NoC에서 면적 효율적인 Network Interface 구조에 관한 연구

준회원 이 서 훈*, 정회원 황 선 영*

An Area Efficient Network Interface Architecture

Ser-Hoon Lee* *Associate Member,* Sun-Young Hwang* *Regular Member*

요 약

여러개의 프로세서와 IP들로 이루어진 MPSoC 시스템은 모듈간 통신을 위해 NoC가 지원되어야 한다. NoC는 스위치의 추가만으로 시스템을 쉽게 확장할 수 있는 장점을 가지고 있으나, 시스템의 복잡도가 증가함에 따라 NoC를 구성하는 스위치의 수가 증가하게 되며, 증가된 스위치로 인해 전체 시스템 면적과 데이터 전송 latency가 증가하게 된다. 본 논문에서는 network interface를 공유하여 시스템에서 요구되는 스위치의 수를 감소시켜 전체 시스템의 면적 및 데이터 전송 latency를 감소시키는 방안을 제시한다. Network interface에 연결된 모듈간 버퍼를 공유하는 방식을 사용하여 network interface의 면적을 감소시켰다. 실험결과 스위치 수 및 network interface의 면적감소로 인해 전체 시스템의 면적은 기존에 비해 평균 46.5% 감소하였으며, 데이터 latency는 평균 17.1% 감소하였다.

Key Words : NoC, Network interface, MPSoC

ABSTRACT

NoC is adopted for data communication between processors and IPs in MPSoC system. NoC has an advantage of scalability in that system can be easily expanded just by adding switches. However, as the number of switches increases, chip area increases as well as data transfer latency. This paper proposes an architecture that can reduce the number of switches in the system by sharing network interfaces. To reduce NI area, the modules sharing network interface use a common buffer in network interface.

Experimental results show that the chip area has been reduced by 46.5% and data transfer latency by 17.1%, respectively, compared to conventional architecture.

Ⅰ. 서 론

반도체 공정이 deep sub micron으로 가면서 임베디드 시스템은 다양한 IP (Intellectual Property)들과 프로세서들을 하나의 칩에 넣는 SoC (System-On-a-chip) 형태로 발전했다. SoC의 성능 개선과 복잡도가 큰 어플리케이션을 지원하기 위해 SoC는 단일 프로세서 형태가 아닌 상당 수의 프로세서들을 내장한 구조의 MPSoC (Multi-Processor SoC)로 발전해 가고 있다[1][2][3][4]. 수많은 프로세서들 또는 PE(Process Element)들로 구성된 시스템은 모듈간 통신을 위해 대용량의 데이터 전송 대역폭과 고속의 데이터 전송 속도를 요구한다. 이러한 시스템에서 기존의 버스 구조를 사용할 경우 시스템

※ 본 연구결과물 (예: 논문, 특허, 표준, IP, SoC, IT 시스템 등)은 2008년도 「서울시 산학연 협력사업」의 「나노 IP/SoC 설계기술혁신사업단」의 지원으로 이루어졌습니다.
* 서강대학교 전자공학과 CAD & ES. 연구실 (lovemyself@sogang.ac.kr)
논문번호 : KICS2007-12-572 , 접수일자 : 2007년 12월 17일, 최종논문접수일자 : 2008년 5월 13일

확장을 위해선 연결선 수가 기하급수적으로 증가한다[5][6]. 연결선은 시스템에 capacitance, resistance을 증가시켜 데이터 전송시 RC delay를 증가시키며, 이로 인해 데이터의 전송을 지연하게 한다. 이를 해결하기 위해 "fat wire"와 "wide wire"가 제안되었으나 이로인해 칩의 면적과 파워 소모의 증가를 초래하여 시스템의 확장에는 한계가 있다[7]. 버스와 같은 구조는 통신을 위한 모듈간의 통신 선로를 공유하기 때문에 한번에 한 모듈만이 통신 선로를 사용할 수 있다. 이러한 구조는 여러 모듈이 동시에 통신 선로를 사용하고자 하는 경우를 지원하기 위해 arbitration이 반드시 요구된다. 복잡도가 높은 시스템에서는 동시에 통신을 해야 하는 모듈의 수가 크므로, 이들 모듈간의 arbitration으로 인한 데이터 전송 지연과 요청되는 통신 데이터 용량에 비해 전송되는 데이터 용량의 비율이 낮아 전체 시스템의 성능을 저하시킨다. 따라서 버스와 같은 구조는 수개 이하의 프로세서와 10개 이하의 버스 마스터를 가진 현재의 SoC에는 적합하나, 그 이상의 복잡도가 높은 시스템에서는 사용하기 부적합하다[8].

NoC (Network-On-a-Chip) 또는 OCN (On-Chip Network)은 높은 throughput과 시스템 확장의 용이성과 유연성 등을 지원하기 위해 제안되었다[7][8]. NoC에서는 스위치와 NIU (Network Interface Unit)의 추가만으로 쉽게 시스템을 확장 할 수 있을 뿐 아니라, 어플리케이션에 적합한 스위치 구조 토폴로지의 선택으로 인해 시스템의 thoughput을 향상시킬 수 있다[9]. Arteris 사의 technical report에 따르면, 72개의 IP들을 NoC를 적용하여 설계한 TDMA 시스템은 기존의 버스 구조를 적용한 시스템에 비해 데이터 전송 속도는 3배 이상, throughput은 10배 이상 증가함을 보인다[9].

NoC에서 시스템을 확장하기 위해서는 추가적인 스위치를 설치하고, 라우터에 추가된 스위치에 대한 정보를 업데이트하며 추가되는 모듈과 네트워크와 통신을 위해 NI (Network Interface)를 설계해야 한다. 추가되는 IP와 프로세서의 수가 증가함에 따라 요구되는 스위치의 수가 증가되어 이로 인한 칩 면적이 증가한다. 내부 네트워크에서 스위치간 통신 선로를 결정하는 라우터에서는 스위치의 수가 증가함에 따라 통신 선로를 결정하기 위한 알고리듬의 복잡도가 증가하여 선로의 재탐색이 요구될 경우 알고리듬 수행에 따른 추가적인 latency가 발행하다. 또한, 데이터가 목적지에 전송되기까지 경유하는 스위치의 수가 증가함으로 스위치를 통과할 때마다 발생하는 hopping delay가 중첩되어 전체 데이터 전송 latency가 증가하게 된다. IP 추가에 의한 칩 면적과 데이터 전송 latency의 증가는 전체 시스템의 성능 저하와 설계 비용을 증가시키므로 이를 보완하기 위해서는 새로운 설계 방법이 개발되어야 한다.

