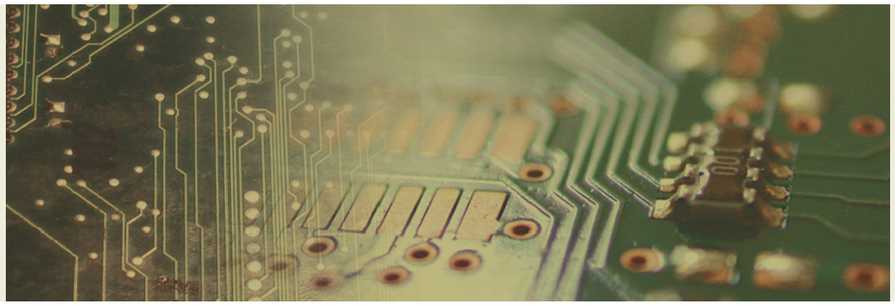
시스템수준 언어: SystemC & SystemVerilog

충북대 전자정보대학 반도체공학과

송기용 교수
연구분야 : SoC 설계 · 검증, SystemC 모델링
E-mail : gysong@cbnu.ac.kr
http://bnadi.cbnu.ac.kr/~gysong

충북대 전자정보대학 반도체공학과

오영진 박사 과정
연구분야 : SystemVerilog기반 하드웨어 검증
플랫폼, SystemC기반 설계
E-mail : goodmen913@cbnu.ac.kr



서론

시스템수준 언어의 필요성

최근 시스템반도체(SoC)가 디지털시스템의 핵심이 되면서 '시스템'이라는 개념을 바꾸고 있다. 고전적인 시스템의 개념은 마이크로프로세서, 메모리 칩, 아날로그 부품 그리고 기타 ASIC과 같은 개별적인 부품들을 모아 하나의 결과를 산출하는 것이었다. 이러한 시스템의 설계는 시스템설계 전문가가 하드웨어와 소프트웨어로 개발 부분을 분할하고, 이에 따른 사양이 결정되면 RTL 수준으로 설계하는 방식으로 진행되었다. 이때 HDL은 매우 적절한 선택이다[1].

그러나 시스템반도체는 다수 프로세서와 DSP, 특정기능의 수행성능 향상을 위한 하드웨어 부분과 주변 제어장치들을 버스를 통해 연결하고, 각 모듈은 자체적으로 작동하며 버스의 프로토콜에 따라 통신을 하게 된다. 이는 필연적으로 하드웨어와 결합할 소프트웨어의 필요성을 증대시키고, 하드웨어를 제어하는 소프트웨어의 복잡성을 증가시킨다.

하드웨어와 소프트웨어, 다양한 기능의 주변장치들 결합과 통신 등 시스템의 복잡한 사양을 작성하고 모델링, 설계 및 검증수행을 RTL수준에서 진행하기에는 그 복잡도를 다루기가 매우 어렵다. 이에 '시스템수준'이라고 하는 RTL보다 좀 더 높은 추상화 수준에 적합한 언어에 의해 각 과정을 진행할 필요가 있다. 본 고에서는 시그널, 이벤트, 인터페이스 등과 객체지향개념을 지원하는 SystemC와 SystemVerilog에 대해 간단히 살펴보고자 한다.

본론

SystemC

SystemC는 상위수준 언어인 C/C++에 하드웨어 설계개념을 도입한 형태로, 개발대상 시스템이나 모듈의 구조탐색, 성능모델링을 위한 TLM을 포함한 여러 추상화 수준의 모델링, 또는 초기 소프트웨어 개발을 위한 가상 프로토타입 작성에 적합한 객체지향 언어이다.

SystemC는 하드웨어 모델링을 위해 모듈 내부에서의 모듈 사례(instantiation)를 통한 계층화, SC_METHOD, SC_THREAD와 SC_CTHREAD를 통한 프로세스 수행의 동시성, sc_int〈〉, sc_logic 등의 하드웨어 데이터타입, sc_time, sc_clock 클래스가 제공하는 시간 모델 등의 개념을 포함하는 C++ 클래스 라이브러리와 사건구동(event-driven) 방식의 시뮬레이션 커널로 구성되며, 하드웨어 모듈, 소프트웨어 코드, 그리고 이 둘이 결합한 시스템을 통합 설계할 수 있는 환경을 제공한다.

