네트워크 디바이스의 프로토타입 개발 환경을 위한 시스템-온-칩 시뮬레이터와 네트워크 시뮬레이터의 통합 시뮬레이터 설계 및 구현

이호응* · 박수진* · 곽동은* · 박현주*

A Design of a Co-simulator that Integrates a System-on-Chip Simulator and Network Simulator for Development Environments of Prototype Network Devices

Hoeung Lee* · Soojin Park* · Dongeun Gwak* · Hyunju Park*

본 논문은 2단계 BK21 사업의 지원을 받아 수행된 연구임

요 약

무선 통신 프로토콜에서 하위 계층을 담당하는 부분은 네트워크 디바이스이다. 네트워크 디바이스는 하드웨어/소프트웨어로 구성되기 때문에 시스템-온-칩 시뮬레이터를 이용하여 설계할 수 있다. 하지만 네트워크 디바이스는 다양한 상위 계층 통신 프로토콜과 상호 동작하기 때문에 시스템-온-칩 시뮬레이터뿐 아니라 네트워크 시뮬레이터의 지원이 필요하다. 그러므로 이 두개의 시뮬레이터를 결합하면, 이러한 요구를 만족하는 네트워크 디바이스의 개발 환경이 될 수 있다. 본 논문에서는 이러한 환경을 제공하는 통합 시뮬레이터를 제안한다. 제안하는 통합 시뮬레이터는, 통합으로 인한 성능 저하가 발생하지 않는다. 또한, 각 시뮬레이터의 커널 구현에 독립적이므로 통합이 용이하다.

ABSTRACT

In the wireless communication protocols, a network device is responsible for the operation of lower-layers. The network device consists of hardware and software modules, so it can be designed using system-on-chip simulator. The simulator design needs the support of a network simulator as well as system-on-chip simulator, because the network device interact with various higher layer communication protocols. Therefore the co-simulator can become a development environment of the network device through the combination of them. In this paper we propose a co-simulator combining these two simulators. The proposed co-simulator does not degrade performance due to integrations. Also, it is easy to integrate them because the implementation of the kernel is independent.

키워드

통합시뮬레이터, 네트워크 시뮬레이터, SystemC, NS-3, 네트워크 디바이스.

Key word

Co-simulator, Network Simulator, SystemC, NS-3, Network Device.

* 한밭대학교 정보통신전문학원 전파공학과

접수일자 : 2009. 09. 14
심사완료일자 : 2009. 10. 07

I. 서 론

유무선 매체 기술이 지속적으로 발전함에 따라 이를 기반으로 하는 다양한 통신 서비스들이 출현하고 있다. 특히 무선 통신 서비스는 통신 거리, 통신 속도, 이동성, 소모 전력 등에 따라 특화된 무선 통신 프로토콜을 기반으로 한다. 예를 들어 IEEE 802.15.4는 저 전력, 저속 특징을 기반으로 하는 센서 네트워크 서비스에 적합한 프로토콜이고, IEEE 802.15.3a는 UWB 기술을 기반으로 근거리 고속 무선 통신 서비스를 위한 프로토콜이며, IEEE 802.11n은 고속 무선 인터넷 연결 서비스를 지원하는 프로토콜이다. 유비쿼터스 컴퓨팅을 기반으로 하는 다양한 응용이 제시되면서 이러한 무선 통신 프로토콜의 종류는 더욱 증가하고 있다. 업계에서는 이러한 통신 매체 기술의 표준을 구현한 네트워크 디바이스를 모듈 형태로 구현하여 시장에 출시한다. 이 때, 시장적시성은 사업의 성공을 위해 가장 중요한 요소 중 하나이기 때문에 네트워크 디바이스의 신속한 개발 및 검증을 할 수 있는 시뮬레이터 환경이 필요하다.

이러한 네트워크 디바이스를 위한 시뮬레이터 환경을 고려할 때 사용 목적에 따라 두 가지 시뮬레이터 중 하나를 선택할 수 있다. 첫 번째는 NS-2 또는 NS-3 와 같은 네트워크 시뮬레이터를 이용하여 대규모 네트워크를 구성한 후 통신 프로토콜의 절차적인 동작이나 순수 알고리즘만을 평가하는 것이다. 두 번째는 SystemC와 같은 시스템-온-칩 설계를 위한 시뮬레이터를 이용하여 통신 프로토콜을 가상의 하드웨어와 소프트웨어로 구현한 후 단일 모듈에 대한 동작 가능성을 검증하고 실제 시스템의 성능을 예측하는 것이다. 본 논문에서는 이러한 시뮬레이터의 장점을 모두 활용할 수 있도록 통합 시뮬레이터를 구축하여 네트워크 디바이스의 프로토타입 개발 환경을 제공하는 것을 목표로 한다.

그림 1은 이러한 목표를 달성하기 위한 본 논문의 통합 시뮬레이터 접근 방법을 나타낸다. 그림 1의 (a)는 다양한 노드와 이들 간의 연결로 이루어진 네트워크 구성 예제를 보인다.

각각의 노드는 (b)에 제시된 바와 같이 다수의 통신 계층으로 이루어져 있으며, 특히 물리 계층부터 데이터 링크 계층까지를 TCP/IP 모델에서는 네트워크 디바이스 계층으로 사상한다. (c)에서는 본 논문의 통합 시뮬레이터 접근 방법을 나타낸다. 즉, 시스템-온-칩 시뮬레이터를 이용하여 네트워크 디바이스를 설계하고, 그 주변의 통신 계층이나 대규모 통신망은 네트워크 시뮬레이터를 이용하여 설계가 가능한 통합 시뮬레이터를 제공한다면, 개발자가 설계한 네트워크 디바이스에 대한 하드웨어-소프트웨어 설계의 검증뿐 아니라 대규모 네트워크가 구성된 환경에서 네트워크 디바이스의 다양한 관점에 대한 성능 평가를 할 수 있다.

본 논문의 통합 시뮬레이터 접근 방법과 유사한 연구가 [1]에서 진행되었다. [1]의 연구에서는 이러한 목표를 만족하기 위해 네트워크 시뮬레이터인 NS-2와 시스템-

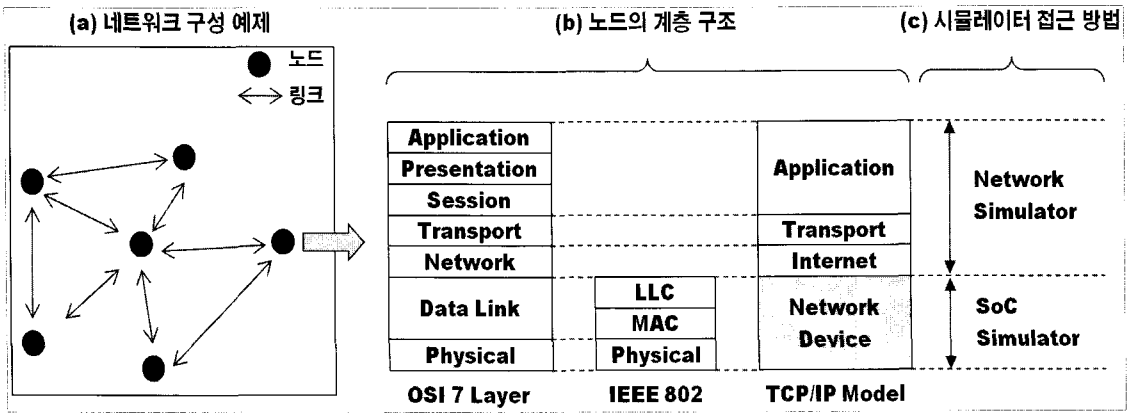

그림 1. 통합 시뮬레이터 접근 방법
Fig. Co-simulator approach

온-칩 시뮬레이터인 SystemC를 결합하는 통합 시뮬레이터에 대한 설계를 제안하였다.

