상위레벨에서의 VHDL에 의한 순차회로 모델링과 테스트생성

이 재 민† · 이 종 한††

요 약

본 논문은 상위레벨에서 VHDL을 사용하여 순차회로의 주요 구성요소인 플립플롭을 모델링하는 방법과 고장을 검출하기 위한 테스트생성 알고리듬을 제안한다.

RS, JK, D, T플립플롭은 데이터 흐름형을 이용하여 모델링한다. 칩레벨 모델의 기본 구조인 마이크로 오퍼레이션 시이퀀스를 하나 이상의 다른 마이크로 오퍼레이션 시이퀀스에 연결된 제어점으로 나타낸다. 다른 마이크로 오퍼레이션을 제한하고 있는 마이크로 오퍼레이션고장(FMOP고장)을 효과적으로 나타내기 위하여 고울트리의 개념을 사용하며 고울을 처리하기 위해서 휴리스틱 조건을 이용한다. FMOP나 제어점 고장(FCON)이 발생할때 고장 활성화, 경로 활성화 및 활성화된 경로를 유지하기 위한 명료화과정을 거쳐 테스트 패턴을 생성한다. 회로의 고장 모델은 VHDL의 ARCHITECTURE문의 데이터 흐름형으로 제한한다.

제안한 알고리듬을 C 언어로 실현하고 예제를 통하여 유효성을 확인한다.

High-level Modeling and Test Generation with VHDL for Sequential Circuits

Lee Jae-Min † · Lee Jong-han ††

ABSTRACT

In this paper, we propose a modeling method for the flip-flops and test generation algorithms to detect the faults in the sequential circuits using VHDL in the high-level design enviroment.

RS, JK, D and T flip-flops are modeled using data flow types. The sequence of micro-operation which is the basic structure of a chip-level leads to a control point where branching occurs to one of two micro-operation sequence. In order to model the fault of one micro-operation(FMOP) that perturb another micro-operation effectively, the concept of goal trees and some heuristic rules are used.

Given a faulty FMOP or fault of control point(FCON), a test pattern is generated by fault sensitization, path sensitization and determination of the input combinations that will justify the path sensitization. The fault models are restricted to the data flow model in the ARCHITECTURE statement of VHDL.

The proposed algorithm is implemented in the C language and its efficiency is confirmed by some examples.

1. 서 론

최근 디지탈시스템의 집적도와 복잡도가 급격히 증가되면서 기존의 하위레벨에서의 상향식(bottom-up) 설계방식은 시간이 많이 소비되어 비효율적이므로 기술이 간단하고 편리한 상위레벨에서의 하향식(top-down) 설계가 필요하게 되었다. VHDL은 이러한 하향식 설계에 적합한 하드웨어 설계언어로서 최근 많은 관심을 모으고 있으며 이에 관한 연구가 활발

† 정 회 원:관동대학교 전자공학과
†† 비 회 원:관동대학교 전자공학과
논문접수:1996년 1월 27일, 심사완료:1996년 4월 17일

히 이루어지고 있다[1-5].

게이트레벨 및 회로레벨에서 테스트를 생성하는데 있어서의 문제점은 테스트생성 알고리듬이 NP 완전(NP complete) 문제이기 때문에 테스트생성에 많은 시간을 소비하여 비효율적이고 또한 하위레벨의 정보로부터 상위레벨에서의 시스템 동작을 추정하기가 매우 어렵다는 것이다[6].

기존의 대표적인 상위레벨 테스트생성 방법중 버클리가 제안한 알고리듬[7]은 상위레벨에서 고장을 정의하고 고장을 활성화하기 위한 요구사항으로부터 회로를 모델링하였다.

