(19)대한민국특허청(KR)
(12) 등록특허공보(B1)

(51) 。Int. Cl.[7]
G06F 15/16

(45) 공고일자 2003년06월18일
(11) 등록번호 10-0387541
(24) 등록일자 2003년06월02일

(21) 출원번호 10-2000-0050729
(22) 출원일자 2000년08월30일

(65) 공개번호 특2001-0050269
(43) 공개일자 2001년06월15일

(30) 우선권주장 09/392,833 1999년09월09일 미국(US)

(73) 특허권자 인터내셔널 비지네스 머신즈 코포레이션
미국 10504 뉴욕주 아몬크

(72) 발명자 데시팬드산재이라군나스
미국78726텍사스주오스틴세자네스트리트11209

찬티나슈이원
미국78729텍사스주오스틴파트리지벤드드라이브13009

(74) 대리인 김창세
장성구
김원준

심사관 : 신준호

(54) 멀티프로세서 시스템에서 캐쉬 코히어런시 유지 방법, 멀티프로세서 시스템 및 노드 제어기

요약

본 발명에서는 버스-기반 캐쉬-코히어런스 프로토콜(a bus-based cache-coherence protocol)을 이용하는 대규모 대칭형 멀티프로세서 시스템(a large-way symmetric multiprocessor system)을 위한 분산 시스템 구조(a distributed system structure)가 제공된다. 분산 시스템 구조는 어드레스 스위치(address switch)와, 다중 메모리 서브시스템(multiple memory subsystem)과, 다중 마스터 디바이스(multiple master devices), 노드 제어기(node controller)에 의해 지원되는 일련의 노드로 구성된 프로세서, I/O 에이전트(I/O agent) 또는 코히어런트 메모리 어댑터(coherent memory adapter)를 포함한다. 노드 제어기는 마스터 디바이스로부터 커맨드(command)를 수신하고, 또 다른 마스터 디바이스 또는 슬레이브 디바이스(slave device)로서 마스터 디바이스와 통신하고, 마스터 디바이스로부터 수신된 커맨드를 큐(queue)한다. 노드 제어기는 어드레스 스위치에 의해 브로드캐스트(broadcast)된 스누프(snoop)된 커맨드를 래치하는 시점과 그 커맨드를 노드 제어기의 마스터 디바이스 버스에 실어 마스터 디바이스에게 제공하는 시점사이에서 결정적인 지연(a deterministic delay)을 갖는다. 코히어런시(coherency)는 시간적으로나 공간적으로 분산되어 성취되기 때문에, 노드 제어기는 그 노드 제어기내에 큐된 커맨드의 타입(type)과 커맨드의 상태(phase)에 근거하여, 하나 이상의 입력을 커맨드에 대한 코히어런시 응답을 결정하는 데 부여함으로써 커맨드에 대한 캐쉬 코히어런시를 유지하는데 도움을 준다. 응답 조합 블록은 마스터 디바이스에 의해 발행된 커맨드와 연관된 커맨드 상태 신호와 커맨드 응답 신호를 논리적으로 조합하여 만들어낸 다음 전송한다. 본 시스템은 이들 코히어런시 입력을 이용하여 트랜잭션에 대한 정확한 완료 순서를 성취할 수 있다.

대표도

도 4

명세서

도면의 간단한 설명

도 1은 통상의 멀티프로세서 컴퓨터 시스템의 기본 구성을 도시하는 블록 구성도,
도 2는 전형적인 구조를 도시하는 블록 구성도,
도 3은 세 개의 프로세싱 유닛을 갖는 SMP 컴퓨터 시스템을 도시하는 블록 구성도,
도 4는 SMP 시스템내의 어드레스 경로(address path)의 관점으로 바라다본 보조(supporting) 버스-기반 캐쉬-코히어런스 프로토콜(bus-based cache-coherent protocol)을 가진 대칭형 멀티프로세서 시스템의 분산 시스템 구조를 예시하는 블록 구성도,
도 5는 SMP 시스템내 데이터 경로(data path)의 관점으로 바라다본 보조 버스-기반 캐쉬-코히어런스 프로토콜을 가진 대칭형 멀티프로세서 시스템의 분산 시스템 구조를 예시하는 블록 구성도,
도 6은 노드 제어기 내부의 어드레스 경로를 도시하는 블록 구성도,
도 7은 노드 제어기와 메모리 서브시스템을 연결하는 어드레스 스위치의 내 부 어드레스 경로를 도시하는 도면,
도 8은 본 발명의 분산 시스템의 어드레스 스위치에 접속된 메모리 서브시스템을 도시하는 도면,
도 9a 및 도 9b는 노드 제어기 내부의 데이터 경로를 도시하는 블록 구성도,
도 10a-10b는 분산 시스템 구조에 필요한 버스 응답 신호를 결정하는 시스템 구조를 도시하는 블록 구성도,
도 10c-10d는 신호가 로컬 사이클(local cycle)과 글로벌 사이클(global cycle)에 관여하는 구성요소를 도시하는 블록 구성도,
도 11은 본 시스템내의 트랜잭션의 상태(phase)를 정의하는 테이블,
도 12a-12b는 충돌 트랜잭션 쌍의 검출에 응답하여 노드 제어기에 의해 생성된 응답을 예시하는 테이블.

도면의 주요 부분에 대한 부호의 설명

100 : 멀티프로세서 컴퓨터 시스템 112 : 프로세싱 유닛
114 : I/O 에이전트 120 : 버스
122 : 프로세서 124 : 명령어 캐쉬
126 : 데이터 캐쉬 410, 420, 510, 520 : 노드
411, 412, 421, 511, 512, 521 : 프로세서
413, 414, 416, 417, 423, 424, 513, 514, 523, 524 : 버스
415, 425, 515, 525 : 노드 제어기 430 : 어드레스 스위치
442, 444 : 메모리 서브시스템

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

본 발명은 전반적으로 개선된 데이터 프로세싱 시스템에 관한 것으로, 특히 데이터 프로세싱 시스템내에서 데이터 처리량(data throughput)을 개선시킨 방법 및 시스템에 관한 것이다. 보다 상세하게는, 본 발명은 캐쉬-코히어런스(cache-coherence)를 이용하여 저장장치의 액세스 및 제어의 성능을 개선시킨 방법 및 시스템에 관한 것이다.
통상적으로, 대칭형 멀티프로세서는 모든 프로세서들과 메모리 및 I/O와 같은 다른 디바이스들이 버스 신호를 전달하는 와이어(wire)와 단순한 물리적 접속을 이룸으로써 연결되는 공통 시스템 버스(common system bus) 주변에 설계된다. 이러한 공통 버스는 디바이스들간의 커맨드(command)와 데이터를 전달하고 또한 시스템의 캐쉬와 메모리들 사이에서 코히어런스를 성취하기 위한 경로이다. 단일의 공통 버스 설계(a single-common-bus design)는 시스템 구성의 간결함 때문에 멀티프로세서의 연결을 위해 대중적으로 선택되는 설계로 지속되고 있다.
이러한 구성은 또한 시스템의 캐쉬들 사이에서 코히어런스를 성취하는 작업을 단순화시켜주기도 한다. 디바이스에 의해 발행된 커맨드는 그 커맨드가 버스에 실리는 그 동일 클럭 사이클 내에 동시적으로 다른 모든 시스템 디바이스들에게 브로드캐스트된다. 버스는 그 버스에 실리는 모든 커맨드가 일정 순서를 이루게 한다. 이러한 순서는 시스템내 모든 디바이스들이 동일한 커맨드를 준수하기 때문에 이들에게 합의되어 있다. 또한 디바이스들은 특별한 대가없이 커맨드 순서의 최종 결과를 따를 수 있기도 하다. 이것은 단일 버스 기반(a single-bus-based) 멀티프로세서의 중요한 장점이다.
그러나, 단일 공통 버스 설계는 시스템의 성능을 낮추어 선택하지 않는다면 시스템의 크기를 제한한다. 전형적으로 기술적인 제한은 버스가 스위치하는 속도와 시스템이 구동하는 속도를 손상시키지 않으면서 단지 몇 개 디바이스만이 버스에 연결되게 한다는 것이다. 프로세서 및 I/O 에이전트(I/O agent)와 같은 더 많은 마스터 디바이스가 버스에 배치된다면, 버스는 더욱 느려진 속도로 스위치 하여야 하고, 이로써 버스의 이용가능한 대역폭은 더욱 낮아진다. 대

역폭이 낮아지면 큐잉 지연(queuing delay)을 증가시킬 수도 있어서, 결국 프로세서의 이용성을 저하시키고 시스템 성능을 저하시킨다.
단일 버스 시스템의 또 다른 단점은 데이터의 전달을 위한 단일 데이터 경로의 유용성이다. 이것은 큐잉 지연을 더욱 악화시키고 시스템 성능을 더 낮아지게 하는 요인이 된다.
두 가지의 넓은 부류의 캐쉬-코히어런스 프로토콜이 존재한다. 그 하나는 버스 기반 스누핑 프로토콜(bus-based snooping protocol)로서, 이 프로토콜에서는 시스템내 모든 캐쉬들이 공통 버스에 연결되어 다른 캐쉬들에 의해 공통 버스에 발행된 트랜잭션을 스누핑한 다음 서로 코히어런시를 지속시키도록 적절한 조치를 취한다. 다른 부류는 각각의 메모리 어드레스가 "홈(home)" 사이트(site)를 가지고 있는 디렉토리 기반 프로토콜(directory-based protocol)이다. 캐쉬가 그 어드레스를 액세스할 때마다, 홈 사이트에 있는 "디렉토리"는 캐쉬의 정체(a cache's identity)와 캐쉬내 데이터의 상태를 저장하도록 갱신된다. 그 캐쉬내 데이터의 상태를 갱신하는 것이 필요할 때, 홈 사이트는 캐쉬에게 메시지를 전달하여 적절한 조치를 취하도록 요청한다.
구현과 검증의 복잡성의 관점에서 보면, 버스 기반 스누핑 프로토콜은 디렉토리 기반 프로토콜보다 상당히 단순하며 대칭형 멀티프로세서((symmetric multiprocessor)(SMP) 시스템이 선택할 수 있는 프로토콜이다. 그러나, 버스 기반 스누핑 프로토콜은 소수개의 프로세서, 통상 2 내지 4개의 프로세서만을 갖춘 시스템에서 효과적으로 이용된다.
그래서, 비록 단일 시스템 버스 설계가 현재 코히어런스 프로토콜을 구현하는 데 선호하는 선택할만한 설계일지라도, 많은 프로세서를 갖춘 대규모 SMP(a large-way SMP)에는 이용될 수 없다.
그러므로, 버스 기반 캐쉬-코히어런스 프로토콜을 이용하여 대규모 SMP를 설계하는 것이 바람직할 것이다.

발명이 이루고자 하는 기술적 과제

그러므로, 본 발명은 버스 기반 캐쉬-코히어런스 프로토콜을 이용하여 대규모 대칭형 멀티프로세서 시스템용 분산 시스템 구조를 제공하는 것을 그 목적으로 한다.
분산 시스템 구조는 어드레스 스위치와, 다중 메모리 서브시스템과, 다중 마스터 디바이스 및 노드 제어기에 의해 지원된 일련의 노드로 구성된 프로세서, I/O 에이전트 또는 코히어런트 메모리 어댑터를 포함한다. 노드 제어기는 마스터 디바이스로부터 커맨드를 수신하고, 또 다른 마스터 디바이스 또는 슬레이브 디바이스로서 마스터 디바이스와 통신하고, 마스터 디바이스로부터 수신된 커맨드를 큐한다. 노드 제어기는 어드레스 스위치에 의해 브로드캐스트된 스누프된 커맨드를 래치하는 시점과 그 커맨드를 노드 제어기의 마스터 디바이스 버스에 실어 마스터 디바이스에게 제공하는 시점사이에서 결정적인 지연(a deterministic delay)을 갖는다. 코히어런시는 시간적으로나 공간적으로 분산되어 성취되기 때문에, 노드 제어기는 그 노드 제어기내에서 큐된 커맨드의 타입(type)과 커맨드의 상태(phase)에 근거하여, 하나 이상의 입력을 커맨드에 대한 코히어런시 응답을 결정하는 데 부여함으로써 커맨드에 대한 캐쉬 코히어런시를 유지하는데 도움을 준다. 응답 조합 블록은 마스터 디바이스에 의해 발행된 커맨드와 연관된 커맨드 상태 신호와 커맨드 응답 신호를 논리적으로 조합하여 만들어낸 다음 전송한다. 본 시스템은 이들 코히어런시 입력을 이용하여 트랜잭션에 대한 정확한 완료 순서를 성취할 수 있다.

발명의 구성 및 작용

본 발명의 신규한 특징은 첨부된 청구범위에서 설명된다. 그러나, 본 발명 그 자체 뿐 아니라 바람직한 이용 모드의 목적 및 장점은 첨부된 도면을 참조한 예시적인 실시예의 상세한 설명을 참조하여 가장 잘 이해될 것이다.
도 1을 참조하면, 통상의 멀티프로세서 컴퓨터 시스템(110)의 기본 구조가 도시된다. 컴퓨터 시스템(110)은 여러 프로세싱 유닛(112a,112b,112c)을 구비하며, 이들 프로세싱 유닛은 모니터 어댑터(102)와 디스플레이 모니터(105), 키보드 어댑터(104)와 키보드(107) 및 디스크 어댑터(103)와 영구 저장 디바이스(106)로부터 데이터를 주고 그곳으로 데이터를 제공하는 입력/출력 (I/O) 에이전트(114), 프로세싱 유닛에 의해 사용되어 프로그램 명령어(program instructions)를 전달하는 (다이내믹 랜덤 액서스 메모리, 즉 DRAM과 같은) 메모리 디바이스(116) 및 펌웨어(118)와 같은 각종 주변 디바이스에 연결되어 있다. 여기서, 펌웨어(firmware)(118)의 기본적인 목적은 컴퓨터가 턴온(turn-on)될 때마다 주변장치들 중의 하나(통상 영구 메모리 디바이스)로부터 오퍼레이팅 시스템을 찾아 로딩하는 것이다. 프로세싱 유닛(112a - 112c)은 버스(120)를 포함하는 각종 수단을 통하여 주변 디바이스들과 통신한다. 컴퓨터 시스템(110)은 도시되지는 않았지만, 모뎀 또는 프린터와 같은 주변 디바이스들과의 연결을 위한 직병렬 포트와 같은 많은 부가적인 구성요소를 가질 수도 있다. 본 기술분야에서 통상의 지식을 가진 자에게는 도 1의 블록 구성도에 도시된 구성요소들과 관련하여 사용될 수 있는 다른 구성요소들이 있음을 인식할 것이다. 예를 들면, 디스플레이 어댑터는 비디오 디스플레이 모니터를 제어하는데 사용될 수 있고, 메모리 제어기는 메모리(116)를 액세스하는데 사용될 수 있다. 부가적으로, 컴퓨터 시스템(110)은 더 많은 또는 더 적은 수의 프로세서로 구성될 수도 있다.
대칭형 멀티프로세서(SMP) 컴퓨터에서, 통상적으로 프로세싱 유닛(112a - 112c)은 모두 동일하다. 즉, 이들은 모두 공통의 세트 및 서브세트의 명령어 및 프로토콜을 사용하여 구동되고 대체로 동일한 아키텍처(architecture)를 가지고 있다.
이제 도 2를 참조하면, 전형적인 구조가 도시된다. 프로세싱 유닛(112)은 컴퓨터를 구동시키기 위하여 프로그램 명령어를 실행하는 다수의 레지스터들과 실행 유닛들을 가지고 있는 프로세서(122)를 포함한다. 또한 프로세서(122)는 명령어 캐쉬(124)와 데이터 캐쉬(126)와 같은 캐쉬를 가질 수 있다. 이들 캐쉬는 프로세서의 레지스터와 실행 유닛과 함께 일체로 패키지되어 있을 때 "온-보드(on-board)"라고 지칭된다. 통상적으로, 캐쉬는 프로세서에 의해 반복적으로 액세스될 수도 있는 값을 일시적으로 저장하는 데 사용됨으로써, 도 1에 예시된 메모리(116)와 같은 메모리로부터

