특 집

SoC를 위한 통신망 설계 동향

천 익 재, 김 보 관
충남대학교 전자공학과

I. 서 론

최근의 실리콘 기술의 진보는 단일 실리콘 칩 안에 더욱 많은 기능들을 집적시킬 수 있는 기반을 제공하고 있다. 이러한 진보는 카드형태로 프로세서에 추가되던 주변장치들을 프로세서와 함께 같은 실리콘 칩 내에 집적되도록 하였다. 그 결과, 칩 설계자들은 시스템 설계자들에 의해 다루어지던 문제들을 새롭게 고려하게 되었다. 특히, 다양한 기능 블록들이 계속해서 설계되고 시장 선점에 대한 경쟁이 치열해지면서, SoC에 사용되는 통신망(communication network)은 다양한 기능 블록들을 손쉽게 연결할 수 있어야 하고, 또한 다양한 응용분야에 적용될 수 있도록 확장가능하며 강력한 기능을 수행해야만 한다.

SoC 플랫폼에 사용되고 있는 on-chip 통신망의 형태는 크게 bit나 word 단위 전송에 기반을 둔 버스 구조와 통신 채널에서의 패킷 데이터 전송에 기반을 둔 네트워크 구조로 나눌 수 있다. 버스 구조는 SoC 플랫폼에서의 가장 일반적인 통신망의 형태로써 널리 이용되고 있으며 현재 상용화가 이루어지고 있다. 네트워크 구조는 현재 학술적인 연구단계에 있으며 이와 관련된 다양한 논문이 나오고 있다. 이 구조는 일반적으로 채널 구조 또는 네트워크 라우터의 구조를 갖으며 전송할 데이터를 패킷화하여 전송하는 형태를 갖는다.[1] 본 논문에서는 SoC 플랫폼을 구성하기 위한 통신망 중으로 가장 일반적으로 사용되고 있는 버스 구조를 중심으로 기존의 버스 구조와 SoC 버스 구조의 차이점, 통신망으로써의 SoC 버스의 구조 및 설계 시의 고려사항과 현재 사용되고 있는 상용화된 on-chip 버스들의 구조에 대해서 살펴보고자 한다.

II. On-chip 버스로의 변화

PCI나 ISA와 같은 대부분의 기존 버스 구조들은 PCB 기판 위에서 서로 떨어져 있는 칩들을 연결하는 시스템 레벨 버스로써 설계되어졌다. 그러나 이러한 버스 구조는 보드 레벨에서 버스 신호들의 수를 얼마나 줄일 수 있느냐 하는 문제를 가지고 있었다. 그 이유는 핀과 신호선의 수가 칩의 패키지(package)와 PCB 비용에 직접적인 영향을 주었기 때문이며, 많은 수의 핀은 패키지의 크기를 증가시키고 보드 위에 집적될 수 있는 칩의 수를 제한하는 결과를 가져왔다. 더욱이 시스템 레벨 버스들은 add-in card들과 PCB backplane들을 지원해야 했고, 그것들을 위한 커넥터의 크기와 비용 또한 버스 신호선 개수에 영향을 받았다. 이러한 문제점을 해결하기 위하여 전통적인 시스템 레벨 버스들은 shared tri-state 버스 구조를 사용하게 되었으며, PCI 버스의 경우에 같은 신호선을 어드레스 버스와 데이터 버스의 서로 다른 용도로 공유하게 되었다.[2] 그러나 버스와 여러 기능을 수행하는 기능 블록들이 칩 내부에 구현되면서, 신호선의 증가에 따른 패키지 비용 및 핀 수의 문제는 칩 내부 실리콘 영역에서의 신호선의 증가 문제 즉, 버스가 실리콘 영역을 데이터 전송을 위한 물리적인

부분으로 사용하면서 트랜지스터의 집적도에 영향을 주는 문제로 변화되었다. 또한, 같은 신호선에 연결된 여러 부하들과 수신단들을 갖는 shared tri-state 버스 구조는 합성 툴에서의 처리와 static timing analysis가 어렵다는 문제점을 가지고 있다. 그리고 하나의 버스를 공유하는 구조에 있어서 큰 버스 부하는 시스템의 성능을 저하시키는 원인이 된다. 그러므로 SoC의 버스 구조는 PCB 상에서의 구현을 목적으로 하는 버스 구조와는 다른 특성을 가져야 한다.

On-chip 버스 설계에 있어서 칩 영역의 손실을 줄이면서 응용분야에 알맞은 전송률을 지원하는 버스의 설계는 SoC 설계의 중요한 요소가 되고 있으며, 다양한 기능의 회로들을 추가 회로의 재설계없이 사용할 수 있도록 하는 환경에 대한 고려가 요구되고 있다.

III. On-chip 통신망

하나의 SoC를 구성하기 위해 사용되는 다양한 회로요소(IP)들을 연결하는 것은 아주 복잡하고 시간이 많이 소요되는 작업이라 할 수 있다. 오늘날 대부분의 SoC 응용분야들은 멀티미디어(multimedia), 네트워킹(networking), 텔레커뮤니케이션(telecommunication) 등과 같은 실시간 데이터 전송 효율의 중요성을 강조하는 분야가 중심이 되고 있다. 따라서 데이터의 전송 지연에 기반한 CPU의 address/data 전송 특성과 집적시키고자 하는 회로요소들의 데이터 전송률을 적절히 조율함으로써 성능을 유지하거나 또는 향상시킬 수 있는 통신망이 요구되고 있다.

