효율적인 브로드캐스트 통신을 지원하는 MPI 하드웨어 유닛 설계

준회원 윤 희 준*, 정회원 정 원 영*, 이 용 석*°

The Design of MPI Hardware Unit for Enhanced Broadcast Communication

Heejun Yun* *Associate Member*, Wonyoung Chung*, Yong-surk Lee*° *Regular Members*

요 약

본 논문에서는 분산 메모리 아키텍처를 사용하는 멀티프로세서에서 가장 병목 현상이 심한 집합통신 중 브로드캐스트를 위한 알고리즘 및 하드웨어 구조를 제안한다. 기존 시스템에서 집합통신은 프로세싱 노드의 통신포트 상태가 busy 혹은 free 인지를 고려하지 않고 MPI library cell 에 의해서 점대점 통신으로 변환되어 진다. 만약 브로드캐스트 통신을 하는 동안에 간섭하는 점대점 통신이 있다면, 브로드캐스트 통신의 전송 속도는 저하된다. 따라서 본 논문에서는 각각의 프로세싱 노드의 상태를 고려하여 통신 순서를 결정하는 브로드캐스트 통신 알고리즘을 제안하였다. 제안하는 구조의 알고리즘은 각 프로세싱 노드의 상태에 따라, free 상태의 통신 포트를 가진 프로세싱 노드의 통신 포트에게 우선적으로 메시지를 송신하여 전체적인 집합통신 시간을 단축하였다. 본 연구에서 제안하는 브로드캐스트 통신을 위한 MPI 유닛은 SystemC로 모델링하여 평가하였다. 또한 본 구조는 16노드에서 브로드캐스트 통신의 성능을 최대 78% 향상시켰고, 이는 MPSoC(Multi-Processor System-on-Chip)의 전체적인 성능을 높이는데 유용하다.

Key Words : MPSoC, Message passing, Distributed memory, Broadcast, Collective operation

ABSTRACT

This paper proposes an algorithm and hardware architecture for a broadcast communication which has the worst bottleneck among multiprocessor using distributed memory architectures. In conventional systems, collective communication is converted into point-to-point communications by MPI library cell without considering the state of communication port of each processing node which represents the processing node is in busy state or free state. If conflicting point-to-point communication occurs during broadcast communication, the transmitting speed for broadcast communication is decreased. Thus, this paper proposed an algorithm which determines the order of point-to-point communications for broadcast communication according to the state of each processing node. According to the state of each processing node, the proposed algorithm decreases total broadcast communication time by transmitting message preferentially to the processing node with communication port in free state. The proposed MPI unit for broadcast communication is evaluated by modeling it with systemC. In addition, it achieved a highly improved performance for broadcast communication up to 78% with 16 nodes. This result shows the proposed algorithm is useful to improving total performance of MPSoC.

※ 본 논문은 지식경제부 출연금으로 ETRI, 시스템반도체진흥센터에서 수행한 IT SoC 핵심인력양성사업의 연구결과임.
* 연세대학교 전기전자공학과 프로세서 연구실(hjyun@mpu.yonsei.ac.kr, wychung@mpu.yonsei.ac.kr, yonglee@yonsei.ac.kr), (° : 교신저자)
논문번호 : KICS2011-06-240, 접수일자 : 2011년 6월 7일, 최종논문접수일자 : 2011년 10월 28일

Ⅰ. 서 론

최근 임베디드 시스템(embedded system)에서도 다양한 애플리케이션(application)과 그 연산의 복잡도가 증가하므로 고성능 프로세서 시스템에 대한 요구가 높아지고 있다. 하지만 단일 프로세서의 성능을 높이는 방법으로는 애플리케이션의 발전 속도를 따라가는데 한계가 있다. 하나의 칩 안에 다수의 프로세서와 메모리를 집적시키는 MPSoC (Multi-Processor System-on-Chip)는 단일 프로세서의 처리량을 여러 프로세서로 분산시킴으로써 시스템 전체의 성능을 향상 시킬 수 있다. 또한 전력 소모가 적기 때문에 임베디드 시스템 분야에서 연구가 활발히 진행되고 있다. 예로는 ARM사의 cortex-A9, Texas Instruments사의 OMAP, STI(Sony/Toshiba/ IBM)의 Cell 프로세서 등을 들 수 있다.

병렬 프로세서 간 데이터를 공유하는 방법은 크게 공유(shared) 메모리 방식과 분산(distributed) 메모리 방식으로 나뉜다. 공유 메모리 방식은 안전한 데이터 교환이 보장되고 프로그래밍이 쉽다는 장점이 있다[1]. 하지만 연결 노드의 개수가 늘어남에 따라 공유된 메모리에서 버스 트래픽 병목 현상에 의해 성능이 하락되고, 캐시 메모리의 일관성을 유지하기 위해 스눕(snoop) 오버헤드가 급격히 증가하는 단점을 가지고 있다[2]. 반면 분산 메모리 방식은 메시지 전달 방식을 통해 데이터를 공유하기 때문에 프로그래머가 상세히 지정해 주어야 하며, 데이터 전송 시 교착상태(deadlock)가 발생할 수 있다는 단점이 있다. 하지만 프로세서의 수가 늘어날수록 소비전력 및 수행 시간 등 데이터 전송 오버헤드가 작다는 장점이 있어 본 연구에서는 분산 메모리 구조를 사용하였다[3].

메시지 패싱(message passing)은 분산메모리 구조에서 가장 흔한 프로그래밍 모델로써 클러스터 기반의 시스템과 같은 고성능 컴퓨팅(HPC : High Performance Computing)에서 자주 사용된다[4]. 비록 메시지 패싱이 일반적으로 HPC에서 사용되고 있지만, 최근 임베디드 애플리케이션(embedded application)에서의 관심이 높아지고 있다[5]. MPI(Message Passing Interface) 표준은 메시지 패싱 라이브러리를 위한 API(Application Programming Interface)를 제공하며, MPI 표준은 사실상 현재 메시지 패싱의 표준으로 사용되고 있다.

