# IP 설계 환경을 위한 VHDL Code Coverage Checker

김 영 수, 류 광 기, 배 영 환, 조한진
한국전자통신연구원
전화 : 042-860-1645 / 핸드폰 : 011-9701-8960

# VHDL Code Coverage Checker for IP Design and Verification

Youngsoo Kim, Kwangki Ryoo, Younghwan Bae and Hanjin Cho
Electronics and Telecommunications Research Institute
E-mail : youngsoo@etri.re.kr

## Abstract

This paper describes a VHDL code coverage checker for IP design and verification. Applying the verification coverage to IP design is a methodology rapidly gaining popularity. This enables the designers to improve the IP design quality and reduces the time-to-market by providing the quantitative measure of simulation completeness and test benches. To support this methodology, a VHDL code coverage model was defined and the measurement tool was developed.

## I. 서론

SOC(System-On-a-Chip) 설계에서 이미 설계된 IP의 재사용은 증가하는 설계 복잡도의 문제를 해결하기 위한 방법으로 대두되고 있으며 이에 IP 설계 및 재사용에 적합한 CAD 툴의 개발이 중요한 문제가 되고 있다. IP 설계 과정은 개념설계->설계->검증->구현 과정을 통하여 이루어지며 이중에서 설계와 검증에 많은 시간이 소모되며 특히 IP의 검증에 40%의 시간이 소모되고 있음이 문헌에 의해 보고되고 있다.

IP 설계 환경에서 요구되는 기능 검증의 문제를 해결하기 위한 방법에는 기존의 시뮬레이션에 의존한 방법과 formal verification등의 formal method가 적용되고 있다. 시뮬레이션에 의존한 방법은 test bench에 의하여 설계를 검증하는 방법으로 test bench의 질에 따라 의존하는 단점 및 어느 정도까지 검증하여야 하는지에 대한 정량적인 수치화가 힘들며 설계의 시험되지 못한 부분에 대한 정보를 직접 얻을 수 없는 단점을 가진다. 또한 후자의 경우는 이상적인 기능 검증 방법이나 실제 설계에 적용하기 곤란한 문제점을 가진다.

집적도가 큰 IP 설계의 검증은 전통적인 시뮬레이션 방법과 설계방법론을 적용할 경우 제한점을 가지고 있다. 검증초기에는 RTL 코드 중에서 검증된 부분의 양이 급격히 증가하나 검증시간이 진행됨에 따라 그 비율이 감소하며 모든 RTL 코드가 검증되었는지 정량적으로 사용된 test bench의 질을 측정하기 매우 힘들다. 검증이 진행됨에 따라 나타나는 설계 오류에 대하여 regression test를 위하여 매회 시뮬레이션으로 검증하여야 하므로 설계시간에 bottleneck으로 작용하고 있다.

이러한 문제점들을 해결하기 위하여 code coverage checker를 이용하면 시뮬레이션 사이클 수행 이후에 code coverage checker에서 발견된 untested RTL 코드에 대해서 test bench를 작성하도록 피드백을 받을 수 있으므로 보다 빠른 시간 안에 100% 시험 가능하도록 test bench를 작성할 수 있다.

code coverage checker는 이러한 문제점을 해결하기 위하여 RTL 코드의 기능 검증 단계에서 test bench의 질을 정량적으로 측정하고 RTL 코드의 질을 향상시키기 위한 방법으로 필요하게 되었다.

본 논문에서는 VHDL 코드를 대상으로 하여 하드웨어 coverage 모델을 정의하고 각각의 coverage 항목을 확인할 수 있는 도구를 개발하였다.

## II. Code coverage checker를 이용한 IP 검증 설계 흐름 및 관련 도구

기존의 전통적인 시뮬레이션 기반의 검증 설계 흐름

에 본 논문에서 개발한 code coverage checker를 적용한 IP 설계와 검증을 위한 설계 흐름을 그림 1에 표시하였다.

