계층화된 테스트벤치를 이용한 검증 환경 구현
Implementation of a Verification Environment using Layered Testbench

오영진*, 송기용*
Young-Jin Oh*, Gi-Yong Song*

요 약

최근 시스템의 규모가 커지고 복잡해지면서, 시스템 수준에서의 기능 검증방법론이 중요해지고 있다. 기능블록의 검증을 위해서는 주로 BFM(bus functional model)이 사용되며, 기능 검증에 대한 부담이 증가할수록 올바른 검증환경 구성의 중요성은 더욱 증가한다. SystemVerilog는 Verilog HDL의 확장으로 하드웨어 설계언어의 특징과 검증언어의 특징을 동시에 갖는다. 동일한 언어로 설계기술, 기능 시뮬레이션 그리고 검증을 진행할 수 있다는 것은 시스템개발에서 큰 이점을 갖는다. 본 논문에서는 SystemVerilog를 이용하여 AMBA 버스와 기능블록으로 구성된 DUT를 설계하고, 계층적 테스트벤치를 이용한 검증환경에서 DUT의 기능을 검증한다. 기능 블록은 Adaptive FIR 필터와 Booth's 곱셈기를 사용한다. 이를 통하여 검증환경이 DUT와 연결되는 인터페이스의 부분적인 변경을 통하여 다른 하드웨어의 기능을 검증하는데 재사용되는 이점을 가지고 있음을 확인한다.

Abstract

Recently, as the design of a system gets larger and more complex, functional verification method based on system-level becomes more important. The verification of a functional block mainly uses BFM(bus functional model). The larger the burden on functional verification is, the more the importance of configuring a proper verification environment increases rapidly. SystemVerilog unifies hardware design languages and verification languages in the form of extensions to the Verilog HDL. The processing of design description, function simulation and verification using same language has many advantages in system development. In this paper, we design DUT that is composed of AMBA bus and function blocks using SystemVerilog and verify the function of DUT in verification environment using layered testbench. Adaptive FIR filter and Booth's multiplier are chosen as function blocks. We confirm that verification environment can be reused through a minor adaptation of interface to verify functions of other DUT.

Keywords : SystemVerilog, verification environment, layered testbench, interface

I. 서 론

반도체 공학의 기술 발전으로 IC칩의 집적도가 높아짐에 따라 하나의 기능블록에 포함되는 하드웨어의 기능과 구조가 점점 복잡해지고 있다. 최근 시스템의 설계가 대형화되고, 복잡해지면서 효율적인 설계를 위하여 이미 설계되어진 기능블록의 재사용은 반드시 적용될 수밖에 없는 설계 방법이 되었다. 기능블록의 재사용을 위해서는 시스템을 구성하고 있는 기능블록들의 기능검증이 무엇보다 중요하다[1]. 다양한 기능을 가지고 있는 기능블록의 동작을 시스템 수준에서 검증하기 위해서는 복잡한 시험 각본이 필요하고, 올바른 검증환경의 구성과 검증의 복잡도를 다루기 위한 설계 및 검증언어의 선택이 중요하다[2].

현재 기능적 검증방법론에서는 설계한 기능블록의 검증을 위해 실제 프로세싱 코어를 대신하는 BFM(bus functional model)이 주로 사용되고 있으나, 최근 단일언어로 시스템의 설계 및 검증이 가능한 SystemVerilog를 이용한 방법론이 대두되고 있다.

기존의 HDL은 RT 수준의 하드웨어 구성요소 설계에는 매우 적합하지만, 객체지향프로그래밍 기법의 특징인 상속, 참조, 스래드, 그리고 형변환과 같은 개념을 제공하지 못하기 때문에 상위수준 추상화 개념을 이용한 프로그래밍 기법을 처리하기에는 부적합하다[3]. 그러나 상위수준추상화를 이용할 수 있는 SystemVerilog는 단일언어로 설계기술과 기능 시뮬레이션, 그리고 검증을 진행할 수 있다[3][4].

