論文2001-38SD-3-4

VHDL 행위-레벨 설계의 코딩오류 검출을 위한 패턴 생성

(Pattern Generation for Coding Error Detection in VHDL Behavioral-Level Designs)

金 宗 賢 *, 朴 承 奎 *, 徐 英 鎬 *, 金 東 郁 *

(Jong-Hyeon Kim, Seung-Kyu Park, Young-Ho Seo, and Dong-Wook Kim)

요 약

최근 VHDL 코딩 및 합성방법에 의한 설계가 널리 사용되고 있다. 집적도가 증가함에 따라 VHDL에 의한 설계 또한 그 분량이 증가하여 많은 코딩오류가 발생하고 있으며, 이를 검색하는데 많은 시간과 노력이 소요되고 있다. 본 논문에서는 VHDL 행위-레벨 설계를 대상으로 코딩오류를 검색하는 방법을 제안하였다. 그 방법에 있어서는 검색패턴을 생성하여 오류가 없는 응답과 설계의 응답을 비교함으로써 설계오류를 찾는 방법을 택하였다. 따라서 본 논문에서는 코딩오류를 검색하기 위한 검색패턴을 생성하는 알고리듬을 제안하였다. 검색패턴 생성은 각 코드에 대해 수행하며, 할당오류와 조건오류를 구분하여 수행하였다. 패턴생성을 위해 VHDL 코드를 CDFG로 변환하여 사용하며, CDFG상의 경로를 탐색하여 패턴생성에 필요한 정보를 추출한다. 경로탐색은 오류가 발생하였다고 가정한 지점으로부터 역방향 탐색과 정방향 탐색을 수행하여 패턴을 생성한다. 제안한 알고리듬은 C-언어로 구현하였다. 펜티엄-II 400MHz의 환경에서 여러 가지 VHDL 행위-레벨 설계를 대상으로 제안한 알고리듬을 적용하였다. 그 결과, 고려한 모든 설계의 모든 코드에 대한 검색패턴을 생성할 수 있었으며, 가정한 모든 오류를 검색할 수 있었다. 검색패턴 생성에 소요되는 시간은 고려한 모든 대상 설계에서 1초 미만의 CPU 시간을 보여 속도면에서도 매우 우수함을 나타내었다. 따라서 본 논문에서 제안한 검색방법은 VHDL에 의한 설계에서 설계검증에 필요한 시간과 노력을 상당히 감소시킬 것으로 기대된다.

Abstract

Recently, the design method by VHDL coding and synthesis has been used widely. As the integration ratio increases, the amount design by VHDL at a time also increases so many coding errors occur in a design. Thus, lots of time and effort is dissipated to detect those coding errors. This paper proposed a method to verify the coding errors in VHDL behavioral-level designs. As the methodology, we chose the method to detect the coding error by applying the generated set of verifying patterns and comparing the responses from the error-free case(gold unit) and the real design. Thus, we proposed an algorithm to generate the verifying pattern set for the coding errors. Verifying pattern generation is peformed for each code and the coding errors are classified as two kind: condition errors and assignment errors. To generate the patterns, VHDL design is first converted into the corresponding CDFG(Control & Data Flow Graph) and the necessary information is extracted by searching the paths in CDFG. Path searching method consists of forward searching and backward searching from the site where it is assumed that coding error occurred. The proposed algorithm was implemented with C-language. We have applied the proposed algorithm to several example VHDL behavioral-level designs. From the results, all the patterns for all the considered coding errors in each design could be generated and all the coding errors were detectable. For the time to generate the verifying patterns, all the considered designed took less than 1 [sec] of CPU time in Pentium-II 400MHz environments. Consequently, the verification method proposed in this paper is expected to reduce the time and effort to verify the VHDL behavioral-level designs very much.

* 正會員, 光云大學校 電子材料工學科
(Dept. of Electronic Materials Eng. Kwangwoon Univ.)

※ 본 연구는 1998-2000년 한국학술진흥재단의 자유공모과제 지원에 의해 수행되었음. (KRF-1998-001-200847)

接受日字:1999年8月23日, 수정완료일:2001年2月3日

Ⅰ. 서 론

반도체 집적도의 증가는 한 IC에 구현 가능한 회로의 양을 급속도로 증가시켰다. 이에 따라, 종래에 사용되어 오던 스케매틱(schematic)에 의한 설계는 대형회로 설계시 설계시간이 과다하게 요구되어 하드웨어 표현 언어(Hardware Description Language, HDL)에 의한 설계방법이 지속적으로 연구되어 왔다. IEEE에서는 1987년에 VHDL(VHSIC HDL)[1]을 표준화함으로써 HDL에 의한 설계가 더욱 활성화되어 현재 ASIC 설계의 50% 이상이 HDL에 의해 설계되고 있으며, 향후 이 비율은 더욱 증가할 것으로 전망하고 있다[2,3].

현재 널리 사용되고 있는 HDL은 VHDL 및 Verilog HDL[4]이며, 두 언어 모두 추상화 레벨에 따라 몇 가지의 설계 레벨을 제공하고 있다. 그 중 가장 추상화 레벨이 높은 레벨을 행위-레벨(behavioral-level) 또는 함수-레벨(functional-level)이라 하며, 이 레벨에서 이루어진 설계는 합성(synthesis) 단계를 거쳐 하드웨어로 변환된다[5]. 최근의 많은 설계가 이 레벨에서 이루어지고 있으며, 본 논문도 이 레벨에서의 설계를 대상으로 한다.

회로의 대형화에 따라 행위-레벨의 설계 또한 그 설계량이 급속히 증가하고 있어, 설계검증에 있어서의 비용이 설계비용에서 차지하는 비율이 점차 과다해지고 있다[3]. 일반적으로는 기능적 시뮬레이션(functional simulation)을 통해 설계검증이 이루어지나, 최근에는 HDL에 의한 설계 단계의 진척에 따른 설계검증 방법들이 개발되고 있으며, 실제로 CAD 툴에서 사용되고 있는 것들도 있다. 가장 대표적인 방법이 정형검증(formal verification)[6~9]으로, 현재 SYNOPSIS[9]나 CADENCE[10] 등의 설계 툴에서 실제로 사용되고 있다. 정형검증의 원 의도는 VHDL에 의한 설계가 사양에 부합하는지를 검증하는 것이나, 현재 사용되고 있는 상용화 툴들은 설계에 대한 기능 시뮬레이션을 통과하였다고 가정한다. 즉, 원 설계에 테스트를 위한 전략적 하드웨어를 부가한다던가 특별한 기술에 매핑할 때 또는 합성할 때 그 결과가 그 전의 설계와 동일 기능을 갖는가를 검증하는 것이다. 정형검증을 이용한 툴은 현재 데모 버전 정도가 선보이고 있으며[9,10], 각 툴에서 정의하는 특별한 형식으로 사양을 정의하도록 되어 있다. 따라서 HDL설계와 사양설계의 이중 설계가 필요하다. 또한 [11]에서는 시뮬레이션에 의한 기능적 검색의 검색률을 계산하는 방법을 제안하였는데, 이 방법에서는 4치 연산을 수행하고 있다. 최근 소프트웨어 테스팅 방법을 응용하여 기능적 검색을 위한 검색패턴을 생성하는 방법을 제안하였다[12]. 이 방법은 경로탐색(path enumeration), 제한조건 생성(constraint generation) 및 제한조건 해결(constraint solving)의 세 단계를 통해 패턴을 생성하는데, 계산량이 과다히 많은 단점을 갖고 있다.