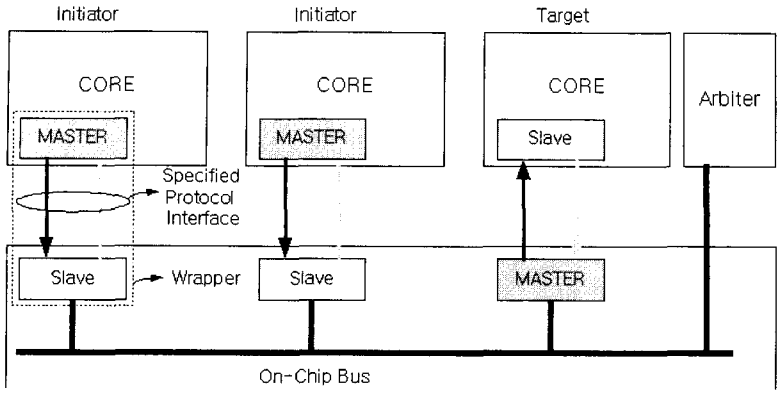
칩 내부에서 통신망은 기능 블록들 사이의 정보 공유 및 데이터와 제어 신호의 전달을 담당하며, 버스 구조에 있어서 통신망은 기능 블록들 사이의 정보 전달을 수행하는 물리적인 버스를 의미한다. SoC에서 회로요소들을 서로 연결하는 통신망은 전체 시스템의 성능을 결정하는 중요한 부분이기 때문에, 설계된 버스 구조의 성능에 따라서 IP 블록들의 원활한 동작과 시스템의 전체 성능이 좌우된다고 할 수 있다.

SoC에 사용되는 기존에 설계된 대부분의 IP들은 재사용을 고려하여 설계되지 않았으며 각자의 독립적인 인터페이스 프로토콜을 사용하고 있다. 그러므로 이러한 IP의 통합을 위해서는 인터페이스 프로토콜을 재수정 하거나 브리지(bridge)의 역할을 수행하는 회로를 재설계해야 하는 문제점이 있다. 여기에 각 IP 회로들이 서로 다른 동작주파수를 이용함으로써 발생되는 시스템 기능 블록간 동기화의 문제도 IP의 재사용에 걸림돌이 되고 있으며, IP 통합에 따른 인터럽트, 핸드 쉐이킹, 데이터 흐름 제어(data flow control) 신호들과 같은 제어 인터페이스(control interface)의 필요성이 요구되고 있다. 본 절에서는 on-chip 버스 구조를 갖는 통신망의 구성요소 및 구조에 대하여 알아보기로 한다.

1. 버스 구성요소

SoC 설계에 있어서 버스의 중요한 역할은 다양한 버스들의 사용을 통해 여러 IP 블록들을 서로 연결하여 사용할 수 있도록 하고, 기능 블록간의 point-to-point 연결을 가능하도록 하는데 있다. 특히, 버스 설계에 있어서 가장 중요한 요소는 IP의 재사용에 있다. IP의 재사용을 위해 해결해야 할 문제점은 기존의 IP 블록들이 서로 다른 인터페이스를 갖는다는 것이다. 이를 해결하기 위하여 서로 다른 프로토콜 인터페이스를 갖는 기능 블록들을 연결시키는 wrapper 로직이 사용된다. 다음의 〈그림 1〉은 SoC에서 IP 재사용을 위해 사용되고 있는 버스 구조와 버스 구성 요소들을 보인다.

버스를 이루는 기본 구성 요소들은 〈그림 1〉에서 보이는 것처럼 개시자(initiator), 타깃(target), 마스터 인터페이스(master interface), 슬레이브 인터페이스(slave interface), 중재기(arbiter)로 이루어진다. 이와 더불어 서로 다른 버스들 사이의 통신을 수행하는 브리지가 사용되기도 한다. 개시자는 전송을 시작하는 마스터 코어를 의미한다. 개시자는 액세스하고자 하는 타

