VHDL 모델의 상위레벨고장 검출방법

김강철*
*여수대학교 컴퓨터공학과

New Fault-detection Methodology of high-level event in VHDL models

Kang Chul Kim*
*Yosu National University
E-mail : kkc@yosu.ac.kr

요 약

본 연구에서는 HDL에서 블록 사이, 프로세스문장, 또는 일을 할당하는 순서를 조절하는 상위레벨 사건를 하위레벨 사건과 비교하여 정의하며, 상위레벨 사건은 자원충돌과 프로토콜 또는 사양의존 충돌로 구성된다는 것을 보여준다. 그리고 상위레벨 사건을 검출하기 위하여 2가지 검출방법이 제안된다.

ABSTRACT

In this paper, high-level events that adjust or control the conflicts between blocks or process statement, or job sequences are defined compared to low-level ents. This paper proposes that high-level events consist of resources conflicts and protocol or specification-dependent conflicts, and two low-level coverage metrics can be used to defect high-level events.

키워드
검증, 검출율 측정법, 상위레벨 고장, 문장 검출율 측정법, 경로 검출율 측정법

Ⅰ. 서 론

설계(design)를 검증(verification)하는 방법에는 정확성의 정형 증명법(formal proof)과 모의실험을 이용하는 방법이 있다. 모의실험은 설계가 100% 검증되었다는 것을 보증할 수는 없지만 아직까지 모든 설계 단계에서 사용되고 있다.[1-3] 그러나 모의실험은 집적도가 커짐에 따라 속도가 너무 느리고, 더 큰 모의실험 모델과 많은 컴퓨팅 자원이 필요하다는 문제점을 가지고 있다. 정형 증명법은 설계의 정확성을 수학적으로 증명하는 것으로 크게 동가검증(equivalence checking)과 모델검증(model checking) 두 부류로 나눌 수 있다.[4][5] 이러한 방법들은 최근에는 전체 IC의 검증에 대하여 사용되기도 하지만 아직까지는 큰 회로에 대해서는 불가능하기 때문에 블록레벨 설계 검증에 제한되고 있는 실정이다. 그러므로 설계 전체 과정에 대해서는 모의실험이 유일한 검증 수단되고 있다.

VHDL 언어는 동시에 수행되는 많은 프로세스를 가지고 있는 복잡한 병렬처리언어(concurrent language)이며, 이를 사용하여 검증된 설계가 정확하다고 증명하는 것은 불가능하다. 그리고 테스트벤치(testbench)를 사용하여 성공적으로 시뮬레이션을 하였다해도 검증자가 알지 못하는 다른 고장들이 존재할지도 모른다. 따라서 이러한 문제점을 해결하기 위한 방법으로 소프트에어 검증에 사용되던 코드 검출율 측정법(code coverage metric))을 빌려와서 VHDL 프로그램의 정확성을 검증할 때에도 사용하고 있다.

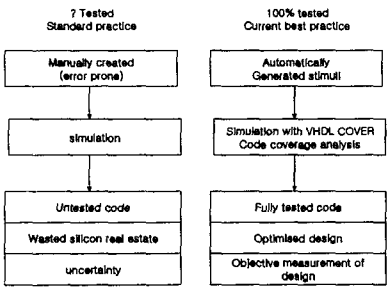
최근에 칩의 제조기술의 급격한 발전은 시스템을 하나의 칩으로 구현하는 새로운 하드웨어 설계 방법을 제시하고 있지만, 마이크로프로세서, 공유메모리의 캐쉬제어기, 버스제어기, SoC 등의 복잡한 칩의 설계와 검증은 제조기술을 따라가지 못하고 있는 실정이다. 시스템 레벨에서 칩을 완전하게 검증하기 위하여 여러 가지 검증 방법들이 개발되고 있다. [6]은 모토롤라의 G4 세대 마이크로프로세서에서 사용될 다중처리(multiprocessing)의 설계검증 방법을 개발한 것으로 시뮬레이션 방법을 사용하여 설계 시작부터 각 불럭을 시뮬리이션하여 다중 불럭에서도 재사용이 가능하도록 하였고, 최종적으로 칩이나 시스템 레벨에서도 검증된 블