그 값을 로딩하는 오랜 과정을 없애 처리 속도를 높여줄 수 있다.
프로세싱 유닛(112)은 캐쉬(128)와 같은 부가적인 캐쉬를 포함할 수 있다. 캐쉬(128)는 온-보드 (레벨 1) 캐쉬(124, 126)를 지원하기 때문에 레벨 2(L2) 캐쉬라 지칭된다. 환언하면, 캐쉬(128)는 메모리(116)와 온-보드 캐쉬들 사이에서 중재자로서의 역할을 수행하며, 비록 액세스가 길다는 단점에도 불구하고 온-보드 캐쉬보다 훨씬 더 많은 양의 정보(명령 및 데이터)를 저장할 수 있다. 예를 들면, 캐쉬(128)는 256 또는 512 킬로바이트의 저장 용량을 가진 칩일 수 있으며, 프로세서(112)는 총 저장용량이 64 킬로바이트인 온-보드 캐쉬를 갖는 IBM Power PC 604 계열 프로세서일 수 있다. 캐쉬(128)는 버스(120)에 연결되며, 메모리(116)로부터 프로세서(112)로 정보를 로딩하는 모든 행위는 캐쉬(128)를 통하여 이루어진다. 비록 도 2에서는 2단계 캐쉬 계층구조만을 도시하고 있지만, 캐쉬를 일렬로 여러 단계 연결시킨 다단 캐쉬 계층구조가 제공될 수 있다.
SMP 컴퓨터에서는 코히어런트 메모리 시스템을 제공하는 것이 중요하다. 즉, 모든 프로세서가 어떤 순서대로 연속성을 가지고 개개의 메모리 장소(memory location)에 기록하게 해주는 것이 중요하다. 예를 들면, 메모리내의 한 장소가 연속적인 기록에 의해 어떤 값 1, 2, 3, 4를 갖는 것으로 변경되었다고 가정해보자. 캐쉬-코히어런트 시스템에 있어서, 모든 프로세서는 소정 장소에 기록된 값이 예시된 순서대로 이루어져 있는 지를 관찰할 것이다. 그러나, 프로세싱 소자가 메모리 장소에 기록하는 것을 빠뜨릴 수 있다. 그 메모리 장소를 판독하는 소정의 프로세싱 요소가 값 2로의 갱신을 빠뜨린 시퀀스 1, 3, 4를 인지할 수 있다. 각각의 프로세서가 유효 데이터 순서(valid data order)를 가지고 있음을 보장하는 시스템은 "코히어런트(coherent)"로 불린다. 실제로 모든 코히어런시 프로토콜은 캐쉬 블록의 크기의 그래뉼러티(granularity)에 대해서만 작동한다는 사실을 인식하는 것은 중요하다. 즉, 코히어런시 프로토콜은 데이터의 기록 허용 동작을 캐쉬 블록 단위로(the movement of the write permissions for data on a cache block basis) 제어하는 것이지, 개개의 메모리 장소마다 개별적으로 제어하는 것은 아니다.
본 기술분야에서 통상의 지식을 가진 자들에게 알려진 캐쉬 코히어런스를 성취하는 많은 프로토콜과 기술이 있다. 코히어런시를 유지하는 이들 모든 메커니즘의 중추에는 프로토콜이 단지 하나의 프로세서에게만 어느 주어진 시점에서 주어진 메모리 장소(캐쉬 블록)에 기록을 허락하는 "허용(permission)"을 제공해주는 요건이 있다. 이러한 요건의 결과로서, 프로세싱 요소가 메모리 장소에 기록하고자 할 때마다, 프로세싱 요소는 먼저 다른 모든 프로세싱 요소들에게 그 메모리 장소에 기록하려 한다는 자신의 희망을 알려주고 다른 모든 프로세싱 유닛들로부터 그 기록 명령을 실행하도록 하는 허용을 받아야만 한다. 중요한 논점은 시스템내 다른 모든 프로세서들은 개시 프로세서(the initiating processor)에 의해 기록 명령을 전달받은 다음에 그 기록을 실행하여야한다는 것이다. 캐쉬 코히어런스가 다단 계층구조에서 어떠한 방식으로 구현되는 지를 보다 상세히 설명하기 위하여, 도 3을 참조하자.
이제 도 3을 참조하면, SMP 컴퓨터 시스템은 프로세서(140a, 141a, 142a)로 이루어진 세 개의 프로세싱 유닛(140, 141, 142)을 갖는 것으로 도시되며, 프로세서들은 각기 L1 캐쉬(140b, 141b, 142b), L2 캐쉬(140c, 141c, 142c) 및 L3 캐쉬(140d, 141d, 142d)를 가지고 있다. 이러한 계층 구조에서, 각각의 캐쉬는 저단(즉 L3 캐쉬는 L2 캐쉬보다 저단이다)으로 갈수록 크기가 커지고 그보다 고단의 캐쉬보다 액세스 시간이 길어진다. 더욱이, 비록 절대적으로 요구된 것은 아니지만, 저단 캐쉬는 고단 캐쉬에 존재하는 모든 블록의 사본들(copies)을 가지고 있다. 예로, 어떤 블록이 소정 프로세싱 유닛의 L2 캐쉬 내에 존재하는 경우, 이것은 프로세싱 유닛의 L3 캐쉬 역시 그 블록의 (어쩌면 오래된) 사본을 가지고 있다는 것을 의미한다. 또한, 어떤 블록이 소정 프로세싱 유닛의 L1 캐쉬 내에 존재하는 경우, 그 블록은 그 프로세싱 유닛의 L2 및 L3 캐쉬 내에도 존재한다. 이러한 특성은 포함(inclusion)으로 알려져 있으며 본 기술분야의 통상의 지식을 가진 자에게도 알려져 있다. 이후, 포함의 원리는 본 발명과 관련된 캐쉬에 적용된다고 가정한다.
도 3은 시스템에서 캐쉬 코히어런시를 구현하는 것을 도시하는 것으로, 프로세서들은 공통의 범용 상호접속부(a common generalized interconnect)(143)를 통하여 통신한다. 프로세서들은 메모리 장소를 판독 또는 기록하려는 자신의 희망을 나타내는 메시지를 상호접속부를 통하여 전달한다. 소정의 오퍼레이션(operation)이 상호접속부에 실릴 때, 다른 모든 프로세서들은 이 오퍼레이션을 "스누프(snoop)"하고 자신의 캐쉬의 상태가 그 요청된 오퍼레이션을 진행하도록 할 수 있는지, 그러하다면, 어떠한 조건 하에서 가능한지를 결정한다. 캐쉬를 가지고 있는 시스템에서 소정 메모리 블록의 가장 최근의 유효 사본이 시스템 메모리(144)로부터 시스템내 하나 이상의 캐쉬들로 옮겨졌을 수도 있기 때문에 이러한 통신이 필요하다. 프로세서(예로, 140a)가 자신의 캐쉬 계층(140b, 140c, 140d)내에 존재하지 않는 메모리 장소를 액세스하고자 하면, 그 메모리 장소의 실제값을 가지고 있는 블록의 정본(the correct version)이 시스템 메모리(144)내에 있을 수 있거나 프로세싱 유닛(141,142)내 어느 한 캐쉬 내에 있을 수 있다. 그 정본이 시스템내 다른 캐쉬들 중의 한 캐쉬에 있다면, 시스템 메모리 대신 시스템내 캐쉬로부터 그 정본을 구하는 것이 필요하다.
예를 들어, 프로세서(예로, 140a)가 메모리내 장소를 판독하고자 한다고 가정하자. 프로세서는 먼저 자신의 L1 캐쉬(140b)를 폴링(polling)한다. 그 블록이 L1 캐쉬(140b)내에 존재하지 않으면, 메모리 장소 판독 요청이 L2 캐쉬(140c)로 전달된다. 그 블록이 L2 캐쉬(140c)내에 존재하지 않으면, 그 요청은 L3 캐쉬(140d)로 전달된다. 그 블록이 L3 캐쉬(140d)내에 존재하지 않으면, 그 요청은 범용 상호접속부(143)에 실려 전달된다. 일단 오퍼레이션(operation)이 범용 상호접속부에 실리면, 다른 모든 프로세싱 유닛들은 그 오퍼레이션을 "스누프"하고 그 블록이 자신의 캐쉬 내에 존재하는 지를 판단한다. 소정 프로세싱 유닛(예로, 142)이 프로세싱 유닛(140)에서 요청한 데이터 블록을 자신의 L1 캐쉬(142a)내에서 가지고 있고, 그 데이터가 포함 원리에 따라 수정되어 있다면, L2 캐쉬(140c)와 L3 캐쉬(142d)도 역시 그 블록의 사본을 가지고 있다. 그러므로, 프로세싱 유닛(142)의 L3 캐쉬(142d)가 판독 오퍼레이션을 스누프할 때는 요청된 블록이 L3 캐쉬(142d)내에 존재하고 수정되어 있는 지를 판단할 것이다. 이렇게 되었을 때, L3 캐쉬(142d)는 메시지를 범용 상호접속부에 실어 프로세싱 유닛(140)에게 판독 오퍼레이션에 필요로 하는 메모리 장소의 최근 갱신된 값이 메인 메모리(144)의 밖에 있는 L3 캐쉬(142d)내에 있으며, 프로세싱 유닛(140)의 판독 요청에 이용가능토록 하기위한 조치가 취해져야 하므로 나중에 그 오퍼레이션을 다시 한번 "재시도(retry)"하여야 한다고 알려준

다.
L3 캐쉬(142d)는 수정된 데이터를 L3 캐쉬로부터 메인 메모리(144)로 푸시(push)하는 프로세스를 시작할 수 있다. 그 다음에 메모리 장소에 대한 최근 갱신된 값이 다른 프로세서들에게도 유용하게 된다.
이와 달리, "개입(intervention)"이라 지칭되는 프로세스에 있어서, L3 캐쉬(142d)는 메모리 장소의 최근 갱신된 값을 그것을 요청한 프로세싱 유닛(140)에게 직접 보낼 수도 있다. 그 다음에 L3 캐쉬(142d)는 수정된 데이터를 L3 캐쉬(142d)로부터 메인 메모리로 푸시하는 프로세스를 시작할 수도 있다. 프로세싱 유닛(140), 특히 그의 L3 캐쉬(142d)는 궁극적으로 범용 상호접속부상에 판독 요청을 제시한다. 그러나, 이 시점에서, 수정된 데이터는 프로세싱 유닛(142)의 L1 캐쉬로부터 인출되어졌고 프로세서(140)로부터의 판독 요청은 만족될 것이다. 이와 같이 기술된 시나리오는 통상 "스누프 푸시(snoop push)"라 지칭된다. 판독 요청은 범용 상호접속부상에서 스누프되어, 프로세싱 유닛(142)이 그 블록을 계층의 하단으로 "푸시"하도록 함으로써 프로세싱 유닛(140)에 의해 형성된 판독 요청을 만족시킨다.
유념하여야 하는 키 포인트는, 프로세서가 블록을 판독 또는 기록하고자 할 때, 프로세서는 캐쉬 코히어런스를 유지하기 위하여 시스템내 다른 프로세싱 유닛들과 희망 내용을 통신하여야 한다는 것이다. 이것을 성취하기 위하여, 캐쉬-코히어런스 프로토콜은 캐쉬 계층 구조의 각 단에서 각 블록을 그 블록의 현재 "상태(state)"를 나타내는 상태 지시자(state indicator)와 연계시킨다. 상태 정보는 코히어런시 프로토콜에서 범용 상호접속부(143) 및 캐쉬간 접속들(inter-cache connections)(140x,140y,141x,141y,142x,142y)상의 메시지 트래픽(message traffic)을 줄여주는 최적화에 사용된다. 이러한 메커니즘의 한가지 예로서, 프로세싱 유닛이 판독을 실행할 때, 프로세싱 유닛은 나중에 판독 동작을 재시도하여야 하는지의 여부를 나타내는 메시지를 수신한다. 판독 오퍼레이션이 재시도되지 않는다면, 그 메시지는 프로세싱 유닛으로 하여금 다른 어떤 프로세싱 유닛 역시 여전히 액티브한 사본 블록(a still active copy of the block)을 가지고 있는지를 판단하도록 하는 정보를 포함하는 것이 통상적이다. (이것은 다른 최저 레벨 캐쉬에게 이들이 재시도 하지 않는 모든 판독 동작에 대한 "공유(shared)" 또는 "공유 안함(not shared)"이라는 표시를 제공하도록 함으로써 성취된다).
이러한 방식으로, 프로세싱 유닛은 시스템내 다른 어떤 프로세서가 그 블록의 사본을 가지고 있는 지를 판단할 수 있다. 다른 모든 프로세싱 유닛도 그 블록의 액티브한 사본을 가지고 있지 않다면, 판독 프로세싱 유닛은 그 블록의 상태를 "배타(exclusive)"라고 마크(mark)한다. 블록이 배타라고 마크되어 있으면, 다른 모든 프로세싱 유닛도 그 블록의 액티브한 사본을 가지고 있지 않기 때문에 프로세싱 유닛으로 하여금 먼저 시스템내 다른 프로세싱 유닛과 통신하지 않고도 나중에 그 블록을 기록하도록 하는 것이 허용가능하다. 그러므로, 통상적으로, 프로세서가 자신의 의도한 바를 상호접속을 통하여 먼저 통신하지 않고도 메모리 장소를 판독 또는 기록하는 것이 가능하다. 그러나, 이것은 단지 코히어런시 프로토콜이 그 블록에 관심을 가지고 있는 프로세서가 없다는 것을 보장하는 경우에만 일어난다. 다단 캐쉬 코히어런스 프로토콜의 정확한 동작은 간략한 설명을 위하여 생략된다. 그러나, 본 발명에 포함된 본질적인 국면은 설명된다. 본 명세서에서 설명되지 않는 국면은 본 기술분야에서 통상의 지식을 가진 자들에게 잘 알려져 있다. 본 발명과 관련된 다단 캐쉬 구조의 또 다른 국면은 할당해제(deallocation)라 알려진 동작이다. 어떤 캐쉬 내에서의 블록은 "세트(set)"라는 블록들의 그룹으로 나뉘어진다. 한 세트는 소정 메모리 블록이 속할 수 있는 블록들의 집합이다. 소정 메모리 블록에 있어서, 캐쉬 내에는 블록이 프리셋 맵핑 기능(preset mapping function)에 따라서 맵핑될 수 있는 유일한 세트가 있다. 세트내 블록의 개수는 캐쉬의 연관성(the associativity of the cache)(예로, 소정 메모리 블록의 경우, 2-방향 세트 연관성(2-way set associativity)은 캐쉬 내에서 메모리 블록이 맵핑될 수 있는 블록이 두 개가 있다라는 것을 의미한다)이라 지칭된다. 그러나, 메인 메모리(main memory)내에는 여러 상이한 블록들이 소정 세트로 맵핑될 수 있다.
소정 캐쉬에서 세트내 블록들이 모두 풀(full)상태에 있고 그 캐쉬가 풀 상태의 세트에 맵핑하는 메모리 장소로 판독 또는 기록하는데 관한 요청을 수신할 때, 캐쉬는 현재 그 세트 내에 있는 블록들 중의 하나를 할당해제(deallocate)하여야 한다. 캐쉬는 본 기술 분야에서 통상의 지식을 가진 자들에게 알려진 다수의 수단(최근에 가장 적게 사용(LRU), 랜덤, 의사-LRU (least recently used(LRU), random, pseudo-LRU))중의 하나를 이용하여 축출시킬 하나의 블록을 선택한다. 선택된 블록내 데이터가 수정되어 있다면, 그 데이터는 (L1 또는 L2 캐쉬의 경우에는) 또 다른 캐쉬일 수도 있고 (L3 캐쉬의 경우에는) 메인 메모리일 수도 있는 메모리 계층 구조의 다음 번으로 낮은 레벨에 기록된다. 포함의 원리에 의하면, 계층 구조의 낮은 레벨은 기록된 수정 데이터를 보유하는 데 유용한 블록을 이미 가지고 있을 것이다. 그러나, 선택된 블록내 데이터가 수정되지 않았다면, 그 블록은 어려움없이 버려지고 계층 구조에서 다음으로 낮은 레벨에는 기록되지 않는다. 블록을 계층 구조의 한 레벨로부터 제거하는 이러한 프로세스는 "축출(eviction)"이라고 알려져 있다. 이러한 프로세스의 종료시점에서, 캐쉬는 더 이상 축출된 블록의 사본을 보유하고 있지 않으며, 캐쉬가 오퍼레이션을 (범용 상호접속부(143) 또는 캐쉬간 커넥션(140x,141x,142x,140y,141y,142y)에서 수누프할 때, 그 블록을 캐쉬 내에서 찾을 수 없기 때문에, 축출된 블록을 찾기 위하여 코히어런시 프로토콜에는 더 이상 능동적으로 참여하지 않는다.
본 발명은 멀티프로세서 시스템에서 단일 공통 버스의 한계를 극복하면서 버스 프로토콜의 변형을 요구하지 않고 단일 버스의 특성을 이용하는 분산 하드웨어 구조(a distributed hardware structure)를 개시한다. 그 결과적인 시스템은 알려진 시스템 버스의 메커니즘을 손상시키지 않고도 시스템 크기를 개선한다. 본 발명은 SMP 시스템내 다수의 디바이스들을 서로 연결시킬 수 있고 단일 버스 기반 설계의 제한을 극복할 수 있다.
비록 다음의 설명은 6XX 버스 아키텍처에 관한 본 발명을 설명하고 있을 지라도, 아래에 제시된 시스템이 다른 버스 아키텍처에도 적용될 수 있으므로 본 발명은 특정한 버스 아키텍처로 한정되는 것은 아니다.
시스템 어드레스 경로 토폴로지(System Address Path Topology)
도 4를 참조하면, SMP 시스템내 어드레스 경로의 관점으로 바라다 본 보조 (supporting) 버스 기반 캐쉬 코히어런스를 가진 대칭형 멀티프로세서 시스템의 분산 시스템 구조를 예시하는 블록 구성도가 도시된다. 도 4는 메모리 트랜잭

