행위 수준의 VHDL 설계 검증 방법[1)]

박 승 규(朴 承 奎), 김 종 현(金 宗 賢), 서 영 호(徐 英 鎬), 김 동 욱(金 東 郁)
광운대학교 전자재료공학과
전화 : (02) 940-5167 / 팩스 : (02) 919-3940

Method for Verification of VHDL Behavioral-level Design

Seung-Kyu Park, Jong-Hyeon Kim, Young-Ho Seo, Dong-Wook Kim
Electronic Material Engineering, Kwangwoon University
E-mail : vlsicad@daisy.kwangwoon.ac.kr

Abstract

This paper proposed a method to detect and locate coding errors in HDL behavioral descriptions (designs). The target coding errors are the ones that the compiler cannot find out. As the method, this paper used verification pattern generation method. Thus, an algorithm to generate the verification patterns was proposed, in which the pattern generation is performed by a path-searching method. Various example designs were applied to this algorithm to verify the correctness and effectiveness of the proposed method.

I. 서론

최근의 설계방법에 대한 동향은 스케메틱 설계 방법에서 VHDL 또는 Verilog-HDL등의 HDL(Hardware Description Language)을 사용하는 설계 방법으로 변화하고 있다. 이러한 HDL 코드는 합성(Synthesis)이라는 단계를 거쳐 게이트 수준 등의 하위 수준의 회로로 변환되어 사용된다. 특히 회로가 커지고 IC의 기능이 복잡해짐에 따라 행위적(기능적) 수준에서 HDL을 이용한 설계방법이 각광받고 있다.

회로의 크기가 커지며, 기능이 복잡해지는 추세에 따라 HDL 설계의 크기 또한 증가한다. 이러한 이유에서 HDL 설계 코드의 코딩 오류를 찾아내고 고치는 시간 또한 급격히 증가하게 되었다. 이러한 설계 검증(design verification)은 일반적으로 HDL 코드를 컴파일(compile)한 후 동작적 시뮬레이션을 수행함으로써 이루어진다. 최근의 formal verification 기술은 이런 설계 검증을 위한 연구 중 하나 이다. 그러나 지금까지 제안된 formal verification은 코딩 자체에서의 오류를 찾는 것이 아니며, 특별한 형태로 기술된 사양이 있어야 코딩 방법에서의 오류를 찾아낼 수 있다.

본 논문에서는 행위 수준의 HDL 설계의 코딩 에러 중 컴파일러가 발견하지 못하는 코딩 에러에 초점을 맞추었다. 본 논문에서는 이와 같은 코딩에러를 찾아내고 또한 위치를 알아내어(detect and locate : D&L), 설계자의 설계 검증을 보다 용이하게 수행하는 방법을 제안한다. 본 논문에서는 현존의 HDL 중 VHDL을 대상으로 하였다. 또한 gold-unit 이라는 설계사양을 필요로 하는데, 이것은 C-언어 또는 합성되지 않는 HDL 코드 등으로 기술되었다고 가정한다. 따라서 gold-unit은 주입력(primary inputs : PIs)에 의한 결과 값이 주출력(primary outputs : POs)에 나타날 수 있다고 가정한다. 본 논문에서는 코딩에러의 D&L을 수행하기 위하여 검증패턴을 생성하는 방법을 사용한다.

II. 오류의 검출

1) 본 연구 1999~2001 한국 학술 진흥 재단의 연구비에 의해 수행되었음.

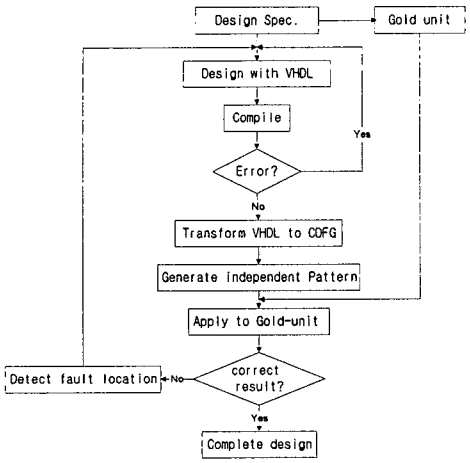
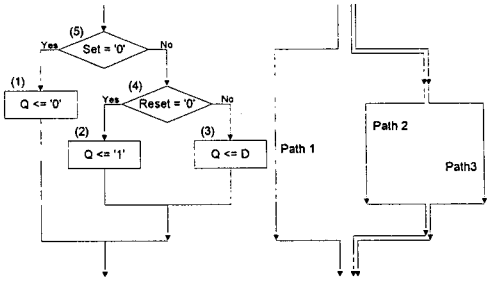