록을 사용하여 설계비용과 시간을 줄이고자 하였다. [7]은 캐쉬일관성을 검증하기 위하여 function level까지를 고려한 사양(specification)을 개발하여 정형검증법을 사용하였다. [8]은 복잡한 파이프라인을 검증하기 위하여 BDD(binary decision diagram)를 사용하여 테스트 프로그램 생성 툴을 개발하였으며, 파이프라인과 관련된 98%의 고장을 검출하여 RTL 검증효과를 증가시켰다. 그러나 이 방법들은 개발된 IC나 시스템에만 사용이 가능하고, 다른 시스템에 적용시키기 어려워 일반화된 검증기법으로 사용하기 어려운 단점을 가지고 있다.

국내에서도 VHDL 모델의 검증에 관한 연구가 수행되고 있다. [9]에서는 SoC 설계에 관점을 두고 IP의 검증에 관한 논문을 발표하고 있으며, [10]에서는 정형 증명법을 사용하여 IC의 검증에 관련된 논문을 발표하고 있다. [11]에서는 코드 검출율 측정법을 사용하여 IC의 검증 방법을 연구하고 있다.

II. 기존의 하위레벨 고장 검출방법

VHDL은 복잡한 병렬처리 언어이므로 소프트웨어 검증에 사용되던 코드 검출율 측정법을 빌려와서 VHDL 언어의 정확성을 검사할 때 하드웨어 설계에도 사용되고 있다. <그림 1>은 코드 검출율 측정법을 수행한 설계과정과 그렇지 않은 설계과정의 차이를 보여주는 것으로 시간과 설계비용이 훨씬 줄일 수 있다는 것을 예측할 수 있다. [12]



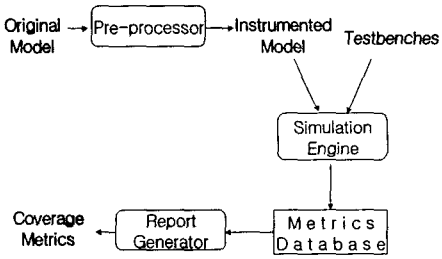
<그림 1> 코드 검출율 측정법 과정이 있는 경우와 없는 경우의 설계과정

<표 1>은 많이 사용되는 코드범위검사법을 요약한 것이다. 여기서 문장, 분기, 경로, 조건 검출율 측정법 등이 많이 사용된다. 문장 검출율 측정법은 얼마나 많은 신호(signal)와 변수(variable)가 시뮬레이션 동안에 활성화되는 가를 측정하는 것이다. 분기 검출율 측정법은 IF와 CASE 문장이 얼마나 많이 활성화되는 가를 측정하는 것이며, 경로 검출율 측정법은 코드의 순차적인 경로를 계산하여 얼마나 많은 경로가 테스트되는 가를 검사하는 방법이다.

<표 1> 코드 검출율 측정법의 종류

Metric	Description
Statement	All statemenrts should be executed
Branch	All branched should be visited
Condition	All branch conditions should be exercised
Path	How many sequences through branchesare executed
Toggle	All signals' bit should change states
Trigger	All signals in the processes' sensitivity listd are activated
State	All possible states in the state machine are visited
Arc	How many transitions have been made between states
Expression	State transition controlling expressions are tested
Aequence	How many sequences of states are executed

<그림 2>는 코드 검출율 측정법의 과정을 보여준다. 먼저 소스코드가 인가되고, 특정한 구문이 수행되는 가를 기록하기 위하여 소스코드의 검사하고자 하는 위치에 체크포인트를 첨가한다. 체크포인트가 포함된 코드는 가능한 모든 테스트벤치를 사용하여 시뮬레이션된다. 시뮬레이션으로부터 얻어진 데이터는 데이터베이스에 저장되고, 설계에 대한 여러 가지 코드 검출율 측정법을 결정하기 위하여 기록된다.



