FSM을 이용한 표준화된 버스와 IP간의 인터페이스 회로 자동생성에 관한 연구

준회원 이 서 훈*, 문 종 욱**, 황 선 영***

A Study on Automatic Generation of Interface Circuits Based on FSM between Standard Buses and IPs.

Ser-Hoon Lee* *Associate Member*, Jong-Uk Moon**, Sun-Young Hwang***

요 약

SoC 설계 복잡도의 증가로 인한 설계 비용 감소 및 짧은 time-to-market의 만족을 위해 IP에 기반한 설계 방식이 사용되고 있다. 기존에 설계 검증된 IP를 사용할 경우 시스템 버스와의 통신을 가능하게 하는 인터페이스 회로를 설계해 주어야 하며, 설계 비용을 감소시키기 위해서는 인터페이스 회로의 자동생성이 요구된다. 본 논문에서는 IP프로토콜을 기술하는 방법과 이 기술을 통하여 IP의 프로토콜 제어를 위한 FSM(Finite State Machine)을 생성하여 버스와의 인터페이스 회로를 자동생성하는 방법을 제안한다. 제안한 시스템에서는 프로토콜 분석의 어려움을 줄이기 위해 표준화된 버스의 FSM을 라이브러리화 하였다. 제안된 방법으로 AMBA AHB에 사용되는 슬레이브 형태 IP의 인터페이스 회로를 자동생성한 결과 매뉴얼로 설계한 인터페이스 회로에 비해 면적은 4.5%의 증가를 보였다. 100 Mhz의 버스 동작 속도와 34 Mhz의 슬레이브 모듈의 동작 속도 환경에서 16개의 32 비트 데이터를 버스트 모드로 전송시 latency는 평균 7.1%의 증가를 보였으나, 시스템 버스의 점유는 평균 64.9% 정도로 감소하였다. 본 논문에서 제안한 시스템을 사용하여 시스템 버스의 효율을 증가한 인터페이스 회로를 생성해 낼 수 있다.

Key Words : Interface, Standard Bus, SoC, FSM, Bus Occupation

ABSTRACT

IP-based design methodology has been popularly employed for SoC design to reduce design complexity and to cope with time-to-market pressure. Interface modules for communication between system buses and IPs are required, since many IPs employ different protocols. Automatic generation of these interface modules would enhance designer's productivity and IP's reusability. This paper proposes an automatic interface generation system based on FSM generated from the protocol description of IPs. The proposed system provides the library modules for the standard buses to reduce the burdens of describing the protocols for data transfer from/to standard buses. Experimental results show that the area of the interface circuits generated by the proposed system had been increased slightly by 4.5% on the average when compared to manual designs. In the experiment, where bus clock is 100 Mhz and slave module clock is 34 Mhz, the latency of the interface had been increased by 7.1% in burst mode to transfer 16 data words. However, occupation of system bus can be reduce by 64.9%. A chip designer can generate an interface that improves the efficiency of system bus, by using this system.

* 서강대학교 전자공학과 CAD & ES 연구실 (hwang@ccs.sogang.ac.kr)
논문번호 : KICS2004-08-142, 접수일자 : 2004년 8월 9일
※본 연구는 2004년도 서강대학교 교내 연구비 지원에 의해 이루어졌음.

Ⅰ. 서 론

SoC 설계 복잡도의 증가로 인하여 기존의 IP를 재사용하는 IP 기반 설계 방식이 채택되고 있다[1][2]. 이미 설계 검증된 IP는 특정한 모듈과의 데이터 통신을 위한 특정한 프로토콜을 갖고 있어 다른 모듈과의 통신을 위해서는 이들 간에 통신이 가능한 프로토콜로 맞추어주는 인터페이스 회로를 필요로 한다[3][4][5]. 이를 위한 인터페이스 회로의 설계는 대부분이 매뉴얼로 설계되고 있는 실정이다[6]. 설계자가 프로토콜을 분석하고 IP의 동작 속도와 특성에 따라 제약 조건을 만족하는 인터페이스 회로를 설계할 수 있으나, 시스템을 구성하는 다양한 IP 모듈 간 인터페이스 회로의 매뉴얼 설계는 긴 설계 시간과 비용을 요구한다. 이의 극복을 위하여 인터페이스 회로를 자동으로 생성할 수 있는 자동 생성 시스템이 필요하다[7][8]. 서로 다른 프로토콜을 갖는 모듈간의 인터페이스 회로의 매뉴얼 설계 과정은 각 모듈의 동작 프로토콜을 기술한 데이터 쉬트를 기반으로 통신 프로토콜을 분석하여 신호의 인과 관계를 찾는다. 그리고 한 모듈에서 발생된 신호에 대하여 통신코자하는 모듈의 프로토콜에 맞는 신호로 변환하여 해당되는 포트에 인가하는 인터페이스 회로를 구성한다[5]. 인터페이스 회로의 자동생성을 위해서 주어진 데이터 쉬트에 해당하는 프로토콜의 표현이 필요하고, 두 모듈이 요구하는 프로토콜의 분석과 신호의 인과관계를 찾는 알고리듬이 필요하다. 임의의 프로토콜을 갖는 IP 모듈간의 인터페이스 회로 생성은 IP에 따라 포트 이름이 상이하고 같은 이름을 갖는 포트간에도 요구되는 신호의 특성이 다를 수 있어 주어진 프로토콜 정보만으로 해당 포트에서 나오는 신호의 특성을 분석하고 인과관계까지 자동으로 찾아내는 알고리듬의 구현은 매우 어렵다.

사용하는 IP 모듈에 대한 프로토콜의 특성 분석의 어려움을 극복하고 포트 이름과 기능이 다른 포트간의 인터페이스 회로에 대한 정확한 동작을 위해 IP 프로토콜과 듀얼 구조를 가지는 FSM을 생성하였다[6][12]. 생성한 FSM을 표준 버스의 FSM과 연결하여 표준 버스와 IP 간의 데이터 교환을 지원하는 인터페이스 회로를 자동 생성하는 시스템을 구축하였다. 대부분의 SoC 플랫폼이 표준화된 버스를 채택하고 있다[9]. 표준화된 버스에 대하여 대응되는 신호를 발생하는 동작과 구조를 라이브러리에 미리 구축하여 공통된 버스에 대한 IP의 인터페이스 회로 생성 시 공통 버스의 프로토콜을 기술해야 하는 부담감을 줄였다.