MPI 표준은 다양한 점대점 통신(point-to-point communication), 집합 통신(collective communication)을 포함하고 있다. 이중 MPI 라이브러리에 정의된 집합 통신 함수는 MPI library cell에 의해 점대점 통신 함수들의 집합으로 변환되기 때문에 사용자에게 프로그래밍의 편리함을 제공하며, 이로 인해 사용자들이 자주 사용한다. 한 프로파일링 연구에선 특정 애플리케이션에서 전송 시간의 약 80%가 집합통신에 의한 것이라 명시하고 있다[6]. 이러한 집합 통신 함수 중에서 가장 흔히 사용되는 것 중 하나가 MPI_Bcast이다.

MPI_Bcast 루틴은 루트 노드의 메모리 데이터를 같은 커뮤니케이터(communicator) 내의 모든 노드의 메모리에 복사하는 통신 형태로써 일대다 통신에 속한다. 이러한 브로드캐스팅을 효과적으로 수행하기 위해 크게 3가지 형태로 연구가 진행되고 있다.

MPI 브로드캐스트 동작을 효과적으로 수행하기 위한 연구로는 우선 브로드캐스트 알고리즘에 대한 연구가 있다. Sequential tree 알고리즘을 시작으로 chain tree, binomial tree, binary tree, MST(Minimum Spanning Tree), distance MST, hybrid(Van de Gejin) broadcast, modified Van de Gejin broadcast 등 많은 브로드캐스트 알고리즘이 제안되었다[7][8]. 이러한 알고리즘 중에서 성능 향상과 함께 구현이 용이한 알고리즘을 MPI library cell에 포함하여 알고리즘에 따라 점대점 통신으로 변환한다.

두 번째로 이러한 여러 알고리즘 중 수 개의 알고리즘을 MPI library cell에 포함시키고 MPI 함수에 표현된 정보를 통해 알고리즘을 선택하는 연구가 진행중이다[9]. MPI_Bcast 루틴은 다음과 같다.

MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm.)

즉 "buffer"(버퍼의 시작 주소), "count"(버퍼 주소의 개수), "datatype"(데이터 종류)을 통해 데이터 메시지 크기, "root"를 통해 루트 노드의 위치, "comm."을 통해 커뮤니케이터에 속한 프로세스의 개수를 알 수 있고, 이러한 정보를 통해 알고리즘을 선택한다.

마지막으로 MPI 통신을 수행하는 하드웨어를 설계하여 점대점 통신을 가속화하는 연구도 진행 중이다[10]. 이와 같이 MPI 통신을 지원하는 하드웨어를 사용할 경우, 소프트웨어적으로 처리할 때보다 처리 속도가 빠르며, MPI 통신을 하드웨어가 전담 함으로써 프로세서의 CPI(Clock per Instruction)를 올릴 수 있다. 이러한 장점으로 인해 이전 연구에서는 MPI 하드웨어 유닛을 제안하고 설계하였다[11].

이러한 많은 연구가 진행되었으나, 현재까지 프로세싱 노드(processing node)의 상태를 고려한 연구는

진행되지 않았다. 현재의 브로드캐스트 알고리즘 중 sequential tree 알고리즘은 MPI_Bcast 루틴을 노드 번호순서의 점대점 통신으로 변환하여 순차적으로 진행한다. 따라서 브로드캐스트에 해당하지 않은 점대점 통신이 동시에 진행되는 경우, 브로드캐스트 통신이 노드번호순서에 따라 순차적으로 진행하여야 함으로 브로드캐스트 전의 간섭하는 점대점 통신이 종료된 후에 브로드캐스트 통신을 실행한다. 따라서 본 연구에서는 노드의 상태를 고려한 새로운 방식의 브로드캐스트 알고리즘을 제안하였고, 이를 적용한 MPI 하드웨어 유닛을 SystemC를 이용하여 설계하였다.

MPI 통신 방식은 동기송신(synchronous send), 준비송신(ready send), 버퍼송신(buffered send), 표준송신(standard send) 의 4가지 통신 모드로 구분할 수 있다. 먼저 동기송신은 송신 노드에서 송신 준비가 되었음을 수신노드에 알린다. 수신 노드는 수신 메시지를 호출하고 수신 준비가 되었음을 송신 노드에 알리고 송신 노드가 수신준비 완료 메시지를 받으면 데이터 전송이 시작된다. 준비 송신은 수신 노드에서 수신 준비가 되었음을 송신노드에 알리고, 송신 노드가 수신 준비 완료 메시지를 받으면 데이터 전송이 시작된다. 버퍼송신은 버퍼로 데이터를 복사해 두고 수신 준비가 완료되면 이 버퍼로부터 수신 노드로 송신을 하게 된다. 표준 송신은 메시지 크기에 따라 두 가지 모드로 작동한다. 메시지 크기가 임계값보다 작으면 시스템에 준비된 시스템 버퍼를 이용하여 버퍼 송신과 같이 작동하며, 임계값보다 메시지 크기가 크면 동기 송신처럼 작동하게 된다. 임계값은 시스템에 따라 다르게 설정할 수 있다. 이 중 새로운 브로드캐스트 알고리즘을 적용하기 위하여 4가지 통신모드 중 준비송신 방식을 사용한다. 준비송신의 경우 수신 노드에서 준비 메시지(Recv)를 송신 노드로 보냄으로(즉 수신 노드 쪽의 통신 포트가 비어 있음), 송신 노드는 노드들의 통신 포트 상태를 고려하여 준비 메시지를 받은 순서대로 메시지를 보낼 수 있다. 현재의 user-level의 MPI library cell 에서는 MPI_Bcast 루틴을 점대점 명령(MPI_Send, MPI_Recv)으로 풀어서 각각의 노드에 전해 준다. 하지만 제안하는 브로드캐스트 알고리즘이 적용되기 위해서는 각각 노드의 통신 포트 정보를 알 수 있는 MPI 유닛에서 MPI_Bcast 루틴을 처리하여야 한다. 따라서 MPI library cell에서는 MPI_Bcast를 루트(root) 노드에 해당하는 경우 MPI_Bcast_root 명령을, 잎(leaf) 노드에 해당하는 경우 MPI_Bcast_leaf 명령을 전해 주어야 한다. 이러한 MPI 통신을 위해 이전 연구에서는 점대점 통신을 지원하는 MPI 하드웨어 유닛을 설계하였고[11], 본 연구에서는 설계한 MPI 하드웨어 유닛에 브로드캐스트 유닛(broadcast unit)을 추가하여 최적화 하였다.