본 논문에서는 NoC를 이용하여 시스템을 설계할 경우 IP와 프로세서의 증가로 인해 발생하는 스위치의 증가를 최소화 할 수 있는 NI 구조를 제안한다. NI는 시스템에 추가되는 IP와 내부 네트워크간에 통신을 위해 프로토콜을 변형시켜 주는 역할을 수행한다. NI는 IP가 추가될 때마다 매뉴얼로 설계할 경우 설계 시간과 비용이 증가하게 되므로 자동으로 생성하는 방법이 요구되고 있다. UC Berkeley의 Sangiovanni-Vincentelli와 UC Irvine의 D. Gajski는 상이한 통신 프로토콜을 가진 모듈간 통신을 위한 인터페이스 로직 자동생성에 관한 방법을 제시하였다[10][11]. 그러나 NoC를 사용하는 MPSoC 환경에 적용하기 위해서는 확장과 수정이 요구된다. NoC를 사용하는 MPSoC 시스템에 적용 가능한 NI의 생성 방법은 A. Jerraya에 의해 제안되었다[11]. VP (Virtual Port)와 프로토콜을 정의하여 설계 되어지는 IP의 프로토콜을 VP의 프로토콜에 맞게 변형시키는 방법을 사용한다. 이 알고리듬은 NoC에서 모듈과 네트워크간 인터페이스 로직의 자동 생성을 지원하여 설계 시간을 감소시킬 수 있으나, 인터페이스 로직의 구조로 인한 성능상 장점을 가지지 못한다. P. Wielage는 다양한 버스와 NoC간 연결을 위하여 버스들이 하나의 NI를 공유하는 구조를 제안했다[13]. 제안된 NI는 버스들이 하나의 NI를 공유함으로써 스위치의 수를 감소시킬 수 있으나 데이터의 전송량이 많은 버스들이 하나의 NI를 공유하기 때문에 NI에서 요구되는 버퍼의 용량이 매우 크다는 단점이 있다. 또한 NI 구조가 매우 복잡하여 데이터의 전송 latency와 면적이 증가하며 자동생성을 지원하기 어려워 IP 형태의 NI로 존재해야 한다.

본 논문에서는 두 개의 모듈이 하나의 NI를 공유하는 구조를 제안하여 전체 시스템에서 요구하는 스위치의 수를 감소시키도록 하였다. NI의 생성 과정은 상위 수준에서의 프로토콜 기술을 기반으로 자동생성 시스템을 이용하여 설계된다. 버퍼의 제어 신호와 모듈과 통신을 위한 모듈 프로토콜 신호를 출력하는 FSM은 NI를 공유하는 모듈의 상위 수준에서 기술한 프로토콜을 병합하여 생성한다. 생성된 FSM, 라이브러리에 저장된 네트워크 프로토콜

FSM과 버퍼를 연결하여 NI를 자동설계 한다. NI의 면적을 감소시키기 위해 공유되는 두 모듈이 NI 내부의 버퍼를 공유할 수 있도록 하였다.

본 논문의 Ⅱ절에서는 제안된 NI의 설계 구조와 자동설계 알고리듬에 대해 기술하며 Ⅲ절에서는 시뮬레이션 결과 및 성능 분석을 보인다. 마지막으로 Ⅳ절에서는 본 연구의 결론을 제시한다.

Ⅱ. 제안된 NI 구조와 자동생성 방법

본 절에서는 NI의 구조에 대해 기술한다. 2.1절에서는 기존의 모듈간 인터페이스 로직 자동생성 알고리듬을 확장하여 생성된 NI 구조를 기술하며, 2.2절에서는 스위치 수를 감소시키며 동시에 NI 면적을 감소시키는 구조를 제안한다.

2.1 모듈간 인터페이스 로직 자동생성 알고리듬을 이용한 일반적인 NI구조

기존의 인터페이스 로직 구조와 이의 자동생성 알고리듬을 기반으로 NoC에 확장하여 적용하였다[14]. 설계자가 모듈에 대한 통신 프로토콜을 FSM 형태로 기술하면 자동생성 시스템은 FSM 형태로 기술된 프로토콜을 이용하여 데이터 전송/수신 시 모듈을 오류없이 동작시키게 하는 FSM을 생성한다. 네트워크 프로토콜에 따라 오류없이 동작하게 하는 FSM은 모듈의 인터페이스 생성시마다 동일한 FSM을 생성하므로 라이브러리에 저장하여 NI의 자동생성시 마다 재사용할 수 있다. 그림 2.1은 네트워크 프로토콜 FSM, IP 프로토콜 FSM, 그리고 dual-port buffer를 연결하여 생성한 인터페이스 로직을 보인다. 그림 2.1의 인터페이스 로직은 네트워크 프로토콜로 동작하는 FSM이 연결된 스위치와 통신하여 데이터를 버퍼에 저장하게 된다. 데이터가 저장되면 네트워크 프로토콜로 동작하는 FSM은 IP 프로토콜로 동작하는 FSM에 임의의 신호를 전송하며 신호를 받은 IP 프로토콜 FSM은 버퍼에 있는 데이터를 IP에 전송할 수 있는 동작을 수행한다. 데이터를 IP에서 네트워크로 전송하기 위해서는 IP 프로토콜 FSM을 이용하여 데이터를 내부 버퍼에 저장하며, 타겟 선택을 위한 주소는 address mapping table을 이용하여 모듈이 사용하는 실제 주소를 네트워크 주소로 변환하여 출력한다. 네트워크 프로토콜 FSM은 변환된 네트워크 주소를 라우터에 출력하여 목적지까지의 경로가 설정되도록 하며, 경로가 설정된 후 내부 버퍼의 데이터를 네트워크에 전송한다. Address mapping table은 시스템 설계시 정의한 파라메터 값으로 모듈에서 사용하는 실제 주소를 네트워크 주소로 변환하는 기능을 수행한다. 내부에 사용되는 버퍼는 dual-port buffer를 사용하여 각 포트에서 동시에 access가 가능하게 하여 데이터 전송 지연을 최소화 하였으며, 각 포트에서는 상이한 클록으로 동작할 수 있게 설계하여 네트워크와 IP간 동작 속도의 차이가 발생하더라도 동작이 가능하게 설계하였다.

