동작적 모델 검증의 상위레벨 사건에 대한 검출률 측정법

김강철* · 임창균* · 류재홍* · 한석붕**

Coverage metrics for high-level events in behavioral model verification

Kim, Kang Chul · Im, Chang Gyun · Ryu, Jae Hung · Han, Suk Bung

본 연구는 전남테크노파크 지원으로 수행되었음

요 약

최근에 CAD 툴의 비약적인 발전으로 인하여 대부분의 디지털 회로들은 VHDL 언어를 사용하여 설계된다. 그리고 IC 공정기술의 발달에 따라 하나의 칩에 많은 회로를 포함할 수 있으므로 VHDL 코드의 크기가 방대해져 이에 대한 검증(verification)은 칩 설계에 있어서 어렵고, 많은 시간을 소모하는 과정이 되고 있다. 본 연구에서는 SoC 용 IP 사이에서 발생할 수 있는 자원충돌과 프로토콜의 오류를 검증하는 새로운 방법을 제시한다. VHDL 모델의 블록 또는 SoC용 IP 사이에서 발생할 수 있는 상위레벨 고장을 정의하고 분류하고, 하위 레벨 검증(low-level code verification)에 사용되는 검출률 측정법을 사용하여 IP 사이에서 발생하는 데이터 충돌과 프로토콜 또는 알고리즘의 오류를 검증하는 방법을 제안한다.

ABSTRACT

The complexity of IC has rapidly increased as VLSI fabrication technology has grown up quickly. This paper proposes verification methods for data conflicts and protocol between IPs for SoC with coverage metrics. The high-level events is defined to cooperation between blocks or process statement in HDL, or a sequence of performing a job compared to low-level event. They are classified into two categories, resource conflicts and protocol or specification-dependent conflicts. And two coverage metrics used for code coverage in low-level event are proposed to verify the high-level events. The events of resource conflicts can be detected by using statement coverage metric if global signal or variable has flags in a testbench program, and protocol-dependent events can be checked by data flow metric or path metric.

키워드

Coverage metric, Low-level event, High-level event, SOC, IP

* 전남대학교 공학대학 컴퓨터공학과

** 경상대학교 공과대학 전자공학과

접수일자 : 2006. 1. 11

Ⅰ. Introduction

As VLSI technology has been developed, the complexity of IC has rapidly been increased, so design engineers and verification engineers work together from the beginning of IC design[1,2,3]. Verification is often confused with testing. Testing verifies that the design was manufactured correctly. Verification is to ensure that a design meets its functional intent, and it is a process used to demonstrate the functional correctness of a design[4,5]. Verification of complex behavioral models has become a critical and time-consuming process in hardware design. Verification consumes about 70% of the design effort in SoC(system on chip), IP(intellectual property), ASIC[6,7,8,9].

Design verification is considered one of the most serious bottleneck for multimillion-gate chip design. Two mechanisms to help achieve design verification are formal proofs of correctness and thorough simulation. Simulation can be applied to all levels of a design(from unit to chip) even though there is no guarantee that the design is 100% verified. It has problems that simulation speed is slow and it needs bigger simulation model and a lot of computing resources in simulation environment. Formal verification mathematically proves the correctness of a design and falls under two broad categories, Equivalence checkingand Model checking [10,11,12]. But it is limited to block level design verification due to its inability in handling large circuitry, therefore, simulation is the only practical means for all levels of a design

When VHDL code is simulated, a testbench program is used.[8] It is the code used to create a pre-determined input sequence to a design, then optionally observe the response, and it is commonly implemented using VHDL or Verilog and may also include external data files or C routines. The test bench will be used to apply input stimuli to the design description to observe the outputs and compare them to expected results. The advantage of creating a test bench versus simulating the design interactively using the command language of a given simulator is that it allows to document the test bench in a human readable form[13].

