SystemVerilog와 SystemC 기반의 통합검증환경 설계 및 구현

Design and Implementation of Co-Verification Environments based-on SystemVerilog & SystemC

유명근*, 송기용**
Myoung-Keun You*, Gi-Yong Song**

요 약

시스템수준 설계방법론에서 널리 사용하고 있는 설계흐름도는 시스템명세, 시스템수준의 HW/SW 분할, HW/SW 통합설계, 가상 또는 물리적 프로토타입을 이용한 통합검증, 시스템통합으로 구성된다. 본 논문에서는 SystemVerilog와 SystemC를 기반으로 하여 신속한 기능검증이 가능한 native-code 통합검증환경과 클럭수준 검증까지 가능한 계층화 통합검증환경을 각각 구현하였다. Native-code 통합검증환경은 시스템수준 설계언어인 SystemC를 이용하여 HW/SW 분할단계를 수행한 후, SoC 설계의 HW부분과 SW부분을 각각 SystemVerilog와 SystemC로 모델링하여 상호작용을 하나의 시뮬레이션 프로세스로 검증한다. 계층화된 SystemVerilog 테스트벤치는 임의의 테스트벡터를 생성하여 DUT의 모서리 시험을 포함하는 검증환경으로 본 논문에서는 SystemC를 도입하여 다중 상속을 가지는 통합검증환경의 구성요소를 먼저 설계한 후, SystemVerilog DPI와 ModelSim 매크로를 이용하여 SystemVerilog 테스트벤치와 결합된 통합검증환경을 설계한다. 다중 상속은 여러 기초클래스를 결합한 새로운 클래스를 정의하여 코드의 재사용성을 높이는 장점을 가지므로, 본 논문의 SystemC를 도입한 통합검증환경 설계는 검증된 기존의 코드를 재사용할 수 있는 이점을 가진다.

Abstract

The flow of a universal system-level design methodology consists of system specification, system-level hardware/software partitioning, co-design, co-verification using virtual or physical prototype, and system integration. In this paper, verification environments based-on SystemVerilog and SystemC, one is native-code co-verification environment which makes prompt functional verification possible and another is SystemVerilog layered testbench which makes clock-level verification possible, are implemented. In native-code co-verification, HW and SW parts of SoC are respectively designed with SystemVerilog and SystemC after HW/SW partitioning using SystemC, then the functional interaction between HW and SW parts is carried out as one simulation process. SystemVerilog layered testbench is a verification environment including corner case test of DUT through the randomly generated test-vector. We adopt SystemC to design a component of verification environment which has multiple inheritance, and we combine SystemC design unit with the SystemVerilog layered testbench using SystemVerilog DPI and ModelSim macro. As multiple inheritance is useful for creating class types that combine the properties of two or more class types, the design of verification environment adopting SystemC in this paper can increase the code reusability.

Keywords : SystemVerilog, SystemC, co-verification environment, native-code, layered-testbench

I. 서 론

현재 SoC(System-on-a-Chip)의 설계생산성을 높이기 위해 상위수준 추상화에 기반한 시스템수준 설계 및 하드웨어의 기능적 검증을 위한 방법론이 대두되고 있다. 시스템수준 설계방법론에서 널리 사용하고 있는 설계흐름도는 시스템명세, 시스템수준의 HW/SW 분할, HW/SW 통합설계, 가상 또는 물리적 프로토타입을 이용한 통합검증, 시스템통합으로 구성된다. 설계시스템의 디자인 범위탐색을 통한 HW/SW 분할은 최종적인 시스템의 구조 및 성능에 영향을 미치는 중요한 단계이며, 분할 후 HW부분과 SW부분의 상호작용을 검증하는 단계는 그 중요성이 더욱 증가하였다 [1]-[4]. SoC를 고려하여 설계되는 대부분의 IP(Intellectual Property)는 버스에 연결되어 동작하며 구현

*충북대학교
**교신저자: 충북대학교 전자정보대학 교수
투고 일자 : 2009. 8. 31 수정완료일자 : 2009. 10. 26
게재확정일자 : 2009. 10. 29