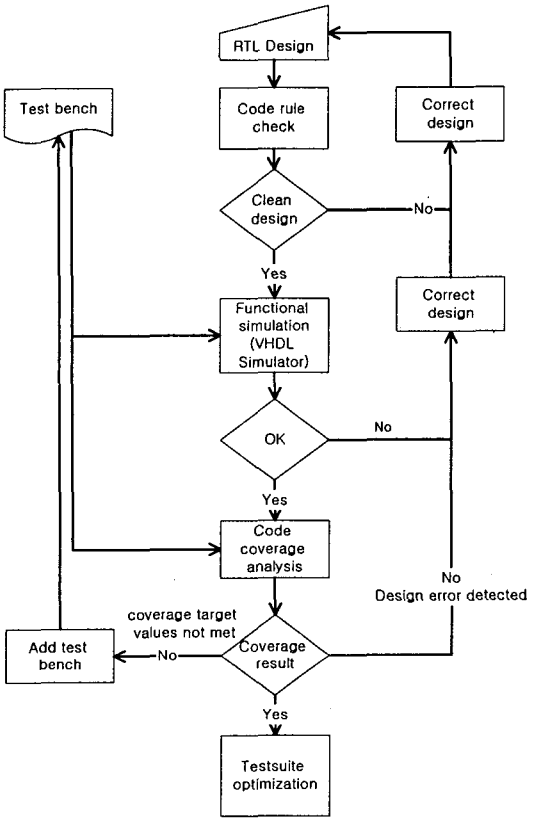


그림 1. code coverage checker를 이용한 설계 흐름

RTL 설계가 끝난 후에 rule checker를 이용하여 시뮬레이션과 회로합성단계에서 문제가 될 수 있는 코드를 확인한다. 이후 설계자가 작성한 test bench를 이용하여 기능 시뮬레이션을 수행하게 된다. 설계자가 작성한 specification과 비교하여 기능 검증을 수행하게 되며 검증결과는 설계자의 test bench에 종속되므로 기능 시뮬레이션 후에 code coverage checker를 이용하여 미리 정한 coverage metric을 만족할 때까지 test bench를 보강하여 시험되지 않은 RTL 코드 부분이 없도록 이 과정을 반복하게 된다. 이러한 과정을 통하여 test bench의 완전성을 확신할 수 있기 때문이다.

위의 설계 흐름에서 적용 가능한 상용 도구는 표 1과 같다.

| | 제작사 | 지원 HDL | 기능 |
|---|---|---|---|
| CoverMeter | Synopsys | Verilog | -statement<br>-condition<br>-user expression<br>-toggle |
| VN-Cover | TransEDA | Verilog,VHDL | -statement<br>-branch<br>-condition<br>-path<br>-toggle |
| Coverscan | Cadence | Verilog | -statement<br>-expression<br>-decision<br>-state machine |
| HDLScore | Innoveda | Verilog | -block<br>-path<br>-expression<br>-toggle |
| VeriCover | Veritools | Verilog | -branch<br>-multiconditional |
| SureCov | Verisity | Verilog | -block<br>-arc<br>-toggle |

표 1. 상용 code coverage checker의 기능 비교

각 상용 도구의 기능과 지원되는 HDL를 위주로 비교하였다. 위의 결과를 보면 HDL이 대부분 Verilog 위주임을 알 수 있다. 개발이 용이한 이유 이외에 IP 검증 환경에 초점이 맞추어져 있다기보다는 기존의 논리 설계에 집중하는 모습을 보이고 있다. 각각의 기능에도 RTL 코드 검증에 필요한 항목들이 누락되어 있는 점들을 볼 때 기존에 이미 설계된 verilog 코드를 이용한 설계환경에 적합한 도구들로 생각된다.

위와 같은 상용 도구들의 비교를 기반으로 하여 본 논문에서는 IP 설계 환경의 기능 검증을 위하여 필수적인 VHDL 언어를 지원하며 IP의 설계 및 기능 검증을 위하여 RTL 코드상의 statement coverage, branch coverage, condition coverage, path coverage등을 모두 지원할 수 있는 code coverage checker를 개발하였다.

## Ⅲ. Code coverage 모델 및 code coverage checker의 구현

### 3.1 전체 구조

본 논문에서 구현한 VHDL code coverage checker의 전체 구조는 그림 2와 같다.