Ⅱ. 제안하는 알고리즘 및 MPI 하드웨어 유닛 구조

MPI 집합통신에는 MPI_Bcast, MPI_Gather, MPI_Reduce 등이 있다. 이 중 MPI_Bcast 는 루트 프로세스의 메모리에 있는 데이터를 동일 커뮤니케이터 내의 다른 모든 프로세스들의 같은 위치로 복사하는 일 대 다(one-to-all) 통신이다. 이때 MPI_Bcast를 실행하는 루트 프로세스의 노드를 루트 노드(root_node)라하고, 동일 커뮤니케이터 내의 다른 모든 프로세스의 노드를 잎 노드(leaf_node)라 한다. 만약 MPI 브로드캐스트 통신을 실행 시 다른 점대점 통신이 수행 중이면, MPI_Bcast 통신은 이 통신포트를 사용하기 위해 점대점 통신이 끝나기를 기다린다. 이와 같은 상황 발생 시 성능 저하의 요인이 된다.

2.1 제안하는 브로드캐스트 알고리즘

MPI 브로드캐스트 루틴(MPI_Bcast)은 컴파일러가 MPI library cell 내의 알고리즘을 사용하여 점대점 통신으로 변환한다. 그림 1은 브로드캐스트 명령이 sequential tree 알고리즘을 사용하여 실행되는 것을 보여주고 있다. 그림 1에서 PN은 프로세싱 노드를, 화살표와 번호는 통신의 순서를 나타내고 있으며 프로세싱 노드의 아래에는 해당하는 노드의 명령어 큐의 상태를 나타낸다. Sequential tree 알고리즘은 MPI 브로드캐스트 루틴을 그림 1(노드 0이 루트 일 때)과 같이 자신의 노드를 제외한 다른 노드들에게 노드번호 순서로 점 대 점 통신으로 변환한다. 즉, 루트(root) 노드는(노드0) MPI_Send 명령 3개로, 수신 노드들은(노드 1,2,3) MPI_Recv 명령으로 변환을 하고, MPI 유닛은 이러한 순서로 통신을 실행한다. 하지만 그림 2처럼 다른 점대점 통신이 실행되고 있는 상황에서, 노드 0은 메시지 큐에 쌓여 있는 순서대로 통신을 진행하기 위해 기존의 점대점 통신이 끝나기를 기다리므로, 그림 3의 순서로 통신을 실행 할 것이다. 즉, 진행하고 있는 노드 1, 2 간의 점대점 통신이 끝나고, 노드 1부터 3까지 순서대로 브로드캐스트 통신을 실행한다. 노드 1과 노드 2의 점대점 통신의 데이터 크기가 클수록 브로드캐스트 통신을 위해 기다리는 시간이 길어진다.

본 논문에서 제안하는 구조는 이러한 손실을 줄이기 위하여 그림 4와 같은 순서로 통신을 한다. 즉, 루

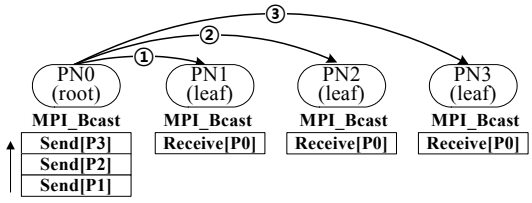


그림 1. 기존의 브로드캐스트 통신 순서
Fig. 1. The order of conventional broadcast communication

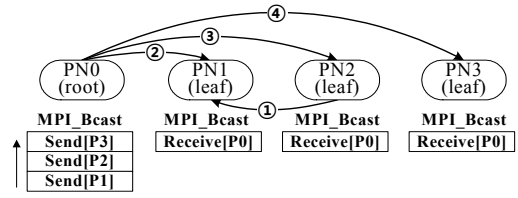


그림 2. 점대점 통신의 간섭 시 기존 구조의 브로드캐스트 통신 순서
Fig. 2. The order of broadcast communication during interference in point-to-point communications in conventional architecture

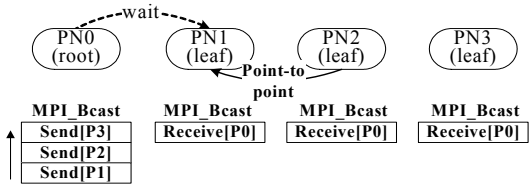


그림 3. 기존의 브로드캐스트 통신에서 점대점 통신의 간섭
Fig. 3. The interference in point-to-point communication in conventional broadcast communication

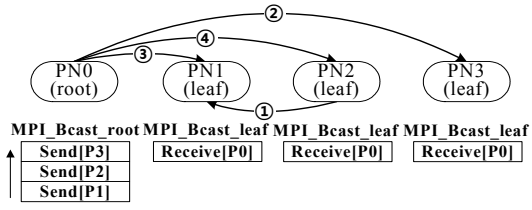


그림 4. 제안하는 구조의 브로드캐스트 통신 순서
Fig. 4. The order of broadcast communication in the proposed architecture

트 노드(0번째 노드)에서 비어있는 3번째 프로세싱 노드에 먼저 전송함으로써 손실을 줄일 수 있다. 이와 같이 제안하는 브로드캐스트 알고리즘이 적용되기 위해서는 user level 의 MPI library cell에서 MPI_Bcast 루틴을 루트 노드의 경우 MPI_Bcast_root 명령으로, 동일 커뮤니케이터 내의 루트 노드가 아닌 다른 모든 노드인 잎 노드의 경우 MPI_Bcast_leaf 명령으로 전해 주어야 한다. 또한 각 노드의 통신포트 상태를 고려하여 MPI_Bcast_root, MPI_Bcast_leaf 명령을 처리하는 하드웨어가 필요하다. 제안하는 알고리즘이 적용된 하드웨어로 브로드캐스트 통신을 실행하면 기다리는 시간을 최소화 하면서 브로드캐스트 통신을 완료할 수 있다.

2.2 제안하는 MPI 유닛 구조

그림 5는 제안하는 구조에서 사용하는 과정을 나타내고 있다. S는 송신 노드, R은 수신 노드를 나타내며, 시간 축에 따라 그림에 표시한 번호 순서대로 왼쪽에서 오른쪽으로 송수신이 진행된다. 즉, 준비(ready) 메시지를 시작으로 데이터 온(data on), 데이터(data), 완료(complete) 메시지 송수신을 거쳐 하나의 점대점 통신이 완료된다. 각각의 수신 노드와 송신 노드의 프로세서는 MPI Unit에서 MPI 명령이 끝날 때(response 신호)까지 대기한다. 그림 5처럼 준비 송신은 수신 노드가 준비 메시지를 송신 노드에게 보낸다. 즉 해당하는 노드의 통신 포트는 비어 있을 것이다. 따라서 브로드캐스트 루트 노드에서 점대점으로 변환되지 않은 브로드캐스트 명령을 처리할 수 있다면, 먼저 통신 포트가 비어 있음을 준비 메시지를 통해 알린 노드로 메시지 송신 작업을 수행할 수 있을 것이다.