본 논문에서는 설계와 검증을 단일언어로 수행할 수 있는 SystemVerilog를 이용하여 능동적인 검증환경을 구현하고, Adaptive FIR 필터와 Booth's 곱셈기가 AMBA 버스 내의 기능

* 충북대학교
투고 일자 : 2009. 12. 18 수정완료일자 : 2010. 8. 11
게재확정일자 : 2011. 4. 30
* 이 논문은 2009년도 충북대학교 학술연구지원사업의 연구비 지원에 의하여 연구되었음(This work was supported by the research grant of the Chungbuk National University in 2009).
* 이 논문은 12권1호(2011.1.30)에 게재 예정이었으나 학회 편집업무 착오로12권2호(2011.4.30)에 게재되었음.

블록으로 구성된 DUT(design under test)의 기능검증을 통해 구현된 검증환경이 재사용 가능함을 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 SystemVerilog와 설계한 DUT 모듈에 대해 간략히 기술하고 3장에서는 계층화된 테스트벤치 기반의 기능 검증환경의 구현에 대하여 기술한다. 4장에서는 설계한 검증환경과 DUT를 결합하여 기능 검증을 수행한 결과를 보이고, 5장에서 결론을 맺는다.

II. Preliminary

2.1 SystemVerilog

시스템을 설계·구현하는 과정과 설계된 시스템을 검증하는 과정은 서로 다른 관점에서 중요 작업들이 수행된다. 이러한 관점의 차이는 각 작업에 최적화된 언어의 개발이 이루어지면서 설계툴(design tool)과 검증툴(verfication tool) 사이에서 구문과 의미의 해석 차이를 발생시켜 많은 어려움을 야기한다[1].

최근 발표된 SystemVerilog는 하드웨어 설계언어인 Verilog HDL을 기반으로 상위수준 언어인 C/C++의 randomization, OOP, 연산자 overloading 그리고 인터페이스 등의 여러가지 개념을 추가하여 단일언어로 시스템의 설계 및 검증 기능을 수행할 수 있도록 하였다[1][2].

SystemVerilog는 기능블록의 설계를 위해 기존의 HDL의 설계 특징을 이용할뿐 아니라, 상위수준 설계를 지원하기 위한 C와 유사한 데이터 타입, typedef라는 키워드를 이용한 사용자 정의 데이터 타입, 데이터 타입의 변경이 가능한 structures, unions, type casting 등을 사용한다. 또한 설계블록간의 protocol checking과 캡슐화, 통신을 위한 인터페이스 등을 제공한다. Verilog HDL에서는 프로시져블록의 수행을 위해 always 구문 하나만을 제공하였지만, SystemVerilog는 이 구문을 세분화하여 설계자의 의도가 분명하게 드러날 수 있도록 한다[3][4].

SystemVerilog는 설계언어로서 추가된 여러 형태의 데이터 타입 외에 검증언어로서 유용하게 사용되는 동적 배열, 큐, 클래스 등을 추가하였다[11]. 검증을 위해 구현된 테스트벤치는 설계블록과는 별도로 모델링된다. Verilog HDL의 경우 설계블록과 테스트벤치 사이에 race condition이 발생할수 있는바, 이를 방지하기 위하여 SystemVerilog는 테스트벤치 작성시 program이라는 블록을 추가하였다. program 블록은 하나의 initial 블록만을 가지며, 정확한 시뮬레이션을 위해 설계블록과 테스트벤치블록이 정해진 순서에 따라 순차적으로 활성화된다[4][11]. 이외에도 SystemVerilog는 OOP기법을 이용하기 위해 클래스를 추가하였고, 상속의 개념을 통해 기존 클래스로부터 새로운 클래스를 유도한다. 또한 Verilog HDL의 fork ... join 구문의 기능을 확장하여 동적 스레드를 생성할 수 있는 fork... join_any 블록과 fork ... join_none 블록을 추가하였고, 스레드간의 상호 통신을 위해 세마포어와 메일박스 등을 이용한다[3]. 이러한 추가 요소들은 복잡한 설계를 쉽게 하고, 지나친 상세화를 피할 수 있어 코드 작성시 오류 발생 위험을 줄인다.

