MPSoC를 위한 저비용 하드웨어 MPI 유닛 설계

정회원 정 하 영*, 정 원 영*, 이 용 석**

The Design of Hardware MPI Units for MPSoC

Hayoung Jeong*, Wonyoung Chung*, Yong-surk Lee** *Regular Members*

요 약

본 논문에선 분산 메모리 아키텍처를 사용하는 멀티프로세서 시스템에서 메시지 전달을 지원하는 하드웨어 MPI(Message Passing Interface) 유닛을 설계하였다. 데이터 전송 동기화 및 데이터 전송, 완료까지의 과정을 하드웨어 MPI 유닛이 담당하여 동기화에 따른 오베헤드를 경감시켰다. 또한 동기화 메시지를 저장 관리하는 요청 큐(Request Queue), 준비 큐(Ready Queue), 예약 큐(Reserve Queue)를 내장하여 병렬적으로 입력받은 동기화 메시지를 관리하고 비순차적 종료(out of order completion)을 지원한다. BMF(Bus Functional Medel)을 제작해 제안한 구조에서의 전송 대역폭 성능을 확인한 결과 다대다 통신에서 25%이상의 성능 향상이 있음을 확인할 수 있었다. 그후 HDL로 기술된 하드웨어를 Magnachip 0.18 공정 라이브러리에서 합성하였으며 프로토 타입 chip으로 제작하였다. 제안한 MPI 유닛은 전체 칩 사이즈의 1%이하의 크기로 높은 성능 향상을 기대할 수 있어, 저비용 설계와 확장성 측면에서 임베디드 MPSoC(Multi-Processor System-on-Chip)의 전체적인 성능을 높이는데 유용하다.

Key Words : MPSoC, Message passing, Distributed memory, Low-cost

ABSTRACT

In this paper, we propose a novel hardware MPI(Message Passing Interface) unit which supports message passing in multiprocessor system which use distributed memory architecture. MPI Hardware unit processes data synchronization, transmission and completion, and it supports processor non-blocking operation so it reduces overhead according to synchronization. Additionally, MPI hardware unit combines ready entry, request entry, reserve entry which save and manage the synchronized messages and performs the multiple outstanding issue and out of order completion. According to BFM(Bus Functional Model) simulation result, the performance is increased by 25% on many to many communication. After we designed MPI unit using HDL, with synopsys design compiler we synthesized, and for synthesis library we used MagnaChip 0.18㎛. And then we making prototype chip. The proposed message transmission interface hardware shows high performance for its increase in size. Thus, as we consider low-cost design and scalability, MPI hardware unit is useful in increasing overall performance of embedded MPSoC(Multi-Processor System-on-Chip).

Ⅰ. 서 론

최근 임베디드 시스템에서도 다양한 어플리케이션의 사용이 늘어남에 따라 그 연산의 복잡도가 증가하고 있다. 이에 따라 고성능 프로세서 시스템에 대한 요구가 높아지고 있지만 단일 프로세서의 동작 주파수를 높이는 방법으로는 어플리케이션의 발전 속도를 따라가는데 한계가 있다. 하나의 칩 안에 다수의 프로

※ 본 연구는 지식경제부 출연금으로 ETRI 시스템반도체진흥센터에서 수행한 시스템반도체 융복합형설계인력양성사업의 연구결과입니다.
* 연세대학교 전기전자공학과 프로세서 연구실(hyjeong@mpu.yonsei.ac.kr, wychung@mpu.yonsei.ac.kr, yonglee@yonsei.ac.kr)
논문번호 : KICS2010-09-453, 접수일자 : 2010년 9월 16일, 최종논문접수일자 : 2010년 12월 30일

세서와 메모리를 집적시키는 MPSoC (Multiprocessor System on a Chip)는 단일 프로세서가 처리하는 여러 개의 태스크(task)를 여러 프로세서로 분산시킴으로써 시스템 전체의 성능을 향상 시킬 수 있다. 또한 전력 소모가 적기 때문에 임베디드 시스템(embedded system) 분야에서 연구가 활발히 진행되고 있다. 예로는 ARM cortex-A9, Texas Instruments사의 OMAP, STI(Sony/ Toshiba/ IBM)의 Cell 프로세서 등을 들 수 있다.