그러나 [1]의 연구에서 제안한 통합 시뮬레이터는 시뮬레이터의 구조, 활용 및 유지보수 측면에서 문제점이 있다. 먼저 통합 시뮬레이터의 구조측면에서는, 각 스케줄러에서 이벤트 처리를 위한 시간 동기화 작업에 연산 부담이 있고, 시뮬레이터간 통신을 위한 데이터 포맷 설계는 추상적인 수준으로 특정한 시뮬레이션 디자인 모델을 위해 구체적인 설계를 하지 않았다. 다음으로 통합 시뮬레이터의 활용 측면에서는, NS-2의 근본적인 시뮬레이터 구조 한계로 인해 프로토콜 설계 및 구현 작업이 복잡하여 유연성이 낮고 다른 분석 도구와 통합하기 위한 확장성이 부족하다. 끝으로 유지보수 측면에서는, 시뮬레이터를 통합하기 위해 각 시뮬레이터의 커널에 대한 세부적인 수정이 필요하므로 각 시뮬레이터가 업그레이드 될 때마다 내부적으로 커널을 수정하는 수작업이 필요하므로 개발자 부주의로 인한 오류의 원인이 될 수 있다.

본 논문에서 제안하는 통합 시뮬레이터는 그림 1의 시뮬레이션 디자인 모델을 지원하기 위해 기존 연구의 단점을 극복한다. 제안하는 통합 시뮬레이터의 주요 특징은 다음과 같다. 첫 번째, 시스템-온-칩 시뮬레이터와 네트워크 시뮬레이터를 통합하는 효율적인 구조를 제안하여 기존 연구에서 문제시 되었던 시뮬레이터의 구조적인 측면과 유지보수적인 측면의 단점을 해결하고 시뮬레이터 동기화를 위한 연산 부담을 완화한다. 두 번째, 네트워크 시뮬레이터기반 상위 계층과 시스템-온-칩 시뮬레이터기반 하위 계층을 연결하는 일관된 인터페이스를 제공하여 시뮬레이터의 활용성을 강화한다.

논문의 구성은 다음과 같다. 2장에서는 기존 통합 시뮬레이터 연구의 한계 및 단점을 세부적으로 분석한 후, 통합 시뮬레이터의 설계 요구 사항을 도출한다. 3장에서는 2장에서 도출한 요구 사항을 만족하는 설계를 제안한다. 4장에서는 실험 결과를 통해 제안하는 통합 시뮬레이터의 성능을 평가한다. 5장에서는 결론을 맺는다.

II. 관련 연구

본 장에서는 기존 네트워크 시뮬레이터와 시스템-온-칩 시뮬레이터의 적용 분야 및 특징에 대해 분석한 후, 본 논문에서 지향하는 접근 목표와 동일한 기존의 통합 시뮬레이터를 분석한다. 끝으로 분석 결과를 통해 기존 통합 시뮬레이터에서 개선해야 할 사항을 도출한다.

2.1 네트워크 시뮬레이터

통신 프로토콜 고유의 성능을 평가하기 위한 네트워크 시뮬레이터는 OPNET[2], QualNet[3]과 같은 상용 제품과 OMNeT++[4], NS-2[5], NS-3[6]의 오픈 소스 기반 시뮬레이터로 나뉠 수 있다. 각 시뮬레이터가 지향하는 목표에 따라 적용 분야의 차이점이 있지만, 노드의 구성/배치 및 성능 평가 측면에서 이러한 네트워크 시뮬레이터들의 공통점이 있다. 각 노드는 시뮬레이션 시나리오에 따라 하위 계층부터 상위 계층 까지 다양한 프로토콜로 구성할 수 있고 각 계층에 대해 임의의 파라미터를 설정할 수 있다. 이렇게 구성된 노드들은 2차원 또는 3차원 공간상에 배치하여 네트워크를 구성한 후, 최종적으로 시뮬레이션을 수행하여 해당 시뮬레이션 시나리오에서 원하는 결과를 추출할 수 있다. 이러한 네트워크 시뮬레이터에서의 성능 평가 결과는 주로 프로토콜 고유의 동작이 네트워크에 미치는 영향에 관한 것이다. 즉, 네트워크 시뮬레이터는 프로토콜이 실제의 특정 하드웨어와 소프트웨어로 구현되었을 경우의 성능을 평가하는 것이 아니라 프로토콜의 정책적인 측면이나 알고리즘 및 이론을 평가하는 것이 주요 용도이다.

이러한 네트워크 시뮬레이터 중에서 NS-2는 오픈 소스 기반이라는 접근 용이성과 다양한 프로토콜을 지원한다는 장점을 기반으로 전 세계의 많은 연구 기관 및 대학에서 그들의 연구에 대한 성능 평가를 위해 주로 사용한다. 하지만 과거 80년대에 개발된 NS-2 시뮬레이터의 커널은 현대의 다양한 유무선 통신 프로토콜의 구현을 적용하기에는 여러 가지 한계가 있다[7]. 최근에는 NS-2의 이러한 단점을 해결한 NS-3가 등장하였다. NS-2와 NS-3의 주요 차이점을 비교하면 다음과 같다.

- 다중 언어 지원 : C++와 OTCL 언어를 혼합하여 활용하는 NS-2에서는, 프로토콜 구현을 위해 C++언어를 주로 사용하고 구현된 프로토콜 모듈의 연결, 배치, 파라미터 설정 등의 시뮬레이션 시나리오에 해당하는 부분은 인터프리터 언어인 OTCL을 사용하여, 시뮬레이터 사용자의 편의성을 높이는 접근방법을 제공한다. 하지만 이런 접근 방법의 경우 시뮬레이션 시나리오 구성을 위해 오로지 OTCL언어만 활용하도록 제한을 하고 있으므로 C++ 언어에 익숙한 사용자의 요구 사항은 만족할 수 없다. 반면, NS-3는 C++와 Python 언어를 채택하였고, 시뮬레이션 시나리오를 위한 언어는 사용자의 편의에 따라 선택하는 환경을 제공한다.