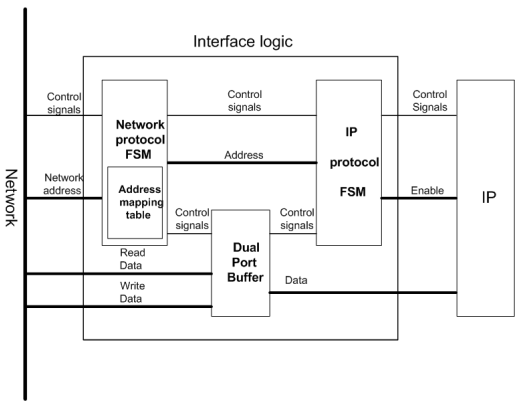


그림 2.1. 모듈간 인터페이스 로직 자동생성 알고리듬을 이 일반적인 NI 구조

2.2 스위치 수의 감소를 위한 NI의 구조

다수의 IP와 프로세서들로 구성되는 MPSoC 시스템에 NoC를 사용하기 위해서는 각 프로세서와 IP마다 NI를 설계하여 wire를 통해 스위치와 연결한다. 그림 2.2는 mesh 구조로 설계된 NoC의 구조를 보인다. 각 모듈은 네트워크와 통신을 위해 자동생성한 NI를 가지고 있으며, NI은 4방향으로 데이터를 보낼 수 있는 스위치와 연결된다. 모듈의 수가 증가함에 따라 발생하는 스위치 수의 증가는 라우터에서의 데이터 전송 경로 설정의 복잡도와 면적을 증가시킨다. 뿐만 아니라, 전송되는 데이터가 스위치를 지날 때마다 발생하는 hop delay의 중첩으

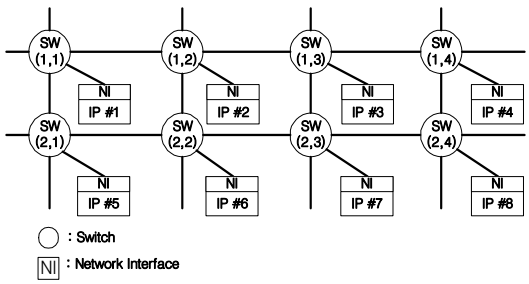


그림 2.2. 일반적인 NI를 사용한 NoC

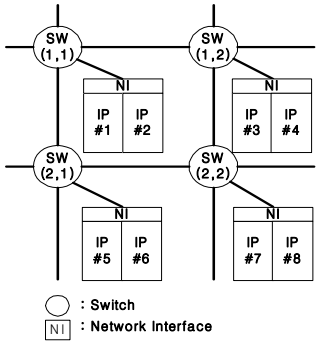


그림 2.3. 두 모듈이 하나의 NI를 공유하는 NoC

로 인해 전체 시스템의 성능이 저하된다.

칩의 집적도가 증가함에 따라 모듈은 massive 블럭이나, clustered PE 들, 프로세서들 그리고 메모리 같은 하나의 큰 단위의 (coarse grain) 모듈을 사용할 경우 하나의 모듈에 하나의 스위치를 할당하는 것이 효율적이 지만, MPEG decoder, LCD driver, memory controller와 같은 작은 단위의 (fine grain) 모듈을 사용할 경우 모듈의 크기에 비해 스위치 수의 증가로 인한 면적 증가가 더 크므로 작은 단위의 모듈에 하나의 스위치를 할당하는 것은 비효율적이다. 작은 단위의 모듈을 사용하여 설계할 경우 하나의 NI를 여러 개의 모듈이 공유하게 된다면 전체 스위치의 수는 감소하게 되어 시스템 전체의 성능을 향상시킬 수 있다. 그림 2.3은 두개의 모듈이 하나의 NI를 공유하도록 설계되는 NoC의 구조를 보인다. 두 개의 모듈이 하나의 NI를 공유할 경우 전체 스위치 수는 반으로 감소시킬 수 있으므로 데이터 전송 속도 증가와 시스템 면적 감소의 효과를 가져온다.

2.2.1절에서는 공유된 모듈에게 연속된 데이터를 분배시켜주는 NI의 구조를 제안하며, 2.2.2절에서는 이를 발전시켜 면적을 크게 줄인 NI를 제안한다.

2.2.1 연속된 데이터를 분배시켜주는 공유된 NI의 구조

네트워크를 통해 전송되는 데이터들은 각각의 목적지를 가지고 있다. 제안된 구조는 이들 데이터를 한 NI에서 받아 각 타겟 모듈에 전송해 주는 방식을 사용하여 모듈이 NI를 공유할 수 있도록 한다. 이러한 일을 수행하기 위해서는 기존에 사용되는 네트워크 주소의 확장이 필요하다. 기존의 네트워크 주소는 mesh 토폴로지를 사용한 NoC 구조에서 스위치의 주소를 메트릭스의 행과 열 정보와 같은 위치 정보로 설정한다[15]. 네트워크 주소는 *(x, y)-tuple*로 표현하며 이때 *x*와 *y*는 각각 스위치가 위치한 열과 행의 정보를 가지고 있다. 확장된 주소는 기존의 네트워크 주소를 유지하면서 새로운 필드를 기존 주소에 추가하여 *(x, y, i)-tuple*로 표현한다. 새로 추가된 *i* 필드는 NI를 공유하는 모듈의 정렬된 순서로 정의된 순번을 나타낸다. 데이터 전송을 위한 경로를 결정하는 라우터는 새로운 주소의 확장과 관계없