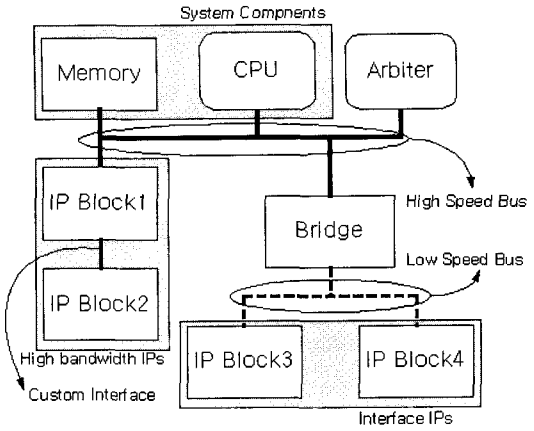


〈그림 1〉 SoC에 사용되는 버스의 구성 요소들

깃과 타깃에 할당된 어드레스를 지정하고, 마스터 인터페이스를 통해 버스사용을 요청하고, 버스 중재기로부터 버스사용 허가를 받는다. 타깃은 슬레이브 코어를 의미하고 전송 요청에 대한 응답만을 수행하며, 데이터 전송을 요구할 수 없다. 마스터 인터페이스와 슬레이브 인터페이스는 규격화된 프로토콜 인터페이스를 사용하여 서로 연결되고 버스와 코어를 연결하는 기능을 수행한다. 이러한 마스터 인터페이스와 슬레이브 인터페이스의 연결 블록을 wrapper 로직이라 한다. 현재 이와 같이 다양한 IP들을 규격화함으로써 SoC에 손쉽게 사용될 수 있도록 하기 위한 프로토콜 인터페이스에 대한 연구가 진행되고 있다.[3][4] 브리지는 두 버스를 연결하는 역할을 수행하며 브리지의 한쪽은 개시자와 같은 기능을 다른 한쪽은 타깃과 같은 기능을 수행한다. 중재기는 버스에 대한 개시자의 접근을 제어한다. 버스 사용 요청(request)에 관한 모든 정보는 버스 중재기에서 받아 들여지고 중재기는 어떤 개시자가 버스를 액세스할 것인지를 결정한다. 중재기는 버스 상에 여러 개의 개시자가 존재할 때 중요한 역할을 수행하며, 하나 이상의 개시자가 버스사용을 요청할 경우 버스를 사용할 개시자를 결정한다.[5]

2. 일반적인 버스 구조

시스템을 구성하는 기능 블록간 통신에 있어서의 버스 설계는 고성능을 요구하는 address/data 전송을 담당하는 고속(high-speed) 버스와 주변장치 연결을 위한 저속(low-speed) 버스, 그리고 특성화된 point-to-point 연결에 중점을 두고 있다고 할 수 있다. 각 주변장치들의 제어 신호들은 병렬 인터페이스를 갖으면서 서로 분리된다. 이러한 통신 구조들은 기본적인 개인용 컴퓨터 시스템의 구조 특성을 반영하여 설계된 구조들로써 독특한 메모리 구조를 요구하고 있다. 다음의 〈그림 2〉는 SoC에 일반적으로 사용되는 버스 구조를 보이고 있다. 고속 버스는 프로세서 버스로써 프로세서를 포함한 고성능 코어들이 이 버스에 연결된다. 이 버스는 일반적으로 데이터와 어드레스가 분리되어 있고 파이프라인(pipeline) 구조를 갖는다. 저속 버스는 주로 주변장치들을 연결하는 버스로써 브리지를 통해서 고속 버스와 연결된다. 이 버스에서는 일반적으로 하나의 개시자가 존재하며 브리지가 그 역할을 수행한다. 저속 버스는 일반적으로 외부 시스템과의 데이터 전송을 지원하기 위한 인터페이스 모듈을 연결하기 위하여 사용된다. 따라서 저속 버스는 고속 버스에 비하여 단순하고, 속도가 느린 특성을 갖는다.[6] 전통적인 버스기반 구조에 있어서 칩 내부의 주변장치들 간의 통신은 블로킹(blocking) 프로토콜을 사용한다. 즉, 어떤 개시자와 타깃사이의 데이터 전송은 다른 장치들이 그 버스를 공유할 수 없도록 제한한다. 예를 들어 오늘날의 버스기반 SoC에서의 블로킹의 문제는 DMA(direct memory access)와 연관된다.



〈그림 2〉 일반적인 SoC 버스 인터페이스 구조.

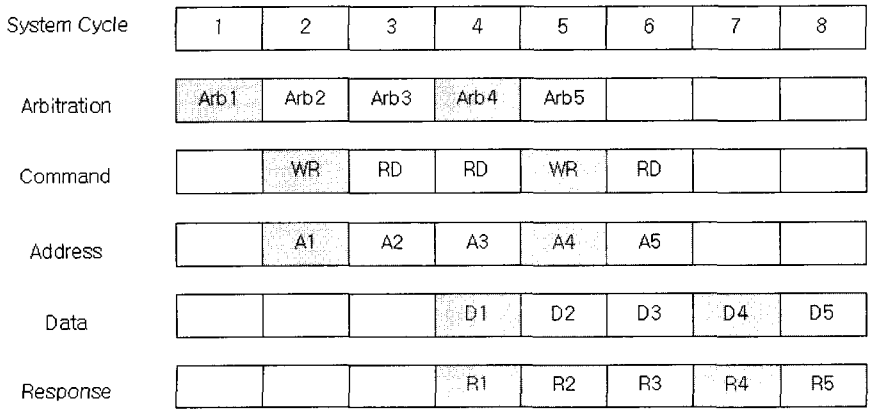
DMA 전송이 수행되는 동안 다른 개시자는 버스에 연결된 어떠한 주변장치들과도 통신할 수 없다. 이러한 동작은 칩의 성능을 저하시키게 되고, 이로 인해 낮아진 데이터 전송률을 보상하기 위하여 개시자에 구현되는 버퍼의 크기를 증가시키는 결과를 가져온다. 또한 DMA 전송의 지속 시간에 따른 지연상태(wait-state)도 성능 저하를 야기하는 원인이 된다.