2.2 DUT 모듈의 설계

디지털 신호처리의 주요한 구성요소인 디지털 필터는 신호에 대한 잡음특성에 따라 신호 성분의 제거 및 보상 등에 대한 성능과 시스템 구성의 용이성 등의 장점으로 인해 널리 쓰이고 있으며, 크게 FIR(finite impulse response) 필터, IIR(infinite impulse response) 필터 등이 사용된다.

IIR 필터는 필터의 특성은 우수하나, 회로의 복잡도가 크고 궤환 구조로 인한 불안정 요인이 있다. 반면 FIR 필터는 선형위상 특성 및 안정성이 좋고, 다양한 필터의 특성에 대하여 설계의 용이성을 지니고 있어 널리 사용된다[9]. FIR 필터는 순차적으로 입력되는 신호와 계수들의 곱셈연산을 통해 중간결과를 저장 후 이 값들을 연속적으로 더함으로써 최종 결과값을 생성한다. 본 논문에서는 데이터 처리량의 극대화를 위해 병렬 처리 방식을 고려한 유연한 하드웨어 구조를 가지며, 고속의 덧셈기를 필요로 하는 직접형 구조의 FIR 필터를 구현하였다. 또 다른 설계블록으로 Booth's 곱셈기를 구현하였다. Booth's 곱셈기는 양수와 음수의 승수를 동일하게 취급하여 부호가 있는 2진 보수 곱셈을 효율적으로 수행하는데 적합하다. 구현된 곱셈기의 대략적인 구조는 그림 1과 같다.

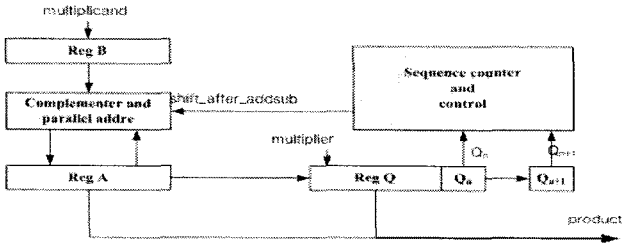

그림 1. Booth's 곱셈기의 구조
Fig. 1. Structure of the Booth's multiplier

검증 대상인 DUT모듈은 ARM 프로세서 기반에서 각종 입출력장치간 신호교환을 위한 표준시스템 버스설계인 AMBA 버스에 Adaptive FIR 필터 또는 Booth's 곱셈기 IP(intellectual property)를 연결하여 구성한다. AMBA 버스를 이용한 설계는 IP의 재사용이 용이하여 시스템 설계시간을 단축하고 오류를 줄일 수 있어서 현재 대부분의 임베디드 시스템 또는 SoC에 채용되고 있다. 그림 2는 AMBA 버스와 ahb_bfm, ahb_decoder, 설계 IP로 구성된 DUT의 내부 구조이다.

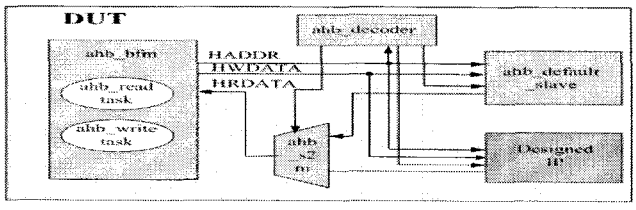

그림 2. DUT 모듈의 내부 구조
Fig. 2. Internal structure of DUT module

ahb_bfm은 읽기와 쓰기 태스크를 이용하는 포트를 가지고 있다. 본 논문에서는 SystemVerilog의 인터페이스 생성자를 이용하여 ahb_bfm 모듈이 ahb_if 포트를 갖도록 구현하였다. ahb_if 는 읽기 및 쓰기 프로토콜에 맞는 신호를 생성하는 태스크를 정의한다. 그림 3은 ahb_if 인터페이스 코드의 일부를 보인다.

```
interface ahb_if (input bit HCLK, input bit HRESETn);
  bit [31:0] HADDR;
  bit [1:0] HTRANS;
  // ...
  task ahb_read;
  input [31:0] address;
  input [2:0] size;
  ref [31:0] data;
  begin
    // ...
  endtask
  task ahb_write;
  input [31:0] address;
  input [2:0] size;
  input [31:0] data;
  begin
    // ...
  endtask
endinterface
```