- 시뮬레이터 구조 : NS-2는 두 개의 언어를 채택한 본래의 목적과는 다르게 시뮬레이터의 커널에 해당하는 모듈들은 엄격한 분할 기준 없이 C++과 OTCL 언어에 분산하여 구현된 결과를 확인할 수 있다[5]. 이러한 특징으로 인해 모듈들 간의 결합도(Coupling)가 높고 시뮬레이터 구조가 복잡하며 시뮬레이터 변경을 위한 유연성이 낮다. NS-3는 이러한 NS-2의 단점을 해결한다. NS-3는 시뮬레이터의 구조적인 측면에서 NS-2의 단점을 극복하기 위해 시뮬레이터의 커널을 처음부터 다시 설계한 결과, Smart Pointer/Functor/Callback 등과 같은 다양한 현대적인 C++ 디자인 패턴을 적용하여, 사용자가 프로토콜을 보다 객체 지향적으로 구현할 수 있도록 유도하고 있으며 Python 언어는 보조재의 용도로 사용자의 편의에 따라 활용할 수 있도록 한다.

위에서 제시한 바와 같이 NS-3는 시뮬레이터 구조, 활용 및 성능 측면에서 다양한 장점이 있으므로, 본 논문에서 제안하는 통합시뮬레이터의 네트워크 시뮬레이터는 NS-3로 채택한다.

2.2 시스템-온-칩 시뮬레이터

그림 2는 하드웨어와 소프트웨어의 동시 설계를 요구하는 시스템-온-칩의 개발 및 검증 절차를 나타낸다. System Design 단계에서는, 구현할 대상에 대한 요구사항을 도출하는 것을 시작으로 높은 추상화 수준에서 기능적 검증과 알고리즘의 성능을 평가한다. 이 단계에서는 실제 하드웨어와 소프트웨어 수준에서 정확한 시간 모델을 기반으로 하는 검증을 수행하지 않으며, 주로 UML이나 MATLAB 도구를 이용하여 작업을 진행한다. Partitioning 단계는 여러 기능적 모듈을 하드웨어와 소프트웨어로 분할하는 작업을 담당한다. 이러한 분할 과정은 System Design 단계에서 도출한 성능 평가 정보를 바탕으로 진행할 수 있다. 이어서 각 하드웨어와 소프트웨어의 설계 및 구현 작업을 병렬적으로 진행한 후 통합과정을 마무리 하면 최종적으로 Physical Design 단계에서 칩으로 생산한다. 이러한 진행 과정에서 마지막 단계를 제외하면, 시뮬레이터를 이용한 개발 환경에서 대부분의 작업을 진행하는 것이 일반적이다.

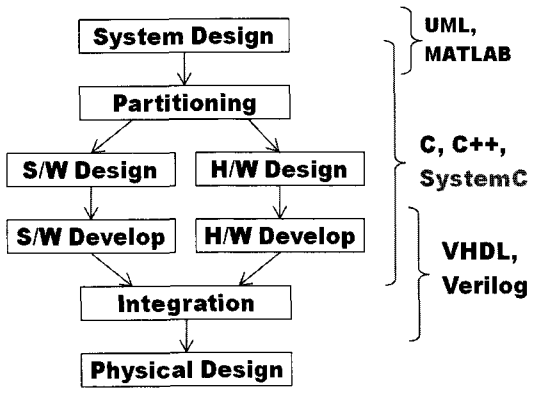

그림 2. 시스템-온-칩의 개발 및 검증 절차
Fig. 2 design and verification procedures for System-on-chip

시뮬레이터의 활용은, 각 단계를 진행하면서 높은 추상화 수준에서 상세 수준으로 이동한다. 예를 들어 Partitioning 단계부터 하드웨어와 소프트웨어의 프로토타입 구현까지는 TLM(Transaction Level Modeling) [8] 수준을 지원하는 시뮬레이터에서 작업을 진행한다. 이 과정을 지원하는 대표적인 시뮬레이터 환경이 SystemC 이다. 이어서 상세 수준으로 설계를 진행함에 따라 정확한 시간 모델로 진행하기 위해 RTL(Register Transfer Level) 수준의 시뮬레이터를 활용한다. VHDL과 Verilog는 RTL을 지원하는 대표적인 하드웨어 설계 환경이고, 소프트웨어 부문의 경우 정확한 시간 모델을 구현한 ISS(Instruction Set Simulator) 기반 환경을 이용한다.

이처럼 상세 시간 모델 단계에서는 하드웨어와 소프트웨어를 검증하는 시뮬레이터 환경이 별도로 존재하

지만, 초기 프로토타입을 개발하고 검증하는 단계에서는 SystemC 언어 기반에서 하드웨어와 소프트웨어를 동시에 설계한다.

SystemC는 하드웨어와 소프트웨어의 통합 개발 및 검증 환경을 지원하는 많은 기능을 제공하지만, 상세 시간 모델을 적용하기에는 무리가 있다[9]. 그러므로 SystemC는 초기 시스템-온-칩의 프로토타입 수준의 개발 및 검증을 위한 시뮬레이터 환경으로 적합하다. 본 논문의 목표는 네트워크 디바이스의 프로토타입 개발 환경을 지원하는 것이므로, 시스템-온-칩 시뮬레이터 부문에 SystemC를 채택한다.

2.3 통합 시뮬레이터 및 통합 디자인

[1]의 연구에서는, 하드웨어/소프트웨어 통합 설계에 장점이 있는 SystemC와 네트워크 프로토콜/알고리즘 설계에 장점이 있는 NS-2를 결합하여, 네트워크 기능을 갖춘 임베디드 시스템을 위한 모델링 및 시뮬레이션 환경을 제안하였다. [10]의 연구에서는 [1]의 통합 시뮬레이터를 기반으로, 네트워크 측면과 시스템 측면을 동시에 고려하면서 특정 시스템을 설계하는 방법론을 제안하여 [1]에서 제안한 시뮬레이터의 활용 가능성을 보였다.

그림 3은 [1]의 연구에서 제안한 통합 시뮬레이터의 구조를 나타낸다.

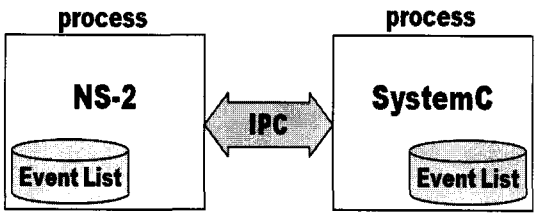

그림 3. 기존 연구의 통합 시뮬레이터 구조
Fig. 3 Co-simulator architecture of previous works

이 그림에서 제시한 바와 같이 두 시뮬레이터는 각각 독립적인 프로세스로 실행하고 기반 운영체제의 IPC (Inter-Process Communication)을 이용하여 프로세스 사이의 통신을 하도록 구성한다. 시뮬레이터 통합을 위한 주요 설계 부분은 시간 동기화와 데이터 교환 처리 부분으로 나뉜다.

