SystemC를 이용한 SOC 설계 방법

홍 진석(洪 振碩)', 김 주선, 배 점한
삼성전자 중앙연구소 NS lab.
전화 : (0331) 200-4769 / 팩스 : (0331) 200-3122

A SOC Design Methodology using SystemC

JinSeok Hong, JooSeon Kim, JumHan Bae
NS Lab., Central R&D Center, Samsung Electronics Co. Ltd.
E-mail : emailed@samsung.co.kr

Abstract

This paper presents a SOC design methodology using the newly-emerging SystemC. The suggested methodology firstly uses SystemC to define blocks from the previously-developed system level algorithm with internal behavior and interface being separated and validate such a described blocks' functionality when integrated. Next, the partitioning between software and hardware is considered. With software, the interface to hardware is described cycle-accurate and the other internal behavior in conventional ways. With hardware, I/O transactions are refined gradually in several abstraction levels and internal behavior described on a function basis. Once hardware and software have been completed functionally, system performance analysis is performed on the built model with assumed performance factors and influences such decisions regressively as on optimum algorithm selection, partitioning and etc. The analysis then gives constraint information when hardware description undergoes scheduling and fixed-point transformation with the help of automatic translation tools or manually.

The methodology enables C/C++ program developers and VHDL/Verilog users to migrate quickly to a co-design & co-verification environment and is suitable for SoC development at a low cost.

요 약

본 논문은 SystemC의 특징과 어떻게 SOC 설계 방법에 응용될 수 있는지 고려한다. 먼저, 기존 개발된 시스템 알고리듬을 기초로 하여 SystemC로 기능 블럭과 인터페이스를 분리하여 정의한다. 이렇게 정의된 기능 블럭과 인터페이스를 모듈화하고 묶어서 실행 가능한 사양을 만들어 충분한 기능 검증을 수행한다. 두번째로 S/W로 구현할 부분과 H/W로 구현할 부분을 나누어, S/W 부분의 인터페이스는 사이클 정확도를 갖도록 기술하며 기능 블럭은 기존 S/W 개발 환경을 사용하여 구현한다. H/W 부분의 IO 는 다양한 추상화단계로 이벤트를 기술하고 내부 동작은 기능에 기반을 두고 작성한다. 이 사양이 만족해야 할 시스템 요구 성능을 발휘하도록 성능분석을 수행하고, 이 결과가 S/W, H/W 분할 과정과 인터페이스 구체화 과정에 영향을 미친다. 시스템 성능을 내는 이 사양을 기초로 하여 사이클 정확도를 갖는 H/W 부분은 변환 프로그램을 이용하거나 직접 HDL RTL 설계로 변환한다.

이 방법은 기존 C/C++ 프로그램 개발자와 VHDL/Verilog 설계자가 쉽게 적응할 수 있어 기존 ASIC 개발자가 저렴한 비용으로 시스템 통합 설계 및 검증을 통하여 SoC를 개발하고자 할 때 특히 더 적합하다.

I. 서 론

시스템 설계과정을 시스템 개념을 구체화하는 과정으로 볼 수 있다. 구체화하는 과정이란 시스템 개념을 실현하기 위한 요구사항(requirement) 및 기능들을 정의/기술하고 검증한 후 요구와 기능을 수행할 수 있는 구조(architecture)를 찾아내는 과정을 말한다. 이때 구조적 특성에 의한 상세정보에 따라 기술수준(description level)을 말할 수 있으며 상세정보가 높을수록 기술수준은 낮다 혹은 추상화 수준(abstraction level)이 낮다고 할 수 있다.[1] 위와 같은 추상화단계(abstraction level)에 비추어 실재 시스템의 설계를 어느 단계에서부터 시작할지를 고려해 보아야 한다. 비교적 하위단계에서부터 설계되는 시스템은 구현과 검증에 시간과 노력이 많이 소요되며, 미처 검증되지 못하는 시스템의 오류가능성이 높아지고, 차후에 일부 변형된 시스템 설계에 소모되는 비용이 크게 줄어들지 않는 등 단점을 가지고 있다. 따라서 시스템 설계의 추상화단계를 높이는 것이 매우 중

요하다고 할 수 있다. 이에 고수준설계(high-level design)가 의미를 갖는다.

또 설계과정에서 상위단계는 하위단계로 구현상의 요구사항을 제공하고, 하위단계는 이를 바탕으로 구현을 진행한다. 그리고 진행된 결과로부터 구현정보를 상위단계로 피드백함으로써 구현의 정확도 검증이나 문제발생시 상위모델의 재설계를 유도하는 식으로 최초의 설계목적을 달성할 수 있도록 한다. 최근 이동무선장비를 중심으로 한 시스템 설계 경향도 시스템 구현유무보다는 시스템의 최적 성능이나 전력소모 등의 문제에 중점을 두는 경향에 비추어 볼 때, 위와 같이 설계과정이 최상위단계에서부터 최하위까지 유기적으로 연결되는 구조가 필요하다. 이는 고수준설계가 필요한 또 다른 이유이다.