2절에서 인터페이스 회로와 자동생성기의 구조를 설명하고, 3절에서는 IP의 프로토콜 기술 방식, 표현된 프로토콜을 기반으로 IP에 대한 FSM 구성과 이를 이용한 전체적인 인터페이스 회로를 생성하는 방법에 대해서 제시한다. 4절에서는 사용자 편의를 위한 GUI(Graphic User Interface) 환경의 소개와 자동 생성된 인터페이스 회로와 매뉴얼로 작성된 인터페이스 회로의 성능을 비교하며, 마지막으로 5절에서는 결과 및 추후과제를 제시한다.

Ⅱ. 시스템 개관

그림 1은 전형적인 AMBA 시스템의 구조를 보인다. AMBA는 ARM 코어에 기반을 둔 SoC 설계를 위한 on-chip 통신의 표준안이 되고 있다[10]. AMBA와 같이 표준화된 버스에 IP를 사용하기 위해서는 그림 1에서 보는 것과 같이 버스와의 프로토콜을 맞추기 위해 인터페이스 회로의 설계가 요구된다. 시스템의 설계가 집적도 대비 생산성을 time-to- market에 만족하게 하기 위해 IP 사용이 증가하여 이에 따른 인터페이스 회로의 설계 비용도 증가하게 된다[11]. 인터페이스 회로의 설계 비용 감소를 위해 인터페이스 회로 자동생성기가 요구된다. 본 연구에서 인터페이스 회로를 자동 생성하는 시스템을 구축하였다.

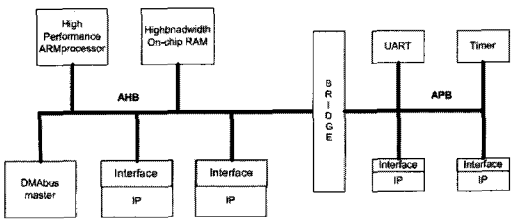


그림 1. 전형적인 AMBA 시스템 구조

그림 2에 제안한 인터페이스 회로 자동생성기의 흐름을 보인다. GUI(Graphic User Interface)를 통해 인터페이스 회로 설계 스펙을 입력하고 IP 프로토콜 입력기를 사용하여 IP의 프로토콜을 입력하면 인터페이스 회로 생성기에 입력으로 들어갈 문자열을 생성해 준다. IP의 스펙 및 프로토콜에 대한 기술은 문자열로 인터페이스 회로 생성기의 입력으로 들어가게 되어 IP의 프로토콜과 듀얼 구조인 FSM을 생성하게 되고, 라이브러리에 저장된 표준화된 버스 프로토콜에 듀얼 구조인 FSM과 버퍼를 사용하여 인터

페이스 회로를 생성해 낸다. 결과 인터페이스 모듈은 IP의 듀얼 FSM, 버스의 듀얼 FSM 그리고 버퍼를 연결한 구조로 RTL 수준의 HDL 코드를 출력하게 된다.

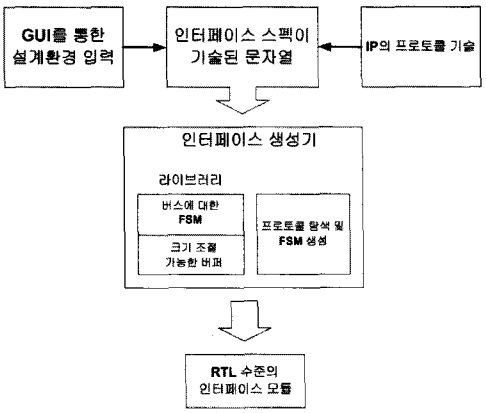


그림 2. 인터페이스 생성기 흐름도

그림 3은 슬레이브 형태의 IP 모듈에 대한 생성된 인터페이스 회로를 보인다. 일반적으로 슬레이브 모듈은 버스의 동작 속도와 같거나 낮은 속도로 동작하는 모듈이 사용된다. 인터페이스 회로 내부의 FSM 간 제어 신호는 특정 상태를 천이 시키는 역할을 한다. 예를 들어, 버스트 모드로 데이터를 전송할 때 데이터들을 순차적으로 버퍼에 입력시키고, 데이터 write 상태에서 벗어났을 때 이를 IP의 듀얼 FSM에 전달하여, IP와의 통신을 시작하는데 필요한 제어신호는 IP의 듀얼 FSM을 초기상태에서 동작 시작상태로 천이시킨다.

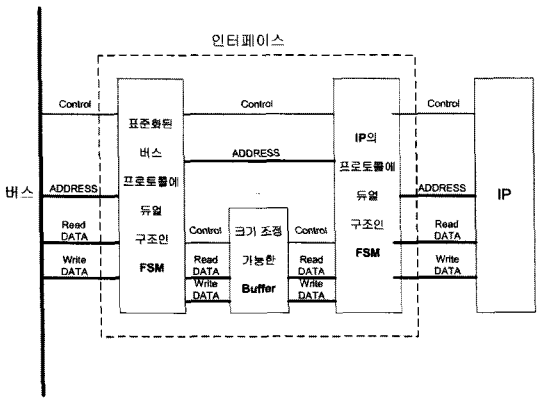


그림 3. 생성된 인터페이스 회로 형태.

블록 중 표준화된 버스 프로토콜에 듀얼 구조인 FSM과 버퍼는 라이브러리에 구현되어 있으며, 자동생성 프로그램은 IP 설계자가 제공한 프로토콜 기술을 입력으로 받아 IP의 듀얼 FSM을 구성하고 IP와의 통신에 필요한 신호들을 파싱하여 최종적인 블록들 간의 연결을 한다. 대부분의 경우에 버스의 동작 속도와 IP의 동작속도가 다르기 때문에 하나의 클럭으로 동작하는 FSM으로 인터페이스 회로를 구성할 경우, 인터페이스 회로의 FSM과 IP 구동 클럭이 동기화 되지 않기 때문에 클럭 에지가 미스매치 되는 경우 인터페이스 회로의 출력 신호를 IP에서 인식할 수 없는 문제점이 발생한다. 이 문제점을 해결하기 위해서는 인터페이스 회로에서 출력된 신호가 느린 클럭으로 동작하는 IP에서의 수신을 확인한 후에 다음 상태로 천이되는 루프를 모든 상태에 대해서 고려해 주어야 한다. 이러한 구현 방식은 자동생성 과정이 복잡하게 되며, 느린 IP를 사용할 시에 IP의 동작이 모두 끝날 때까지 버스의 사용권을 가지고 있어야 하므로 버스에 대한 사용 효율이 저하된다. 버스의 듀얼 FSM과 IP의 듀얼 FSM을 따로 구성하는 제안 된 인터페이스 회로의 구조는 버스와 통신하는 듀얼 FSM이 데이터를 버퍼에 모두 전송한 후, IP의 듀얼 FSM에 데이터를 IP에 송신해도 된다는 신호를 발생한다. 이를 IP의 듀얼 FSM에서 수신하였는지 확인하는 상태만을 추가함으로써 다른 클럭 스피드로 동작하는 FSM으로 구성하여, 버스와 IP 간의 동작 속도 차이가 상당한 경우에 대해서도 완벽하게 동작한다.