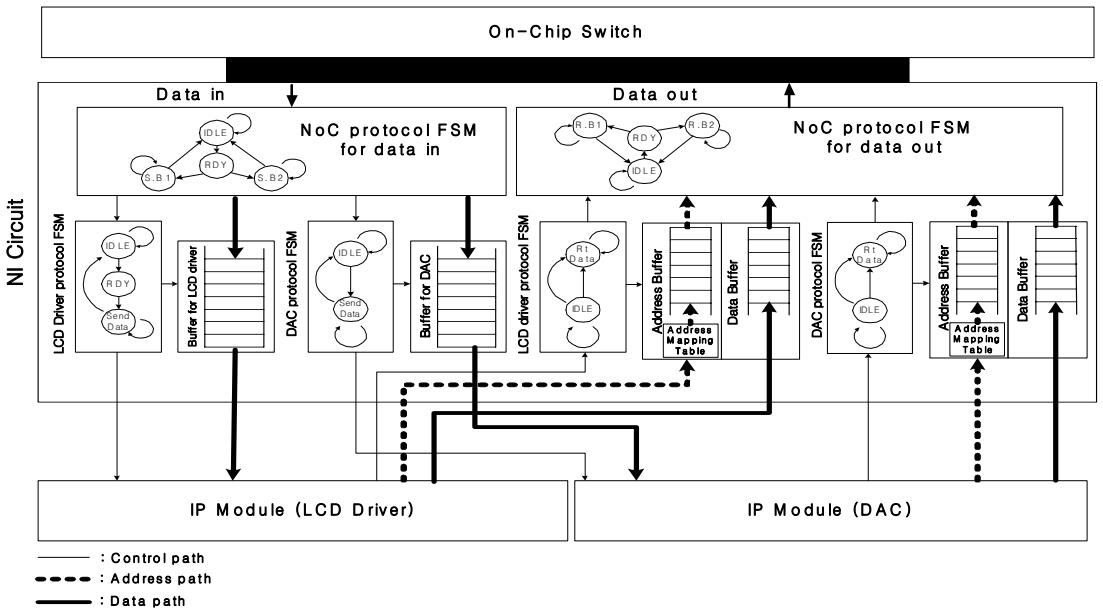


그림 2.4. 모듈에 데이터를 분배해 주는 공유된 NI 구조

이 기존의 주소 필드 만으로 경로를 결정하게 된다. 네트워크를 통해 NI까지 전송된 데이터는 *i* 필드를 디코딩하여 해당 타겟 모듈에 데이터를 분배하게 된다. 따라서, 주소의 확장과 관계없이 기존의 라우터의 경로 결정 알고리듬 및 스위치를 재사용 할 수 있다. 그림 2.4는 두개의 모듈이 하나의 NI를 공유할 경우에 각 모듈에 데이터를 전송시켜 줄 수 있는 NI의 구조를 기술한다. On-Chip 스위치를 통해 전송되는 데이터는 확장된 네트워크 주소의 *(x, y, i)-tuple* 중에 NI를 공유하는 모듈의 순서에 따라 인덱스한 *i* 의 값에 해당하는 타겟 버퍼에 NoC 프로토콜 FSM을 통해 저장된다. 공유된 각 모듈에 해당하는 타켓 버퍼에 데이터가 저장된 후 모듈에 데이터를 전송하기 위한 동작은 2.1절에서 기술한 일반적인 NI의 동작과 동일하게 수행된다.

두개 이상의 모듈이 하나의 NI를 공유할 경우 NI를 공유한 내부 모듈간 데이터 통신이나 모듈의 동시 데이터 전송 요청이 있을 경우 내부 communication media를 사용하기 위한 arbitration 과정이 필요하게 된다. 이 경우 arbitration으로 인한 overhead로 인해 버스 구조를 사용할 경우 같은 데이터 전송 지연이 발생하여 전체 시스템의 성능을 저해한다. 본 논문에서는 두개의 모듈이 하나의 NI를 공유하도록 하였다. 또한, packet-switching 방식의 NoC 구조에서는 산발적으로 수신되는 패킷의 재배열(scheduling)을 위한 buffer pool과 scheduler가 요구되어 NI 면적을 급격히 증가시키는 문제가 발생하므로, 데이터가 순차 전송되어 buffer pool이 필요하지 않은 circuit-switching 방식의 NoC를 적용하여 설계하였다.

2.2.2 면적 효율적인 공유된 NI 구조

앞절에서 제안한 확장된 주소를 이용하여 여러개의 모듈이 하나의 NI를 공유할 수 있는 인터페이스 로직을 생성해 낼 수 있다. 그림 2.4와 같이 NI를 공유하는 모듈당 자신의 버퍼를 가질 경우 상이한 속도로 동작하는 모듈 간에도 하나의 NI를 공유할 수 있는 장점이 있으나, 대부분 GALS (Globally Asynchronous Locally Synchronous)으로 설계되는 시스템에서는 특별한 장점을 갖지 못한다. 오히려 버퍼의 증가로 인해 NI의 면적이 증가하는 단점이 발생한다. 이러한 면적 증가를 감소시키기 위해 버퍼를 공유할 수 있는 방법을 제안한다.

기존 구조에서 버퍼를 공유하기 위해서 각 모듈별로 존재하던 FSM을 통합하였으며, 하나의 버퍼에 데이터를 저장하는 것과 동시에 해당 데이터의 타겟 모듈에 해당하는 인덱스도 동시에 저장하게 하였다. 그림 2.5는 공용 버퍼를 사용하는 NI의 구조를 보여준다. 데이터가 모듈로 전송되어질 경우 확장된 네트워크 주소의 *(x, y, i)-tuple*에서 *x, y*의 값을 참조하여 전송되어진 모든 데이터는 NoC 프로토콜 FSM을 통해 내부 버퍼에 저장되며, i 값은 인덱스 버퍼에 저장된다. 버퍼에 데이터가 전송되면 병합된 프로토콜 FSM (Merged protocol FSM)에

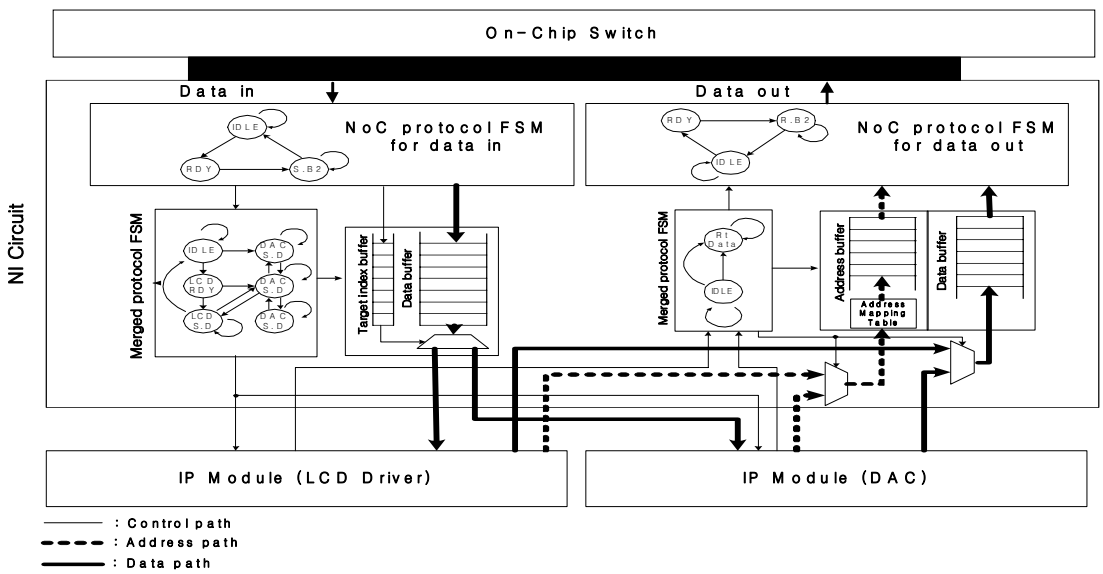
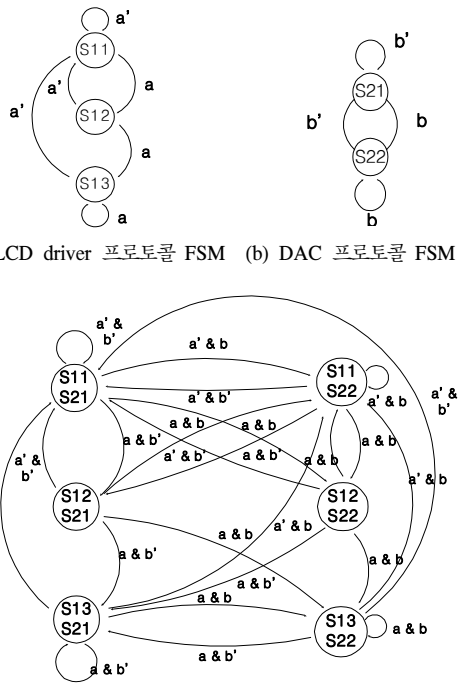


