계층성을 이용한 VHDL행위 수준에서의 설계 오류 탐색 알고리듬

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

Design Error Searching Algorithm in VHDL Behavioral-level using Hierarchy

Seongwook Yoon, Hyunkweon Jung, Jinju Kim, Dongwook Kim
Dept. of Electronic Materials Eng. Kwangwoon Univ.
447-1 Wolgye-Dong Nowon-Ku, Seoul, Korea

Abstract

A method for generation of design verification tests from behavior-level VHDL program is presented. Behavioral VHDL programs contain multiple communicating processes, signal assignment statements. So for large, complex system, it is difficult problem to test or simulation.

In this paper, we proposed a new hardware design verification method. For this method generates control flow graph(CFG.) and process modeling graph(PMG) in the given under the testing VHDL program. And this method proved very effective that all the assumed design errors could be detected.

Ⅰ. 서론

근래에 들어 반도체 설계방법의 발달로 고집적화 회로 설계가 가능하게 되었으며, 그에 따라 종래의 회로 설계 방법은 오히려 회로 설계에 있어서 시간과 비용 낭비만을 가져올 뿐이다. 이런 이유로 최근에는 이에 대해 설계 시간을 단축시키고 비용을 효과적으로 절감할 수 있는 방법으로 하드웨어 표현 언어 (Hardware Description Language: HDL)를 사용하여 설계를 보다 효율적으로 하고 있다.[1]

VHDL의 특징중의 하나가 행위 수준(Behavioral level)에서 설계가 가능하다는 것인데, 행위 수준에서는 스케매틱 방법에서와는 다르게 원하는 회로의 동작을 프로그래밍 방법으로 설계를 할 수 있기 때문에 훨씬 복잡한 설계에서도 보다 효율적으로 설계를 할 수 있다.

그리고 임의의 VHDL파일이 실제 회로로 구현되는 과정에서도 컴퓨터를 이용하여 컴퓨터 상에서 설계 프로그래밍, 게이트 레벨로의 합성, 실제 회로의 레이아웃까지가 한 과정에서 모두가 이루어질 수 있으며, 이때 설계된 회로와 실제로 구현된 회로간은 거의 완벽한 동일성을 가지고 있기 때문에 하드웨어 설계 검증의 필요성이 작아지고 있다.

오히려 프로그래밍 상태의 VHDL을 시뮬레이션 함으로써 실제 구현 회로에 대한 오류를 예상하여 설계하는 것이 일반화 되어 있다.

이에 대해 본 논문은 실제 구현되는 회로에 대한 검증이 아니라, 설계후 컴파일과 시뮬레이션 과정에서 발견되지 않는 코딩 에러를 찾는 것을 본 논문의 목적으로 잡았다. 코딩 에러는 설계자가 원하는 특성(Specification)[2]과 설계자의 설계 특성이 프로그래밍된 VHDL파일의 동작(implementation)[3]에서 입.출력 정보를 서로 비교하여 설계자의 의도한 바로 설계가 잘 되었는지를 검증하는 것이 본 논문에서 제시하는 설계검증인 것이다..[2][3][4]

설계 검증을 위한 방법으로는 기존의 Formal Verification 방법에서 Equivalent를 추측하는 방법의 형식을 응용한 검증 방법이며, 보다 효율적으로 검증을 수행하기 위해서 Control Flow Graph(CFG), Process Modeling Graph(PMG)의 그래프적 해석 방법을 사용한다. 마지막 단계에서는 비교결과 비교 기준이 되는 설계자의 특성과 비교하여 불일치 하는 패턴에 대하여 경로 검색을 통하여 오류를 찾아내어 수정

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

(a)

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

(b)

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

(c)

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

그림1. 골드유닛과 오류

하는 것이다. 본 논문에서 사용하는 오류의 종류와 알고리듬은 다음에 설명한다.

Ⅱ. 오류의 분류와 등가 오류

이 논문에서 검출 목적으로 잡는 오류는 일반적인 논리적 오류가 아니고 코딩 내에서 if, case문 등을 작성해 가는 과정에서 일어나는 코딩 오류들이다. 즉, 컴파일 과정에서 발견되는 논리적 오류를 제외한 설계자의 의도와 다르게 잘못 기술된 오류를 타겟으로 한다.