된 기능도 버스를 통해 제어되므로, 프로세싱코어가 없는 상태에서 버스에 연결된 IP의 기능 검증에는 BFM(Bus Functional Model)을 사용한다. 시스템수준 설계언어인 SystemC [5]-[7][14]를 이용한 설계는 점진적인 설계구체화를 통해 다양한 추상화수준에서 설계할 수 있으며, 최근 발표된 SystemVerilog [8]-[12]는 Verilog HDL(Hardware Description Language)의 확장으로 HW 검증기능을 추가하여 상위수준에서 설계 및 검증을 단일 언어로 수행할 수 있다는 특징을 갖는다. 본 논문에서는 SystemVerilog와 SystemC를 기반으로 하여 신속한 기능검증이 가능한 native-code 통합검증환경과 클럭수준 검증까지 가능한 계층화 통합검증환경을 각각 구현하였다. Native-code 통합검증환경에서는 DUT(Device Under Test)의 기능검증을 수행하기 위하여 BFM내에서 버스 프로토콜에 맞는 태스크를 정의한 후 SystemVerilog DPI(Direct Programming Interface) [9]-[12][15]를 이용하여 export하며, SystemC로 기술된 응용프로그램은 export된 태스크를 호출하여 알고리즘 수준에서 기능검증을 수행한다. 계층화된 SystemVerilog 테스트벤치를 이용하는 계층화 통합검증환경은 임의로 생성된 테스트벡터를 DUT에 적용하여 모서리 테스트를 포함한 기능검증을 수행한다. 특히 통합검증환경 설계의 코드의 재사용성을 높이기 위하여 SystemC를 도입하여 다중 상속을 가지는 통합검증환경의 구성요소를 먼저 설계한 후, SystemVerilog DPI와 ModelSim 매크로 [15]를 이용하여 SystemVerilog 테스트벤치와 결합된 계층화 통합검증환경을 설계한다.

II. Native-code 통합검증환경

Native-code 통합검증환경은 명령어집합시뮬레이터를 사용하지 않으므로, 프로세싱코어에서 수행할 프로그램을 크로스컴파일하지 않고 호스트 컴퓨터에서 바로 컴파일하여 로그램을 크따라서 검증구체성이 떨어지는 단점은 있지만 프로그램에 의은 제어되는 하드웨어 자원 참조는 시뮬레이션을 점은 신속하게 기능검증을 수행할 수 있다는 장점이 있다. 본 논문의 SystemVerilog와 SystemC를 을 크따통합검증환경을 도식화하면 그림 1과 같다. Ahb_bfm 모듈은 내부에서 정의하고 ahb_read와 ahb_write 태스크를 스트컴퓨터에서모듈에서 호출할 수 있도록 SystemVerilog DPI를 이용하여 export하며, SystemC 응용프로그램은 export된 ahb_read와 ahb_write 태스크를 호출함있도써 AMBA(Adv와 ce콤어에iteoconteollocoBus Architecture) 신호들을 구동시킨다.

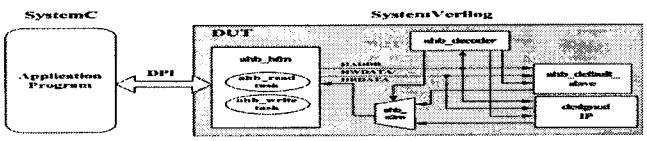

그림 1. Native-code 통합검증환경의 구조

Fig. 1. Structure of a native-code co-verification environment

전형적인 native-code 통합검증환경에서는 C 응용프로그램과 HDL 모듈간의 상호작용을 검증하고자 할 때 소켓 IPC(Interprocess Communication)를 많이 사용한다. HDL 시뮬레이터가 소켓 IPC를 사용하기 위해서는 사용자 정의 시스템함수를 Verilog PLI(Programming Language Interface) [13] 라이브러리로 등록해야하며 또한 등록된 시스템함수 내부에서는 커널의 하위수준 시스템함수를 호출하게 된다. 이러한 과정은 검증의 효율성을 저하시키는데 반해, 본 논문에서 기술하는 바와 같이 SystemVerilog DPI와 SystemC 디자인 모듈까지 시뮬레이션할 수 있는 ModelSim를 이용하면 하나의 시뮬레이션 프로세스로 처리가 되기 때문에 효과적으로 기능검증을 수행할 수 있다.