SystemC를 사용한 RTL 수준의 설계도 가능하지만 RTL보다 높은 추상화 수준에서 하드웨어, 소프트웨어 또는 이 둘이 결합한 시스템의 모델링에 주로 이용된다. Doulos 사의 조사결과[2]에 의하면 SystemC는 성능모델링(68%), 구조탐색(68%), TLM(56%), 하드웨어-소프트웨어 통합시뮬레이션(56%) 등에 주로 사용되고 있다.

SystemVerilog

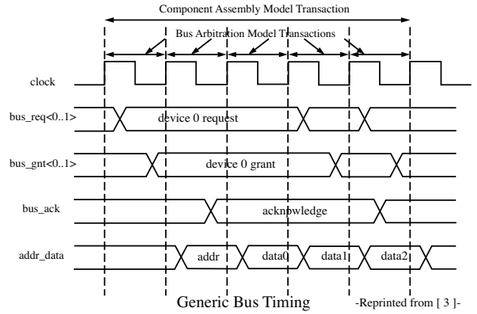
SystemVerilog는 Verilog HDL에 객체지향 개념을 도입한 형태로, 연산자 오버로딩, 인터페이스, IPC(interprocess communication)와 같은 개념이 추가되었으며, constrained-random stimuli generation, functional coverage, assertion 등이 테스트벤치 작성에 이용되도록 제공된다. 이러한 확장을 통해 SystemVerilog는 단일언어로 시스템의 설계 및 검증기능을 수행할 수 있다. 또한 DPI(Direct Programming Interface)를 이용하여 System Verilog 테스트벤치에서 SystemC TLM에 접근할 수 있어 두 블록 간의 동시 시뮬레이션이 가능하다.

TLM(Transaction-Level Modeling)

행위수준 모델링이 기능의 추상화를 끌어올리는 반면, TLM은 제어와 자료흐름 등 정보교환의 하위수준 세부내용을 메소드에 은닉하여 블록 간 통신의 추상화를 끌어올린다. TLM은 모듈의 처리기능과 그들 사이의 통신을 분리하는 것으로 데이터 통신의 실제 구현보다 데이터 전송의 기능성을 강조한 형태이다[1]. 모듈은 채널에 구현된 인터페이스 함수를 호출하여 전송을 요구하게 된다.

이처럼 모듈 간 통신을 함수호출 방식으로 진행하고, 기능의 정확성에 비중을 두어 구조적인 측면에 집중하지 않는다. 예를 들면 버스 설계 시 버스가 지원하는 읽기/쓰기 방식에 대한 여러 가지 전송형태(단일전송, 일괄전송 등)를 모델링하며 버스에 연결되는 실질적인 비트 폭이나 모듈의 핀 등에 대한 구체적인 기술은 하지 않는다.

마이크로프로세서와 메모리, 기타 주변소자가 버스로 연결된 경우의 일괄전송 타이밍 도표를 가지고 추상화 수준, 즉 상세정도를 비교해 보면[3]

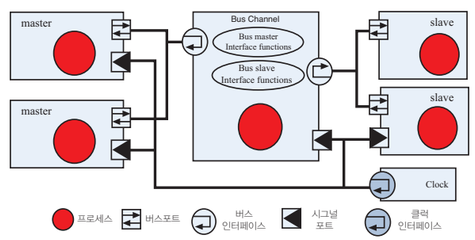


Generic Bus Timing -Reprinted from [3]-

하나의 일괄전송이 버스 사이클 타임의 설정 등으로 상세정도가 사이클 상세수준이 되면, 각 클럭-사이클에서의 전달이 이벤트가 되며, 버스사양이 충분히 구체화하면 각 신호 트랜지션에서 이벤트가 발생한다. 즉 상세 정도가 심화할수록 단위전송이 세분화되는 형태를 보이게 된다.

SystemC TLM

계층채널의 한 경우로 두 개의 마스터와 두 개의 슬레이브가 연결되는 간단한 기능의 버스를 모델링하여 TLM의 예를 보인다[4].



