SystemC 구성요소를 이용한 SystemVerilog 기반 검증환경

SystemVerilog-based Verification Environment using SystemC Constructs

오 영 진*, 송 기 용**
Young-Jin Oh*, Gi-Yong Song**

요 약

시스템의 복잡도가 증가함에 따라 상위수준 추상화에 기반한 시스템수준 설계 및 하드웨어의 기능적 검증을 위한 방법론의 중요성이 부각되고 있으며, Verilog HDL(Hardware Description Language)에 하드웨어 검증기능이 추가된 SystemVerilog를 이용하는 시스템수준의 기능적 검증방법이 각광받고 있다. SystemVerilog는 Verilog HDL의 확장된 형태로 하드웨어 설계언어와 검증언어의 특징을 모두 포함하나, 다중상속을 허용하지 않는다. 본 논문에서는 SystemVerilog 기반의 검증환경과 다중상속을 허용하는 SystemC의 구성요소를 SystemVerilog DPI(Direct Programming Interface) 및 ModelSim macro를 이용해 결합한 다중상속이 가능한 검증환경을 구성한다. 다중상속이 허용된 검증환경 시스템은 특정부분을 수정 후 재실행으로 DUT(Design Under Test)의 기능 검증을 쉽게 수행할 수 있으며, OOP(Object Oriented Programming) 기법을 이용한 코드의 재사용성이 높아 또 다른 DUT의 동작 검증에 재사용할 수 있다.

Abstract

As a system becomes more complex, a design relies more heavily on a methodology based on high-level abstraction and functional verification. SystemVerilog includes characteristics of hardware design language and verification language in the form of extensions to the Verilog HDL. However, the OOP of SystemVerilog does not allow multiple inheritance. In this paper, we propose adoption of SystemC to introduce multiple inheritance. After being created, a SystemC unit is combined with a SystemVerilog-based verification environment using SystemVerilog DPI and ModelSim macro. Employing multiple inheritance of SystemC makes a design of a verification environment simple and easy through source code reuse. Moreover, a verification environment including SysemC unit has a benefit of reconfigurability due to OOP.

Keywords : SystemVerilog, SystemC, verification environment, layered testbench, OOP, multiple inheritance

I. 서 론

최근 시스템의 규모가 커지고 IC 칩의 집적도가 높아짐에 따라, 설계의 복잡도와 생산성의 증가, 점점 짧아지는 시장공급시간 등에 의해 상위수준 추상화에 기반을 둔 시스템수준 설계 및 기능검증 방법론이 SoC(System-on-a-Chip)의 설계생산성 측면에서 중요해지고 있다. 또한 구현하고자하는 시스템의 기능과 구성이 복잡해질수록 기능검증에 대한 부담이 증가하고 이를 다루기 위한 언어의 선택이 중요해지면서 HDL(Hardware Description Language)을 기반으로 상위수준의 검증기능을 추가한 SystemVerilog를 이용한 시스템수준 검증방법이 새롭게 주목받고 있다.

단위기능을 수행하는 IP(Intellectual Property)들이 복잡하게 연결되어 다기능을 수행하는 현대의 시스템들은 IP 사이의 상호 영향을 평가, 정확한 동작을 수행하고, 시스템을 사용하는 사용자의 활용 환경까지를 고려한 시스템수준의 검증은 필수적인 조건이 되고 있다[1]. 일반적으로 검증하고자하는 DUT(Design Under Test)의 기능을 시스템수준에서 검증하기 위해서 구성된 계층적 검증환경은, DUT에 맞도록 객체들을 모두 새롭게 구현해야하고, 이는 검증에 소비되는 시간과 노력의 증가를 수반한다. 그러나 상위수준 언어의 다중상속 기능을 이용하여 검증환경을 구현한다면 각 객체의 구성요소 코드를 재사용하게 되어, 소요되는 노력과 시간을 줄일 수 있다. 또한 설계된 하드웨어와 소프트웨어 사이의 상호동작은 통합검증에 의해 확인 가능하다[1]-[3]. 통합검증[2]-[5]에 관한 기존연구의 대부분은 HDL 모듈과 C 코드 사이의 통신을 위해 소켓(socket), 파이프(pipe),

* 충북대학교 ** 충북대학교(교신저자)
투고 일자 : 2011. 5. 23 수정완료일자 : 2011. 10. 25
게재확정일자 : 2011. 11. 1
* 이 논문은 2010년도 충북대학교 학술연구지원사업의 연구비 지원에 의하여 연구되었음.