그림 1. 코딩오류 검출 및 위치부각 순서

그림 1은 본 논문에서 제안한 코딩오류를 D&L하는 방법을 간략하게 나타내고 있다. 코딩오류를 D&L하기 위해, 코딩오류의 유무를 판별하는 패턴생성 방법이 필요하다. 이를 위해 먼저 VHDL 코드를 CDFG (Control and Data Flow Graph)로 변환시킨다. 그림 2의 (b)는 그림 2의 (a)의 VHDL code를 CDFG로 변환시킨 것이다. 이 CDFG에서 오류를 가정하고 그 오류를 검출하기 위한 패턴을 생성한다. 표 1은 VHDL을 이용하여 회로 설계 시 생길 수 있는 코딩오류를 나타내고 있다. 행위레벨에서 회로 설계 시 생길 수 있는 오류는 조건을 변화시키는 것에 관계된 조건오류와 값을 할당하는 값과 관계된 할당오류로 크게 두 가지로 나눌 수 있으며, 본 논문에서는 행위레벨에서의 코딩오류만을 고려한다. 제안한 방법은 설계에서 단일오류 발생을 가정하여, 두개 이상의 오류에 대해서는 향후에 연구할 계획이다.

```
entity example is
port ( set, reset, D : in bit;   Q : out bit);
end example;
architecture behave of example is
begin
 process
  begin
  if set = '0' then
   Q <= '0';
  else if reset = '0' then   Q <= '1';
     else   Q <= D ;
     end if;
  end if;
 end process;
end behave;
```

(a) VHDL 코딩



(b) CDFG (c) Path

그림 2. 행위수준에서의 VHDL 코딩 예제

```
1   Procedure: Gen_Pattern{ }
2   Input: C/DFG, Error list
3   Output: A set of verificadion patterns
4   Begin
5       For all errors in error list
6       loop
7       Assume error;
8           Backward trace from error site to START;
9               Find signal values to activate the path;
10                  Select the values for Pis among the signals;
11          Forward trace from error site to END;
12              Find signal values to activate the path;
13              If PO changes its vlare, then
14                  Take it as a verification pattern;
15                  Calculate output vlaue;
16              Else
17                  Goto 11;
18      End loop
19  End
```

그림 3. 설계검증 패턴생성 알고리듬

표 1. 고려된 오류의 종류

Class	Keyword	Error
Condition Statement	if	Error in assignment for condition
		Error in priority order
		Error in 'event' statement
	case	Error in assignment for condition
		Error in target statement by condition
Assignment Statement	Inside or outside of condition	Error in using operator
		Assignment error
Repeat Statement	loop	Error in the # of repetition
structural Statement	component	Error in port mapping
	generate	Error in the # of repetition
Sequential Statement	process	Error in sensitivity list

그림 3은 코딩오류를 검출하기 위한 패턴생성 알고리듬을 나타낸다. 변환된 CDFG에서 오류를 검출하기 위해 그림 2(c)와 같이 CDFG의 START노드에서 END 노드까지 대상오류를 포함한 경로를 찾아내면서 패턴을 생성한다. 즉, 오류를 가정한 노드에서 역방향 추적

알고리듬을 사용하여 START 방향으로 경로를 찾고, 정방향 추적 알고리듬을 사용하여 END 방향으로의 경로를 찾아내며 패턴을 생성한다. 이때 코딩오류가 있을 경우 gold unit의 응답과 설계된 회로의 응답이 다르도록 패턴을 생성하며 생성한 패턴에 대한 gold unit과 설계된 회로의 응답차이를 이용하여 코딩오류를 D&L하게 된다.

표 2. 그림 2.의 검증패턴

Pattern #	PIs			PO	Gold Unit Response for Error					Path #
	SET	RESET	D	Q	(1)	(2)	(3)	(4)	(5)	
A	0	1	0	0	1	0	0	0	1	1
B	1	0	1	1	1	0	1	1	0	2
C	1	1	1	1	1	1	0	1	0	3
D	1	1	0	0	0	0	1	1	0	3
E	0	0	0	0	1	0	0	0	1	1

그림 2(a)의 예제에서, 5개의 오류를 가정하고 5개의 패턴을 생성하였다. 생성된 패턴은 표 2에 나타내었다. 표 2에서 경로는 그림 2(c)에서 정의된 경로를 의미한다. 그림 3은 코딩오류를 검출하기 위한 패턴생성 알고리듬을 나타낸다.