먼저 시간 동기화 부분에 대해 살펴보면, 시간 동기화란 각 시뮬레이터에 개별적으로 존재하는 이벤트들을 종합하여 엄격한 시간 순서 실행을 보장하는 것이다. [1]의 연구에서는 이러한 문제를 해결하기 위해 각 시뮬레이터가 자신의 이벤트 리스트에 있는 이벤트를 처리하기 전에 상대측 시뮬레이터의 이벤트에 대한 정보를 검사한 후 진행한다. 예를 들어 상대측 시뮬레이터의 이벤트가 시간 순서상에서 먼저 실행해야 한다면, 상대측 시뮬레이터의 해당 이벤트가 실행될 때 까지 대기한 후 실행 권한을 얻어 실행한다. 이러한 경우 각 시뮬레이터는 매번 하나의 이벤트를 처리할 때마다 상대측 시뮬레이터의 이벤트 시간 정보를 확인하는 작업을 처리해야 한다. 이벤트 시간 정보의 확인 작업은 상대측 시뮬레이터에 해당하는 프로세스와의 IPC 연결을 이용하며[11], 운영체제의 IPC 연산은 일반적으로 고비용의 연산이다[12]. 시뮬레이터의 이벤트가 대부분 소규모의 작업 단위로 이루어져 있고 시뮬레이터는 방대한 규모의 이벤트 개수를 처리해야 한다는 사실을 감안하면, 하나의 이벤트를 처리할 때마다 운영체제의 IPC 연산을 수행하는 것은 시뮬레이터의 성능을 급격히 저하시키는 원인이 될 수 있다. 이미 통합 시뮬레이터에 관한 다수의 기존 논문에서는 시간 동기화 방법으로 IPC를 사용하는 설계는 많은 연산 부하가 발생하여 성능을 급격히 저하시킨다는 연구 결과를 제시하였고, IPC 연산 횟수를 줄이거나 제거하는 방법이 시뮬레이터의 성능을 높일 수 있다는 사실을 입증하였다[13].

다음으로 데이터 교환 처리 부분에 대해 살펴보면, 데이터 교환 처리란 각 시뮬레이터에서 설계한 모듈을 서로 연결할 수 있는 인터페이스 구조를 정의하는 것이다. [1]의 연구에서는 NS-2의 Agent와 SystemC의 Port를 서로 연결할 수 있다는 가능성만 추상적으로 제시하였고 구체적인 설계에 대해 제안하지 않았다. 데이터를 교환하는 부분은 시간 동기화와 밀접히 관련되어 있으므로 이러한 설계를 사용자에게 일임하는 것은 설계의 복잡도를 증가시킨다.

시뮬레이터의 활용적인 측면에서 살펴보면, 네트워크 시뮬레이터 부문에서 NS-2를 채택하고 있기 때문에 위의 2.1절에서 설명한 바와 같이 NS-3에 비해 설계의 편의성 및 확장성이 높지 않다.

끝으로 시뮬레이터의 유지보수 측면에서도 단점이 있다. [1]의 알고리즘을 적용하기 위해서는 시뮬레이터

스케줄러의 세부적인 부분을 수정해야 하기 때문에, 만약 시뮬레이터의 스케줄러가 상위 버전으로 변경된다면 통합 스케줄러를 위한 수정 작업을 다시 적용해야한다. 이러한 경우 각 시뮬레이터의 스케줄러에 대한 깊은 이해가 필요하므로 유지보수 작업이 어렵다.

이상의 내용을 종합하여 고찰하면, 시간 동기화 부분에서는 두 시뮬레이터간의 시간 정보를 교환하기 위한 연산 부담을 완화하는 설계가 필요하고, 데이터 교환 처리 부분에서는 이 기종 시뮬레이터에서 각각 설계한 모듈을 쉽게 통합할 수 있는 인터페이스 구조가 필요하다. 부수적으로는 NS-2의 근본적인 한계를 극복하는 네트워크 시뮬레이터인 NS-3를 도입하여 설계의 편의성을 높이고, 시뮬레이터 스케줄러의 수정 작업을 최소화 하여 유지보수에 대한 용이성까지 만족해야 한다. 본 논문에서는 이러한 요구사항을 만족하는 네트워크 디바이스의 프로토타입 개발 환경을 위한 통합 시뮬레이터의 설계를 제안하고 구현한다.

Ⅲ. 시스템-온-칩 시뮬레이터와 네트워크 시뮬레이터의 통합 시뮬레이터 설계

위의 2장 관련 연구에서 분석한 결과를 바탕으로, 본 논문에서 제안하는 통합 시뮬레이터의 시스템-온-칩 시뮬레이터와 네트워크 시뮬레이터는 각각 SystemC와 NS-3를 채택한다.

3.1 절에서는 단일 프로세스로 두 개의 시뮬레이터를 구동하는 간단한 시간 동기화 방법을 제안한다. 제안하는 시간 동기화 모듈은 시뮬레이터 간의 통신에 대한 연산 부담을 완화하므로 시뮬레이션 속도를 저하시키지 않는다. 또한, 두 시뮬레이터를 통합하기 위해 간단한 수정 작업만 요구하므로 통합이 쉽고 유지보수성이 높다.

3.2 절에서는, NS-3로 설계한 상위 계층 모듈과 SystemC로 설계한 하위 계층 모듈을 통합하여 데이터 교환을 할 수 있는 인터페이스 구조를 제안한다. 제안한 인터페이스를 이용하면 서로 다른 시뮬레이터에서 설계한 모듈들 간의 데이터 교환 방법은 통합 시뮬레이터의 세부 구현에 독립적이므로 사용자가 설계한 모듈의 복잡도를 낮춘다.

3.1 시간 동기화 설계

NS-3와 SystemC는 시간 순서로 정렬된 이벤트를 차례대로 처리하는 이벤트 기반 시뮬레이터라는 공통점이 있다. 특히 NS-3는 기본적인 형태의 이벤트 처리만을 지원하지만 SystemC는 이것 이외에 스레드 기반 이벤트 처리까지 지원한다. 그러므로 NS-3의 이벤트 처리 기능은 SystemC의 이벤트 처리 기능의 부분 집합이라고 할 수 있다.

이러한 사실을 바탕으로 하여, 본 절에서 제안하는 시간 동기화 설계의 착안점을 그림 4에서 제시한다. 그림 4에 나타낸 바와 같이, 만약 SystemC 스케줄러가 NS-3의 이벤트를 처리할 수 있다면, 단일 실행 흐름만 필요하게 되므로 SystemC와 NS-3를 하나의 프로세스로 통합할 수 있다. 하나의 프로세스로 통합하는 구조는 IPC 연산을 완전히 제거하므로 시뮬레이션 수행 속도 저하를 완화할 수 있다[13].