그림 6은 제안하는 구조에서 루트 프로세싱 노드 0번이 브로드캐스트를 할 때 1번 프로세싱 노드와 통신하는 예를 나타낸다. MPI 유닛 내의 메시지 프로세싱 유닛을 기준으로 하여, 준비 송신 순서를 번호로 표시하였다. 수신노드인 프로세싱 노드 1의 메시지 프로세싱 유닛(message processing unit)에서 준비 메시지를 시작으로 완료 메시지까지의 과정을 번호로 나타내었고, 프로세서부터 메시지 프로세싱 유닛까지의 과정은 송수신 노드 따로 실행되기 때문에 번호 앞에 ' 표시를 첨부하였다.

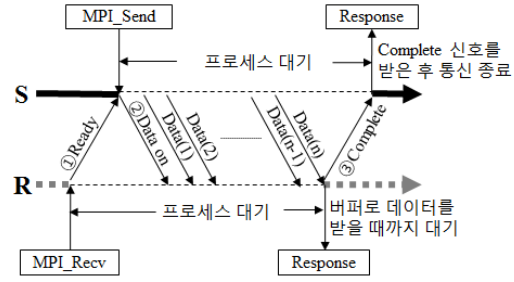


그림 5. 준비 송신 모드의 처리과정
Fig. 5. The process of ready send mode

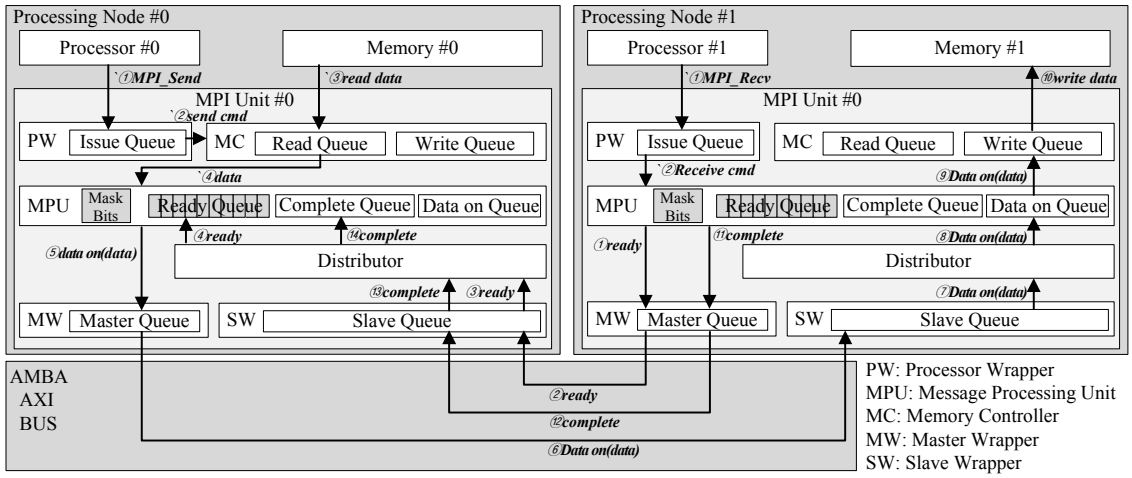


그림 6. 제안하는 구조의 MPI 통신과정
Fig. 6. The process of MPI communication in the proposed architecture

프로세서는 명령어 중 MPI 명령을 MPI 유닛을 이용하여 실행한다. MPI 유닛내의 프로세서 래퍼(processor wrapper)는 프로세서에서 MPI 유닛에게 주는 명령을 MPI 유닛에 맞추어 변환하는 역할을 하고, 메모리 컨트롤러(memory controller)는 데이터 메모리(Data Memory)에서 MPI 통신에 필요한 데이터를 읽고 쓰는 역할을 한다. 마스터 래퍼(master wrapper)와 슬레이브 래퍼(slave wrapper)는 AXI BUS와 MPI 유닛과의 인터페이스 역할을 한다.

제안하는 MPI 유닛의 구조는 MPI 유닛에서 브로드캐스트 통신을 루트 노드에서 MPI 유닛으로 준비(MPI_Bcast_leaf 명령) 메시지가 수신된 순서로 데이터 통신을 처리하기 위하여 분배기(distributor)가 추가되었고, 메시지 프로세싱 유닛의 내부구조가 변경되었다. 메시지 프로세싱 유닛 내에는 수신 노드와 데이터 통신의 완료를 의미하는 브로드캐스트 마스크 비트(broadcast mask bit)가 추가되었다. 따라서 브로드캐스트 루틴(MPI_Bcast)은 프로세서 래퍼에서 점대점 통신으로 변환 없이 메시지 프로세싱 유닛에서 수행하므로 점대점 통신의 간섭 시 손실을 최소화 한다.

2.2.1 분배기(Distributor)

분배기는 슬레이브 래퍼를 통하여 AXI BUS에서 들어오는 준비 메시지를 분리하여 메시지 프로세싱 유닛의 수신 버퍼에 저장한다. MPI 통신 시 루트 노드로는 데이터 값, 준비와 완료 메시지가 수신된다. 브로드캐스트 통신 시 준비 메시지가 해당하는 노드의 포트 상태를 나타내기 때문에 분리하여 주는 역할이 필요하다.

2.2.2 메시지 프로세싱 유닛(Message Processing Unit)

메시지 프로세싱 유닛은 분배기로부터 버스 수신신호를, 메모리 컨트롤러로부터 MPI 명령을 받아 처리한다. 또한 마스터 래퍼를 통하여 처리된 신호를 버스로 전송을 하는 유닛이다. 메시지 프로세싱 유닛에는 브로드캐스트 명령(MPI_Bcast_root, MPI_Bcast_leaf)을 효율적으로 처리하기 위하여 마스크비트가 추가되었다.