버퍼의 사용은 데이터의 버스트 모드 전송일 때 효율적인 동작을 하기 위한 것이다. IP에 싱글 전송 모드로 전송할 경우에는 단일 데이터를 전송하고 올바른 수신에 대한 응답 신호를 받았을 경우에만 다음 상태로 전이되기 때문에 버퍼가 필요 없다. 버스와 속도가 다른 IP 간에 버스트 모드로 동작할 경우에는 연속된 데이터를 전송하기 때문에 타겟 모듈에서 데이터의 수신을 확인할 수 없게 된다. 연속적인 데이터를 정확하게 전송하기 위해서는 버퍼에 데이터의 wirte 속도와 read 속도가 버퍼를 동작시키는 모듈의 클럭에 동기되어 동작하도록 하였고 데이터 전송 latency를 감소시키기 위해 듀얼 포트의 기능을 갖도록 설계하였다.

Ⅲ. FSM을 통한 인터페이스 회로 자동생성

본절에서는 IP 프로토콜 기술로부터 인터페이스 회로를 자동으로 생성하는 과정을 설명한다. 최초 IP의 프로토콜을 입력받는 방법과 이를 사용하여 듀얼

FSM을 만들어 내며, 생성된 듀얼 FSM과 라이브러리에 저장되어 있는 버스의 듀얼 FSM, 버퍼를 사용하여 최종적인 인터페이스 회로를 생성하는 과정을 설명한다.

3.1 IP 프로토콜 기술

IP 프로토콜 기술시 일반 문자열 형식으로 기술하는 것은 에러를 발생하기 쉬운 문제점이 있으므로 GUI를 지원하는 IP 프로토콜 입력기를 통해 입력하게 된다. 입력된 프로토콜은 설계자가 선택한 인터페이스 회로 설계 스펙 정보와 함께 인터페이스 회로 자동생성기의 입력으로 들어갈 중간형태의 문자열을 생성한다. IP 프로토콜 입력 시 상태명, 다음 상태, 다음 상태로의 천이 신호, 그리고 현재 상태의 출력 값을 입력하여 Moore 머신을 기술하는 것과 유사하다. 그림 4는 리드 솔로몬 인코더 모듈(CS3110)의 타이밍 다이어그램을 보여주며 이를 프로토콜 입력기에 입력하기 위해서 상태를 정의하고 상태에 따른 입출력 신호를 기술한 것을 보인다.

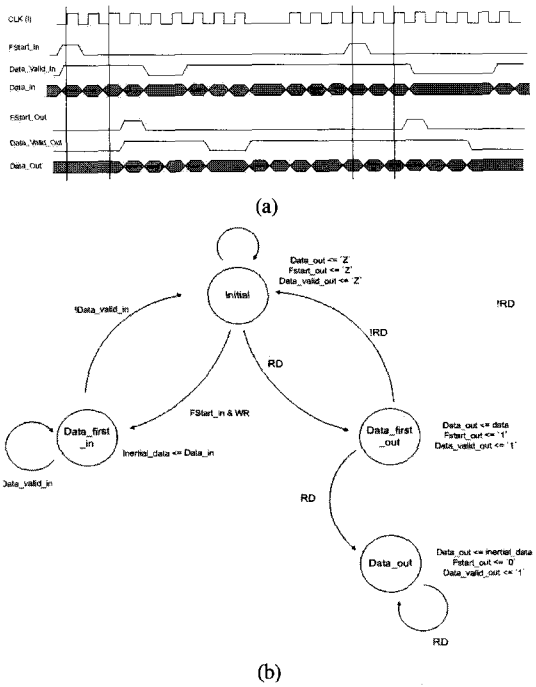


그림 4. 프로토콜 기술의 예. (a) 리드 솔로몬 인코더 모듈(CS3110)의 타이밍 다이어그램, (b) 스테이트 머신 형태로의 표현.

설계자는 IP 프로토콜을 입출력 값의 변화를 중심으로 상태들을 정의하여 상태를 중심으로 입출력 값 및 상태 천이 신호들을 정의하여 입력한다.

3.2 IP에 대한 듀얼 FSM 생성

IP의 설계자가 구축한 GUI를 통해 프로토콜을 기술하여 제공하면, 인터페이스 회로 자동생성 프로그램은 IP의 프로토콜에 대응하는 신호를 발생하는 듀얼 FSM을 생성한다. 듀얼 FSM은 IP의 초기 상태에서 동작 상태로 천이하게 하는 신호를 발생하며, IP의 출력 신호를 입력 신호로 받아 상태를 천이한 후 다음 상태에서 IP의 상태 천이 신호를 발생하는 구조를 갖는다. IP의 듀얼 FSM 상태들은 입력된 IP의 상태들과 동일하게 구성되며 IP의 입력신호가 듀얼 FSM의 출력신호로, IP의 출력신호가 듀얼 FSM의 입력신호로 된다[6][12]. 그림 5는 IP의 프로토콜과 서로 대응하는 신호를 주고 받는 듀얼 FSM을 보인다.