Ⅲ-1. 할당 상태 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. (case문 검색할 때 이용)

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. (if문 검색할 때 이용)

Ⅲ-3. 등가오류(equivalent)

위에서 살펴본 두 가지의 오류는 if문에서 보면 조건이 두가지로 한정되어 있기 때문에 결과적으로는 같은 내용을 가지고 있고 이는 잘못된 조건이나 할당이 수정되었을 때는 HDL코딩의 전,후라는 차이뿐이지 원하는 조건에 할당되는 값들은 결과적으로 같음을 알 수 있다.

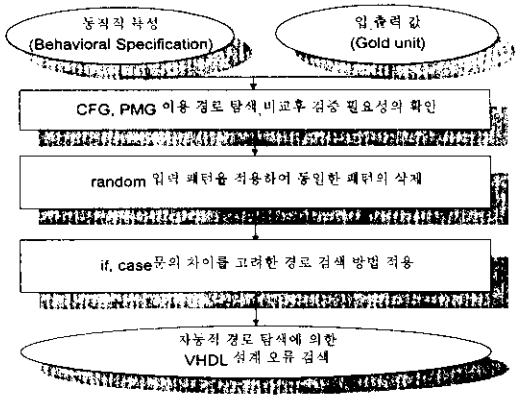


그림2. VHDL에 대한 설계 검증 구조

그러므로 if문에서의 오류 검색 시에는 두가지중 한 가지를 선택하여 보다 효율적으로 오류 검색을 수행할 수 있다.

Ⅲ. VHDL 설계 검증 구조

그림 2에 VHDL에 대한 설계 검증 구조의 개략도를 나타내었다. VHDL에 의한 설계는 다른 언어 내지 방법으로 코딩된 설계사양을 기준으로 이루어진다. 이때 설계자는 대상 HDL 파일을 모든 동작의 기본이 되는 입·출력 값, 즉 골드 유닛(Gold Unit)의 값을 얻을 수 있다. 골드유닛의 입출력 정보는 검증과정에서 코딩의 오류와 비교대상이 되므로 매우 중요하며 따로 파일로 기록되어 사용되는 값이다.

다음으로 검증의 필요성을 알기 위해 CFG, PMG를 이용하여 원래의 코딩에 대한 경로 탐색을 수행하여 대상 VHDL에 대해서 입,출력 값을 획득하고, 그 결과가 골드 유닛과 차이가 난다면 잘못된 오류의 위치를 파악하여 수정하기 위해 검증과정이 필요하게 되는 것이다.

다음으로 random 입력 패턴을 사용하여 골드유닛과 동일한 경로를 삭제 함으로써 검색의 효과를 높이고, 마지막으로 if, case 의 다른 조건과 할당을 이용하여 검색하는 것이다.

Ⅳ. 오류 검색 알고리듬

오류 검색 알고리듬을 그림 3에 전체적인 흐름도를 이용하여 나타내었다. 이 흐름도에 의하면 처음으로 골드유닛과 대상 VHDL의 입,출력 패턴이 차이가 있을 경우 검색의 필요성을 확인한 다음 오류 검색을 시작한다. 다음으로는 경로 삭제인데 이것은 단일 프로세