브로드캐스트 통신은 루트노드의 데이터를 같은 커뮤니케이터 내의 모든 노드에게 전송한다. 따라서 마스크 비트를 이용하여 각각의 노드들과의 데이터 전송의 유무를 표시하여 준다면, 오류 없이 브로드캐스트 통신이 진행 된다. 마스크 비트를 사용한 브로드캐스트 통신의 방법은 아래와 같다.

메시지 프로세싱 유닛에 브로드캐스트 루트(MPI_Bcast_root) 명령이 들어오면 내부에 있는 모든 마스크비트가 0이 되고, 수신 버퍼에서 쌍이 되는 수신 노드의 준비(MPI_Bcast_leaf 명령) 메시지를 찾는다. 쌍일 경우 데이터를 전송하고 전송한 노드에 해당하는 마스크 비트를 1로 표시한다. 아닌 경우에는 쌍이 될 때 까지 기다린다. 모든 마스크 비트가 1로 채워지면 데이터를 나머지 노드에 모두 보낸 상태이므로 브로드캐스트 루트 명령어가 종료된다.

수신 노드의 MPI 유닛이 입력받은 MPI_Bcast_leaf 명령은 노드 순서에 상관없이 준비 메시지로 변환하여 루트 노드로 전송한다. 따라서 루트 노드에서는 이 준비 메시지들을 수신 버퍼에 저장하고, 브로드캐스트

루트 명령(MPI_Bcast_root)과 쌍을 이룰 시 노드 순서에 관계없이 데이터를 전송한다.

이 구조를 사용하면 그림 4와 같은 순서로 브로드캐스트 통신을 할 수 있다. 노드 3으로부터 노드 0(루트노드)은 가장 먼저 준비(MPI_Bcast_leaf 명령) 메시지를 받게 되고, 노드 0의 메시지 프로세싱 유닛의 수신 버퍼에는 다른 노드의 준비 메시지보다 먼저 저장된다. 따라서 메시지 프로세싱 유닛에서는 브로드캐스트 루트 명령과 노드 3의 준비 메시지가 가장 먼저 쌍을 이루고, 노드 3과 데이터 통신을 한다.

Ⅲ. BFM 시뮬레이션 결과

제안하는 MPI 유닛의 성능을 측정하기 위해 이전에 연구되었던 모델을 수정하여 제안하는 모델과 비교하였다. 이전에 연구되었던 모델[11]은 MPI 동기 송신을 지원하고, MPI 명령의 비순차 (Out-of-Order)실행을 위하여 송신 명령을 위한 요청 큐(request queue), 수신 명령을 위한 준비 큐(ready queue), 명령의 펜딩(pending)을 위한 예약 큐(reserve queue) 총 3가지의 명령어 큐를 가지고 있다. 하지만 이전모델은 비순차 실행으로 인해 명령어간의 종속도(dependency) 검사(check) 유닛의 복잡도 증가로 이어졌다. 따라서 제안하는 구조에서는 비순차 실행이 아닌 순차실행을 사용하였고, 그 결과 3개의 큐가 아닌 이슈 큐(issue queue) 하나만 사용하였다.

이전에 연구되었던 모델은 브로드캐스트 통신에서 각 노드의 통신 포트 상태(busy, free)를 고려하지 못하여 그림 3의 순서로 통신을 하는 모델이고, 제안하는 모델은 통신 포트 상태를 고려하여 그림 4의 순서로 브로드캐스트 통신을 하는 모델이다. 각 모델은 SystemC를 사용하여 BFM(Bus Functional Model)을 설계하였다. BFM은 각 블록의 지연시간을 고려하여 동작을 기술하였고, 특정 시뮬레이션 환경에 따라 통신 트래픽을 생성할 수 있다. 그 후 생성된 통신 트래픽으로 제안한 MPI 유닛의 전송 시간을 확인하였다.

여러 가지 점대점 통신의 간섭 상황에서 성능을 평가하기 위해 case 별로 나누었다. 간섭하는 점대점 통신 없이 브로드캐스트 통신만 있는 경우 마지막 case(000...000), 간섭하는 점대점 통신이 노드 1번과 마지막 노드의 통신인 경우 case 1(010...01), 노드 2번과 마지막 노드의 통신인 경우 case 2(0010...01) 등 전체 node - 1 만큼의 case 로 나누어서 실험하였다. 표 1, 2, 3, 4 에서 case의 괄호안의 1은 간섭하는 통신의 노드를 나타낸다.

그림 7, 8과 표 1, 2는 8개의 프로세서 노드를 사용하여 브로드캐스트 통신의 성능을 평가한 것이다. 가로 축은 브로드캐스트 데이터 사이즈이고, 세로축은 식 (1)과 같은 브로드캐스트 통신에 걸린 시간이 기존의 모델에 비해 제안하는 모델이 빨라진 정도(speed-up ratio) 이다.

$$\text{speed-up-ratio}=\frac{\text{기존시스템의실행시간(sec)}}{\text{제안하는구조의실행시간(sec)}} \quad (1)$$

그림 7과 표 1은 간섭하는 노드의 데이터가 512Byte 인 경우고, 그림 8과 표 2는 간섭하는 노드의 데이터가 2048Byte 일 경우 성능 평가한 것이다. case 1 의 경우 성능이 가장 향상된 것을 확인할 수 있다.

그림 11은 기존구조의 case 1와 case 3의 브로드캐스트 통신 순서를 보여준다. case 3의 경우는 간섭하는 점대점 통신과는 상관없이 노드 2까지 브로드캐스트 통신을 하는 반면, case 1의 경우는 간섭하는 통신 때문에 브로드캐스트 통신을 진행하지 못하는 것을

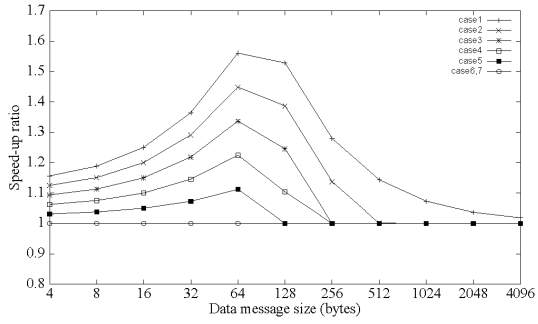


그림 7. 8 노드에서의 실험결과 (512 byte 의 간섭하는 점대점 통신)
Fig. 7. The simulation results for 8 nodes (512 bytes size of interfering point-to-point communication)