그러나 이 방법은 VHDL로 모델링하는 기술의 초기단계이었기 때문에 알고리듬의 최적화가 이루어지지 않았고 필요한 VHDL의 구조를 첨가하지 못한 문제점을 갖고 있다[7]. 또 다른 테스트생성 방법으로서, 암스트롱은 상위레벨에서의 모델링을 중요시하면서 순차회로의 일부인 D 플립플롭을 모델링하고 이에 대한 알고리듬으로 버클리의 알고리듬을 이용한 테스트생성 방법을 제시하였다[8]. 이 방법은 버클리의 방법보다는 테스트생성 알고리듬이 간단하지만 고장모델링이 정확하지 못하며 표준 테스트 생성기(Standard Test Generator)에 테스트벡터의 제한조건을 설정하고 테스트패턴은 휴리스틱 테스트생성기(Heuristic Test Generator)로부터 생성하도록 하였다. 본 논문에서는 암스트롱의 방법에서 두가지로 분리되었던 테스트생성기를 하나로 통합하여 상위레벨에서 테스트시이퀀스를 얻기 위하여 칩레벨에서 순차회로의 핵심요소인 플립플롭을 데이터 흐름형으로 모델링하고 고장을 검출하기 위한 휴리스틱 테스트생성 알고리듬을 제시한다. 제시한 알고리듬은 상위레벨에서 모델링을 하고 고장 활성화와 경로 활성화를 위해 고장목록을 기술하며 이것으로 goal type을 정의하고 goal tree로 이루어지는 휴리스틱 알고리듬을 구성한다. 고장은 칩레벨의 기본구조인 마이크로오퍼레이션 시이퀀스(Micro-operation Sequence)에 연결되어 있는 다른 마이크로오퍼레이션 시이퀀스를 발생하는 제어점을 정의하여 모델링한다. 제어점고장(FCON)을 IF-THEN-ELSE 또는 CASE문으로 나타내고 하나의 마이크로오퍼레이션 고장에 의해 발생하는 다른 마이크로오퍼레이션 고장을 검출하기 위한 테스트생성을 위하여 휴리스틱 방법을 이용한다. 이 FMOP와 FCON고장모델에 대한 테스트생성 알고리듬은 고장을 활성화하는 과정과 경로를 활성화하는 과정 그리고 경로를 활성화시키기 위한 명료화 과정으로 이루어진다.

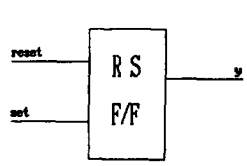
2. VHDL에 의한 순차회로의 고장모델링과 테스트생성

이 장에서는 순차회로의 주요 구성요소인 플립플롭의 고장모델링과 테스트생성에 대하여 기술한다. 플립플롭의 모델링은 ARCHITECTURE문에서 behavior형으로 모델링하는 방법과 data flow형으로 모델링하는 방법이 있다. 일반적으로 Behavior형 모델링에서는 순차적 기술문을 이용한다. 그러나 이러한 방법은 기술이 복잡하여 이용하기에 용이하지 않으므로 기술이 간단하고 편리한 data flow형으로 모델링 하는 것이 효과적이다.

게이트레벨의 대표적인 stuck-at고장 개념과 같이 순차회로의 핵심요소인 플립플롭의 고장을 칩레벨에서 data flow형으로 모델링하고 이에 대한 테스트생성 방법을 제시하고자 한다.

2-1. 플립플롭의 모델링

그림 1은 RS플립플롭과 VHDL모델링을 나타낸 것이다.

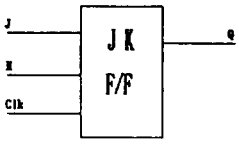



```
library IEEE;
USE IEEE.std_logic_1164.all;
ENTITY rs_latch IS
     PORT ( reset, set : in std_logic;
            y : out std_logic );
END rs_latch;
ARCHITECTURE chip OF rs_latch IS
 BEGIN
  PROCESS (reset, set)
  BEGIN
S1 : IF (reset = '0' ) THEN
S2 :   y <= '0';
S3 : ELSE IF ( set = '0') THEN
S4 :   y <= '1' ;
     END IF;
  END PROCESS;
END chip;
```

(그림 1) RS 플립플롭과 VHDL 모델링.
(Fig. 1) RS Flip-Flop and VHDL Modeling.

여기서 프로그램 내부의 'S'구 들은 data flow형태로 기술한 구이고 테스트생성을 위한 고장모델링이 된다.