세마포어(semaphore) 같은 IPC(Interprocess Communication)를 사용하는데, 이 경우에는 사용자 정의 시스템 함수들이 Verilog PLI (Programming Language Interface)를 통해 HDL 시뮬레이터에 추가되고, 디바이스 드라이버의 시스템 함수 또는 커널에 추가된 시스템 함수의 내부에서 호출되어 사용된다.

SystemVerilog는 다중상속[7][9]를 허용하지 않기 때문에 본 논문에서는 다중상속을 적용할 수 있는 SystemC 설계유닛[12]-[14]를 제안하고, 이를 SystemVerilog 기반의 검증환경과 결합한다. 이를 위해서 SystemVerilog DPI(Direct Programming Interface)[7][9]-[11]과 ModelSim Macro[15]를 이용하여 검증환경의 구성요소들이 공유 라이브러리로 등록되어 컴파일 되도록 해야 한다.

SystemVerilog DPI는 다른 언어와의 인터페이스 방법을 제공하며, DPI를 이용한 공유 라이브러리 등록을 통해 해당 함수와 Task를 호출한다. 최근의 ModelSim은 SystemC 설계유닛을 위한 SystemC 시뮬레이션 컴파일러를 포함하고 있으나, ModelSim을 이용하여 시뮬레이션을 하기 위해서는 SystemC 설계유닛이 ModelSim이 제공하는 Macro를 이용할 수 있도록 수정한다.

본 논문은 2장에서 SystemVerilog 기반의 검증환경에 대해 간략하게 기술하고, 3장에서는 다중상속을 통해 구현된 SystemC의 구성요소를 적용한 시스템수준 검증환경을 보인다. 4장에서는 SystemVerilog의 구성요소들만 이용한 검증과 SystemC의 다중상속이 적용된 검증을 수행하여 그 결과를 보인다. 마지막으로 5장에서 결론을 맺는다.

II. SystemVerilog 기반의 검증환경

대규모 디지털 시스템의 효율적인 검증과 상위수준 모델링을 허용하는 SystemVerilog[6]-[11]은 Verilog HDL의 확장된 형태로 OOP(Object Oriented Programming), randomization, Thread, IPC 등과 같은 하드웨어 기능검증 구성요소의 추가로 시스템의 설계 및 검증을 단일 언어로 수행 가능하다. 그러나 SystemVerilog는 다른 상위수준 언어들이 제공하는 다중상속을 허용하지 않고 오직 단일상속만을 허용한다.

현대 검증방법론의 핵심개념은 테스트벤치를 설계할 때 발생할 수 있는 문제들을 관리하고 설계 자체의 복잡도를 조정할 수 있는 계층적 테스트벤치이다. 계층적 테스트벤치의 대표적인 2개의 구조는 참고문헌 [7],[8]에서 소개하고 있다.

본 논문에서 채택한 참고문헌 [7]의 계층적 테스트벤치 구조는 그림 1에서 보인다. 채택된 검증환경은 generator, agent, driver, scoreboard, monitor, 그리고 checker로 구성된다.

generator는 DUT의 검증을 위한 임의의 테스트벡터를 생성하고, 생성된 테스트벡터를 다른 계층 또는 다른 객체에 전달하기 위해서 agent가 사용된다. driver는 전달받은 테스트벡터를 DUT에 실제 적용하고 monitor에서는 DUT의 수행결과를 표시한다. checker는 DUT의 수행결과와 참고모델에서의 예측결과를 비교하여 DUT를 검증한다. test는 Environment 클래스의 객체들을 생성한 후 순차적으로 시뮬레이션을 시작하는 메소드를 호출할 수 있는 프로그램(program) 블록이다.

