VHDL 행위 레벨 설계 검증

윤성욱, 김종현, 박승규, 김동욱
광운대학교 전자재료공학과
139-701 서울시 노원구 월계동 447-1
e-mail : rasicon@netian.com

VHDL Behavioral-level Design Verification from Behavioral VHDL

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

Abstract

Hardware Formal verification involves the use of analytical techniques to prove that the implementation of a system conforms to the specification. The specification could be a set of properties that the system must have or it could be an alternative representation of the system behavior. We can represent our behavioral specification to be written in VHDL coding.

In this paper, we proposed a new hardware design verification method. For this method, we assumed that a verification pattern already exists and try to make an algorithm to find a place where a design error occurred. This method uses an hierarchical approach by making control flow graph(CFG) hierarchically. From the simulation, this method was turned out to be very effective that all the assumed design errors could be detected.

Ⅰ. 서론

최근 반도체 기술의 발달로 회로가 고속화, 정밀화, 그리고 대형화되고 있다 따라서 schematic capturing 에 의한 종래의 설계방법은 과다한 설계 시간을 요구하여 생산비를 크게 증가시키고 있다. 이에 대해 설계 시간을 단축하는 방법으로 최근에는 하드웨어 표현 언어 (Hardware Description Language: HDL)를 사용하여 설계를 보다 효율적으로 하고 있다.[1]

위의 방법으로 설계가 이루어지는 경우, 설계자는 마지막 단계로서 설계 주문자가 의도하는 특성(specification)[2]과 설계자의 설계 특성이 하드웨어(implementation)[3]로 실현된 동작을 비교하여 얼마나 정확히 설계자의 의도가 하드웨어로 구현되었는지를 알아보아야 하며 이를 검증하는 것이 설계 검증(design verification)이라 한다.[2][3][4]

설계와 HDL 코딩으로 설계된 회로를 실제 회로로 구현하는 툴들의 눈부신 발전에 힘입어 컴퓨터 상에서 설계할 때 설계 단계 자체 내에서 검증이 이루어지며, 일단 설계된 내용을 그대로 설계 검증단계가 필요 없을 정도로 정확하고 최적화 하여 실제 회로로 구현하는 툴들이 개발되고 있어서 설계자의 회로에 대한 다양과 실제 회로의 표현(implementation)간의 검증이 의미를 잃어가고 있다. 이에 본 논문은 HDL중 최근 많이 이용되고 있는 VHDL(VHSIC Hardware Description Language)로 설계할 때 실제 구현되는 회로에 대한 검증이 아니라, 설계후 컴파일과 시뮬레이션 과정에서 발견되지 못하는 코딩 에러를 설계사양(specification or gold unit)과 비교하여 코딩자체의 오류를 발견하는데 에 목적을 두고 있다

본 논문은 검증을 위한 입력패턴은 기 생성된 것으로 간주한다. 즉 본 논문은 설계검증을 위한 패턴 생성이 아닌, 그 패턴을 입력하였을 때, 거기에 해당하는 설계오류를 탐색하여 그 오류를 정정하는데 그 목적이 있는 것이다. 본 논문에서 사용하는 검증방법은 계층적 방법으로, 오류발생가능한 설계블럭을 먼저 검출하고 그 후 그 내부의 설계를 검사하는 것이다. 이 방법의 목적은 설계검증을 위해 최소한의 검색시간을 사용하기 위함이다. 이 방법에 사용되는 재원은

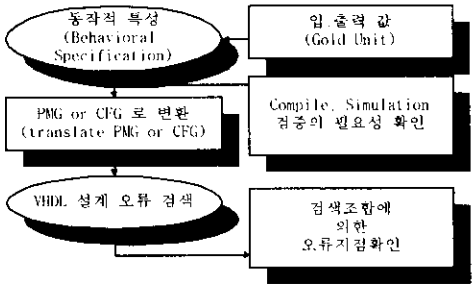


그림 1. 하드에어 표현 언어 검증 구조

PMG(Process Modeling Graph)와 CFG(Control Flow Graph)이다.