III. 오류의 위치 판별

표 2에서 보듯이 그림 3의 패턴생성 알고리듬을 적용하여 생성한 패턴으로 모든 에러를 검출할 수 있음을 알 수 있다. 그러나 빠른 코딩오류의 수정을 위해서는 코딩오류의 발생과 함께 오류의 위치를 신속하게 파악할 수 있어야 한다. 이를 위해서는 각 오류에 대해 생성한 패턴의 응답이 독립적이어야 한다. 즉, 각각의 에러에 대한 모든 응답이 서로 다르다면, 오류 발생 위치를 알 수 있다. 표 2에서, 1번 오류와 5번 오류를 검출하기 위한 패턴인 A와 E에 대한 출력 값이 같다. 즉, A와 E의 패턴으로 1번과 5번 오류를 검출할 수는 있지만, 1번 오류인지, 5번 오류인지를 판별을 할 수 없다. 따라서 오류의 위치 판별을 위한 방법이 필요하다.

그림 2의 각 오류에 대하여 생성한 패턴이 다른 오류도 검출할 수 있는지를 나타낸 결과를 행렬 R로 표현하면 다음과 같다.

$$R=\begin{bmatrix} A1 & A2 & A3 & \dots & An \\ B1 & B2 & B3 & \dots & Bn \\ C1 & C2 & C3 & \dots & Cn \\ . & . & . & . & . \\ N1 & N2 & N3 & \dots & Nn \end{bmatrix}=\begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

여기서 Ki는 패턴 K에서의 오류 i의 반응을 나타낸다. 오류를 가정하고 패턴을 적용시켰을 때의 결과값이 gold unit과 다른 경우를 1로 표시하고, 결과 값이 같은 경우를 0으로 표시한다. 즉, 식 (1)에서 1은 오류를 검출함을 나타내고, 0은 오류를 검출하지 못함을 나타낸다. 만약, 행렬의 한 행에서 1이 하나만 있다면, 그 패턴은 1개의 오류만을 검출할 수 있음을 의미하며 또한 오류위치를 알 수 있는 독립적인 패턴이다.

한 패턴이 2개의 오류에 반응할 경우 행렬 R의 대각선(diagonal)을 기준으로 대칭이 아닌 패턴 2개로 어떤 오류인지 검출 및 오류위치를 판별 할 수 있다. 예를 들어, 오류 (3)과 오류(4)를 행렬로 나타내면 식 (2)와 같이 된다.

$$R_{3,4}=\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \quad (2)$$

이러한 경우 두 패턴으로 두 오류를 검출할 수 있음과 동시에 오류위치 또한 파악할 수 있다. 그러나, 오류 (1)과 오류 (5)를 행렬로 나타내면 식 (3)과 같으며,

$$R_{1,5}=\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad (3)$$

이 식으로는 오류위치를 파악 할 수 없다. 이러한 경우 다른 패턴을 적용하여 오류 위치를 파악한다. 코딩오류의 위치를 파악하기 위한 방법은 다음과 같다.

[1] 반응 값의 행렬에서, 어떤 행과 열에서 오직 하나만의 1이 있는 경우 그 1을 기준으로 그 행과 열을 삭제한다.

[2] 전체 행렬에서 두 오류로 이루어진 한 쌍에 대한 부분 행렬을 구한다. 이에서 대각선을 기준으로 비대칭이거나, 1이 오직 대각선에만 있는 경우, 그 두 가지 오류와 패턴을 D&L 할 수 있는 것으로 표시한 후 삭제한다.

[3] 부분 행렬에서 모든 행과 열이 대칭적인 경우 구별할 수 있는 행을 삽입하여 D&L을 할 수 있도록 한다.

[3]의 예제는 식 (1)의 첫 번째와 다섯 번째 행을 구별하기 위하여 3번째 행을 삽입하여 오류위치를 알 수 있다.