In this paper, the high-level events is definedto cooperation between blocks or process statement in HDL, or a sequence of performing a job compared to low-level event. They are classified into two categories, resource conflicts and protocol or specification-dependent conflicts. And two coverage metrics used for code coverage in low-level event are proposed to verify the high-level events. The events of resource conflicts can be detected by using statement coverage metric if global signal or variable has flags in testbench program, and protocol-dependent event can be checked by data flow metric or path metric.

In section 2, previous works on low-level event and coverage metrics are described. In section 3, the terminology of high-level events is defined, and the classification of high-level event compared to low-level eventis explained in section 4. Two proposed coverage metrics for high-level event coverage metrics are explained in section 5. Finally, conclusion is describe in chapter 6.

Ⅱ. Previous work

It is impossible to know that the design being verified is indeed functionally correct, with 100% certainly. A popular and precisely defined metrics in the software world is code coverage[14][15]. Code coverage comes in a range of forms. Code coverage is a tool that can identify what code has been executed in the design under verification[8]. VHDL is a complex concurrent language with a typical model having many processes all conceptually executing at the time. So coverage metrics defined in software have been borrowed and used in hardware design when the correctness of VHDL Program is checked. <Fig. 1> shows that design flow with coverage check can reduce time and cost of design[15].

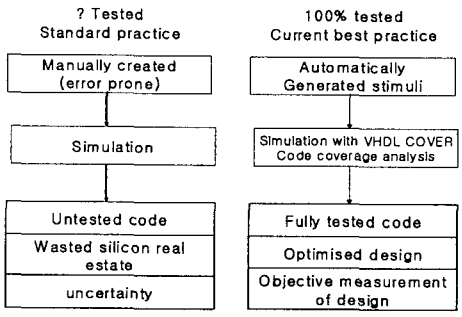


그림 1. 코드검출률 측정장비의 유무에 따른 설계과정
<Fig. 1> Design flow with and without code coverage tools

표 1. 코드검출률 측정법의 종류
<Table 1> Code coverage metrics

Metric	Description
Branch	All branches should be visited
Condition	All branch conditions should be exercised
Path	How many sequences through branches are executed
Toggle	All signals' bit should change states
Trigger	All signals in the processes' sensitivity lists 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
Sequence	How many sequences of states are executed

<Table 1> shows most popular code metrics [9,13,16,17,18]. Statement metricmeasures how many signal and variable assignments have been activated during simulation. And branch metric measures how many branches of IF and CASE statement have been activated. Path metric calculates the sequential paths through the code, and measures how many of the code paths have been tested.

```
1 if RST='0' then
2  COUNT='0'
3  CNT='0'
  end
4 SR(0) <= SR(1);
5 SR(1) <=SR(2);
6 if(clk'event and clk='1') then
7  CNT <= not(CNT);
8  DOUT <= DIN
  end;
```

그림 2. 검출률 측정법을 위한 예제 프로그램
<Fig. 2> Example VHDL program for coverage metrics.

<Fig. 2> is a program for coverage metrics with 4 paths, depending on if conditions are true or false. If both conditions are false, only 2 statements are exercised, but if both conditions are true, 8 statements are exercised.

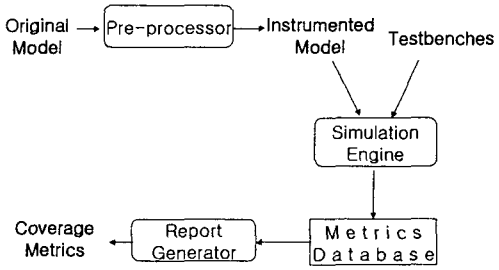


그림 3. 코드검출 과정
<Fig 3> Code coverage process

<Fig. 3> shows how a code coverage tool works[8]. The source code is first instrumented. The instrumentation process simply adds checkpoints at strategic locations of the source code to record whether a particular construct has been exercised. The instrumented code is, then simulated normally using all available testbenches. The cumulative traces from all simulations are collected into a database. From the database, reports can be generated to determine various coverage metrics of the verification suite on the design. While verifying VHDL code using low-level event and metrics, it is very important to determine the stopping point for the current test strategy, the location of stopping point is highly dependent on the statistical model that is chosen to describe the coverage behavior during the verification process[5][7].