그림 2.5. 공용 버퍼를 사용하는 NI 구조

데이터의 저장을 알린다. 병합된 프로토콜 FSM은 기존에 각 모듈별로 존재하는 프로토콜 FSM을 하나로 병합한 것으로써 데이터의 타겟 모듈의 프로토콜에 따른 적절한 신호를 출력한다. 버퍼에 데이터가 전송되었다는 신호를 받은 병합된 프로토콜 FSM은 인덱스 버퍼에 값을 lookahead 하여 전송되어질 데이터가 전송을 위하여 모듈에 출력하여야 할 제어 신호들을 미리 출력한다. 이를 위해 FSM의 초기 상태로부터 데이터의 전송을 위한 상태까지 trace하여 제어 신호를 출력하는데 요구되는 사이클을 줄였다. 데이터가 모듈에서 네트워크로 전송되어질 경우 해당 모듈은 통합된 프로토콜 FSM과 통신을 통해 내부 버퍼에 데이터와 네트워크 주소로 변환된 주소를 저장한다. 이때 두개의 모듈이 동시에 접속할 경우 mux를 이용하여 충돌을 방지한다. 통합된 프로토콜 FSM은 미리 설정된 우선순위를 적용하여 두 모듈의 데이터 중 우선순위가 높은 모듈의 데이터를 먼저 받아들인다. 버퍼에 데이터가 저장되면 통합된 프로토콜 FSM은 NoC 프로토콜 FSM에 신호를 보내며, 신호를 받은 NoC 프로토콜 FSM은 버퍼가 empty 될 때까지 주소 버퍼에 저장된 값을 이용하여 데이터를 On-Chip 스위치에 전송한다. 통합된 프로토콜의 생성과 동작방법은 데이터의 receive/ send 의 경우 유사하므로 receive의 경우에 대해서만 기술하도록 한다. 모듈의 프로토콜을 FSM으로 기술한 후 각 상태와 상태 천이 신호를 Cartesian product하여 기본적으로 병합된 프로토콜 FSM을 생성할 수 있다. 그림 2.6은 LCD driver, DAC (Digital Analog Converter)의 프로토콜 FSM와 이를 병합한 FSM을 보인다. 프로토콜 FSM의 병합에 사용된 각 모듈의 프로토콜 기술은 잘못된 상태로의 분기로 인한 오류를 방지하기 위해 completely specified FSM으로 기술되어 있다. 따라서, 이를 이용하여 Cartesian product의 결과로 생성된 병합된 프로토콜 FSM은completely specified FSM이므로 입력의 모든 경우에 대한 분기를 가져 잘못된 상태로의 분기로 인한 오류를 유발하지 않는다[16]. 그림 2.6의 LCD driver의 프로토콜 FSM에서 데이터의 전송이 이뤄지는 상태는 "S13"이며 DAC의 프로토콜 FSM에서 데이터의 전송이 이뤄지는 상태는 "S22"이다. 그 외 상태는 모듈의 프로토콜에 맞는 제어 신호를 전송해주는 상태이다. 그림 2.6의 병합된 프로토콜 FSM에서 "S13S22"는 LCD driver와 DAC에 데이터를 동시에 전송해 주는 상태이다. 그림 2.6에서 제시한 NI의 구조에서



(a) LCD driver 프로토콜 FSM (b) DAC 프로토콜 FSM

(c) 병합된 프로토콜 FSM.

그림 2.6. 프로토콜 FSM의 병합

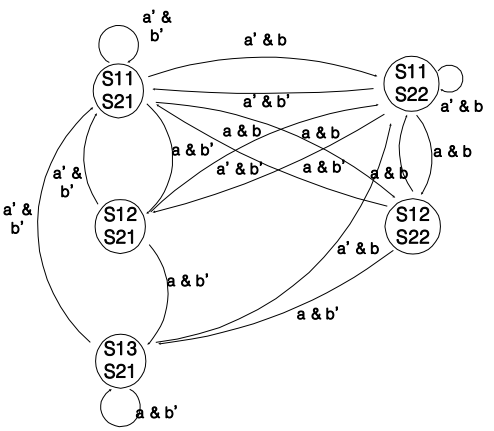
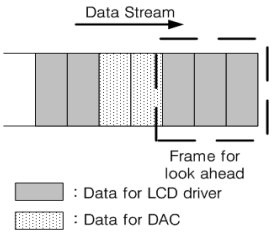


그림 2.7. 재설계된 프로토콜 FSM

입력된 데이터는 버퍼에 저장된 후 순차적으로 모듈에 전송되므로 공유된 모듈에 데이터가 동시에 전송되는 상태로 분기되지 않는다. 따라서, 그림 2.6의 병합된 프로토콜 FSM의 "S13S22" 상태는 제거할 수 있다. 그림 2.7은 이러한 방법으로 재설계된 프로토콜 FSM을 보여준다. 재설계된 FSM은 그림 2.5의 타겟 인덱스 버퍼에 있는 i 값을 lookahead하여 데이터 전송 latency를 최소로 하는 상태로 분기



(a) 입력 데이터 순서

현재상태	현재 데이터	Lookahead		다음상태
		+1 사이클	+2 사이클	
S13S21				S13S21
S13S21				S13S21
S13S21				S11S22
S11S22				S12S22
S12S22				S13S21
S13S21			None	S13S21

(b) 데이터 순서에 따른 상태 천이.