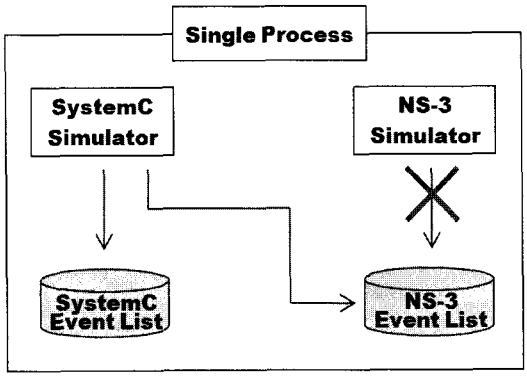

그림 4. 시간 동기화 설계의 착안점
Fig 4. a motivation of the time Synchronization design

이러한 착안점을 기반으로 하여, 그림 5는 SystemC에서 NS-3의 이벤트를 구동할 수 있는 단일 프로세스 기반 시간 동기화 설계를 제안하며, 동작 방식은 다음과 같다.

SystemC 기반의 설계는 다수의 SC_MODULE로 구성되어 있고, SC_MODULE은 내부에 독립적인 실행 흐름을 갖는 스레드 형태의 함수를 다수 포함할 수 있다[14]. 이러한 SC_MODULE의 특징을 이용하여, NS-3의 이벤트를 전용으로 처리하는 SC_MODULE을 만드는 것이

다. 그림 5에서 NS3Adaptor 가 이러한 처리를 담당하는 SC_MODULE 이다. NS3Adaptor의 내부에는 독립적인 실행을 갖는 Adaptor라는 이름의 함수가 존재하고 이 함수가 NS-3의 이벤트들을 NS-3 스케줄러에 대신하여 처리한다.

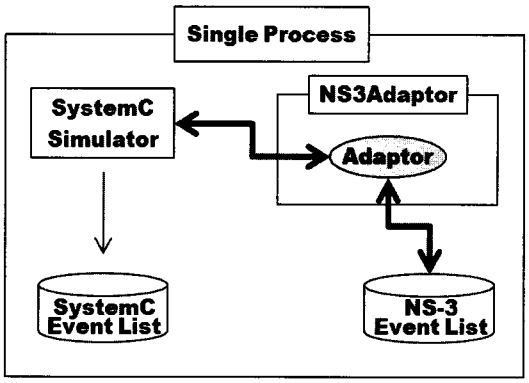

그림 5. SC_MODULE을 이용한 이벤트 처리
Fig 5. The event-handling using SC_MODULE

그림 6은 Adaptor의 자세한 동작 알고리즘을 제시한다. NS-3 이벤트 리스트의 가장 첫 이벤트는 가장 먼저 실행해야 할 이벤트가 담겨 있다. 이 이벤트를 추출하여 Event 변수에 저장한 후, Event의 실행 예약 시점에 해당하는 값인 Time을 추출하여 Adaptor의 대기 시간으로 설정한다. SystemC는 Time 시간이 지난 후 Adaptor를 다시 실행 가능 상태로 전환한다. Adaptor가 깨어난 시점은 바로 Event가 실행해야 할 시점이다. 그러므로 Event의 작업 내용이 담긴 함수 포인터인 Func을 실행한다. 이러한 과정을 반복하면 NS-3의 이벤트를 시간 순서대로 처리할 수 있다. 결국 Adaptor의 역할은, NS-3의 이벤트를 SystemC 스케줄러로 중계하는 역할을 담당하는 것이다.

이 알고리즘의 특징은 NS-3의 이벤트를 처리할 때 SystemC의 이벤트에 대해서 전혀 고려하지 않아도 된다는 점이다. 그 이유는 NS-3의 이벤트 처리를 중계하는 NS3Adaptor가 SystemC 기반 모듈이므로, SystemC 스케줄러는 NS3Adaptor와 SystemC의 다른 모듈들 간의 실행 순서를 자체적으로 처리하기 때문이다.

```
1   void NS3Adaptor::Adaptor()
2   {
3       Setup Phase;
4
5       do
6       {
7           Event := ExtractNS3Event();
8           Time = GetTime(Event);
9           wait( Time );
10          Func = GetFunction(Event);
11          Func();
12      }while( there are messages );
13  }
```

그림 6. 시간 동기화 알고리즘
Fig. 6 The algorithm of the time synchronization

끝으로, 그림 6의 알고리즘이 적용된 통합 시뮬레이터의 완전한 실행 순서는 그림 7에서 나타내며 (a)부터 (d)의 순서로 실행하며 자세한 처리 과정은 다음과 같다.

(a) 통합 시뮬레이터를 처음 구동하면, NS-3의 main() 함수가 먼저 실행되면서 NS-3의 예제 파일을 읽은 후 해당 예제에 따른 NS-3 이벤트 리스트를 생성한다.

(b) NS-3 이벤트 리스트 생성이 끝나면, 이벤트 처리 작업을 진행하기 위해 NS-3 시뮬레이터는 NS-3 스케줄러를 호출한다. NS-3 스케줄러는 이벤트를 실행할 책임이 있지만, 본 논문에서 제안하는 방법은 SystemC에서 NS-3의 이벤트를 처리하는 구조이므로 SystemC 시뮬레이터 호출함으로써 실행 권한을 이전한다.

(c) SystemC 시뮬레이터가 실행되면 (a) 단계와 마찬가지로 SystemC 예제 파일을 읽은 후 해당 예제에 따른 SystemC 이벤트 리스트를 생성한다.

(d) NS-3와 SystemC의 이벤트 리스트가 모두 생성되었으므로 그림 6의 알고리즘을 가동하여 시뮬레이션을 진행한다.

그림 7에서 제시한 실행 과정을 통해 알 수 있는 사실은, 본 알고리즘을 적용하기 위해 SystemC의 스케줄러

를 수정할 필요가 없고, NS-3의 스케줄러는 단지 실행 권한을 SystemC 에게 위임하는 간단한 수정만 필요하다는 것이다. 결론적으로, 각 스케줄러의 세부 내용에 독립적이기 때문에 알고리즘을 적용하기 쉽고 향후 각 시뮬레이터가 상위 버전으로 변경되었을 때 본 알고리즘을 신속히 다시 적용할 수 있다.

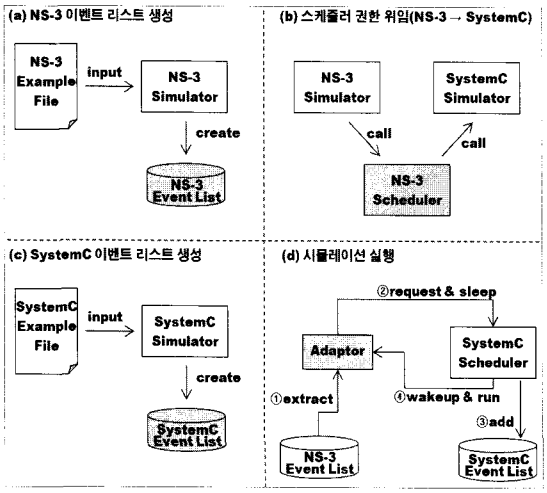

그림 7. 통합 시뮬레이터 실행 과정
Fig. 7 The execution process of the co-simulator

3.2 데이터 교환 설계

본 절에서는 SystemC로 설계한 네트워크 디바이스 모듈을 NS-3에서 제공하는 상위 계층과 연결하는 구조를 제안한다.