<그림 2> 코드범위검사 과정

칩의 제조기술의 급격한 발전은 시스템을 하나의 칩으로 구현하는 새로운 하드웨어 설계방법을 제시하고 있지만, 마이크로프로세서, 공유메모리의 캐쉬제어기, 버스제어기, SoC 등의 복잡한 칩의 설계와 검증은 제조기술을 따라가지 못하고 있는 실정이다. 시스템 레벨에서 칩을 완전하게 검증하기 위하여 여러 가지 검증 방법들이 개발되고 있다.

III. 상위레벨 고장검출 방법

3.1 상위레벨 사건과 고장의 정의 및 사례

상위레벨 사건은 크게 2가지로 분류될 수 있다. 먼저 HDL에서 블록간 또는 process 문장 사이의 상호작용이고, 두 번째는 한 작업을 수행하는 일련의 알고리즘 또는 프로토콜로 정의한다.

첫 번째 예제는 버스에서의 충돌이다. 두개의 소자가 동시에 버스를 사용할려고 할 경우 충돌이 발

생하며, 이러한 고장이 상위레벨 고장이 될 수 있다. 두 번째 예제는 파이프라인 프로세서에서 데이터해저드(data hazard)이다.[13] 데이터 해저는 데이터 충돌(conflict)와 분기 충돌(branch conflict)로 분류된다. 데이터 충돌은 소프트웨어나 하드웨어적으로 해결이 가능하다. NOP 명령어나 명령어의 순서를 바꿔서 소프트웨어적인 방법으로 해결할 수 있으며, 명령어의 수행을 지연(stall)시키거나 데이터를 미리준비(forwarding)하여 하드웨어적으로 해결할 수 있다. 분기 충돌은 조건분기와 무조건분기로 나누어질 수 있다. 무조건 분기의 충돌은 NOP의 삽입, 명령어의 수행순서 변경,또는 지연를 삽입하여 해결이 가능하며, 조건분기의 충돌은 조건에 의존하녀며, 무조건분기에서 사용하는 방법 외에 폐기(annulling)와 예측(prediction)를 사용할 수 있다.

세 번째 예제는 공유 메모리에서 캐쉬 일관성(cache coherence) 문제이다. 다중 프로세서는 각 프로세서에 각각의 캐쉬를 가지고 있으며, 2개 이사의 캐쉬가 같은 메모리 위치에 있는 값을 동시에 가지고 있을 수 있으며, 이들 중의 하나가 바뀌면 두 값은 서로 다르게 되어 캐쉬의 일관성이 사라지게 된다. Write-through 기법은 주기억장치를 갱신할 수 있으나, 다른 캐쉬의 값을 바꿀 수 없으므로 캐쉬 일관성 문제를 해결할 수 없다. 그러나 캐쉬 불가능(non cacheable), 캐쉬 디렉토리(cache directory), 캐쉬감시(snooping) 방법은 캐쉬 일관성 문제를 해결할 수 있다. 캐쉬불가능에서는 컴파일 동안에 컴파일러가 모든 공유데이터를 캐쉬불가능으로 표시할 수 있으며, 모든 데이터의 접근을 공유메모리로부터 가능하게 한다. 그러나 이 방법은 캐쉬 성공률이 낮아서 전체 시스템의 성능을 떨어뜨린다. 캐쉬 디렉토리는 각 메모리 블록의 상태를 기록하며, 어느 캐쉬가 메모리 블록의 내용을 가지고 있는가에 대한 정보를 가지고 있는 디렉토리를 구현하는 것이다. 캐쉬감시에서는 각 캐쉬가 시스템 버스 상에서 메모리 활동을 감시하는 것으로 많이 사용되고 있는 프로토콜이다.

다중 캐쉬 시스템에는 데이터를 표시하고 조작하는 많은 방법이 있으며, 이중에서 MESI가 가장 많이 사용되는 방법이다. MESI 프로토콜은 modified, exclusive, shared, invalid 4 개의 상태가 있으며, 각각은 읽기성공(read hit), 일기실패(read miss), 쓰기성공(write hit)과 쓰기실패(write miss)이다.