병렬 프로세서 간 데이터를 공유하는 방법은 크게 공유 메모리 방식과 분산 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, 캐쉬 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 줘야 하며, 데이터 전송 시 교착상태(deadlock)가 발생할 수 있다는 단점이 있다. 하지만 프로세서의 개수가 늘어날수록 소비전력 및 수행 시간 등 데이터 전송 오버헤드가 작아지는 장점을 갖는다[3]. 최근 프로세서의 개수가 점차적으로 많아지고 있기 때문에 프로세서의 개수가 많을 때 장점을 갖는 분산 메모리 구조에 대한 연구가 주목 받고 있다.

최근 분산 메모리 구조의 시스템은 표준 MPI (message passing interface)를 최적화하여 통신 성능을 향상시키는데 초점이 맞추어지고 있으며, 특히 MPI를 하드웨어적으로 처리하는 연구가 활발하다. 또한 네트워크 인터페이스를 통해 하드웨어 간 통신을 하며, 큐 내의 태스크로 메시지를 관리하여 통신 성능을 향상시키는 연구도 진행 중이다[4]. 본 논문에서는 분산 메모리 구조에서 프로세서가 직접 접근, 제어할 수 있는 메시지 전송 인터페이스 알고리즘 및 저비용 하드웨어 MPI 구조를 제안하고 설계하였다. 제안하는 MPI 유닛을 통해 동시성을 관리할 수 있으며, 소프트웨어 MPI 라이브러리 실행의 무거움을 줄였다.

메시지 전달 방식은 크게 동기(Synchronous) 송신, 준비(ready) 송신, 버퍼(buffer) 송신, 표준(Standard) 송신 방식이 있는데, 이중 동기송신은 수신측에 데이터를 저장할 메시지 버퍼의 크기가 충분하며, 송신측이 송신할 데이터가 준비되었을 때 전송이 진행된다. 반면, 준비송신이나 버퍼송신은 수신 노드의 상태가 항상 데이터를 받을 준비가 되어있다는 가정 하에 데이터 전송이 이루어진다. 따라서 동기송신은 데이터 손실에 대한 위험이 없고 가장 안전한 방식이라 할 수 있다. 하지만 동기송신의 경우 송신측에서 데이터 전송을 위해 MPI_SSEND()함수 호출 시부터 데이터 전송이 완료되는 시점까지 기다려야 하는 블록킹(blocking) 방식이라는 단점이 있다. 이러한 전송의 약점을 보완하기 위해 DMA(Direct Memory Access)을 이용하여 그 효율성을 높이고자 하는 연구가 이루어지고 있다[5]. 하지만 DMA 방식으로 데이터 전송의 오버헤드는 경감시킬 수 있으나 동기화를 위해 서로의 송수신 상태 및 메시지 버퍼의 여유를 확인하는 작업을 여전히 프로세서가 담당하고 있어 동기화에 따른 오버헤드가 여전히 존재한다는 단점도 있다. 따라서 본 논문에서는 이러한 단점을 개선하고자 데이터 전송뿐만 아니라 동기화 과정까지 담당하는 MPI 하드웨어 구조를 제안한다.

본 논문에서는 제안하는 MPI 유닛은 분산 메모리 구조에서 병렬 프로그래밍을 지원하기 위한 하드웨어 플랫폼의 아키텍처를 기반으로 한다. MPI 유닛은 표준 MPI를 지원하며, 메시지 전달을 통한 데이터 공유를 지원한다. 제안하는 MPI 유닛의 주요 장점으로는 다음과 같다.