3. 버스 속성

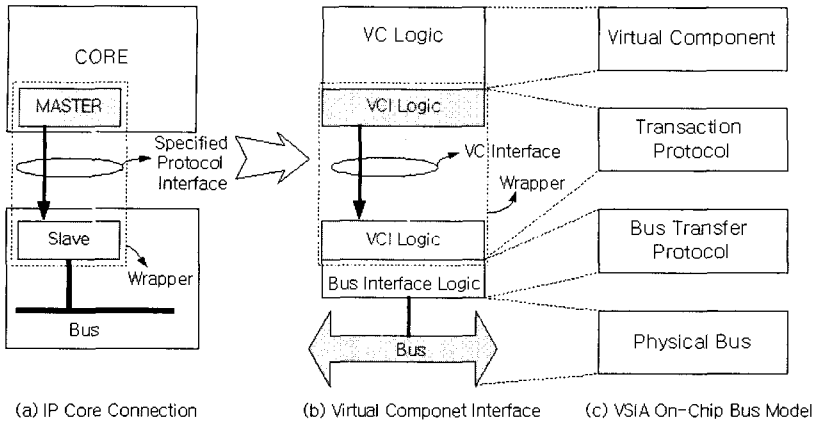
버스 설계에 있어서의 기본적인 고려사항 즉, 버스를 정의하는데 필요한 요소들은 버스의 지연(latency), 대역폭(bandwidth), 엔디언 순서(endian order), 파이프라인 등이 있다. 버스 지연을 결정하는 요소로는 두 가지를 들 수 있다. 첫째는 버스의 프로토콜과 사용도(utilization)에 의한 버스 액세스 시간, 둘째는 프로토콜과 전송되는 데이터의 크기에 의한 데이터 전송 시간이다. 대역폭은 데이터 전송의 최대 크기를 나타낸다. 엔디언 순서는 각 컴포넌트마다 다를 수 있으며 버스는 서로 다른 엔디언 순서를 처리할 수 있어야 한다. 일반적으로 버스내의 엔디언 순서는 고정되며 버스 구성요소 중의 하나인 wrapper 로직에서 그 변환을 담당하게 된다. 파이프라인은 다중 클럭 사이클에서의 버스 프로토콜 특징을 나타낸다. 파이프라인의 크기는 버스의 특성에 따라서 달라진다. 파이프라인 버스 프로토콜(pipeline bus protocol)은 파이프라인을 사용하지 않는 버스 프로토콜(non-pipeline bus protocol)에서 발생하여 낭비되는 클럭 사이클을 제거함으로써 대역폭 향상을 도모할 수 있다는 장점을 갖는다.[5] 다음 〈그림 3〉은 파이프라인이 적용된 버스 프로토콜의 파이프라인 다이어그램의 예를 보인다.

4. IP 재사용을 위한 Protocol Interface의 적용

〈그림 1〉에서 보여진 wrapper 로직은 버스와 코어 블록사이의 연결을 위하여 사용되고 있다. 이와 같이 다양한 컴포넌트들을 버스에 연결하기 위해서는 규정된 표준 인터페이스가 필요하며,



〈그림 3〉 Pipeline diagram of a pipelined bus protocol



〈그림 4〉 Virtual component interface and on-chip bus model

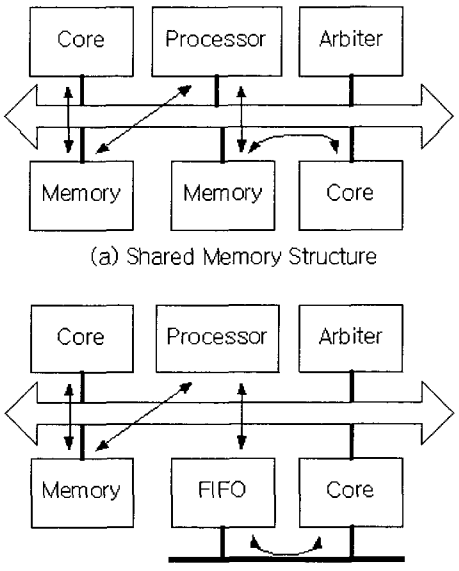
VSI(Virtual Socket Interface) Alliance는 wrapper 로직에 코어를 연결하는 Virtual Component(VC) interface에 대한 표준을 제안하고 있다.[4] VC interface는 두개의 VC들을 버스 인터페이스를 통해 연결할 수 있도록 하는 일을 수행한다. 〈그림 4〉는 버스와 콤포넌트를 연결하는 VC interface를 보이고 있다. VC interface와 같은 프로토콜 인터페이스는 IP 재사용을 위하여 각 IP가 규정된 인터페이스를 갖도록 설계되어야 한다는 것을 의미하고 이는 IP설계에 또 다른 제약을 가져오게 된다. 또한 다른 프로토콜을 사용하는 IP를 VC interface에 맞도록 변환하는 wrapper 로직의 설계는 시스템 성능과 칩 면적에 있어서의 오버헤드(overhead)로 나타나게 된다.

Ⅳ. On-Chip 통신망 설계의 고려사항

SoC을 위한 버스 구조에는 다양한 회로요소들이 포함되며 여러 가지 고려해야 할 사항들이 발생하게 된다. 이러한 고려사항들은 대부분 설계하고자 하는 시스템의 성능에 중점을 두어 이루어지고 있다. 다음은 버스 설계 시에 고려해야 할 몇 가지 사항을 살펴보고자 한다.