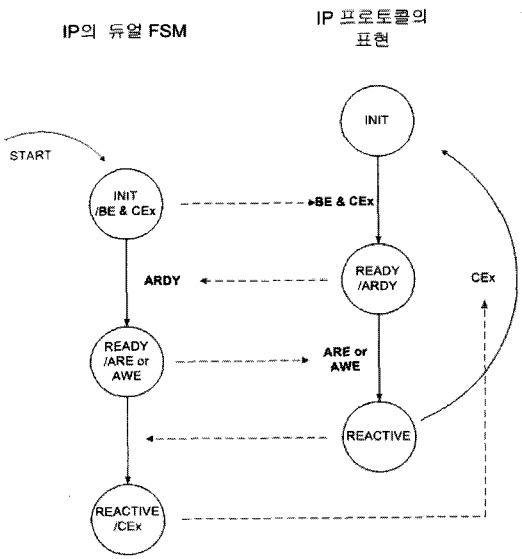


그림 5. IP의 듀얼 FSM.

IP의 출력신호를 듀얼 FSM에서 입력신호로 받아 상태를 천이시키고, 상태가 천이된 후에 듀얼 FSM에서 IP의 상태를 천이시킬 신호를 출력하게 된다. 이러한 형태를 가지는 듀얼 FSM은 IP와 정확히 동일한 프로토콜을 가지고 통신하게 되므로 정의되지 않은 상태로의 천이와 인터페이스 회로와 속도가 다른 클럭을 사용 시 다음 상태로 천이를 못하는 등의 에러를 발생하지 않는다. IP와 오동작 없이 통신 과정을 수행하는 듀얼 FSM을 버스의 듀얼 FSM이 구동시키게 되면, IP는 IP의 듀얼 FSM과 통신하여 버퍼에 저장되어 있는 데이터를 받아들이게 된다.

IP의 프로토콜을 분석할 때 현재 상태에 따라 적절한 상태 천이 신호를 출력해 주어야 하는데, 적절한 상태 천이 신호를 찾아내기는 IP 마다 프로토콜

특성이 다르기 때문에 매우 어렵다. IP의 듀얼 FSM을 생성하는 것은 신호의 특성을 파악해야 하는 과정 없이, IP의 상태에 따라 필요한 천이 신호를 입력 받아 오동작 없이 통신 과정을 수행할 수 있는 컨트롤러가 생성되는 것이다.

3.3 Library

라이브러리는 표준화된 버스에 대한 듀얼 FSM과 크기를 조절할 수 있는 버퍼를 저장한다. 인터페이스 회로 생성 시 IP의 프로토콜을 분석하고 IP의 듀얼 FSM을 생성하는 과정은 전체 프로그램 과정 중 가장 복잡한 과정이다. 버스에 대한 프로토콜을 입력으로 받아 위의 과정을 반복한 후 전체 인터페이스 회로를 생성할 수도 있다. 설계 경향이 표준화된 버스를 많이 사용하는 경우에 프로그램의 복잡도를 줄이기 위해 버스에 대한 듀얼 FSM은 미리 제작하여 라이브러리화 하였다. 설계자는 시스템의 버스에 대한 듀얼 FSM을 제작해 라이브러리화 한다면 설계하고자 하는 IP에 대해서 시스템 버스와의 인터페이스 회로를 자동으로 생성할 수 있다.

2절 시스템 개관에서 보았듯이 버퍼는 각 IP의 인터페이스 회로에 설계되어 있으며, IP의 버스트 동작 시에 데이터를 일시 저장한다. IP의 버스트 모드에 따라 전송 데이터가 일정 개수로 한정되어 있을 경우 버퍼는 IP에서 사용할 용량만큼만 필요하게 되므로 사이즈를 조절할 수 있는 버퍼가 요구된다. 버스와 모듈의 동작 속도가 다른 경우 버퍼의 데이터 입출력 시 다른 클럭 속도를 가지고 동작하는 기능이 요구되므로 이런 기능을 지원하는 버퍼를 설계하여 라이브러리화 하였다.

3.4 자동생성된 인테페이스 회로 구조

인터페이스 회로 자동생성기는 라이브러리에 있는 버스의 듀얼 FSM, 버퍼 그리고 자동생성된 IP의 듀얼 FSM을 사용하여 인터페이스 회로를 구성하게 된다. 버스의 듀얼 FSM은 마스터 모듈에서 송신하는 데이터를 받아 송신이 완료될 때까지 데이터를 버퍼에 저장한 후 IP의 듀얼 FSM에게 동작을 시작하게 하는 신호를 전송하게 된다. 동작 시작을 알리는 신호를 받은 IP의 듀얼 FSM은 버스의 듀얼 FSM에 신호를 받았다는 ACK 신호를 보내고 버퍼에 저장된 데이터를 이용하여 IP와 통신을 시작하게 되며, ACK을 받은 버스의 듀얼 FSM은 마스터 모듈에 전송이 완료되었다는 신호를 송신하게 된다. 이때, 마스터 모듈은 버스 락을 풀어 다른 마스터가 버스를 사용할 수 있게 하기 때문에 전체 시스템 버스의 사용 효율을 향상시킨다. 그림 6은 슬레이브 모듈에 데이터를 쓰는 동작일 때의 버스의 듀얼 FSM과 IP의 듀얼 FSM 사이의 신호 연결을 보인다. 이때 사용한 버스 프로토콜은 AMBA AHB를 사용하였으며 IP는 리드 솔로몬 인코더 모듈(CS3110)을 사용하였다.

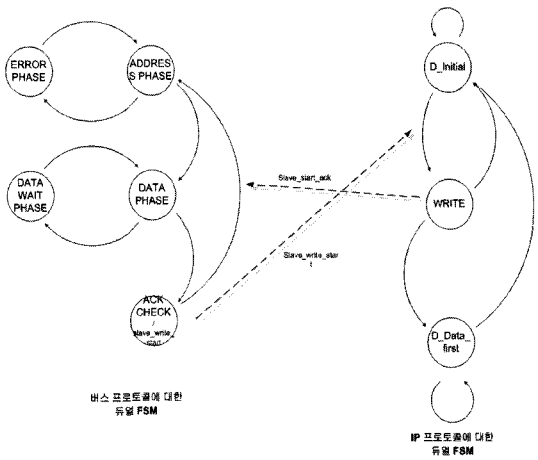


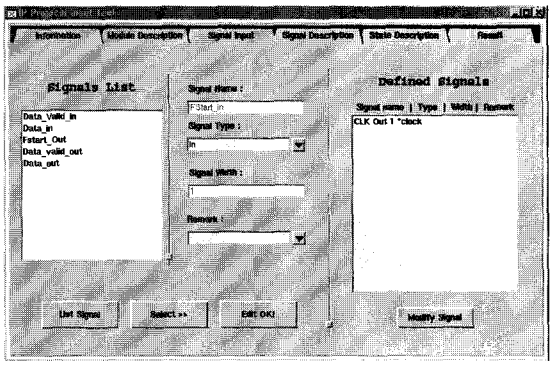
그림 6. 버스 듀얼 FSM과 IP 듀얼 FSM의 연결.