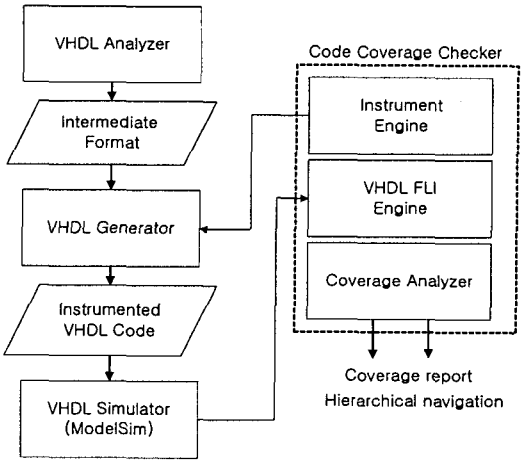


그림 2. VHDL code coverage checker 전체 구조

대상 설계의 VHDL 파일을 분석하여 중간 형식으로 저장한 후 이를 VHDL 코드로 재생성하는 과정에서 coverage 모델의 세부 항목에 해당하는 VHDL 문들의 실행을 확인하기 위하여 부가적인 Tag 문장들이 추가(instrument) 된다. Tag 문장들이 추가된 VHDL 코드는 VHDL simulator(Mentor Graphics사의 ModelSim)를 통하여 시뮬레이션을 수행하게 된다. 시뮬레이션이 수행되는 과정에서 추가된 tag 문장들을 통하여 각 VHDL 문들의 실행 및 실행 횟수 등의 필요한 정보들을 VHDL FLI(Foreign Language Interface) 방식을 통하여 시뮬레이터의 내부에 접근하여 code coverage analyzer가 수집하게 된다.

### 3.2 VHDL 코드 추가(instrument)

VHDL로 작성된 RTL 코드를 중간형식으로 저장한 후 해당하는 coverage 모델에 근거해서 VHDL 코드를 추가한 후 다시 VHDL 코드로 생성한다. 이 과정에서 각각의 coverage 항목에 대해서 수행하는 추가 절차는 다음과 같다.

(1) Statement coverage를 위한 추가

Statement coverage의 계산을 위하여 RTL 코드에 추가되는 절차는 표 2와 같다.

표 2. statement coverage 추가

| statement coverage 항목 | 추가 방법 |
|---|---|
| process begin calls | process 내에 boolean tag expression을 추가 |
| signal assignments | assignment문 이후에 boolean tag expression을 추가 |
| procedure calls in processes | procedure body에 boolean tag expression을 추가 |
| variable assignments | assignment문 이후에 boolean tag expression을 추가 |
| concurrent assignment | 별도의 process를 생성하여 boolean tag expression을 추가 |

위와 같이 변형된 RTL 코드는 원래의 코드를 대체하여 시뮬레이터에서 수행하게 되며 code coverage checker에서 시뮬레이터와 접속하여 변형된 코드의 삽입된 boolean 변수들의 값에 접근하여 실행여부를 판단하게 된다.

(2) Branch coverage를 위한 추가

Statement coverage의 계산을 위한 추가 절차와 마찬가지로 Branch coverage의 계산을 위해서는 표 3과 같은 RTL 코드의 추가절차를 거치게 된다.

표 3. branch coverage 추가

| branch coverage 항목 | 추가 방법 |
|---|---|
| if .. elsif .. else ... end if | 각 branch문에 integer tag expression을 추가 |
| case | 각 branch문에 integer tag expression을 추가 |
| next, exit | next, exit문의 조건에 해당하는 if문을 생성하여 삽입 |
| conditional and selected signal assignment | condition에 해당하는 if문을 생성하여 삽입 |

(3) Condition coverage를 위한 추가

Condition coverage의 계산을 위해서는 논리 연산자로 연결된 각각의 조건을 확인하는 if 문을 추가적으로 삽입한다.