1) 데이터 전송, 완료뿐만 아니라 동기화 과정까지 프로세서가 아닌 MPI 유닛이 담당하여 하드웨어적으로 처리하므로 전체적인 송수신 시간을 단축시킨다.
2) MPI 유닛 내부에 준비 큐, 요청 큐, 예약 큐 및 제어 모듈이 내장되어 있어 동기화 메시지를 저장 및 관리한다. 이러한 모듈로 인하여 병렬적으로 입력되는 동기화 메시지 중 처리되지 않은 동기화 메시지를 이슈(multiple outstanding issue)하고 비순차적 종료(out-of-order comple -tion)를 지원하여 프로세서 간 복잡한 데이터 의존성에 의해 발생하는 대기시간을 줄여 결과적으로 메시지 전달에 있어서 지연시간을 최소화하였다. 특히 예약 큐로 인해 펜딩(pending) 기능이 추가되어 프로세서 노드 간의 불필요한 트래픽 교환을 최소화 한다.
3) 프로세서 노드들 간에 과도한 송수신으로 인한 병목현상을 방지하고 전송 대역폭을 극대화하기 위해 crossbar 형태인 ARM사의 AMBA 3.0 AXI 프로토콜을 사용하였다[6].

Ⅱ. 제안하는 MPI 유닛 구조

2.1 전체 시스템 구조

그림 1은 MPI 유닛이 연결된 전체적인 시스템을

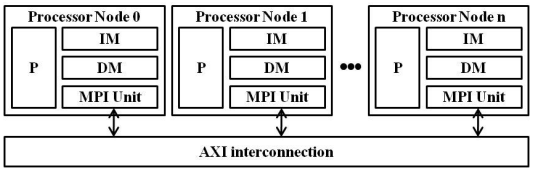

그림 1. 전체 시스템 구조
Fig 1. The structure of whole system

나타낸다. 하나의 프로세서 노드는 하나의 프로세서, 데이터 메모리, 명령어 메모리 그리고 MPI 유닛으로 구성되어 있다. MPI 유닛은 프로세서와 데이터 메모리에 직접 연결되어 있다. MPI 유닛은 프로세서로부터 메시지 전송을 위한 제어 메시지(control message)를 입력받는다. MPI 유닛은 전송 동기화에서부터 데이터 전송, 완료까지 담당한다. 많은 개수의 프로세서가 연결될 경우 공유된 버스에서 프로세서 노드들 간의 과도한 송수신으로 인하여 병목현상이 일어날 수 있다. 따라서 이러한 병목현상을 방지하고 전송 대역폭을 최대화하기 위해 크로스바(crossbar) 형태인 ARM사의 AMBA 3.0 AXT 프로토콜을 사용하였다.

2.2 제어 메시지

다른 프로세서 메모리에 접근이 이루어져야 할 경우 프로세서는 MPI 유닛으로 명령어를 보내게 되며, MPI 유닛은 자신의 상태를 체크한 후 자신과 통신할 프로세서 노드의 MPI 유닛에게 제어 메시지를 보낸다. 제어 메시지는 그림 2와 같이 송신 프로세서 랭크(Send processor rank), 수신 프로세서 랭크(Receive processor rank), 순서 ID(Sequence ID), 명령 비트(Command bit), 전체 데이터의 크기(Data size), 데이터 메모리의 시작 주소(Start address)에 대한 정보가 담겨있다. 또한 송신 프로세서 랭크, 수신 프로세서 랭크, 순서 ID를 나타내는 정보는 동기화를 위한 매치 비트(Match bits)로 사용된다. 명령 비트에 따른 제어 메시지의 기능은 표 1과 같다.

표 1. 제어 메시지의 기능
Table 1. The fuction of control message

제어메시지	기능
m_request	송신측 MPI 유닛에 데이터를 요청
m_ready	송신한 데이터가 메시지 버퍼에 준비되었음을 수신측 MPI 유닛에 알려줌
m_accept	받은 request 요청에 데이터가 준비되었음을 수신측으로 응답
m_busy	받은 request 요청에 데이터가 준비되지 않았음을 수신측으로 응답
m_pend	받은 request 요청에 아직 데이터가 준비되지 않았지만 request 요청을 수납하였음을 응답
m_complete	전송이 완료되었음을 수신측에 알려줌
m_data_on	동기화 과정이 끝나고 메모리로부터 데이터를 실어서 보냄을 알려줌