본 논문에서는 기능적 검색의 일환으로, VHDL 설계의 코딩오류를 검색하는 방법을 제안한다. 본 논문에서 제안하는 방법은 원시 VHDL 코드에 대해 기능적 시뮬레이션이나 정형검증 등의 검색을 수행하기 전에 적용하는 것을 목적으로 하며, 이러한 검색을 대신할 수 있다. 검색방법은 VHDL의 각 코딩에 대한 검색패턴을 생성하여 적용하는 방법을 사용한다. 현재까지의 설계 검증 방법들은 모두 기능적인 사건에 대한 검증을 주 목적으로 연구하였으나, 본 논문에서는 VHDL 코딩 자체에 대한 검색을 목적으로 한다.

본 논문의 검색방법은 VHDL 설계에 대해 검색패턴을 생성하여 그 패턴들을 설계와 오류가 발생하지 않은 설계 데이터(gold unit)에 각각 대입하여 그 응답을 비교하는 방법이다. 따라서 본 논문에서는 코딩오류의 검색패턴을 생성하는 방법을 제안하며, 이를 실제 설계에 적용하여 본 논문에서 의도하는 바대로 동작하는지를 보인다. 본 논문에서는 주어진 VHDL 코드에서 단일 코딩오류를 가정한다.

다음 장에서는 본 연구의 대상 코딩오류에 대해 설명하고, 이 코딩오류들을 검색하는 패턴생성 방법 및 패턴생성 절차를 3장에서 설명한다. 본 논문에서 제안하는 방법을 구현하고 이를 실험한 결과를 4장에서 보이고, 5장에서 본 논문의 결론을 맺는다.

Ⅱ. 코딩오류의 종류 및 분류

집적도 증가에 따르는 회로 크기의 증가는 VHDL 설계에서 그 코딩 자체의 양을 증가시킨다. VHDL 코딩은 사람의 손을 거치지 않을 수 없으므로 분명 코딩오류를 발생시키며, 코딩오류의 확률은 회로의 양이 커질수록 높아진다. 본 장에서는 본 논문에서 고려하는 코

딩오류의 종류를 피력하며, 이를 분류하여 코딩오류 검색패턴 생성에 사용한다.

본 논문은 VHDL의 추상화 레벨을 행위-레벨로 설정하였으므로 그 대상 코딩오류 또한 행위-레벨 설계에서의 오류들을 선택하였다. 표 1은 본 논문에서 고려하는 코딩오류를 분류하여 간단한 예와 같이 나타낸 것이다. 일반적으로는 '조건문'과 '할당문'의 두 종류로 구분하나, 본 논문에서는 '반복문'과 '순차문'을 구분하였으며, 'component'문과 'generate'문을 구조문으로 분류하여 따로 분리하였다. 이들 분류된 각 종류는 그 자체적인 특성을 지니고 있는데, 예를 들어, 할당문에서의 오류는 잘못 할당된 변수가 출력에 영향을 미쳐 출력값에 오류를 발생시킨다. 그러나 조건문에서의 오류는 데이터의 흐름을 바꾸어 잘못된 데이터 흐름경로를 선택함으로서 결과에 오류를 발생시킨다.

Ⅲ. 검색패턴 생성

본 논문의 목적은 VHDL 행위-레벨에서의 코딩오류를 검색하는 검색패턴을 생성하는 것이다. 본 논문에서 제안하는 검색패턴이 설계과정에서 어떻게 사용되는지를 그림 1에 나타내었다. 먼저 본 논문은 설계사양으로

표 1. VHDL 행위-레벨 코딩오류의 분류
Table 1. Classification of VHDL coding errors in behavioral-level.

분류	Keyword	오류 및 예
조건문	if	(1) 조건 할당 오류 * *if **a='1'** then → if **a='0'** then*
		(2) 우선순위 오류 * *if **a='1'** then → if **b='1'** then* *if **b='1'** then if **a='1'** then*
		(3) 'event'문 * *if **a** = **'1' and a'event** then* → *if **a** = **'1'** then*
	case	(1) 조건 할당 오류 * *case a is* *when **"01"** => c <= '1';* → *case a is* *when "10" => c <= '1';*
		(2) 조건 대상 오류 * *case **a** is → case **b** is*
할당문	조건문 내 또는 조건문 외	(1) 연산자 사용 오류 * *and, or, nand, nor, +, - 등* → *다른 것 사용*
		(2) 할당오류 * *a <= **'1'**; → a <= **'0'**;* * *a <= **b**; → a <= **c**;* * *a <= **b**; → c <= **b**;*
반복문	loop	(1) 반복회수 오류 * *for a in **0** to **9** loop* → *for a in **1** to **10** loop*
구조문	component	(1) port mapping 오류 * *component port map(**a, b, c**)* → *component port map(**b, a, c**)*
	generate	(1) 반복회수 오류 * *for a in **0** to **9** generate* → *for a in **1** to **10** generate*
순차문	process	(1) process의 sensitive list 오류 * *process(**a, b, c**)* → *process(**a, b**)* → *process(**a, b, c, d**)*

부터 gold unit이 생성되거나 설계사양자체를 gold unit으로 사용한다고 가정한다. 이 때 gold unit으로부터 최소한 입력패턴에 대한 오류가 없는 출력패턴을 얻을 수 있는 것으로 가정한다.

설계사양에 따른 VHDL 설계가 이루어지면, VHDL 코드 자체를 컴파일하여 컴파일 오류를 먼저 제거한다. 컴파일 에러가 없는 VHDL 코드는 CDFG(Control & Data Flow Graph)로 변환되는데, 이것에 대해서는 뒤에 다시 설명하기로 한다. CDFG를 대상으로 코딩 오류 목록을 생성하고 이 목록의 각 오류에 대해 검색패턴을 생성한다. 생성된 검색패턴은 gold unit과 VHDL 설계에 동시에 적용하여 그 응답을 비교하여 오류의 발생여부를 판단한다. 오류가 없을 때까지 반복하여 설계를 완성한다.

이 과정에서 본 논문의 주된 목표는 검색패턴을 생성하는 것이므로 이를 위한 재원, 방법, 과정 등을 이장에서 설명한다.

1. Control & Data Flow Graph (CDFG)

본 논문의 목적인 코딩오류 검색패턴을 생성하기 위하여 본 논문에서는 CDFG(Control Data Flow Graph)를 재원으로 사용한다. CDFG는 CFG와 DFG를 결합한 형태로서 조건문은 ◇내에, 할당문은 □내에 각각 나타낸다. CDFG는,

$$CDFG \in \{nodes, edges\} \quad (1)$$

로 구성되며, 이 중 노드들은,

$$nodes \in \{assignment(\square), condition(\diamond)\} \quad (2)$$

이고, 에지들은 모두 방향성 에지(directed edge)이다. 그림 2(b)에 그림 2 (a)의 VHDL 행위-레벨 코드에 대한 CDFG를 나타내었다.