버스의 듀얼 FSM에서 추가된 'Ack Check' 상태는 IP의 듀얼 FSM에 데이터 전송이 끝났다는 신호인 'slave_write_start' 신호를 주게 되고 IP의 듀얼 FSM은 이 신호를 받았다는 Ack 신호를 보낸다. 버스의 듀얼 FSM은 Ack 신호를 받을 때까지 'slave_write_start' 신호를 계속해서 출력하고 Ack 신호를 받은 후 버스의 듀얼 FSM은 초기화 되고 버스의 사용을 끝내게 된다. 'slave_write_start' 신호를 받은 IP의 듀얼 FSM은 초기상태에서 write 상태로 천이하여 슬레이브 모듈에 데이터를 전달하는 동작을 시작하게 된다.

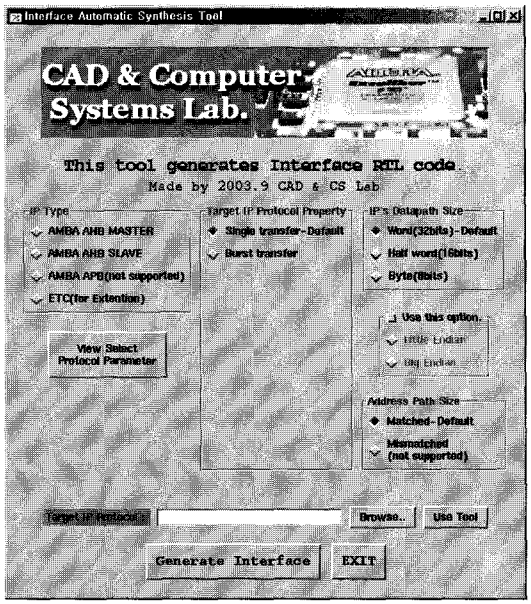
IP의 듀얼 FSM과 IP간 통신이 끝나지 않은 경우, 버스의 듀얼 FSM이 버퍼를 다시 억세스할 때는 'Error Phase'로 천이되어 버퍼에 있는 데이터를 IP가 모두 읽어들여 버퍼 empty 신호를 발생할 때까지 기다렸다가 전송하게 된다. 버퍼의 쓰기와 읽기 동작이 동시에 발생하지 않게 배타적으로 수행할 수 있도록 버스의 듀얼 FSM의 write/read 신호와 IP의 듀얼 FSM의 write/read 신호간의 로직을 구현하였다.

Ⅳ. GUI 환경과 실험결과

본 시스템은 사용자와의 편의를 위해 Unix Tcl/Tk를 이용하여 GUI 환경을 구축하였다. 2, 3절에서 언급한 GUI는 그림 7과 같이 프로토콜 입력기와 설계 스펙을 입력한 후 인터페이스 회로를 생성하는 자동생성기로 구현되었다.



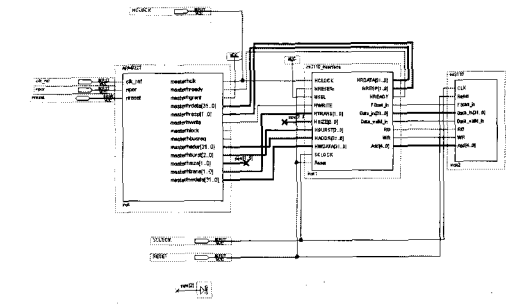
(a)



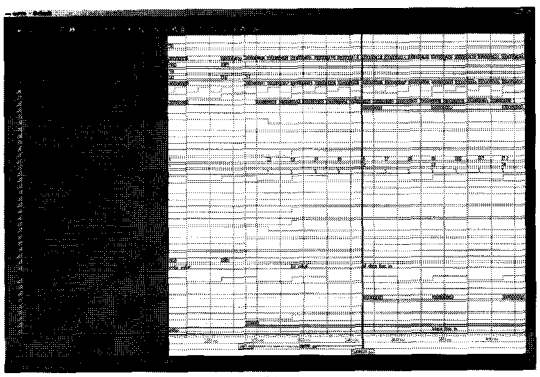
(b)

그림 7. GUI 환경. (a) 프로토콜 입력기, (b) 인터페이스 회로 자동생성기.

실험을 위하여 AMBA AHB 버스 프로토콜을 라이브러리화 하였고, 프로토콜 입력기를 사용하여 CS3110 모듈의 프로토콜을 입력시켰으며, 인터페이스 회로 자동생성기를 구동하여 AMBA AHB와 CS3110 모듈간의 인터페이스 회로를 생성하였다. 생성된 결과의 정확한 시뮬레이션을 위해 그림 4-2과 같이 Quartus II 2.2 을 사용하여 ARM 코어와 연동하여 설계한 후, Quartus에서 제공하는 테스트 벤치 자동생성 툴을 사용하여 AMBA BFM(Bus Functional Model)에 따른 신호들을 생성하였다. 생성된 테스트 벤치를 사용하여 Modelsim XE II 5.6e에서 시뮬레이션 하였다.



(a)



(b)

그림 8. 자동생성된 인터페이스 회로 검증. (a) Quartus를 이용한 인터페이스 회로 검증 환경 설계, (b) Modelsim을 이용한 시뮬레이션 결과 파형.

시뮬레이션 결과에서 첫 번째 커서는 마스터 모듈에서 'masterhwdata' 포트를 통해 데이터를 발생시키는 것을 볼 수 있다. 이를 버퍼에 저장하여 두 번째 커서가 있는 시점부터 CS3110 모듈에 데이터를 전송하여 CS3110 모듈의 'data_in' 포트에 데이터가 입력되고 있는 것을 볼 수 있다.

구축한 시스템에서 생성한 인터페이스 회로 모듈과 Altera사의 Excalibur 설계 예제에서 지원하는 구조와 동일하게 설계한 인터페이스 회로 모듈 간의 성능을 비교하였다.