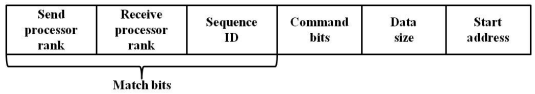

그림 2. 제어 메시지 구조
Fig 2. The structure of control message

2.3 메시지 전송 메커니즘

그림 3은 제안하는 하드웨어 MPI 유닛의 메시지 전송 메커니즘을 나타낸다. 1번 프로세서 노드에서 값을 읽어 0번 프로세서 노드에 저장하는 예이다. 0번 프로세서(P0)는 MPI Unit 0에게 Receive 명령어(Request에 해당)를 보내고, 1번 프로세서(P1)는 MPI Unit 1에게 Send 명령어(Ready에 해당)를 보낸다. MPI Unit 0의 Processor Wrapper(PW)는 이를 요청 큐에 저장하고, MPI Unit 1의 Processor Wrapper (PW)는 이를 준비 큐에 저장한다. Request Sender (RS)에선 요청 큐에 저장된 메시지를 Command Message Sender(CMS)로 보낸다. CMS에 입력된 메시지는 통신할 프로세서 노드에 해당하는 채널을 통해 통신할 프로세서 노드의 MPI Unit에 전송된다. 프로세서 노드 1의 Command Message Receiver(CMR)로 입력된 m_request는 매치 비트를 비교하기 위해 Message Passing Unit(MPU)로 보내진다. 현재 MPI Unit 의 준비 큐에 매치 비트가 같은 엔트리가 존재하기 때문에 CMS를 통해 MPI Unit 0로 m_accept 제어 메시지를 보내는 동시에 MPI Unit 1에서는 메모리로부터 데이터를 준비하게 된다. Data Sand Scheduler (DSS)에선 라운드 로빈 방식으로 보낼 데이터 선택하고 Memory Access Scheduler(MAS)를 통해 메모리로부터 값을 읽는다. Data Message Sender (DMS)를

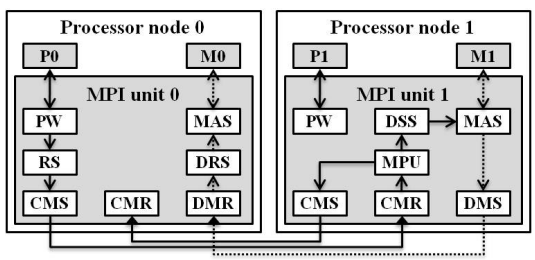

그림 3. 데이터 전송 메카니즘
Fig 3. The mechanism of data transfer

통해 데이터 메시지가 보내질 때 제어 메시지는 m_data_on이며, Data Message Receiver(DMR)를 통해 해당 채널로 받은 데이터는 Data Receive Scheduler(DRS)를 거쳐 선택되어진 데이터가 MAS를 통해 메모리에 저장된다. 마지막으로 Data 전송이 완료되면 m_complete 제어 메시지를 MPI Unit 1에게 보냄으로써 통신이 완료된다.

2.4 하드웨어 구조

그림 4는 하드웨어 MPI 유닛의 구조를 나타내고 있다. 본 연구에서 설계한 하드웨어 MPI 유닛은 3가지의 큐 즉, 요청 큐(Request Queue), 준비 큐(Ready Queue), 예약 큐(Reserve Queue)를 사용하여 메시지를 관리한다.