IV. 효과적인 패턴생성

설계한 회로의 크기가 증가할수록 코딩오류의 D&L을 위한 패턴수가 증가하며, 따라서 코딩 오류를 D&L하기 위한 시간 또한 증가한다. 이러한 이유에서 효율적인 패턴을 생성하는 방법이 필요하며 그 방법을 그림 4에 나타내었다. 즉, 조건문의 오류를 대상으로 패턴을 생성한다. 따라서 패턴수가 감소한다. 이 패턴을 오류 목록에 적용시켜 검출하지 못하는 할당 오류가 있으면 패턴을 더 생성한다. 모든 오류를 검출할 수 있는 패턴을 생성한 후, 오류위치를 파악할 수 있도록 추가적으로 패턴을 생성한다. 이 방법으로 패턴을 생성한 결과 패턴의 수가 줄어들었으며 대부분의 할당오류를 검출할 수 있었다.

그림 2의 예제에 이 방법으로 패턴을 생성한 경우의 결과는 표 4에 나타내었다. 즉, 5개 패턴이 3개 패턴으로 줄었다. 이 방법으로 표 3의 예제에 적용한 결과를 표 5에 나타내었다. 그 결과로 패턴의 개수는 줄었지만 D&L은 모두 가능하였다.

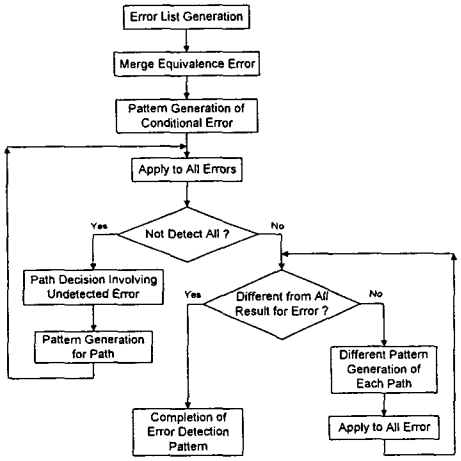


그림 4. 효과적인 패턴 생성 알고리듬

표 3. 여러 회로에 적용 결과

coding	# of Error	# of pattern	Detection (%)	Location (%)
3-6 decoder	16	10	100	100
4 bit comparator	8	5	100	100
8 bit register	12	10	100	100
74LS163 4-bit counter	19	22	100	100

표 4. 효과적 패턴생성 알고리듬을 적용한 결과

Pattern #	PIs			PO	Gold Unit Response for Error					Path #
	SET	RESET	D	Q	(1)	(2)	(3)	(4)	(5)	
A	0	1	0	0	1	0	0	0	1	1
B	1	0	0	1	1	0	1	0	0	2
C	1	1	0	0	1	0	1	1	0	3

표 5. 효과적 패턴생성 방법의 여러 회로 적용 결과

coding	# of Error	# of pattern	Detection (%)	Location (%)
3-6 decoder	16	9	100	100
4 bit comparator	8	4	100	100
8 bit register	12	6	100	100
74LS163 4-bit counter	19	16	100	100

V. 실험 결과 및 고찰

제안된 방법을 적용한 몇 가지 실험결과를 표 3에 나타내었다. VHDL 설계에서 발생할 수 있다고 생각되는 모든 설계 상의 오류를 검출하고 또한, 오류위치까지 파악할 수 있었다.

제안한 방법은 설계오류를 검출하고, 오류위치를 파악하는데 지금까지의 수동적인 방법보다 더 효과적인 방법이다. 이 방법에 의해 설계 시간을 훨씬 단축시킬 수 있으며, 따라서 TTM(Time-To-Market)을 단축시켜 경쟁력을 강화할 수 있을 것으로 기대된다.

참고문헌

[1] D. Perry, *VHDL* , McGraw-Hill, 1998

[2] Z. Navabi, *VHDL Analysis and Modeling of Digital Systems* , McGraw-Hill, 1998

[3] H. Fujiwara, *Logic Testing and Design for Testability*, MIT Press, 1985

[4] D. Smith, *HDL Chip Design*, Doone Pub., 1996

[5] K. McMillan, *Symbolic Model Checking*, Kluwer Academic Publishers, 1993

[6] P. Jorgensen, *Software Testing*, CRC press, 1995.

[7] D. Knapp, *Behavioral synthesis*, Prentice-Hall, 1996

[8] Abramovici, Breuer, Friedman, *Digital Systems Testing and Testable Design*, AT&T , 1990

[9] A. Ghosh, S. Devadas, A. Newton, *Sequential Logic Testing and Verification*, Kluwer Academic Publishers, 1992

[10] M. Masud, M. Karunaratne, *Test Generation based on Synthesizable VHDL Descriptions*, EURO-DAC 93, 446-451, 1993

[11] J.Armstrong, *Hierachical Test Generation : Where We Are, And Where We Should Be Going*, International Test Conference 1993, 434-439, 1993