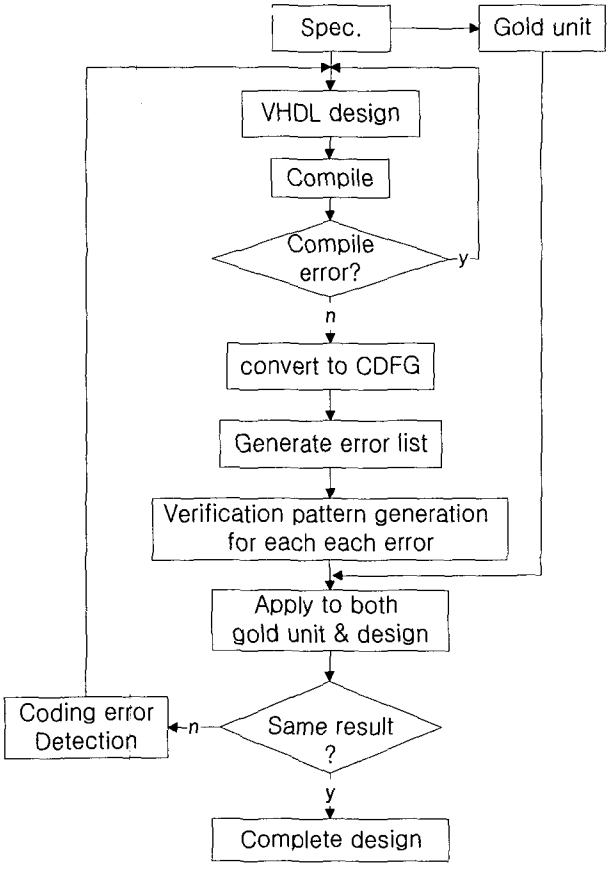


그림 1. 검색패턴에 의한 코딩오류 검색과정
Fig. 1. Verification procedure for coding errors with verification patterns.

2. Gold Unit

모든 시뮬레이션 및 검색은 그 기준이 되는 설계 또는 데이터가 필요하며, 그 설계 또는 데이터는 오류가 없는 것이어야 한다. 일반적으로 이 데이터를 gold unit이라 하며, 모든 설계 또는 검색에 기준이 된다.

본 논문에서도 코딩오류의 검색에 필요한 기준 설계(gold unit)가 필요하며, 검색을 위해 또 다른 특별한 형태의 설계를 하여야 하는 기존의 방법과는 달리, 본 논문에서는 주어진 설계의 주입력(primary inputs)의 자극에 대한 주출력(primary outputs)의 응답을 구할 수 있다는 가정만을 한다. 즉, 본 논문에서 제안하는 패턴생성 방법으로 생성된 패턴을 인가하여 오류가 발생하지 않는 주출력의 응답을 구할 수 있다고 가정한다. 따라서 본 논문의 목적에 맞는 gold unit은,

(1) 설계 사양

(2) VHDL이 아닌 상위-레벨 언어로 기술된 동작

(3) 합성 불가능한 VHDL 코드로 구현된 설계

중 어느 것도 무관하며, 단지 입력의 자극에 대한 출력의 응답만을 비교대상 데이터로 사용한다.

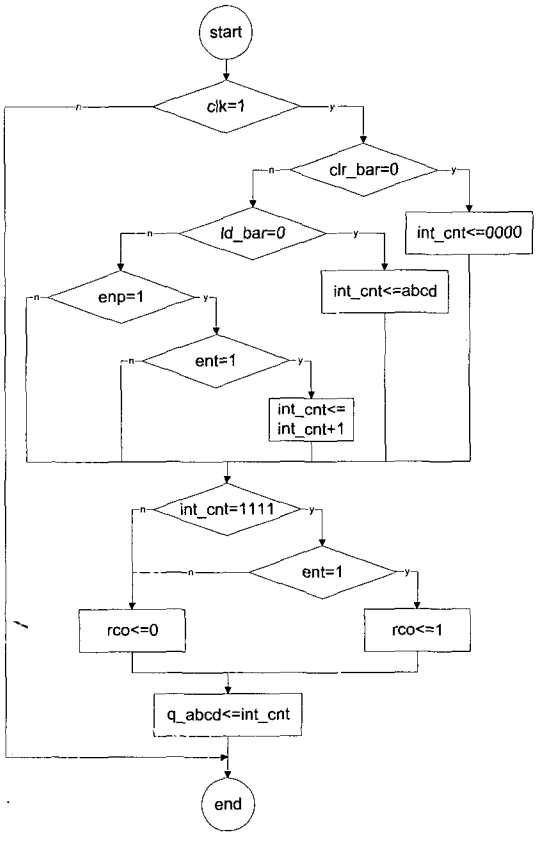
3. 코딩오류 검색패턴 생성 방법

앞에서 설명한 바와 같이 본 논문에서는 주입력의 자극에 대한 주출력의 응답만을 사용하여 코딩오류를

```
library ieee;
use ieee.std_logic_1164.all;
entity ls163_counter is
        port(clk,clr_bar,ld_bar,enp,ent:in std_logic;
             abcd:in std_logic_vector(3 downto 0);
             rco:out std_logic;
             q_abcd:out std_logic_vector
             (3 downto 0));
end ls163_counter;

architecture behavior of ls163_counter is
 signal int_cnt:std_logic_vector(3 downto 0):="0000";
begin
process(clk,clr_bar,ld_bar,enp,ent,abcd)
begin
if clk='1' then
        if clr_bar='0' then
                int_cnt<="0000";
        elsif ld_bar='0' then
                int_cnt<=abcd;
        elsif enp='1' and ent='1' then
                int_cnt<=int_cnt+"0001";
        end if;
if int_cnt="1111" then
        rco<='1';
else
        rco<='0';
end if;
q_abcd<=int_cnt;
end if;
end process;
end behavior;
```

(a)



(b)

그림 2. VHDL 코딩 예와 CDFG
(a) VHDL 코딩, (b) CDFG

Fig. 2. VHDL coding example and its CDFG
(a) VHDL coding, (b) CDFG

검색한다. 따라서 오류가 발생한 지점의 오류정보를 주출력까지 전달하여 한 개 이상의 주출력에서 그 결과가 가시화되어야 한다. 또한 오류 발생지점에서 오류가 발생되도록 신호의 흐름을 제어해 주고 오류를 유발시키는 주입력의 패턴이 생성되어야 한다. 이 과정은 논리회로의 테스트 패턴 생성방법 중 경로부각 방법[12-15]과 유사하다고 볼 수 있다. 그러나, 논리회로의 경로부각의 경우 경로를 따라 지속적으로 오류를 전달하면 종국에는 항상 주출력을 만나게 되지만, VHDL 코드의 경우 마지막 코드가 반드시 주출력을 나타내는 것은 아니다. 또한 회로의 경우 시발점은 반드시 주입력이나 VHDL은 그렇지 않을 수 있다.

따라서 논리회로의 경로부각 방법을 그대로 사용할 수 없다. 본 논문에서는 기본적으로는 논리회로의 경로부각 방법을 사용하나, 그 세부적인 방법은 VHDL 코드의 탐색에 맞도록 수정 또한 개발하여 검색패턴을 생성한다.

본 논문에서는 할당오류와 제어오류를 구분하여 검색패턴을 생성하며, 그 방법은 다음과 같다.

1) 할당오류

먼저, 할당에 대한 오류는 변수에 잘못된 값이 할당되거나 올바른 값이 다른 변수에 할당되는 경우에 발생한다. 따라서 잘못 할당된 값이 출력에 영향을 주어 출력의 논리적인 응답이 오류가 발생하지 않는 경우와 반대의 값이 출력되도록 하여야 한다.

본 논문에서는 주어진 VHDL 설계에 대해 각 할당문에 오류가 발생되었다고 가정하고, 각 할당오류에 대해 검색패턴을 생성한다. 또한 본 논문에서는 가능한 오류만을 선택하는데, 그 예를 그림 3에 나타내었다. 이 그림은 A와 B를 비교하는 비교기를 표현한 것이다. 그림 3 (a)는 오류가 없는 코드를 나타내고 있는데, A<B의 경우 Y에 '1'을, 그렇지 않은 경우는 Y에 '0'을 할당한다. 이 경우의 현실적인 할당오류는 A<B인 경우 Y에 '0'을 할당하고 A≥B인 경우 Y에 '1'을 할당하는 것이다. 그러나 (c)와 같이 두 경우 모두 Y에 '1'을 할당한다거나 두 경우 모두 Y에 '0'을 할당하는 것은 비현실적이라 볼 수 있다. 본 논문에서는 (b)와 같은 오류만을 고려한다.