Ⅱ. VHDL 설계 검증 시스템의 구조

그림 1에 VHDL설계검증 시스템의 개략도를 나타내었다. VHDL에 의한 설계는 코딩된 설계사양을 기준으로 이루어진다. 이때 설계자는 설계자의 설계범위 및 방법을 결정하고 모든 동작의 기본이 되는 입·출력 값, 즉 골드 유닛(Gold Unit)의 값을 얻을 수 있다. 골드유닛의 응답은 검증과정에서 코딩의 오류와 비교대상이 되므로 매우 중요하며 따로 파일로 기록되어 사용되는 값이다.

다음으로 검증의 필요성을 알기 위해 원래의 코딩을 사용하여 시뮬레이션을 수행, 그 결과가 골드 유닛과 차이가 난다면 잘못된 오류의 위치를 파악하여 수정하기 위해 검증과정이 필요하게 되는 것이다.

이때 코딩자체에서 검증을 하는 것이 아니라 그래프적인 해석을 바탕으로 검증을 수행하게 되는데, 여기에 쓰이는 그래프에는 PMG(Process Modeling Graph)[6]와 CFG(Control Flow Graph)[7][8][9]가 있다 PMG는 다중 프로세스 구문에서 프로세스와 프로세스 간의 중간 값을 표현한 그래프이고 CFG는 프로세스를 보다 자세히 조건(condition)과 할당(assignment statement)을 표현한 그래프이다.

Ⅲ. 오류의 분류와 등가 오류

이 논문에서 타겟(target)으로 잡는 오류는 일반적인 논리적 오류가 아니고 코딩 내에서 if, case문 등을 작성해 가는 과정에서 일어나는 코딩 오류들이다. 즉, 컴파일 과정에서 발견되는 구문 오류나 논리적 오류를 제외한 설계자의 의도와 다르게 설계된 오류를 타겟으로 한다.

Ⅲ-1. 할당 상태 오류

```
if en = '0' then f <= 'a';
   else f <= 'b';
```

(a)

```
if en = '1' then f <= 'a';        if en = '0' then f <= 'b';
   else f <= 'b';                    else f <= 'a';
```

(b) (c)

(a) 골드 유닛(Gold Unit)
(b) 조건(Condition) 오류
(c) 할당 상태(Assignment statement) 오류

그림 2 골드유닛과 오류

이 오류는, VHDL 코딩에 의해서 설계자가 원하는 조건에 원하는 값을 할당(assignment)하여 원하는 입력에 대한 출력 값을 얻어야 하지만, 설계자의 실수로 할당문을 잘못 생성하여 발생하는 오류를 말한다. 예를 들어 그림 2 (c)를 보면 조건문은 골드유닛과 같으나 할당에 있어서 f <= a이어야 하는 것이 f <= b로 잘못되어 출력에는 설계자가 원하는 값이 나오지 않고 있음을 알 수 있다.

Ⅲ-2. 조건 오류

이 오류는 할당 상태오류와는 달리 할당이 제대로 이루어졌지만 그 할당에 대한 조건이 잘못되어 일어나는 오류이다 예를 들면 그림 2 b)와 같이 f <= a에 대한 조건이 en = '0' 이어야 하는데 en = '1'임을 알 수 있다

Ⅲ-3. 등가오류(equivalent)

위에서 살펴본 두 가지의 오류는 결과적으로는 같은 내용을 가지고 있는데 이는 잘못된 조건이나 할당이 수정되었을 때는 HDL코딩의 앞이나 뒤라는 차이뿐이지 원하는 조건에 할당되는 값들은 결과적으로 같음을 알 수 있다.

그러므로 오류 검색 시에는 두가지중 한가지를 선택하여 보다 효율적으로 오류 검색을 수행할 수 있다.