그림 3. ahb_if 인터페이스 코드의 부분

Fig. 3. Partial code of a ahb_if interface code

III. 계층화된 테스트벤치를 이용한 검증

3.1 능동적인 검증 환경

하나의 시스템을 구축할 때, 구현할 기능들이 복잡해짐에 따라서 전체 시스템을 새롭게 구현하지 않고 이미 기능검증이 끝난 IP들의 결합을 통해 원하는 기능을 수행하도록 전체 시스템을 개발하는 것이 일반적인 방법이다. 설계한 시스템을 검증하는 가장 단순한 방법은 설계블록의 입출력 신호와 내부 신호의 동작 파형을 분석하는 것이다. 그러나 입출력 신호가 많거나 특정 신호에 많은 의미가 포함되어 있고, 복잡한 기능을 포함한 여러 설계블록들이 상호 연결되어 있다면 파형분석만으로는 완벽한 기능검증을 할 수 없다[1]. 시스템의 복잡한 기능들을 모두 검증하기 위해서는 다양한 시험각본이 필요하고 검증과정에서 발생하는 설계 오류를 수정할 경우, 시험각본을 처음부터 다시 진행시켜야 한다. 이는 많은 시간과 노력을 필요로하여, 설계 생산성 측면에서 매우 불리하다. 따라서 입력에 대한 출력결과의 자동적인 분석을 통한 올바른 동작 검증이 가능한 능동적이고 지능적인 검증환경이 필요하다[1][4-5].

본 논문에서는 그림 4에서 DUT를 검증함에 있어 입력을 생성하는 블록(stimulus generator), 출력을 분석하는 블록(postprocessor)으로 구성된 검증환경을 구성한다.

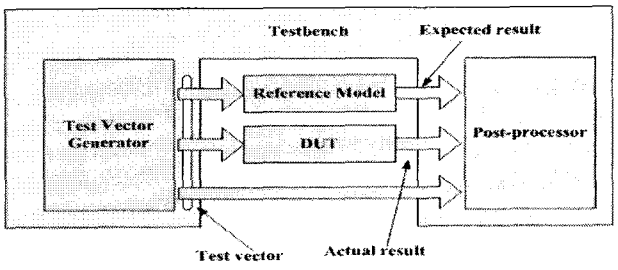

그림 4. 검증환경의 구성

Fig. 4. Configuration of a verification environment

검증하고자 하는 DUT의 기능이 복잡하다면 모든 기능을 검증할 수 있는 test vector의 생성이 어려울 수 있으므로, DUT의 기능을 충실하게 소프트웨어적으로 모델링하는 참고모델(reference model)을 동시에 사용한다[1]. 이를 통해 실제 DUT의 출력값과 알고리즘 수준에서의 기대 출력값을 비교함으로써, DUT의 올바른 동작 여부를 확인하여 검증의 정확성을 높인다.

3.2 계층적 검증환경의 동작 수행

본 절에서는 SystemVerilog의 OOP 기법을 이용하여 앞 절에서 구성한 검증 환경을 구체화하여 그림 5에 보인바와 같은 계층화된 테스트벤치 구조를 구현한다.[3-5].

