논문 2011-48SD-8-6

MPSoC 인터커넥션을 위한 AXI 하이브리드 온-칩 버스구조 설계

(A Design of AXI hybrid on-chip Bus Architecture for the Interconnection of MPSoC)

이 경 호*, 공 진 흥**

(Kyung-Ho Lee and Jin-Hyeung Kong)

요 약

본 연구에서는 AMBA 3.0 AXI 프로토콜을 사용하여 고성능 및 저전력이 요구되는 MPSoC에 적합한 하이브리드 온-칩 버스구조를 설계하였다. AXI의 채널 중에서 트래픽이 많은 쓰기데이터 채널 및 읽기데이터 채널은 Crossbar 버스구조로 설계하여 고속 처리를 가능하게 하였다. 또한 MPSoC에서의 컴포넌트 추가에 따른 오버헤드(회로크기, 연결회선, 전력소모 등)를 줄이기 위해 트래픽이 적은 주소 채널과 쓰기 응답 채널은 Shared 버스구조로 공유하도록 설계하였다. 본 연구에서는 Hybrid 버스구조의 검증을 위해 Shared 버스구조 및 Crossbar 버스구조와 함께 시간, 공간, 파워 영역에서 각각 비교 실험하였다. 16x16 버스 실험에서 Hybrid 버스구조는 Crossbar 버스구조와 비교해서 마스터의 대기시간은 약 9%, 전체 실행시간은 약 4%의 차이에 그쳐 비슷한 성능을 보여준다. 반면 오버헤드에서는 Crossbar 버스구조와 비교하여 회로 크기는 47%, 연결 회선 수는 52%, 동적 전력 소모는 66%의 감소 효과를 보인다. 따라서 본 연구에서 설계한 하이브리드 온-칩 버스구조는 Crossbar 버스구조와 비교하여 고성능 및 저전력이 요구되는 MPSoC 인터커넥션에 매우 효과적임을 보이고 있다.

Abstract

In this paper, we presents a hybrid on-chip bus architecture based on the AMBA 3.0 AXI protocol for MPSoC with high performance and low power. Among AXI channels, data channels with a lot of traffic are designed by crossbar-switch architecture for massively parallel processing. On the other hand, addressing and write-response channels having a few of traffic is handled by shared-bus architecture due to the overheads of (areas, interconnection wires and power consumption) reduction. In experiments, the comparisons are carried out in terms of time, space and power domains for the verification of proposed hybrid on-chip bus architecture. For 16x16 bus configuration, the hybrid on-chip bus architecture has almost similar performance in time domain with respect to crossbar on-chip bus architecture, as the masters's latency is differenced about 9% and the total execution time is only about 4%. Furthermore, the hybrid on-chip bus architecture is very effective on the overhead reduction, such as it reduced about 47% of areas, and about 52% of interconnection wires, as well as about 66% of dynamic power consumption. Thus, the presented hybrid on-chip bus architecture is shown to be very effective for the MPSoC interconnection design aiming at high performance and low power.

Keywords : AXI, MPSoC, Hybrid bus, Interconnect architecture, On-chip bus

* 학생회원, ** 평생회원, 광운대학교 컴퓨터공학과
(Department of Computer Engineering, Kwangwoon University)

※ "본 논문은 2009년도 광운대학교 교내학술연구비 지원에 의해 연구되었으며, 지식경제부가 지원하는 산업융합원천기술개발사업을 통해 개발된 결과임을 밝힙니다.(10039173, 융복합 혁신반도체 기술개발)."
접수일자: 2010년12월30일, 수정완료일: 2011년7월20일

Ⅰ. 서 론

과거 임베디드 시스템은 성능 향상을 위해서 싱글 프로세서 기반에서 동작 주파수를 높이는 방법을 사용하였다. 하지만 단순히 동작 주파수의 향상만으로는 성능의 한계와 함께 경제성 및 전력소모 등의 문제를 발생시켜, 임베디드 시스템의 성능 향상이 어려운 한계에 이르렀다. 이를 해결하는 방법으로 다수의 임베디드 프로세서를 내장시켜 시스템을 구성하는 MPSoC (Multi-Processor System-on-Chip)구조를 이용하게 되었다. MPSoC 임베디드 시스템은 고성능 싱글 프로세서 기반의 임베디드 시스템에 비해 공정 비용, 전력소모, 복잡한 어플리케이션의 고속처리 등에서 매우 우수함을 보였다. 따라서 임베디드 시스템은 고성능 싱글 프로세서 기반에서 멀티 프로세서 기반의 시스템으로 진화 발전하고 있다.[1~4]

임베디드 시스템에서 MPSoC는 칩 안의 임베디드 프로세서 코어간 통신이 급격하게 증가하는 문제점을 나타내고 있다. 이러한 멀티코어간 통신에 의한 온-칩 버스 시스템의 병목현상은 전체 MPSoC 시스템의 성능을 저하시키게 된다. 따라서 온-칩 버스 시스템은 MPSoC 임베디드 시스템의 설계에 있어서 성능을 결정하는데 중요한 역할을 하게 된다. 온-칩 버스 시스템은 다수의 임베디드 프로세서 코어간 통신의 효율을 높이기 위해서 짧은 대기시간을 요구한다. 또한 임베디드 프로세서 코어 추가에 따른 회로크기 및 복잡도 등에서 오버헤드가 매우 급격히 증가하기 때문에 MPSoC 시스템의 오버헤드를 최적화시킨 온-칩 버스 시스템 설계가 요구된다.[1~4]

멀티 코어간 통신을 효과적으로 하기 위한 온-칩 버스 시스템에서 회로크기 및 복잡도 등의 오버헤드는 기본적으로 프로토콜과 버스구조에 따라 결정된다. AXI[5]는 MPSoC를 포함한 임베디드 시스템 시장에서 70% 이상을 점유하고 있는 ARM사에서 개발한 차세대 고성능 온-칩 버스 프로토콜이다. 일반적으로 AXI와 같은 패킷기반 프로토콜은 복잡도가 매우 높고, 구현된 하드웨어 비용이 높다는 단점이 있다. 하지만 AXI는 표준 버스구조가 없고 목표로 하는 임베디드 시스템에 따라서 유연한 설계가 가능하다.[5] 본 연구에서는 MPSoC를 위한 AXI프로토콜에 대해서 효과적인 온-칩 버스구조를 설계하고자 한다.

현재 MPSoC의 높은 병렬 처리를 통한 고성능을 보장하기 위해 Crossbar 버스구조[6]가 많이 사용된다. AXI프로토콜에 관하여 ARM사는 Crossbar 버스구조의 상용 IP를 지원하고 있다. 하지만 AXI프로토콜을 사용하는 MPSoC에서의 Crossbar 버스구조는 멀티 코어 증가에 따른 연결 회선 수, 회로크기, 전력소모 등 오버헤드가 급격히 증가하는 문제점을 보이고 있다. 한편 AXI 프로토콜은 5개의 독립적인 채널로 구성되는데, 주소 채널과 쓰기응답 채널은 읽기/쓰기 트랜잭션 발생 시 최대 1개의 데이터 트래픽 전송이 이루어진다. 반면 읽기 및 쓰기 데이터 채널은 최대 16개의 데이터 트래픽 전송이 이루어진다.[5] 이 같은 AXI프로토콜의 다중채널 트래픽 특징을 이용하여 주소버스를 공유하는 AXI 크로스바 스위치의 면적 및 전력소모를 감소시키는 방법[10]이 제안되었다.

본 연구에서는 AXI프로토콜의 주소채널뿐만 아니라 데이터채널보다 트래픽이 작은 쓰기응답채널을 Shared 버스구조로 설계하여 Hybrid버스 아키텍처를 설계 및 구현하였다. AXI 다중채널 트래픽 특징을 이용하여 트래픽이 많은 데이터채널은 높은 병렬성의 Crossbar 버스구조로 설계해서 대부분의 트래픽을 고속처리하고, 트래픽이 적은 읽기/쓰기주소채널과 쓰기응답채널은 Shared 버스구조로 설계하였다. 또한 이같은 주소 및 쓰기응답채널 공유버스구조가 AXI 프로토콜 버스에서 마스터 및 슬레이브 컴포넌트 추가에 따른 면적 및 전력 증가의 문제를 해결하는데 얼마나 어떻게 효과적인지 실증하고자 한다.