(4) Path coverage를 위한 추가 절차

동일한 process, procedure 혹은 function안에 있는 브랜치 간에 형성된 논리 경로를 검출하기 위해서 그림 3과 같은 방법을 이용한다.

```
FUNCTION INSTRUMENT_FOR_PATH_COVERAGE
    Find blocks(branch) in the sample process, procedure
    and function

    WHILE not all the blocks which constitutes the branch
            have been processed
    BEGIN
        IF Current branch corresponds the first block
           in the process, procedure and function
            instrument the mark code by a unique
            prime number
        ELSE
            instrument the code that multiply the next
            unique prime number
        ENDIF
    END
ENDFUNCTION
```

그림 3. path coverage를 위한 추가 절차

위와 같은 절차에 의해서 추가된 코드는 시뮬레이션 시 test bench에 의해서 해당하는 경로가 수행되었는지 추가된 정수형 변수의 값을 factoring함으로써 알 수 있다.

### 3.3 FLI(Foreign Language Interface)

추가된 VHDL 코드는 상용 시뮬레이터를 사용하여 시뮬레이션을 수행하게 되며 이때 code coverage analyzer에서 FLI를 이용하여 coverage 값들을 얻어낼 수 있다. 본 논문에서는 멘토사의 ModelSim 시뮬레이터를 대상으로 하여 구현하였다. 추가된 VHDL 코드가 분석되고 elaborate된 이후에 미리 FLI를 통해 등록한 콜백 함수들이 호출되며 FLI 엔진에서 필요한 변수들에 접근하여 code coverage 계산에 필요한 정보를 얻을 수 있다.

### 3.4 Code coverage analyzer

Code coverage의 수행결과를 설계자에게 보다 효과적으로 전달하기 위해 Java 언어를 이용하여 사용자 인터페이스를 개발하였다. Java 언어를 사용하여 개발되었기 때문에 개발 플랫폼인 Sun의 Solaris 운영체제 뿐만 아니라 Windows 환경 및 다른 환경으로의 이식이 용이하다. 그림 4에는 구현된 code coverage checker의 결과를 보여주는 code coverage analyzer의 사용자 인터페이스가 표시되어 있다.

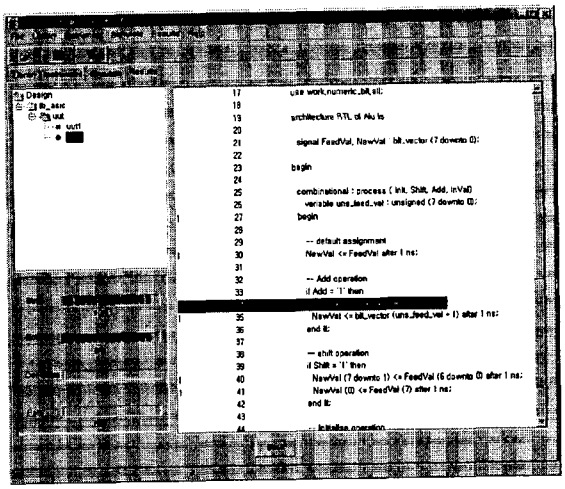

그림 4. code coverage analyzer 사용자 인터페이스

## Ⅳ. 결론

본 논문에서는 IP 설계 및 검증을 위한 VHDL code coverage 모델을 정의하고 coverage 모델에 근거한 code coverage checker를 개발하였다. 이는 IP 검증 도구로 사용되어 IP 설계 환경 구축에 기여할 것으로 생각한다. 앞으로의 연구과제는 toggle coverage, signal tracing 기능을 추가하고 상용 Tool과의 Benchmark를 통하여 개발된 code coverage checker의 우수성을 입증하는 것이다.

## 참고문헌

[1] Farzan Fallah, Coverage-Directed Validation of Hardware Models, Ph.D Thesis, Massachusetts Institute of Technology, June 1999.

[2] R. Grinwald, E. Harel, M Orgad, S. Ur and A. Ziv, User Defined Coverage - A Tool Supported Methodology for Design Verification, In Proceedings of the 35th design Automation Conference, June 1998.

[3] Boris Beizer, Software Testing Techniques, International Thomson Computer Press, 1990.

[4] B. Marick, The Craft of Software Testing, Subsystem testing Including Object-based and Object Oriented Testing. Prentice-Hall, 1985.