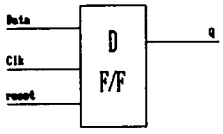
그림 2는 JK플립플롭과 VHDL 모델링을 나타낸 것이다.




```
library IEEE;
USE IEEE.std_logic_1164.all;
ENTITY jk IS
   PORT (J, K, CLK : in std_logic;
         Q : out STD_LOGIC );
END jk;
ARCHITECTURE chip OF jk is
  signal Q : STD_logic;
BEGIN
PROCESS (CLK)
  Variable JK : std_logic_vector ( 1 downto 0);
BEGIN
S1:    IF (clk'event and CLK = '1') THEN
S2:        JK := (J & K);
S3:        CASE JK IS
S4:        when "00" => Q <= 'Q';
S5:        when "01" => Q <= '0';
S6:        when "10" => Q <= '1';
S7:        when "11" => Q <= not (Q);
             when others => Q <= 'X';
           END CASE;
      END IF;
S8:     Q <= Q;
  END PROCESS;
 END chip;
```

(그림 2) JK 플립플롭과 VHDL모델링.
(Fig. 2) JK Flip-Flop and VHDL Modeling.

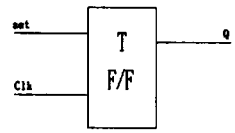
그림 3은 D 플립플롭과 VHDL 모델링을 나타낸 것이다.




```
library IEEE;
USE IEEE.std_logic_1164.all;
ENTITY dff IS
PORT (data, clk, reset : in std_logic ;
        q : out std_logic);
END dff;
ARCHITECTURE chip OF dff IS
  BEGIN
  PROCESS (clk) BEGIN
S1: IF (clk'event and clk = '1' ) THEN
S2:  IF (reset = '0') THEN
S3:     q <= '0';
     ELSE
S4   q <= data;
     END IF;
  END PROCESS;
END chip;
```

(그림 3) D 플립플롭과 VHDL모델링.
(Fig. 3) D Flip-Flop and VHDL Modeling.

그림 4는 T플립플롭과 VHDL 모델링을 나타낸 것이다.

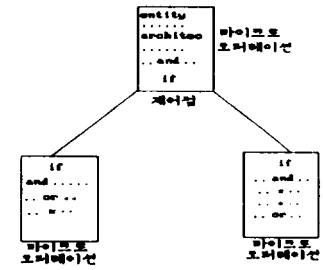



```
library IEEE;
USE IEEE. std_logic_1164.all;
ENTITY tff IS
    PORT (SET, CLK : in std_logic;
          Q : out std_logic );
END tff ;
ARCHITECTURE chip OF tff IS
    signal tmp_q : std_logic;
  BEGIN
    PROCESS (CLK, SET) BEGIN
S1:   IF (SET = '1') THEN
S2:     tmp_q <= '1' ;
S3:  ELSIF (CLK'event and CLK = '1') THEN
S4:  tmp_q <= not (tmp_q);
     END IF;
   END PROCESS;
S5:  Q <= tmp_q;
END chip;
```

(그림 4) T 플립플롭과 VHDL모델링.
(Fig. 4) T Flip-Flop and VHDL Modeling.

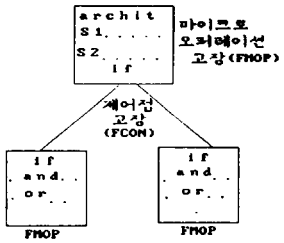
2-2. 플립플롭의 고장 모델링과 테스트생성 알고리듬

그림 5는 고장모델링을 위한 칩레벨 모델의 기본구조를 나타낸다.



(그림 5) 칩레벨 모델의 기본구조.
(Fig. 5) Basic structure of chip-level model.

칩레벨 모델의 기본구조는 마이크로오퍼레이션 시이퀀스에 연결되어 있는 다른 마이크로오퍼레이션 시이퀀스에 영향을 받는 제어점으로 구성된다. 여기서 마이크로오퍼레이션 시이퀀스를 마이크로오퍼레이션 고장 FMOP이라 하고 제어점을 제어점 고장 FCON이라 나타낸다. 그림 6은 칩레벨 고장모델을 나타낸 것이다.