그림 8은 NS-3 노드 구조를 중심으로 통합 시뮬레이션의 디자인 모델을 나타낸다. NS-3 노드 구조에서 App, Socket, Protocol은 OSI 모델에서 3 ~ 7계층에 해당하고 Channel과 NetDevice는 1 ~ 2 계층에 해당하며 그림 1에서 제시한 바와 같이 인터넷 모델에서는 1 ~ 2 계층을 네트워크 디바이스 계층으로 표현한다. 여기서 SystemC 기반의 구현 환경을 제공해야 하는 부분이 1 ~ 2계층이다.

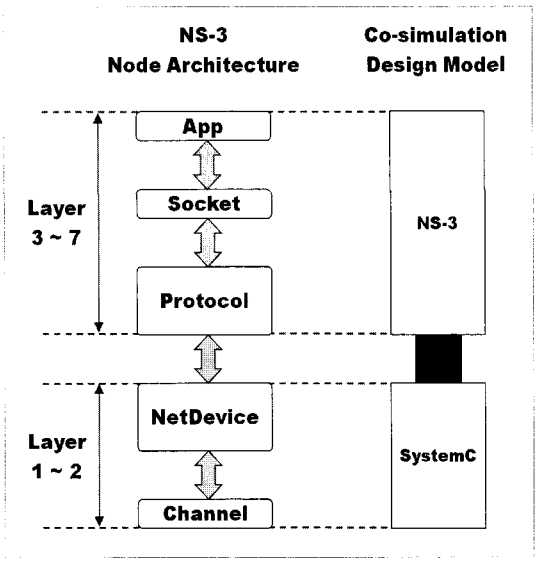

그림 8. NS-3 노드 구조와 디자인 모델
Fig. 8 NS-3 node architecture and the design model

그림 9는 본 절에서 제안하는 인터페이스 구조의 개념도를 나타낸다.

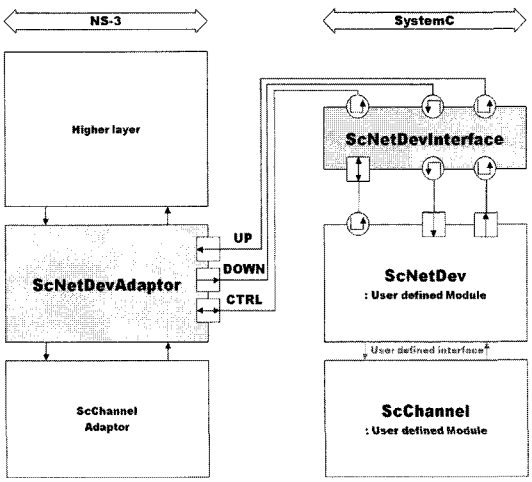

그림 9. NS-3와 SystemC의 연결 구조
Fig. 9 The connection architecture of NS-3 and SystemC

이 그림의 표기법은 SystemC기반 설계내용을 시각화할 때 주로 사용한다[15].

제안하는 설계에서 NS-3 노드 구조의 Channel 부분은 단지 노드의 기본 형태를 갖추기 위해 존재할 뿐 아무런 기능을 하지 않는다. NetDevice 부분은 ScNetDevAdaptor라는 이름으로 개정하였고 NS-3와 SystemC를 연결하는 가교역할만 담당한다. SystemC 영역에서는 ScNetDevAdaptor와 직접 연결하는 ScNetDevInterface 모듈이 있다. ScNetDevInterface는 NS-3의 세부적인 구현 특징을 캡슐화 하고 SystemC 구현 부분에게 일관성 있는 인터페이스를 제공한다. ScNetDev와 ScChannel은 각각 NS-3의 NetDevice와 Channel에 대응하는 SystemC 모듈 구현을 나타낸다. 즉, 이 부분은 본 논문에서 제안하는 통합 시뮬레이터의 사용자가 향후 구현하는 부분이다.

그림 10에서는 ScNetDevAdaptor 클래스의 상속 구조를 UML 다이어그램으로 나타낸다. ScNetDevAdaptor는 NS-3의 상위 계층과 통신함과 동시에 SystemC에서 동작하는 ScNetDevInterface와 통신해야 하므로 NS-3와 SystemC의 기능을 모두 포함해야 한다. 또한, NS-3와 SystemC의 기능을 동시에 필요하므로 각 시뮬레이터의 핵심 기능을 다중 상속한다. 그러므로 ScNetDevAdaptor는 NS-3로 구현된 상위 계층과의 패킷 송수신이 가능할 뿐 아니라 SystemC로 구현된 ScNetDevInterface와의 패킷 교환 및 세부 제어가 가능하다.

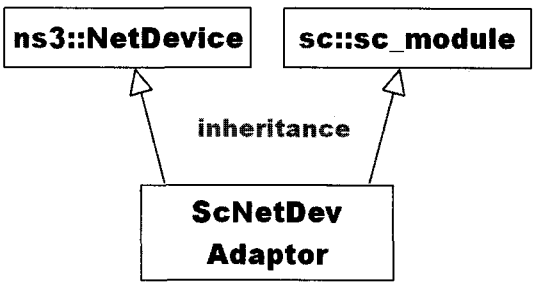

그림 10. ScNetDevAdaptor 클래스의 상속 구조
Fig. 10 The inheritance architecture of ScNetDevAdaptor class

그림 11은 ScNetDevAdaptor의 세부 구조를 나타낸다. ScNetDevAdaptor는 NS-3의 NetDevice로부터 상속된 여러 가지 매소드들을 재 정의하여 ScNetDevInterface에 연결하는 기능을 한다. NS-3의 NetDevice로부터 상속한 다양한 매소드들은 각 매소드의 특징에 따라 Up, Down, Control 로 분류한다. Up은 패킷 수신을 처리하는 매소드이고, Down은 패킷 송신을 처리하는 매소드이며, Control은 제어와 관련된 처리를 담당하는 메소드들이다.

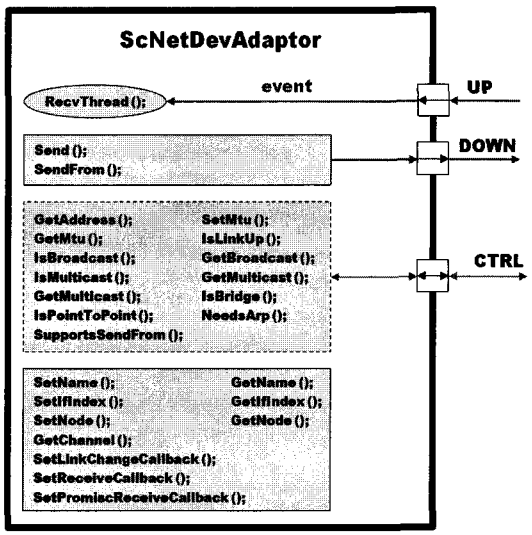

그림 11. ScNetAdaptor의 메소드와 SystemC 포트의 관계
Fig. 11 The relationship between ScNetAdaptor's method and SystemC's port

특히, 그림 11에서 타원으로 표시한 RecvThread 매소드는 수신 패킷을 NS-3로 구현된 상위 계층으로 전달하는 기능을 담당한다. 이 매소드가 필요한 구체적인 이유는 다음과 같다.