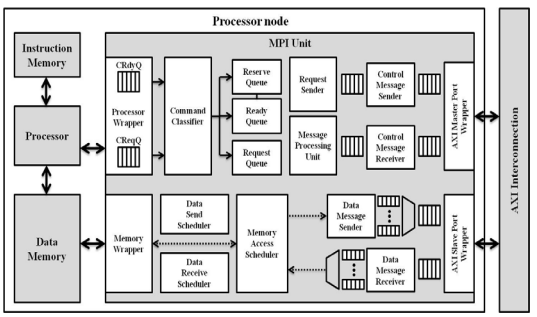

그림 4. MPI 하드웨어 구조
Fig 4. MPI hardware structure

2.4.1 요청 큐

전송 요청 명령어는 valid bit가 off인 라인에 저장하며, 저장 후 valid bit은 on 시킨다. 그 후 valid bit은 on이고 아직 issue 되지 못한 라인이 control message sender로 제어 메시지를 전송한다. 송신할 프로세서 노드의 준비가 늦춰짐에 따라 특정 제어 메시지가 자원을 독점하여 정체 현상이 일어날 수 있다. 이를 방지하기 위해 요청 큐에서는 이들 요청을 스케줄링 할 필요가 있으며, 본 논문에선 라운드 로빈(Round Robin)을 선택하여 설계하였다.

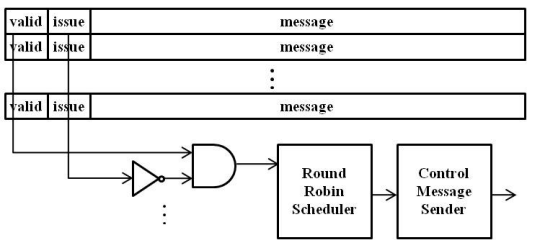

그림 5. 요청 큐
Fig 5. Request queue

2.4.2 준비 큐

준비 큐는 프로세서로부터 입력된 송신 명령어 중에서 전송할 준비가 된 명령어들을 저장하는 장소이다. 명령어가 준비 큐로 입력되면, valid bit가 off인 라인의 매치 비트열과 명령(command) 비트열에 나누어 저장되며, valid bit은 on 된다. 수신할 프로세서 노드의 MPI unit으로 부터 요청 제어 메시지가 입력되면 요청 제어 메시지 중 배치 비트열이 준비 큐로 접근하게 된다. 이때 control message receiver 로부터 동기화 요청 메시지를 받으면 준비 큐의 저장된 모든 라인과 동시 비교를 하게 된다. 일치하는 데이터가 있을 경우 FIFS(First In First Service) 스케줄링을 통해 data transfer scheduler 모듈로 송신 데이터 정보를 입력하게 된다. 일치하는 데이터가 존재하지 않을 경우에는 입력된 동기화 요청 메시지는 예약 큐로 보내지게 된다.

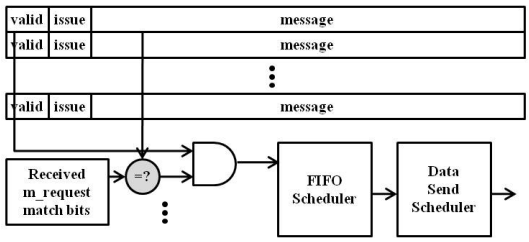

그림 6. 준비 큐
Fig 6. Ready queue

2.4.3 예약 큐

예약 큐는 아직 준비되지 못한 데이터 전송에 대한 동기화 요청 메시지를 저장하여 또 다른 동기화 요청 메시지에 따른 트래픽을 줄이기 위한 모듈이다. 송신할 데이터가 준비되면 received ready message match bit를 통해 예약 큐 모듈로 매치 비트열이 입력되고, 모든 라인에 일치하는 데이터를 찾아 그 일치 여부를 준비 큐에 request deposit 신호를 전달하게 된다. 또한 control message sender 모듈을 통해 m_reply_

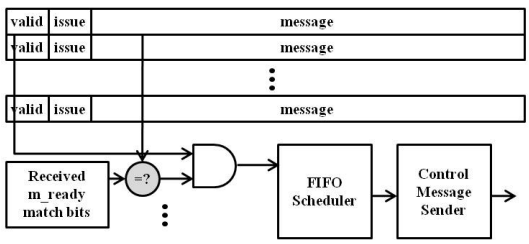

그림 7. 예약 큐
Fig 7. Reserve queue

grant 동기화 메시지 전송을 요청하여 수신 프로세서 노드의 MPI 유닛에게 전달하게 된다.