그림 2.8. 입력 데이터 순서에 따른 상태 천이

하게 된다. Lookahead하는 인덱스의 값은 현재 데이터로부터 최대치 FSM의 “IDLE” 상태부터 데이터 전송을 위한 상태까지의 걸리는 사이클 수만큼을 읽어야 한다. 이는 데이터의 전송 상태 이전에 프로토콜에 맞는 신호를 출력하기 위한 상태들의 trace로 발생하는 지연을 최소화하기 위해서 버퍼에서 해당 데이터가 출력되기 이전에 프로토콜 신호들을 출력하기 위한 상태를 모두 trace해야 하기 때문이다. 그림 2.8에서는 현재기준으로 2개의 데이터를 lookahead해야 한다. 그림 2.8는 입력되는 데이터의 순서에 따라 병합된 프로토콜 FSM의 상태 천이되는 것을 보인다. 그림 2.8의 (a)는 인덱스가 저장된 버퍼이며 lookahead 프레임 만큼의 인덱스값을 읽어온다. (b)는 (a)에서 읽어온 인덱스 값에 따라 천이해야 하는 상태를 나타낸 표이다. 데이터의 타겟을 lookahaed하여 데이터를 전송하기 위해 IDLE 상태인 ″S11S12″에서부터 데이터의 전송 상태까지 소비되는 사이클을 다른 모듈에 데이터가 전송되는 사이클에 병행하여 수행함으로써 데이터 전송 latency를 감소시켰다.

그림 2.4와 그림 2.5에서 제안한 NI를 적용할 경우 입력되는 데이터가 블록 단위의 데이터로 전송된다면 두 방법의 NI의 성능 차이는 거의 없을 것이다. 오히려 그림 2.4의 NI에서 사용되는 버퍼의 사용효율은 그림 2.5에 비해 떨어진다. 입력되는 데이터의 타겟이 교차되어 전송되어질 경우, 그림 2.4의 NI를 적용하면 데이터를 분산하여 버퍼에 저장하기 위해 NoC 프로토콜 FSM에서 소비되는 사이클에 의해 데이터 전송 latency가 증가한다. 그러나, 모듈에 비해 On-Chip 스위치 및 NoC 프로토콜 FSM이 높은 동작 속도를 갖는다면 성능상 저하는 발생하지 않는다. 반면 그림 2.5의 구조를 사용한다면 버퍼가 쉽게 full 될 수 있지만, NI를 공유하는 모듈의 동작이 서로 orthogonal 하거나 버퍼의 용량을 적절히 조절하면 그림 2.5에서 제안하는 NI는 그림 2.4의 것과 유사한 성능을 갖게 되며, 오히려 버퍼의 감소로 인한 NI의 면적 감소 면에서 매우 효율적이다.

Ⅲ. 실험결과

제안한 NI의 성능검증을 위해 circuit-switching 방식의 NoC를 적용하여 RTL 수준의 시스템을 구축하였다. 타겟 모듈은 LCD driver와 DAC의 데이터 송/수신 프로토콜을 VHDL을 이용하여 모델링하였다. 각 모듈에 일반적인 NI와 두개의 모듈이 공유하는 NI를 자동설계 시스템을 이용하여 생성한 RTL 수준의 인터페이스 로직을 연결하여 시뮬레이션을 수행하였다. 성능측정을 위해서 SunBlade 2500 Sparc, SunOS9 환경에서 Synopsys 사의 design compiler와 Mento graphics 사의 modelsim을 이용하여 수행하였다. NI의 면적 측정을 위해서는 TSMC 사의 90nm TCBN90GHP 공정 라이브러리를 사용하여 측정하였다. 제안된 NI의 동작검증을 위해 그림 2.2와 같은 4x4 mesh 구조의 NoC를 설계하였다. 네트워크 주소 (1,1)에 메모리와 DMA (Direct Memory Access)를 설계하였으며, 네트워크 주소 (1,1)과 가장 긴 거리를 보이는 네트워크 주소 위치에 타겟 모듈을 두어 DMA를 이용하여 메모리에 있는 데이터를 타겟 모듈에 전송하도록 하였다. NoC 및 모듈의 데이터와 주소는 32 비트로 하였으며, NoC의 동작 속도는 25 Mhz, 각 모듈의 동작 속도는 10 Mhz로 하여 수행하였다. 제안된 NI를 적용하기 위해 LCD driver와 DAC가 하나의 NI를 공유하도록 하였다.

표 3.1은 NI의 형태별 DMA의 전송 시작시부터 모듈에 데이터가 도달하는데 걸리는 latency의 결과를 보인다. 실험 결과 각 형태별 NI의 데이터 전송

표 3.1 NI 형태별 데이터 전송 latency

NI 형태		일반적인 NI		데이터를 분배해주는 공유된 NI		공용 버퍼를 사용하는 공유된 NI	
타겟 모듈		LCD Driver	DAC	LCD Driver	DAC	LCD Driver	DAC
네트워크 주소		(4,4)	(4,3)	(2,4)	(2,4)	(2,4)	(2,4)
데이터 전송 latency (μs)	소스모듈에서 NoC	0.20	0.20	0.20	0.20	0.20	0.20
	경로 설정 latency	5.17	4.63	4.19	4.19	4.19	4.19
	스위치 패스 latency	0.28	0.24	0.20	0.20	0.20	0.20
	NoC 에서 타겟모듈	0.40	0.30	0.40	0.30	0.40	0.30
	총합	6.02	5.37	4.99	4.89	4.99	4.89

표 3.2 NI의 형태에 따른 칩의 면적

면적 (cell) \ NI 형태		일반적인 NI	데이터를 분배해주는 공유된 NI	공용 버퍼를 사용하는 공유된 NI
모델링된 모듈	DMA	103	103	103
	LCD Dirver	114	114	114
	DAC	80	80	80
NI	NoC 프로토콜 FSM	784	437	335
	버퍼	8,414	8,414	3,890
	모듈 프로토콜 FSM	207	207	373
	그 외	64	38	38
	총합	9,472	9,096	4,636
스위치		13,968	6,984	6,984
라우터		5,179	3,547	3,547
총합		28,916	19,924	15,464