실제로 Hybrid 버스구조는 쓰기/읽기 데이터 채널의 통신 시스템과 나머지 3개 채널들의 통신 시스템으로 구성된다. 대다수 트래픽의 고속처리를 위해 Crossbar 버스구조로 설계되는 쓰기/읽기 데이터 채널은 마스터와 슬레이브간 1:1 통신 접속을 유지하므로 각각 별도의 통신 시스템을 가져야 한다. 따라서 마스터의 개수가 m개이고 슬레이브 개수가 n개일 경우 마스터 포트 통신 시스템과 슬레이브 포트 통신 시스템의 개수도 각각 m개, n개가 된다. 즉 읽기/쓰기 데이터 채널은 컴포넌트의 개수에 따라 디코더 및 중재기 등을 포함하는 통신 시스템의 개수도 선형적으로 증가하게 된다. 반면 주소 채널과 쓰기 응답 채널은 모든 마스터 및 슬레이브가 통신 시스템을 공유하는 Shared 버스구조를 가진다. 따라서 읽기주소 채널, 쓰기주소 채널, 쓰기응답 채널은 각각 중재기와 라우터, 디코더로 구성된 하나의 통신 시스템을 공유하게 된다.

본 논문의 Ⅱ장에서는 AXI프로토콜 및 기존 온-칩 버스구조와 AXI 채널의 트래픽을 고려한 Hybrid 온-칩 버스구조를 비교하고, Ⅲ장에서는 본 연구에서 제안한 AXI Hybrid 온-칩 버스구조의 아키텍처 설계에 대해 설명하고, Ⅳ장에서는 Shared 버스구조와 Crossbar 버스구조, 제안된 Hybrid 버스구조 등의 성능과 오버헤드를 비교 실험하고 분석 결과를 기술하며, Ⅴ장에서 결론을 맺는다.

Ⅱ. AXI프로토콜과 온-칩 버스구조

본 연구에서 사용하는 AXI프로토콜과 기존 온-칩 버스에 대해 설명한다. 그리고 AXI프로토콜의 채널 트래픽을 고려한 온-칩 버스에 대해 분석한다.

1. AXI 기반의 기존 온-칩 버스

가. AMBA 3.0 AXI프로토콜

AXI는 2003년 ARM사에 의해 발표된 온-칩 버스시스템인 AMBA 3.0의 프로토콜이다[5]. MPSoC의 성능개선을 위해서 AXI프로토콜은 트랜잭션의 병렬처리를 위한 Multiple outstanding transaction과 트랜잭션의 지연시간을 줄이기 위한 Out-of-order completion 기능을 포함하고 있다. 그리고 AXI는 특정 ID를 가진 트랜잭션 사이에 다른 ID를 가진 트랜잭션을 삽입하는 데이터 인터리빙(Data Interleaving) 기능을 통해서 데이터 버스의 사용 효율을 높일 수 있다.

그림 1과 같이 AXI는 읽기와 쓰기 채널의 분리, 그리고 주소/제어 신호와 데이터 신호로 분리된 5개의 독립 채널을 가진다. AXI는 읽기와 쓰기 채널의 분리로 읽기와 쓰기 동작이 동시에 가능해서 높은 병렬성을 제공한다. 그러나 AXI는 채널마다 통신 시스템의 구현이 필요하다는 문제를 가지고 있다. 하지만 AXI는 주소 채널 및 쓰기응답 채널에서 최대 1번의 주소 트래픽의 전송이 이루어지며, 데이터 채널에서는 최대 16번의 데이터 트래픽 전송이 이루어진다. 따라서 AXI는 시작 주소에 기반을 둔 버스트 트랜잭션을 통해 불필요한 주소 트랜잭션을 제거하여 주소 채널의 사용 효율 및 복잡도를 줄임으로써 효과적인 처리가 가능하다.[5]

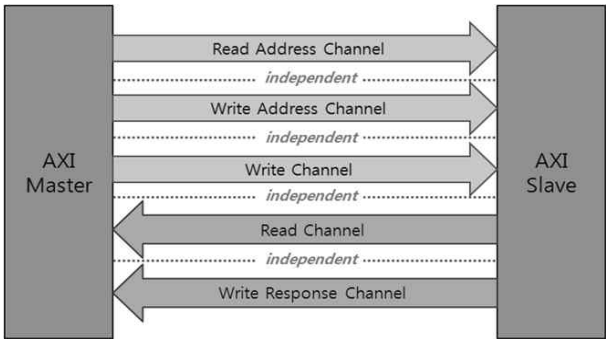


그림 1. AXI의 다중 채널
Fig. 1. Multiple channels of AXI.

나. 기존 온-칩 버스

현재 널리 사용되고 있는 온-칩 버스구조에는 Shared 버스, Crossbar 버스, NoC 등이 있다.[7~8] 그림 2의 Shared 버스는 기본적으로 시스템을 구성하는 모든 컴포넌트들이 통신회선을 공유한다. 통신회선 공유로 마스터들 간의 버스점유 요청이 증가하면 버스 채널 블록킹 현상이 초래하며, 이로 인해 다른 마스터들은 긴 대기시간을 갖게 된다. 하지만 Shared 버스구조는 통신회선의 공유로 복잡도를 최소로 유지하는 것이 가능해서, 작은 회로크기 및 전력소모 등을 장점으로 갖는다. 따라서 AXI 5개의 채널 중 읽기/쓰기주소 채널과 쓰기응답 채널은 트래픽이 적기 때문에, Shared 버스구

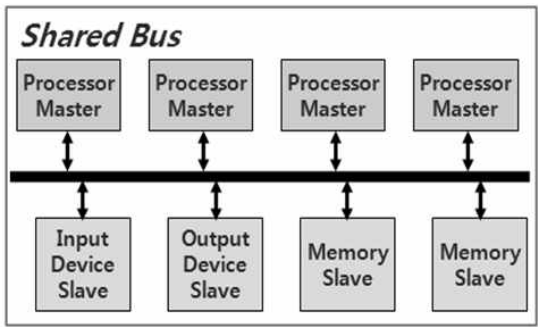


그림 2. Shared 버스
Fig. 2. Shared bus.

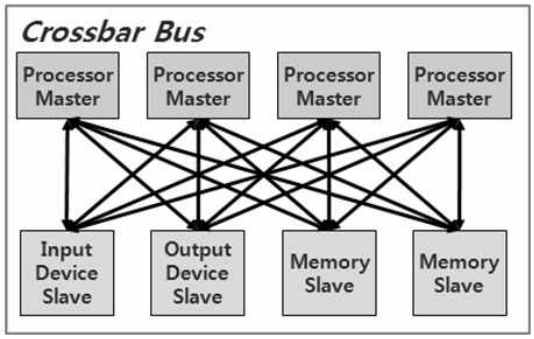


그림 3. Crossbar 버스
Fig. 3. Crossbar bus.

조로 설계하여 온-칩 버스 시스템의 오버헤드를 줄일 수 있게 된다.

Shared 버스의 단점인 마스터들의 긴 대기시간을 해결하고 고성능이 요구되는 시스템에 적합한 인터커넥션이 그림 3의 Crossbar 버스구조이다.[6] Crossbar 버스구조는 모든 마스터와 슬레이브간 1:1 통신 접속을 유지하며, 마스터들의 대기시간이 짧아짐에 따라 신속한 트랜잭션의 처리가 이루어진다. AXI 5개의 채널 중 전송 트래픽이 많은 읽기/쓰기 데이터 채널들은 트래픽의 고속 처리를 위해 마스터의 짧은 대기시간을 갖는 Crossbar 버스구조로 설계하는 것이 유리하다. 하지만 Crossbar 버스구조는 복잡도의 급격한 증가때문에 높은 하드웨어 비용과 많은 전력소모 등의 한계를 보이고 있다.