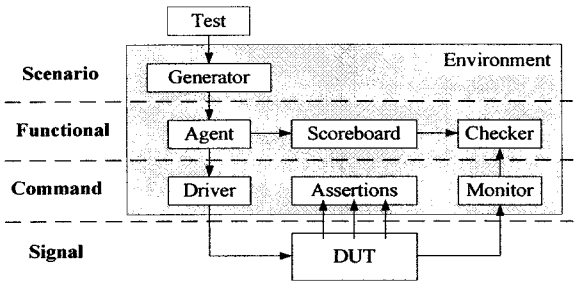

그림 1.계층적 테스트벤치의 구조

Fig. 1. Structure of a layered testbench

Command 계층의 구성객체인 driver와 monitor 그리고 DUT 사이의 통신은 가상인터페이스를 이용하고, 다른 계층의 구성요소 사이의 통신은 IPC를 이용한다. SystemVerilog 인터페이스는 모듈 사이의 통신프로토콜을 정의하여 검증환경내의 데이터 전송을 위한 메커니즘을 제공한다. SystemVerilog에서 IPC는 세마포어와 메일박스(mailbox)를 이용하여 구현된다.

III. SystemC 구성요소를 적용한 검증환경

SystemC는 다양한 추상화 수준의 시스템수준 설계를 위한 언어이다. 표준 C/C++ 언어를 기반한 SystemC는 시간(time), 하드웨어 데이터 타입, 동시성(concurrency), 계층화(hierarchy)와 같은 개념을 제공하여, 다양한 추상화 수준에서의 기능을 서술할 수 있게 하므로 다중상속을 적용하는 검증환경의 구현을 위해 SystemC를 채택하였다. SystemC의 다중상속을 이용한 예로는 참고문헌 [3]에 검증환경의 구성요소간 데이터 전송을 위한 사용자정의 채널의 구성을 보여주고 있다. 참고문헌 [3]은 데이터 전송을 위한 IPC에 집중해 채널을 새롭게 구성하였으나, 본 논문에서는 데이터 전송을 위한 IPC가 아니 검증하고자하는 DUT에 적합한 테스트벡터 생성을 위한 객체의 구성을 다중 상속을 통해 구현하였다. 구현된 객체는 Verilog HDL을 이용한 시스템 설계에서와 같이 하나의 모듈로서 검증환경을 구성가능하고, 구현된 객체는 하나의 모듈로서 검증환경 내에서 모듈의 변형 또는 재사용이 용이함을 확인할 수 있다.

현대의 칩들은 대개 많은 기능을 가지고 있기 때문에, 실제 칩 환경을 모방하는 시스템수준 검증은 디바이스 사이의 상호동작을 검증하기 위해 필요하다. 이를 위한 시스템수준 검증환경의 구조를 그림 2에 보인다. 그림 2의 Environment 클래스 각 객체는 각 DUT에 공통적으로 적용되는 함수를 공유하고, 기본적인 동작을 구현하고 있는 기본 클래스로부터 파생되어 다양한 동작을 구현하고 있는 파생 클래스들은 Environment 클래스를 통해 코드의 재사용성을 증가시키고, 각 DUT에 맞는 검증 환경이 상속을 통해 구현한다 Environment 클래스의 구성요소들은 SystemVerilog를 이용하여 정의되고, 선언된 변수들과 routine을 갖는 단일클래스의 형태로 정의되어질 수 있다. 그러나 만약 클래스가 단순한 1차원 계층구조를 갖는다면, 검증하고자하는 DUT가 변경되는 경우, 기존 클래스 코드의 직접적인 수정 없이 재사용이나 새로운 코드의 추가를 통한 확장이 어렵다. 만약 다중상속을 도입하여

기본적인 동작을 수행하는 기초 클래스에 다양한 메소드를 다중 상속을 통해 상속 받는다면, 클래스 내부 코드의 수정 없이 메소드의 결합만으로 새로운 동작을 수행하는 확장된 형태의 클래스를 쉽게 구현할 수 있다. 이러한 이유로 Environment 클래스의 구성요소를 다중상속을 허용하는 SystemC로 설계하고, SystemVerilog 기반의 검증환경에 연결한다.