표 1은 이들 모듈 코드를 Hynix 0.35 um 공정 라이브러리를 사용한 Synopsys의 Design Analyzer로 합성하여 면적을 비교하였다. 여기서 면적 측정 단위는 2-input NAND 게이트를 1로 하였다.

표 1. 결과의 면적 비교

IP 명		매뉴얼 설계	자동 설계	비교
REED Solomon Encoder (CS3110)	싱글모드	6	9	50.0 %
	버스트모드	51	54	5.8 %
MPEG-3 Audio 디코더 (CS4620)	싱글모드	6	9	50.0 %
	버스트모드	51	57	11.7 %
Turbo Encoder (CS3530)	싱글모드	6	9	50.0 %
	버스트모드	53	55	3.7 %
평균	싱글모드	6	9	50.0 %
	버스트모드	51.7	55.4	7.1 %

이때 사용된 버퍼는 AMBA AHB의 버스트 모드 전송시의 최대 데이터 전송 사이즈인 32 비트 데이터 16개의 용량(64 byte)으로 설정하였다. 결과에서 볼 수 있듯이 매뉴얼로 작성된 인터페이스 회로와 자동 생성된 인터페이스 회로는 경미한 면적차이를 보인다. IP에 대한 듀얼 FSM이 데이터 패스에 대한 제어를 하지 않고 IP가 일반적으로 복잡하지 않은 프로토콜로 동작하게 설계되므로 합성된 IP의 듀얼 FSM의 면적은 크지 않게 된다.

데이터 전송 latency는 100 Mhz의 버스 동작 클럭을 기준으로 데이터 전송이 시작한 시점부터 IP에 데이터 전송이 종료되는 시점까지의 소비된 버스 클럭을 측정하였다. IP 모듈이 34 Mhz의 클럭으로 동작하여, 싱글 모드 전송시와 16개의 32 비트 데이터를 버스트 모드로 전송할 경우의 latency를 측정하여 표 2에 제시하였다.

표 2. 결과의 latency 비교

IP 명		매뉴얼 설계	자동설계		비고
Reed Solomon Encoder (CS3110)	Glue Logic	404	버스 듀얼 FSM	407	8.9%
			IP 듀얼 FSM	30	
			기타	3	
	버퍼	905	905		0.0%
	합계	1,309	1,345		2.7%
MPEG-3 Audio 디코더 (CS4620)	Glue Logic	369	버스 듀얼 FSM	407	20.3%
			IP 듀얼 FSM	34	
			기타	3	
	버퍼	905	905		0.0%
	합계	1,274	1,349		5.8%
Turbo Encoder (CS3530)	Glue Logic	377	버스 듀얼 FSM	407	17.5%
			IP 듀얼 FSM	33	
			기타	3	
	버퍼	905	905		0.0%
	합계	1,282	1,348		5.1%
평균	Glue Logic	383	버스 듀얼 FSM	407	15.4%
			IP 듀얼 FSM	32	
			기타	3	
	버퍼	905	905		0.0%
	합계	1,288	1,347		4.5%

매뉴얼 설계된 인터페이스 회로에서는 데이터를 버퍼에 저장함과 동시에 IP에서 데이터를 전송받을 수 있는 신호를 직접 보낼 수 있으나, 자동생성된 인터페이스 회로는 데이터가 버퍼에 저장되면 IP의 듀얼 FSM에 이를 알려주고 IP의 듀얼 FSM이 IP에게 데이터를 전송받을 수 있게 신호를 생성해주는 동작 시퀀스로 인해 자동설계된 인터페이스 회로의 데이터 전송 latency는 매뉴얼로 작성된 인터페이스 회로의 데이터 전송 latency보다 크다는 것을 볼 수 있다. 데이터의 버스트 모드 전송시 듀얼 포트 버퍼의 사용으로 데이터를 저장함과 동시에 출력할 수 있어 자동생성된 인터페이스 회로와 매뉴얼로 작성된 인터페이스 회로의 latency가 큰 차이가 없음을 알 수 있다. 그러나, 매뉴얼로 작성된 인터페이스 회로는 하나의 FSM으로 구성되어 있기 때문에 슬레이브 모듈에 데이터 전송이 완료될 때까지 시스템 버스를 점유하고 있어야 한다는 단점이 있다.

표 3은 마스터 모듈이 슬레이브 모듈로 데이터 전송시 버스를 점유하고 있는 시간을 측정하였다. 실험 환경은 이전과 같이 버스와 슬레이브 모듈의 동작 속도를 각각 100 Mhz, 34 Mhz로 설정하였다. 전송되는 데이터는 16개의 32 비트 데이터를 버스트 모드로 전송하여 마스터 모듈이 버스를 점유하고 있는 시간까지 소비하는 버스 클럭을 측정하였다.

표 3. 버스 점유 cycle 비교

IP 명		매뉴얼 설계	자동 설계	비교
REED Solomon Encoder (CS3110)	싱글모드	6	7	16.6 %
	버스트모드	51	18	- 64.7 %
MPEG-3 Audio 디코더 (CS4620)	싱글모드	4	7	75.0 %
	버스트모드	49	18	- 63.3 %
Turbo Encoder (CS3530)	싱글모드	4	7	75 %
	버스트모드	51	17	- 66.7 %
평균	싱글모드	4.7	7	48.9 %
	버스트모드	50.4	17.7	- 64.9 %

자동으로 생성된 인터페이스 회로는 버스에 대한 듀얼 FSM이 데이터를 모두 전송하고 초기 상태로 돌아오는데 소비되는 클럭 사이클을 측정하였으며, 매뉴얼로 작성한 인터페이스 회로는 인터페이스 회로 FSM이 데이터를 모두 전송하고 초기상태로 돌아왔을 때까지 소비되는 클럭 사이클을 측정하였다. 표 2에서 자동으로 생성된 인터페이스 회로가 매뉴얼로 작성된 인터페이스 회로에 비해 데이터 전체의 전송 latency는 높은 특성을 보이지만 표 3에서 마스터의 버스 점유 cycle을 비교하면 자동으로 생성된 인터페이스 회로가 매뉴얼로 작성된 인터페이스 회로보다 버스트 모드시 평균 64.9% 감소한 버스 점유 cycle을 가지는 것을 볼 수 있다. 반면 데이터의 싱글 모드 전송일 경우 평균 48.9%의 큰 증가를 보였다. 그러나 싱글 전송시 자동생성된 인터페이스 회로와 매뉴얼로 작성된 인터페이스 회로의 전송 latency가 10 클럭 미만의 작은 크기를 보이기 때문에 증가량이 큰 문제점으로 볼 수 없다. 표 4는 16개의 32 비트 데이터를 버스트로 전송시 전송 latency와 버스 점유 cycle을 비교하였다.