2.1 HW부분 설계

HW부분은 그림 1의 DUT와 같은 구조이며, DUT 내의 설계된 IP는 AHB(Advanced High Performance Bus) 슬레이브 인터페이스를 통하여 간단한 ALU 연산을 수행한다. SystemVerilog의 최상위 모듈은 내부에서 정의한 ahb_read와 ahb_write 태스크를 SystemC 모듈에서 호출할 수 있도록 SystemVerilog DPI를 이용하여 export한다. SystemVerilog의 최상위 모듈의 코드 일부를 그림 2에 보인다.

```
module top;
  export "DPI-SC" task ahb_read;
  export "DPI-SC" task ahb_write;
  logic clk, reset;
  // ...
  SC_testbench  SC_testbench ( );
  DUT_SV DUT_SV ( // ... );
  task ahb_read;
   input [ 7:0] addr;
   output [ 31:0] data_out;
        // ...
  endtask
  task ahb_write;
   input [ 7:0] addr;
   input [ 31:0] data_in;
        // ...
  endtask
endmodule
```

그림 2. SystemVerilog 최상위 모듈의 코드 일부

Fig. 2. Partial code of a SystemVerilog top-level module

DPI-SC 변경자(modifier)로 export된 태스크들은 ModelSim의 Verilog 모듈 컴파일러를 이용하여 컴파일할 때, 해당 C 함수 프로토타입 구문으로 sc_dpiheader.h 파일에 자동적으로 기록된다.

2.2 SW부분 설계

SystemVerilog에서 export한 태스크를 호출하는 SystemC 모듈을 ModelSim을 이용하여 시뮬레이션하기 위해서 다음과 같은 단계를 수행한다 [15].

- sc_dpiheader.h 파일을 전처리 지시자를 이용하여 포함
- sc_main( )을 SC_MODULE로 선언한 후, 최상위 모듈을 SC_MODULE_EXPORT 매크로를 이용하여 export하도록 소스코드 수정
- sccom 컴파일러와 MinGW gcc 컴파일러를 이용하여 소스코드를 컴파일하여 공유 라이브러리로 생성
- vsim 명령어를 이용하여 시뮬레이션 수행

SystemC 모듈의 코드 일부를 그림 3에 보인다. SC_THREAD 매크로를 이용하여 등록된 프로세스인 proc_SC_testbench( ) 함수는 ALU 연산에 필요한 2개의 피연산자와 연산자를 임의의 값으로 생성하여 ahb_write 태스크를 호출하여 DUT로 전달한다. 또한 ALU의 결과 값을 ahb_read 태스크를 호출하여 읽어들인 후, 레퍼런스 함수인 ref_func 함수의 반환 값과 비교하여 올바른 동작 여부를 판별한다.

```
#include "sc_dpiheader.h"
SC_MODULE (SC_testbench) {
  svLogicVecVal addr;
  svLogicVecVal data_in, data_out;
  // ...
  void proc_SC_testbench ( );
  sc_uint<32> ref_func( // ... );

  SC_CTOR (SC_testbench)
  {   SC_THREAD (proc_SC_testbench); }
  ~SC_testbench ( ) { };
}
void SC_testbench::proc_SC_testbench ( ) {
  svSetScope (svGetScopeFromName ("top"));
  // ...
  while (true) {
  // ...
  rand_value =  (unsigned) rand ( );
  data_in.aval = rand_value;
  data_in.bval = 0;
  ahb_write ( &addr, &data_in );
  // ...
  expected_data_out=ref_func( // ... );
  // ...
  }
}
sc_uint<32> SC_testbench::ref_func( // ... ) {
  // ...
  switch (op) {
    case 0 : expected = op_a + op_b;
    // ...
  }
  return expected;
}
SC_MODULE_EXPORT(SC_testbench);
```