3.2 상위레벨 사건의 분류

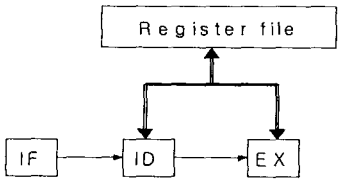
상위레벨 사건을 검출하기 위한 방법을 제안하기 위하여 상위레벨 사건의 종류를 분류할 필요가 있다. 대부분의 상위레벨 사건은 프로토콜에 관련되거나 시스템의 사양에 의존하게 된다. 본 연구에서는 크게 두 부류로 나눈다. 하나는 자원충돌에 관한 사건이고, 다른 하나는 프로토콜 또는 사양에 관련된 사건이다. 자원 충돌에 관련된 고장들은 파이프라인을 사용할 경우 데이터 충돌, 버스 중재기(arbiter)에서 버스 충돌, 병렬처리 시스템에서 자원 충돌, VHDL 프로그램의 병렬처리 문장에서의 자원 충돌이다. 프로토콜, 알고리즘 또는 어떤 작업을 수행하는 일련의 순서에 관련되는 고장은 분기 충돌, 캐쉬일관성, 메모리 접근, 명령어를 수행하는 일련의 순서 등이다.

3.3 코드 검출율 측정법을 이용한 상위레벨 사건 검출율 측정법

자원 충돌에 관련된 고장으로 블록이나 프로세스 문 사이의 데이터 이동은 전역 신호나 변수에 의해 수행되므로 전역신호나 변수가 테스트벤치 프로그램에 플래그(flag)를 가지고 있으면 문장 검출율 측정법에 의하여 검출이 가능하다.

VHDL 프로그램이 시뮬레이션되는 동안 신호나 변수가 입력 또는 출력으로 사용된다는 것을 알고 있으면 자원의 충돌을 인지할 수 있다. 그러나 프로토콜이나 알고리즘 관련 충돌은 검출이 쉽지 않지만, 데이터 흐름 또는 경로 검출율 측정법을 사용하면 알고리즘을 수행하는 일련의 과정들을 추적할 수 있으므로 고장이 검출이 가능하다.

아래 예제는 문장 검출율 측정법에 의하여 데이터 충돌이 검출될 수 있다는 것을 보여준다. <그림 3>과 같이 명령어 페취(IF), 디코딩(ID), 실행(EX)의 3 단계 파이프라인을 가진 프로세서를 가정하자.

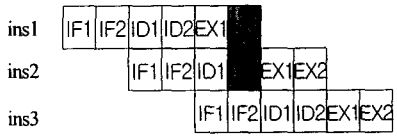


<그림 3> 3단계 파이프라인

주소 M이 IF 단계의 첫 번째 반주기 동안에 메모리로 보내지고, 주소 M에 있는 데이터가 다음 반주기 동안에 명령어 레지스터(IR)에 저장된다. ID 단계의 첫 번째 반주기 동안에 명령어를 해석하고 나머지 반주기 동안에 오퍼랜드를 가져온다. EX의 처음 반주기 동안에는 명령을 실행하고, 나머지 반주기 동안에 연산결과를 저장한다. <그림 4>의 (a)는 이 프로세서에서 수행되는 3개의 명령어 순서를 나타낸 것으로 처음 두 명령어를 고려해 보자. ins1은 R2와 R3의 내용을 합하여 결과를 R1에 쓰고, ins2는 ins1과 같지만 ins1의 결과가 R1에 저장되기 전에 R1을 사용한다. 데이터 충돌을 해결하기 위한 아무런 조치를 취하지 않으며 그림 (b)에서와 같이 ins1의 EX 단계와 ins2의 ID 단계에서 충돌이 발생한다.