Ⅲ. BFM 시뮬레이션 및 결과

제안하는 MPI 유닛의 데이터 통신 대역폭을 측정하기 위해 System C를 사용하여 BFM(Bus Functional Model)을 설계하였다. BMF는 각 블록의 지연시간을 고려하여 동작을 기술하였고, 특정 시뮬레이션 환경에 따라 통신 트래픽을 생성할 수 있다. 그 후 생성된 통신 트래픽으로 제안한 MPI 유닛의 전송 성능을 확인하였다. 데이터 전송 성능에 영향을 미치는 포트 버터 크기는 전송 word 단위인 32비트 width에 depth 5로 제한하였다. 시뮬레이션은 일 대 일, 다 대 일, 다 대 다 통신환경에서 실험을 하였다. 또한 각 통신 환경에서 데이터 메시지의 사이즈, 코어의 개수, 엔트리의 개수의 변화를 통하여 대역폭을 측정하였다.

그림 8은 MPI 유닛의 각 큐의 엔트리 개수가 2일 때 1 대 1 통신에서의 시뮬레이션 결과이다. 엔트리를 2개로 하였기 때문에 채널 동기화에 따른 수행 시간 오버헤드가 데이터 메시지 전송에 의해 가려지는 latency- hiding 효과를 확인할 수 있었다. 데이터 메시지 크기가 1024바이트에 이르러서는 전송 대역폭이 임계경로 메모리 대역폭인 500MB/s에 근접함을 확인하였다.

그림 9는 4개의 프로세서 노드에서 하나의 프로세서 노드로 데이터 메시지를 전송하는 상황에서 데이터 메시지 크기와 엔트리의 개수를 변화해 가면서 시뮬레이션 한 결과이다. 하나의 프로세서 노드에 4개의 프로세서 노드가 순차적으로 연결되어 데이터를 전송하기 때문에 엔트리 개수가 2 초과 될 때부터는 성능 향상의 폭이 작았다. 그림 10와 11은 각각 4 대 4, 8 대 8 통신에서의 시뮬레이션 결과이다. 다 대 다 통신의 경우 프로세서 노드 간 데이터 의존성이 복잡해지