VHDL 코드 내부의 한 할당문에서 오류가 발생한 경우, 그 할당문이 주출력 중 하나의 값을 할당할 수 있다. 이 경우에 대해서 다음의 정리가 성립한다.

[정리 1] *VHDL 코드의 내부 할당문이 주출력(Y)의 값을 할당할 경우, CDFG상에서 그 할당문 이전의 코드들에 의해 Y를 부각시키는 주입력의 조건이 반드시 존재한다.*

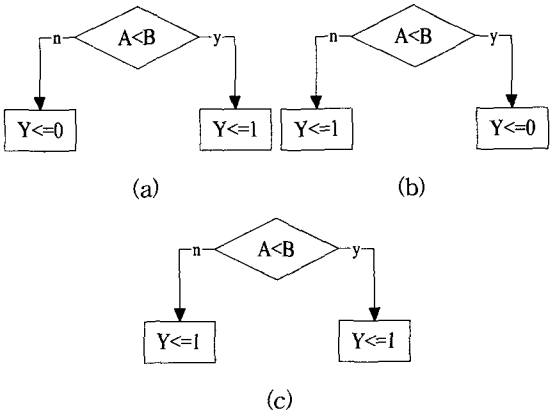


그림 3. 할당오류의 현실성
(a) 오류가 없는 코드, (b) 현실적인 오류, (c) 비 현실적인 오류

Fig. 3. Reality of assignment error.
(a) Error-free code, (b) realistic error, (c)unrealistic error

(증명) 간단한 할당문의 예인 'Y<= A'로 증명한다. 여기서 A는 상수, 변수, 표현식일 수 있다. 표현식의 경우는 그 자체로 하나의 상수 또는 변수로 취급할 수 있으므로 상수인 경우와 변수인 경우만을 증명한다. 먼저 A가 상수인 경우, CDFG상에서 이 할당문 이전의 코드들에 주입력이 포함되지 않는다면 하드웨어적으로 전원과 접지의 의 조합이 직접 출력에 연결되어야 하고 이 때의 출력은 항상 동일한 결과를 출력하므로, 이 경우는 설계에서 제외되어야 한다. 또한 이 경우는 컴파일 과정에서 에러 메시지를 유발한다. A가 변수인 경우는 A가 주입력일 경우와 그렇지 않은 경우로 나누어 볼 수 있다. 먼저 A가 주입력이 아닌 경우, A는 주출력일 수 없으므로 내부변수임에 틀림없다. 만약 이 내부변수를 부각시키는 주입력이 그 전의 코드에 존재하지 않는다면, 그 내부변수는 하드웨어적으로 주입력과 연결상태가 없다는 뜻이므로, 이런 경우는 있을 수 없다. 마지막으로 A가 주입력인 경우는 A가 주출력 Y를 부각시키는 것이므로 위의 정리는 성립한다. ◆

이 정리를 이용하면 대상 오류가 주출력에 대한 할

당을 포함하는 경우는 그 뒤의 코드를 탐색할 필요없이 그 전의 코드만으로 패턴을 생성할 수 있다.

그러나, 주출력에 대한 할당을 포함하지 않는 경우는 그 변수가 영향을 미치는 주출력을 찾을 때까지 그 뒤의 코드들을 탐색하여야 한다.

2) 조건오류

조건오류는 근본적으로 신호의 부각경로를 변경시킨다. 따라서 조건오류는 할당오류와는 달리 신호흐름의 변화에 맞추어 오류에 의한 신호경로와 오류가 발생하지 않았을 때의 신호경로에서 서로 다른 결과를 도출하여야 한다. 조건오류를 고려함에 있어서 다음의 세 경우를 고려한다.

(a) 일반적인 조건오류

그림 4 (a)의 예는 일반적인 경우로 볼 수 있는데, 이 구문 이전 또는 이후의 코드에 따라 I1과 I2의 값이 서로 같을 수 있다. 따라서 VHDL 코드 전체를 검색하여 A<B를 만족하는 경우와 그렇지 않은 경우에 있어서의 Y값이 서로 다르도록 패턴을 생성하여야 한다. 따라서 I1과 I2가 서로 다른 값을 갖도록 주입력의 값들을 결정하여야 한다.

(b) 보유상태를 포함하는 오류

그림 4 (b)의 경우는 조건에 따라 보유상태(하드웨어적으로 보면 레치나 플립-플롭을 포함하여야 하는 경우)를 가질 수 있다. 즉, A<B의 조건을 만족하지 않고 이전 또는 이후에 Y에 대한 할당이 새로 이루어지지 않는다면 Y는 그 전 시간대에 가졌던 값을 그대로 보유하게 될 것이다. 이런 경우는 단일 패턴으로는 검색할 수 없으며, 순차회로에서와 같이 최소 두 개의 패턴이 필요하다. 이 두 패턴은 A<B일 때와 A≥B일 때의 Y값이 서로 다르도록 패턴을 생성하여야 한다.

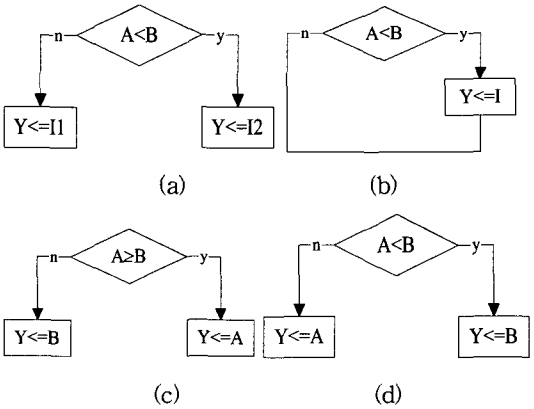


그림 4. 조건오류에서 고려할 세 가지
(a) 일반적인 경우, (b) 보유상태를 포함하는 오류, (c),(d) 등가오류

Fig. 4. Three considerable condition error.
(a) general case, (b) error with state retainment case, (c),(d) equivalent errors

이 외에 두 개 이상의 분기를 형성할 때 분기된 후의 할당문에서 서로 다른 변수에 대한 할당이 이루어지는 경우도 보유상태를 포함한다. 그림 4 (a)에서 분기된 후의 두 할당문이 A<B일 때 'Y<= I1'대신 'Y<=a', A≥B일 때 'Y<=I2' 대신 'Z<=a'로 각각 할당이 이루어진다면 이 경우에 해당한다. A와 B의 값에 따라 두 할당문 중 하나를 반드시 수행하는데, Y<=a가 수행될 때는 Z가 그전 값을 보유하고 Z<=a가 수행될 때는 Y가 그 전값을 보유한다. 그러나 이 경우는 반드시 두 개의 패턴이 필요한 것은 아니다. 분기된 후 수행하는 할당문 이 외의 할당문이 포함하는 변수가 조건오류의 검색에 영향을 미치지 않을 경우는 한 개의 패턴으로 검색이 가능하다. 만약 두 변수가 모두 이 조건오류 검색에 영향을 미친다면 이 경우는 그림 4 (b)와 같이 보유상태를 포함하는 경우로 간주하고 패턴을 생성한다.