표 4. 전송 latecy와 버스 점유 cycle 비교

IP 명		매뉴얼 설계	자동 설계	비교
REED Solomon Encoder (CS3110)	전송 latency	51	54	5.8 %
	버스점유cycle	51	18	- 64.7 %
MPEG-3 Audio 디코더 (CS4620)	전송 latency	51	57	11.7 %
	버스점유cycle	49	18	- 63.3 %
Turbo Encoder (CS3530)	전송 latency	53	55	3.7 %
	버스점유cycle	51	17	- 66.7 %
평균	전송 latency	51.7	55.4	7.1 %
	버스점유cycle	50.4	17.7	- 64.9 %

버스트 모드 전송시 전송 latency는 평균 7.1%의 증가를 보였으나 버스 점유 cycle은 64.9%의 감소를 보였다. 버스 점유 cycle의 감소는 다른 마스터 모듈의 버스 사용 가능 cycle을 증가시켜 주므로 시스템 버스의 사용 효율을 증가시킨다. 그림 4-3은 타이밍 다이어그램 관점에서 버스 점유 cycle의 감소가 시스템 버스의 사용 효율을 증대시키는 것을 보여준다.

Bus_Lock 신호는 마스터 모듈이 시스템 버스를 사용하기 위해 버스 락을 하는 신호이며 HDATA, Buffer_data, Slave_datain은 각각 마스터, 버퍼 그리고 슬레이브의 데이터 포트이다. 매뉴얼로 설계된 인터페이스 회로는 버스 락이 된 후부터 슬레이브에 데이터가 모두 전송될 때까지 버스 락을 풀지 않게 되는 반면 자동생성기를 통해 생성된 인터페이스 회로는 버퍼에 데이터가 저장되는 시점에서 버스 락을 풀게 된다. 결과, 자동생성된 인터페이스 회로는 다른 모듈이 시스템 버스를 사용할 수 있는 cycle을 증가시켜 시스템 버스의 사용 효율을 증대시킨다. 이 특성은 슬레이브 모듈의 동작속도가 느릴수록 시스템 버스의 효율 면에서 성능이 높은 인터페이스 회로의 결과를 얻을 수 있다.

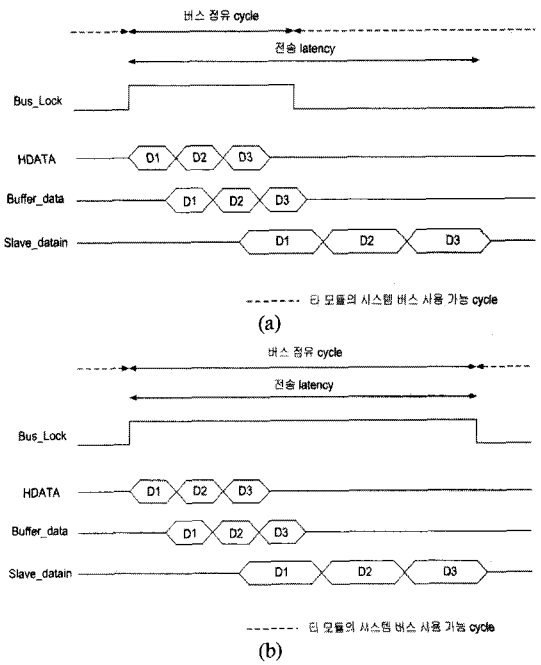


그림 9. 시스템 버스 점유 cycle. (a) 매뉴얼로 설계된 인터페이스 회로의 버스 점유 cycle, (b) 자동생성된 인터페이스 회로의 버스 점유 cycle.

표 5에서는 버스 동작 속도가 100Mhz인 환경에서 MPEG-3 오디오 디코더 모듈(CS3110)의 동작 속도에 따라 16개의 32 비트 데이터의 전송을 위한 버스 점유 cycle을 비교하였다.

표 5. 슬레이브 모듈의 동작 속도에 따른 버스 점유 cycle 비교

IP 동작 속도	매뉴얼 설계	자동설계	비교
34 Mhz	51	18	- 64.7 %
10 Mhz	166	18	- 89.1 %
5 Mhz	321	18	- 94.4 %

자동 설계된 인터페이스 회로는 내부 버퍼에 데이터가 저장되면 버스와의 통신이 종료되므로 버스 점유 cycle은 슬레이브 모듈의 동작속도와 관계없이 18 클럭으로 동일하였고 매뉴얼로 설계된 인터페이스 회로는 슬레이브 모듈의 동작이 완료될 때까지 버스와의 통신을 계속해야 하기 때문에 슬레이브 모듈의 동작속도가 느려짐에 따라 버스 점유 cycle이 매우 커지는 것을 볼 수 있다. 구현된 인터페이스 회로 자동생성기는 저속으로 동작하는 IP의 인터페이스 회로 생성시 높은 성능을 갖는 인터페이스 회로의 결과를 얻을 수 있다.

자동생성된 인터페이스 회로는 매뉴얼로 작성된 인터페이스 회로와 성능 비교한 결과 면적은 평균 4.5%의 증가를 보이고 100 Mhz로 동작하는 버스와 34 Mhz로 동작하는 슬레이브 모듈로 구성되어 있는 환경에서 16개의 32 비트 데이터 버스터 모드 전송 시 latency는 평균 7.1%의 증가를 보이나, 동일 실험 환경에서 버스 점유 cycle은 평균 64.9%의 감소를 보였다. 구현한 자동생성 시스템을 사용하여 버스 사용 효율이 향상된 인터페이스 회로를 생성해 낼 수 있었다.