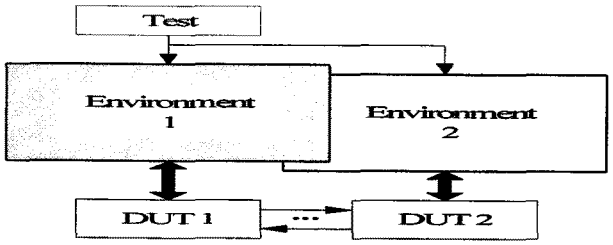

그림 2. 시스템 수준 검증환경의 구조

Fig. 2. Structure of a verification environment in a system-level

각 구성요소의 객체들은 여러 기본 클래스로부터 함수 또는 Task들을 상속받아 다양한 기능을 수행하기 때문에 각 메소드들의 재사용에 따른 이점을 얻기 위해서 다중 상속을 통해 파생 클래스를 유도한다. 객체들의 설계를 위해 적용된 다중상속은 시뮬레이션 성능의 저하 없이 코드의 재사용성을 증가시키기 위해 행위 수준에서 진행되며, SystemC의 다중상속이 적용된 설계는 검증환경의 구축을 쉽게 해준다.

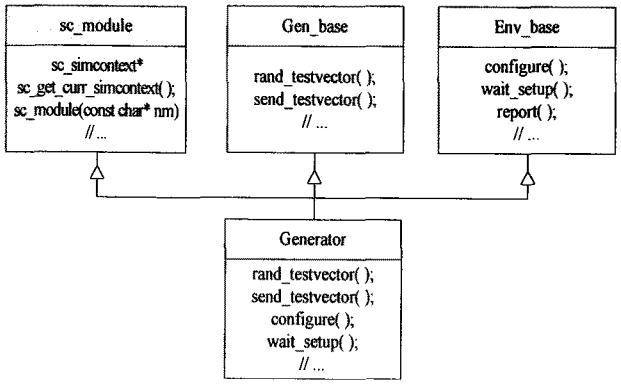

그림 3. Generator의 상속 구조

Fig. 3. Hierarchical structure of a generator

그림 3은 다중상속이 적용된 generator 모듈의 계층적 구조를 보이고 있다. sc_module는 모듈 설계를 위한 기본적인 SystemC 모듈이고, gen_base 클래스와 env_base 클래스는 generator 구성요소들의 기본적인 함수들을 포함하고 있다. 검증환경과 generator의 연결을 위해서는 SystemVerilog DPI와 ModelSim Macro를 이용하기 위한 generator 구성요소의 수정이 필요하다. 그리고 SystemVerilog 기반 검증환경을 위해 공유 라이브러리로 컴파일한다. 그림 4는 generator 구성요소들의 부분코드로 sc_dpiheader.h 파일은 "DPI-SC" import/export와 함수/태스크의 선언으로 구성된 C 함수의 프로토 타입을 포함한다. 이는 Verilog HDL의 소스파일을 컴파일 할 때, 자동적으로 ModelSim의 Verilog 모듈 컴파일러에 의해 만들어진다.

Generator 모듈의 멤버함수들은 SystemVerilog 코드로부터 호출되기 전에 SC_DPI_REGISTER_CPP_MEMBER_FUNCTION Macro를 사용하여 공유 라이브러리에 저장되어야 한다.

```
#include "Gen_base.h"
#include "Env_base.h"
#include "sc_dpiheader.h"
class Generator : public sc_module, public Gen_base, public
Env_base
{
 public :
  void rand_testvector(unsigned int* i);
  void send_testvector(const unsigned int* i);
  void configure( );
  int wait_setup( );
  void report( );
  // ...
 SC_CTOR(Generator)
 {   // ...
   SC_DPI_REGISTER_CPP_MEMBER_FUNCTION
   ("Gen_rand_testvector", &Generator::rand_testvector);
   SC_DPI_REGISTER_CPP_MEMBER_FUNCTION
   ("Gen_send_testvector", &Generator::send_testvector);
   // ...
   SC_DPI_REGISTER_CPP_MEMBER_FUNCTION
   ("Gen_report", & Generator::report);
 }
};
// ...
SC_MODULE_EXPORT(Generator);
```