션과 같은 커맨드를 개시할 수 있는 다수의 마스터 디바이스를 도시한다. 프로세서, I/O 에이전트 및 코히어런트 메모리 어댑터와 같은 마스터 디바이스들은 노드라 불리는 다수의 N 그룹들 중의 클러스터내에 분산된다. 각각의 노드는 그의 마스터가 연결되는 노드 제어기에 의해 제어된다.

도 4는 시스템 요소들이 그룹화된 노드(410,420)를 도시한다. 노드의 개수는 시스템의 구조에 따라 변할 수도 있다. NODE $_0$ 으로 표시된 노드(410)는 노드(410)의 마스터가 되는 프로세서 P $_0$ 및 프로세서 P $_{P-1}$ 로 표시된 프로세서(411,412)를 포함한다. 각각의 노드 제어기는 마스터들이 분산 시스템에 연결되는 다중 표준 양방향 프로세서 어드레스-데이터 버스(multiple standard bidirectional processor address-data buses)를 가지고 있다. 프로세서(411,412)는 각기 P $_0$ Bus 및 P $_{P-1}$ Bus로 표시된 버스(413,414)를 통하여 노드 제어기 NC $_0$ 으로 표시된 노드 제어기(415)에 연결되어 있다. NODE $_{N-1}$ 로 표시된 노드(420)는 노드(420)의 마스터인 프로세서(421) 및 I/O 에이전트(422)를 포함한다. 프로세서(421) 및 I/O 디바이스(422)는 각기 버스(423,424)를 통하여 노드 제어기 NC $_{N-1}$ 로 표시된 노드 제어기(425)에 연결된다. 노드당 마스터의 개수는 시스템의 구성에 따라 변할 수도 있으며, 각 노드에서의 마스터의 개수는 시스템내 모든 노드들 전체에 대하여 균일하게 될 필요는 없다.

노드 제어기는 시스템의 마스터와 나머지 것들 사이에서 물리적인 인터페이스를 구성하며, 시스템내 각 노드 제어기는 개개의 프로세서 버스를 중재하고 또 다른 마스터 또는 슬레이브(즉 마스터 커맨드를 수용하고 이를 실행하지만 마스터 커맨드를 발생하지 못하는 디바이스)로서의 로컬 마스터들과 통신하는데 필요한 모든 로직을 가지고 있다. 프로세서는 그의 로컬 노드 제어기를 통하여 커맨드를 시스템으로 전송한다. 도 4에는 포트당 하나의 마스터를 도시하고 있지만, 그 포트의 버스에 적절한 중재 구성이 갖추어진다면 포트당 다수개의 마스터도 가능하다. 예로, 프로세서(411)는 버스(413)에 연결된 많은 프로세서들 중의 하나일 수도 있다. 그러나, 많은 프로세서가 단일 포트에 연결된다면, 이들의 어드레스 버스는 버스 사이클 시간이라는 관점에서 더욱 느리게 동작할 것이다.

이와 달리, 노드(420)의 마스터들 중의 하나는 캐쉬 코히어런스를 유지하는 다른 데이터 프로세싱 시스템과 통신을 제공하는 코히어런트 메모리 어댑터를 포함할 수도 있다. 코히어런트 메모리 어댑터는 가까이 있을 수도 멀리 있을 수도 있으며 마스터/슬레이브 디바이스로서 I/O 에이전트와 유사한 방식으로 작동하도록 하기 위하여 메모리 트랜잭션을 전송하고 수신하는 노드 제어기의 한 포트를 차지할 수도 있다. 일례로서, 다른 데이터 프로세싱 시스템의 다른 노드 제어기 또한 코히어런트 메모리 어댑터에 연결될 수 있어서 본 발명을 이용하는 데이터 프로세싱 시스템이 서로 체인화될 수도 있다.

노드 제어기(415,425)는 쌍을 이루는 단방향 어드레스 전용 버스(pairs of unidirectional address-only buses)를 통하여 어드레스 스위치(address switch : ASX)라 지칭되는 디바이스에 연결된다. AOut $_0$ 및 AIn $_0$ 로 표기된 버스(416 및 417)는 각기 어드레스 스위치 포트의 쌍에 의해 노드 제어기(415)를 어드레스 스위치(430)에 연결한다. AOut $_{N-1}$ 및 AIn $_{N-1}$ 로 표기된 버스(426,427)는 각기 어드레스 스위치 포트의 쌍에 의해 노드 제어기(425)를 어드레스 스위치(430)에 연결한다. 도시된 바와 같이, 버스 AOut $_x$ 는 어드레스를 노드 제어기들로부터 어드레스 스위치로 전달하며, 버스 AIn $_x$ 는 어드레스를 어드레스 스위치로부터 노드 제어기들로 전달한다.

어드레스 스위치(430)는 메모리 서브시스템 MS $_0$ 및 MS $_{S-1}$ 로 표기된 메모리 제어기 또는 메모리 서브시스템(442,444)에 연결되는 AIn $_N$ 및 AIn $_{(N+S-1)}$ 로 표기된 부가적인 단방향 어드레스 버스 접속(431,432)을 가지고 있다. 메모리 제어기들은 슬레이브 디바이스이고 커맨드를 분산 시스템으로 발행(issue)하지 못하는 것으로 가정된다. 메모리 서브시스템의 개수는 시스템의 구성에 따라 변할 수도 있다.

<u>시스템 데이터 경로 토폴로지(System Data Path Topology)</u>

이제 도 5를 참조하면, SMP 시스템내 데이터 경로의 관점으로 바라다 본 보조 버스 기반 캐쉬 코히어런스 프로토콜을 가진 대칭형 멀티프로세서 시스템의 분산 시스템 구조를 예시하는 블록 구성도가 도시된다. 도 4와 유사한 방식으로, 도 5는 다수의 마스터 디바이스를 보여주고 있다. 이들 마스터 디바이스들은 노드라 불리는 다수의 N 그룹들 중의 클러스터내에 분산된다. 각각의 노드는 그의 마스터가 연결되는 노드 제어기에 의해 제어된다. 도 5에는 프로세서(511,512)를 포함하는 노드(510,520)가 도시된다. 프로세서(511,512)는 버스(513,514)를 통하여 노드 제어기(515)에 연결된다. NODE $_{N-1}$ 로 표기된 노드(520)는 버스(523,524)를 통하여 각기 노드 제어기 NC $_{N-1}$ 로 표기된 노드 제어기(525)에 연결된 프로세서(521) 및 I/O 디바이스(522)를 포함한다.

도 4 및 도 5에 도시된 노드 제어기들은 물리적으로 동일한 시스템 구성요소일 수 있지만, 노드 제어기들에 의해 수행된 상이한 기능을 예시하기 위하여 상이한 블록구성으로 예시되어 있다. 도 4는 SMP 시스템내 어드레스 경로를 도시하고, 도 5는 SMP 시스템내 데이터 경로를 도시한다. 이와 달리, 본 발명의 바람직한 실시예에 있어서, 어드레스 경로 및 데이터 경로는 물리적으로 개별적인 구성요소, 칩 또는 회로에서 노드 데이터 제어기 또는 노드 어드레스 제어기와 같이 보조적인 기능을 가지도록 구현될 수도 있다. 데이터 및 어드레스 기능이 별개인 또는 합쳐진 노드 제어기를 구현하는 것은 다른 시스템 구성요소들의 파라메터에 따라 선택될 수도 있다. 예를 들면, 시스템 내부에서 지원된 버스의 크기가 충분히 작으면, 어드레스 및 데이터 기능은 단일의 노드 제어기 구성요소내에 만들어질 수도 있다. 그러나, 버스가 128 데이터비트를 지원한다면, 핀의 제한으로 인하여 어드레스 및 데이터 기능은 별개의 노드 제어기 구성요소들내에 배치될 수 있다.

이와 달리, 하나의 독립적인 노드 데이터 제어기는 노드마다 다수의 노드 데이터 제어기로 분리되어 각각의 노드 데이터 제어기가 노드의 데이터 경로의 일부를 지원하도록 할 수도 있다. 이러한 방식으로, 노드의 데이터 경로는 하나의 노드 데이터 제어기 이상으로 나뉘어진다.

도 5에서, 각각의 노드 제어기는 메모리 서브시스템 MS $_0$ 및 MS $_{S-1}$ 과 같은 다수의 메모리 제어기에 연결되어 있는 것으로 도시된다. 비록 각각의 노드 제어기가 독립적인 데이터 버스를 경유하여 각각의 메모리 제어기에 연결된 것으로 도시되지만, 적절한 중재 메커니즘이 제공된다면 다중 노드 및/또는 다중 메모리 제어기가 동일한 데이터 버스에 연결될 수도 있다. 다수의 마스터 디바이스들을 단일의 버스를 경유하여 단일의 노드 제어기에 접속하는 것과 비교해

보면, 스위칭 속도(switching rate)는 버스에 접속된 디바이스들의 개수의 함수가 될 것이다. 노드 제어기(515)는 $N_0 D_0$ 로 표기된 데이터 버스(516)를 통하여 메모리 서브시스템(542)에 그 노드 제어기를 그리고 $N_0 D_{S-1}$ 로 표기된 데이터 버스(517)를 통하여 메모리 서브시스템(544)에 그 노드 제어기를 접속시키는 다수의 메모리 서브시스템 포트를 포함한다. 노드 제어기(525)는 $N_{N-1} D_{S-1}$ 로 표기된 데이터 버스(527)를 통하여 메모리 서브시스템(544)에 그 노드 제어기를 그리고 $N_{N-1} D_0$ 로 표기된 데이터 버스(526)를 통하여 메모리 서브시스템(542)에 그 노드 제어기를 접속시키는 다수의 메모리 서브시스템 포트를 포함한다.

모든 마스터들의 데이터를 전달하는 단일의 데이터 버스 대신에, 마스터들이 단일 버스에 연결되어 있는 경우에 전달되었을 데이터 트래픽의 소량만을 각기 전달하는 다중 데이터 버스가 있다. 이렇게 함으로써, 구성요소 인터페이스들은 단일 버스에서 가능하였던 것보다 빨리 클럭될 수 있다. 이러한 구성은 단일 버스에서 가능하였을 것보다 더 많은 마스터당 버스 대역폭이 할당되게 해줌으로써, 큐잉 지연을 줄여준다.

노드 제어기 내부 어드레스 경로(Node Controller Internal Address Paths)

이제 도 6을 참조하면, 노드 제어기의 내부의 어드레스 경로를 도시하는 블록 구성도가 도시된다. NC_X 로 표기된 노드 제어기(600)는 도 4에 도시된 노드 제어기(415,425) 또는 도 5에 도시된 노드 제어기(515,525)와 유사하다. 노드 제어기(600)의 개개의 포트들은 마스터로부터의 커맨드가 노드 제어기로 입력될 때 그 커맨드를 버퍼링(buffering)하는 자신들의 큐를 가지고 있다. 커맨드는 어드레스 스위치를 향하여 점진적으로 선택되기 때문에 이들 버퍼내에서 대기하는 동안에 비결정적인 지연(non-deterministic delay)을 유발할 수도 있다.

노드 제어기(600)는 마스터 디바이스와 연결되는 양방향 버스(601-604)를 가지고 있다. 버스(601-604)는 버스 트랜시버(605-608)를 통하여 입력 경계 래치(input boundary latches : IN-BL)(609-612) 및 출력 경계 래치(output boundary latches : IN-BL)(613-616)에 연결된다. 입력 경계 래치(609-612)는 마스터 디바이스로부터의 커맨드를 보유하는 버퍼(617-620)로 커맨드를 공급한다. 마스터 디바이스로부터의 커맨드는 트랜잭션 태그, 트랜잭션 타입, 타겟 또는 소스 어드레스 및 이와 연관된 기타 가능한 정보로 이루어진다. 버퍼(617-620)는 필요하다면 커맨드와 관련한 모든 정보를 보유할 수도 있고 또는 이와 달리 노드 제어기 내부의 어드레스 경로를 작동시키는데 필요한 정보만을 보유할 수도 있다. 입력 버퍼에 의해 보유된 정보는 다른 구성의 노드 제어기에 따라 변할 수도 있다. 버퍼(617-620)는 한번에 하나의 커맨드를 선택하여 래치(622), 전송기(623) 및 $AOut_X$ 로 표기된 버스(624)를 경유하여 어드레스 스위치로 전송하는 제어 유닛/멀티플렉서(621)로 커맨드를 제공한다.