그림 3. SystemC 모듈의 코드 일부

Fig. 3. Partial code of a SystemC module

III. 계층화 통합검증환경

3.1 SystemVerilog 테스트벤치 계층화

최근 시스템검증방법론의 중요한 개념은 테스트벤치 설계에서 발생되는 복잡도를 해결하기 위한 계층적 구조의 테스트벤치 설계이다. [9]에서 소개된 계층화된 테스트벤치의 구조를 그림 4에 보인다. DUT의 동작검증 환경은 그림 4에 보인바와 같이 임의의 테스트벡터를 생성하는 부분(Generator), 생성한 테스트벡터를 전달하는 부분(Agent), DUT에 테스트벡터를 적용하는 부분(Driver), DUT의 정상동작을 예단하는 부분(Scoreboard), DUT로부터 응답을 읽는 부분(Monitor), 그리고 DUT의 올바른 동작여부를 검사하는 부분(Checker)으로 구성된다. Test는 Environment 객체를 생성한 후, 시뮬레이션을 시작하는 program 블록이다.

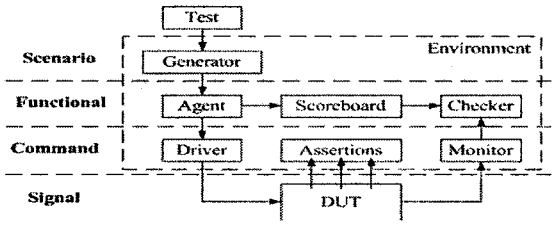

그림 4. 계층화된 테스트벤치 구조

Fig. 4. Structure of the layered testbench

Command 계층의 구성요소인 Driver와 Monitor는 가상 인터페이스(virtual interface)를 통하여 DUT와 통신하며, 나머지 구성요소간의 통신은 IPC를 통하여 이루어진다. SystemVerilog 인터페이스는 모듈연결에 필요한 데이터뿐만 아니라 통신 프로토콜까지 정의할 수 있으며, SystemVerilog IPC 구성요소에는 Verilog의 IPC 구성요소에 세마포(semaphore)와 메일박스(mailbox)가 추가되었다.

3.2 SystemC 모듈-SystemVerilog 테스트벤치 결합형 통합검증환경

시스템수준 설계언어인 SystemC는 시간, 하드웨어 데이터 타입, 병렬성과 계층적 구조의 개념을 제공하기 때문에 다양한 추상화수준에서 시스템의 기능(functionality) 및 통신(communication) 설계가 가능하다.

본 논문에서는 상위수준 통신에서 사용되는 FIFO(First In, First Out) 채널을 사용자정의 계층채널로 설계한다. FIFO 채널은 sc_channel, fifo_if와 payload_if 클래스로부터 다중 상속되는데, sc_channel은 SystemC의 사용자정의 계층채널의 기초클래스이며 fifo_if는 채널의 인터페이스를 정의하는 기초클래스, payload_if는 채널을 통하여 전송되는 데이터에 대하여 동작하는 인터페이스를 정의하는 기초클래스이다.

FIFO 채널은 단일 상속만을 허용하는 SystemVerilog를 이용하여 그림 5와 같이 정의할 수 있다.

```
class fifo_if extends payload_if;
{   // … }
class FIFO extends fifo_if;
{   // … }
```

그림 5. SystemVerilog를 이용한 FIFO 채널선언

Fig. 5. FIFO channel declaration with SystemVerilog

그림 5에서 채널 인터페이스인 fifo_if는 데이터 페이로드 인터페이스인 payload_if를 포함하게 되므로, 채널 인터페이스의 본래 성질이 변질된다. 또한 FIFO 클래스는 payload_if의 protected 제한자로 선언된 멤버 변수들을 접근할 경우 접근함수를 추가하여 사용하여야 한다.