1. 메모리의 공유

정보의 전달은 일반적으로 메모리를 통해서 이루어지며 가장 단순하고 일반적인 구조는 모든 컴포넌트들이 하나의 메모리를 공유하는 것이다. 이러한 구조에서 메모리를 컴포넌트들 사이의 데이터 전송관계에 따라 영역을 구분하여 사용함으로써, 컴포넌트들 사이에 전송되는 데이터를 저



〈그림 5〉 Shared memory structure and memory structure converted into FIFO

장하는 커다란 저장 영역으로 사용할 수 있다. 이와 같은 메모리 공유 구조는 데이터 스트림을 저장하는 브리지 또는 FIFO로 구현될 수 있다. 즉, 만약 하나의 컴포넌트가 메모리에 데이터를 연속적으로 쓰고, 다른 컴포넌트가 그 데이터를 연속적으로 읽어간다면 〈그림 5〉에서 보이는 것처럼 그 메모리는 FIFO로 구현될 수 있다.

On-chip 공유 메모리의 사용은 off-chip 메모리에 비하여 제약점이 많다. 이 때문에 off-chip 메모리를 사용하게 된다. 그러나 이러한 off-chip 메모리의 사용은 on-chip 메모리보다 더 많은 클럭 사이클의 손실을 가져올 수 있다. 그러므로 데이터 전송 방식에 따른 메모리 구조의 결정이 필요하게 된다.[5]

2. FIFO 설계

FIFO는 두 블록간 데이터 전송을 위한 저장소로써 사용된다. 다양한 IP들이 서로 다른 데이터 전송 방식과 동작 주파수를 이용함에 따라, 각 IP 간의 원활한 데이터 전송을 위한 FIFO의 크기 및 데이터 제어 방식에 대한 고려가 FIFO 설계에 중요한 요소가 되었다. FIFO의 크기는 FIFO에 입력되고 출력되는 데이터 전송 특성에 따라 결정된다. FIFO의 크기는 일반적으로 전송될 전체 데이터의 크기를 D_{total}, 읽기 동작의 평균 횟수를 R, 쓰기 동작의 평균 횟수를 W, $W<R$ 라 할 때 다음과 같이 얻어질 수 있다.

$$Queue\ Size = D_{total} - \left[D_{total} \frac{R}{W} \right]$$

3. DMA와 브리지 구조

DMA 모듈은 하나의 컴포넌트에서 다른 컴포넌트로 정보를 전달하는 대행자의 역할을 수행한다. 이는 주로 메모리와 I/O 장치 사이에서 이루어지며 프로세서에서 DMA 모듈로 제어 신호가 연결된다. 이 제어 신호를 통해 프로세서는 DMA 모듈에 동작에 대한 정보를 전달하고 DMA 모듈은 버스를 통해 데이터 전송을 수행한다. DMA는 하나의 타깃에서 다른 타깃으로 데이터를 전달한다는 점에서는 브리지와 같은 기능을 수행한다. 그러나 DMA는 단지 데이터 전송을 제어하는 제어로직이라는 점이 다르다.

브리지는 버스의 크기와 속도가 서로 다른 버스들 사이에서 데이터 전송을 수행하며 데이터 전송을 동기화 하기 위한 저장장치를 가지고 있다. 브리지는 두 버스 사이의 성능과 데이터 폭의 차가 클수록 더 많은 저장장치를 필요로 하며 이 저장장치는 일반적으로 FIFO로 구현된다.

4. 단일 버스 구조와 계층적 버스 구조 (Flat and Hierarchical Bus Structure)

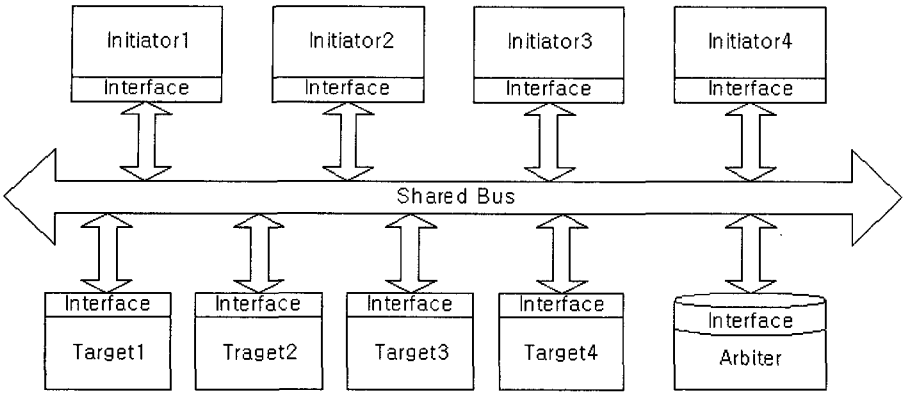
두 컴포넌트 사이의 데이터 전송에 있어서 지연이 작을 경우, 그 컴포넌트들은 하나의 단일 버스에 연결되는 것이 더 효율적일 수 있다. 그러나 컴포넌트들이 많아지면서 버스의 부하가 증가하게 되면 버스의 성능저하를 가져온다. 그러므로 이러한 문제를 해결하기 위해 부하를 분산시킬 수 있는 계층구조를 갖는 버스 구조를 사용하게 된다. 또한 버스에 있어서의 대역폭 제약도 계층적인 버스 구조를 사용하게 하는 원인이 된다.

V. 통신망 설계 동향

우리는 앞에서 일반적인 SoC 버스 구조와 설계 시 고려사항에 대하여 살펴보았다. 그러나 모든 시스템이 하나의 버스 구조에서 최상의 동작을 수행하지 않으므로 각 시스템의 특성에 따른 버스 구조를 선택하고 사용해야 한다. 다음은 현재 사용되고 있는 다양한 버스 구조에 대하여 설명한다.