Ⅳ. 오류 검색 알고리듬

오류 검색 알고리듬을 그림 3에 흐름도를 이용하여 나타내었다. 이 흐름도에 의하면 처음으로 골드유닛과 코딩된 설계의 응답이 다른 경우 필요성을 확인한 다음 오류 검색을 시작한다. 다음으로는 PMG와 조건 오류로써 오류가 있는 프로세스를 찾고 더 세부적으로 들어가 프로세스 내부의 검색을 시작한다. 이때는 먼저 설정된 조건문들이 맞는다고 가정하고 설정된 CFG

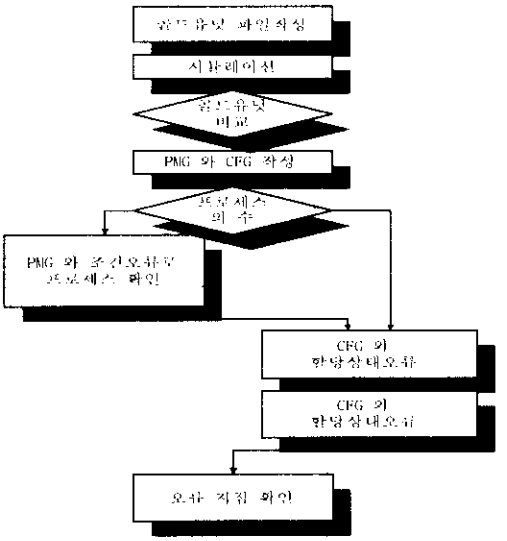


그림 3. 오류 검색 알고리듬

를 이용하여 각각의 우선 조건을 찾아 할당 오류위치를 찾게 되는 것이다. 이 검색에서 오류를 찾지 못하는 경우, 이번에는 할당문이 옳다고 가정하고 그에 해당하는 조건 오류를 CFG를 이용하여 검색한다.

위의 알고리듬을 VHDL의 행위레벨로 설계된 ALU에 적용을 시키는 데, 이에 대한 VHDL코딩과 PMG, CFG를 그림 4, 그림 5에 나타내었다.

먼저 완성된 코딩을 시뮬레이션을 통하여 골드유닛과 비교, 검증의 필요성을 따진다. 만약 검증의 필요성을 느꼈을 때는 그림에서와 같은 과정을 통해서 설계 검증을 수행한다.

오류검색이 필요한 경우 즉 골드유닛과 코딩된 설계의 응답이 다른 경우 다음과 같은 과정을 거치며 검증을 수행한다.

i) PMG에서 출력방향으로부터 조건(condition)오류를 가정하여 잘못된 값이 어느 프로세스로부터 나왔는지를 입력과 함께 검토한다.

여기서는 입력 (enable, enable2, con, clk) = (1, -, 10, 1) 에 대해 골드유닛에서는 '0' 값이 출력되야 되는데 그림 4의 코딩에서는 '1'이 출력되고 있다. 그러므로 검증이 필요하며 그림 5 (a)의 PMG에서 p3에 대해서 조건을 검토한다. 그러면 골드유닛의 clk값이 '1' 이므로 이 잘못된 값은 p1으로부터 나옴을 알 수 있고, 다음으로 p1의 CFG를 검색한다

ii) p1의 CFG를 검색하기 위해 그림 5 (b)를 이용한다. 이번에는 (a)에서 조건오류를 이용한데 반해 할당 상태오류를 이용한다.

```
1   Entity ALU is
2     port (enable, enable2, con, clk : in bit;
3           fout : out bit);
4   end ALU;
5
6   Architecture behavioral of ALU is
7     signal f_1 : bit ;
8     signal f_2 : bit ;
9   begin
10  p1 : process (con, enable)
11    begin
12      if enable = '1' then :
13        case con is
14          when "00" -> f_1 <= 0;
15          when "01" -> f_1 <= 1;
16          when "10" -> f_1 <= 1;     오류값
17         (when "10" -> f_1 <= 0;) 정상값
18          when "11" -> f_1 <= 1;
19        end case ;
20      else f_1 <= 0;  end if ;
21  end process p1 ;
22
23  p2 : process (enable2)
24    begin
25      if enable2 = '1' then f_2 <= 1 ;
26      else f_2 <= 0 ;
27  end process p2 ;
28
29  p3 : process (clk)
30    begin
31      if clk = '1' then fout <= f_1 ;
32      else fout <= f_2 ;
33      end if ;
34  end process p3 ;
35  end behavioral;
```