노드 제어기(600)는 버스(601-604)를 통하여 커맨드를 수신한 다음 경계 래치(622) 및 전송기(623)를 경유하여 버스(624)를 통해 어드레스 스위치로 전달한다. 대응하는 방식으로, 노드 제어기(600)는 AIn_X 로 표기된 버스(625) 및 수신기(626)를 통하여 FROM_ASX_BL로 표기된 경계 래치(627)에 제공될 어드레스 스위치로부터의 커맨드를 수신한다. 이들 커맨드는 중개 래치(intermediate latch)(628) 및 출력 경계 래치(613-616)와 같이 일정한 지연을 갖는 일정한 개수의 래치를 통과하는 어드레스 경로를 따라가서 버스(601-604)에 도달한다. 부가적으로, 마스터 디바이스에게 인가된 커맨드는 제어 유닛/멀티플렉서(629-632)와 같이 일정한 지연을 갖는 포트마다 구성된 멀티플렉서를 통과한다. 이러한 방식으로, 버스(625)를 경유하여 도달하는 커맨드는 결정적인 개수의 사이클만큼의 일정한 지연을 갖는 경로를 통과한다. 환언하면, 일정한 지연 기간은 커맨드가 래치 FROM_ASX_BL에 도달한 때의 시점에서부터 노드 제어기에 접속된 한 세트의 프로세서와 같은 각각의 마스터 디바이스가 도달하는 커맨드를 건네 받는 시점 사이에서 발생한다.

마스터에 연결된 포트의 중재기(arbiter)는 포트 버스를 구동하는 노드 제어기에게 최고 우선순위를 주도록 설계된다. 만일 마스터가 노드 제어기가 버스를 구동하기를 바라는 시점과 같은 시점에 버스를 구동하려고 요청을 한다면, 노드 제어기는 최고의 우선순위를 가진다. 바람직한 실시예에서, 이러한 중재 시나리오를 조성하기 위하여, "스누프 유효(SnoopValid)"(도시 안됨) 라는 신호가 어드레스 스위치에 의해 보내지는 커맨드의 앞에 발행된다. 이것은 노드 제어기와 그의 마스터간의 버스 액세스에 대한 중재가 충분히 일찍 이루어지게 하여 AIn_X 버스를 통하여 어드레스 스위치로부터 도달하는 커맨드가 노드 제어기내에 있는 동안에 확실하게 한 사이클이 지연되지 않도록 해준다. 이것은 AIn_X-P_X 버스 경로를 따라서 배열된 일정한 개수의 래치의 지연 기간이 결정적인 개수의 사이클로 귀착하는 것을 보장한다.

제어 논리 유닛(633)은 FROM_ASX_BL에 래치된 커맨드를 받아들이고 노드 제어기(600)내의 다른 유닛 또는 구성요소에 필요한 적절한 제어 신호를 결정한다. 예로, 제어 논리 유닛(633)은 제어 신호(634)를 통하여 버퍼(617-620)와 통신하고, 제어 신호(636)를 통하여 제어 유닛/멀티플렉서(621)와 통신하고, 제어 신호(635)를 통하여 제어 유닛/멀티플렉서(629-632)와 통신하면서 커맨드를 선택하고, 충돌을 해결하고, 필요하다면 커맨드의 타입을 포함하는 커맨드의 필드를 변경함으로써, 노드 제어기(600)내부에서 커맨드의 흐름이 연속되게 한다. 제어 논리 유닛(633)은 다른 제어 신호(637)를 수신하기도 한다.

어드레스 스위치 내부 어드레스 경로(Address Switch Internal Address Paths)

도 7을 참조하면, 노드 제어기와 메모리 서브시스템을 연결하는 어드레스 스위치의 내부 어드레스 경로를 도시하는 도면이 도시된다. 어드레스 스위치(700)는 한 세트인 네 개의 노드 제어기와 두 개의 메모리 서브시스템을 연결한다. 커맨드는 $AOut_0$ - $AOut_3$ 으로 표기된 버스(701-704)로부터 수신기(709-712) 및 입력 경계 래치(713-716)를 경유하여 선입 선출(FIFO) 큐(721-724)에 도달한다. 이들 커맨드는 제어 유닛/멀티플렉서(725)에 의해 선택되기 전까지 FIFO내에서 상주할 수도 있다. 커맨드는 FIFO내에 머무는 동안 유한한 개수의 값이지만, 비결정적인 개수의 지연 사이클을 받게된다. 제어 논리 유닛은 입력되는 커맨드를 선택하는 것을 결정하기 위하여 제어 유닛/멀티플렉서(725) 및 FIFO(721-724)와 통신할 수도 있다. 제어 논리 유닛(726)은 또한 다른 제어 신호(733)를 수신한다.

제어 유닛/멀티플렉서(725)는 몇 개의 지연 사이클 개수라는 점에서 결정적인 경로를 통하여 노드 제어기 및 메모리 서브시스템으로 브로드캐스트될 커맨드를 한번에 하나씩 선택한다. 도 7에 예시된 실시예에 있어서, 커맨드는 출력

경계 래치(727,728), 송신기(729,730) 및 AIn $_4$ -및AIn $_5$ 로 표기된 단방향 버스(731,732)를 경유하여 메모리 서브시스템으로 전달된다. 커맨드는 출력 경계 래치(717-720), 송신기(741-744) 및 AIn $_0$ -및 AIn $_3$ 로 표기된 단방향 버스(705-708)를 경유하여 노드 제어기로 전달된다. 본 실시예에 있어서, 출력 경계 래치(717-720,727,728)에서는 단지 한번의 지연 사이클이 있다.

도 4 내지 7의 설명으로부터, 트랜잭션은 마스터 디바이스에 의해 발행되어 그의 버스 및 포트를 경유하여 그의 노드 제어기로 전달된다는 것이 이해될 수 있다. 노드 제어기는 버스를 통하여 마스터 디바이스에게 소정 타입의 중개 응답을 제공할 것이며 트랜잭션을 큐하여 나중에 시스템의 나머지 디바이스들에게 발행할 수도 있다. 일단 트랜잭션이 시스템의 나머지 디바이스에게 발행되면, 어드레스 스위치는 트랜잭션이 알려진 전파 지연을 가지고 시스템의 나머지 디바이스에게 브로드캐스트될 수 있게 하여 그 밖의 다른 디바이스가 트랜잭션을 스누프할 수 있게 보증한다.

본 발명의 분산 시스템 구조에 따르면, 시스템내 각각의 디바이스는 같은 사이클에서 그 트랜잭션을 확인할 수 있을 것이고, 같은 사이클 내에서 코히어런스 응답을 제공할 것이다. 어드레스 스위치는 트랜잭션을 발행한 디바이스를 갖는 노드의 노드 제어기를 포함하는 모든 노드 제어기에게 트랜잭션을 브로드캐스트할 수 있다. 각각의 노드 제어기에는 적절한 로직이 내포되어 있어서, 노드 제어기는 스누프되는 입력 트랜잭션이 그의 어느 한 포트에 있는 디바이스에 의해 최초로 발행되었는 지를 결정할 수 있다. 그러하다면, 노드 제어기는 트랜잭션을 발행한 포트상의 버스가 그 포트로부터 수신된 트랜잭션을 스누프하지 않도록 보증한다. 그렇지 않다면, 디바이스는 그 자신의 트랜잭션을 스누프함으로써 "혼란(confused)"에 빠질 수도 있다. 디바이스가 자신의 트랜잭션을 스누프한다면, 그 디바이스는 자신의 시초 트랜잭션(its original transaction)과 충돌을 나타내는 응답을 발행할 수도 있다. 그러한 경우라면, 시초 트랜잭션은 실제로 스누프되어지는 트랜잭션이기 때문에, "충돌(collision)"은 결코 해결되지 않으며, 트랜잭션은 결코 완료될 수 없을 것이다.

트랜잭션이 발행되고 완료되는 방식에 대하여는 하기에 보다 상세히 설명된다.

메모리 서브시스템 내부 어드레스 경로(Memory Subsystem Internal Address Paths)

도 8을 참조하면, 본 발명의 분산 시스템의 어드레스 스위치에 연결된 메모 리 서브시스템이 도시된다. 도 8은 MS $_X$ 로 표기된 메모리 서브시스템(800)을 도시한다. 메모리 서브시스템(800)내 메모리 제어기(801)는 어드레스 스위치로부터의 커맨드를 AIn $_X$ 로 표기된 단방향 버스(802)와 단순하게 일정한 지연 파이프(a fixed delay pipe)인 다수의 래치 FD(803)를 경유하여 수신한다. 이러한 방식으로, 어드레스 스위치에 의해 전송된 커맨드는 일정한 수만큼의 사이클의 지연을 받은 다음 메모리 제어기에게 이용가능해진다.

이전에 도시된 바와 같이, 버스 AIn $_X$ 를 경유하여 노드 제어기에 도달하는 커맨드는 FROM_ASX_BL 래치에서 보존되는 시점부터 마스터 디바이스에 전달되기까지 결정적인 지연 경로를 통과한다. 유사하게, 커맨드는 어드레스 스위치내 제어 유닛/멀티플렉서로부터 메모리 서브시스템내 일정 지연 파이프까지의 결정적인 지연 경로를 통과한다. 메모리 서브시스템내 래치 FD(803)의 지연이 적절한 값으로 조절되면, 노드 제어기의 포트에 연결된 마스터가 동일한 커맨드를 넘겨받는 시점과 같은 시간에 메모리 제어기가 그 커맨드를 넘겨받는 것이 보장될 수 있다. 그러므로, 어드레스 스위치내 제어 유닛/멀티플렉서가 트랜잭션을 브로드캐스트하는 시점과 마스터 및 메모리 제어기가 그 커맨드를 수신하는 시점 사이에는 결정적인 개수의 지연이 존재한다.

단지 소수의 마스터만이 노드 제어기의 각 포트에 연결되어 있기 때문에, 이들 포트에 연결된 각각의 버스가 동작될 수 있는 속도는 시스템내 총 포트 수에 무관하다. 예를 들면, 단일의 마스터가 각각의 포트에 접속된다면, 그의 버스는 최선 으로 가능한 속도로 포인트-투-포인트 모드(point-to-point mode)로 구동될 수 있다. 그러므로, 본 발명의 분산 구조는 시스템의 대역폭을 향상시키기 위하여 잘 이해되게 축척할 수 있고 멀티프로세서용 버스-기반 캐쉬-코히어런트 프로토콜을 용이하게 검증할 수 있다.

노드 제어기 내부 데이터 경로(Node Controller Internal Data Path)

도 9a 및 도 9b를 참조하면, 노드 제어기의 내부의 데이터 경로를 예시하는 블록 구성도가 도시된다. NC $_X$ 로 표기된 노드 제어기(900)는 도 4의 노드 제어기(415,425) 또는 도 5의 노드 제어기(515,525)와 유사하다. 노드 제어기(900)의 개개 포트는 마스터로부터 데이터가 노드 제어기로 입력될 때 그 데이터를 버퍼하는 자신의 큐를 가지고 있다. 데이터는 목적지를 향하여 점진적으로 이동되기 위하여 이들 버퍼내에 대기하는 중에 비결정적인 지연을 유발할 수도 있다.

노드 제어기(900)는 마스터 디바이스를 접속하는 P $_X$ 버스로 표기된 양방향 버스(901-904)를 가지고 있다. 버스(901-904)는 버스 트랜시버(905-908)를 경유하여 입력 경계 래치(909-912) 및 출력 경계 래치(913-916)에 연결된다. 입력 경계 래치(909-912)는 마스터 디바이스로부터의 데이터를 보유하는 데이터 버퍼(917-920)에 그 데이터를 공급한다. 노드 제어기의 포트중의 한 포트로부터 입력되는 데이터는 메모리 서브시스템 또는 다른 캐쉬로 향해질 수도 있다.

도 6에 예시된 실시예의 연속판인 도 9a 및 9b에 예시된 실시예에 있어서, 노드 제어기의 포트들 중의 한 포트로부터 입력되는 데이터는 세 개의 장소, 즉 노드내부의 데이터를 전달하기 위한 메모리 서브시스템 MS $_0$, 메모리 서브시스템 MS $_{S-1}$ 또는 캐쉬-투-캐쉬 FIFO(FIFO C2C)중의 한 곳으로 보내질 수 있다. FIFO C2C 메커니즘과 함께, 각각의 노드는 그의 포트들 중의 한 포트에서 다른 포트로 데이터를 전송할 수 있음으로써, 하나의 마스터로부터 다른 마스터로의 데이터 전송이 이루어진다. 버퍼(917-920)는 데이터를 발송하는 데이터 소스를 선택하는 멀티플렉서(925-927)에 데이터를 공급한다. 제어 논리 유닛(939)은 메모리 서브시스템(MS $_{0)}$ 으로 전송될 데이터를 선택하는 멀티플렉서(925)에 필요한 제어신호와 메모리 서브시스템(MS $_{S-1)}$ 으로 전송될 데이터를 선택하는 멀티플렉서(926)에 필요한 제어신호를 제공한다. 노드 제어기(900)는 멀티플렉서(925,926)로부터 데이터를 경계 래치(931,933) 및 트랜시버(935,936)를 통과하고 N $_X$ D $_0$ 및 N $_X$ D $_{S-1}$ 로 표기된 양방향 버스(937,938)를 경유하여 메모리 서브시스템(MS $_0$,MS $_{S-1}$)으로 전달한다. 제어 논리 유닛(939)은 노드 내부에서 전달될 데이터를 선택하는 멀티플렉서(927)에

필요한 제어 신호를 제공한다. 그 다음에 데이터는 FIFO(928)내로 큐된다.
대응하는 방식으로, 노드 제어기(900)는 양방향 버스(937,938)를 경유하여 메모리 서브시스템(MS_0) 및 메모리 서브시스템(MS_{S-1})으로부터 트랜시버(935,936) 및 경계 래치(932,934)를 통하여 데이터를 수신한다. 그 다음에 데이터는 적절한 FIFO(929,930)내에서 큐된다. FIFO(928-930)로부터 데이터는 제어 유닛/멀티플렉서(921-924)와 같은 포트당 하나씩의 멀티플렉서를 통과한다. 제어 논리 유닛(939)은 마스터 디바이스로 전송될 데이터를 선택하는 멀티플렉서(921-924)에 필요한 제어신호를 제공한다. 또한 제어 논리 유닛(939)은 다른 제어 신호(940)를 수신하기도 한다. 그러므로, 노드 제어기는 데이터 버스에 대한 중재 로직을 가지며 데이터 전송을 병렬로 제어한다는 점에서 자급자족인 면(self-sufficient)이 있다. 이러한 방식으로, 본 발명의 분산 시스템 구조는 시스템 데이터 처리량을 개선시킬 수 있다.