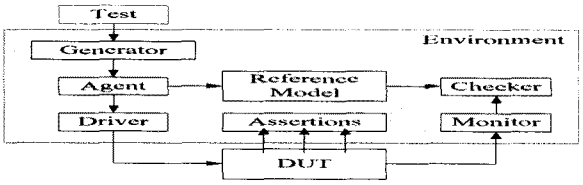

그림 5. 계층화된 테스트벤치 구조

Fig. 5. Structure of a layered testbench

DUT의 동작 검증환경은 임의의 테스트벡터를 생성하는 부분(Generator), 생성한 테스트벡터를 참고모델과 다른 계층에 전달하는 부분(Agent), 검증하고자 하는 DUT 모듈에 테스트벡터를 적용하는 부분(Driver), DUT의 수행결과를 읽는 부분(Monitor), 그리고 DUT의 올바른 동작을 참고모델의 결과와 비교하여 검사하는 부분(Checker)으로 구성된다. Test는 Environment 객체를 생성한 후 시뮬레이션을 시작하는 program 블록이다. Environment 클래스의 구성요소인 Driver와 Monitor는 가상인터페이스를 통하여 DUT와 통신하며, 나머지 구성요소간의 통신은 IPC를 통하여 이루어진다. SystemVerilog의 인터페이스는 모듈 상호간의 연결에 필요한 데이터뿐만 아니라, 통신 프로토콜까지 정의할 수 있어 interface객체에 대한 포인터로 하드웨어 모듈과 검증환경간의 데이터를 전송하는 메커니즘을 제공한다.

Environment 클래스와 Test program 블록의 코드 일부를 그림 5에 간략히 보인다.

```
class Environment;
  Generator gen;
  Agent agt;
  Driver drv;
  Monitor mon;
  Checker chk;
  ReferenceModel rm ;
  Config cfg;
  mailbox gen2agt, agt2drv, mon2chk;
  virtual MULT_BOOTH_if MULT_BOOTH_if_;
// virtual FIR_if FIR_if_;
  extern function new ( );
  extern function void gen_cfg ( );
  extern function void build ( );
  extern task run ( );
  extern task wrap_up ( );
endclass

program automatic test;
  Environment env;
  initial begin
    env = new ( );
    env.gen_cfg ( );
    env.build ( );
    env.run ( );
    env.wrap_up ( );
  end
endprogram
```

그림 5. Environment 클래스와 Test 코드의 일부

Fig. 5. Partial code of Environment class and Test

계층화된 테트스벤치는 정확한 동작을 위해 크게 3단계로 나누어 수행된다[4][5][11].

- Build 단계
  테스트벤치의 구성요소와 DUT를 연결하여 하나의 검증환경을 생성하고, DUT의 동작을 위해 초기상태로 변경한다.
- Run 단계
  검증환경의 동작이 시작되고, Test 코드를 시작시켜 DUT의 동작을 시뮬레이션한다.
- Wrap-up 단계
  DUT의 모든 동작이 종료될때까지 기다린 후 최종 수행결과를 확인한다.

Ⅳ. 기능 검증 결과

본 절에서는 설계한 검증환경에 DUT내의 IP를 Adaptive FIR 필터와 Booth's 곱셈기로 교체하면서 기능검증을 수행한 결과를 그림 6과 그림 7에서 보인다.

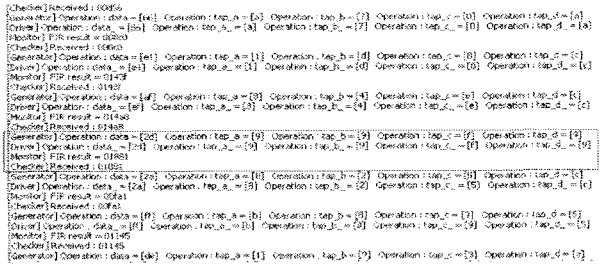

그림 6. 기능 검증 결과 (Adaptive FIR 필터)
Fig. 6. Results of a functional verification (Adaptive FIR Filter)

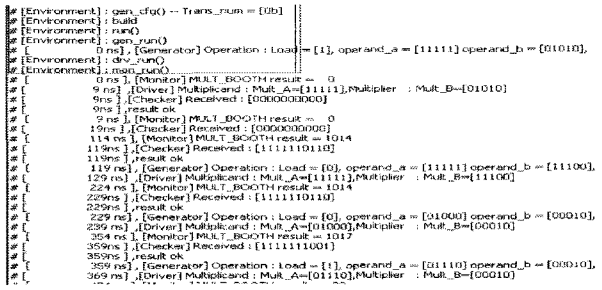

그림 7. 기능 검증 결과 (Booth's 곱셈기)
Fig. 7. Results of a functional verification (Booth's multiplier)

그림 6과 그림 7에서 검증환경이 Test program 블록의 초기 실행 순서에 따라 3단계 동작을 수행하고 DUT가 각각의 기능 블록에 따라 정상적인 동작을 수행함을 확인할 수 있다.