2. AXI기반 Hybrid 온-칩 버스

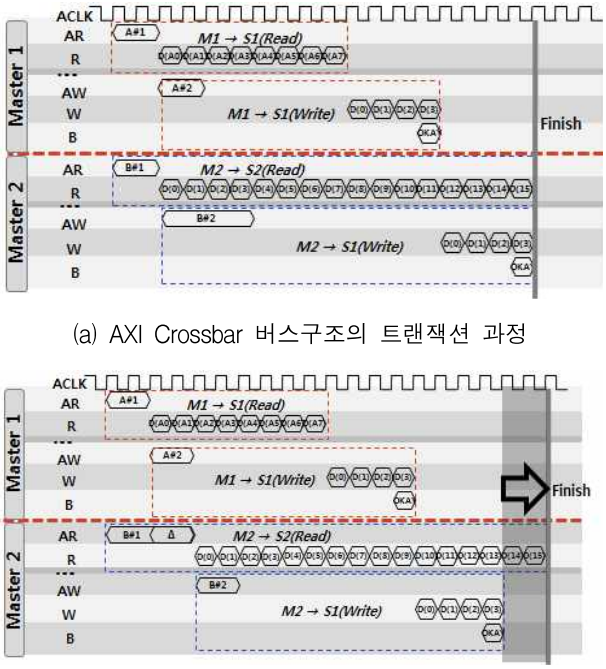
본 연구는 AXI프로토콜에서 많은 트래픽이 발생하여 짧은 대기시간이 요구되는 데이터 채널을 Crossbar 버스구조로 설계하고, 트래픽 발생이 적은 주소 및 쓰기응답 채널들을 Shared 버스구조로 구현한 Hybrid 형태의 버스구조를 설계하였다.

가. Hybrid 버스의 트랜잭션

2개의 마스터와 슬레이브로 구성된 버스에서 표 1의 트랜잭션 동작에 대해 그림 4(a)는 AXI의 모든 채널에서 동시 처리가 가능한 AXI Crossbar 버스구조의 트랜잭션 처리 결과를 보여준다. 그림 4(b)는 AXI의 각 주소 채널 및 쓰기응답 채널은 공유하고 데이터 채널은 Crossbar 버스의 병렬성을 유지하는 AXI Hybrid 버스구조의 트랜잭션 처리 과정을 보여주고 있다. 먼저 마스터1과 마스터2는 각각 슬레이브1과 슬레이브2에 동시에 읽기 트랜잭션을 발생시킨다. 실제로 그림 4(a)의 마스터1과 마스터2는 읽기주소 채널(AR)에서 동시에 트랜잭션을 요청 및 처리하는 것이 가능하다. 하지만 4(b)의 Hybrid 버스구조는 읽기주소 채널(AR)의 중재기 정책에 의해 마스터1이 먼저 트랜잭션에 대한 응답을 받

표 1. 2x2 버스의 트랜잭션 동작
Table 1. Operations of 2x2 bus transaction.

마스터1	① 마스터1 ← 슬레이브1(읽기)
	② 마스터1 → 슬레이브2(쓰기)
마스터2	① 마스터2 ← 슬레이브2(읽기)
	② 마스터2 → 슬레이브1(쓰기)



(a) AXI Crossbar 버스구조의 트랜잭션 과정

(b) AXI Hybrid 버스의 트랜잭션 과정

그림 4. AXI Crossbar 및 AXI Hybrid 버스의 트랜잭션
Fig. 4. Transactions of AXI Crossbar and AXI Hybrid bus.

고, 마스터2는 대기하게 된다. 반면에 읽기데이터 채널에서의 트랜잭션은 그림 4(a)와 4(b) 모두 동시에 처리가 된다. 결국 그림 4(b)에서는 동일한 동작에 대해 모든 트랜잭션 완료까지 주소 채널에서 마스터2가 대기한 시간(Δ)만큼 더 소요된다. 하지만 주소 채널 및 쓰기응답 채널의 트래픽이 적기 때문에 채널을 공유함에 따른 전체적인 성능 저하는 적을 것으로 예상된다.

나. Crossbar 및 Hybrid 버스구조의 오버헤드

그림 5와 6은 각각 AXI Crossbar 버스와 AXI Hybrid 버스에서 요구되는 통신 시스템을 보인다. 각각의 통신 시스템은 중재기 및 디코더 등의 컴포넌트와 통신회선을 포함한다. 그림 5의 Crossbar 버스는 모든 채널의 마스터 포트와 슬레이브 포트에서 별도의 통신 컴포넌트를 가져야 한다. 따라서 마스터 및 슬레이브가 추가될 때마다 5개의 통신 컴포넌트들이 추가적으로 요구된다. 즉, 마스터의 개수 m, 슬레이브의 개수 n인 경우 Crossbar 버스에서 필요한 통신 컴포넌트는 5(m+n)이 된다. 반면에 그림 6의 AXI Hybrid 버스는 주소 채널 및 쓰기응답 채널의 통신 컴포넌트를 공유하는 구조를 갖는다. 따라서 마스터 및 슬레이브가 증가할 때마

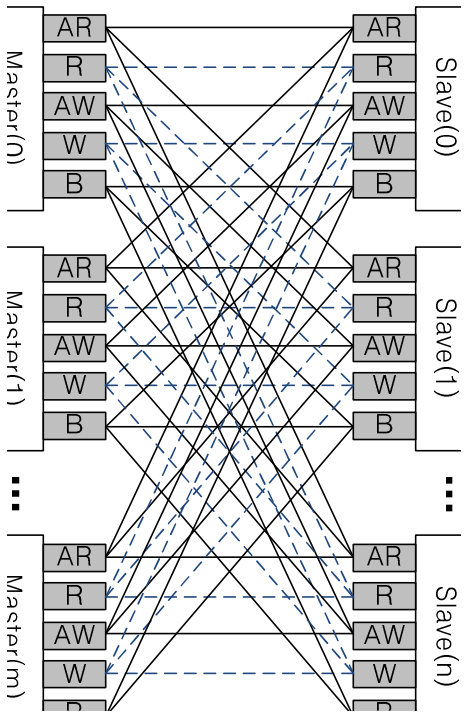


그림 5. AXI Crossbar 버스구조의 통신 시스템
Fig. 5. Communication system of AXI Crossbar bus architecture.

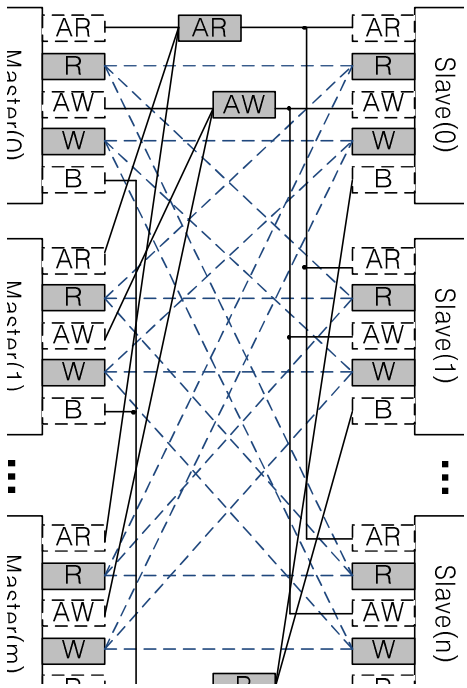


그림 6. AXI Hybrid 버스구조의 통신 컴포넌트
Fig. 6. Communication components of AXI Hybrid bus architecture.

다 쓰기 및 읽기 데이터 채널 2개의 통신 컴포넌트만 추가적으로 요구된다. 즉, Hybrid 버스에서 요구되는

표 2. 각 버스구조별 통신 컴포넌트
Table 2. Communication components of each bus architecture.

구분	2x2	4x4	8x8	16x16
Shared Bus	5개			
Crossbar Bus	20개	40개	80개	160개
Hybrid Bus	11개	19개	35개	67개

표 3. 각 버스구조별 통신회선
Table 3. Interconnection wires of each bus architecture.