이러한 이유로 본 논문에서는 FIFO 채널을 다중 상속 설계가 가능한 SystemC로 설계하여 계층화된 SystemVerilog 테스트벤치와 결합하여 통합검증환경을 설계한다. 다중 상속은 여러 기초클래스를 결합한 새로운 클래스를 정의하여 코드의 재사용성을 높이는 장점을 가지므로, 본 논문의 SystemC를 도입한 통합검증환경 설계는 검증된 기존의 코드를 재사용할 수 있는 이점이 있다.

SystemVerilog DPI와 ModelSim 매크로를 사용하여 설계한 FIFO 채널은 계층화된 SystemVerilog 테스트벤치가 접근할 수 있도록 공유 라이브러리로 컴파일된다. SystemVerilog 비트벡터 타입의 데이터를 전송하는 FIFO 채널의 부분 코드를 그림 6에 보인다.

```
#include "fifo_if.h"
#include "payload_if.h"
#include "sc_dpiheader.h"
class FIFO : public sc_channel, public fifo_if,
  public payload_if
{ public :
   void FIFO_write(const unsigned int* i);
   void FIFO_read(unsigned int* i);
   void print( );
   unsigned int* _buf;
   int _full, _empty;
   // ...
   SC_HAS_PROCESS(FIFO);
   FIFO(sc_module_name name, int size=16) :
     sc_channel(name), _full(false), // ...
   { // ...
     SC_DPI_REGISTER_CPP_MEMBER_FUNCTION
     ("FIFO_write", &FIFO::FIFO_write);
     SC_DPI_REGISTER_CPP_MEMBER_FUNCTION
     ("FIFO_read", &FIFO::FIFO_read);
     SC_DPI_REGISTER_CPP_MEMBER_FUNCTION
     ("FIFO_print", &FIFO::print);
   }
}
// ...
SC_MODULE_EXPORT(FIFO);
```

그림 6. FIFO 채널의 코드 일부

Fig. 6. Partial code of a FIFO channel

sc_dpiheader.h 파일은 DPI-SC 변경자를 이용하여 import/export한 함수 또는 태스크의 선언과 일치하는 C 함수 원형을 포함한다. FIFO 채널의 멤버 함수들은 SC_DPI_REGISTER_CPP_MEMBER_FUNCTION 매크로를 이용하여 공유 라이브러리로 등록하여야 SystemVerilog 모듈에서 호출할 수 있다. 매크로의 첫 번째 전달인자는 호출시 사용되는 함수의 이름으로 정의한 함수의 이름과 다를 수 있으나 SystemVerilog의 import한 함수의 이름과 동일하여야 한다. 두 번째 전달인자는 공유 라이브러리에 등록된 함수포인터이다. 또한 SystemC 모듈의 객체를 생성하기 위해서는 SC_MODULE_EXPORT 매크로를 이용하여 export하여야 한다 [15].

fifo_if로부터 상속된 FIFO_write( )와 FIFO_read( ) 함수는 내부 배열에 데이터를 저장하거나 배열로부터 데이터를 인출하는 작업을 하며, payload_if로부터 상속된 print( ) 함수는 내부 배열에 저장되어 있는 데이터를 보여주는 동작을 한다. _buf, _full 또는 _empty와 같은 멤버 변수는 FIFO 알고리즘을 모델링하는데 이용된다.

SystemVerilog 모듈은 SystemC로 설계된 FIFO 채널의 멤버 함수를 그림 7과 같이 DPI-SC 변경자를 이용하여 import하여 해당 함수를 호출할 수 있다. DPI-SC 변경자는 SystemVerilog 컴파일러에게 import하는 함수가 SystemC 공유 라이브러리에 정의된 함수임을 알려준다. FIFO_wrtie( )와 FIFO_read( ) 함수는 34-비트 폭의 파라미터를 가지는데 이는 ALU 연산을 위한 2개의 16-비트 피연산자와 2-비트 연산자를 나타낸다.

```
module top ;

FIFO i_FIFO( );
import "DPI-SC" context function
void FIFO_write(bit [33:0] i);
import "DPI-SC" context function
void FIFO_read(output bit [33:0] i);
import "DPI-SC" context function void FIFO_print(
);

  bit HRESETn;
  bit HCLK;
  ahb_if AHB_IF (HCLK, HRESETn);
  ahb_bfm AHB_BFM (AHB_IF);
  // ...
endmodule
```

그림 7. SystemVerilog 최상위 모듈의 코드 일부

Fig. 7. Partial code of a SystemVerilog top-level module

SystemVerilog 테스트벤치와 SystemC로 설계된 FIFO 채널을 결합한 통합검증환경의 구조를 그림 8에 보인다.