그림 4. Generator의 부분코드

Fig. 4. Partial code of a generator

Macro의 첫 번째 인자는 실제 함수의 이름과는 별개의 함수 이름이다. 이 이름은 SystemVerilog 코드의 import 선언과 반드시 일치해야한다. 두 번째 인자는 저장된 함수를 위한 함수 포인터이고, SystemC 설계유닛은 반드시 SC_MODULE_EXPORT Macro를 이용하여 검증환경에 연결되어야 한다[15].

gen_base 클래스로부터 상속되는 rand_tset_vector()와 send_test_vector() 함수는 무작위로 테스트벡터를 만들고, 이를 driver에 전달하는 역할을 수행한다. configure(), wait_setup() 그리고 report() 함수는 env_base로부터 상속된다. configure() 함수는 매개변수 등을 초기화하고, wait_setup() 함수는 디바이스들이 안정상태가 될 때까지 기다린다. report() 함수는 내부 상태를 보여주는 역할을 수행한다.

```
module top ;

Generator i_ Gen( );
import "DPI-SC" context function void Gen_rand_testvector output
bit [33:0] i);
import "DPI-SC" context function void Gen_send_testvector (bit
[33:0] i);
import "DPI-SC" context function void Gen_configure( );
import "DPI-SC" context task Gen_wait_setup( );
// ···

bit RESETn;
bit CLK;
DUT_if DUT_IF (CLK, RESETn);
DUT1 DUT1(DUT_IF)
// ···
endmodule;
```

그림 5 Top-level 모듈의 부분코드
Fig. 5. Partial code of a top-level module

그림 5에서 보인 것처럼 최상위 모듈에서는 SystemVerilog 기반의 검증환경에 다중상속이 적용된 SystemC generator를 연결하기 위해 DPI-SC 변경자를 이용한다. SystemVerilog 컴파일러는 SystemC 생성자로 정의된 함수 또는 Task와 같은 구성요소들을 DPI-SC 변경자를 통해 인식하고, 최상위 모듈의 하위 블록들은 SystemVerilog 참조규칙을 통해서 함수 또는 Task들을 참조한다. DPI-SC 변경자를 통해 전달되는 변수들은 SystemVerilog와 SystemC 양 측면에서 모두 일치해야 한다.

Ⅳ. 실험 결과

그림 6은 다중상속으로 구현된 SystemC 설계유닛인 generator가 SystemVerilog기반의 계층적 테스트벤치에 연결된 Environment 클래스의 구조를 보인다. Environment의 구성요소들은 구현방식 따라 SystemC와 SystemVerilog 어느 쪽을 선택해도 무방하다.

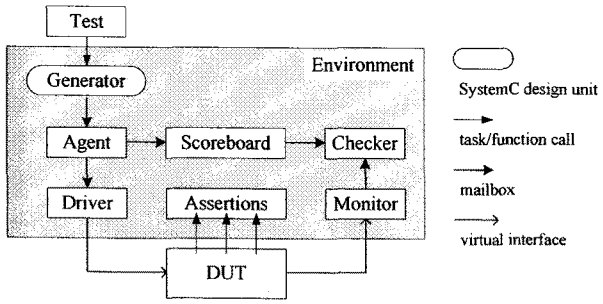

그림 6. Environment 클래스의 구조
Fig. 6. Structure of a Environment class

본 논문에서 구현한 DUT의 구조를 그림 7에서 보인다. DUT1은 ALU 동작을 수행하는 IP를 버스 인터페이스에 연결하여 구현하고, DUT2는 DUT1과 동일한 버스 인터페이스에 연결되어 background traffic을 발생시키는 IP로 구현한다. background traffic은 검증하고자하는 DUT를 제외한 다른 디바이스에서 발생한다.