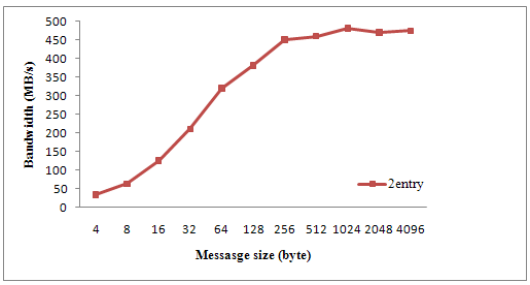

그림 8. 1 대 1 통신
Fig 8. 1 to 1 communication

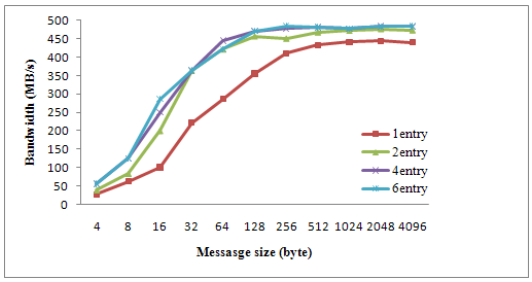

그림 9. 1 대 4 통신
Fig 9. 1 to 4 communication

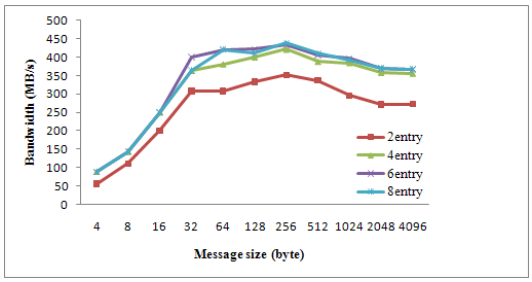

그림 10. 4 대 4 통신
Fig 10. 4 to 4 communication

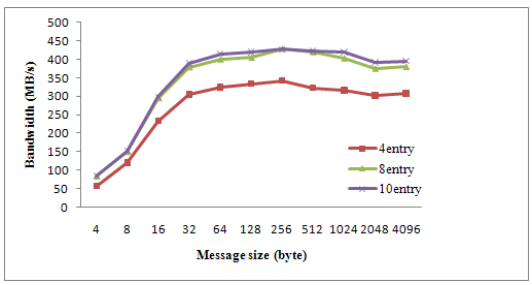

그림 11. 8 대 8 통신
Fig 11. 8 to 8 communication

고, 이로 인해 데이터 전송 대기시간이 증가하게 된다. 따라서 전체 시스템 측면에서 보았을 때, 메시지 전달 동기화로 인한 오버헤드가 증가하였기 때문에 1 대 1 통신에 비해서 20%~25%의 성능이 떨어진 결과를 확인할 수 있었다. 하지만 다 대 다 통신에서 요청 큐, 준비 큐, 예약 큐 엔트리의 개수를 증가시키면 각 프로세서 노드 간 데이터 의존성 때문에 생기는 대기시간이 줄어들기 때문에 전체 전송 대역폭이 향상된다. 예를 들어 4 대 4 통신에서는 엔트리의 개수가 2개일 때보다 4개일 때 약 20%의 대역폭 증가를 보였으며, 8 대 8 통신에서도 엔트리가 4개일 때 보다 8개일 때 약 28%의 대역폭이 증가하여 높은 성능 향상을 보였다. 4 대 4 통신, 8 대 8 통신에서도 엔트리가 각각 4,

8까지 올라갈 때는 대역폭이 증가하지만 그 이상이 되었을 때는 성능 향상의 폭이 작았다.

그림 12는 4:4, 8:8, 12:12 노드의 시뮬레이션 결과를 보여준다. 엔트리가 증가하면 대역폭이 증가하지만 프로세서 노드의 수를 초과하여 엔트리 수를 증가시키면 성능 향상의 폭이 작음을 확인할 수 있다. 따라서 설계할 시스템의 프로세서 노드의 개수에 비례하여 엔트리의 개수를 결정하면 비용과 성능 측면에서 최적화된 시스템을 설계할 수 있다.

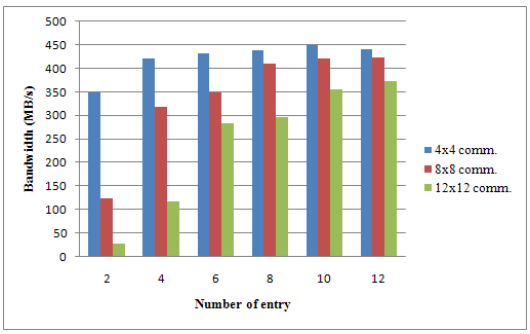

그림 12. 4:4, 8:8, 12:12 노드의 시뮬레이션 결과
Fig 12. The simulation results of 4:4, 8:8, 12:12 node

Ⅳ. 하드웨어 설계

BFM을 통해 성능을 검증한 후 Verilog HDL 언어를 이용하여 설계하였다. synopsys design compiler를 사용하여 합성하였으며, synthesis library는 MagnaChip 0.18㎛를 사용하였다. 제안하는 하드웨어 MPI 유닛은 3개의 큐의 엔트리 개수 변화에 따른 큐 크기와 데이터 채널 포트단의 버퍼 크기에 따라 면적이 크게 변화한다. 표 2는 포트 버퍼 크기를 32비트 width, depth 5로 고정하였을 경우 엔트리의 개수에 따른 면적 및 상대적 크기를 나타내고 있다.