마스터는 버스전송을 개시할 수 있으며, 전송요청이 버스 내부의 큐에 저장되면, 중재기에 의해 사전에 설정된 우선순위에 따라 처리되어 슬레이브에 전달되고, 지정된 슬레이브가 정상적으로 응답하면 해당 전송이 완료된다. 클럭이 연결되지 않은 슬레이브는 단일 사이클 내에 읽기 또는 쓰기 동작 수행이 가능한 '빠른' 메모리 모델의 경우이고, 클럭이 연결된 슬레이브는 읽기 또는 쓰기 동작 수행에 다수의 클럭 사이클을 필요로 하는 '느린' 메모리모델의 경우로 필요한 클럭 수를 설정할 수 있다[1]. 버스는 포트를 통한 인터페이스 함수호출로 접근되고, sc_event&를 통해 포트에 감응(sensitive)하는 모듈의 프로세스를 기동시킨다. 위 그림의 버스는 제어신호, 자료형 및 타이밍 등을 고려한 구체화 과정을 거쳐 SystemC RTL로 구현되며, 이때 어댑터, 컨버터 또는 wrapper의 삽입이 필요하게 된다.

SystemVerilog TLM & BFM

SystemVerilog는 TLM을 동작적 모델링(behavioral modeling)의 부가적인 것으로 정의 한다. SystemVerilog의 TLM은 인터페이스를 통해 데이터 흐름과 제어의 상세내용을 은닉하여 정보교환의 추상화 수준을 높이며, 인터페이스와 태스크 호출 매커니즘은 TLM의 작성을 용이하게 해준다. TLM이 하드웨어 설계로 구체화하려면 통신프로토콜을 구현하고 하드웨어와 신호를 교환하는 BFM이 RTL모듈에 연결되어야 한다. Intel Multibus를 기본으로 하는 인터페이스를 이용하여 RTL로 구현된 메모리모듈과 연결되는 BFM의 개략적 형태를 아래 그림에서 살펴본다[5].

```
module Top;
   Multibus      TLMbus();                          // 각 모듈의 사례화
   Tester        T(TLMbus);
   MultibusMaster MM(TLMbus, ...);
   ... ...
endmodule
interface Multibus;
   parameter int MASTERS = 1;
   tri   [19:0]          ADR;                       // 사용 신호 설정
   ... ...
   extern forkjoin task ReadMem ( ...);
endinterface
module MultibusMaster(...);
   logic     [19:0]     adr  ='z;  assign wires.ADR   = adr;    // 사용 신호 설정 및 할당
   logic                breq  = 1;  assign wires.BREQ[Number]  = breq;
   ... ...
   task Tasks.ReadMem( .....)
        if (.....) GetBus()                         // 버스요청 함수호출
        .....
        fork
          begin
            @(negedge Wries.XACK)                   // 신호 대기
            ... ...
          end
        join
                                                    // 신호 할당, 응답
        FreeBus();                                  // 버스반환 함수호출
endmodule
module Tester  (interface Bus);
   initial  begin
      .....
      Bus.ReadMem(.....);                           // 태스크 호출
      ... ...
   end
endmodule

module Memory(... ...);
   // RTL 수준의 읽기 및 쓰기동작 memory 회로구현
endmodule
```

인터페이스를 선언 및 정의하고 TLM과 PLM(Pin-Level-Model) 버스로 생성한 후 이를 각 모듈에 맵핑시킨다. 이는 System Verilog가 Verilog의 확장형태로 RTL수준의 모듈에 어떠한 변경

이나, 추가적인 코드작성 없이 바로 연결할 수 있기 때문이다.

SystemC 와 SystemVerilog의 연계

SystemVerilog의 DPI기능을 이용하여 SystemC와 SystemVerilog로 구현된 객체들을 연계할 수 있다. 이러한 연계의 핵심은 SystemVerilog의 태스크로부터 직접 SystemC 메소드를 호출하거나 반대로 SystemVerilog 태스크를 SystemC 메소드가 직접 호출하는 것이다. DPI는 설계자가 C/C++/SystemC 기능을 SystemVerilog 모듈에 포함할 수 있도록 하는 인터페이스로 2개의 레이어로 구성되며 기존의 C 코드를 쉽게 재사용할 수 있도록 한다.

C에서 구현된 기능은 SystemVerilog의 import "DPI" 선언을 이용하여 호출될 수 있고, SystemVerilog로 구현된 기능을 C 모듈에서 사용하기 위해 export "DPI" 선언을 통해 호출할 수 있다[6].

SystemVerilog DPI는 다음과 같은 방식으로 SystemC에 접근한다[7].

- SystemC 모델의 입출력 포트를 나타내기 위한 Verilog shell모듈을 생성한다.
- DPI와 SystemC 모델 사이에 값을 전달하는 C 함수를 만든다.
- Verilog shell 모듈에서 SystemC 모델을 제어하는 C 함수를 호출한다.