구분	Shared	Crossbar	Hybrid
2x2	184~204	736~816	412~456
4x4		2,944~3,264	1,324~1,464
8x8		11,776~13,056	4,972~5,496
16x16		47,104~52,224	19,564~21,624

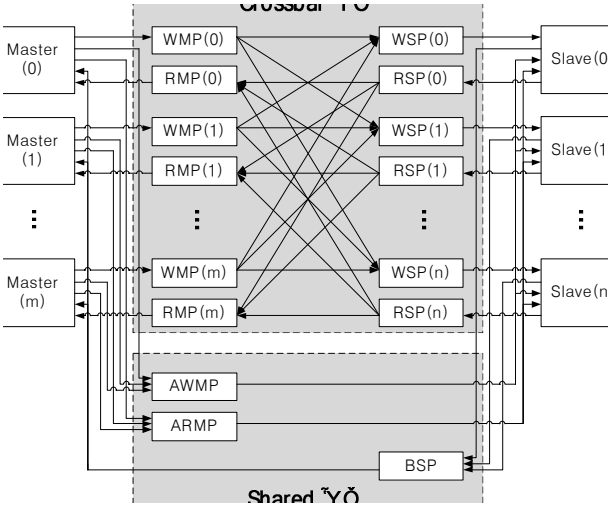
통신 컴포넌트는 2(m+n) + 3이 된다. 표 2는 4x4 이상의 Hybrid 버스에서 요구되는 통신 컴포넌트 개수가 Crossbar 버스에서 요구되는 통신 컴포넌트의 개수의 절반 이하로 줄어든다는 것을 보인다. 게다가 마스터와 슬레이브의 개수가 증가할수록 Crossbar 버스와 Hybrid 버스에서 요구되는 통신 컴포넌트 개수의 차이는 커지게 된다.

한편, AXI프로토콜의 주소 채널(AW/AR)은 32비트 데이터폭을 기준으로 할 경우 52~56개의 통신회선을 가진다. 그리고 쓰기데이터 채널(W)의 통신회선은 39~43개, 읽기데이터 채널(R)의 통신회선은 37~41개, 쓰기응답 채널(B)은 4~8개의 통신회선을 가진다.[5] Crossbar 버스는 그림 5와 같이 모든 마스터와 모든 슬레이브간 통신회선을 독립적으로 연결하고 있다. 반면에 그림 6의 Hybrid 버스는 주소 및 쓰기응답 채널에 대해서 통신회선을 공유하게 된다. 표 3은 각 버스구조별 마스터와 슬레이브의 개수 변화에 따라 사용되는 통신회선 수를 나타낸다. AXI Hybrid 버스의 통신회선 수는 Crossbar 버스와 대비할 때 2x2에서는 약 45% 감소, 8x8에서는 약 58%의 감소효과를 보인다. 또한 컴포넌트의 개수가 증가할수록 AXI Crossbar 버스에 비교해서 Hybrid 버스의 통신회선 감소 효과가 커진다.

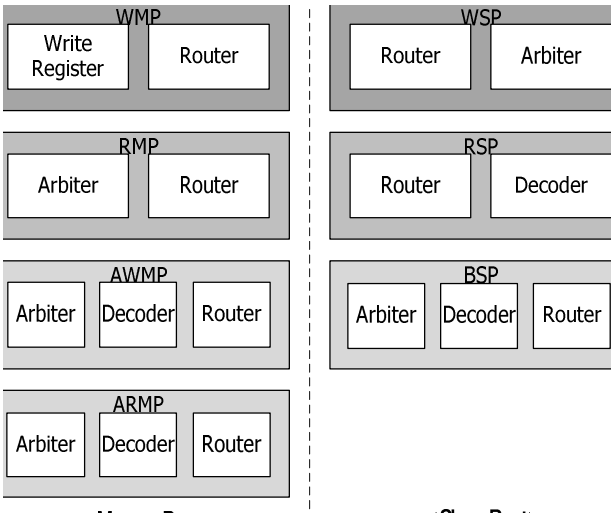
Ⅲ. AXI Hybrid 온-칩 버스의 아키텍처 설계

1. 버스 아키텍처 설계

본 연구에서 제안하는 AXI Hybrid 버스의 아키텍처



(a) Hybrid 버스 아키텍처



(b) Hybrid 버스의 통신 컴포넌트

그림 7. AXI 하이브리드 버스 아키텍처
Fig. 7. AXI Hybrid bus architecture.

를 그림 7에 보인다. AXI Hybrid 버스의 아키텍처는 쓰기/읽기 데이터 채널의 통신 시스템과 나머지 3개 채널들의 통신 시스템으로 구분할 수 있다. 쓰기/읽기 데이터 채널은 대부분의 트래픽이 처리되는 채널로서, Crossbar 버스구조로 설계된다. 그리고 쓰기/읽기주소 채널 및 쓰기응답의 3개 채널은 Shared 버스구조로써 설계된다. 버스의 마스터는 마스터 포트와 연결되고 각 슬레이브는 슬레이브 포트와 연결되며, 마스터 포트 및 슬레이브 포트는 각 채널의 통신 시스템을 구성한다.

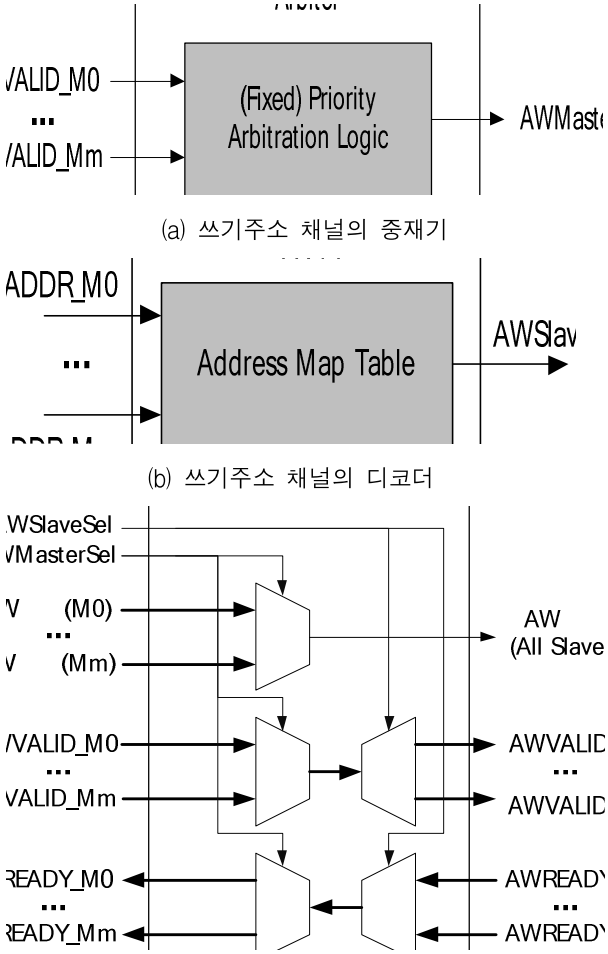
Crossbar 버스구조는 마스터와 슬레이브간 1:1 통신 접속을 유지하므로 쓰기/읽기 데이터 채널은 별도의 통신 컴포넌트를 가져야 한다. 쓰기데이터 채널의 마스터 포트(WMP)에는 쓰기주소 채널의 디코더로부터 발생한 슬레이브를 선택하는 신호를 저장하는 레지스터와 멀티플렉서 및 디멀티플렉서로 이루어진 라우터로 구성된다. 쓰기데이터 채널의 슬레이브 포트(WSP)에는 모든 마스터들로부터 전송된 트랜잭션을 중재하기 위한 중재기와 라우터로 구성된다. 읽기데이터 채널은 슬레이브에서 마스터로 전송방향을 가진다. 따라서 읽기데이터 채널의 슬레이브 포트(RSP)에는 디코더와 라우터, 마스터 포트(RMP)에는 중재기와 라우터가 각각 포함된다. 마스터의 개수가 m개일 경우 그림 7의 WMP와 RMP의 통신 시스템의 개수도 각각 m개가 된다. 그리고 슬레이브의 개수가 n개일 때, WSP와 RSP의 통신 시스템도 각각 n개가 된다. 즉, 통신 컴포넌트의 개수는 마스터 및 슬레이브 컴포넌트의 개수에 따라 선형적으로 증가하게 된다.

쓰기 및 읽기 주소 채널과 쓰기응답 채널은 Shared 버스구조와 같이 모든 마스터 및 슬레이브들이 통신 컴포넌트를 공유하는 구조를 가진다. 따라서 주소 및 쓰기응답 채널들은 각각 중재기와 디코더, 라우터로 구성된 통신 시스템을 공유하게 된다. 즉, 쓰기주소 채널은 마스터 및 슬레이브의 개수에 상관없이 하나의 AWMP 통신 시스템을 공유하게 된다. 같은 방법으로 읽기주소 채널과 쓰기응답 채널도 각각 ARMP와 BSP 통신 컴포넌트를 공유하도록 설계된다.