현재 진행되는 설계과정에서는 각 부분이 구현된 후 전체적인 시스템 통합검증과정이 보드가 구성된 후에야 가능하다. 그전에는 주로 상위단계에서 주어지는 요구사항을 일방적으로 설계단위 별로 준수하기 위해 노력했던 경우가 많았다. 이런 과정을 반복하면 잘못 지정된 요구사항을 만족하기 위한 설계의 불필요한 설계요소가 발생하고 결과적으로 부적절한 시스템이 구성될 수 밖에 없다. 그리고 각 인터페이스 부분의 오류나 성능검증이 이뤄지지 않은 채 진행되므로 설계규모가 커지는 현 추세에서는 시스템 설계초기부터 전체를 통합하여 요구사항분석 및 오류/성능 검증을 하는 것이 필요하다. 현재 이를 위한 부분적인 문제해결을 위한 도구(tools)나 언어(languages)는 있지만 어는 것도 만족할 만한 효과를 내지는 못하는 실정이다.

위에서 언급한 고수준설계의 필요성과 설계초기 통합검증의 대안으로 SystemC를 고려할 수 있다. 상위단계에서 H/W나 S/W를 같은 환경에서 기술할 수 있도록 기존의 C++를 확장한 형태로 개방형 라이센스로 제공되고 있기 때문에 저렴한 수단이 될 수 있다. 아래에서는 이런 SystemC의 특징, 구문론/의미론적 구조를 간략히 소개하고 SoC 설계에 어떻기 이용될 수 있는지 언급한다.

II. SystemC 특징

SystemC는 소프트웨어 알고리듬, 하드웨어 아키텍처, 인터페이스를 Cycle-accurate model로 효과적으로 생성하기위해 만들어진 C++ class library와 방법론(methodology)를 말한다. SystemC와 기존 C++ 개발환경을 사용해서 다양한 알고리듬 검토 및 system-level model의 생성, 시뮬레이션, 검증, 최적화 등을 수행하고 HW/SW 팀에게 시스템의 실행 가능한 명세(executable specification)를 제공한다.[2]

C나 C++는 간결하고 효과적인 시스템 기술수단 이므로 줄곧 소프트웨어 알고리듬이나 인터페이스를 작성하는데 적당한 언어로 사용되었으며, 이미 많은 디자이너가 이 언어에 익숙할 뿐만 아니라 많은 개발환경이 구비되어 있다. 그러나, 시스템을 기술하는데 필요한 concurrency, hierarchy, coordination 등의 요소들이 부족하다. 이에 SystemC는 이런 단점을 보완하기 위하여 기존의 표준 C나 C++에 특별한 구문구조를 추가하지 않고 C++ OOP 언어구조를 활용하여 보완하였다.

SystemC의 기본적 구조는 기존의 HDL과 유사한 Module, Process를 가지고 있으며, Module 간의 통신을 위해 Port, Signal이 제공된다. 사용되는 데이터형(Data type)은 표준 C++에서 사용되는 기본형뿐만 아니라 임의의 비트사이즈 데이터형을 사용할 수 있다. 이를 이용한 사칙, 논리연산은 물론, 하드웨어부분 기술에 유용한 비트선택(bit-selection), 비트병합(concatenation) 등의 연산도 제공되고, SystemC 환경에서 설계 상세화(refinement) 과정을 일관되게 제공하기 위해 데이터형은 floating type, fixed type, bit-level 단계로, 시간개념은 untimed, bus-cycle accurate, cycle-accurate 단계로, 그리고 다양한 구조의 모듈간 통신(communication)을 제공한다.