(c) 등가오류

그림 4 (c)의 경우는 (a)의 경우와 유사해 보이나, 그 상황이 다를 수 있다. 이 예에서 A≥B를 A<B로 잘못 코딩하였다고 가정하자. 이 예는 A≥B를 만족하느냐에 따라 Y에 A 또는 B를 할당하는 것이다. 즉, A와 B 중 작지 않은 것을 Y에 할당하는 코드로서, 설계자는 Y에 A나 B 이외의 변수를 Y에 할당할 가능성은 거의 없다. 따라서 A≥B의 조건을 잘 못 코드화하여 A<B와 같은 오류를 범할 수 있는 것이다. 이 오류를 범하였을 경우는 그림 4 (d)의 경우에서 보인 바와 같이 조건에 따른 할당을 서로 반대로 한 할당오류와 그 효과가 동일하며, 따라서 검색패턴도 동일하다. 이와 같은 조건오류와 할당오류를 '등가오류(equivalent error)'라 한다. 본 논문에서는 이와 같은 등가오류는 둘 중 하나만을 고려한다.

4. 검색패턴 생성 알고리듬

· 코딩오류에 대한 검색패턴은 앞에서 설명한 할당오류와 조건오류에 대해 따로 수행하며, 그림 5에 전체 알고리듬을 나타내었다. 먼저 VHDL 코드를 검색하여 검색 대상오류들을 추출하고 분류하여 오류목록을 생성한다. 오류목록에 있는 각 오류에 대해 할당오류와

조건오류를 구분하여 따로 검색패턴을 생성한다.

1) 할당오류 검색패턴 생성

먼저 할당오류는 그림 4 (b)와 같이 한 개의 할당오류가 단독적인 영향을 미치는 경우와 그림 4 (a)와 같이 두 개 이상의 할당이 서로 연관된 오류가 있을 수 있다. 단독적인 오류는 그 오류의 결과를 주출력에 나타나도록 하면 되지만, 서로 연관된 할당의 오류는 연관된 할당을 모두 검색하여야 한다. 그 방법에 있어서는 조건문에 의해 정확한 신호흐름을 결정하고 그 경로상에서 주입력과 주출력을 찾아야 한다.

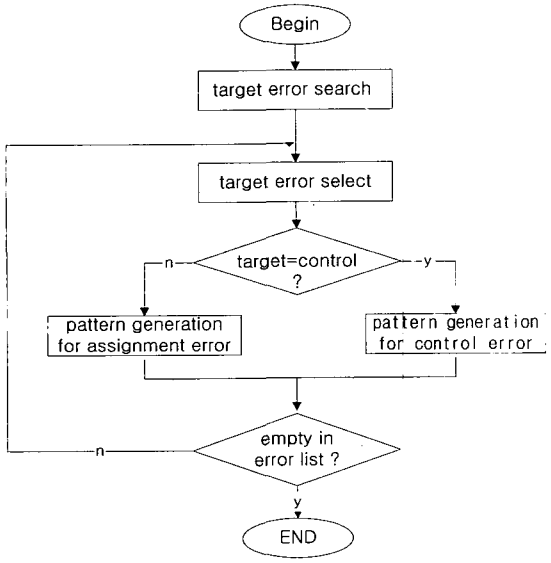


그림 5. 코딩오류 검색패턴 생성 주 알고리듬
Fig. 5. Main algorithm to generate detection patterns of coding errors.

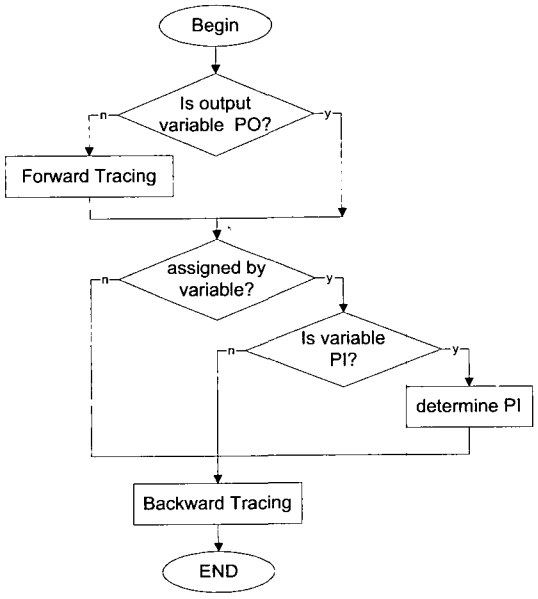


그림 6. 할당오류에 대한 검색패턴 생성
Fig. 6. Detection pattern generation for assignment errors.

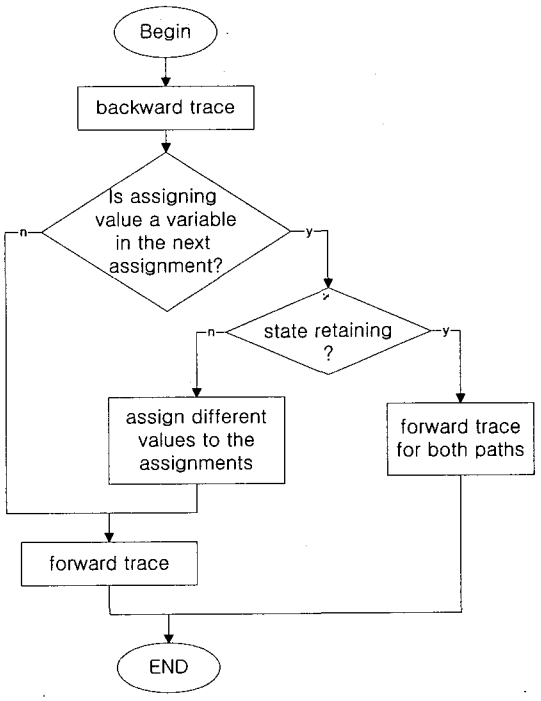


그림 7. 조건오류에 대한 검색패턴 생성
Fig. 7. Detection pattern generation for conditional errors.

그림 6에 나타낸 바와 같이, 두 경우 모두 오류지점으로부터 주출력을 먼저 찾고, 그 후 주입력을 찾는 과정에 따라 검색패턴이 생성된다. 이 때 오류가 발생되었다고 가정한 할당문에 주출력이 포함되어 있으면 그 주출력으로 검색결과가 출력되도록 하며, 그렇지 않은 경우는 신호의 진행방향으로 전진하여 출력변수를 찾는다. 또한 할당되는 값이 변수에 의해 전달되는지와 그 변수가 주입력인지, 또는 상수로 할당되는지에 따라 주입력을 찾는 과정이 결정되도록 하였다. 만약 변수로 할당이 이루어지고 그 변수가 주입력이면, 그 입력에 해당 오류를 유발시키는 것으로 한다. 그러나 변수가 주입력이 아니거나 상수 할당이 이루어지는 경우는 그 오류를 유발시키는 주입력을 찾기 위해 신호흐름의 역방향으로 탐색(역방향 탐색)을 시행한다. 지정된 할당오류에 대해 주출력의 값이 구별되도록 하는 주입력의

값이 결정되면 이 알고리듬은 끝난다.

그림 6에서 첫 번째 판단기호는 정리 1의 내용을 나타낸 것으로, 대상 오류가 주출력에 대한 할당을 포함하면 더 이상의 주출력을 찾는 과정은 필요없고 그 주출력을 여기시키는 주입력만 찾으면 된다. 또한 대상 할당문이 주출력을 포함하지 않는 경우도 일단 오류의 영향을 출력할 수 있는 주출력을 찾으면 그 뒤의 검색은 필요없고 그 주출력을 여기시키는 주입력을 역방향 탐색으로 찾으면 된다.