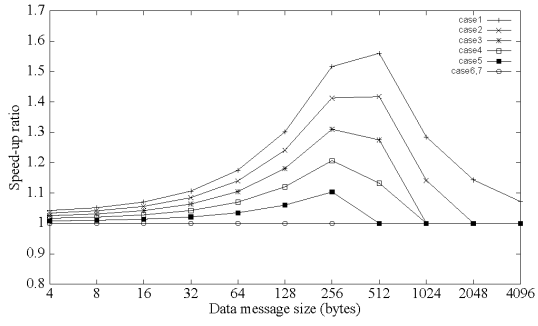


그림 8. 8 노드에서의 실험결과 (2048byte 의 간섭하는 점대점 통신)
Fig. 8. The simulation results for 8 nodes (2048 bytes size of interfering point-to-point communication)

표 1. 8 노드에서의 실험결과 (512 byte 의 간섭하는 점대점 통신)
Table 1. The simulation results for 8 nodes (512 bytes size of interfering point-to-point communication)

	4B	8B	16B	32B	64B	128B	256B	512B	1024B	2048B	4096B
case1(01000001)	1.16	1.19	1.25	1.36	1.56	1.53	1.28	1.14	1.07	1.04	1.02
case2(00100001)	1.13	1.15	1.20	1.29	1.45	1.39	1.14	1.00	1.00	1.00	1.00
case3(00010001)	1.09	1.11	1.15	1.22	1.34	1.25	1.00	1.00	1.00	1.00	1.00
case4(00001001)	1.06	1.08	1.10	1.15	1.22	1.10	1.00	1.00	1.00	1.00	1.00
case5(00000101)	1.03	1.04	1.05	1.07	1.11	1.00	1.00	1.00	1.00	1.00	1.00
case6(00000011)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
case7(00000000)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

표 2. 8 노드에서의 실험결과 (2048 byte 의 간섭하는 점대점 통신)
Table 2. The simulation results for 8 nodes (2048 bytes size of interfering point-to-point communication)

	4B	8B	16B	32B	64B	128B	256B	512B	1024B	2048B	4096B
case1(01000001)	1.04	1.05	1.07	1.11	1.17	1.30	1.52	1.56	1.28	1.14	1.07
case2(00100001)	1.03	1.04	1.06	1.08	1.14	1.24	1.41	1.42	1.14	1.00	1.00
case3(00010001)	1.03	1.03	1.04	1.06	1.10	1.18	1.31	1.27	1.00	1.00	1.00
case4(00001001)	1.02	1.02	1.03	1.04	1.07	1.12	1.21	1.13	1.00	1.00	1.00
case5(00000101)	1.01	1.01	1.01	1.02	1.03	1.06	1.10	1.00	1.00	1.00	1.00
case6(00000011)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
case7(00000000)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

확인할 수 있다. 따라서 기존 구조에서는 case 1에서 가장 심한 성능하락으로 인해, 가장 긴 전송 시간을 소모한다. 따라서 식 (1)으로 case 1에서 가장 높은 성능 향상을 볼 수 있다.

그림 9, 10과 표 3, 4는 16개의 프로세싱 노드를 사용하여 브로드캐스트 통신의 성능을 평가한 것이다. 그림 9는 간섭 통신의 데이터 크기가 512Byte 인 경우고, 그림 10는 간섭 통신의 데이터 크기가 2048Byte 일 경우 성능 평가한 것이다. 16 개의 노드로 구성된 시스템 에서도 case 1의 경우가 위와 같은 이유로 성능이 가장 향상된 것을 볼 수 있다.

그림 7, 8, 9, 10에서 최대 성능을 보여준 것의 수행 사이클을 추출한 것이 표 5이다. 이전의 모델은 간섭하는 점대점 통신에 case에 따라 수행 사이클이 달라지는 반면, 제안하는 모델의 간섭하는 점대점 통신 노드는 브로드캐스트 통신 순서에서 가장 마지막이 되기 때문에 case 에 상관없이 수행 사이클이 일정한 것을 확인할 수 있다. 실험 결과 8 개의 노드로 구성된 시스템에서는 최대 56%, 16개의 노드의 경우 최대 78%의 성능향상이 있다.

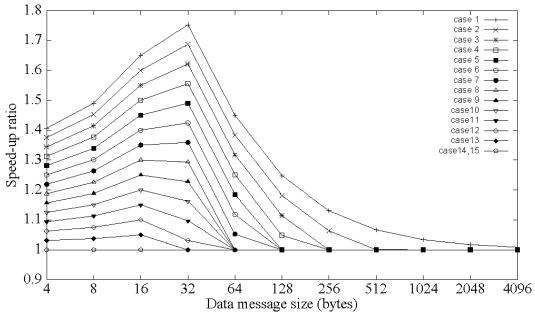


그림 9. 16 노드에서의 실험결과 (512 byte 의 간섭하는 점대점 통신)
Fig. 9. The simulation results for 16 nodes (512 bytes size of interfering point-to-point communication)

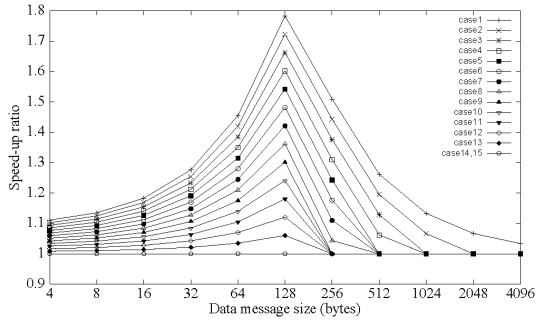


그림 10. 16 노드에서의 실험결과 (2048 byte 의 간섭하는 점대점 통신)
Fig. 10. The simulation results for 16 nodes (2048 bytes size of interfering point-to-point communication)

Ⅳ. 결 론

MPSoC는 처리 속도의 향상, 설계의 유연성, 저전력 소비, 설계시간 단축 등 많은 이점이 있어 임베디

표 3. 16 노드에서의 실험결과 (512byte 의 간섭하는 점대점 통신)
Table 3. The simulation results for 16 nodes (512 bytes size of interfering point-to-point communication)