The rapid progress of chip fabrication technology has given rise to new hardware solutions consisting of chips that contain whole systems, but the design and the verification of complicated chips has not advanced rapidly, such chips are microprocessor, cache controller in a shared memory, bus controller, and system on chip. To completely verify the chips in a system level, various verification methodologies have been developed[19][20]. But there is no research to use low-level coverage metrics for high-level events. If the low level coverage metrics cab be used to verify high-level events instead of developing new high level coverage metrics, the

cost and time can be reduced. Aim of this paper is to search the possibility to detect high-level events by low-level coverage metrics.

Ⅲ. Definition of high-level events

In this paper, the terminology of high-level events is defined as follows, 1)cooperation between blocks or process statement in HDL, or 2) algorithms or protocols, that is, a sequence of performing a job.

Some examples explain it in details. First, when two components are using a single bus simultaneously, bus conflict occurs. Second example is data hazard in pipelined processors. Data hazard consists in data conflict and branch conflict. Data conflicts can be solved in software or hardware methods, and solutions in software are to insert no-op instruction and to reorder instructions. Solutions in hardware are to stall instruction and to forward data. Branch conflicts are a little complicated because of conditional branch. Branch conflicts of unconditional branch can be solved by inserting no-op, reordering instructions, or inserting stall. Conditional branch conflicts are dependent on conditions. Three methods used in unconditional branched are used for unconditional branches, and additionally annulling and prediction methodscan be used.

Third example is cache coherence in shared memory. Multiprocessors have individual caches for each processor and when two or more caches hold the value of the same memory location simultaneously, one of them may be modified, then two values differ each other, this problem is called cache coherence or cache consistency. Write-through cache can update main memory but not the other cache, so can not solve the problem and extra writes to main memory decrease system performance. Non-cacheable, cache directory and snooping methods can solve the cache coherency. During program compilation in noncacheable, compiler can mark all shared data as non-cacheable andforce all accesses to this data to be from shared memory. It can lower the cache hit ratio and reduce overall system performance. In cache directory, cache controller is integrated with the main memory controller and directory controller updates the cache directory. Each cache monitors memory activity on the system bus in snooping. Popular protocol for marking and manipulating data within multiple caches is snooping and MESI protocol has four states, Modified, Exclusive, Shared, Invalid[21]. Four possible memory access scenarios are read hit, read miss, write hit, and write miss.

Ⅳ. Classification of high-level event

It is necessary to classify the high-level events in order to find high-level coverage metrics. Most high-level events are relevant to protocol or specification-dependent. This paper proposes that high-level events can be classified into two categories, that is, resource conflicts and protocol or specification-dependent conflicts.

Some high-level events are related to resource conflicts as explained in the previous chapter, for example, data conflicts in pipeline, bus conflict in arbiter, resource conflicts in concurrent system, concurrent process statement in VHDL program. The others are related to protocols, algorithm or a sequence of performing some operation, for example, branch conflict, cache coherency, memory access, a sequence of instruction execution.

Ⅴ. Coverage metrics for high-level event

This paper proposes how to verify high-level events by low-level coverage metrics. The events of resource conflicts can be detected by using statement coverage metric if global signal or variable has flags in a testbench program and protocol-dependent conflicts can be checked by data flow or path cover metrics.

Data transfer between blocks or process statements in VHDL program is done through global signals or variables. IF it has been known which signals or variables are used as input or output while VHDL program is being simulated, the conflicts of resource can be recognize.