(그림 6) 칩레벨 고장모델.
(Fig. 6) Chip-level fault model.

테스트생성 알고리듬을 위하여 이 고장모델은 칩레벨에서 순차회로의 핵심요소인 플립플롭에 대한 고장 부분을 테스트하기 위한 상위레벨에서의 모델링으로 이용한다.

고장을 활성화하는 조건으로부터 테스트입력을 구하기 위한 고장목록을 기술하고 이것을 고울형태로 정의한 다음 고울트리를 구성하는 휴리스틱 조건을 이용한다.

FMOP와 FCON 모델을 VHDL로 기술하여 D 알고리듬과 등가적으로 나타낼 수 있도록 하고 테스트생성은 고장을 활성화하는 과정과 경로를 활성화하는 과정 그리고 경로를 활성화시키기 위한 명료화 과정을 거쳐서 이루어진다.

알고리듬은 그림 3의 고장모델로부터 표 1에 나타낸 고울형태를 이용함으로써 테스트패턴을 생성한다. 또한 이 알고리듬에서는 순차회로 뿐만아니라 조합회로의 테스트생성을 위해 PROCESS문을 이용하며 고장모델에 대한 기술형태는 data flow형으로 기술한다.

표 1에 테스트생성 알고리듬을 위한 고울형태와 그 의미를 나타내었다.

〈표 1〉 고울 형태.
〈Table 1〉 Goal Type.

VIO	EXEC	OBSOBJ	DND
VIE	DNE	OBSEXPR	TR
	EXG	OBSEXEC	

표에서 각 고울형태들이 나타내는 의미는 다음과 같다.

(1) VIO : 어떤 시간에서 오브젝트(object)에 필요로 하는 값.
(2) VIE : 어떤 시간에서 표현문(expression)에 필요로 하는 값.
(3) EXEC : 어떤 시간에서 기술문을 실행함.
(4) DNE : 어떤 시간에서 기술문을 실행하지 않음.
(5) EXG : 주어진 기술문이 실행 되었다면 어떤시간에서 다른 기술문을 실행함.
(6) OBSOBJ : 어떤 시간에 대하여 오브젝트에서 관측된 값으로 0 또는 1의 값.
(7) OBSEXPR : 어떤 시간에 대하여 표현문에서 관측된 값으로 0 또는 1의 값.
(8) OBSEXEC : 어떤 시간에 대하여 기술문의 구에서 관측되어 실행하는 것으로 실행문에 대한 구가 0 또는 1의 값으로 나타낸다.
(9) DND : 주어진 시간에 대하여 대상에서 방해받지 않는 값.
(10) TR : 두 시간 사이의 관계를 기술함.

이들 고울의 해결방법을 적용하여 RS, JK, D, T 플립플롭의 테스트생성 알고리듬을 기술한다.

RS 래치의 테스트생성 알고리듬은 다음과 같다.

```
/* RS 래치의 테스트생성 알고리듬 */
[단계 1]
G1 : VIO at time t0 : reset = 1;
G4 : VIO at time t0 : set = 1;
[단계 2]
G2 : EXECUTE S2 at time t0;
G3 : OBSERVOBJ at time t0 for 0 or 1 value;
[단계 3]
G5 : EXECUTE S4 at time t0;
G6 : OBSERVOBJ at time t0 for 0 or 1 value;
```

고장을 활성화하기 위해 reset = 1, set = 1로 가정한다. G1은 G2을 먼저 실행해야 G3을 통하여 고장을 알 수 있다. 이 G3는 고장을 알 수 있기 때문에 주고울이 된다. G4는 앞서 기술한 바와 같은 실행으로써

다음과 같은 고울을 얻을 수 있고 G6은 주고울이 된다.

JK 플립플롭의 테스트생성 알고리듬은 다음과 같다.