2) 조건오류 검색패턴 생성

조건오류는 할당오류와는 달리 신호의 흐름을 결정한다. 따라서 검색패턴은 오류에 의한 신호흐름 방향과 오류가 발생하지 않았을 때의 신호흐름 방향을 모두 검색하여야 하며, 서로 다른 결과가 주출력으로 출력되게 하여야 한다.

조건오류에 대한 검색패턴 생성 알고리듬은 그림 7에 나타내었는데, 조건오류는 그 조건을 수행하기 위한 입력조건을 먼저 역방향 탐색으로 찾는다. 그 후 오류가 발생한 조건 다음의 할당문이 변수에 의한 할당인지 상수에 의한 할당인지를 구별하고 상수에 의한 할당인 경우는 그 후의 코드들을 검색하여 주출력까지의 경로를 부각시킨다. 변수에 의한 할당인 경우, 보유상태를 포함하는 형태이면 두 경로를 모두 탐색하여 두 개 이상의 패턴이 필요한지를 판단하고, 그렇지 않는 경우는 분기된 할당문 각각에 서로 다른 값을 할당하여 구별한다. 변수에 의한 할당의 경우도 그 후의 코드를 탐색하여 주출력을 여기시키면 알고리듬을 끝내게 된다.

3) 역방향 탐색 알고리듬

할당오류와 조건오류 모두에서 신호흐름의 방향과 반대방향의 검색이 필요한데, 이런 검색을 역방향 검색이라고 한다. 역방향 검색의 목적은 주입력에 의해 지정된 오류를 야기시키는 것이다. 따라서 역방향 탐색시에는 가능한 한 많은 노드에 영향을 주어 오류의 정보를 가능한 한 많은 주출력으로 전달하도록 하여야 한다.

그림 8에 역방향 탐색 알고리듬을 개략적으로 나타내었다. 역방향 검색의 가장 주된 목적은 대상 오류지점까지의 경로를 탐색하여 그 오류를 부각시키는 입력조건을 찾는 것이다. 그 방법으로는 오류 발생지점으로부터 신호흐름의 역방향으로 한 노드씩 검사하여 필요한 정보를 추출한다. 이 때 이전 노드가 조건에 의한 제어문인지 단순한 할당문인지에 따라 처리를 달리한다. 단순한 할당문일 경우는 그 할당문을 수행하기 위한 변수의 값을 지정한다. 조건문의 경우는 조건문 내의 조건에 부합하는 할당을 수행하도록 변수값을 결정하는데, 변수의 값이 상수로 결정되지 않는 경우 즉, A ≠'00'나 A<'9' 등과 같은 경우 그 조건을 탐색경로를 따라 전달한다. 이 조건은 START를 만나면 해당 변수의 조건을 모두 검색하여 값을 결정한다. 그림에서 보는 바와 같이 이 알고리듬은 VHDL 코드의 시점(CDFG의 START 노드)를 만나면 끝난다. 여기서 START 노드까지 탐색하는 이유는 앞에서 언급한 주입력에 대한 모든 조건을 패턴생성에 포함시키기 위함이다.

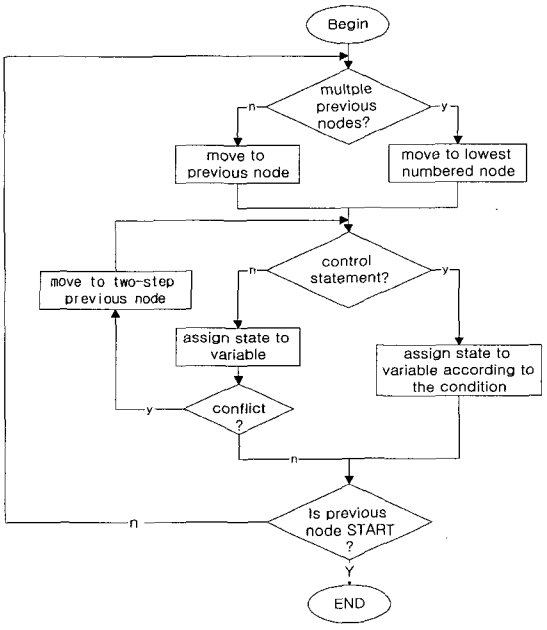


그림 8. 역방향 검색 알고리듬
Fig. 8. Backward Searching algorithm.

4) 정방향 탐색 알고리듬

역방향 탐색과 달리 정방향 탐색은 신호흐름 방향을 따라 탐색하는 알고리듬이다. 일반적으로, 정방향 진행은 조건분기를 만나게 되며, 이 때 신호의 방향은 그 조건에 따라 두 방향 이상으로 분기된다. 이 때 분기된 두 방향 중 한 방향을 선택하여 탐색하는데, 분기에 대한 특별한 조건이 부여되지 않은 경우는 임의로 한 방향을 선택한다.

그림 9에 정방향 탐색 알고리듬을 나타내었는데, 먼저 현재의 코드 다음에 코드가 존재하지 않으면 더 이

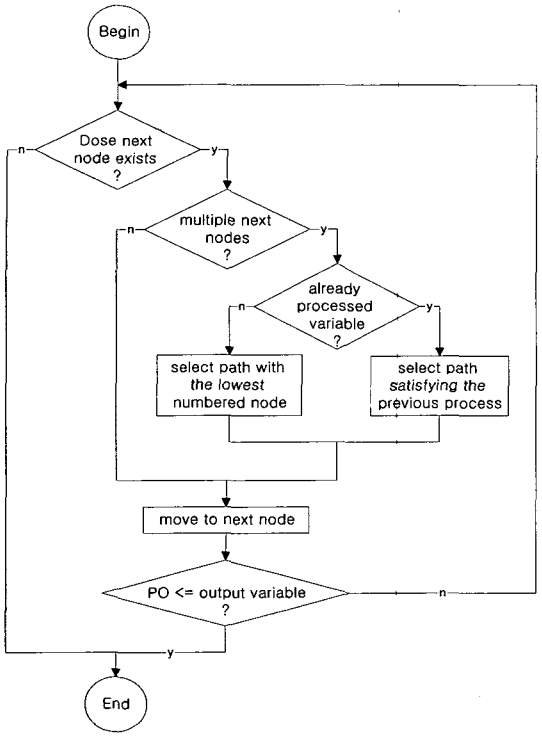


그림 9. 정방향 탐색 알고리듬
Fig. 9. Forward searching algorithm.

상의 정방향 탐색이 불가능하므로 정방향 탐색을 끝낸다. 다음 코드가 있 는 경우 정방향 탐색을 실행하는데, 연결된 다음 노드가 한 개인 경우와 두 개 이상인 경우를 나누어 고려한다. 연결된 노드가 한 개인 경우는 분명 이 오류에 대한 패턴생성 과정에서 처음 만난 것이므로 그냥 그 노드로 이동하여 진행하면 된다. 그러나 두 개 이상의 노드가 연결된 경우는 이 전에 처리를 한 경로가 있을 수 있으므로 미리 처리된 경로가 있는지 확인하고 그 전 과정의 조건에 부합하는지를 확인하여야 한다. 그렇지 않으면 다중 경로 중 임의의 것을 선택하는데, 여기서는 편의상 각 노드에 번호를 부여하고 부여된 번호가 작은 노드를 선택하는 것으로 하였다. 이런 과정을 거친 후 모든 경우에 있어서 다음 노드가 주출력과의 관계가 있는지를 파악하는 과정을 거친다.

이 과정은 주출력을 만나면 끝나게 되는데, 역방향 탐색과는 달리 정방향 탐색은 코드(CDFG)의 끝까지 탐색할 필요는 없다. 단지 오류의 정보를 가시화할 수 있는 주출력을 만나면 알고리듬을 끝낸다.