1. 고정 우선순위 기반 공유 버스 (Static priority based Shared Bus)

중재 프로토콜(arbitration protocol)을 기반으로 하는 고정된 우선순위(priority)를 갖는 공유 버스 구조는 가장 일반적으로 사용되는 on-chip 버스 구조이다. 이 버스는 어드레스, 데이터, 그리고 제어 신호들로 구성된다. 중재기는 개

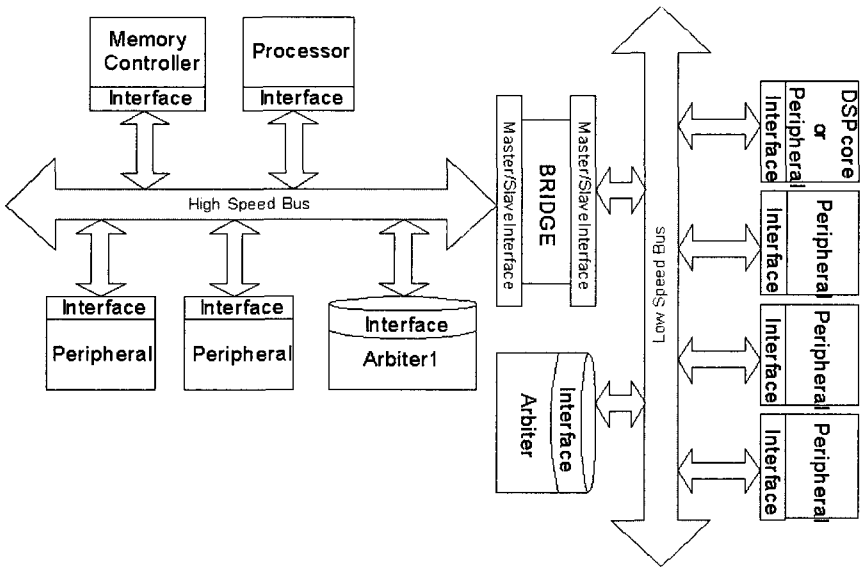


〈그림 6〉 하나의 버스를 공유하는 일반적인 버스 구조

시자들의 우선권에 따라서 버스를 사용할 수 있는 권한을 개시자들에게 부여하고, 버스 사용권한이 부여된 개시자는 버스를 통해 주변장치들을 제어하고 데이터를 송수신한다. 그러나 이러한 구조는 버스에 연결되는 장치들이 증가할수록 버스에 인가되는 부하가 커지게 되며, 이것이 버스의 성능저하를 발생시키는 원인이 된다. 이것을 해결하기 위해 계층구조를 갖는 버스 구조가 사용되고 있다. 〈그림 6〉은 하나의 버스를 공유하는 일반적인 버스 구조를 보인다.

2. 계층구조 버스(Hierarchical Bus)

계층구조를 갖는 버스 구조는 브리지에 의해 연결되는 다중 버스 구조로 나타난다. 일반적으로 두 계층으로 이루어진 버스 구조가 사용되고 있다. 이러한 버스 구조는 〈그림 7〉에서 보이는 것처럼 고속의 특징을 갖는 고속 버스와 저속의 특징을 갖는 저속 버스의 두 계층으로 나뉘며, 브리지의 인터페이스는 일반적으로 고속 버스와 연결되는 슬레이브 인터페이스와 저속 버스와 연결되는 마스터 인터페이스를 함께 갖는다. 고정된