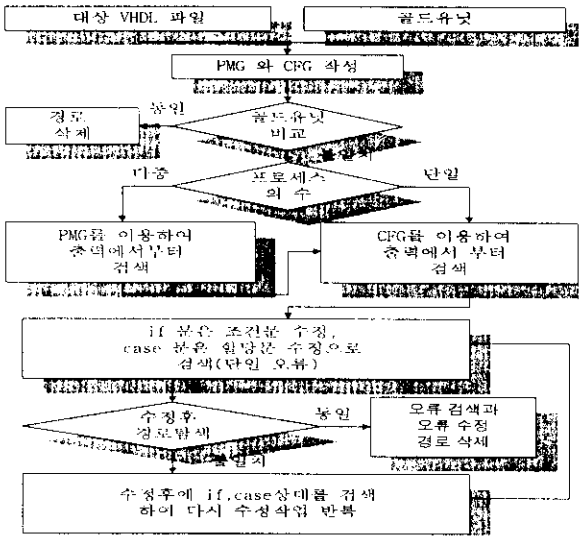


그림 3. 오류 검색 알고리듬

스 과정에서 입.출력 패턴 비교시에 수행하는 과정이다. 이 것은 나중에 검색할 때 경로 탐색의 시간과 수고를 줄이기 위한 방법이다. 그리고 다음으로 코딩이 단일 프로세스인지, 다중 프로세스인지 확인한다. 만약에 다중 프로세스인 경우 PMG와, 일치 하지 않는 출력으로부터 잘못된 상태, 오류가 있는 프로세스를 찾고 더 세부적으로 들어가 프로세스 내부의 검색을 시작한다. CFG를 이용하며, 이때는 if문과, case문의 검색 방법이 틀린데, 일단 if문은 조건이 두가지이기 때문에 일치하지 않는 조건에 대해서 반대되는 조건으로 경로 탐색을 확인한다. 그리고 if문에서 조건문 끝까지 가서도 출력이 다르다면 마지막 할당을 바꾸어 준다. case문인 경우는 다중 할당이기 때문에 할당을 가지고 출력을 동일하게 만들어 주면서 검색을 수행하게 된다. 여기서 만약에 오류가 발생한 위치를 찾지 못한다면 단일 오류 조건에 따라서 임의로 수정한 오류가 맞는다는 것이다. 그러므로 현 상태 이전의 if, case문을 찾아서 다시 수정과 경로 검색 과정을 거치면서 찾게 되는 것이다.

위의 알고리듬을 VHDL의 행위 레벨로 설계된 예에 적용 시키는데, 이에 대한 VHDL코딩과 PMG, CFG를 그림4, 그림5 에 나타내었다.

과정을 설명하면 다음과 같다.

i) PMG와 CFG에서 입.출력 정보가 골드 유닛과 동일한지 비교 검토를 처음으로 시작한다. 여기서는 입력 패턴 (en, en_1, en_2, en_3, con) = (0, 0, 1, 1, 10)에 대한 출력 패턴이 골드유닛의 '1' 값과 다른 '0'이 나오므로 검색을 시작한다.

```
1   Entity example is
2     port (en, en_1, en_2, en_3, con : in bit;
3           out_1, out_2 : out bit);
4   end example ;
5
6   Architecture behavioral of example is
7   begin
8   P1 : process (en, en_1, en_2)
9     begin
10      if en = '1' then out_1 <= 1 ;
11        elsif en_1 = '0' then
12          if en_2 = '0' then out_1 <= 0 (1) ;
13            else out_1 <= 1 ;
14          end if ;
15      else out_1 <= 0 ;
16      end if ;
17    end process;
18
19  P2 : process (en_3, con)
20    begin
21      if en_3 = '1' then ;
22        case con is
23          when "00" => out_2 <= 0 ;
24          when "01" -> out_2 <= 0 ;
25          when "10" -> out_2 <= 0 ;
26          when "11" -> out_2 <= 0 ;
27        end case ;
28      else out_2 <= 0;
29  end ifl
30  end process;
```