2. 버스 구조 및 동작

가. 쓰기/읽기주소 채널

쓰기주소 채널의 중재기, 디코더, 라우터의 구조가 그림 8에 보인다. 먼저 그림 8(a)의 중재기에서는 (고정) 우선순위 중재 알고리즘을 구현한다. 그리고 중재기는 모든 마스터들로 부터 쓰기주소 채널의 유효함을 의미하는 신호인 AWVALID 신호를 입력으로 받는다. 그림 8(b)의 디코더는 시스템의 메모리맵에 의해 정해진 주소 맵 테이블에 따라 슬레이브를 선택하는 신호를 출력한다. 중재기와 디코더의 출력신호는 주소 채널의 라우터로 전송된다. 그림 8(c) 라우터는 전송 받은 선택 신호를 이용하여 해당 마스터의 신호를 목표로 하는 슬레이브로 전송할 수 있도록 연결시켜 준다. 중재기와 디코더에 의해 선택받지 못한 마스터 및 슬레이브의 AWVALID와 AWREADY신호는 '0'신호를 갖게 된다. 읽기주소 채널의 통신 컴포넌트는 쓰기주소 채널의 통신 컴포넌트와 동일한 구조를 가진다.



(a) 쓰기주소 채널의 중재기

(b) 쓰기주소 채널의 디코더

(c) 쓰기주소 채널의 라우터

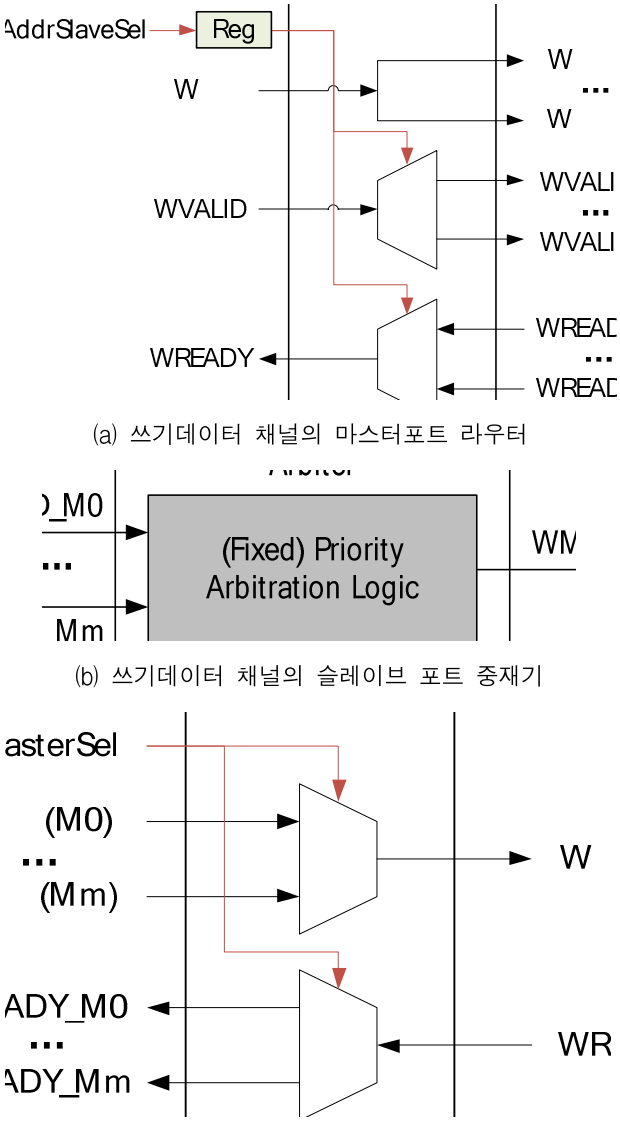
그림 8. 쓰기주소 채널의 통신 컴포넌트
Fig. 8. Communication components architecture of write address channel.

나. 쓰기응답 채널

쓰기응답 채널은 슬레이브에서 마스터로 트랜잭션을 전송한다. 쓰기응답 채널의 통신 시스템은 디코더의 입력이 주소가 아닌 BID라는 것을 제외하고 주소 채널의 통신 시스템과 동일한 구조로 설계된다.

다. 쓰기데이터 채널

마스터 포트(그림 9(a))에서는 쓰기주소 채널 디코더의 출력인 슬레이브 선택신호를 입력받아 레지스터에 저장한다. 그리고 레지스터에 저장된 슬레이브 선택신호는 라우터의 입력으로서 마스터의 신호를 목표 슬레이브로 전송하기 위해 멀티플렉서 및 디멀티플렉서 선택신호의 기능을 가진다. 그림 9(b)와 같이 쓰기데이터 채널에서 슬레이브 포트의 중재기 구조는 모든 마스터



(a) 쓰기데이터 채널의 마스터포트 라우터

(b) 쓰기데이터 채널의 슬레이브 포트 중재기

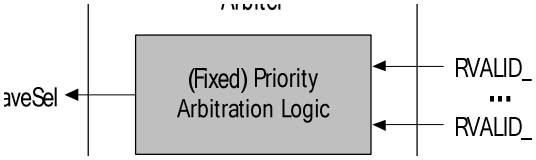
(c) 쓰기데이터 채널의 슬레이브 포트 라우터

그림 9. 쓰기데이터 채널의 통신 컴포넌트
Fig. 9. Communication components architecture of write data channel.

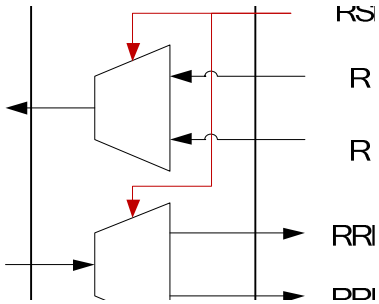
로 부터 전송된 WVALID 신호를 입력으로 받아 (고정) 우선순위 중재 알고리즘에 의해 마스터 선택신호를 출력한다. 그림 9(c)는 쓰기데이터 채널 슬레이브 포트의 라우터 구조를 보여준다. 라우터는 중재기의 출력신호를 입력으로 받아 선택 신호로 사용한다. 라우터에서는 중재 알고리즘에 의해 선택된 마스터와 슬레이브간의 통신 접속을 연결시켜 준다.

라. 읽기데이터 채널

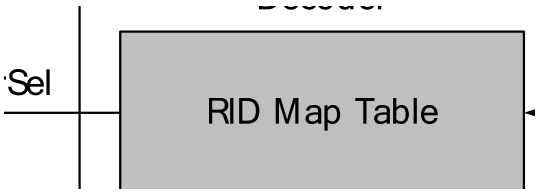
읽기데이터 채널 마스터 포트(그림 10(a))의 중재기 구조를 보여준다. 중재기에서는 RVALID 신호를 입력



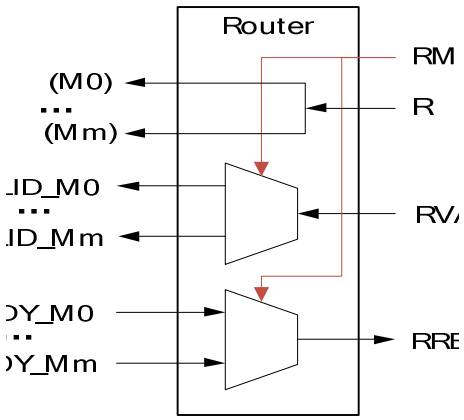
(a) 읽기데이터 채널의 마스터 포트 중재기



(b) 읽기데이터 채널의 마스터 포트 라우터



(c) 읽기데이터 채널의 슬레이브 포트 디코더



(d) 읽기데이터 채널의 슬레이브 포트 라우터

그림 10. 읽기데이터 채널의 통신 컴포넌트
Fig. 10. Communication components architecture of read data channel.