단위 : 1 cell = 2 NAND

이 오류 없이 전송되었다. 일반적인 NI를 사용할 경우 네트워크 주소 (4,4)에 위치한 LCD driver 모듈과 (4,3)에 위치한 DAC 모듈의 실험 결과 값은 네트워크의 주소에 따라 상이하게 도출된 것을 볼 수 있다. 두개의 모듈이 NI를 공유할경우 시스템에서 요구하는 스위치 수는 감소하게 되어 LCD driver와 DAC를 네트워크 주소 (2,4)에 위치시킬 수 있다. 결과 제안한 NI를 적용하여 설계하였을 경우 데이터 전송 latency는 일반적인 NI를 적용하여 설계하였을 경우에 비해 17.1% 향상되었다. 이는 제안된 구조의 NI를 사용할 경우 데이터의 전송 요청시 경로를 설정하는 과정에서 요구되는 latency와 데이터가 지나가는 스위치의 감소로 인한 스위치 경로 latency의 감소에 의한 것이다. 실험에서 측정된 스위치 한개 증가시 경로 설정 latency는 0.44μs, 스위치 경로 latency는 0.04μs 증가하였다. 스위치 경로 latency의 중첩과 경로 재설정 시간을 감소시켜 전체 시스템의 성능 향상이 가능하였으며, 제안된 NI를 사용하여 전체 시스템에서 요구하는 스위치의 수도 감소하여 면적을 크게 줄일 수 있다.

표 3.2는 제안된 NI의 사용으로 인해 감소된 칩의 면적을 보여준다. 제안된 데이터를 분배하는 공유된 NI와 공용 버퍼를 사용하는 공유된 NI를 적용할 경우 일반적인 NI를 적용할 때와 비교하여 시스템 면적이 각각 31.0%, 46.5% 감소하였다. NoC에서 측정한 스위치 하나의 면적은 873cell이며, 제안된 NI의 적용으로 인한 스위치 수의 감소, NI의 면적감소와 라우터의 로직 복잡도 감소로 전체 면적이 감소하였다. NI의 면적은 NI내에 존재하는 버퍼가 16개의 Word 데이터를 저장할 수 있도록 하여 측정하였다. 데이터를 분배해 주는 공유된 NI는 일반적인 NI에 비해 51.0%, 데이터를 분배해 주는 공유된 NI에 비해 49.1% 감소하였다. 데이터를 분배해 주는 공유된 NI는 LCD driver와 DAC에 각각 사용되는 일반적인 NI 내의 NoC 프로토콜 FSM이 통합된 FSM으로 구성되어 있어 소폭의 면적 감소를 보이며, 공용 버퍼를 사용하는 공유된 NI는 각 모듈의 데이터와 주소를 저장하기 위해 분리되어 사용되는 버퍼를 하나의 공유된 버퍼로 통합한 결과 큰 면적 감소를 보인다. 그러나, 공용 버퍼를 사용하는 공유된 NI는 버퍼의 full 이 발생하게 되면 모듈에서 버퍼의 데이터를 전송받아 버퍼가 full 에서 벗어날 때까지 스위치에서 해당 NI로의 데이터 전송이 불가능하다. 버퍼의 full 되는 횟수와 시간은 NI 내부 버퍼의 용량에 따라 변화되므로 버퍼의 용량 최적화 작업이 필요하다.

전송되는 데이터의 형태에 따라 성능을 비교해 보기 위해 네트워크 주소 (1,1)에 위치한 하나의 데이터 소스에서 LCD driver와 DAC 모듈에 16개의 데이터를 하나씩 교차적으로 전송할 경우와 16개의 데이터를 4개 단위로 전송할 경우에 대해 실험하였다. LCD driver와 DAC 모듈이 공유한 NI가 위치한 네트워크 주소는 (2,4)이다. 표 3.3에서 하나의 데이터 소스에서 두 개의 타겟 모듈에 데이터를 교

표 3.3 전송되는 데이터 형태에 따른 데이터 전송 latency

NI 형태			데이터를 분배해주는 공유된 NI	공용 버퍼를 사용하는 공유된 NI
데이터 전송 Latency (μs)	데이터 교차 전송시	소스모듈에서 NoC	6.40	6.40
		경로 설정 latency	4.19	4.19
		스위치 패스 latency	6.40	6.40
		NoC에서 타겟모듈	11.20	2.40
		총합	28.19	19.39
	데이터 블록 전송시	소스모듈에서 NoC	4.80	4.80
		경로 설정 latency	4.19	4.19
		스위치 패스 latency	6.40	6.40
		NoC에서 타겟모듈	5.60	2.40
		총합	20.99	17.79

차 전송할 경우 일반적인 NI에 비해 데이터를 분배해 주는 공유된 NI를 적용한 구조에서 적은 데이터 전송 latency를 보인다. 하나의 데이터 소스에서 일반적인 NI를 사용한 두 개의 타겟 모듈에 데이터를 교차 전송할 경우 데이터 전송 시마다 데이터 전송 경로를 재설정해 주어야 하는 반면, NI를 공유한 모듈에 전송할 경우 타겟 모듈까지의 데이터 전송 경로를 재설정하지 않아도 되기 때문에 데이터 전송 latency가 감소하였다. 데이터를 분배해 주는 공유된 NI에 비해 공용 버퍼를 사용하는 공유된 NI를 적용할 경우 약간의 성능 향상을 보인다. 이는 데이터를 분배해 주는 공유된 NI 사용시 NoC 프로토콜 FSM에서 공유된 버퍼에 분배해서 데이터를 저장하기 위해 FSM의 상태를 천이 시켜야 하나 공용 버퍼를 사용하는 공유된 NI의 경우 공용 버퍼에 순차적으로 데이터를 저장하면 되기 때문에 상태 천이를 위한 latency가 요구되지 않기 때문이다. 버퍼에 저장된 데이터를 모듈에 전송할 경우, 인덱스 버퍼의 값을 lookahead하여 제어 신호를 출력하기 위해 필요한 상태 trace를 공유하는 다른 모듈에 데이터를 전송하는 시간에 병행하여 수행하기 때문에 NI에서 타겟 모듈까지 데이터를 전송하는 시간이 감소한다. 데이터를 블록 단위로 전송할 경우에는 일반적인 NI를 사용할 시 경로 재설정을 위해 요구되는 횟수가 데이터를 교차 전송할 경우에 비해 감소하여 전체 데이터 전송 latency를 감소시켰다. 데이터를 분배해주는 공유된 NI, 공용 버퍼를 사용하는 공유된 NI를 사용할 경우 경로 재설정을 위한 시간이 요구되지 않고 전송되는 데이터에 대한 look-ahead를 하므로 일반적인 NI를 사용할 경우에 비해 적은 데이터 전송 latency를 보인다.