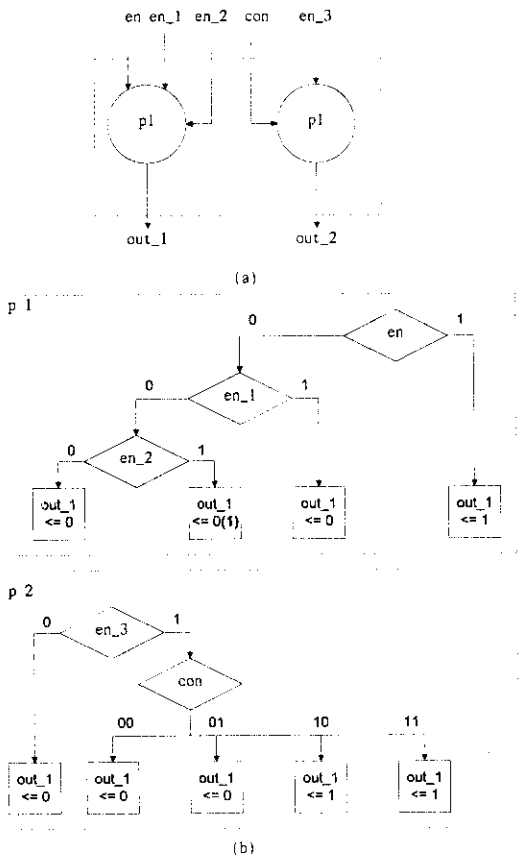
그림4. example을 위한 HDL 코딩 예

ii) 다음으로 PMG에서 의 출력을 보면 out_1, out_2,가 있는데 현재 골드 유닛과 차이가 나는 출력은 out_1 이다. 그러므로 PMG에서는 p1으로부터 잘못된 출력 값이 나온다는 것을 알 수 있다. 그러므로 p1의 CFG를 이용하여 더욱 세세한 경로 검색을 한다.

iii) 다음으로 p1의 CFG를 보면 if문으로 이루어 졌으므로 출력으로부터 가장 가까운 조건 선택 상태를 찾는다. 여기서는 en_2이며 다음으로는 반대되는 조건이 삭제된 경로인지 확인한다. 여기서는 삭제된 조건이기 때문에 상태 en_2에서는 조건을 검색할 필요 없이 en_1 상태로 올라가 조건을 바꾸어 경로 검색을 한다. 그러나 바뀌어도 출력 값은 변화가 없고 en 상태까지 온다. en 상태에서도 처음 비교때 삭제된 경로가 있기 때문에 결국에는 조건이 잘못된 것이 아니라 출력 쪽의 할당이 잘못된 것을 알 수 있다. 그리고 할당을 수정함으로써 오류 검색을 끝마치게 된다.

V. 오류검색 실험 결과

본 논문에서는 표1의 4가지에 대한 회로실험 적용은 if, case문만으로 이루어진combinational 회로에 대해서만 검증을 수행하였다. 오류는 단일오류만을 고려하였으며, 조건과 할당 상태 오류를 단일오류로써 고려하



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

그림5. PMG 와 CFG

였다. 실험은 본 논문의 목적인 자동적인 오류 검색이었기 때문에 C언어로 프로그램은 짰으며, 각 대상 VHDL파일을 읽어들여 골드유닛과 비교 할 수 있도록 CFG, PMG 작성과 마찬가지인 표를 작성하여 경로를 형성 시켰다. 각 회로의 특성과 본 논문에서 제안된 알고리듬의 적용 결과를 같이 나타내었다. 결과는 오류 검출률이 다 100%를 나타내는 데 반해 case 문을 포함하고 있는 것은 반복횟수가 늘어나는 것을 볼 수 있는데 이는 case문이 if문과 틀리게 두 개 이상의 조건을 가지고 할당을 주기 때문에 모든 할당을 맞추어 주려면 case문의 할당 수만큼의 반복이 필요하다.

그리고 다중 process인 경우도 역시 process들간의 상호 입,출력 정보 때문에 검색 반복 횟수가 늘어 나는 것을 알 수 있다. if문은 오류의 동일성으로 인하여 하나의 if문마다 한번씩의 과정으로 오류 검색이 가능하다.

name	코딩 특성			설계 오류 특성	
	process	if문	case문	error수	coverage
shift regiter	2	2	1	6	100%
resource share	3	2	1	6	100%
example	2	4	1	7	100%
8_1 mux	1	-	1	8	100%
8_decoder	4	12	-	18	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, if문에서는 조건을 case문에서는 할당에 대한 오류를 중심으로 검색 결과, 모든 단일 오류를 검출 할 수 있음을 알 수 있었다. 그러나 case 문에서와 다중 프로세스문에서는 할당 수만큼, 또는 프로세스 수만큼의 반복을 수행해야만 원하는 검출률을 얻을 수 있었으므로 반복 회수가 늘어남을 알 수 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] H. Trickey, "Flamel: A High-Level Hardware Compiler" IEEE Trans. CAD, Vol. CAD-6, No.2, pp.259-269, March 1987
[8] 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
[9] 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