그림3. ALU를 위한 HDL 코딩 예

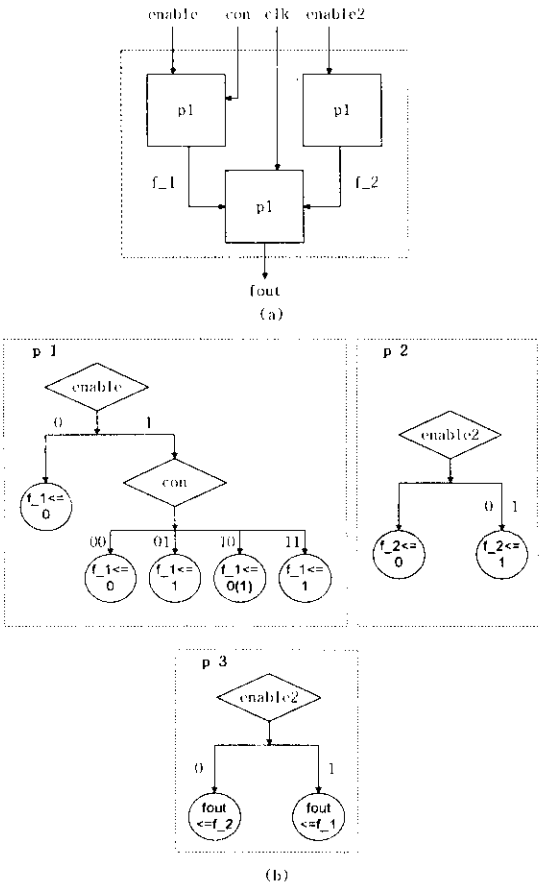
이유는 CFG에서는 출력에 대한 값보다 입력에서부터 원하는 조건을 비교해가며 우선 조건을 계층적으로 검색하는 것이 효과적이기 때문에 할당 상태 오류를 가정하여 검색한다.

검색 과정은 먼저 골드유닛의 입력이 (1, -, 10, 1)이고 여기서 조건이 enable <= '1', con <= "10"이므로 그래프에서 먼저 enable <= '1' 조건을 찾고 다음으로 con <= '10'을 찾아 검색을 수행할 수 있다.

iii) 마지막으로 CFG에서 검색을 하여 f_1의 값이 '0' 이 '1'로 잘못 할당되어 있다는 것을 찾을 수 있고 그림 4 의 코딩 라인 16 에서 잘못된 오류가 있음을 알 수 있는 것이다.

V. 오류검색 실험 결과

실험 적용은 if, case문만으로 이루어진 combinational 회로에 대해서만 검증을 수행하였다. 오류는 단일오류만을 고려하였으며, 조건과 할당 상태 오류를 고려하였다. case문에서는 중복 할당이 가능한 것도 에러로 간주 하여 포함시켰다. 본 논문에서는 표 1의 4가지회로에 대해 실험하였으며, 각 회로의 특성과 본 논문에서 제안된 알고리듬의 적용 결과를 같이 나타내었다. 4_bit ALU와 resource sharer에서 오류검출률이 낮은 이유는, 본 논문에서는 단일 오류를 가정하여 발생된 오류를 검출하고 그 오류를 정정하면 골드유닛과 동일한 결과가 도출 되는 것을 검색의 타



(a) PMG(Process Modeling Graph)
(b) CFG(Control Flow Graph)