받아 (고정) 우선순위 중재 알고리즘에 의해 슬레이브 선택신호를 출력한다. 그림 10(b)는 읽기데이터 채널 마스터 포트의 라우터 구조를 보여준다. 라우터는 모든 슬레이브로부터 전송된 신호와 중재기로부터 전송된 슬레이브 선택신호, 그리고 마스터에서 슬레이브로 전송할 핸드쉐이크 신호인 RREADY 신호를 입력받는다. 라우터 내부의 멀티플렉서 및 디멀티플렉서는 중재기로부터 전송된 슬레이브 선택신호에 의해 선택된다. 그림 10(c)는 읽기데이터 채널 슬레이브 포트의 디코더 구조를 보여준다. 디코더는 RID를 입력으로 받아 RID 맵 테이블에 의해 마스터 선택신호를 출력한다. 이 출력신호는 라우터에 입력된다. 그림 10(d)는 읽기데이터 채널 슬레이브 포트의 라우터 구조를 보여준다. 라우터는 모든 마스터와 해당 슬레이브의 신호를 연결시킨다. 슬레이브로부터의 신호 중 핸드쉐이크 신호를 제외한 나머지 신호는 모든 마스터로 전송된다. 핸드쉐이크 신호인 RVALID 신호와 RREADY 신호는 디코더로부터 전송된 마스터 선택신호에 의해 선택되어 출력된다.

Ⅳ. 실험 및 분석

1. 실험 방법

본 연구에서는 AXI 온-칩 버스의 비교 실험을 위해서 Shared/Crossbar/Hybrid 버스구조를 Verilog-HDL로 구현하였다. 그리고 실행시간/대기시간/연결회선/회로크기/전력소모 등의 실험을 위해 0.13㎛ 공정 라이브러리를 사용하였다. 기본적으로 모든 버스의 중재기는 고정 우선순위 알고리즘을 사용하였고, 모든 버스는 32비트 데이터폭을 가진다.

본 연구에서 제안하는 버스구조의 실험을 위해 트래픽 생성모델로서 M/G/1 큐잉 모델[9]을 사용하였다. 트래픽 실험을 위해서 마스터 개수(2, 4, 8, 16)와 트랜잭션 생성빈도(λ=0.1/0.2/0.3/0.4)을 갖는 테스트벤치를 구현하였다. 실제로 대표적인 H.263 영상 압축 코덱이나 MP3 음성코덱의 경우 0.15 이하의 트랜잭션 생성빈도를 가진다고 하는데, 0.2 이상의 값은 버스와 메모리 시스템에 매우 큰 부하를 주는 트랜잭션 생성빈도라고 할 수 있다.[9] 마스터의 개수와 트랜잭션 생성비율의 조합에 대해 각 마스터들은 포아송 분포에 따라 100,000번 트랜잭션을 발생시키는 테스트벤치를 구현하였다. 그리고 하나의 트랜잭션은 2, 4, 또는 8번의 데이터 읽기 및 쓰기 동작을 하며 이는 균일 분포의 확률에 의해 선택된다.

2. 실험 결과 및 분석

가. 시간영역(Time domain)

시간영역의 성능실험에서는 포아송 프로세스의 트랜잭션 생성빈도(λ=0.1/0.2/0.3/0.4)에 따라 모든 마스터들

의 트랜잭션이 완료될 때까지 실행시간과 대기시간의 실험 결과를 보여준다.

(1) 마스터 트랜잭션 실행시간(Execution time)

트랜잭션 생성빈도(λ=0.1/0.2/0.3/0.4)에 따라 모든 마스터의 트랜잭션이 완료되기까지의 실행시간이 그림 11에 보인다. 또한 다양한 마스터와 슬레이브들이 구성한 버스에서의 실행시간을 비교한 결과가 그림 12에 보인다. 그림 11(a)와 같이 트랜잭션 생성빈도(λ)가 낮을 경우 마스터의 개수가 증가할수록 issue capability가 1인 Shared 버스의 실행시간이 다른 버스구조에 비해 확연히 길어짐을 보인다. 그리고 Crossbar 버스와 Hybrid 버스의 실행시간은 완만하게 증가한다. 여기서 issue capability란 버스가 동시에 처리할 수 있는 트랜잭션의 최대 개수를 뜻한다. 그림 11(d)와 같이 트랜잭션 생성빈도가 매우 높을 경우 마스터의 개수가 증가할수록 3개 버스구조의 실행시간 증가폭이 급격히 증가함을 보이고 있다. 한편 마스터의 개수를 기준으로 살펴보면 그림 12(a)과 같이 마스터의 개수가 적을 경우 트랜잭션 생성빈도가 높아질수록 실행시간의 증가폭이 선형적임을 보여준다. 그리고 그림 12(d)과 같이 마스터의 개수가 많을 경우에도 트랜잭션 생성빈도가 높아질수록 실행시간의 증가폭이 선형적임을 보여준다. 하지만 마스터의 개수가 적을 경우에 비해서 증가폭이 높다는 것을 알 수 있다. 실제로, Hybrid 버스는 트랜잭션 생성빈도가 가장 높고 마스터의 개수가 가장 많으며 트래픽의 양이 가장 많을 때에도 Crossbar 버스구조와 비교하여 약 4%의 실행 시간의 차이를 보이며 근접된 성능을 나타냈다. 즉 Hybrid 버스는 대다수의 데이터 트래픽을 고속으로 처리함으로써 Crossbar버스와 근접한 성능으로 동작함을 보여주고 있다. 이는 Hybrid 버스가 대다수의 트래픽 전송을 Crossbar 버스로써 처리해서 데이터의 병렬 인터커넥션이 가능했기 때문이다.

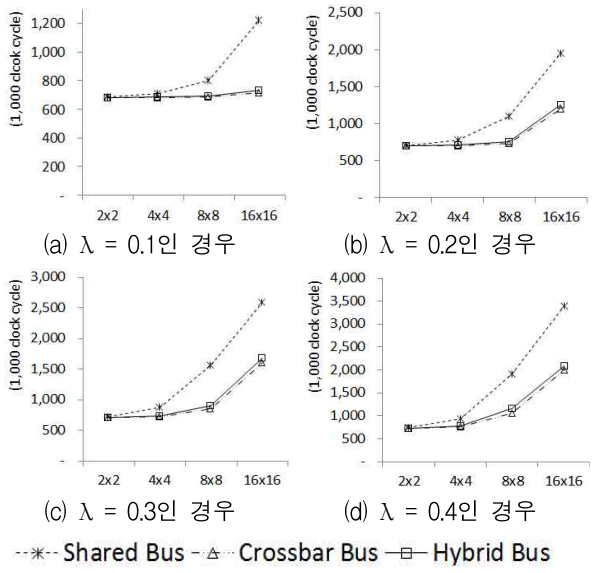


그림 11. 트랜잭션 생성빈도(λ)에 따른 트랜잭션 실행시간

Fig. 11. Transaction execution time with λ.

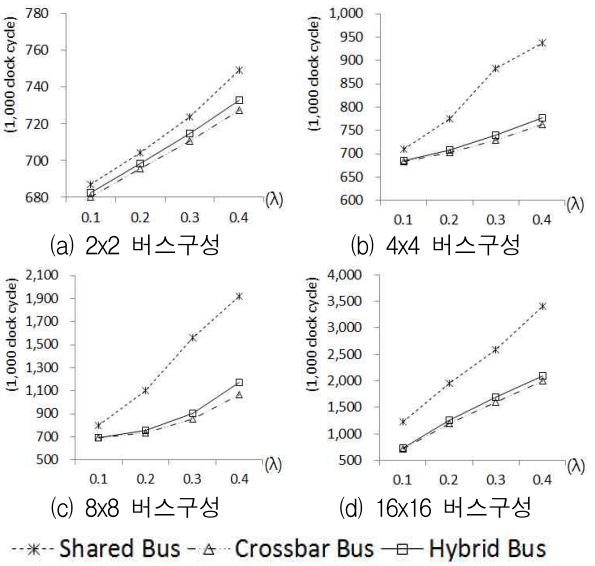


그림 12. 버스구성에 따른 트랜잭션 실행시간

Fig. 12. Transaction execution time with bus configuration.