응답 조합 블록(Response Combination Block(RCB)

도 10a 및 도 10b를 참조하면, 도 4 및 도 5에 도시된 것과 유사한 분산 시스템 구조의 버스 응답 신호를 결정하는 시스템 구조를 예시하는 블록 구성도가 도시된다. 도 10a 및 도 10b는 각기 본 발명의 분산 시스템 구조내 디바이스들과 버스 신호 (응답) AStat 및 AResp를 조합하는 제어 논리 블록과의 연결을 도시한다. 석명을 요하기 위하여, AStat 신호와 AResp 신호는 개별적으로 도시된다. I/O 에이전트는 도 10a 및 도 10b에 도시된 노드 제어기의 포트에 연결된 마스터 디바이스로서 작용할 수 있다는 것을 다시 한번 주목하여야 한다.
도 10a에 도시된 바와 같이, P_X로 표기된 프로세서(1001-1004)는 프로세서를 응답 조합 블록(RCB)(1000)에 연결해주는 $P_X N_X$ AStOut로 표기된 단방향 AStatOut 신호 및 $P_X N_X$ AStIn로 표기된 단방향 AStatOut 신호를 가지고 있다. MS_X로 표기된 메모리 서브시스템(1005 및 1006)과 같은 슬레이브 디바이스는 M_X_AStOut로 표기된 AStatOut 신호(1013,1014)와 M_X_AStIn로 표기된 AStatIn 신호(1015,1016)를 RCB와 연결한다. NC_X로 표기된 노드 제어기(1017,1018)는 상기한 바와 유사하게 포트당 한 세트의 $M_X P_X$ AStOut로 표기된 AStatOut 신호(1019-1022)와 $N_X P_X$ AStIn로 표기된 AStatIn 신호(1023-1026)에 의해 RCB와 연결된다. ASX로 표기된 어드레스 스위치(1027)는 브로드캐스트 신호(1028) 및 마스터 디바이스가 시스템으로 트랜잭션을 발행한 노드내부의 포트 식별자와 함께 노드 식별자를 인코딩한 것인 트랜잭션 소스 ID(1029)를 공급함으로써, 트랜잭션을 시스템에서 처리하는 데 적절한 로직을 결정하는 데 개입한다.
도 10B에 도시된 바와 같이, 프로세서(1001-1004)는 프로세서를 RCB(1000)에 연결해주는 $P_X N_X$ AReOut로 표기된 단방향 ARespOut 신호 및 $P_X N_X$ AReIn로 표기된 ARespIn 신호를 가지고 있다. 메모리 서브시스템(1005,1006)은 RCB와 M_X_AReIn로 표기된 ARespOut 신호(1065,1066)를 연결한다. 메모리 서브시스템(1005,1006)은 이들 슬레이브 디바이스들에 의해 구동되지 않는 ARespOut 라인과는 연결되지 않는다. 노드 제어기(1017,1018)는 또한 상기한 바와 유사한 포트당 한 세트의 $N_X P_X$ AReOut로 표기된 단방향 ARespOut 신호(1069-1072)와 $N_X P_X$ AReIn로 표기된 ARespIn 신호(1073-1076)를 이용하여 RCB에 연결된다. 되풀이하여 말하건대, 어드레스 스위치(1027)는 브로드캐스트 신호(1028) 및 트랜잭션 포트 ID(1029)를 공급함으로써 트랜잭션의 적절한 로직을 결정하는 데 개입한다.
도 10a 및 도 10b로부터 명백해진 바와 같이, 마스터 디바이스로의/로부터의 한 세트의 AStatIn/AStatOut 신호 및 ARespIn/ARespOut 신호는 상기한 바와 유사하게 그의 노드 제어기로/로부터 AStatIn/AStatOut 신호 및 ARespIn/ARespOut 신호와 쌍을 이루고 있다. 이러한 짝짓기는 포트마다 하나씩 이루어진다. 상기 기술된 바와 같이, 본 실시예에서 각 포트에는 단일의 마스터 디바이스가 접속되어 있는 것으로 도시되어 있다. 그러나, 하나 이상의 마스터 디바이스가 포트마다 접속되어 있는 경우라면, AStatIn/AStatOut 신호 및 ARespIn/ARespOut 신호의 쌍은 표준 단일 버스 구성에서와 같이 그 포트상의 버스에 접속된 일련의 마스터 디바이스들에 의해 사용된다.
바람직한 실시예에 있어서, RCB는 각종 소스 디바이스로부터의 AStatOut 신호와 ARespOut 신호를 조합하고 6XX 버스 사양(6XX bus specification)(본 명세서에서 참조문헌으로 인용되는, IBM Server Group Power PC MP Systwm Bus Description, Version 5.3에 개시됨)에 의거하여 AStatIn 신호와 ARespIn 신호를 생성한다. RCB는 AStatOut 신호 및 ARespOut 신호를 수신하고 AStatIn 신호 및 ARespIn 신호를 출력한다. 모든 디바이스가 특정 트랜잭션마다 동일한 응답을 수신하는 것은 아니다. 각 디바이스에 의해 수신된 신호는 하기 보다 상세히 설명되는 바와 같이 사이클 단위로 결정된다.

로컬/글로벌 사이클(Local/Global Cycles)

어느 소정의 시스템 사이클 동안, 어느 포트의 마스터 디바이스가 트랜잭션을 그의 노드 제어기를 통해 수신하는 그의 포트 버스를 통하여 발행할 수도 있고 또는 노드 제어기가 마스터 디바이스에게 어드레스 스위치를 통하여 전달된 트랜잭션을 제공하여 그 트랜잭션을 스누프하도록 할 수도 있다. 마스터 디바이스가 트랜잭션을 발행할 때의 사이클은 "로컬(local)"이라 불리고, 노드 제어기가 트랜잭션을 발행할 때의 사이클은 "글로벌(global)"이라 불린다.
상기 기술된 바와 같이, 어드레스 스위치는 한 번에 하나의 트랜잭션을 모든 노드 제어기로 브로드캐스트하며, 어드레스 스위치가 그러한 트랜잭션을 발행한 시점과 그 트랜잭션이 각각의 노드 제어기의 포트에 출현하는 시점간에는 일정한 지연이 존재한다. 이러한 방식 하에서, 노드 제어기가 어드레스 스위치로부터 브로드캐스트 트랜잭션을 수신한 다음, 기설정된 수의 사이클이 경과한 후에 한 사이클동안 그 트랜잭션을 노드 제어기의 포트의 버스상의 디바이스들에게 제공한 이후에는, 하기 설명되는 바와 같이 한 포트를 예외로 하고는, 모든 노드 제어기가 같은 사이클 동안 자신의 모든 포트들에서 동일한 동작을 수행한다. 그러므로, 포트들 중의 한 포트의 버스상에서 실행되는 글로벌 사이클이 있을 경우, 시스템내 모든 포트에서 글로벌 사이클이 실행된다. 나머지 모든 사이클은 로컬 사이클이다.
로컬 사이클 동안, 포트에서의 활동(activity)은 시스템내 다른 포트에서의 활동과 상관되지는 않는다. 디바이스가 트랜잭션을 발행할 필요가 있는지의 여부에 따라서, 로컬 사이클이 점용(occupied)되기도 하고 유휴상태(idle)에 놓이기도 할 것이다. 그러므로, 트랜잭션이 시스템내 모든 디바이스에 의해 스누프될 때 글로벌 사이클이 발생하며, 로컬

사이클은 단지 디바이스에 의해 트랜잭션을 발행하는 데 사용될 수 있다.

로컬 사이클 대 글로벌 사이클 동안 RCB의 동작(Operation of RCB During Local Vs Global Cycles)

전체 시스템의 사이클이 로컬 또는 글로벌로서 "선정(colored)"되어 있다면, 트랜잭션의 발행 이후 일정 수의 사이클이 경과한 다음 발생하는 응답 생성, 응답 조합 및 응답 수신 사이클은 마찬가지로 로컬 응답 윈도우 또는 글로벌 응답 윈도우라고 불린다. 이러한 이유로 인하여, RCB의 응답 조합 기능은 소정 사이클 동안 대응적으로 로컬 또는 글로벌 모드에 있는 것으로 간주된다. 로컬 사이클 동안, RCB는 응답을 포트당 하나씩 조합한다. 즉, RCB는 포트의 응답과 노드 제어기가 그 포트에 대응하여 발생한 응답을 조합한다. 글로벌 사이클 동안, (되풀이하여 말하건대 하기 설명되는 바와 같이 한 포트만을 제외하고는) RCB는 모든 포트들로부터의 응답과 시스템내 노드 제어기들로부터의 응답을 조합한다.

로컬 조합 모드와 글로벌 조합 모드간의 적절한 스위칭을 성취하기 위하여, RCB는 도 10a의 브로드캐스트 신호(1028) 및 트랜잭션 소스 ID 신호(1029)로 도시된 바와 같이, 어드레스 스위치에 의한 노드 제어기들로 제공된 트랜잭션의 브로드캐스트임을 나타내는 신호를 가지고 있다. RCB에 저장된 구성 정보(configuration information)는 브로드캐스트 트랜잭션 신호의 도달 이후에 브로드캐스트 트랜잭션 에 대한 응답의 조합이 수행되는 정확한 사이클을 나타낸다. 이러한 방식으로, 매 글로벌 사이클마다, RCB는 적절한 소스로부터의 응답을 조합한다.

일차 대 이차 로컬 사이클(Primary Vs Secondary Local Cycles)

프로세서는 로컬 사이클 동안에만 트랜잭션을 발행한다. 소정 타입의 트랜잭션의 경우, 프로세서는 한 번만 트랜잭션을 발행한다. 다른 소정 타입의 트랜잭션의 경우, 프로세서는 여러 번에 걸쳐 트랜잭션을 발행하도록 요구받을 수도 있다. 프로세서는, RCB와 함께, 수행되어야 하는 행위에 관하여 AStatIn/AStatOut 신호 및 ARespIn/ARespOut 신호를 통하여 그의 노드 제어기에 의해 지시 받는다.

프로세서가 처음에 트랜잭션을 발행하는 로컬 사이클은 "일차 로컬 사이클(primary local cycle)"이라 불리고, 다른 모든 사이클은 "이차 로컬 사이클(secondary local cycle)"이라 불린다. 6XX 버스 구조에 있어서, 이차 트랜잭션은 "1"로 세트되는 "R" 비트로 마크된다. 다시 말해서, 그의 응답과 관련한 사이클은 트랜잭션의 발행에 대응하는 적절한 방식으로 일차 또는 이차로 표기된다.

시간 및 공간 분산 시스템에서 스누핑에 의한 코히어런스의 성취(Achievement of Coherence by Snooping in a Temporally and Spatially Distributed Manner)

전술한 설명으로부터 프로세서 및 디바이스는 시스템으로 발행된 사이클과는 상이한 사이클 동안 다른 프로세서 및 디바이스로부터의 트랜잭션을 관찰한다는 사 실이 명백해질 것이다. 이것은 시스템내 모든 디바이스들이 트랜잭션이 발행된 시점과 동일한 시점에 그 트랜잭션을 관찰함과 동시에 그에 대한 코히어런스 응답을 발생하는 그리고 그 트랜잭션의 발원지가 그 동일 시점에 응답을 수신하는 단일 버스 환경에서의 스누핑 프로토콜을 갖춘 상황과는 다르다. 그러므로, 본 시스템에 있어서, 코히어런스의 성취는 시간적으로 분산되고 공간적으로 분산된다. 즉, 여러 사이클과 여러 노드 제어기에 접속된 여러 버스에 걸쳐서 이루어진다.

분산 시스템 구조를 사용함에 있어서, 글로벌 코히어런스를 효과적인 방식으로 성취하는 것이 중요하다. 이렇게 하기 위하여, 모든 트랜잭션은 다음과 같은 두 가지 카테고리로 분류된다. 즉, (1) 글로벌 코히어런스 응답을 예측하고 이것을 일차 응답 윈도우내에 전달하는 것이 가능한 트랜잭션, (2) 궁극적인 코히어런스 응답이 계산될 수 있기 전에 전역적으로 스누프하는 것을 필요로 하는 트랜잭션.

첫 번째 경우에 있어서, 노드 제어기는 트랜잭션을 수신하고 일차 응답 윈도우내에 그 트랜잭션을 발행한 실체에게 글로벌 코히어런스 응답을 발행한다. 그 다음에 노드 제어기는 나중에 시스템내에서 트랜잭션을 완료시키고 글로벌 응답을 성취하는 전적인 책임을 진다.

두 번째 경우에 있어서, 노드 제어기는 세 단계를 행한다. 첫 단계로, 노드 제어기는 트랜잭션을 수신하고 글로벌 응답의 성취와 전달의 연기(postponement)를 나타내는 일차 응답을 전달한다. 6XX 버스 아키텍처에 있어서, 이러한 응답은 "재실행(Rerun)" 응답이라 한다. 두 번째 단계로, 차후에 노드 제어기는 그 트랜잭션에 대한 글로벌 코히어런스 응답을 성취한다. 세 번째 단계로, 노드 제어기는 프로 세서가 이차 트랜잭션을 발행하고 이차 응답 윈도우내에 글로벌 응답을 전달할 것을 요구한다. 6XX 버스 아키텍처에 있어서, 프로세서에게 이차 트랜잭션을 발행할 것을 요구하는 것은 고유 트랜잭션에 대응하는 태그(a tag)를 갖는 Rerun 커맨드를 발행함으로써 이루어진다. 그러면 프로세서는 그의 트랜잭션중의 어느 것이 재실행되어야 하는 지를 식별하는 태그를 사용할 수도 있다.

재실행 커맨드 및 이차 응답(Rerun Commands and Secondary Responses)

상술한 바와 같이, 디바이스로부터 수신된 트랜잭션은 시스템의 나머지 디바이스들에 의해 스누프된다. 그러한 스누프동안, 트랜잭션을 발행한 디바이스는 그 디바이스가 자신의 트랜잭션을 가지고 스누프됨으로써 혼동이 일어나지 않도록 스누프되지 않는다.

사실상, 상기 첫 번째 경우의 트랜잭션, 즉, 노드 제어기가 트랜잭션을 수신하고 일차 코히어런스 응답내에 그 발행 실체에게 글로벌 코히어런스 응답을 발행하는 트랜잭션의 경우, 트랜잭션을 발행한 디바이스에 대응하는 포트는 트랜잭션의 스누프 사이클 내에서 로컬 모드로 유지됨으로써, 프로세서가 또 다른 트랜잭션을 발행하도록 할 수도 있다. 상술한 바와 같이, 트랜잭션의 스누프 사이클에 대응하는 응답 윈도우 동안, RCB는 트랜잭션을 발행한 노드 제어기상의 포트와 다른 모든 소스로부터의 응답들을 조합하도록 구성되어 있다. 그러면 노드 제어기는 프로세서가 트랜잭션을 발행하기 위하여 선택하는 포트를 통하여 일차 또는 이차 응답을 공급할 수 있다.

상술한 두 번째 경우의 트랜잭션, 즉 궁극적인 코히어런스 응답이 계산될 수 있기 전에 전역적으로 스누프하는 것을 필요로 하는 트랜잭션의 경우, 노드 제어기는 특정 포트를 로컬 모드로 유지하지만 재실행 트랜잭션을 발행한다. 포트의 출력 경계 래치에 연결되는 제어 유닛/멀티플렉서는 노드 제어기로 하여금 이러한 기능을 성취하게 해준다.

이와 달리, 노드 제어기는 적극적으로 선택하는 것은 아니고, 디바이스에게 트랜잭션을 발행하도록 허용해주는 대신, 필요하다면, 노드 제어기는 디바이스의 트랜잭션이 시스템의 나머지 디바이스들에서 스누프되는 동안의 사이클에서 디바이스에게 널(null) 또는 재실행 트랜잭션을 자체적으로 발행할 수도 있다.