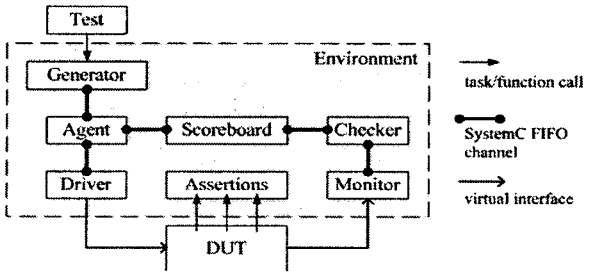

그림 8. 제안된 통합검증환경의 구조

Fig. 8. Structure of the proposed co-verification environment

IV. DUT 기능검증

앞 절에서 기술한 SystemVerilog와 SystemC 기반의 native-code 통합검증환경인 그림 1과 계층화 통합검증환

경인 그림 8을 이용하여 간단한 ALU 연산을 수행하는 DUT의 기능검증 결과를 그림 9와 그림 10에 보인다.

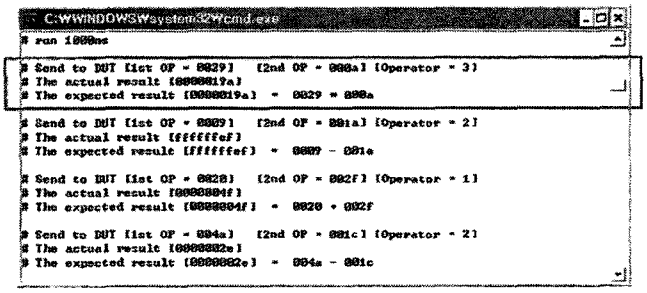

그림 9. Native-code 통합검증 결과

Fig. 9. Result of a native-code co-verification

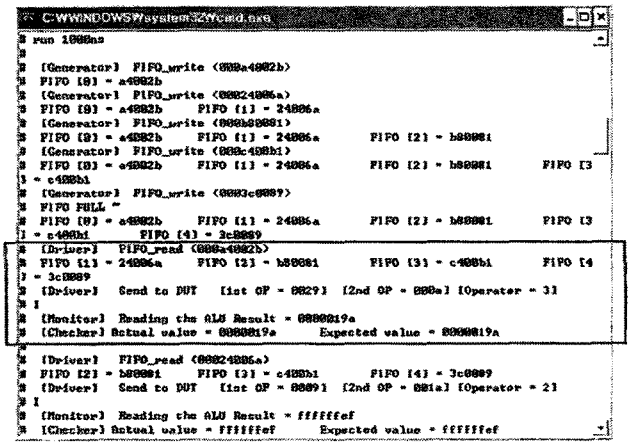

그림 10. 계층화 통합검증 결과

Fig. 10. Result of a layered co-verification

그림 9와 10에 표시된 메시지는 연산자의 값이 3, 즉 곱셈인 경우 DUT의 결과 값과 레퍼런스 함수의 반환 값을 비교하여 출력한 것이다. 그림 10에서는 FIFO 채널의 크기를 5로 설정하여 Generator는 FIFO_write( ) 함수를 호출하여 테스트벡터를 FIFO 채널에 쓴 후에 print( ) 함수를 호출하여 채널에 저장된 데이터를 출력하고, Driver는 FIFO_read( ) 함수를 호출하여 FIFO 채널로부터 테스트벡터를 인출한 후 print( ) 함수를 호출하여 저장된 데이터를 출력하도록 하였다.