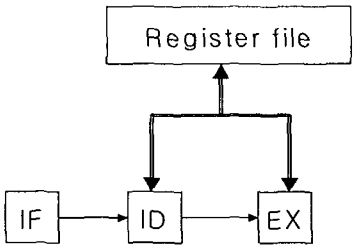
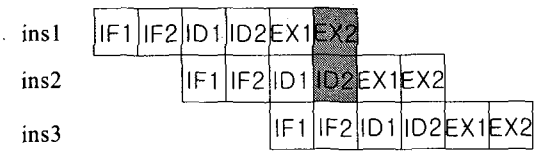


그림 4. 데이터 충돌을 나타내는 3 단계 파이프라인의 예제
<Fig. 4> An example of 3 stage pipeline for data conflicts

<Fig. 4> shows an example of 3 stage-pipeline architecture of microprocessor that is composed of instruction fetch(IF), instruction decode(ID), and execute(EX). Assume Register file used in ID and EX for temporary storage in a microprocessor has a lot of registers. <Fig. 5>(a) shows each stage of pipeline has two clock cycles. Address M is sent to memory for first half cycle of IF stage and the contents of address M is stored to IR for second half cycle of IF. Consider two sequential instructions ins1 and ins2 in <Fig. 5>(b). Ins1 adds two contents of register R2 and R3, and then writes the result to register R1. Ins2 is sameto ins1, but R1 is used before R1 is stored in ins1. If there is not any methods to solve data conflicts, data conflict occurs between EX stage of ins1 and ID stage of ins2 for R1.



(a) EX와 ID 단계 사이의 데이터 충돌
(a) Data conflict between EX stage and ID stage

ins1 : add R1, R2, R3
ins2 : add R4, R1, R5
ins3 : sub R6, R7, F8

(b) 예제 프로그램
(b) Example program

그림 5. 데이터 충돌 예제
<Fig. 5> Example of data conflict

<Fig. 6>(a) is a pseudo HDL code. R1 is simultaneously used in id and ex blocks together, so signal R1 has to be declared as global signal or variable because R1 is a path to send or take data. Assume r and w flags are inserted in R1 signal in a testbench program. r flag is set when operand R1 is fetched for ID or Ex stage , and w is set when EX writes a value to R1. <Fig. 6>(b) is a testbench program for <Fig. 6>(a). In this case, the condition of if statement of testebnch becomes to 1, so it has been recognized that data conflict occurs in R1 at that time cycle.

```
global signal R1;
regfile : process();
if : process();
id : process();
ex : process()
```

(a) HDL 의사코드
(a) HDL pseudo code

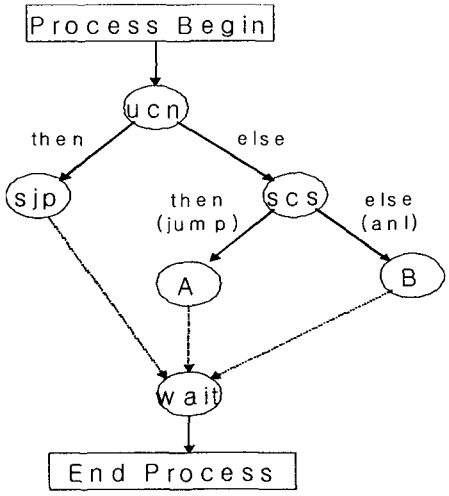
```
R1 with r and two w flags;
if((r=1 and (w1=1 or w2=1)) or
 (w1=1 and w2=1)
        conflict;
```

(b) 테스트벤치 프로그램
(b) Testbench program

그림 6. HDL 의사코드 및 테스트벤치 프로그램
<Fig. 6> HDL pseudo code and Testbench program

It is not easy to check protocol or algorithm-dependent conflicts, but data flow or path coverage metric can check the sequence of execution of algorithm. Branch prediction or snoopy protocol can be expressed as state diagrams. Each state can be translated to IF or CASE statements. So protocol-dependent event can be checked by data flow metric or path metric.

Assume that <Fig. 7>(a) is a stage diagram that execute an algorithm and it is similar to execute branch instructions in a microprocessor. ucn, sjp, scs, and anl in the diagram represent unconditional jump, short jump, success, and annulling respectively. This state diagram can be expressed with IF and ELSE statement in VHDL program like <fig. 7>(b).