이제, 도 10c-10d를 참조하면, 로컬 및 글로벌 사이클에서 신호들이 함께 참여하는 구성요소를 예시하는 블록 구성도가 도시된다. 도 10c는 글로벌 사이클 동안 RCB에 의해 고려되는 신호를 도시한다. 도시된 실시예에 있어서, 단일의 마스터 디바이스, 즉 프로세서(1001)의 신호는 RCB가 다른 디바이스들, 즉 글로벌 응답에 관한 노드 제어기 및 메모리 서브시스템에게 제공되는 적절한 신호를 결정하는데는 관여하지 않는다. 프로세서(1001)에 관련한 신호는 그의 노드 제어기로부터 대응하는 신호와 함께 쌍을 이루는데, 이들 신호는 글로벌 응답에는 고려되지 않는다. 프로세서(1001)의 블록으로부터 알 수 있듯이, 프로세서(1001)에 의해 발행된 트랜잭션이 시스템의 나머지 디바이스에 의해 스누프되는 동안 로컬 사이클로 유지된다. 위에서 언급된 바와 같이, 비록 프로세서(1001)가 도시되어 있지만, 신호들은 포트당 하나씩 고려되며, 시스템의 나머지 디바이스가 글로벌 사이클로 유지되는 동안 특정 포트의 버스는 로컬 사이클로 유지된다.

도 10d는 로컬 사이클 동안 RCB에 의해 고려되는 신호를 도시한다. 도시된 실시예에 있어서, 단일 마스터 디바이스, 즉 프로세서(1001)로부터의 신호는 프로세서(1001)와 그의 노드 제어기에게 되돌려질 RCB에 의한 적절한 신호의 결정에 관여한다. 이와 동시에, 노드 제어기 및 메모리 서브시스템과 같은 다른 디바이스로부터의 신호가 글로벌 응답에 필요한 응답을 하는데 관여할 수도 있다.

프로세서(1001)에 관련한 신호는 그의 노드 제어기로부터의 대응하는 신호와 함께 쌍을 이루는데, 이들 역시 글로벌 응답에는 영향을 미치지 않는다. 프로세서(1001)의 견지에서 그 시스템의 나머지 디바이스에 의해 다른 트랜잭션이 스누프되는 동안 그 다른 트랜잭션을 발행할 수도 있다. 석명을 위하여, 어드레스 스위치로부터의 신호는 비록 RCB가 이들 신호를 사용하여 어느 포트를 로컬 사이클에 배치할 것인지를 결정하지만 도 10d의 로컬 사이클에서는 도시하지 않았다.

<u>버스 메모리 트랜잭션들의 정확한 순서 정하기(Achieving Correct Order Among Bus Memory Transactions)</u>

정확하게 동작하는 컴퓨터 시스템의 경우, 마스터 디바이스에 의해 발행된 소정의 메모리 액세스 트랜잭션 및 다른 유형의 트랜잭션은 정확하고 모호하지 않게 정렬되어야 한다. 단일 시스템 버스를 갖춘 시스템에서, 이러한 작업은 힘겹지 않게 성취되는 데, 이것은 트랜잭션이 버스에 실리는 순서가 이들 트랜잭션에 포함되기 때문이다. 그러나, 다중 버스를 갖춘 분산 시스템에서는 시스템의 도처에서 큐된 트랜잭션에 순서를 부과하는 작업을 요한다. 본 발명의 분산 아키텍처는 정확 하고 명백한 순서가 일련의 트랜잭션에 포함되게 한다. 또한 본 발명은 스누핑, 하드웨어 캐쉬-코히어런스 프로토콜이 지원될 수 있도록 순서를 성취하는 효과적인 수단을 제공한다.

SMP 시스템내 디바이스가 프로그램의 영향하에서 또는 제어 시퀀스의 영향 하에서 메모리를 액세스할 때, 디바이스는 메모리 트랜잭션을 발행한다. 이들 디바이스는 또한 다른 버스 트랜잭션을 발행하여 시스템내에서 코히어런스, 정렬, 인터럽트 등을 성취하도록 할 수도 있다. 이들 트랜잭션은 통상 다른 트랜잭션과 간섭 없이 병행하여 완성시킬 수 있다. 그러나, 두 개의 트랜잭션이 동일한 이중 워드 내의(within the same double word) 어드레스를 지명할 때, 예로 6XX 버스 용어에 따르면, 소위 "충돌(collide)"이라 불리게 되는 경우, 두 개의 트랜잭션은 어떤 특정 순서로 완료되어야만 한다. 몇몇 경우에 있어서, 어느 쪽도 무관하게 완료의 순서가 허용가능하고, 또 어떤 때에는 그 순서가 일정하고 트랜잭션의 타입에 의해 암시되어 있기도 하다. 예를 들면, 판독(read) 트랜잭션과 기록(write) 트랜잭션이 메모리 코히어런스 불필요(Memory Coherence Not Required)로서 선언된 어드레스를 액세스하고자 한다면, 두 트랜잭션에 대하여 어떠한 완료 순서도 허용가능하다. 그러나, 판독 트랜잭션과 기록 트랜잭션이 코히어런시가 유지되는 캐쉬가능 어드레스(cashable address)를 조회하는 경우, 완료 순서는 기록 트랜잭션 다음에 판독 트랜잭션이 진행되게 하여야 한다.

<u>디폴트 순서를 트랜잭션에 부과하는 수단(Means of Imposing a Default Order on Transactions)</u>

도 4-10d를 참조하여 설명된 분산 SMP에 있어서, 다중 프로세서 및 기타 디바이스들은 시스템내 다중 버스를 통하여 트랜잭션을 동시에 발행할 수 있다. 그래서, 초기에는 트랜잭션이 발행될 때 그 트랜잭션의 순서와 관련하여 모호함이 존재한다. 첫 단계로서 트랜잭션이 시스템을 통과할 때, 시스템은 합당하고 공정한 "발견적 도달 순서(heuristic order of arrival)"를 트랜잭션에 부과한다. 이러한 예비적인 순서는 반드시 트랜잭션이 궁극적으로 시스템내에서 완료되는 순서일 필요는 없다. 충돌하는 두 트랜잭션들이 시스템에서 동시에 액티브하다면, 발견적 도달 순서에 의한 "둘 중의 먼저 것(earlier of the two)"으로 평가된 하나의 트랜잭션이 코히어런스가 달리 요구되지 않는 한 먼저 완료되는 것으로 예정될 것이다.

커맨드가 시스템으로 인가되자마자, 커맨드는 노드 제어기에 의해 "등록(registered)"된다. 즉 커맨드는 노드 제어기에 의해 저장되고 분석 및 충돌 체크에 이용된다. 노드 제어기는 등록된 트랜잭션들을 한 번에 하나씩 어드레스 스위치로 전달한다. 어드레스 스위치는 전달되어진 트랜잭션들을 공정한 중재에 의거 한 번에 하나씩 선택한 다음 선택된 트랜잭션을 다시 노드 제어기 및 메모리 서브시스템으로 브로드캐스트한다. 어드레스 스위치에 의해 브로드캐스트된 트랜잭션의 어드레스 부분은 먼저 노드 제어기내 경계 래치(FROM_ASX_BL)에 래치된다. 상기 기술된 바와 같이, 어느 사이클에서도 유일한 하나의 트랜잭션이 모든 노드 제어기 및 메모리 서브시스템내 FROM_ASX_BL에 래치되며, 그 사이클에 이르기까지 입력되고 등록된 다른 모든 트랜잭션, 즉 현재 FROM_ASX_BL에 있는 트랜잭션을 포함하는 트랜잭션들은 이 트랜잭션을 보게될 것이다. 이러한 두 가지 특성은 다음과 같은 근거와 공정한 발견법, 즉 "트랜잭션의 시스템 도달 순서는 FROM_ASX_BL에 래치된 트랜잭션의 도달 순서와 동일하다" 라는 것을 이용하여 트랜잭션의 도달 순서를 규정하는 데 사용된다.

처음에 트랜잭션이 FROM_ASX_BL에 도달할 때, 그 트랜잭션은 "스누프되는" 것으로 마크됨으로써, 현재 사이클 다음의 일정한 수의 사이클 동안 그 트랜잭션이 스누핑을 위하여 시스템내 모든 디바이스들에게 제공될 것이라는 것을

나타낸다. 다음의 규칙, 즉 "스누프될 것으로 이미 마크된 등록 트랜잭션은 FROM_ASX_BL에 래치된 현재 트랜잭션보다 먼저 시스템으로 입력된 것으로 명목상 규정된다"라는 규칙은 시스템으로 입력되는 실제 시간과 무관하게 트랜잭션이 완료될 순서대로 트랜잭션을 그의 상대적 위치에 할당하는 데 사용된다. 스누프될 것으로 마크된 트랜잭션들은 FROM_ASX_BL에 래치된 현재 트랜잭션보다 나중에 시스템으로 입력된 것으로 명목상 규정된다.

트랜잭션의 정확한 완료 시퀀스를 성취하는 방법(Method for Achieving the Correct Completion Sequence for Transactions)

FROM_ASX_BL에 래치된 트랜잭션은 한 사이클 동안 대기한다. 그 사이클 동안, 트랜잭션은 충돌 검출과 순서 결정을 위하여 전체 시스템에 현재 등록된 모든 트랜잭션과 비교된다. 이러한 짝짓기형 비교 결과는 두 가지가 나올 수 있다. 첫 번째는 FROM_ASX_BL에 현재 래치된 트랜잭션의 완료에 영향을 미치는 것이며, 두 번째는 소정의 다른 트랜잭션의 완료에 영향을 미치는 것이다.

각각의 비교 결과는 FROM_ASX_BL에 래치된 트랜잭션을 스누핑을 위하여 지금 제공하여 완료하게 하는 것 또는 이의 완료를 나중으로 연기하게 하는 것으로 판정된다. 연기하는 것은 AStat Retry 신호 또는 AResp Retry 신호를 계산함으로써 실행된다. 개개의 비교 결과로서 발생되는 이들 신호는 노드 제어기내에서 노드 단위로 하나씩 조합된다. 연기한다는 판정은 최고의 우선순위를 가지며, 그래서 한번의 비교로도 연기를 요청하는 것이 획득되고 그 결과 노드가 트랜잭션을 연기할 것을 투표한다. 노드내부에서 이루어지는 모든 비교 결과가 지금 스누프하여 완료하는 것으로 투표되는 경우에만 노드가 그 트랜잭션을 완료하도록 결정한다.

AStat Retry 신호 및 AResp Retry 신호는 노드 제어기에 의해 AStat Retry 및 AResp Retry코드로 인코딩되어, 스누프되는 트랜잭션의 글로벌 AStat 및 AResp 윈도우에 관여를 위해 RCB에 전달된다. 이들 윈도우 동안 트랜잭션을 발행한 디바이스가 아닌 다른 모든 디바이스와 노드 제어기로부터의 응답은 RCB에 의해 조합되어, 도 10a 내지 도 10d를 참조하여 설명된 바와 같이, 관련된 모든 것들에게 반송되는 글로벌 응답으로 생성된다. 재차 말하건대, 이러한 글로벌 레벨에서, 재시도응답은 (에러 코드를 제외하고는) 최고 우선순위를 가지며, 입력 응답이 모두 재시도인 경우라면 최종 응답이 될 것이다. 글로벌 재시도 응답(a global retry response)의 결과는 트랜잭션을 현재 스누프하는 것을 취소하는 것이다. 트랜잭션에 대한 글로벌 재시도 응답을 알았을 때, 트랜잭션이 등록되어 있는 노드 제어기 는 글로벌 스누프를 위하여 트랜잭션을 재발행(reissue)하거나 상기 트랜잭션을 전달했던 선행 트랜잭션을 회수(retire)시킨다.

이러한 글로벌 재시도는 정확한 순서가 성취될 때까지 반복될 수 있다.

만일 어떠한 이유로 인하여 트랜잭션이 재시도 응답을 수신하는 경우에는, 그 트랜잭션에 대하여 스누프되게 마크된 것은 리셋(reset)되며, 그래서 시스템내에서 트랜잭션 순서인 그의 현재 명목상 위치는 사라지게된다. 트랜잭션이 다시 스누프되게 되면, 트랜잭션은 상술한 규칙에 따라 새로운 위치를 갖게 된다. 이러한 메커니즘은 재발행된 트랜잭션이 그 보다 나중에 시스템에 입력된 다른 트랜잭션 이후로 정렬될 가능성을 방지하는 것은 아니다. 한편, 현재 트랜잭션이 완료하면, 다른 트랜잭션을 재시도하도록 할 수 있다.

트랜잭션의 상태(Phases of a Transaction)

프로세서, I/O 에이전트 등을 연결시키는 공통 버스를 사용하기 보다 본 발명은 노드 제어기를 사용하여 분산 SMP 시스템을 구성한다. 위에서 언급된 바와 같이, 코히어런스를 성취하는 것은 현재 시스템에서 시간과 공간적으로 분산되어 있다. 즉, 다중 노드 제어기에 접속된 다중 버스와 다중 사이클을 거쳐서 이루어진다. 이러한 아키텍처에 따르면, 어느 소정 프로세서의 버스에 나타나는 트랜잭션들 중에서 타이밍 패러독스(timing paradox)가 발생할 수도 있다. 패러독스는 트랜잭션을 프로세서와 그의 노드 제어기에 의해 상이한 시각으로 바라보는데서 발생할 수도 있다. 특히, 프로세서와 그의 노드 제어기는 프로세 서의 버스에 나타내는 트랜잭션들의 시작 순서에 대하여 상이한 시각을 가질 수 있다. 제 1 프로세서가 제 1 트랜잭션을 시스템에 발행하고, 제 2 프로세서가 제 2 트랜잭션을 시스템에 발행하면, 제 1 트랜잭션이 제 2 트랜잭션에 앞서 스누프되는지의 여부에 관하여, 두 트랜잭션의 순서를 보는 제 1 프로세서의 관점은 시스템의 나머지 것들의 관점과 일치할 것이다. 이것은 제 1 프로세서가 정확하게 그의 트랜잭션을 제 2 트랜잭션에 앞서 발행되어진 것으로 보기 때문이다.

그러나, 프로세서가 노드 제어기에 의해 발행된 트랜잭션보다 한 사이클만큼 앞선 트랜잭션을 발행하면, 프로세서는 그 자신의 트랜잭션을 노드 제어기에 의해 발행된 트랜잭션에 앞서서 발행된 것으로 볼 수도 있다. 실제로, 시스템에서 바라보았을 때, 후자의 트랜잭션은 전자의 트랜잭션보다 몇 사이클 앞서서 시스템으로 입력되었을 것이다. 트랜잭션 순서를 보는 두 시각의 불일치는, 두 트랜잭션이 충돌한다면, 프로세서가 시스템에서 본 관점과 달리 코히어런시 응답을 부정확하게 만들게 한다. 노드 제어기는 상이한 시각에 대한 책임을 져야하며, 그래서 자신의 응답을 조정하고 그에 따라 순서 패러독스를 해결한다.

노드 제어기의 코히어런스 동작을 조직하기 위하여, 트랜잭션의 수명은 트랜잭션의 타입에 따라 여러 상태로 분리된다. 트랜잭션은 노드 제어기에 의해 수신되는 시점으로부터 시스템의 관점에서 완료될 때까지 액티브한 것으로 간주된다. 트랜잭션을 고려한 노드 제어기의 코히어런스 동작은 트랜잭션의 현재 상태 및 다른 충돌하는 트랜잭션의 현재 상태의 함수이다.

도 11을 참조하면, 본 발명의 시스템내부에서 트랜잭션의 상태를 정의하는 테이블이 도시된다. 트랜잭션의 상태는 상태 1a 로부터 상태 5까지 연대순으로(chronologically) 정렬된다. 각 상태의 길이, 상태의 시작과 종료의 결정 및 시스템내 트랜잭션의 위치 또는 시스템내 트랜잭션에서 실행되는 동작은 테이블에서 제공된다.