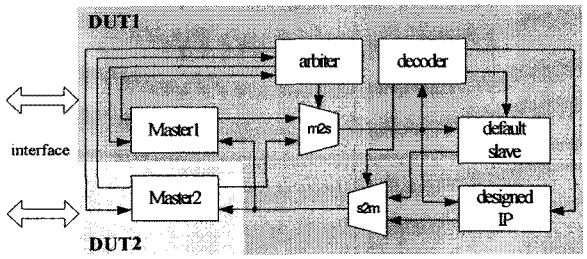

그림 7. DUN의 구조
Fig. 7. Structure of a DUN

그림 8은 그림 7에 적용된 DUT1과 DUT2의 기능적 검증 결과를 보인다.

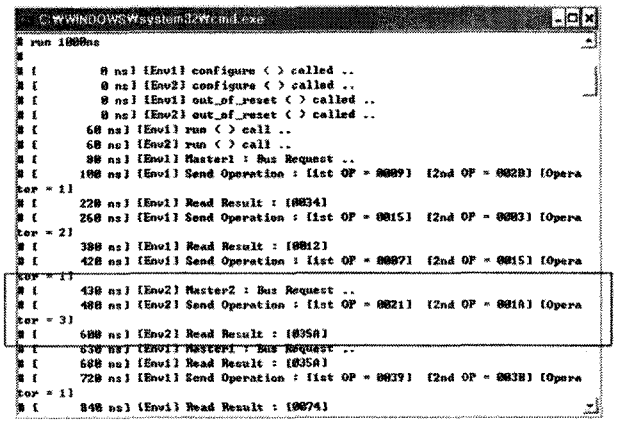

그림 8. 기능 검증의 결과
Fig. 8. Result of a functional verification

DUT2의 master2는 DUT1의 master1이 데이터를 전달하는 동안, bus를 요청하고 사용을 허가받아 설계한 IP에 데이터를 쓰거나 읽기 동작을 수행한다는 것을 보여준다.

Systemic의 다중상속기능을 이용하여 구성한 generator와 SystemVerilog로 구성한 generator의 수행 시간을 비교하여, 크기가 1인 메일박스를 통한 일정수의 데이터 전송시 각 generator의 수행시간을 표 1에 보인다.

표 1. 성능 비교
Table 1. Performance comparison

테스트벡터 수 (개)	SystemVerilog를 이용한 설계	SystemC를 이용한 설계
500	46ms	47ms
5000	301ms	321ms
10000	584ms	635ms

SystemC의 다중상속을 이용하여 설계된 generator와 SystemVerilog로 설계된 generator는 성능이 거의 비슷하지만, 전송되는 데이터의 수가 많을수록 시뮬레이션 수행시간이 증가

한다. 이러한 현상은 HDL 중심의 시뮬레이터인 ModelSim 기반에서 SystemC 코드를 시뮬레이션 함으로써 발행하는 오버헤드로 추정된다. ModelSim이 사용하는 시뮬레이션 커널은 HDL을 시뮬레이션하기 위한 것으로 SystemC의 시뮬레이터 커널과는 다르다. 따라서 ModelSim이 SystemC 코드를 수행하기 위해서는 시뮬레이션 커널에서 HDL 코드 수행을 일시적으로 멈추고 SystemC 코드를 인식하기 위한 스위칭 시간을 필요로 한다. 따라서 수행해야 되는 SystemC 코드가 많을수록 HDL 코드와 SystemC 코드 사이의 스위칭 횟수가 증가하고 이는 필연적으로 전체 시뮬레이션 수행 시간의 증가를 가져온다. 그러나 새롭게 검증시스템을 개발하는 것보다 다중상속을 통해 이미 정확성과 신뢰성을 확보한 검증시스템을 부분적으로 수정하여 재사용하는 것이 시스템의 전체 개발 시간을 단축할 수 있다.

V. 결 론

시스템 복잡도의 증가에 따라 상위수준에서의 설계 및 검증이 부각되고 있으며, Verilog HDL의 확장인 SystemVerilog는 하드웨어 설계와 동시에 검증언어로 각광받고 있다.