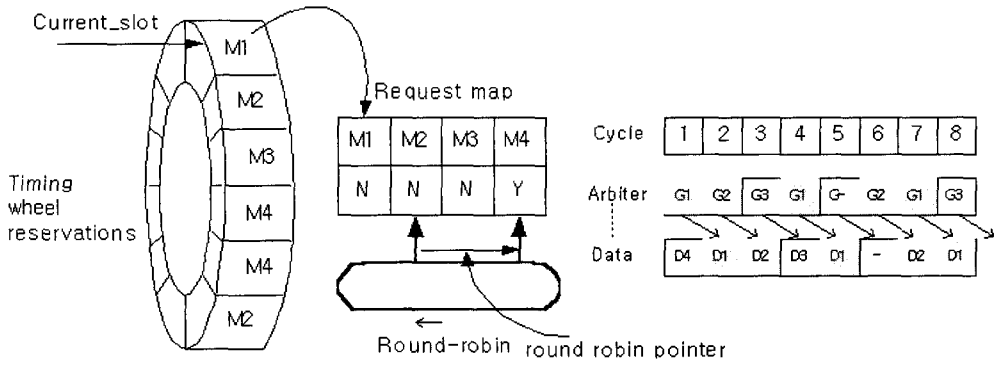


〈그림 7〉 계층구조를 갖는 버스 구조

우선순위에 기반한 중재기가 두 버스 각각에 따로 구현되고, 각 버스는 이 중재기에 의해 버스 액세스 권한을 부여받는다. 저속 버스에는 일반적으로 외부 회로와 데이터 전송을 담당하는 통신 블록들이 연결되고 이것들은 대부분 타깃의 기능을 수행한다. 이와 같은 경우 저속 버스는 중재기를 필요로 하지 않는다. 이 버스 구조에 있어서는 시스템을 이루는 각 컴포넌트들이 연결될 버스를 어떤 방법으로 구분하고 연결할 것인지에 대한 고려가 필요하며 브리지에서 구현되어야 할 버퍼의 크기 결정이 중요한 요소로가 된다. 이러한 구조는 ARM의 AMBA[7], IBM의 Core-Connect[8] 버스 구조에서 잘 나타난다. 또한 Philips의 멀티미디어 플랫폼인 Nexperia 플랫폼에서도 이러한 구조를 사용하고 있다. 이 플랫폼은 두 개의 서로 다른 프로세서 코어(MIPS와 DSP core)가 각각의 독립적인 버스를 사용하고 그 두 개의 버스가 브리지로 연결된 구조를 가지고 있으며, 이 구조에서는 주변장치들의 속도측면에서 두 버스를 구분하지 않으며, 어떤 면에서는 서로 다른 구조의 공유 버스를 브리지로 연결한 것과 같은 구조를 갖는다.

3. 2단계 TDMA기반 버스 구조(Two-level TDMA-based Bus Architecture)

세 번째 버스 구조는 시분할 다중접속 방법에 기반하고 있으며 모든 콤포넌트들은 단일 통신망에 연결된다. 이 구조에 있어서 각 컴포넌트는 두 단계의 중재 프로토콜에 의한 인터리브(interleave)한 방법에 의해서 버스를 액세스할 수 있다. 〈그림 8〉의 (a)에서 보이는 것과 같이 중재 프로토콜의 첫 번째 단계는 타이밍 휠(timing wheel)을 사용한다. 여기서 각 개시자는 타이밍 휠의 슬롯(slot)에 고정적으로 예약된다. 타이밍 휠의 동작에 있어서, 하나의 슬롯 보다 많은 슬롯에 예약된 개시자는 다른 개시자들에 비하여 더 많은 시간동안 버스를 액세스할 수 있는 권한을 부여받는다. 만약 지정된 타임 슬롯에 해당하는 개시자가 버스사용을 요구한다면 그 개시자는 지정된 슬롯 타임에 해당하는 단일 워드 전송을 수행하게 되고, 타이밍 휠은 다음 슬롯으로 이동하게 된다. 그러나 이와 같은 방법은 타임 슬롯에 할당된 개시자가 버스의 사용을 요구하지 않음에도 불구하고 그에 해당하는 타임 슬롯을 낭비해야 하는 문제점을 갖는다. 이러한 문제점을 해결하기 위하여 중재 프로토콜의 두 번째 단계로써 라운드 로빈(round robin) 방법을 사용한다. 이것은 중재 프로토콜의 두 번째 단계에 의해 버스 액세스 권한이 부여될 때까지 타이밍 휠의 타임 슬롯에 지정된 개시자를 유지하다가, 라운드 로빈 방법에 의하여 버스사용을 요청한 다른 개시자로 사용 권한을 넘기는 방법이다. 예로써 현재 버스를 점유하고 있는 개시자는 타이밍 휠에 예약된 M1 마스터 코어이다. 그러나 그것이 전송할 데이터



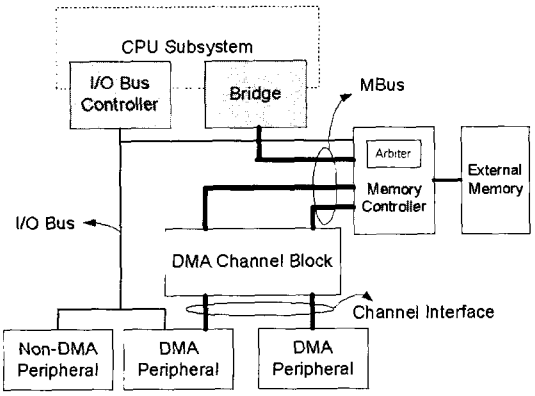
(a) TDMA time wheel and round-robin scheme (b)Pipelined word level arbitration

〈그림 8〉 Two level TDMA 기반 버스 구조

를 갖지 않을 경우 두 번째 단계의 라운드 로빈 스케줄러(scheduler)가 라운드 로빈 포인터(round robin pointer)를 버스사용을 요청하는 마스터 코어 M4로 이동시키고 중재기는 M4에 버스 사용권한을 부여한다. 이와 같은 2단계 중재 프로토콜을 사용함으로써 타임 슬롯의 낭비를 줄일 수 있다. 이와 더불어 〈그림 8〉의 (b)에서와 같이 파이프라인 방법이 데이터 전송을 위해 적용된다. 이러한 구조는 Sonics, Inc의 MicroNetwork에서 사용되는 구조이다.[6][9]

4. 메모리 버스와 I/O 버스 분리 구조 (Isolated Memory & I/O Bus Architecture)

이 구조는 Palmchip사의 CoreFrame 구조에서 사용하고 있는 것으로써 메모리 버스와 I/O 버스를 분리한 것을 특징으로 한다. 저속 데이터 전송은 I/O 버스를 통해서, DMA 연결과 고속 데이터 전송을 요구하는 블록은 메모리 버스(MBus)를 통해서 데이터를 전송한다. 여기서 I/O 버스와 메모리 버스는 I/O 버스 콘트롤러와 메모리 콘트롤러에 의해 제어된다. I/O 버스는 CPU와 주변장치들 간의 통신을 담당하며 메모리 버스에 연결된 고속 데이터 전송 특성을 갖는 블록들의 제어를 위하여 사용된다. 이 버스는 단일 마스터에 의한 master-slave interface를 갖는다. Mbus는 메모리 제어기(memory controller)와 DMA 채널 블록 사이의 연결을 담당한다.[2]