그렇지만 SystemC를 이용한 SoC 설계를 하려면 현실적인 제약사항을 고려해야만 한다. SystemC는 근본적으로 cycle-based simulator를 내장하고 있으며 동질모델(homogeneous model)을 대상으로 한기 때문에 SystemC를 이용한 시뮬레이션을 수행할 때, 자신을 마스터(master)로 해서 다른 이질모델을 스레이브(slave)로 결합하는 것보다는 SystemC 시뮬레이션 영역에 포용 가능한 하위 시스템을 대상으로 하는 것이 바람직하고, 이런 각 하위시스템을 결합하여 시뮬레이션이 가능한 상위 시스템을 따로 작성하는 것이 필요하다. SystemC는 co-design에서 요구하는 과정(partitioning, architecture exploration, performance simulation) 등을 수행하는데 적절한 해결책을 아직 제안하지 못하고 있으며, 구현단계에도 현재까지는 수작업(manual refinement)에 의존하므로 종국에 HDL 형태로 변형/검증의 과정을 별도로 다시 필요로 한다. 또 이와 같은 상세화 과정은 동적 시뮬레이션(dynamic simulation)에만 의존한 검증절차를 이용한

다. 그리고 디자인이 상세화가 진행되면서 실제 탑재 또는 구현되는 아키텍처의 효과(arbitration, delay, etc)를 추정하는 작업을 할 수 없거나 SystemC 이외의 환경에 의존할 수 밖에 없다. 마지막으로는 SystemC 는 Co-verification 환경이 아니라는 점이다. 비록, Co-verification 단계에서 하드웨어 측면에 Gate-level 에서 추출한 타이밍정보를 내장한 상위 모델로 SystemC 를 사용하는 경우는 가능하겠으나 아직은 S/W 부분을 결합한 Co-verification 커널(kernel)로서는 적합하지 않다.

III. SystemC 를 이용한 SoC 방법론

이러한 제약조건을 고려하여 다음과 같이 SystemC 를 사용하여 SoC 설계를 할 수 있다. 우선, 미리 언급한 대로 SystemC 로 시스템 전체를 기술하는 것은 적합하지 않으므로 [그림 1]과 같이 SystemC 의 특징을 고려한 적당한 영역을 정한다. 하드웨어 영역도 event-driven simulation 이나 continuous signal simulation 을 요구하는 것은 적당하지 않고, 디자인의 내부 구성으로 보면 콘트롤과 데이터가 적절히 혼합된 부분이 적합하다. S/W 영역은 응용프로그램(application program)이 아니라 H/W 와 연관이 많은 디바이스 드라이버를 중심으로 선택한다. 만약 응용프로그램을 포함한다면 분할(partitioning)과 성능검증 시뮬레이션(performance simulation) 등과 같은 작업에서는 제외되어야 한다.

이렇게 지정된 영역에 대해서 순수한 기능동작(functional behavior)을 기술한다. 이렇게 만들어진 모델은 타이밍 개념은 없고 각 이벤트의 논리적 전후관계만 존재하는데, 형태상으로 communicating concurrent processes 로 생각할 수 있다. ([그림 2])

다음 단계는 순수한 기능동작을 검증하고, 구체적인 아키텍처로 매핑(mapping)이 될 현재 설계중인 시스템의 동작을 보장하기 위해, 각 프로세스(process)가 처리되어야 하는 지연성분(latency)을 지정해 준다. 그리고 다시 한 번 시뮬레이션을 통해 성능측면에서 기준에 합당한지를 검증한다. ([그림 3])

이렇게 정해진 타이밍 요구사항(timing requirement)을 만족하도록 실제 구현이 가능한 구조가 어떤 것인지 파악하고 하드웨어와 소프트웨어의 특징에 따라 분할을 수행한다. ([그림 4])

다음은 분할이 수행된 후 하드웨어의 인터페이스를 구체화하는 단계로, 각 인터페이스 포인트에 원하는 프로토콜(protocol)을 지정하고 시뮬레이션을 수행한다. 하드웨어는 동작이 정적으로 정해지기 때문에 이 단계에서 타이밍적으로 상당히 정확한 구조를 갖게 된다. 소프트웨어는 본질적인 동적특성 때문에 co-verification 단계에서 잠정적으로 지정된 성능에 대한 검증이 더 필요하다. ([그림 5])

하드웨어 부분에 지정된 요구사항(타이밍, 면적, 전력소모)을 만족하도록 구현과정이 뒤따른다. 데이터형은 고정 소수점 데이터형(fixed point data type)으로 변환하고, 스케쥴링(scheduling)을 수행하여 RT-level SystemC 단계까지 진행한다. 이 과정 중에 다수의 시뮬레이션을 통해 기능이 일관성 있게 구현되도록 한다. ([그림 6])

지금까지의 과정을 수행하고 나면, 미리 고려하였거나 적합하다고 생각되는 목적구조(target architecture)를 지정한다. 이때 대체적인 아키텍처 선택 이후에 세부적으로 조정 가능한 요소를 확인하고, 이후에 연계되는 co-verification 단계에서 체계적으로 적합한 조정치를 찾을 수 있도록 한다.([그림 7])

마지막은 co-verification 단계로서 가상구조(virtual architecture)를 대상으로 이전 단계에서 추정한 기능과 성능에 도달했는지 검증하고, 문제가 발생했을 때는 오류탐색의 과정을 수행한다. 이 단계의 결과는 상위 단계에 피드백(feedback)되어 재설계나 차후에 변형/개선된 시스템 설계에 활용될 수 있다.