5. 예제

앞 절에서 설명한 코딩오류 검색패턴 생성방법을 그림 1의 설계를 예로 들어 그 과정을 설명한다. 그림 2의 설계에 대한 CDFG를 참조하면 이 VHDL 코드는 6개의 할당오류와 7개의 조건오류를 가진다. 이 예제의 총 13개의 오류에 대한 목록은 표 2와 같다. 이 표에서 '번호'는 코드번호 즉 오류번호를 나타내며, 이 번호순으로 패턴을 생성한다. 본 절에서는 할당오류와 조건오류 각각 한 개씩만 예를 들어 설명한다.

먼저 할당오류의 예로써, 표 2의 오류목록 중 5번 오류를 고려하자. 할당오류는 그림 6에서와 같이 먼저 피할당 변수가 주출력인지 확인한다. int_cnt가 주출력이 아니므로 그림 9의 정방향 탐색을 통하여 주출력을 찾는다. 가장 먼저 만나는 주출력은 rco로서, 11번 또는 12번 코드에서 만난다. 여기서는 낮은 번호인 11번을 지정하고 그림 8의 역방향 탐색에 의해 주입력의 조건을 찾는다. 이 과정에서 9번 코드에서의 역탐색 방향은 오류발생 지점이므로 5번으로 그 경로를 잡는다. 이 역방향 탐색에서 ent=1, int_cnt=1111, ld_bar=0, clr_bar=1, clk=1이어야 하는데, 그 중 주입력에 대한 검색패턴은 (clk, clr_bar, ld_bar, enp, ent, abcd)=(1, 1, 0, 0, 1, 1111)이며, 이에 대한 정상적인 출력은 (rco, q_abcd)=(1, 1111)이고 오류가 발생한 경우의 출력은 (rco, q_abcd)=(0, 0000)이다.

표 2. 그림 2의 예제에 대한 코딩오류 목록
Table 2. Coding error list for the example in Fig. 2.

#	Kind of error	error
1	condition	clk=1
2	condition	clk_bar=0
3	assignment	int_cnt<=0000
4	condition	ld_bar=0
5	assignment	int_cnt<=abcd
6	condition	enp=1
7	condition	ent=1
8	assignment	int_cnt<=int_cnt+1
9	condition	int_cnt=1111
10	condition	ent=1
11	assignment	rco<=1
12	assignment	rco<=0
13	assignment	q_abcd<=int_cnt

'enp=1'을 조건 할당문으로 갖는 6번 조건문의 코딩오류는 먼저 그 조건문에 이르도록 하는 주입력의 조

건을 역방향 탐색에 의해 찾는다. 이 역방향 탐색에 의해서 ld_bar=0, clr_bar=1, clk=1이 정해진다. 그 다음은 정방향 탐색에 의해 주출력을 찾는다. 6번 코드는 두 개의 경로를 갖고 있고 이 두 경로는 앞에서 설명한 보유상태를 포함하는 경우이다. 따라서 두 개의 패턴이 생성되어 이 조건오류를 검색하여야 한다. 그림 2의 코드는 오류가 없는 코드라고 하였으므로, 이 조건오류는 'enp=0'이라고 코딩한 경우를 생각해 볼 수 있다. 이 오류는 오류가 발생하지 않는 경우와 신호흐름 경로가 달라진다. 따라서 6번 코드의 두 경로 중 어떤 경로를 선택하여도 무방하다. 여기서는 오른쪽 경로를 선택한다. 이 경로를 따라 정방향 탐색을 시행하면 6번 조건분기가 만나는 점이 9번 조건문이라는 것을 알게 되고 이 분기에 의해서는 rco 주출력에서 차이가 나는 것을 알 수 있다. 여기서는 오류가 발생하지 않은 경우의 rco를 1로 잡는 것으로 한다. 이를 위해서는 enp=1, int_cnt-1=1111, ent=1의 조건을 갖게 되는데, 이 중 int_cnt는 앞에서 수행한 역방향 탐색에서 포함되어 있지 않으므로 그 전 패턴에 의해 할당되어야 함을 알 수 있다. 따라서 초기화 패턴으로 clk=1, clr_bar=1, ld_bar=0, abcd=1110의 주입력 조건이 할당된다. 그래서 최종적인 두 패턴은 (clk, clr_bar, ld_bar, enp, ent, abcd)=(1, 1, 0, X, X, 1110)와 (1, 1,1, 1, 1, XXXX)이고 이 두 패턴에 대해 오류가 발생하지 않았을 때는 (rco, q_abcd)=(0, 1110)와 (1, 1111), 오류가 발생하였을 때는 (rco, q_abcd)=(0, 1110)와 (0, 1110)가 출력되어 오류가 검출된다.

그림 2의 코드에 대한 검색패턴 세트를 정상출력과 오류를 가정한 출력을 함께 나타내었다. 이 표에서 보는 바와 같이 가정한 모든 오류를 검색하는 패턴 세트가 생성 가능하며, 모든 오류는 검출가능하다.

Ⅳ. 구현 및 실험

이상에서 설명한 코딩오류 검색패턴 생성 알고리듬은 C-언어로 구현되었으며, 여러 가지 VHDL 설계를 대상으로 실험을 수행하였다. 실험한 환경은 팬티엄-II 프로세서를 사용하는 컴퓨터였으며, 프로세서의 클럭 주파수는 400MHz였다.

실험한 결과를 요약하여 표 4에 나타내었다. 이 표에서 '# of lines'는 VHDL 코드의 line 수를 나타내고 '# of errors'와 '# of patterns'는 가정한 코딩오류의 수와 그 코딩오류를 검색하는 검색패턴의 수를 각각 나타낸다. 이 표에서 보는 바와 같이 실험한 모든 설계의 모든 대상오류에 대해 검색패턴을 생성할 수 있었으며 검출률 또한 모두 100%를 나타내었다. 표에서 오류의 수보다 패턴 수가 많은 경우는 보유상태를 포함한 코드들이 존재함을 의미하고, 오류 수보다 패턴수가 적은

표 3. 그림 2의 예제에 대한 검색패턴 세트 및 출력
Table 3. Verification pattern set and outputs for the example in Fig. 2.

target	PI						PO (Correct)		PO (error)	
	clk	clr_bar	ld_bar	enp	ent	abcd	rco	q_abcd	rco	q_abcd
1	1	1	0	1	0	1111	1	1111	X	XXXX
	0	1	0	1	1	0000	1	1111	0	0000
2	1	1	0	0	1	1010	1	1111	0	0000
3	1	0	1	1	1	1111	0	0000	1	1111
4	1	1	1	1	1	0000	1	1111	0	0000
5	1	1	0	0	1	1111	1	1111	0	0000
6	1	1	1	0	X	1110	0	1110	0	1110
	1	1	1	1	1	XXXX	1	1111	0	1110
7	1	1	1	1	1	1111	1	1111	0	0000
	1	1	1	1	0	1111	1	1111	0	0000
8	1	1	1	1	1	0101	1	1111	0	0000
9	1	0	1	1	0	1111	0	0000	1	1111
10	1	1	0	0	1	1111	1	1111	0	0000
11	1	0	1	0	1	1111	1	1111	0	0000
12	1	0	0	1	1	0000	0	0000	1	1111
13	1	1	0	1	1	1111	1	1111	0	0000

경우는 등가오류를 포함하고 있음을 의미한다.