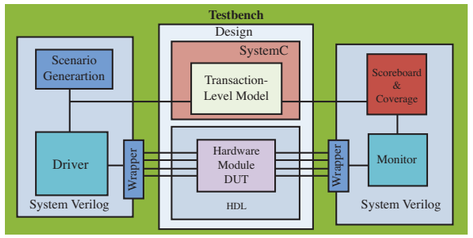
다음은 DPI를 사용하는 예를 간단히 보인 것이다[8].

```
SystemVerilog 부분

module Bus(input In1, output Out1);
   import "DPI" function void slave_write(input int address, input int data);
                                          // 함수가 외부에 정의되어 있음을 선언
   export "DPI" function write;           // 함수를 외부의 C가 호출할 수 있음을 선언

   function void write(int address, int data);
    slave_write(address, data);           // C 코드의 함수를 호출
   endfunction
   ...
   endmodule

-------------------------------------------------------------------

C 부분

#include "svdpi.h"                        // SystemVerilog의 dpi라이브러리를 헤더파일로 사용
   extern void write(int, int);           // SystemVerilog에 정의되어 있음
    void slave_write(const int I1, const int I2)
    {
        buff[I1] = I2;
     ...
     }
```

이 경우 유의할 점은 다음과 같다.

- slave_write C 함수는 SystemVerilog 함수의 내부에서 호출되며, 인자들은 값에 의해 전달되어야 한다.
- import 되는 함수의 인자들은 'const'로 선언되어야 한다. DPI의 import/export 함수는 모듈, 프로그램, 인터페이스 등 SystemVerilog 함수가 선언되는 모든 영역에서 선언될 수 있며 ero simulation time에 수행된다.

이러한 기능을 통해 아래 그림[2]과 같이 SystemC TLM을 SystemVerilog 테스트벤치에 결합하여, 하드웨어의 검증과정에서 SystemC TLM을 참조모델로 사용할 수 있다.



결론

SystemC는 C++에 하드웨어 개념을 도입한 경우이고, SystemVerilog는 Verilog HDL에 OOP 개념을 도입한 경우이다. 상위수준에서의 시스템 모델링을 위하여는 SystemC가 우선 선택될 것이고, 구현된 모델은 설정되어 있지만 복잡한 구체화 과정을 거쳐 SystemC RTL에 이르게 된다.

SystemVerilog의 경우 TLM의 태스크 내부에 함수호출과 신호대기, 할당 등이 동시에 나타나게 되어 구조화 관점에서 다소 혼란스러울 수 있다. SystemC 또는 SystemVerilog를 이용하여 시스템 모델링, 구체화 과정을 통한 RTL구현 및 검증플랫폼 구축이 가능하며 SystemVerilog의 DPI를 이용하여 각 언어에 기반을 둔 모듈을 결합할 수 있다.

참고로, HDL은 델타 타임의 이해에 어려움이 있으며 SystemC 할당은 회로의 wire에 해당하는 채널 sc_signal 에서의 임시저장을 통한 평가와 경신 과정의 이해에 어려움이 따른다. 이는 사건구동 시뮬레이터에 의한 하드웨어 모델링을 위한 전제라 할 수 있다.

Reference

[1] 국일호 역, "SystemC를 이용한 시스템 설계", 에이콘, (2003), Thorsten Grotker, Stan Liao, Grant Martin, Stuart Swan, "System Design with SytemC", Kluwer Academic Publishers

[2] http://www.soccentral.com/results.asp?categoryID=488&EntryID=18241

[3] David C. Black and Jack Donovan, "SystemC: From The Ground Up", Springer, (2004)

[4] 기안도, "SystemC 시스템모델링 언어", IDEC 교재개발 시리 35, 대영사,(2005)

[5] Stuart Sutherland, Simon Davidmann and Peter Flake. "SystemVerilog for Design (2nd Edition): A Guide to Using SystemVerilog for Hardware Design and Modeling", Springer Science+Business Media, LLC(2006)

[6] Myoung-Kenu You, Gi-Yong Song. 'SystemVerilog-ased Verification Environment Employing Multiple Inheritance of SystemC' ,IEICE Trans. Japan(2010),Vol E-3A,No5, pp.989-992

[7] Stuart Sutherland, 'Integrating SystemC Models with erilog and SystemVerilog Models Using the SystemVerilog Direct Programming Interface', SNUG, USA, Boston, (2004).

[8] http://www.asic-world.com/systemverilog/tutorial.html