제안한 방법론은 다른 환경에서도 이용될 수 있는 방법론이기는 하지만 이미 여러 분야의 많은 엔지니어에게 친숙한 C/C++환경에서 하드웨어 및 소프트웨어를 자유롭게 기술하고 시뮬레이션할 수 있는 저렴한 환경을 제공하는 것이 장점이다.

IV. 결 론

SystemC 는 하드웨어 엔지니어 입장에서 보면 RT-level 에서의 설계되었던 환경을 보다 높은 추상화 단계(abstraction-level)로 끌어올릴 수 있고, 개발초기부터 소프트웨어와 상호동작(interaction)을 검증할 수 있는 장점을 갖는다. 반면에 소프트웨어 엔지니어는 디바이스 드라이버와 같은 비교적 하드웨어 의존적 소프트웨어의 개발을 조기에 보다 구체적으로 시작할 수 있어서 전체적인 개발기간을 단축할 수 있다. 개발과정을 진행하면서 언제라도 부분적인 검증뿐만 아니라 전체 시스템 관점에서 동작을 검증할 수 있게 된다. 또한 제한적이기는 하지만 분할을 통해 성능트레이드오프(performance trade-off)를 관찰할 수 있다. 비록, 다른 co-design 관점에서 보면 다양한 편의성을 제공하지 못하지만 기존의 방법론을 이용하는 엔지니어라면 제안된 방법론에 매우

쉽게 적응할 수 있을 뿐만 아니라 경제적인 설계 환경을 갖출 수 있다.

참 고 문 헌

[1] VSI Alliance
- Architecture Document Version 1.0
- System-Level Interface Behavioral Documentation Standard Version 1.0

[2] Synopsys, INC., et al. "SystemC Version 1.1 Beta User Guide"

[3] Jorgen Staunstrup and Wayne Wolf
Hardware Software Co-design: Principles and Practices
Kluwer Academic Publisher, 1997

[4] Cadence "VCC Modeling Reference Manual - Version 1.2"

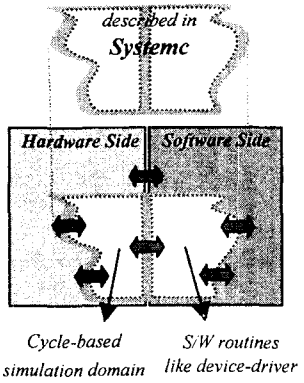

그림 1. 설계영역 선택

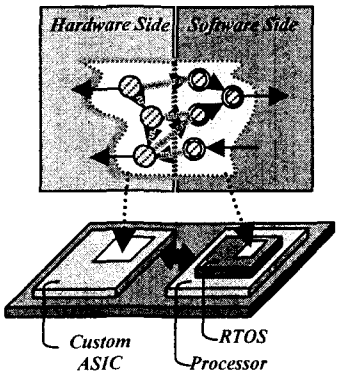

그림 7. Target Architecture 지정

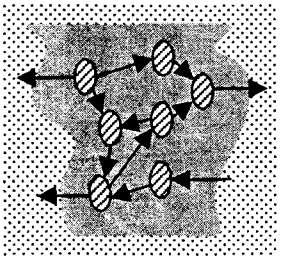

그림 2. Pure Function 설계

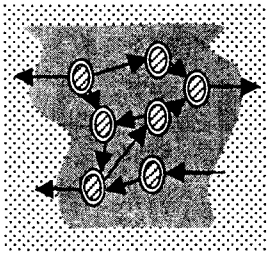

그림 3. Delay factor 추가

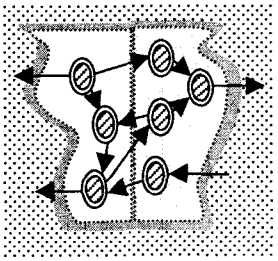

그림 4. 분할

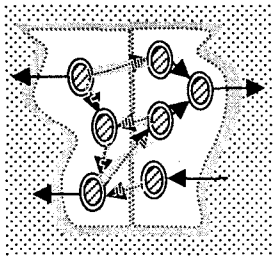

그림 5. Protocol 지정

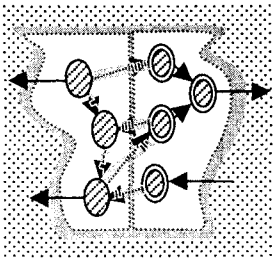

그림 6. RTL SystemC 변환