구현된 계층적 구조의 검증환경은 입출력 인터페이스 프로토콜의 변경으로 여러 다른 DUT의 기능검증에 재사용된다.

Ⅴ. 결 론

SystemVerilog는 Verilog HDL을 기반으로 하드웨어 설계언어의 특징뿐 아니라 상위수준의 OOP 기법, randomization, 스레드, IPC 등을 이용한 하드웨어의 검증언어로서의 특징을 갖는다. SystemVerilog는 OOP 기법을 이용하여 기본 클래스의 property와 method를 공유하는 유도 클래스의 생성으로 코드의 재사용성을 높일 수 있다.

본 논문에서는 시스템수준에서 하드웨어의 설계와 검증을 단일언어로 수행할 수 있는 SystemVerliog를 이용하여 간단한 Adaptive FIR 필터와 Booth's 곱셈기를 구현한 후 검증을 위한 계층화된 테스트벤치 기반의 검증환경을 구성하였다. 검증환경과 DUT 사이의 부분적인 인터페이스 변경을 통해 구현된 검증환경이 각 IP에 대하여 재사용 가능함을 확인하였다. 이는 상위 수준에서 검증환경을 구축하고 부분적인 변경을 통하여 다른 하드웨어 구성요소를 손쉽게 검증할 수 있음을 보여주고 있다. 구현된 계층적 구조의 검증환경은 테스트벤치의 단계적인 세분화와 부분적인 수정을 통해 시스템 검증의 신뢰성을 높이고 개발 소요 시간을 줄일 수 있다.

참고 문헌

[1] Ando Ki, *SoC Design and Verification: Methodologies and Environments*, pp.2-8, Hongreung Science, 2008.

[2] Chris S.Pear, *SystemVerilog for verification.* Springer, pp15 -24, 2008.

[3] Stuart Sutherland, Simon Davidmann, Peter Flake, *SystemVerilog for Design*, pp 2-6, Springer 2006.

[4] 유명근, 송기용, "SystemVerilog와 SystemC 기반의 통합검증환경 설계 및 구현",신호처리 · 시스템 학회 논문지, 제10권, 4호, pp.274-279, 2009.

[5] 유명근, 오영진, 송기용, "시스템수준의 하드웨어 기능검증 시스템",신호처리 · 시스템 학회 논문지, 제11권, 2호, pp.177-182, 2010.

[6] SystemVerilog 3.1a Language Reference Manual: Accellera's Extensions to Verilog, Accellera, Napa, California, 2004

[7] Jason R. Andrews, *Co-Verification of Hardware and Software for ARM SoC Design*, pp.119 -129. Elsevier Inc., 2005.

[8] Stuart Sutherland, *The Verilog PLI Handbook : A Tutorial and Reference Manual on the Verilog Programming Language Interface*, pp.27-54, Kluwer Academic Publishers, 2002.

[9] Mike Mintz, Robert Ekendahl, *Hardware Verification with C++:A Practitioner's Handbook,* pp.67-88, Springer, 2006

[10] 백제인, 김진업 "표본화 속도 변환기용 2단 직렬형 다상 FIR 필터의 설계", 한국통신학회논문지, vol 31, No 8C, pp806-815, 2006

[11] Douglas J Smith *HDL Chip Design.* Doone Publications(MadisonAL,1996),pp286-287,297-301.

[12] IDEC, *IDEC newsletter -SystemVerilog를 이용한 설계 및 검증(2) : 검증*, pp14 ~17, 2009.

오 영진(Young-Jin Oh)

2005년 충북대 컴퓨터공학과 학사
2007년 충북대 대학원 컴퓨터공학과 석사
2008년~현재 충북대 대학원 반도체공학과 박사과정

※주관심분야 : SoC 설계 · 검증, 상위수준 설계 · 검증

송 기용(Gi-Yong Song)

1978년 서울대 공교과 학사
1980년 서울대 대학원 전자과 석사
1995년 미 루이지애나대 박사
1983년~현재 충북대 전자정보대학 교수

※주관심분야 : SoC 설계 · 검증, 상위수준설계, C++ 기반 하드웨어 검증