그림 7의 (d)에서 나타낸 바와 같이 NS-3의 이벤트를 실행하는 모듈은 NS3Adaptor라는 SystemC 모듈이다. NS3Adaptor가 NS-3의 이벤트 처리 대행을 위해 수면 상태에 있을 경우 SystemC 의 다른 모듈들이 실행되므로, ScNetDevAdaptor가 패킷을 수신한 경우는 NS3 Adaptor가 수면 상태에 있거나 실행을 대기하고 있는 상태이다. 이러한 경우 ScNetDevAdaptor가 NS-3의 메소드를 직접 호출 형태로 패킷을 전달한다면, NS3 Adaptor가 이벤트를 처리한다는 일관성을 보장할 수 없기 때문에 결국 시간 동기화 기능까지 정상적인 동작을 보장하지 못한다. 그러므로 RecvThread는 수신패킷을 NS-3로 직접 전달하지 않고 NS-3의 이벤트 리스트

에 패킷 수신 처리를 이벤트로 등록하고 실행을 종료한다. 등록된 이벤트는 나중에 NS3Adaptor가 깨어났을 때 처리될 것이다.

Ⅳ. 실험 결과

표 1은 본 논문의 실험을 위한 하드웨어 및 소프트웨어 환경을 나타낸다. 4.1절에서는, 제안하는 통합 시뮬레이터의 이벤트 처리 지연 수준을 평가한다. 4.2절에서는, 시뮬레이터 사이의 통신 부분이 전체 시뮬레이션 시간에서 점유하는 비율을 평가한다.

표 1. 실험 환경
Table. 1 Experimental environment

CPU	Intel Core2 Duo 2.4Ghz
Memory	2048MB
Operating System	Fedora Core 7
SystemC	v2.2
NS-3	v3.3

4.1 실험 1 : 통합 시뮬레이터의 이벤트 처리 지연

제안하는 통합 시뮬레이터의 핵심은 SystemC의 스케줄러를 이용하여 SystemC의 이벤트뿐 아니라 NS-3의 이벤트까지 모두 처리하는 것이다. 이러한 설계의 실용성을 확인하려면, NS-3 이벤트를 NS-3 시뮬레이터에서 수행한 경우와 통합 시뮬레이터에서 수행한 경우를 서로 비교하였을 때, 수행 시간 차이가 아주 미미하여 전체 시뮬레이션 시간에 영향을 미치지 않음을 보여야 한다.

그림 12는 이러한 부분의 성능을 평가하기 위한 절차를 나타낸다. 첫 번째 단계에서는 NS-3 예제를 준비한다. 이 예제에서는 [7]에서 제시한 NS-3의 기본 예제를 활용한다. 노드는 N개로 구성한다. 노드 구조는 Ethernet과 PPP(Point-to-Point Protocol) 가 혼합되었고 UDP 프로토콜을 활용하며 A 노드에서 D노드로 300 Byte의 패킷을 X초 간격으로 송신하는 간단한 응용 계층을 구성한다. 두 번째 단계에서는 time 명령어를 활용하여 각 시뮬레이터를 실행한다. time 명령어는 해당 프로세스의 총 수행 시간을 측정하는 기능을 제공한다. 결론적으로, SystemC의 예제는 배제한 상태에서 NS-3 예제만으로 구성하여 SystemC가 NS-3의 이벤트를 처리하는 성능을 평가하는 것이다.

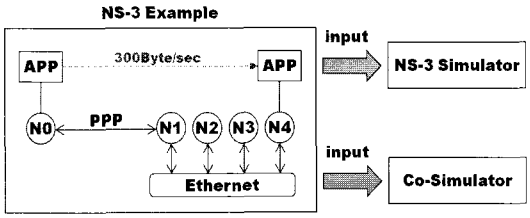

그림 12. 실험 1의 진행 절차
Fig. 12 The procedures of experiment 1

표 2는 그림 12의 절차에서 시뮬레이션 시간 파라미터에 변화를 주어 도출된 각 시뮬레이터의 수행 시간을 나타낸다.

표 2. 시뮬레이션 시간에 대한 처리 시간(단위 : 초)
Table 2. The processing time for the simulation time

	Simulation Time(Minute)		
	10	50	100
NS-3 Simulator	15.53	78.7	167.8
Co-Simulator	17.16	83.24	176.6

이 표에서 나타낸 바와 같이 SystemC 스케줄러를 이용하여 NS-3 이벤트를 대리 수행하는 본 논문의 통합 시뮬레이터는 NS-3 시뮬레이터를 이용한 경우와 수행 시간에서 거의 차이를 보이지 않고 있으므로 이러한 구조가 채택 가능함을 알 수 있다.

4.2 실험 2 : 시뮬레이터간 통신의 지연

기존 논문[1][13]에서는 이기종 시뮬레이터간 통신을 위한 처리(이하 ISC : Inter-Simulator Communication) 시간이 예제 구성의 특징에 따라 전체 시뮬레이터 처리 시간의 30 ~ 60%를 점유함을 보였고, 특히 ISC 횟수가 많아질수록 점유율은 급격히 상승한다는 결과를 도출하였다. 그러므로 시뮬레이터 처리 시간에서 ISC 부분의 수행 점유율이 시뮬레이터의 성능에 큰 영향을 준다.

본 절에서는, NS-3와 SystemC의 혼합 예제를 구성하고 프로파일 도구를 이용하여 본 논문의 통합 시뮬레이터를 실행한다. 실행 결과에서는 전체 시뮬레이터 처리 시간에서 ISC 부분의 실행 점유율을 확인할 수 있다.

그림 13은 노드 N0의 구성을 나타낸다. 상위 계층은 NS-3로 구현되었고 하위 계층은 SystemC로 구현되었으므로 ISC는 그 중간에 위치한다.

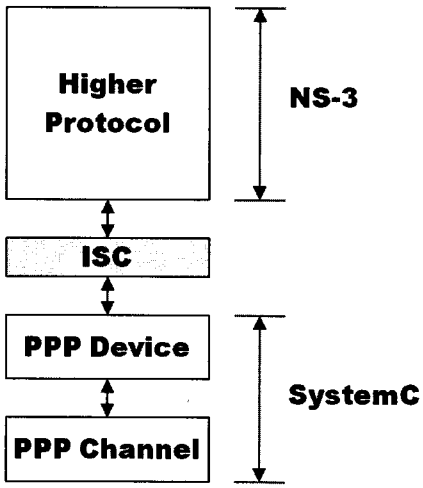

그림 13. N0의 노드 구성
Fig. 13 Configuration of the node N0

기본적인 예제는 4.1절의 실험에서 구성한 토폴로지를 기반으로 하고, PPP 네트워크 디바이스 부분만 SystemC로 구현한 형태로 바꾼다. 이 예제에서 응용 계층이 하위 계층으로 패킷을 송신하는 주기를 짧게 하면 ISC의 횟수는 증가한다. 그러므로 응용 계층의 패킷 전송 주기를 조절하여 이에 따른 ISC의 점유율을 확인할 수 있다.