/* JK 플립플롭의 테스트생성 알고리듬 */

[단계 1]

G1:OBSEXEC S3 at time t_0 ("01"=)Q(='0')=TRUE;

G2:OBSEXEC S4 at time t_0 ("10"=)Q(='1')=TRUE;

G3:OBSEXEC S5 at time t_0 ("11"=)Q(=not(Q))=TRUE;

G4:OBSEXEC S6 at time t_0 ("00"=)Q(=Q)=TRUE;

[단계 2]

G5:VIO S2 at time t_0;

G6:EXECUTE S1 at time t_0;

[단계 3]

G7:VIE at time t_0:(clk'event and clk='1')=TRUE;

G8:VIO at time t_0-1 clk=1;

[단계 4]

G9:VIO at time t_0-1 clk=0;

G10:EXECUTE S7 at time t_0;

CASE문 내부에있는 절들이 참(TRUE)인지 거짓(FALSE)인지 확인한다. 이 고울들은 클럭이 발생되어야만 실행되므로 S1절을 실행한다. 고울 G6은 먼저 G7과 G8이 실행되어야만 한다. 이들 고울은 고장을 위한 테스트입력이기 때문에 주고울이 된다.

이 G6과 G7, G8 고울들이 실행되어야만 한다.

D 플립플롭의 테스트생성 알고리듬은 다음과 같다.

/* D 플립플롭의 테스트생성 알고리듬 */

[단계 1]

G1:VIO at time t_0:Q=1;

G2:VIO at time t_0:Data=1;

[단계 2]

G3:EXECUTE S3 at time t0;

G4:VIE at time t_0-1:0=1;

[단계 3]

G5:EXECUTE S4 at time t_0;

G6:VIE at time t_0-1:D=1;

[단계 4]

G7:EXECUTE S1 at time t_0;

G8:VIE at time t_0:(clk'event and clk='1')=TRUE;

[단계 5]

G9:VIO at time t_0:clk=1;

G10:VIO at time t_0-1:clk=0;

고장을 위한 테스트를 위해서 Q=1과 Data=1 로 가정한다. G1은 G3와 G4로 분리된다. G3의 결과로써 고장을 위한 명료화가 이루어져야 하므로 G5는 주고울이 되고 G6은 다시 G7과 G8로 나누어진다. G8는 G9과 G10이 실행되어야 한다.

T 플립플롭의 테스트생성 알고리듬은 다음과 같다.

/* T 플립플롭의 테스트생성 알고리듬 */

[단계 1]

G1:VIO at time t_0:tmp_q=0;

G2:VIO at time t_0:set=1;

[단계 2]

G3:EXECUTE S2 at time t_0;

G4:EXECUTE S4 at time t_0;

G5:VIE at time t_{0-1}:set=0;

[단계 3]

G6:EXECUTE S3 at time t_0;

G10:EXECUTE S5 at time t_0;

[단계 4]

G7:VIE at time t_0:(clk'event and clk='1')=TRUE;

[단계 5]

G8:VIO at time t_0:clk=1;

G9:VIO at time t_{0-1}:clk=0;

고장의 테스트를 위해 tmp_q=0과 set=1로 놓으면 G2는 주고울이 되고 G1은 먼저 G3과 G4가 실행되어야 한다. 여기서 G4는 G10이 실행되어야 하고 이 G10은 주고울이 된다. G3은 다시 G6이 실행되어야 하고 G6은 G7이 실행되어야 하고 G7은 먼저 G8과 G9이 실행되어야 한다.

따라서 주고울은 G2, G8과 G9가 된다.

3. 실험 및 검토

제안한 테스트생성 알고리듬을 C 언어로 구현하였다. 구현한 알고리듬은 고장활성화, 경로활성화 및 경로활성화를 명료화하기 위한 입력결정 단계로 구성

된다. 앞절에서 기술한 모델링은 제안한 알고리듬으로부터 각 회로의 특성과 테스트패턴을 같이 생성하도록 프로그램하였다. 다음 표들은 앞절에서 기술한 플립플롭의 모델링을 근거로하여 제시한 테스트생성 알고리듬을 구현한 테스트생성기로부터 생성한 테스트패턴들이다.

다음 표 2는 제안한 알고리듬을 통하여 생성된 RS 플립플롭의 테스트시이퀀스이다.

〈표 2〉 RS 플립플롭의 테스트시이퀀스.
〈Table 2〉 RS flip-flop test pattern.