그림 4. PMG 와 CFG

갯으로 정하였으나, 회로에서 두 개의 할당문이 서로 위치가 바뀐 등의 경우 단일 검색으로 두 오류를 검색하지 못하므로 이러한 경우를 오류가 검출되지 않은 것으로 간주 했기 때문이다. 이러한 검출률은 타겟으로 잡은 오류의 종류에 따라 큰 차이를 보이고 있으며, 본 연구에 이은 차후의 연구에서 이러한 다중 오류를 검색하는 방법을 모색하고자 한다.

name	코딩 특성			설계 오류 특성	
	process	if문	case문	error수	coverage
2_1 mux	1	2	-	6	100%
resource sharer	3	2	1	25	52%
alu(예제)	3	3	1	30	80%
8_decoder	4	12	-	32	100%

표 1 알고리듬에 적용 결과

VI. 결론

본 논문에서는 행위 레벨 VHDL코딩 자체의 오류에 대한 검증방법을 제안하였다. 그리고 본 논문에서는 단일 오류를 가정하였으며, 검증방법으로는 PMG, CFG를 사용한 계층적 검색 방법을 택하였다. 제안된 알고리듬을 실제 VHDL행위 레벨설계에 적용한 결과, if문에 대해서는 모든 오류를 검출하는 것을 볼 수 있었으나, case 문에서는 중복 할당 때문에 검출률이 낮아지고, 골드유닛과의 입출력이 같아 질때까지 각 패턴이 단일 오류를 검색하는 것으로 가정하여 검증을 수행하므로 검출률이 낮아짐을 알 수 있었다. 그러나, 제안된 방법이 단일 오류를 가정한 결과로도 오류가 발생한 지점을 찾아낼수 있으므로, 비록 단일 오류를 정정하여 골드유닛과 동일한 결과를 얻을 수 없는 경우라도 오류의 종류에 따라 설계자가 그 오류 및 그와 관련된 오류를 판단할수 있게 하므로, 설계 오류를 단시간 내에 검색하는데는 상당히 효과적이라 할수 있다. .

참 고 문 헌

[1] 박 현 철, "VHDL 회로 설계와 응용", 한성출판사, 1995
[2] T. Kam and P. A. Subrahmanyam, "Comparing Layouts with HDL Models: A Formal Verification Tehcnique", IEEE trans. on CAD, Vol. 14, pp.503-509, April, 1995
[3] M. C. McFarland, "Formal Verification of Sequential Hardware: A Tutorial', IEEE Trans, on CAD, Vol. 12, No. 5, pp.633-654, May, 1993
[4] Alan J. Hu, "Formal Hardware Verification with BDDs : An introduction", IEEE PACRIM 1997
[5] Y. V. Hoskote, J. A. Abraham, "Automatic Verification of implementations of Large Circuits Against HDL Specifications", IEEE Trans. on CAD, Vol. 16, No. 3, pp.217-228, March 1997
[6] J. R. Armstrong, F. G. Gray, "Structured Logic Design with VHDL", Prentice Hall, Englewood Cliffs, N.J., May 1993, ISBN 0-13-885206-1
[7] S. Hayati, A.Parker and J. Granacki, "Representation of Control and Timing Behavior with Applications to Interface Synthesis", ICCD, Proceedings of the International Conference on Computer design, pp. 382-387, 1988
[8] M. J. McFarland, "Value Trace", Carnegie-Mellon University Internal Report, Pittsburgh, PA, 1978
[9] H. Trickey, "Flamel: A High-Level Hardware Compiler" IEEE Trans. CAD, Vol. CAD-6, No.2, pp.259-269, March 1987
[10] R. Vemuri and R. Kalyanaraman, "Generation of Design Verification Tests from Behavioral VHDL Programs Using Path Enumeration and Constraint Programming", IEEE Trans, VLSI, Vol. 3, No. 2,5 pp.201-214, June, 1995