end entity ADD_DEC_CASE;

architecture logic of ADD_DEC_CASE is
begin
        process(Address)
        begin
                AddDec_0to3<='0';
                AddDec_4to7<=(others=>'0');
                AddDec_8to11<='0';
                AddDec_12to15<='0';

        case Address is
                when 0 to 3=>
                        AddDec_0to3<='1';
                when 4=>
                        AddDec_4to7(1)<='1';
                when 5=>
                        AddDec_4to7(2)<='1';
                when 6=>
                        AddDec_4to7(3)<='1';
                when 7=>
                        AddDec_4to7(3)<='1';
                when 8 to 11=>
                        AddDec_8to11<='1';
                when 12 to 15=>
                        AddDec_12to15<='1';
        end case;
end process;
end logic;
```

그림 6. 알고리듬 적용을 위한 VHDL 예제

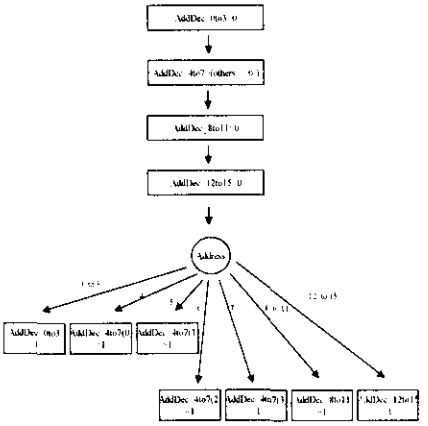

그림 7. 그림 6의 그래프 변환

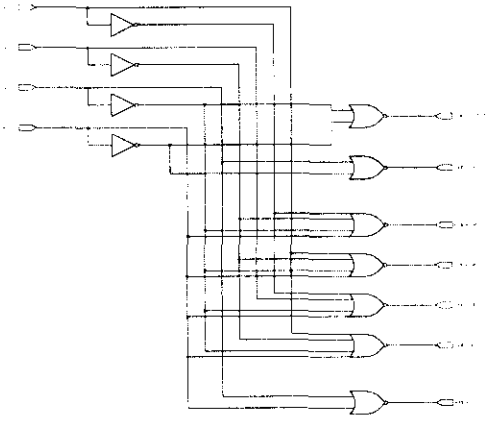

그림 8. 그림 6의 합성된 결과

위의 예제에서 그래프상의 edge는 모두 11개이며 중복되는 edge4개를 제외하면 테스트 대상 edge는 7개가 된다. 7개의 edge를 테스트하기 위해 테스트 패턴 7개가 생성되며 이 패턴을 그림 8의 게이트레벨로 합성된 회로에 적용하여 합성된 회로가 가지고 있는 80개의 stuck-at 고장 가운데 72개가 검출이 되어 90%의 fault coverage를 보였다.

IV. 시뮬레이션 결과

High-level에서 생성한 테스트 패턴을 게이트 레벨로 합성된 회로에 적용하였다. 이때 회로는 모두 조합회로이며 고장 모델은 각 노드마다 stuck-at-0, stuck-at-1으로 하여 C언어로 기술하여 실행하였다.

실험 결과에서 보듯이 하이레벨상에서 얻은 패턴이 게이트 레벨에서도 상당히 높은 고장 검출률을 나타내는 것을 알 수 있다.

회로	게이트수	노드수	고장수	고장검출	검출율
priorty encoder	15	54	108	97	90%
3-6 b-decoder	13	47	94	90	95%
address decoder	11	40	80	72	90%
3×8 decoder	14	56	112	112	100%
2bit wide 8-1mux	44	122	244	173	71%

V. 결론

본 논문에서는 하이레벨에서 하드웨어 고장을 검출할 수 있는 테스트패턴 생성방법을 기술하였다. 현재 디지털 회로설계에 많이 사용되고 있는 VHDL로 설계한 회로에서 테스트 패턴을 생성했으며 게이트레벨 합성 후 시뮬레이션을 통하여 결함검출률을 계산하였다. 하이레벨에서의 테스트패턴 생성은 설계 시간 및 비용을 감소시킬 수 있으며 합성결과에 무관함으로 합성결과에 따라 테스트패턴을 재생성할 필요가 없다. 따라서 기존의 테스트패턴 생성 방법과 비교할 때 보다 효율적인 방법이라 생각된다.

Reference

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

[2] Gabriel M. Silberman, "RIDDLE: A Foundation for Test Generation on a High-Level Design Description", *IEEE Transaction on CAD*, Vol.40, No.1, Janurary 1991

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

[4] Hideo Fujiwara, *Logic Testing and Design for Testability*, MIT Press, 1985

[5] Abramovici, Dreuer, Friedman, *Digital Systems Testing and Testable Design*, IEEE Press, 1990

[6] I.S 1076-1987, *IEEE Standard VHDL Language Reference Manual*. IEEE, 1998

[7] Sudhakar M. Reddy, Irith Pomeranz, and Seiji Kajihara, "Compact Test Set for High Defect Coverage", *IEEE Transaction on CAD*, Vol.16, No.8, August 1997