V. 결 론

본 논문에서는 시스템수준 설계흐름도에서 HW/SW 분할한 후, SystemVerilog와 SystemC를 이용한 통합설계 및 통합검증에 대하여 기술하였다. 설계된 통합검증환경은 SystemC로 작성된 응용프로그램이 SystemVerilog DPI를 이용하여 SystemVerilog 모듈에서 정의한 태스크를 호출함으로 신호들을 구동시키는 native-code 통합검증환경과, 계층화된 SystemVerilog 테스트벤치에 다중 상속을 가지는 통합검증환경의 구성요소를 SystemC로 설계하여 결합한 계층화 통합검증환경이다.

본 논문의 native-code 통합검증환경은 SystemVerilog 모듈과 SystemC 응용프로그램이 하나의 시뮬레이션 프로세스로 처리가 되기 때문에 신속하게 시뮬레이션 된다는 이점을 가진다. 그리고 계층화 통합검증환경은 모서리 테스트를 포함한 기능검증을 수행하며, 특히 다중 상속을 가지는 통합검증환경의 구성요소를 SystemC로 설계한 후 SystemVerilog 테스트벤치에 결합하면 코드 재사용을 통한 통합검증환경 설계가 용이하다는 이점을 가진다.

본 논문에서 설계된 통합검증환경을 간단한 ALU 연산을 수행하는 DUT에 적용하여 올바른 동작여부를 판별하였다. 본 논문에서 제안된 통합시뮬레이션 환경은 시스템수준 설계방법론에 적용되어 통합검증 단계를 효과적으로 수행할 수 있을 것으로 판단된다.

참고문헌

[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] Yongjoo Kim, Kyuseok Kim, Youngsoo Shin, Taekyoon Ahn, Wonyong Sung, Kiyoung Choi, Soonhoi Ha, "An integrated hardware-software cosimulation environment for heterogeneous systems prototyping," *ASP-DAC*, pp.101-106, 1995.

[4] S. Chikada, S. Honda, H. Tomiyama, H. Takada, "Cosimulation of ITRON-based embedded software with SystemC," *HLDVT* , pp.71-76, 2005.

[5] David C. Black, Jack Donovan, *SystemC:From the Ground Up*, pp.12-23, Eklectic Ally, Inc., 2004.

[6] Thorsten Grotker, Stan Liao, Grant Martin, Stuart Swan, *System Design with SystemC*, pp.51-57, Kluwer Academic Publishers, 2002.

[7] 유명근, 송기용, "C-to-SystemC 합성기의 설계 및 구현," 신호처리·시스템 학회 논문지, 제10권, 2호, pp.141-145, 2009.

[8] Stuart Sutherland, Simon Davidmann and Peter Flake, *SystemVerilog for Design (2nd Edition): A Guide to Using SystemVerilog for Hardware Design and Modeling*, pp.1-7, Springer, 2006.

[9] Chris Spear, *SystemVerilog for Verification (2nd Edition): A Guide to Learning the Testbench Language Features*, pp.1-24, pp.381-419, Springer, 2008.

[10] *SystemVerilog 3.1a Language Reference Manual*: Accellera's Extensions to Verilog, pp.1-3, pp.347-358, Accellera, Napa, California, 2004.

[11] Stuart Sutherland, "SystemVerilog, ModelSim, and You," *Mentor User2User*, pp.1-8, 2004.
[12] Stuart Sutherland, "Integrating SystemC Models with Verilog and SystemVerilog Models Using the SystemVerilog Direct Programming Interface," *SNUG Boston*, pp.1-17, 2004.
[13] Stuart Sutherland, *The Verilog PLI Handbook : A Tutorial and Reference Manual on the Verilog Programming Language Interface*, pp.27-54, Kluwer Academic Publishers, 2002.
[14] *SystemC Language Reference Manual*, http://www.systemc.org
[15] *ModelSim SE User's Manual*, http://www.mentor.com

유 명 근(Myoung-Keun You)

2003년 충북대 컴퓨터공학과 학사
2006년 충북대 대학원 컴퓨터공학과 석사
2006년~현재 충북대 대학원 컴퓨터공학과 박사과정

※주관심분야 : 임베디드시스템 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