ins1 : add R1, R2, R3

ins2 : add R4, R1, R5
ins3 : sub R6, R7, R8
(a) 예제 프로그램



(b) EX 단계와 ID 단계의 데이터 충돌
<그림 4> 데이터 충돌 예제

<그림 5>는 <그림 3>을 HDL 언로로 표현 의사코드와 테스트벤치 프로그램이다. R1은 id, if, regfile 블럭에서 동시에 사용되면서 서로 데이터를 주고 받아야 되므로 R1은 전역신호나 변수로 선언되어야 한다. 테스트벤치 프로그램에서 R1 신호에 r와 w 플래그가 포함되어 있다고 가정하자. r 플래그는 ID 단계에서 오퍼랜드 R1을 페취할 때 1이 되며, w 플래그는 EX 단계에서 R1에 값을 쓸 때 1이 된다. <그림 5>의 (b)와 같은 조건이 발생하면 R1에서 데이터 충돌이 발생함을 알 수 있다.

```
global signal R1;
regfile : process()
id : process();
ex : process()
```
(a) HDL 의사코드

```
R1 with r an two w flags;
if((r=1 and (w1=1 or w2=1)) or (w=1 and w2=1)
        conflict;
```
(b) 테스트벤치 프로그램
<그림 5> 의사코드 및 테스트벤치 프로그램

브랜치 예측이나 snoopy 프로토콜과 같은 알고리즘 관련 사건은 상태 다이어그램(state diagram)으로 표현된다. 각 상태는 HDL 언어에서 IF 또는 CASE 문장으로 표현되므로 데이터 흐름 검출율 측정법이나 경로 검출율 측정법에 의하여 검출이 가능할 것이다.

IV. 결 론

본 연구에서는 HDL에서 블록 사이, 프로세스문장, 또는 일을 할당하는 순서를 조절하는 상위레벨 사건를 하위레벨 사건과 비교하여 정의하였으며, 상위레벨 사건은 자원충돌과 프로토콜 또는 사양의존 충돌로 구성된다는 것을 보여주었다. 그리고 상위레벨 사건을 검출하기 위하여 2가지 검출방법을 제안하였다. 전역신호 또는 변수가 테스트벤치 프로그램에서 플래그(flag)를 가지고 있으면 하위레벨 사건 검출율 측정법인 문장 검출율 측정법으로 상위레벨 사건인 자원충돌 사건을 검출할 수 있었으며, 데이터 흐름 검출율 측정법 또는 경로 검출율 측정법으로 프로토콜 또는 설계사양에 관련된 고장을 검출할 수 있다는 것을 보여주었다.

참고문헌

[1] Jim Lipman, "Covering your HDL chip-design bets", EDN, pp 65-74. Oct. 1998.

[2] Brian Barrera, "Code coverage analysis-essential to a safe design", Electronic Engineering, pp 41-43, Nov. 1998.

[3] Janick Bergeron, Writing Testbenches : Functional Verification of HDL models, 2nd edition, Kluwer Academic Publishers, 2003

[4] Rolf Drechsler and Bernd Becker, Binary Design Diagrams : Theory and Implementation, Kluwer Academic Publishers, 1998

[5] Edmund M. Clarke, Orna Grumberg, Doron A. Peled, Model Checking, MIT Press, 2000.

[6] Jen-Tien Yen and Qichao Richard Yin, "Multiprocessing Design Verification Methodology for Motorola MPC74XX PowerPC Microprocessor," DAC, pp 718-723, 2000.

[7] Cindy Eisner, et al, "A Methodology for Formal Design of Hardware Control with Application to Cache Coherence Protocols," DAC,, pp 724-729, 2000.

[8] Kazuyoshi Kohno, Nobu Matsumoto, "A New Verification Methology for Complex Pipeline Behavior," DAC, pp. 816-821, 2001.

[9] Wooseung Yang,Moo-Kyeong Chung and ChongpMin Kyung, "Current Status and Challenges of Soc Verification for Embedded Systems Market," IEEE, pp. 213-216, 2003.

[10] Hoon Choi, Byeongwheee Yun, Yuntae Lee, and Hyunglae Roh, "Model Checking of S3C2400X Industrial Embedded SoC Product," DAC, pp. 611- 616. 2001

[11] 김명균 외 3인, "지연고장 테스팅에 대한 고장 검출율 메트릭", 전자공학회 논문지 제38권 SD 제 4호, pp. 266-276, 2001.

[12] Editor, "Software Techniques applied to Vhdl design", New Electrons, May 1995.

[13] John D. Carpineli, Computer Systems, Organization & Architecturre, Addison Wesley, 2001.