(2) 마스터 트랜잭션 대기시간(Latency)

그림 13은 슬레이브가 10,000개의 트랜잭션을 처리할 동안 모든 마스터의 대기시간을 평균 트랜잭션 생성빈도(λ=0.1/0.2/0.3/0.4)에 따라 그림 13에 보인다. 모든 마스터의 대기시간을 마스터와 슬레이브의 개수 변화에 따라 비교한 결과가 그림 14에 보인다. 그림 13(a)와 같이 트랜잭션 생성빈도(λ)가 낮을 경우에 마스터의 개수가 증가할수록 issue capability가 1인 Shared 버스의 대기시간은 지수적으로 급격히 증가함을 보인다. 반면에 Crossbar 버스와 Hybrid 버스는 선형적으로 완만하게 증가함을 보이고 있다. 그림 13(d)와 같이 트랜잭션 생성빈도가 매우 높을 경우에 마스터의 개수가 증가할수록 Shared 버스는 물론 Crossbar 버스와 Hybrid 버스의 대기시간도 급격히 증가함을 알 수 있다. 이는 트랜잭션 생성빈도가 높아질수록 마스터의 동시 트랜잭션 요청 회수가 높아지기 때문이다. 한편, 마스터의 개수를

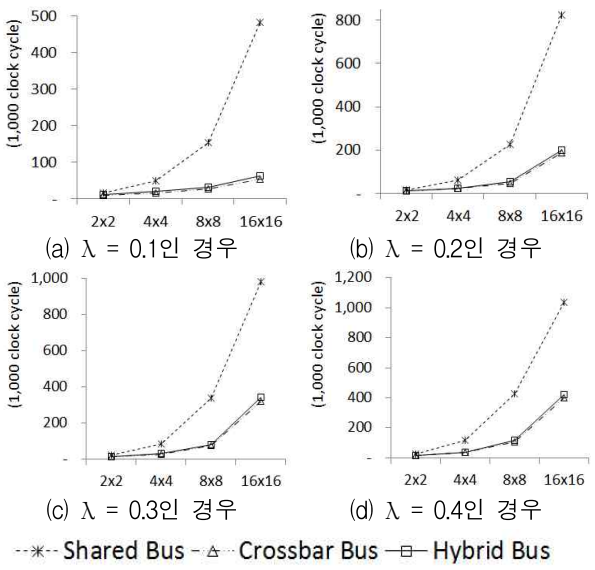


그림 13. 트랜잭션 생성빈도(λ)에 따른 트랜잭션 대기시간
Fig. 13. Transaction latency with λ.

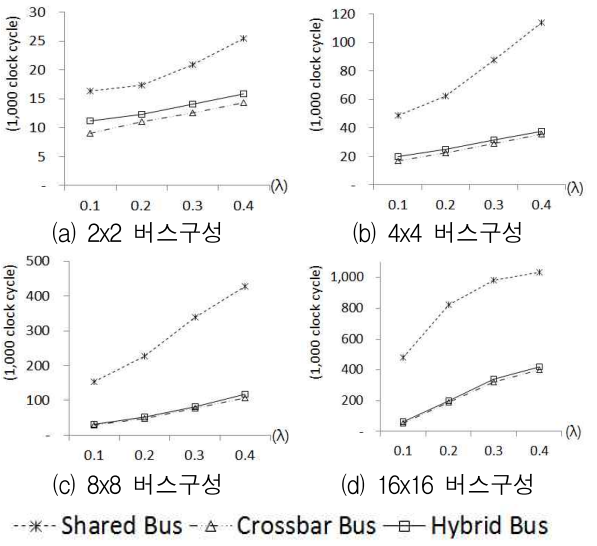


그림 14. 버스구성에 따른 트랜잭션 대기시간
Fig. 14. Transaction latency with bus configuration.

기준으로 살펴보면 그림 14(a)과 같이 마스터의 개수가 적을 경우 트랜잭션 생성빈도가 높아질수록 대기시간은 선형적으로 소폭 증가함을 보여준다. 그러나 그림 14(d)와 같이 마스터의 개수가 많을 경우에 트랜잭션 생성빈도가 높아질수록 대기시간의 증가폭이 커짐을 알 수 있다. 즉, 버스를 구성하는 마스터의 개수가 증가할수록 대기시간은 높아지는데, 이는 마스터 개수의 크기가 마스터의 동시요청 회수 증가에 직접적인 영향을 미치기 때문이다. 특히, 트랜잭션 생성빈도와 마스터의 개수와 무관하게 Hybrid 버스의 대기시간은 Crossbar 버스의 대기시간과 근접된 실험결과로서 나타나고 있다. 실제로 λ=0.1부분만 아니라 λ>0.2의 큰 부하[9]에 대해서도 Hybrid 버스는 Crossbar 버스와 비슷한 대기시간성능을 보이고 있다.

나. 공간영역(Space domain)

공간영역의 오버헤드 실험을 통해서 Crossbar, Hybrid, Shared 버스구조의 마스터와 슬레이브간 연결회선 수(Interconnection wires)와 회로크기(Area) 등을 비교한 결과를 구했다.

(1) 연결회선 수(Interconnection wires)

마스터 포트와 슬레이브 포트간 연결회선 수를 마스터 및 슬레이브 개수에 따라서 비교한 결과가 그림 15에 보인다. 연결회선은 AXI의 모든 데이터 신호와 일부 제어신호를 포함한다. AXI Shared 버스의 경우 모든 마스터와 슬레이브가 통신회선을 공유하기 때문에 마스터/슬레이브의 수에 상관없이 거의 일정하게 연결회선이 유지된다. AXI Crossbar 버스에서는 컴포넌트 증가에 따른 연결 회선의 수가 지수적으로 급격히 증가함을 보이고 있다. AXI Hybrid 버스도 지수적으로 증가하나 주소 채널 및 쓰기 응답 채널의 연결 회선 공유때문에 증가폭이 상대적으로 작다는 것을 알 수 있다. 실제로 8x8 이상에서는 Crossbar 버스 대비 연결 회선수의 증가율이 50%이상 감소한다. 따라서 Hybrid 버스구조는 MPSoC 환경에서 기존 Crossbar 버스구조의 문제점인 마스터 추가에 따른 연결 회선에 대한 부담을 크게 완화시켜 온-칩 시스템의 전체 복잡도를 감소시킬 수 있음을 보이고 있다.

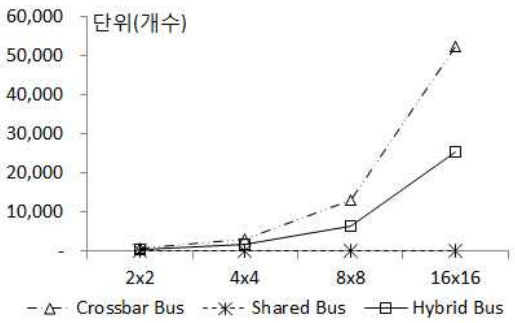


그림 15. 연결회선 수
Fig. 15. Interconnection wires.

(2) 회로크기(Circuit Area)

Synopsys의 Design Compiler를 사용하여 마스터 및 슬레이브 개수별로 설계 합성한 회로크기가 그림 16에 보인다. AXI Shared 버스는 통신회선을 공유하기 때문에 컴포넌트 증가에 따른 회로크기의 증가율이 선형적이며 증가폭도 작다. 반면 AXI Crossbar 및 Hybrid 버스는 모두 회로크기의 증가율이 지수적으로 증가하고 있다. Crossbar 버스는 마스터가 N개 증가하면 각 채널의 통신컴포넌트도 N개가 추가되지만 Hybrid 버스는 마스터의 증가에 대해서 주소 채널 및 쓰기응답 채널이 각각 1개의 통신 컴포넌트를 공유한다. Hybrid 버스구조는 주소 및 쓰기응답 채널에서 통신 시스템을 공유함으로써 Crossbar 버스구조와 비교하여 2x2에서는 15%, 4x4에서는 34%, 8x8에서는 42%, 16x16에서는 47%의 회로크기가 감소하는 것을 보인다. 즉 마스터의 수가 증가할수록 감소 비율은 더욱 커지는 결과를 보이면서, Hybrid 버스는 Crossbar 버스와 비교해 MPSoC에서 코어 추가에 따른 회로크기의 비용을 효과적으로 감소시킬 수 있는 AXI아키텍쳐로서 입증되고 있다.

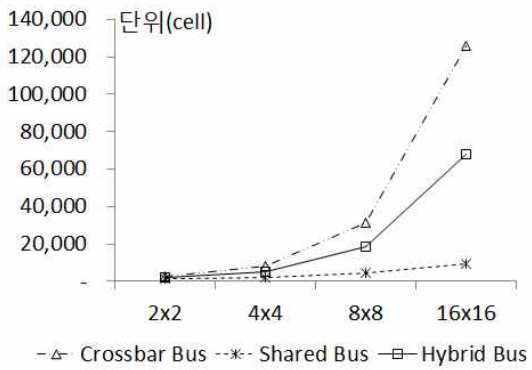


그림 16. 회로크기
Fig. 16. Circuit Area.