〈그림 9〉 Isolated memory & I/O bus architecture

VI. 결 론

지금까지 IBM의 CoreConnect, ARM의 AMBA, Palmchip의 CoreFrame과 같은 on-chip 버스들의 일반적인 구조를 살펴보았다. 이외에도 더욱 다양한 버스 구조들이 현재 연구되고 있으며 SoC 플랫폼에 사용되고 있다. 버스 구조에 따른 시스템 성능은 설계하고자 하는 시스템의 특성에 따라 많은 차이를 나타내며, 이는 메모리 액세스뿐만 아니라 각 주변장치들 및 IP 코어의 실시간 수행 특성에 따라서도 차이를 나타내게 된다. 따라서 설계자는 자신이 원하는 성능을 최대한 발휘할 수 있도록 하는 버스 구조를 선택해야 한다.

한편 버스 구조의 데이터 전송방식을 네트워크, 즉 통신 채널에 의한 데이터 패킷전송 방식으로 변환하려는 노력이 시도되고 있다. 이러한 구조는 bit나 word 단위의 작은 기능 블록들의 연결이 아닌 packet 단위의 시스템 블록간 연결을 위한 구조라 할 수 있다. 이러한 구조는 버스기반의 통신 구조에 있어서의 물리적 버스 영역의 크기 증가에 의한 회로 설계 영역의 손실을 통신 채널을 사용함으로써 줄일 수 있다는 이점을 가지며, 이 구조에 맞게 설계된 시스템을 채널에 연결만 함으로써 추가적인 회로의 설계 없이 전체 시스템을 구성할 수 있다는 장점을 갖는다. 그러나 이 방법은 모든 IP 블록들에 데이터 패킷을 처리하기 위한 채널 코딩 블록, 패킷 검출 블록과 같은 모뎀 블록이 포함되어야 하고, 채널 라우팅(channel routing) 회로 즉, 스위칭 회로가 필요하게 됨으로써 부가 회로의 추가 설계에 따른 부담이 증가하게 된다. 또한 프로세서 기반의 시스템에 사용되는 인터럽트 처리와 같은 제어 신호의 처리가 어렵다는 문제점도 갖고 있다.

시스템의 설계, 검증 시간 그리고 성능을 희생하지 않으면서도 기존에 설계되어 검증된 IP들을 효율적으로 사용할 수 있도록 하는 환경을 제공하는 on-chip 통신 구조의 개발과 더불어 시스템 특성을 반영한 정확한 시스템 레벨 성능 측정

방법의 개발은 SoC 설계에 있어서 보다 빠르고 효율적인 시스템 설계를 가능하게 할 것이다.

참 고 문 헌

[1] S.Kumar et al., "A Network on Chip Architecture and Design Methodology," Annual Symposium on VLSI, pp.105-112, 2002.

[2] B. Cordan, "An efficient bus architecture for system-on-chip design," in Proc. IEEE Custom Integrated Circuits Conf., pp.623-626, May 1999.

[3] Virtual Socket Interface (VSI) Alliance : http://www.vsi.org.

[4] Open Core Protocol (OCP) International Partnership : http://www.ocpip.org.

[5] Henry Chang, Larry Cooke, Merrill Hunt, Grant Martin, Andrew McNelly, and Lee Todd, Surviving the SOC Revolution : A Guide to Platform-based Design, Kluwer Academic Publishers, 1999.

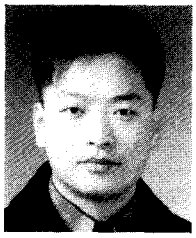
[6] D. Wingard and A. Kurosawa, "Integration Architecture for System-on-a-Chip Design," in Proc. IEEE 1998 Custom Integrated Circuits Conf., pp. 85-88, May 1998.

[7] ARM, Limited, AMBA Specification, Revision 2.0, May 1999, available from http://www.arm.com.

[8] IBM, CoreConnect Bus Architecture, 1999, available from http://www.chips.ibm.com/products/coreconnect.

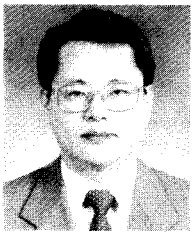
[9] D. Wingard, "MicroNetwork-Based Integration for SOCs," in Proc. the 38th Design Automation Conference, pp. 673-677, Jun. 2001.

저 자 소 개



천 익 재

1998년 2월 충남대학교 전자공학과(공학사), 2000년 2월 충남대학교 전자공학과(공학석사), 2000년 3월~현재 : 충남대학교 전자공학과 박사과정, 〈주관심 분야 : VLSI & CAD, Communication system design, Digital system architecture〉



김 보 관

1976년 2월 서울대학교 전자공학과(공학사), 1978년 2월 한국과학기술원 전기및전자공학과(공학석사), 1989년 2월 University of Wisconsin-Madison 전자 및 컴퓨터공학과(공학박사), 1978년 3월~1980년 2월 : 한국과학기술연구소 연구원, 1980년 3월~1991년 2월 : 금오공대 조교수, 1991년 3월~현재 : 충남대학교 정보통신공학부 교수, 〈주관심 분야 : VLSI & CAD, Digital system design and modeling, HW/SW Co-design〉