본 논문에서는 SystemVerilog 기반의 계층적 테스트벤치에 다중상속을 허용하는 SystemC 설계유닛을 결합한 시스템수준 검증환경을 보인다. 참고문헌 [3]에서는 SystemVerilog로 작성된 모듈에서 생성된 테스트벡터의 전송을 위해 서로 다른 계층의 구성요소 사이의 연결을 위해서 사용자 정의 채널을 다중상속이 허용되는 SystemC를 이용해 구현하였다. 본 논문에서는 테스트벡터 자체의 생성을 위한 generator 객체를 SystemC를 이용하여 구현하였다. 사용자 정의 채널의 구현을 통해 다양한 형태의 데이터 전송이 가능하게 구현 가능함을 보여주었다면, 본 논문에서는 다양한 데이터의 생성이 가능함을 보여주고, 추가적인 객체들의 구현을 통해 전체 시스템을 SystemC의 다중 상속을 이용하는 검증환경 구성 가능함을 확인한다. 이를 통해 검증환경을 구성함에 있어 검증하고자하는 DUT에 맞도록 이미 만들어진 검증환경을 부분적으로 수정함으로써 좀 더 광범위한 재구성이 가능함을 보였다. 본 논문에서는 시스템수준에서 다양한 디바이스 사이의 상호동작을 검증하기 위해서 2개의 Environment 클래스 객체를 사용하였고, Environment 클래스의 generator를 다중상속을 적용하여 SystemC로 설계한 후, SystemVerilog DPI와 ModelSim의 Macro를 이용하여 SystemVerilog 기반의 계층적 테스트벤치에 결합시켰다. SystemC 설계유닛을 갖는 검증환경의 동작은 DUT IP의 시뮬레이션을 통해 확인된다. 다중상속을 적용하여 설계한 객체들은 시뮬레이션 성능 저하 없이 소스코드의 재사용을 통해 행위수준에서 시뮬레이션을 가능하도록 함으로써 검증환경의 구현을 용이하게 한다.

참 고 문 헌

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

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

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

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

[5] T.Jozawa, L.Huang, T.Sakai., S.Takeuchi, M. Kasslin, "Heterogeneous co-Simulation with SDL and SystemC for protocol modeling", RWS, pp. 603-606, 2006,

[6] Stuart Sutherland, Simon Davidmann and Peter Flake, *SystemVerilog for Design(2nd Edition): A Guide to Using SystemVerilog for Hardware Design and Modeling*. pp. 1-6, Springer, 2006.

[7] Chris Spear, *SystmVerilog for Verification(2nd Edition):A Guide to Learning the Testbench Language Features*. pp. 15-24, Springer, 2007.

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

[9] SystemVerilog 3.1a Language Reference Manual: Accellera's Extensions to Verilog, Accellera, pp.7-21, Napa, California, 2004

[10] Stuart Sutherland, "SystemVerilog, ModelSim, and You", pp. 42-54, Mentor User2User, 2004.

[11] Stuart Sutherland, "Intergrating SystemC Models with Verilog and SystemVErilog Models Using the SystemVerilog Direct Programming Interface", SNUG Boston, 2004.

[12] David C. Black, Jack Conovan *SystemC:From the Ground Up*, pp. 8-32, Eklectic Ally, Inc., 2004.

[13] Thorsten Grotker, Stan Liao, Grant Martin, Stuart Swan, *System Design with SystemC*, pp. 123-135, Kluwer Academic Publishers, 2002.

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

[15] *SystemC Language Reference Manual*, http://www.systemc.org.

[16] *ModelSim SE User's Manual*, http://www.mentor.com

오 영 진(Young-Jin Oh)

2005년 충북대 컴퓨터공학과 학사
2007년 충북대 대학원 컴퓨터공학과 석사
2008년~현재 충북대 대학원 반도체공
학과 박사과정
※주관심분야 : SoC 설계·검증, 상위수준 설계·검증

송 기 용(Gi-Yong Song)
正會員

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