상태 1a는 트랜잭션의 제 1 상태로, 이 상태는 일차적으로 어느 한 노드 제어기의 어느 한 포트에서 트랜잭션을 수신하는 것과 관련된다. 상태 1a의 길이는 포트마다 입력 경계 래치에 래치된 트랜잭션을 시작하고 끝내는 한 사이클이다. 도 6을 참조하면, 상태 1a는 트랜잭션이 경계 래치 IN_BL $_{X}$ (여기서 x는 경계 래치(609-612)와 같이 트랜잭션을 수신한 포트 ID 이다)중의 한 곳에 상주하는 동안의 사이클로 이루어진다.

상태 1b는 트랜잭션의 다음번 상태로, 이 상태는 노드 제어기에 의해 수신되는 트랜잭션에 대한 일차 응답 윈도우의 기간으로 이루어진다. 상태 1b의 길이는 수신되는 트랜잭션의 타입에 의존한다. 이 상태는 시스템내 트랜잭션의 두 번째 사이클에서 시작하며, 이 상태는 일차 어드레스 응답 출력(a Primary Address Response Output)이 노드 제어기에 의해 발행된 트랜잭션으로 인하여 영향 받을 수 있는 최종 사이클에서 종료한다. 이 상태동안, 트랜잭션은 그 트랜잭션을 시스템내로 받아들인 노드 제어기내에서 처리되며, 노드 제어기는 트랜잭션을 발행한 마스터 디바이스에게 전달될 적절한 일차 응답을 결정하는 동안 트랜잭션을 큐한다.
상기 기술된 바와 같이, 모든 트랜잭션은 트랜잭션에 대한 글로벌 코히어런스 응답이 일차 응답 윈도우내에 전달될 수 있는지의 여부를 결정하는 두 가지 카테고리로 분류된다. 상태 1b 동안, 노드 제어기는 일차 응답 윈도우내에 글로벌 코히어런스 응답을 트랜잭션 발행 실체에게 제공할 수 있는 지를 결정한다.
상태 2a는 트랜잭션의 그 다음번 상태이며, 이 상태는 글로벌 스누프를 위하여 트랜잭션의 브로드캐스트를 대기시키는 동안, 트랜잭션이 노드 제어기에 상주해 있는 기간과 연관된다. 이 상태의 길이는 미정이다. 이 상태는 상태 1b가 만료된 후의 사이클에서 시작하며, 이 상태는 트랜잭션이 노드 제어기에 의해 수신되어 글로벌 스누프되기 전 사이클에서 종료한다. 이 상태 동안, 트랜잭션은 노드 제어기내에 큐되고, 글로벌 스누프를 위한 브로드캐스트시에 선택된다. 이 상태의 길이는 트랜잭션이 글로벌 스누프를 위하여 선택될 때 전체 시스템의 상태가 영향을 받게 되므로 미정이다. 이 상태는 어느 노드 제어기내에서고 단 하나뿐인 트랜잭션이 큐되어 있는 경우라면 극히 짧을 것이다. 시스템이 과부하를 받고 있다면, 트랜잭션은 상당수의 사이클을 대기한 다음 스누프되도록 선택될 수도 있다. 도 4를 참조하면, 상태 2a는 트랜잭션이 시스템에서 다른 구성요소로 브로드캐스트되도록 선택될 때까지 트랜잭션이 노드 제어기(415)와 같은 노드 제어기내에 상주하고 있는 기간에 관계한다. 그러므로, 트랜잭션을 버스(416)를 경유하여 어드레스 스위치(430)로 전달되고 또다시 버스(417) 및 다른 버스를 경유하여 전달받아 시스템의 다른 부분으로 전달할 때와 같이, 상태 2a는 트랜잭션이 어드레스 스위치를 통과하는 동안의 사이클을 포함한다.
상태 2b는 트랜잭션의 그 다음번 상태이며, 이 상태는 트랜잭션이 글로벌 스누프를 위하여 노드 제어기에 의해 수신된 사이클과 관련된다. 이 상태의 길이는 단일 사이클이며, 이 상태는 트랜잭션이 경계 래치 FROM_ASX_BL에 있는 동안의 사이클에서 시작하고 종료한다. 도 6을 참조하면, 상태 2b는 트랜잭션이 노드 제어기들로 브로드캐스트되고 경계 래치 FROM_ASX_BL라 지칭된 경계 래치(627)내에 래치되어 있는 동안의 사이클이다. 상기 기술된 바와 같이, 모든 노드 제어기에서 FROM_ASX_BL에는 한 번에 하나의 트랜잭션이 래치된다. 단지 하나의 트랜잭션만이 상태 2b에 있을 수 있다. 이러한 특성은 시스템내에서 완료될 트랜잭션의 상대적인 순서를 규정하는 데 사용된다. 트랜잭션이 이 상태에 도달할 때, 이 트랜잭션은 "스누프된 트랜잭션"이라 지칭되며, 트랜잭션이 등록된 노드 제어기는 그 트랜잭션을 스누프되고 있는 것으로 마크한다. 트랜잭션이 이 상태에 있을 때, 트랜잭션은 시스템의 어느 노드 제어기에서 현재 액티브 상태인 다른 어느 트랜잭션과 충돌하는 지의 여부를 결정하는 것으로 글로벌 충돌에 대하여 검출된다. 이러한 충돌의 결과는 응답 조합 블록에 의해 적절한 사이클 동안 조합되어 그 트랜잭션에 대하여 AStat 및 AResp라는 글로벌 응답을 발생한다.
상태 3은 트랜잭션의 다음번 상태로, 이 상태는 트랜잭션이 노드 제어기를 통과하고 마스터 디바이스에게 글로벌 스누프를 위해 브로드캐스트되는 동안의 기간과 관련된다. 이 상태의 길이는 시스템 구현에 좌우되는 일정 개수의 사이클, 즉 구현된 노드 제어기내의 스누프 래치와 포트간의 사이클 수와 같다. 이 상태는 상태 2b가 끝난 이후의 사이클에서 시작하며, 이 상태는 노드 제어기가 트랜잭션에 대한 Global Address Response In을 인식할 때 종료한다. 이 상태 동안, 트랜잭션은 노드 제어기에 접속된 마스터 디바이스에 의해 스누프된다. 도 6을 참조하면, 상태 3은 트랜잭션이 경계 래치(FROM_ASX_BL)로부터 노드 제어기의 포트로 이동되어 노드 제어기에 접속된 버스로 브로드캐스트되는 동안의 사이클을 포함한다. 또한 상태 3은 마스터 디바이스가 응답 조합 블록에 의해 조합된 응답을 발생하여 스누프된 트랜잭션에 대한 글로벌 응답을 생성하는 동안의 사이클을 포함한다.
상태 4는 트랜잭션의 그 다음번 상태로, 이 상태는 트랜잭션의 완료 전에 발생하는 프로세싱과 관련된다. 상태 4는 다음의 두 가지 카테고리의 트랜잭션, 즉, 판독 트랜잭션 및 비-판독 트랜잭션에 관하여 기술될 수 있다. 이 상태의 길이는 트랜잭션의 타입에 따라 달라진다. 이 상태는 상태 3이 끝난 후의 사이클에서 시작하며, 이 상태는 트랜잭션의 카테고리에 예속하는 한 시점에서 종료한다. 판독 트랜잭션의 경우, 이 상태는 요청자에게 데이터 전송이 시작되기 전의 사이클에서 종료한다. 판독 트랜잭션이 아닌 경우, 이 상태는 시스템과 관련하여 트랜잭션의 완료와 함께 종료한다.
상태 5는 트랜잭션의 그 다음번 상태로, 이 상태는 판독 트랜잭션의 완료와 관련된다. 상태 4에 대하여 상기 언급한 바와 같이, 트랜잭션의 완료는 판독 트랜잭션(read transaction)과 비-판독 트랜잭션(non-read transaction)으로 분류된다. 비-판독 트랜잭션의 경우, 상태 4는 트랜잭션의 최종 상태이다. 상태 5는 판독 트랜잭션에 대하여만 규정되며, 상태 5의 길이는 판독 트랜잭션의 타입과 판독 트랜잭션으로 인하여 전송될 데이터의 양에 따라 달라진다. 이 상태는 상태 4가 만료한 후 사이클에서 시작하며, 이 상태는 시스템에 대하여 판독 트랜잭션의 완료와 함께 종료한다.

<u>트랜잭션의 타입(Types of Transactions)</u>

트랜잭션은 하기 사항, 즉 트랜잭션의 가능한 최종 글로벌 코히어런시 응답; 최종 글로벌 코히어런시 응답이 트랜잭션을 발행한 마스터들에게 전달될 수 있을 때; 그리고 트랜잭션 타입에 기초하여 충돌 검출 목적을 위해 분류된다. 하기 카테고리는 글로벌 코히어런시 응답을 결정하는 데 사용된다.
캐쉬 라인의 코히어런시 상태가 데이터와 함께 보고되는 판독 커맨드(Read command),
코히어런시 응답이 널(Null)임을 보장하는 판독 커맨드,
재실행(Rerun)이라는 일차 응답이 주어진 판독 커맨드,
실제로 전역적으로 스누프되어야만 하면서, 6XX 프로토콜의 DClaim 및 RWITM 트랜잭션과 같이 글로벌 응답이 예측될 수 없는 커맨드,
Read와 다르면서, Clean, DKill, Flush 등과 같이, 최종 글로벌 코히어런시가 널인 것으로 예측될 수 있는 커맨드,

WWC/WWK M=0와 같이, 실제로 마스터에 의해 스누프되지 않는 넌-코히어런트 기록 커맨드(Writes),
WWK/WWF M=1와 같은 코히어런트 기록 커맨드,
SYNC TLBIE와 같이, 코히어런시와 관련한 충돌을 받지않는 기타 커맨드.

노드 제어기의 코히어런시 조치(Node Controller Coherency Actions)

스누프된 트랜잭션과 충돌하는 노드 제어기, 즉 로컬 내지 로컬 제어기내에서 등록 또는 큐된 트랜잭션에 대하여 노드 제어기에 의해 부여된(contributed) 일차 및 글로벌 코히어런시 응답은 다음과 같은 조건, 즉 로컬 트랜잭션의 타입과 상태, 그리고 노드 제어기가 응답을 준 시점에 이르기까지 트랜잭션을 수신한 AStast 및 AResp, 스누프된 트랜잭션의 타입, 스누프된 트랜잭션 대 다른 스누프된 트랜잭션의 시간적 근사치 및 시스템에서 구현되는 버스 프로토콜의 함수이다.

노드 제어기내부에서 충돌하는 각 트랜잭션 쌍들이 발생할 때마다, 노드 제어기는 응답 조합 블록에 의해 결정된 응답으로 입력, 즉 AStat 및 AResp 응답을 부여한다. 예로, 6XX 프로토콜에 있어서, AStat 응답은 Null, Ack, 또는 Retry 일 수 있고, AResp는 Null, Shared 또는 Retry일 수 있다. 부가적으로, 충돌하는 각 트랜잭션의 쌍마다, AResp 응답은 조건적 또는 비조건적일 수도 있다. 그러므로, 충돌하는 각 트랜잭션의 쌍의 경우, 각각의 노드 제어기는 그의 응답을 결정하는 데, 그 응답은 응답 결정에 적용되는 조건적인 규칙을 사용할 수도 있다.

이제 도 12a-12b를 참조하면, 트랜잭션 충돌 쌍의 검출에 응답하여 노드 제어기가 발생시킨 응답을 설명하는 테이블이 도시된다.

도 12a는 DClaim 트랜잭션과 Read 트랜잭션의 충돌 쌍의 응답 테이블로서, 이 경우 캐쉬 라인의 코히어런시 상태는 노드 제어기에 의해 생성되어질 데이터와 함께 보고된다. 테이블에서 "X"는 노드 제어기가 이런 경우의 충돌에서 트랜잭션에 대한 "반대(adverse)" 응답을 하지 않는다는 것, 예로, 6XX 버스 프로토콜에서, 노드 제어기는 Null 응답을 제공하고 Retry는 제공하지 않는다는 것을 나타낸다. 본 실시예에서, DClaim은 로컬 트랜잭션, 즉 노드 제어기에서 수신되거나, 큐되거나 또는 등록되어진 트랜잭션이며, Read 트랜잭션은 스누프되는 트랜잭션, 즉 노드 제어기의 경계 래치 FROM_ASX_BL에 상주하면서 그것이 등록되어 있는 노드 제어기에 대하여 상태 2b에 있는 트랜잭션이다.

상태 1a 및 상태 1b는 Primary Response 윈도우내에 있는 상태들을 나타낸다. 그러므로, 노드 제어기는 이들 상태에서 스누프된 트랜잭션에 대하여 Null 응답을 제공한다. 상태 2a에서, 로컬 트랜잭션 또는 글로벌 트랜잭션은 글로벌 응답(Global Response)을 제공받을 수 있다. 상태 2b는 응답 테이블에서 항상 공백으로 제시되는 데, 이것은 스누프된 트랜잭션이 항상 상태 2b에, 즉 항시 FROM_ASX_BL 경계 래치내에 상주하기 때문이며, 시스템내에서는 어느 소정시간에 단지 하나의 트랜잭션만이 이 상태에 있을 수 있으므로, 로컬 트랜잭션과 스누프된 트랜잭션은 충돌하지 않는다. 상태 3 및 상태 4에 있어서, 스누프된 트랜잭션은 로컬 트랜잭션이 비교적 완료에 가까워짐에 따라 그의 글로벌 응답(Global Response)을 수신할 수 있다.

다시 도 12a를 참조하면, 노드 제어기가 상태 1a에서 DClaim 트랜잭션을 가지고 있고 스누프될 Read 트랜잭션을 수신한다면, 노드 제어기는 DClaim 트랜잭션에 대하여 Primary AStat Retry를 제공한다. 그러나, DClaim 트랜잭션에 대한 Primary AResp 응답은 DClaim 트랜잭션이 등록된 노드 제어기에 대하여 영향을 미치지 않는다. Read 트랜잭션에 대한 Global AStat나 AResp 응답은 어느 것도 충돌에 의해 영향 받지 않는다. 노드 제어기가 상태 1b에서 DClaim 트랜잭션을 가지고 있고 스누프될 Read 트랜잭션을 수신한다면, 노드 제어기는 DClaim 트랜잭션에 대한 Primary AStat 응답을 부여하지 못한다. 그러나, DClaim 트랜잭션에 대한 Primary AResp 응답은 DClaim 트랜잭션이 등록되어 있는 노드 제어기로부터 Retry를 수신한다. 다시 한번 언급하건대, Read 트랜잭션에 대한 Global AStat나 AResp 응답은 어느 것도 충돌에 영향 받지 않는다.

노드 제어기가 상태 2a에서 DClaim트랜잭션을 가지고 있고 스누프될 Read 트랜잭션을 수신한다면, DClaim 트랜잭션에 대한 Global AResp 응답은 DClaim 트랜잭션이 등록되어 있는 노드 제어기로부터 Retry를 수신한다. 이러한 특정 응답은 "자기-재시도(self-retry)"라 지칭한다. 트랜잭션의 상태 2a가 트랜잭션이 그의 로컬 노드 제어기내에서 큐되어 있는 기간을 나타내고 있을 때, 이 응답은 나중에 사용하기 위하여 로컬 노드 제어기에 저장된다. 본 실시예에서, DClaim 트랜잭션이 나중에 글로벌 스누프 동안 제시될 때, 그의 로컬 노드 제어기는 적절한 시점에 저장되어 있는 자기-재시도 응답을 발행할 것이다. 비록 글로벌 스누프 동안 DClaim 트랜잭션이 제시되기 이전에 DClaim 트랜잭션과 충돌하는 Read 트랜잭션이 이미 상당한 기간 전에 완료되었을 지라도, 상기 응답이 캐쉬 코히어런시를 유지하기 위하여 트랜잭션의 적절한 완료 순서를 확실히 하는데 필요하므로 이와 같은 특정 충돌 시나리오에서 DClaim은 "낭비적(loses)"이다.