	4B	8B	16B	32B	64B	128B	256B	512B	1024B	2048B	4096B
case1 (0100000000000001)	1.41	1.49	1.65	1.75	1.45	1.25	1.13	1.07	1.03	1.02	1.01
case2 (0010000000000001)	1.38	1.45	1.60	1.69	1.38	1.18	1.06	1.00	1.00	1.00	1.00
case3 (0001000000000001)	1.34	1.41	1.55	1.62	1.32	1.11	1.00	1.00	1.00	1.00	1.00
case4 (0000100000000001)	1.31	1.38	1.50	1.56	1.25	1.05	1.00	1.00	1.00	1.00	1.00
case5 (0000010000000001)	1.28	1.34	1.45	1.49	1.18	1.00	1.00	1.00	1.00	1.00	1.00
case6 (0000001000000001)	1.25	1.30	1.40	1.42	1.12	1.00	1.00	1.00	1.00	1.00	1.00
case7 (0000000100000001)	1.22	1.26	1.35	1.36	1.05	1.00	1.00	1.00	1.00	1.00	1.00
case8 (0000000010000001)	1.19	1.23	1.30	1.29	1.00	1.00	1.00	1.00	1.00	1.00	1.00
case9 (0000000001000001)	1.16	1.19	1.25	1.23	1.00	1.00	1.00	1.00	1.00	1.00	1.00
case10(0000000000100001)	1.13	1.15	1.20	1.16	1.00	1.00	1.00	1.00	1.00	1.00	1.00
case11(0000000000010001)	1.09	1.11	1.15	1.10	1.00	1.00	1.00	1.00	1.00	1.00	1.00
case12(0000000000001001)	1.06	1.08	1.10	1.03	1.00	1.00	1.00	1.00	1.00	1.00	1.00
case13(0000000000000101)	1.03	1.04	1.05	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
case14(0000000000000011)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
case15(0000000000000000)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

표 4. 16 노드에서의 실험결과 (2048byte 의 간섭하는 점대점 통신)
Table 4. The simulation results for 16 nodes (2048 bytes size of interfering point-to-point communication)

	4B	8B	16B	32B	64B	128B	256B	512B	1024B	2048B	4096B
case1 (0100000000000001)	1.11	1.13	1.18	1.28	1.45	1.78	1.51	1.26	1.13	1.07	1.03
case2 (0010000000000001)	1.10	1.12	1.17	1.25	1.42	1.72	1.44	1.19	1.07	1.00	1.00
case3 (0001000000000001)	1.09	1.11	1.15	1.23	1.38	1.66	1.38	1.13	1.00	1.00	1.00
case4 (0000100000000001)	1.09	1.10	1.14	1.21	1.35	1.60	1.31	1.06	1.00	1.00	1.00
case5 (0000010000000001)	1.08	1.09	1.13	1.19	1.31	1.54	1.24	1.00	1.00	1.00	1.00
case6 (0000001000000001)	1.07	1.08	1.11	1.17	1.28	1.48	1.18	1.00	1.00	1.00	1.00
case7 (0000000100000001)	1.06	1.07	1.10	1.15	1.24	1.42	1.11	1.00	1.00	1.00	1.00
case8 (0000000010000001)	1.05	1.06	1.08	1.13	1.21	1.36	1.04	1.00	1.00	1.00	1.00
case9 (0000000001000001)	1.04	1.05	1.07	1.11	1.17	1.30	1.00	1.00	1.00	1.00	1.00
case10(0000000000100001)	1.03	1.04	1.06	1.08	1.14	1.24	1.00	1.00	1.00	1.00	1.00
case11(0000000000010001)	1.03	1.03	1.04	1.06	1.10	1.18	1.00	1.00	1.00	1.00	1.00
case12(0000000000001001)	1.02	1.02	1.03	1.04	1.07	1.12	1.00	1.00	1.00	1.00	1.00
case13(0000000000000101)	1.01	1.01	1.01	1.02	1.03	1.06	1.00	1.00	1.00	1.00	1.00
case14(0000000000000011)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
case15(0000000000000000)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

표 3. 최대 성능 시의 실행 사이클
Table 3. Execution cycle of maximum performance

(a) 8 노드 : 512 byte 크기의 간섭하는 점대점 통신, 64 byte 크기의 브로드캐스트 통신
(a) 8 nodes : 512 bytes size for interfering point-to-point communication, 64 bytes size for broadcast communication

	case1	case2	case3	case4	case5	case6	case7
general	278	543	504	465	426	387	348
proposed	278	348	348	348	348	348	348

(b) 8 노드 : 2048 byte 크기의 간섭하는 점대점 통신, 256 byte 크기의 브로드캐스트 통신
(b) 8 nodes : 2048 bytes size for interfering point-to-point communication, 256 bytes size for broadcast communication

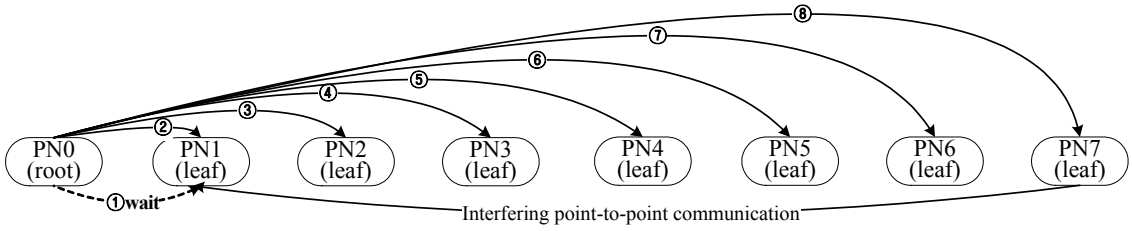
	case1	case2	case3	case4	case5	case6	case7
general	950	1983	1848	1713	1578	1443	1308
proposed	950	1308	1308	1308	1308	1308	1308

(c) 16 노드 : 512 byte 크기의 간섭하는 점대점 통신, 32 byte 크기의 브로드캐스트 통신
(c) 16 nodes : 512 bytes size for interfering point-to-point communication, 32 bytes size for broadcast communication