제안된 NI를 사용하여 시스템 설계시 일반적인 NI를 사용할 경우와 비교하여 전체 시스템 면적은 46.5% 감소하였으며, NI 면적은 51.0% 감소하였다. 제안된 NI를 사용하여 시스템 구현시 데이터 전송 latency는 일반적인 NI를 사용했을 경우보다 17.1% 감소한 것을 볼 수 있다. NoC에서 제안된 NI를 사용하여 시스템의 면적과 latency가 감소된 성능 향상된 시스템을 설계할 수 있다.

Ⅳ. 결 론

본 논문에서는 MPSoC 시스템에서 모듈간 통신을 지원하기 위해 요구되는 NoC 구조의 성능을 향상시키기 위해 하나의 NI를 두 개의 모듈이 공유할 수 있는 구조를 제안하였다. 기존의 NI 공유방식에 비해 구조의 복잡도가 감소하고 자동생성 시스템을 지원하여 NI 설계 비용을 감소시켰다. 제안된 NI를 사용함으로써 시스템에서 요구되는 스위치 수를 감소시켰으며, 결과 라우터의 데이터 전송 경로 재탐색 latency와 스위치 경로 latency 감소로 데이터 전송 latency를 감소시켰다. NI를 공유하는 형태에 따라 데이터를 분배해 주는 NI와 공용 버퍼를 사용하는 NI를 제안하여 연속으로 전송되는 데이터를 타겟 모듈에 분배시키며, NI 자체의 면적을 감소시키기 위해 NI 내에 존재하는 버퍼를 공유하도록 하였다. 실험결과 제안된 NI를 적용시 기존 일반적인 NI를 적용했을 때와 비교하여 전체 시스템 면적은 46.5% 감소하였으며, 데이터 전송 latency는 17.1% 감소하였다.

제안된 NI는 자동생성 시스템을 이용하여 자동으로 생성할 수 있으므로 NoC에서 시스템 확장 시 스위치의 추가와 라우터의 정보 update외에 NI 설계로 인한 overhead는 없다. 그러나 NI 내부에서 사용되는 버퍼의 크기에 따라 성능의 차이를 보일 수 있으므로 보다 성능 좋은 NI를 구현하기 위해서는 버퍼 최적화 과정이 요구된다.

참 고 문 헌

〔1〕 Y.Lin, M.Kudlur, S. Mahlke, and T. Mudge, "Hierarchical Coarse-grained Stream Compilation for Software Defined Radio", in Proc. CASES, Salzburg, Austria, pp.115-124, Sep. 2007.

〔2〕 W. Casario, A. Baghdadi, and A. Jerraya, "Component-Based Design Approach for Multicore SoCs", in Proc. DAC, New Orleans, LA, pp.789-794, June 2002.

〔3〕 S. Borkar, "Thousand Core Chips - A Technology Perspective", in Proc. DAC, San Diego, CA, pp.746-749, June 2007.

〔4〕 J. Darringer, "Multi-Core Design Auto- mation Challenges", in Proc. DAC, San Diego, CA, pp.760-764, June 2007.

〔5〕 "A Comparison of Network-on-chip and Busses", Arteris, white paper, 2005.

〔6〕 R. Ho, K. Mai, and M. Horowitz, "The Future of Wires", proceedings of IEEE, Vol. 89, No.4, pp.490-504, April 2001.

〔7〕 T. Theis, "The Future of Interconnection Technology", IBM J. Research and Development, Vol.44, No.3, pp.379-390, May 2000.

〔8〕 L. Benini and G. Micheli, "Networks on Chips: A New SoC Paradigm", IEEE Computer, Vol.35, no. 1, pp.70-78, Jan. 2002.

〔9〕 W. Dally and B. Towles, "Route Packets, not Wires: On-Chip Interconnection Networks", in Proc. DAC, pp. 684-689, June 2001.

〔10〕 A. RAdulescu, J. Dielissen, K. Goossens, E. Rijpkema and P. Wielage, "An Efficient On-Chip Network Interface Offering Guaranteed Services, Shared-Memory Abstraction, and Flexible Network Configuration", in Proc. Design Automation Test Eur., pp.878-883, Feb. 2004.

〔11〕 R. Passersome, L. Alfaro, A. Henzinger, and A. Sangiovanni-Vincentelli, "Conver- tibility Verification and Converter Synthesis: Two Faces of the Same Coin", in Proc. Int. Conf. CAD, pp.132-139, Nov. 2002.

〔12〕 D. Shin and D. Gajski, "Interface Synthesis from Protocol Specification", Technical Report, CECS-TR-02-13, Univ. of Cali- fornia, April 2002.

〔13〕 A. Grasset, F. Rousseau and A. Jerray, "Network Interface Generation for MPSoC: from Communication Service Requirements to RTL Implementation", in Proc. Int. Workshop RSP, IEEE, pp.66-69, June 2004.

〔14〕 이서훈, 문종욱, 황선영, "FSM을 이용한 표준화된 버스와 IP간의 인터페이스 회로 자동생성에 관한 연구", 한국통신학회 논문지, 제 30권, 2A호, pp. 137-146, 2005년 2월.

〔15〕 G. Ascia, V. Catania, and M. Palesi, "Multi-objective Mapping for Mesh-based NoC Architectures", in Proc. CODES+ISSS, Stockholm, Sweden, pp. 182-187, Sep. 2004.

〔16〕 J. Hayes, *Introduction to Digital Logic Design,* Addison Wesley, 1993.

이 서 훈 (Ser-Hoon Lee) 준회원

2003년 2월 서강대학교 전자 공학과 졸업

2005년 2월 서강대학교 전자공학과 석사

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

<관심분야> SoC 설계, IP Interface 자동설계기법

황 선 영 (Sun-Young Hwang) 정회원

1976년 2월 서울대학교 전자공학과 졸업

1978년 2월 한국 과학원 전기 및 전자공학과 공학석사

1986년 10월 미국 Stanford 대학 전자공학 박사

1976~1981 삼성반도체 주식 회사 연구원, 팀장

1986~1989 Stanford 대학 Center for Integrated 연구소 책임 연구원

Fairchild Semiconductor Palo Alto Research Center 기술 자문

1989~1992 삼성전자(주) 반도체 기술 자문

1989년 3월~현재 서강대학교 전자공학과 교수

<관심분야> SoC 설계 및 framework 구성, CAD시스템, Com. Architecture 및 DSP System Design 등