따라서 본 절의 실험에서는 시뮬레이션 시간을 50초로 고정하고 매 시뮬레이터를 실행시킬 때마다 패킷 전송 간격을 점진적으로 짧게 하여 ISC의 횟수를 증가되게 함으로써 시뮬레이터간 통신에 부하를 주는 스트레스 시나리오를 구성한다.

표 3은 이에 대한 성능 평가 결과를 나타낸다. 시뮬레이터의 동작 시간 비율은 Co-Simulator, ISC로 분류하였다. 패킷 전송 간격의 변화에 따른 각 부문별 수행 비율을 확인할 수 있다. 표 3의 결과에서는, 본 논문의 통합 시뮬레이터가 패킷 전송 간격에 상관없이 ISC의 점유율이 약 10% 수준이므로 시뮬레이터간 통신을 위한 처리 부하가 매우 낮음을 알 수 있다.

표 3. 패킷 전송률 증가에 대한 각 부문별 프로파일 결과
Table 3. Profile results in each packet transmission rate

	패킷 전송 간격(microsecond)			
	1000	500	250	125
Co-Simulator	91.2%	90.8%	90.3%	90.4%
ISC	8.8%	9.2%	9.7%	9.6%

V. 결론 및 향후 연구

본 논문에서는 네트워크 디바이스의 프로토타입 개발 환경을 제공하기 위한 이기종 시뮬레이터의 효율적인 통합 방안을 제시하였다. 본 논문에서 제안한 통합 시뮬레이터의 설계는 이벤트 스케줄러와 데이터 송수신 부분으로 나뉜다.

이벤트 스케줄러 부분에서는, SystemC 모듈로 구현한 NS3Adaptor를 제안하였다. 이 모듈은 SystemC 스케줄러가 NS-3 이벤트를 처리할 수 있기 때문에 단일 프로세스로 NS-3 시뮬레이터와 SystemC 시뮬레이터를 실행할 수 있는 구조이고, 시뮬레이터간 통신을 위해 기반 운영체제의 IPC를 사용할 필요가 없으므로, 기존 논문에서 문제시 되었던 IPC 사용으로 인한 시뮬레이터 성능 저하를 막는다. 또한, 각 시뮬레이터의 커널 접근을 최소화 하므로 통합 시뮬레이터를 구현하기 용이하다.

데이터 교환 부분에서는, 상위 계층으로 구현된 NS-3와 하위 계층으로 구현된 SystemC를 연결하는 모듈을 제안하였다. 이 모듈은 일관성 있는 SystemC 인터페이스를 제공하므로 SystemC로 구현된 네트워크 디바이스 계층에게 NS-3의 세부 구현으로부터 독립적으로 구현할 수 있는 환경을 제공하므로 설계가 용이하다.

향후 연구에서는, 네트워크 디바이스의 프로토타입 개발 환경을 위해 본 논문에서 제안한 두 개의 시뮬레이터에 대한 통합뿐 아니라 다수의 시뮬레이터를 통합할 수 있는 일반화된 통합 설계 연구가 필요하다. 특히, SystemC 환경에서는 RTOS 기반의 소프트웨어 실행 환경을 지원하기 위한 다양한 연구가 진행되고 있으며 이러한 연구 결과를 본 논문의 통합 시뮬레이터에 포함할 수 있는 일반화된 통합 설계 방안을 도출한다면, 네트워크 디바이스의 프로토타입 환경을 더욱 폭넓게 지원할 수 있을 것으로 예상한다.

감사의 글

본 논문은 2단계 BK21 사업의 지원을 받아 수행된 연구임

참고문헌

[1] F. Fummi et al, "A timing-accurate modeling and simulation environment for networked embedded systems", In Proc. ACM Designa and Automation Conf. (DAC), page 42-47, Jun. 2003.

[2] OPNET, http://www.opnet.com

[3] QualNet, http://www.scalable-networks.com/.

[4] OMNeT++, http://www.omnetpp.org/.

[5] NS-2, http://www.isi.edu/nsnam/ns/.

[6] NS-3, http://www.nsnam.org/.

[7] NS-3, "Experimentation with NS-3", Online document, Available at http://www.nsnam.org/tutorials/trilogy-summer-school.pdf/.

[8] L. Cai and D. Gajski, "Transaction level modeling: an overview," in Proc. International Conference on Hardware/Software Codesign and System Synthesis, pp. 19-24, October 2003.

[9] OSCI SystemC TLM 2.0, Draft 2 for Public Review, Open SystemC Initiative, 2006, http://www.systemc.org/.

[10] N. Bombieri, F. Fummi, D. Quaglia, "TLM/network design space exploration for networked embedded systems", CODES+ISSS, October 2006.

[11] A. Silberschatz, "Operating System Concepts ", Wiley, 7th Edition.

[12] D. Bovet, M. Cesati, "Understanding the Linux Kernel", O'REILLY, 3rd Edition.

[13] S. Yoo and K. Choi, "Synchronization Overhead Reduction in Timed Cosimulation", Proc. IEEE International High Level Design Validation and Test Workshop, pp. 157-164, Nov. 1997.

[14] IEEE, "IEEE Standard SystemC Language Reference Manual", IEEE-1666, March 2006.

[15] David C. Black, Jack Donovan, "SystemC: From the Ground Up", Springer, October 2005.

저자소개

이호응(Hoeung Leeng)

2004년 한밭대학교 정보통신공학과 학사
2006년 한밭대학교 정보통신전문대학원 정보통신공학과 석사
2006 ~ 현재 한밭대학교 정보통신전문대학원 전파공학과(박사과정)

※관심분야 : 무선 MAC, 임베디드 소프트웨어, 임베디드 리눅스, 네트워크 시뮬레이터

박수진(Sujin Park)

2008년 한밭대학교 정보통신공학과 학사
2010년 한밭대학교 정보통신전문대학원 전파공학과 석사

※관심분야 : 무선 통신 소프트웨어, 네트워크 시뮬레이터

곽동은(Dongeun Gwak)

2010년 한밭대학교
정보통신공학과 (학사)
2010년~현재 한밭대학교
정보통신전문대학원
전파공학과 석사 과정

※관심분야 : 임베디드 리눅스, 무선 통신 프로토콜

박현주(Hyunju Park)

1990년 서울 시립대학교
전산통계학과 학사
1992년 서울대학교 대학원
전산과학과 석사

1997년 서울대학교 대학원 전산과학과 박사
1997년~현재 한밭대학교 전파공학과 교수

※관심분야 : 데이터베이스, 무선 통신 소프트웨어