〈1〉:R=〉0	S=〉0	Q=〉〉1, 0
〈2〉:R=〉0	S=〉1	Q=〉〉0
〈3〉:R=〉0	S=〉0	Q=〉〉1

Faulty output 1 : T(R, S) = {(0 1), (0, 0)}
Faulty output 0 : T(R, S) = {(1 0), (0, 0)}

출력 1 고장일 때 이를 검출하기 위한 테스트시이퀀스는 {(0 1) , (0 0)}이다. 이 시이퀀스로 출력값이 0이 되는 〈1〉과 〈2〉 동작시 출력 1고장을 검출할 수 있다. (0 1)을 인가하면 정상일 때 항상 출력이 0으로 나타나지만 고장일 때는 1로 나타난다. (0 0)을 인가하면 정상일때는 출력값이 0으로 유지되지만 고장일 때는 1로 나타나 고장이 검출된다.

출력 0 고장일 때 이를 검출하기 위한 테스트시이퀀스는{(1 0) , (0 0)}이다. 이 시이퀀스로 정상출력값이 1이되는 〈1〉과 〈3〉동작시 출력 0고장을 검출할 수 있다. (1 0)을 인가하면 정상일 때 항상 출력이 1로 나타나지만 고장일때는 0으로 나타난다. 또 (0 0)을 인가하면 정상일때는 출력값이 1로 유지되지만 고장일 때는 0으로 나타나 고장이 검출된다.

다음 표 3는 JK 플립플롭의 테스트시이퀀스이다.

출력 1 고장일 때 이를 검출하기 위한 테스트시이퀀스는{(0 1 1), (0 0 1), (1 1 1)}이다. 이 시이퀀스로 정상출력값이 0이 되는 〈1〉, 〈2〉, 〈6〉, 〈8〉동작시 출력 1 고장을 검출할 수 있다. (0 1 1)을 인가하면 정상일 때 현재상태 Q(t)와 상관없이 항상 출력이 0으로 나타나지만 고장일때는 1로 나타나 〈2〉와 〈6〉이 검출되고 (0 0 1)을 인가하면 정상일때는 출력값이 0으로 유지되지만 고장일 때는 1로 나타나 〈1〉을 검출한다. 또한 (1 1 1)을 인가하면 정상일 때는 현재 상태가 토

〈표 3〉 JK 플립플롭의 테스트시이퀀스.
〈Table 3〉 JK flip-flop test pattern.

〈1〉:Q(t)=〉0	J=〉0	K=〉0	CLK=〉1	Q(t+1)=〉〉0
〈2〉:Q(t)=〉0	J=〉0	K=〉1	CLK=〉1	Q(t+1)=〉〉0
〈3〉:Q(t)=〉0	J=〉1	K=〉0	CLK=〉1	Q(t+1)=〉〉1
〈4〉:Q(t)=〉0	J=〉1	K=〉1	CLK=〉1	Q(t+1)=〉〉1
〈5〉:Q(t)=〉1	J=〉0	K=〉0	CLK=〉1	Q(t+1)=〉〉1
〈6〉:Q(t)=〉1	J=〉0	K=〉1	CLK=〉1	Q(t+1)=〉〉0
〈7〉:Q(t)=〉1	J=〉1	K=〉0	CLK=〉1	Q(t+1)=〉〉1
〈8〉:Q(t)=〉1	J=〉1	K=〉1	CLK=〉1	Q(t+1)=〉〉0

Faulty output 1 : T(J, K, CLK) = {(0 1 1), (0 0 1), (1 1 1)}
Faulty output 0 : T(J, K, CLK) = {(1 0 1), (0 0 1), (1 1 1)}

글(Toggle)되어 출력이 1이 되고 고장일때는 0으로 나타나 〈4〉의 고장이 검출된다.