(a) 알고리듬에 대한 상태도
(a) State diagram for an algorithm

```
process()
begin
  if unconditional then
    if short jump then
          ...
    else
          ...
    end if;
  else
    if branch success then
      if A then
          ...
      else
          ...
      end if;
    else
      if B then
          ...
      else
          ...
      end if;
    edn if;
  end if;
End process;
```

(b) VHDL 코드
(b) VHDL code for an algorithm

그림 7. 알고리즘 충돌 예제
<Fig. 7> An example of Algorithm conflicts

If the data flow or path has been checked in VHDL code, the algorithm-dependent conflicts can be verified. The solution of cache coherency can be expressed in such a state diagram, so it can be verified by data flow or path metric

VI. Conclusion

This paper defined the terminology of high-level events and classified it into two classes, resource conflict and algorithm conflict. They can be verified by low-level metrics. The methods can reduce time and cost of IC design when they are used before specification simulation. The high-level event that has protocol fault itself can't be detected by proposed methods. In the future, searching space and searching time for detection of high-level event will be conducted.

References

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

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

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

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

[5] Kangchul Kim, "Efficient methods for reducing clock cycles in VHDL model verificatoin," Journal of Electronics Engineers of Korea, V.40-SD, pp39-45, Dec. 2003.

[6] Michael Keating and Pierre Bricaud, Reuse Methodology Manual, Kluwer Academic Publishers, 1998

[7] Amjad Hajjar and Tom Chen, "An Accurate Forecasting Model for Behavioral Model Verification," Proceedings of the First IEEE International Workshop on Electronic Design, Test and Applications, 2002.

[8] Janick Bergeron, Writing Testbenches : Functional Verification of HDL models, 2nd edition, Kluwer

Academic Publishers, 2003

[9] Qiushang Zhang and Ian G. Harris, "A Data Flow Fault Coverage Metric For Validation of Behavioral HDL Descriptions", ICCAD, pp. 369-372, 2000.

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

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

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

[13] Kevin Skahill, "A Designer's guide to VHDL design and verification", Electronic design, pp. 149-152, Feb. 19, 1996.

[14] W. Howden, "Confidence-based reliability and statistical coverage estimation", ISSRE'97, pp 283-291, Nov, 1997.

[15] B. Dickinson, S. Shaw, "Software techniques applied to VHDL design", New Electronics, N9, pp 63-65, May 1995.

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

[17] Martin Abraham, et al, "Optimize ASIC testsuite using code coverage analysis", EDN, pp. 149-152, Mat 21, 1998.

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

[19] Daniel Geist, et. al, "A Methodology for the verification of on a System on chip", DAC, pp 574-579, 1999.

[20] Gilly Nativ, et. al, "Cost evaluation of coverage directed test generation for the IBM Mainframe", ITC, pp 793-801, 2001.

[21] John D. Carpinelli, Computer Systems Organization and Architecture, Addison Wesley, 2000.

저자약력

김 강 철

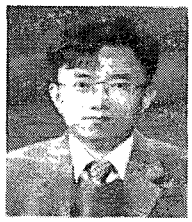
경상대학교 전자공학과, 공학박사
현재 전남대학교 공학대학 컴퓨터공학과 부교수

※관심분야 : VLSI 및 임베디드시스템 설계

임 창 균

Wayne State University, 컴퓨터공학과 졸업, 박사
현재 전남대학교 공학대학 컴퓨터공학과, 부교수

※관심분야 : 인공지능, 임베디드 소프트웨어, 유비쿼터스 응용



류 재 흥

Wayne State University, 전산학과, 박사
현재 전남대학교 공학대학 컴퓨터공학과, 부교수

※관심분야 : 컴퓨터그래픽스, 인공신경망, 패턴인식

한 석 붕

한양대학교 전자공학과 공학박사
현재 경상대학교 전자공학과 교수

※관심분야 : RFIC, SoC 설계 및 테스팅