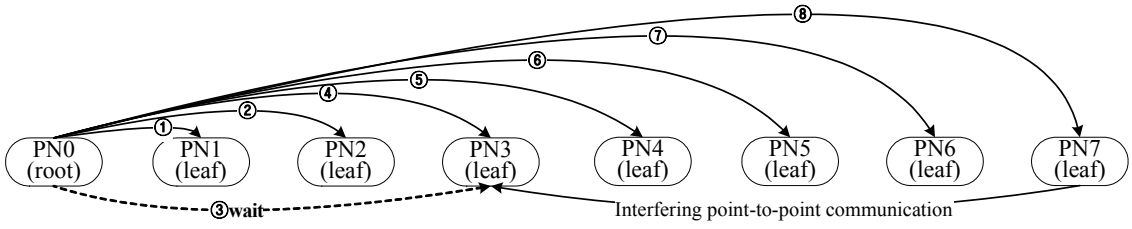
	case1	case2	case3	case4	case5	case6	case7	case8	case9	case10	case11	case12	case13	case14	case15
general	1070	2103	2032	1961	1890	1819	1748	1677	1606	1535	1464	1393	1322	1251	1180
proposed	1070	1180	1180	1180	1180	1180	1180	1180	1180	1180	1180	1180	1180	1180	1180

(d) 16 노드 : 2048 byte 크기의 간섭하는 점대점 통신, 256 byte 크기의 브로드캐스트 통신
(d) 16 nodes : 2048 bytes size for interfering point-to-point communication, 256 bytes size for broadcast communication

	case1	case2	case3	case4	case5	case6	case7	case8	case9	case10	case11	case12	case13	case14	case15
general	351	615	592	569	546	523	500	477	454	431	408	385	362	351	351
proposed	351	351	351	351	351	351	351	351	351	351	351	351	351	351	351



(a) case 1



(b) case 3

그림 11. 8노드에서 기존 구조의 브로드캐스트 통신 순서
Fig. 11. The order of broadcast communication in the conventional architecture with 8 nodes

드 시스템에서 최근 연구가 활발하다. 애플리케이션의 증가와 그에 따른 연산량의 증가로 프로세서의 수는 계속적으로 증가되는 추세지만, 태스크 분할 문제, 통신 오버헤드로 인한 병목현상 등으로 인해 선형적으로 성능향상이 되지 않는다. 따라서 본 논문에선 멀티프로세서 시스템에서 브로드캐스트 통신의 전송시간을 줄이기 위해, 각 노드의 포트의 상황을 고려한 MPI 유닛을 제안하고 설계하였다. MPI 유닛 내부의 메시지 프로세싱 유닛에서는 마스크비트가 추가되었고, 이를 사용하여 브로드캐스트 통신의 흐름을 관리함으로 메시지 전달 효율을 높인다. 본 논문에서 제안하고 설계한 MPI 유닛을 사용하면 다른 통신의 간섭 중에 브로드캐스트 통신을 할 때 손실 없이 전송 대역폭을 극대화 할 수 있기에 분산 메모리 아키텍처를 사용하는 멀티프로세서 시스템에서 매우 효과적이다.

참 고 문 헌

〔1〕 A. C. Klaiber, H. M. Levy, “A comparison of message passing and shared memory architectures for data parallel programs,” Proceedings of the 21st annual international symposium on Computer architecture, Vol 22, pp 94-105, April 1994

〔2〕 P. Stenstrom, “A Survey of Cache Coherence Schemes for Multiprocessors,” Computer, Vol. 23, pp. 12-24, June 1990.

〔3〕 L. Benini and G.de Micheli, “ Networks On Chip: A New SoC Paradigm,” IEEE Computer, Vol 35, No. 1, Jan. 2002, pp. 70-78.

〔4〕 Daniel L. Ly, Manuel Saldana, Paul Chow, “the Challenges of Using An Embedded MPI for Hardware-based Processing Nodes,” Field-Programmable Technology(FPT) 2009, Sydney, NSW, Dec. 2009, pp. 120-127.

〔5〕 T. P. McMahon and A. Skjellum, “eMPI/eMPICH: Embedding MPI,” MPI Developers Conference, 1996, pp. 180-184.

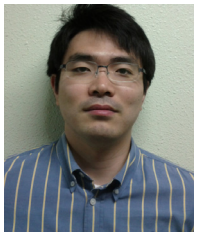
〔6〕 R. Rabenseifner, “Automatic MPI counter profiling of all users: First results on a CRAY T3E 900-512,” Proceedings of the Message Passing Interface Developer’s and User’s Conference 1999(MPIDC99), 1999, pp.77-85.

〔7〕 S. S. Vadhiyar, G. E. Fagg, and J. Dongarra. “Automatically Tuned Collective Communications,” In Proceedings of SC’00: High Performance Networking and Computing, 2000.

〔8〕 Mike Barnett, Satya Gupta, David G. Payne, Lance Shuler, and Robert van de Geijn, “Building a High-Performance Collective Communication Library,” Supercomputing’94, Nov. 1994, pp. 107-116.

〔9〕 Thakur, Rajeev, et al., "Optimization of collective communication operations in mpich," International Journal of High Performance Computing Applications, Feb. 2005, pp. 49 - 66.
〔10〕 Poletti Francesco, Poggiali Antonio, and Paul Marchal, "Flexible hardware/software support for message passing on a distributed shared memory architecture," Design, Automation and Test in Europe 2005, March 2005, Vol. 2, pp. 736-741.
〔11〕 정하영, 정원영, 이용석, "MPSoC를 위한 저비용 하드웨어 MPI 유닛 설계," 한국통신학회지, 제36권, 제1호, pp. 86-92, 2011.

윤 희 준 (Heejun Yun) 준회원



2010년 2월 숭실대학교 정보통신전자공학부 학사 졸업
2010년 3월~현재 연세대학교 전기전자공학과 석사과정
<관심분야> 컴퓨터 아키텍처, ASIC, SoC

정 원 영 (Wonyoung Chung) 정회원

2005년 8월 연세대학교 전기전자공학과 학사 졸업
2005년 9월~현재 연세대학교 전기전자공학과 석박사 통합과정
<관심분야> 네트워크 프로세서, 컴퓨터 아키텍처, 메모리 구조

이 용 석 (Yong-surk Lee) 정회원

1973년 2월 연세대학교 전기공학과 졸업
1977년 2월 Michigan 반도체 설계 공학석사
1981년 3월 Michigan 반도체 설계 Ph.D
1993년 2월~현재 연세대학교 전기전자공학과 교수
<관심분야> 마이크로프로세서 설계, VLSI 설계, DSP 프로세서 설계, 고성능 연산기 설계