출력 0 고장일 때 이를 검출하기 위한 테스트시이퀀스는{(1 0 1), (0 0 1), (1 1 1)}이다. 이 시이퀀스로 정상출력값이 1이되는 〈3〉, 〈4〉, 〈5〉, 〈7〉 동작시 출력 0 고장을 검출할 수 있다. (1 0 1)을 인가하면 정상일 때 현재상태 Q(t)와 상관없이 항상 출력이 1로 나타나지만 고장일때는 0으로 나타나 〈3〉, 〈7〉이 검출되고 (0 0 1)을 인가하면 정상일 때는 출력값이 0으로 유지되지만 고장일때는 1로 나타나 〈5〉를 검출할 수 있다. 또한 (1 1 1)을 인가하면 정상일 때 출력값이 현재상태 Q(t)의 보수가 되어 0이 되고 고장일때는 1로 나타나 〈8〉의 고장이 검출된다.

다음 표 4는 D 플립플롭의 테스트패턴이다.

〈표 4〉 D 플립플롭의 테스트패턴.
〈Table 4〉 D flip-flop test pattern.

Reset=〉1		
〈1〉:D=〉0	CLk=〉1	Q=〉〉0
〈2〉:D=〉1	CLk=〉1	Q=〉〉1

Faulty output 1 : T(D, CLK) = (0 1)
Faulty output 0 : T(D, CLK) = (1 1)

출력 1 고장일 때 이를 검출하기 위한 테스트패턴은 (0 1)이다. 이 패턴으로 정상출력값이 0이되는 〈1〉 동작시 출력 1 고장을 검출할 수 있다. (0 1)을 인가하면 정상일 때 출력이 0으로 나타 나지만 고장일때는 1로 나타나 검출된다.

출력 0 고장일 때 이를 검출하기 위한 테스트패턴

은 (1 0)이다. 이 패턴으로 정상 출력값이 1이되는 〈2〉 동작시 출력 0 고장을 검출할 수 있다. (1 0)을 인가하면 정상일 때 출력이 1로 나타나지만 고장일때는 0으로 나타나 검출된다.

다음 표 5는 T 플립플롭의 테스트시이퀀스이다.

〈표 5〉 T 플립플롭의 테스트시이퀀스.
〈Table 5〉 T flip-flop test pattern.

〈1〉:Q(t)=〉0	T=〉0	Q(t+1)=〉〉0
〈2〉:Q(t)=〉0	T=〉1	Q(t+1)=〉〉1
〈3〉:Q(t)=〉1	T=〉0	Q(t+1)=〉〉1
〈4〉:Q(t)=〉1	T=〉1	Q(t+1)=〉〉0

Faulty output 1 : T(T, CLK) = {(1 1), (1 1)}
Faulty output 0 : T(T, CLK) = {(1 1), (1 1)}

출력 0 또는 1 고장일 때 이를 검출하기 위한 테스트시이퀀스는 {(1 1), (1 1)}이다. 현재상태가 0일 때 (1 1)을 인가하면 출력값이 1로 토글되어야 정상이고 0이면 고장이다. 그런 다음 다시 (1 1)이 주어지면 출력값이 다시한번 토글되어 1이면 정상이고 0이면 고장이다. 현재 상태가 1일 때 (1 1)을 인가하면 출력값이 0으로 토글되어야 정상이고 1이면 고장이다. 다시(1 1)이 주어지면 출력값이 다시 한번 토글되어 1이면 정상이고 0이면 고장이다.

4. 결 론

본 논문에서는 상위레벨에서의 VHDL에 의한 순차회로 모델링과 테스트기법을 제안하였다.

순차회로의 핵심 구성요소인 플립플롭들의 고장모델링 및 테스트생성 알고리듬을 제안하였으며 알고리듬 구성을 위해 순차회로의 플립플롭을 VHDL의 ARCHITECTURE문에서 data flow형으로 기술하였고 상위레벨에서의 고장을 게이트레벨의 stuck-at고장개념으로 모델화하여 테스트시이퀀스를 생성할 수 있게 하였다. 이 알고리듬은 마이크로오퍼레이션 시이퀀스에서 다른 마이크로오퍼레이션에 간섭을 받을 때 이를 해결하기 위하여 휴리스틱방법을 이용하여 고울형태를 정의하고 고울트리를 구성함으로써 테스트시이퀀스를 생성하였다.