또한 검색패턴을 생성하는데 걸린 시간은 몬든 경우에서 1초 미만의 시간이 소요되었는데, 이것은 C-언어에 의해 CPU 시간의 측정을 초 단위로 수행하였기 때문이다. 기존 논문에서 본 논문과 유사한 내용의 아직 없어서 절대적인 비교는 불가능하지만, 그 중 가장 유사한 논문이 [13]이다. 이 논문에서는 C-언어로 구현하고 Sparc Station I의 환경에서 실험을 하였는데, 4개의 경로를 갖는 가산기의 검색패턴을 생성하는데 걸린 CPU 시간은 12초였다. 앞장에서 예제로 들은 VHDL 코드는 표 4의 '74163(4-bit counter)'로, 그림 2 (b)에서 보는 바와 같이 10개의 경로를 포함하고 있다. 실행환경이 서로 다르지만 상대적으로 볼 때 본 논문의 방법이 훨씬 빠른 계산속도를 갖고 있음을 알 수 있다.

표 4. 제안한 알고리듬을 여러적용한 결과
Table 4. Results from applying the proposed algorithm to various circuits.

Circuit name	# of Lines	# of Errors	# of Patterns	CPU Time[sec]
8×1 Multiplexer	32	16	16	<1
16 bit latch	22	13	12	<1
8 bit ram	29	8	8	<1
8×3 encoder	29	17	17	<1
8bit comparator	11	3	3	<1
3×6 decoder	24	15	15	<1
8bit register	28	7	9	<1
74163(4bit counter)	30	13	16	<1
ALU	34	9	9	<1

V. 결 론

본 논문에서는 VHDL 행위-레벨 설계를 대상으로 컴파일 과정에서 발견되지 않는 코딩오류를 검출할 수 있는 방법을 제안하였다. 본 논문에서는 검색패턴을 생성하여 gold unit의 응답과 VHDL 설계의 응답을 비교하는 방법을 택하였으며, 검색패턴 생성 알고리듬을 같이 제안하였다.

제안한 알고리듬은 C-언어로 구현하였으며, 여러 가지 대상 VHDL 설계에 적용한 결과 모든 설계의 모든 가정한 오류에 대해 검색패턴을 생성할 수 있었으며, 검출률 또한 모두 100%를 보여 매우 우수한 검색능력을 갖고 있음이 입증되었다. 검색패턴을 생성하는데 소요되는 CPU 시간 또한 실험 대상 설계 전체에서 1초 미만의 시간을 보여 검색패턴 생성 속도면에서도 매우 우수함을 보였다. 따라서 본 논문에서 제안한 VHDL 행위-레벨 설계의 코딩오류 검색 방법은 VHDL 행위-레벨 설계의 사용 빈도가 증가함에 따라 매우 유용한 설계검증 툴이 될 것으로 기대된다.

본 논문에 이은 향후 연구과제로는 코딩오류의 위치를 자동적으로 검색하는 방법이다. 많은 경우 서로 다른 코딩오류에 대한 검색패턴이 동일할 수 있으므로 검색패턴만으로는 오류의 위치를 정확히 파악할 수 없다. 따라서 출력응답을 같이 고려하여야 하며, 현재 본 연구실에서 이에 대한 연구방법을 구상중이다. 오류위치를 자동적으로 파악하는 툴이 개발되면 설계검증 및 수정의 시간이 한층 단축될 수 있을 것으로 기대된다.

참 고 문 헌

[1] *IEEE Standard VHDL Language Reference Manual*, IEEE, NY, 1987.

[2] 손진우 외 3명, "고속 ASIC 설계기술 동향", 한국전자통신기술연구소 주간기술동향, 제 745호, pp.1-11, 1996년 5월

[3] 유영욱, "System IC의 기술 동향", 대한전자공학회 학회지, 제25권, 제5호, pp.416-424, 1998. 5

[4] IEEE, *IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language*, IEEE Computer Society, 1995.

[5] K. Skahill, "프로그래머블 로직을 위한 VHDL 디자인의 최적화", Electronic Engineering, pp.58-64, 1998년 3월

[6] J. R. Burch, et al., "Symbolic Model Checking for Sequential Circuit Verification", IEEE Trans. CAD, Vol.13, No.4, pp.401-424, April 1994.

[7] M. C. McFarland, "Formal Verification of Sequential Hardware: a Tutorial", IEEE Trans. CAD, Vol.12, Vo.5, pp.633-654, May 1993.

[8] W. Ecker, "Verification Methods for VHDL RTL-Subroutines", J. Systems Architecture, Vol. 42, pp.117-128, 1996.

[9] Formality, Synopsys, 1998.

[10] Affirma, Cadence, 1998.

[11] Y. V. Hoskote, et al., "Automated Verification of Implementations of Large Circuits Against HDL Specifications", IEEE Trans. CAD, Vol.16, No.3, pp.217-228, March 1997.

[12] Fazan Fallah, Srinivas Devadas, and Kurt Keutzer, "OCCOM: Efficient Computation of Observability-Based Code Coverage Materics for Functional Verification", IEEE DAC'98, pp. 152-157, 1998.

[13] R. Vemuri and R. Kalyanaraman, "Generation of Design Verification Tests from Behavior VHDL Programs using Path Enumeration and Constraint Programming", IEEE Trans. on VLSI Systems, Vol. 3, No. 2, pp. 201-214, June 1995.

[14] J. P. Roth, "Diagnosis of Automata Failures: A Calculus and a Metho", IBM J. Res. Dev., Vol.10, No.4, pp.278-291, July 1966.

[15] C. Benmehrez and J. F. McDonald, "Measured Performance of a Programmed Implementation of the Subscribed D-Algorithm", Proc. 20th DA Conf., pp.308-315, 1983.

[16] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits", IEEE Trans. Computers, Vol.C-20, No.12, pp.1496-1506, Dec.1971.

[17] J. J. Thomas, "Automated Diagnostic Test Programs for Digital Networks", Computer Designs, pp.63-67, Aug. 1971.

저 자 소 개

金 東 郁(正會員)

1960년 8월 23일생. 1983년 2월 한양대학교 전자공학과 졸업(공학사). 1985년 2월 한양대학교 대학원 졸업(공학석사). 1991년 9월 Georgia 공과대학 전기공학과 졸업(공학박사). 1992년 3월~현재 광운대학교 전자재료공학과 부교수. 광운대학교 신기술연구소 연구원. 1997년 12월~현재 광운대학교 IDEC 운영위원. 2000년 3월~현재 인티스닷컴(주) 연구원. 주관심분야는 디지털 VLSI Testability, VLSI CAD, DSP 설계

朴 承 奎(正會員)

1974년 10월 4일생, 1998년 2월 광운대학교 전자재료공학과 학사, 2000년 2월 광운대학교 전자재료공학과 석사, 2000년 1월 ~ 현재 (주) 제이슨테크 연구원. 주관심 분야 VHDL, Design verification, VLSI Testability

金 宗 賢(正會員)

1973년 4월 22일생. 1997년 2월 광운대학교 전자재료공학과 졸업(공학사). 1999년 2월 광운대학교 전자재료공학과 졸업(공학석사). 2001년 2월 광운대학교 전자재료공학과 박사수료. 2000년 3월~2001년 2월 인티스닷컴(주) 연구원. 주관심분야는 VHDL, VLSI Testability, DSP, Design verification

徐 英 鎬(正會員)

1975년 5월 30일생, 1992년 광운대학교 전자재료공학과 학사, 2001년 2월 광운대학교 전자재료공학과 석사, 2001년 3월 ~ 현재 광운대학교 전자재료공학과 박사과정, 2000년 3월 ~ 현재 인티스닷컴(주) 연구원. 주관심분야는 FPGA, ASIC design, wavelet codec, DFT, cryptosystem