다. 파워영역(Power domain)

파워영역에서는 다양한 트랜잭션 생성빈도에 따라 각 버스구조의 동적 전력소모를 Synopsys의 Power Compiler를 이용해 측정하였다. 트래잭션의 생성빈도별로 각 버스구조에서 마스터/슬레이브 개수에 따른 동적 전력소모를 그림 17에 보인다. 그림 17에서 AXI Shared 버스구조는 모든 마스터/슬레이브가 통신회선을 공유하기 때문에 마스터/슬레이브 개수에 대한 동적 전력소모의 증가량이 미미하다. 반면, Crossbar 버스의 동적 전력소모 증가량은 지수적으로 급격히 증가하는

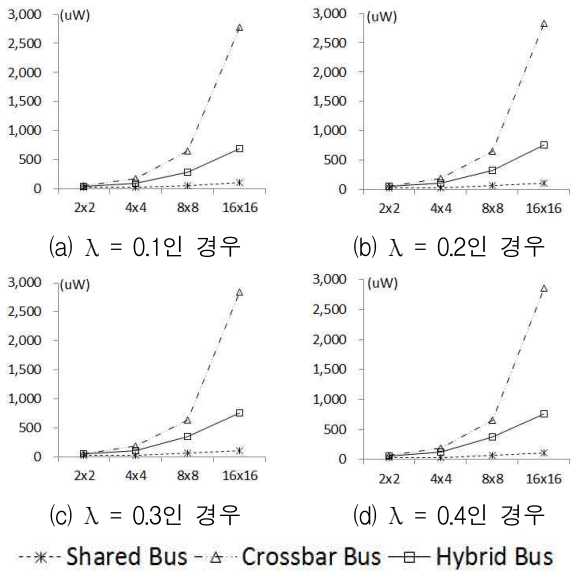


그림 17. 트랜잭션 생성빈도(λ)에 따른 동적 전력소모
Fig. 17. Dynamic power consumption with λ.

것을 알 수 있다. 이는 마스터/슬레이브 추가에 따라 증가하는 중재기, 라우터, 디코더 등의 컴포넌트 로직과 연결회선의 증가에 따른 결과이다. Hybrid 버스에서는 주소 및 쓰기응답의 3개 채널 통신회선을 공유함으로써 동적 전력소모가 선형에 가깝게 증가함을 보여준다. AXI Hybrid 버스는 Crossbar 버스와 비교하여 대략 2x2에서 7%, 4x4에서 40%, 8x8에서 46%, 16x16에서 66%이상 동적 전력소모를 줄이고 있다. 전력소모는 회로크기와 연결회선 수와도 밀접한 관계가 있는데, 마스터/슬레이브 개수가 증가할수록 회로크기 및 연결회선 수가 줄어들게 되고 그에 따라 동적 전력소모를 줄이는 효과는 더욱 커지게 된다.

Ⅳ. 결론 및 추후연구

AXI프로토콜은 기본적으로 5개의 채널로 동작하는데, 이중 주소 채널과 쓰기응답 채널에서는 읽기/쓰기 트랜잭션 발생 시 최대 1개의 데이터 트래픽 전송이 이루어지고 데이터 채널에는 최대 16개의 데이터 트래픽 전송이 이루어진다. 본 연구에서는 AXI프로토콜의 채널별 트래픽을 고려하여 Crossbar 버스와 Shared 버스의 장점을 결합한 Hybrid 온-칩 버스구조를 설계하였다. 트래픽이 많은 데이터 채널은 병렬성을 지원하는 높은 대역폭의 Crossbar 버스구조로 설계하여 고속 트래픽을 처리하였으며, 트래픽이 적은 주소 및 쓰기응답

채널은 Shared 버스구조로 설계하여 컴포넌트 추가에 따른 회로크기, 연결회선 수, 전력소모 등의 오버헤드를 줄였다. 실험결과 AXI Crossbar 버스와 비교하여 AXI Hybrid 버스는 실행시간과 마스터의 대기시간에서 근접한 성능을 보이면서도, 오버헤드에서는 마스터/슬레이브 16x16에서 Crossbar 버스구조와 비교할 때 회로크기는 47%, 연결 회선 수는 52%, 동적 전력소모는 66%의 뚜렷한 개선효과를 보인다. 또한 마스터/슬레이브가 증가할수록 오버헤드의 개선효과는 더욱 커진다는 것을 실험을 통해 확인하였다. 따라서 본 논문에서 설계한 AXI Hybrid 온-칩 버스구조는 고성능과 저전력이 요구되는 MPSoC 환경에서 AXI Crossbar 버스구조보다 효과적으로 나타나고 있다.

본 연구에서 설계한 AXI Hybrid 온-칩 버스구조는 Clock gating 기법과 같은 저전력 설계기법을 고려하지 않았다. 따라서 향후 저전력 설계기법을 적용한 추가 실험과 이종 MPSoC 환경에서의 실험을 위한 다양한 트래픽 생성 모델에 대한 연구, 그리고 실제 어플리케이션을 이용한 연구를 계속 진행할 계획이다.

참 고 문 헌

[1] Xinping Zhu, Sharad Malik, “A Hierachical Modeling Framework for On-Chip Communication Architectures” Proceeding of ICCAD, San Jose, California, pp. 663-670, 2002.

[2] Terry Tao Ye, LUCA BENINI, Giovanni De Micheli, “Packetized On-Chip Interconnect Communication Analysis for MPSoC” Proceeding of the DATE, Messe Munich, Germany, pp. 344-349, 2003.

[3] M. Loghi, F. Angiolini, D. Bertozzi, L. Benini, R. Zafalon, “Analyzing on-chip communication in a MPSoC environment”, Design, Automation and Test in Europe Conference and Exhibition, Proc. Vol.2, pp. 752-757, 2004.

[4] F. Dumitrascu, I. Bacivarov, L. Pieralisi, M. Bonaciu, A.A. Jerraya, “Flexible MPSoC Platform with Fast Interconnect Exploration for Optimal System Performance for a Specific Application”, Design, Automation and Test in Europe, DATE ’06. Proc. Vol. 2, 2006.

[5] ARM, “AMBA AXI Protocol”, June 2003.

[6] M. Nakajima et al., “A 400 MHz 32b embedded microprocessor core AM34-1 with 4.0 GB/s cross-bar bus switch for SoC,”, in Proc. ISSCC, 2002, pp. 274-504.

[7] S.Pasricha, N.Dutt, M.Ben-Romdhane, “Constraint-Driven Bus Matrix Synthesis for MPSoC”, Proc. ASPDAC, 2006.

[8] S. Pasricha, N.D. Dutt, M. Ben-Romdhane, “BMSYN: Bus Matrix Communication Architecture Synthesis for MPSoC”, Computer-Aided Design of Integrated Circuites and Systems, IEEE Trans. Vol. 26. pp. 1454-1464. 2007.

[9] S. Kim, S. Ha, “Fast and Accurate Performance Estimation of Bus Matrix for Multi-Processor System-on-Chip(MPSoC)”, Journal of KIISE, Vol. 35(11), pp. 527-539, December 2008.

[10] 이상택, 전민제, 정의영, “주소 버스 공유를 통한 AXI 크로스바 스위치의 면적 및 전력 소모 감소”, 제16회 한국반도체학술대회, Feb. 2009

저 자 소 개

이 경 호(정회원)

2004년 광운대학교 정보통신 공학과 학사 졸업.

2006년 광운대학교 컴퓨터공학과 석사 졸업.

2006년～현재 광운대학교 컴퓨터 공학과 박사과정

<주관심분야 : 영상신호처리, SoC설계, VLSI, Embedded System>

공 진 흥(평생회원)

1980년 서울대학교 전자공학과 학사 졸업.

1982년 한국과학기술원 전기 및 전자공학과 석사 졸업.

1989년 텍사스주립대학교 컴퓨터공학과 박사 졸업.

<주관심분야 : 영상신호처리, SoC설계, VLSI, Embedded System>