제안한 방법은 게이트레벨에서 테스트를 생성하는 방법보다 기술이 간단한 칩레벨의 data flow형태로 고장을 모델링함으로써 알고리듬의 복잡도를 낮출 수 있으며 고장을 모델링하기에 용이하므로 테스트시이퀀스를 쉽게 생성할 수 있다.

VHDL 논리회로 합성기에 의해 회로를 설계할 때 상위레벨에서도 게이트레벨과 같이 보다 완전한 합성결과를 얻을 수 있어야 하므로 이를 위해 VHDL 합성기에 대한 많은 연구가 요구되며 또한 상위레벨에서 테스트패턴을 효과적으로 생성하기 위하여 고장을 모델링 할 때 VHDL의 합성 프로그램과 공유할 수 있는 방법을 고려하는 것이 필요하다.

앞으로의 연구과제는 VHDL 환경하에서 임의의 순차회로에 적용할 수 있는 테스트 생성 알고리듬을 개발하는 것이다.

참 고 문 헌

[1] J. R. Armstrong, "Chip-Level Modeling and Simulation", Simulation, pp. 141-148, Oct. 1983.

[2] David R. Coelho, *The VHDL Handbook*, Kluwer Academic Publishers, pp. 3-5, 1990.

[3] J. R Armstrong, F. Gail, *Structure Logic Design with VHDL*, Prentice-Hall Inc, 1993.

[4] Douglas L. Perry, *VHDL*, McGraw-Hill Publishing Company, 1991.

[5] Z. Nawabi, VHDL: *Analysis and Modeling of Digital Systems*, McGraw-Hill, Inc., 1993.

[6] James R. Armstrong, "Hierarchical Test Generation: Where We Are, And Where We Should Be Going", *proc. EURO-DAC '93.*, IEEE Computer Society Press, Hamburg Germany, pp. 434-439, 1993.

[7] D. Barclay and J. R. Armstrong, "A Heuristic Chip-Level Test Generation Algorithm", *23rd Design Automation Conference*, pp. 257-262, June. 1986.

[8] J. R. Armstrong, *Chip Level Modeling with VHDL*, Prentice-Hall, Englewood Cliffs, N. J., 1989.

[9] V ronique Pla, Jean-Fran ois Santucci and Norbert Giambiasi, "On the Modeling and Testing

of VHDL Behavioral Descriptions of Sequential Circuits", *proc. EURO-DAC '93.*, IEEE Computer Society Press, Hamburg Germany, pp. 440-445, 1993.

[10] J. R. Armstrong, "Chip-Level Modeling of LSI Devices", *IEEE Trans. CAD of Integrated Circuits and Systems*, pp. 288-297, Oct. 1984.

[11] S. Rao, Pan, and J. R. Armstrong, "Hierarchical Test Generation For VHDL Behavioral Models", *Proceedings of the 1993 European Design Automation Conference*, February, 1993.

[12] *IEEE Standard VHDL Language Reference Manual*, IEEE Inc., New York, NY, March. 1988.

이 종 한

1994년 강릉대학교 전자공학과 졸업(공학사)
1996년 관동대학교 대학원 전자공학과 졸업(공학석사)
1994년~1995년 관동대학교 전자공학과 조교
1996년~현재 Donghwan Ind. Corp. 연구원
관심분야: VLSI CAD, VHDL

이 재 민

1979년 한양대학교 전자공학과 졸업(공학사)
1981년 한양대학교 대학원 전자공학과 졸업(공학석사)
1987년 한양대학교 대학원 전자공학과 졸업(공학박사)
1990년~1991년 University of Illinois at Urbana-Champaign Post-doc.(한국과학재단)
1986년~1988년 관동대학교 전자공학과 전임강사
1988년~1992년 관동대학교 전자공학과 조교수
1992년~현재 관동대학교 전자공학과(전자정보통신공학부) 부교수
1992년~1994년 관동대학교 전자계산소 소장
1995년~현재 대한전자공학회 강원지부 지부장
관심분야: VLSI CAD, 디지탈 시스템 테스팅, VHDL 다치논리회로 설계 및 테스팅