설계한 MPI unit을 검증하기 위해 RISC MIPS DLX 아키텍처를 기본으로 하는 멀티프로세서를 설계하였다. 각 프로세서 노드는 하나의 코어와 16kb 메모리를 포함한 하나의 MPI 유닛으로 구성되어 있다. 그리고 AXI 버스를 통해 프로세서 노드간 통신을 하게

표 2. 합성 결과
Table 2. Synthesize results

엔트리 개수	면적 (2 input NAND gate)	상대적 크기
2	20724.82	1
4	21941.23	1.05
8	24394.56	1.17

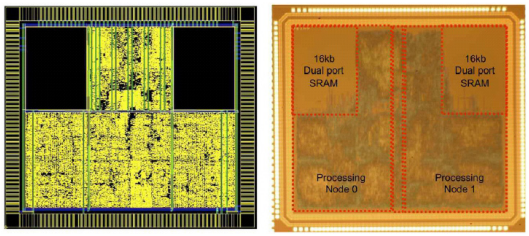

그림 13. 레이아웃 사진
Fig 13. Layout picture

된다. 그림 13은 4 * 4 mm size로 제작한 칩의 레이아웃(layout) 사진이다. 메모리가 대부분의 면적을 차지하는 반면, 프로세서 코어뿐만 아니라 MPI 유닛은 전체 면적에서 차지하는 비율이 작다. MPI 유닛은 칩 전체 면적에 1% 이하를 차지하였으며 이는 전체 칩 사이즈에 비해 무시해도(negligible) 좋을 만한 크기의 증가이다. 그러므로 본 논문에서 설계한 MPI 유닛은 작은 면적의 증가로 전체 시스템 성능을 높일 수 있다.

Ⅵ. 결 론

MPSoC는 처리 속도의 향상, 설계의 유연성, 저전력 소비, 설계시간 단축 등 많은 이점이 있어 임베디드 시스템에서 최근 연구가 활발하다. 어플리케이션의 증가와 그에 따른 연산량의 증가로 인해 프로세서의 개수는 계속적으로 증가되는 추세이지만, 태스크 분할 문제, 통신 오버헤드로 인한 bottleneck 문제 등의 원인으로 선형적으로 성능이 향상되고 있지 않다. 따라서 본 논문에선 멀티프로세서 시스템에서 통신 오버헤드로 인한 bottleneck을 줄이기 위해, 분산메모리 아키텍처에서 MPI 성능을 최적화한 하드웨어 MPI 유닛을 제안하고 설계하였다. MPI 유닛은 MPI library의 부담을 줄인다. MPI 유닛 내부에 3개의 큐(요청, 준비, 예약)을 두어 동기화 메시지를 저장 및 관리한다. 또한 논 블록킹 버스 시스템을 사용하며, 병렬적으로 입력되는 동기화 메시지 중 처리되지 않은 동기화 메시지를 이슈(multiple outstanding issue)하고 비순차적 종료(out-of-order completion)를 지원하여 프로세서 노드 사이의 메시지 전달 성능을 높인다. 본 논문에서 제안하고 설계한 MPI 유닛을 사용하면 작은 하드웨어 오버헤드로 메시지 지연시간을 최소화하고 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, pp.70-78, Jan. 2002

〔4〕 S. Han, A. Baghdadi, M. Bonaciu, S. Chae, and A. A. Jerraya, "An efficient scalable and flexible data transfer architecture for multiprocessor SoC with massive distributed memory," Proceedings of the 41st annual Design Automation Conference, San Diego, CA, USA, pp.250-255, June 2004.

〔5〕 P. Francesco, P. Marchal, D. Atienza, L. Benini, and F. Catthoor, "An integrated hardware/software approach for run-time scratchpad management," Proceedings of the 41st annual Design Automation Conference, San Diego, CA, USA, pp. 238-243, June 2004

〔6〕 AMBA AXI Specification, ARM Limited 2003.

정 하 영 (Hayoung Jeong) 정회원

2003년 2월 중앙대학교 전기전자공학부 학사 졸업

2005년 2월 연세대학교 전기전자공학과 석사 졸업

2005년 3월~현재 연세대학교 전기전자공학과 박사과정

<관심분야> 마이크로 프로세서, 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 전기전자공학과 학사 졸업

1977년 2월 University of Michigan Electrical Engineering 석사 졸업

1981년 2월 University of Michigan Electrical Engineering 박사 졸업

1993년 2월~현재 연세대학교 전기전자공학과 교수

<관심분야> 마이크로프로세서 설계, VLSI 설계, DSP 프로세서 설계, 고성능 연산기 설계