V. 결론 및 추후과제

본 논문에서는 SoC 설계 환경에서 IP 사용으로 발생하는 인터페이스 회로 설계 비용의 증가와 짧은 time-to-market의 만족을 위해 자동으로 인터페이스 회로를 생성하는 시스템의 구축에 대해 제시하였다. 구현한 인터페이스 회로 생성기는 기존에 데이터 쉬트를 보며 설계자가 직접 인터페이스 회로를 설계하는 방식과 다르게 GUI를 통해 IP의 프로토콜을 기술할 수 있으며, 간단한 설계 스펙의 설정만으로 인터페이스 회로를 생성해 주는 장점을 보인다. 인터페이스 회로 생성을 위해 GUI를 통해 IP의 프로토콜이 기술되며, 기술된 프로토콜을 이용하여 프로토콜을 완벽하게 제어하는 듀얼 FSM을 생성한다. 동일한 프로토콜에 대한 분석과 공통으로 쓰이는 모듈에 대해서는 설계 부담을 줄이기 위해 라이브러리에 표준화된 버스의 프로토콜에 대한 듀얼 FSM과 버퍼를 저장하여 인터페이스 회로 설계시 표준화된 버스에 대해서는 프로토콜 분석이 필요없도록 하였다. 제안된 방법으로 리드 솔로몬 인코더, MPEG2 비디오 디코더와의 인터페이스 회로를 생성한 결과 매뉴얼로 작성한 인터페이스 회로에 비해 면적과 데이터 전송 latency는 증가하였지만, 시스템 버스 점유 cycle은 향상된 인터페이스 회로를 얻을 수 있었다. 제안된 인터페이스 회로 자동생성기를 사용하여 시스템 설계 시 매뉴얼 설계에 비해 성능이 떨어지지 않는 인터페이스 회로를 얻을 수 있으며, 매뉴얼로 인터페이스 회로를 설계할 시 소요되는 많은 시간을 절약할 수 있다.

현재 생성할 수 있는 모듈 형태는 슬레이브형 모듈 밖에 지원하지 않는다. 마스터 모듈의 인터페이스 회로는 글루로직과 버퍼뿐만 아니라 write/read 인스트럭션에 따른 포트 출력을 발생하기 위해서는 디바이스 드라이버를 생성해야 한다. 다양한 마스터 모듈에 따라 하위 수준의 디바이스 드라이버를 생성하는 연구가 차후 요구된다.

감사의 글

본 연구는 2004년도 서강대학교 교내 연구비 지원에 의해 이루어졌으며, 설계를 위한 툴은 IDEC에서 제공된 것이 사용되었음.

참 고 문 헌

[1] P. Chou, R. Ortega, and G. Borriello, "IPCHINOOK : An Integrated IP-based Design Framework for Distributed Embedded Systems", in Proc. DAC, New Orleans, LA, pp. 44-49, June 1999.

[2] S. Abdi, D. Shin, and D. Gajski, "Automatic Communication Refinement for System Level Design", in Proc. DAC, Anaheim, CA, pp. 300-305, June 2003.

[3] R. Ortega, L. Lavagno, and G. Borriello, "Models and Methods for HW/SW Intellectual Property Interfacing", in Proc. Int. Symp. on System Synthesis, Hsinchu, Taiwan, pp. 397-432, July 1998.

[4] A. Wenban, J. O'Leary, and G. Brown, "Codesign of Communication Protocols", IEEE Journal of Computer, Vol. 26 No. 12, pp. 46-52, Dec. 1993.

[5] P. Chou, B. Ortega, and G. Borriello, "Interface Co-Synthesis Techniques for Embedded System", in Proc. ICCAD, San Jose, CA, pp. 280-287, Nov. 1995.

[6] D. Shin and D. Gajski, "Interface Synthesis

from Protocol Specification", Technical Report CECS-TR-02-13, Univ. of California, April 2002.

[7] R. Passersome, J. Rowson, and A. Sangiovanni-Vincentelli, "Automatic Synthesis of Interface between Incompatible Protocols", in Proc. DAC, San Francisco, CA, pp. 8-13, June 1998.

[8] E. Walkup and G. Borriello, "Automatic Synthesis of Device Drivers for Hardware/ Software Co-design", Technical Report #94-06-04, Univ. of Washington, August 1994.

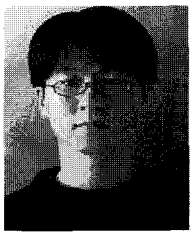
[9] L. Sancho-Pradel and M. Goodall, "System-on-Chip (SoC) Design for Embedded Real-Time Control Applications", in Proc. ESD Group Mini-Conference, Loughborough, UK, pp. 1-3, Sep. 2003.

[10] *AMBATM Specification(AHB) (Rev 2.0)*, ARM Ltd, May 1999.

[11] D. Gajksi, "IP-based Design Methodology", in Proc. DAC, New Orleans, LA, pp 43, June 1999.

[12] R. Passerone, J. Rowson, and A. Sangiovanni-Vincentelli, "Automatic Synthesis of Interface between Incompatible Protocols", in Proc. DAC, San Francisco, CA, pp. 8-13, June 1998.

이 서 훈(Ser-Hoon Lee) 준회원



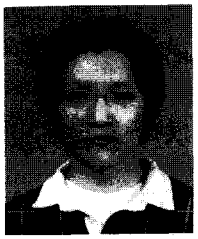
2003년 2월 서강대학교 전자공학과 졸업

2005년 2월 서강대학교 전자공학과 석사학위 취득

2005년 3월~현재 서강대학교 전자공학과 대학원 CAD & Embedded System 연구실 박사과정

<관심분야> SoC 설계, IP Interface 자동설계기법

문 종 욱(Jong-Uk Moon)



2003년 2월 서강대학교 전자공학과 졸업

2005년 2월 서강대학교 전자공학과 석사학위 취득

2005년 3월~현재 LG 전자 전자통신 연구소 연구원

<관심분야> SoC 설계, Bus bridge IP 설계

황 선 영(Sun-Young Hwang)

1976년 2월 서울 대학교 전자공학과 졸업

1978년 2월 한국 과학원 전기 및 전자공학과 공학석사 취득

1986년 10월 미국 Stanford 대학 전자공학 박사학위 취득

1976년~1981년 삼성 반도체 주식회사 연구원, 팀장

1986년~1989년 Stanford 대학 Center for Integrated System 연구소 책임 연구원
Fairchild Semiconductor Palo Alto Reaserch Center 기술 자문

1989년~1992년 삼성전자(주) 반도체 기술 자문

1989년 3월~현재 서강대학교 전자공학과 교수

<관심분야> SoC 설계 및 framework 구성, CAD 시스템, Computer Architecture 및 DSP System Design 등