노드 제어기가 상태 3에서 DClaim 트랜잭션을 가지고 있고 스누프될 Read 트랜잭션을 수신하면, Read 트랜잭션에 대한 Global AResp 응답은 DClaim 트랜잭션이 등록되어 있는 노드 제어기로부터 Retry 응답을 수신할 수도 있다. 이러한 Retry는 충돌 DClaim 트랜잭션의 진행을 조건으로 한다. DClaim 트랜잭션이 Global Retry를 수신하지 못하면, 테이블에 도시된 바와 같이, Read 트랜잭션은 충돌 DClaim 트랜잭션이 등록되어 있는 노드 제어기로부터 Retry를 수신한다. DClaim 트랜잭션이 Global Retry를 수신하면, Read 트랜잭션은 충돌 DClaim 트랜잭션이 등록되어 있는 노드 제어기로부터 Null 응답을 수신한다. 즉 테이블의 Retry가 Null로 변환된다.

노드 제어기가 상태 4에서 DClaim 트랜잭션을 가지고 있고 스누프될 Read 트랜잭션을 수신하면, 테이블에 예시된 바와 같이, Read 트랜잭션에 대한 Global AResp 응답은 DClaim 트랜잭션이 등록되어 있는 노드 제어기로부터 Retry 응답을 수신한다. 이러한 Retry는 충돌 DClaim 트랜잭션의 진행을 조건으로 하지는 않는다.

도 12b는 노드 제어기가 DClaim 트랜잭션과 Read 트랜잭션과의 충돌 쌍에 대하여 생성하였을 응답 테이블을 예시한다. 다시 언급하건대, 테이블에서 "X"는 노드 제어기가 이런 경우의 충돌을 일으키는 트랜잭션에 대한 "반대(adverse)" 응답을 하지 않는다는 것, 예로, 6XX 버스 프로토콜에서, 노드 제어기는 Null 응답을 제공하고 Retry는 제공하지 않는다는 것을 나타낸다. 본 실시예에서, 도 12a와 반대로, Read 트랜잭션은 로컬 트랜잭션, 즉 노드 제어기에서 수

신되거나, 큐되거나 또는 등록되어진 트랜잭션이며, DClaim 트랜잭션은 스누프되는 트랜잭션, 즉 노드 제어기의 경계 래치 FROM_ASX_BL에 상주하면서 그것이 등록되어 있는 노드 제어기에 대하여 상태 2b에 있는 트랜잭션이다.
다시 도 12b를 참조하면, 노드 제어기가 상태 1a에서 Read 트랜잭션을 가지고 있고 스누프될 DClaim 트랜잭션을 수신하면, 노드 제어기는 Read 트랜잭션에 대한 Primary AStat Retry를 제공한다. 그러나, Read 트랜잭션에 대한 Primary AStat AResp는 Read 트랜잭션이 등록되어 있는 노드 제어기에 대하여 영향받지는 않는다. DClaim 트랜잭션에 대한 Global AStat 응답이나 Global AResp 응답 어느 것도 충돌에 영향 받지 않는다. 노드 제어기가 상태 2a에서 Read 트랜잭션을 가지고 있고 스누프될 DClaim을 수신한다면, 노드 제어기는 Read 트랜잭션에 대한 "반대" Global AStat 응답이나 Global AResp 응답 어느 것도 제공하지 않는다. 그러나, DClaim 트랜잭션에 대한 Global AStat 응답은 충돌에 영향 받지 않지만, DClaim 트랜잭션에 대한 Global AResp 응답은 노드 제어기로부터 Retry를 수신한다.
노드 제어기가 상태 3 또는 상태 4에서 Read 트랜잭션을 가지고 있고 스누프될 DClaim을 수신한다면, 노드 제어기는 Read 트랜잭션에 대한 "반대" Global AStat 응답이나 Global AResp 응답 어느 것도 제공하지 않는다. 그러나, DClaim 트랜잭션에 대한 Global AStat 응답은 충돌에 영향 받지 않지만, DClaim 트랜잭션에 대한 Global AResp 응답은 어느 경우에서도 노드 제어기로부터 Retry를 수신한다. 이들 Retry는 양 경우에서 조건부는 아니다.
도 12a와 도 12b의 테이블을 비교함으로써, 이들 테이블들이 서로 똑같지 않은 것을 알 수 있다. 즉 응답 패턴이 충돌 트랜잭션 쌍마다 반드시 대칭은 아니라는 것이다. 그러한 응답은 미리 계산되고 인코드(encode)될 수도 있으며, 이러한 코드는 마이크로프로그램(microprogram)의 일부로서 ROM에 저장될 수도 있다. 충돌이 발생할 때, 적절한 마이크로워드(microword)가 액세스되어 필요한 응답을 재생할 수 있다. 이와 달리, 응답은 논리 게이트를 이용하여 하드코드(hardcode)화될 수도 있다.

발명의 효과

본 발명의 장점은 상세한 설명으로부터 자명해질 것이다. 본 발명은 표준화되고 검증이 용이한 버스-기반 캐쉬-코히어런스 트로토콜을 물리적 버스를 프로세서, 메모리 서브시스템 및 I/O 에이전트와 같은 시스템 구성요소들간에 통신하는데 비능률적인 매체로 만들어버리는 커다란 크기의 대규모 멀티프로세서 시스템으로 확대시켜준다. 본 발명의 분산 시스템 구조를 이용함으로써, 보다 복잡한 디렉토리-기반 프로토콜 등의 개발이 불필요하다. 또한 본 발명은 컴포넌트 인터페이스(component interface)를 단일 버스에서 가능한 것보다 빠르게 클럭되게 해줌으로써, 컴포넌트 인터페이스의 대역폭이 향상되고 그 결과 전체 시스템의 대역폭과 성능이 더욱 높아진다. 본 발명은 또한 다중 데이터 버스를 지원함으로써, 시스템의 데이터 대역폭이 배가되고 프로세서의 효율이 증진된다. 본 발명의 데이터 전송 병행 기법은 또한 전체 시스템의 데이터 처리량을 향상시킨다.
본 발명이 전기능 데이터 프로세싱 시스템의 문맥으로 기술되었지만, 본 기술분야에서 통상의 지식을 가진 자에게는 본 발명의 프로세스가 명령어의 컴퓨터 판독가능 매체의 타입 및 각종 타입으로 분산될 수 있고 본 발명이 그러한 분산을 실행하는데 실제로 사용된 매체를 담고 있는 특정 타입의 신호에도 동일하게 적용가능하다는 것이 인식될 것이다. 컴퓨터 판독가능 매체의 예는 플로피 디스크, 하드 디스크 드라이브, RAM, CD-ROM과 같은 판독가능형 매체와 디지털 및 아날로그 통신 링크와 같은 전송형 매체를 포함한다.
본 발명의 설명은 설명 목적으로 제시되었지만, 개시되어 있는 타입으로 제한하려는 것은 아니다. 본 기술분야에서 통상의 지식을 가진 자에게는 많은 변경과 변화가 자명할 것이다. 본 발명의 원리를 최선으로 설명하고 본 기술분야에서 통상의 지식을 가진 자에게 본 발명을 이해시키기 위하여 실시예가 선택되고 기술되었다.

(57) 청구의 범위

청구항 1.
멀티프로세서 시스템에서 캐쉬 코히어런시를 유지하는 방법에 있어서,
노드 제어기에서 로컬 사이클 동안 트랜잭션을 수신하는 단계와,
글로벌 사이클 동안 글로벌 스누프를 위해 마스터 디바이스에 상기 트랜잭션을 브로드캐스팅하는 단계와,
상기 트랜잭션에 대한 코히어런시 응답을 결정하는 데 하나 이상의 입력을 부여하는(contributing) 단계를 포함하는 멀티프로세서 시스템에서 캐쉬 코히어런시 유지 방법.

청구항 2.
제 1 항에 있어서,
상기 코히어런시 응답을 상기 트랜잭션의 일차 응답 윈도우(a primary response window)에서 상기 트랜잭션의 요청자에게 제공하는 단계를 더 포함하는 멀티프로세서 시스템에서 캐쉬 코히어런시 유지 방법.

청구항 3.
제 1 항에 있어서,
상기 코히어런시 응답을 상기 트랜잭션의 글로벌 응답 윈도우(a global response window)에서 상기 트랜잭션의 요청자에게 제공하는 단계를 더 포함하는 멀티프로세서 시스템에서 캐쉬 코히어런시 유지 방법.

청구항 4.
제 3 항에 있어서,
상기 코히어런시 응답은 상기 트랜잭션이 또 다른 트랜잭션과 충돌하는 결정에 응답하여 이전에 저장되어 있었던 멀

티프로세서 시스템에서 캐쉬 코히어런시 유지 방법.

청구항 5.

제 3 항에 있어서,

상기 코히어런시 응답은 충돌 트랜잭션의 완료에 대하여 조건적(conditional)인 멀티프로세서 시스템에서 캐쉬 코히어런시 유지 방법.

청구항 6.

제 3 항에 있어서,

상기 코히어런시 응답은 충돌 트랜잭션의 완료에 대하여 무조건적인(unconditional) 멀티프로세서 시스템에서 캐쉬 코히어런시 유지 방법.

청구항 7.

제 1 항에 있어서,

다음의 조건들, 즉 상기 트랜잭션의 상태, 상기 트랜잭션의 트랜잭션 타입, 스누프된 트랜잭션의 트랜잭션 타입, 및 상기 노드 제어기와 상기 트랜잭션의 요청자간의 통신을 위한 버스 프로토콜 중의 하나 이상의 조건에 따라서 상기 코히어런시 응답을 위해 부여된 입력을 결정하는 단계를 더 포함하는 멀티프로세서 시스템에서 캐쉬 코히어런시 유지 방법.

청구항 8.

제 1 항에 있어서,

상기 트랜잭션의 상태에 따라 상기 코히어런시 응답을 위해 부여된 입력을 결정하는 멀티프로세서 시스템에서 캐쉬 코히어런시 유지 방법.

청구항 9.

제 8 항에 있어서,

상기 트랜잭션의 상태는 다수의 상태들 중의 하나로서 상기 노드 제어기에 의해 결정가능한 멀티프로세서 시스템에서 캐쉬 코히어런시 유지 방법.

청구항 10.

제 9 항에 있어서,

상기 트랜잭션의 상기 다수 상태들에서 하나의 상태는 상기 노드 제어기내 상기 트랜잭션의 제 1 사이클로 이루어지는 기간으로 규정되는 멀티프로세서 시스템에서 캐쉬 코히어런시 유지 방법.

청구항 11.

제 9 항에 있어서,

상기 트랜잭션의 상기 다수 상태들에서 하나의 상태는 상기 노드 제어기 내 상기 트랜잭션의 제 2 사이클에서 상기 트랜잭션에 대한 일차 응답을 위한 최종 사이클까지로 이루어지는 기간으로 규정되는 멀티프로세서 시스템에서 캐쉬 코히어런시 유지 방법.

청구항 12.

제 9 항에 있어서,

상기 트랜잭션의 상기 다수 상태들에서 하나의 상태는 선행 상태 이후 제 1 사이클에서 상기 노드 제어기가 상기 트랜잭션을 스누프된 트랜잭션으로서 수신하기 이전의 최종 사이클까지로 이루어지는 기간으로 규정되는 멀티프로세서 시스템에서 캐쉬 코히어런시 유지 방법.

청구항 13.

제 9 항에 있어서,

상기 트랜잭션의 상기 다수 상태들에서 하나의 상태는 상기 노드 제어기가 상기 트랜잭션을 스누프된 트랜잭션으로서 수신하는 제 1 사이클로 이루어지는 기간으로 규정되는 멀티프로세서 시스템에서 캐쉬 코히어런시 유지 방법.

청구항 14.

제 9 항에 있어서,

상기 트랜잭션의 상기 다수 상태들에서 하나의 상태는 선행 상태 이후 상기 트랜잭션에 대한 글로벌 코히어런시 응답이 상기 노드 제어기에 의해 수신될 때까지의 제 1 사이클로 이루어지는 기간으로 규정되는 멀티프로세서 시스템에서 캐쉬 코히어런시 유지 방법.

청구항 15.

제 9 항에 있어서,

상기 트랜잭션의 상기 다수 상태들에서 하나의 상태는 선행 상태 이후 상기 트랜잭션이 완료될 때까지의 제 1 사이클로 이루어지는 기간으로 규정되는 멀티프로세서 시스템에서 캐쉬 코히어런시 유지 방법.

청구항 16.

제 9 항에 있어서,

상기 트랜잭션이 판독 트랜잭션(a read transaction)이면, 상기 트랜잭션의 상기 다수 상태들에서 하나의 상태는 선행 상태 이후 제 1 사이클에서 데이터를 상기 트랜잭션의 요청자에게 전송하기 위한 제 1 사이클까지로 이루어지는 기간으로 규정되는 멀티프로세서 시스템에서 캐쉬 코히어런시 유지 방법.

청구항 17.

제 1 항에 있어서,

상기 트랜잭션의 코히어런시 응답은 응답 조합 블록(a response combination block)에 의해 생성되는 멀티프로세서 시스템에서 캐쉬 코히어런시 유지 방법.

청구항 18.

제 1 항에 있어서,
상기 수신된 트랜잭션을 상기 노드 제어기에 등록하는 단계를 더 포함하는 멀티프로세서 시스템에서 캐쉬 코히어런시 유지 방법.

청구항 19.

제 1 항에 있어서,
상기 멀티프로세서 시스템은
상기 노드 제어기와,
다수의 마스터 디바이스와,
다수의 양방향 마스터 디바이스 버스를 포함하며,
각각의 상기 양방향 마스터 디바이스 버스는 노드내의 하나 이상의 마스터 디바이스를 상기 노드 제어기의 포트에 연결하는 멀티프로세서 시스템에서 캐쉬 코히어런시 유지 방법.

청구항 20.

제 19 항에 있어서,
상기 노드 제어기는
다수의 마스터 디바이스 포트와,
한 쌍의 어드레스 스위치 포트와,
다수의 메모리 서브시스템 포트를 포함하며,
각각의 상기 마스터 디바이스 포트는 마스터 디바이스 버스에 연결되며,
각각의 상기 어드레스 스위치 포트는 한 쌍의 단방향 어드레스 스위치 버스중의 하나에 연결되고, 상기 쌍의 어드레스 스위치 버스중의 하나는 어드레스를 상기 노드 제어기로부터 상기 어드레스 스위치로 전달하며, 상기 쌍의 어드레스 스위치중의 나머지 하나는 어드레스를 상기 어드레스 스위치로부터 상기 노드 제어기로 전달하며,
각각의 상기 메모리 서브시스템 포트는 다수의 양방향 메모리 서브시스템 버스들 중의 하나에 연결되고, 각각의 양방향 메모리 서브시스템 버스는 상기 노드 제어기와 다수의 메모리 서브시스템 중의 하나 사이에서 데이터를 전달하는 멀티프로세서 시스템에서 캐쉬 코히어런시 유지 방법.

청구항 21.

멀티프로세서 시스템에서 캐쉬 코히어런시를 유지하는 방법에 있어서,
노드 제어기에서 로컬 사이클 동안 트랜잭션을 수신하는 단계와,
글로벌 코히어런스 응답을 예측할 수 있는지를 결정하는 단계와,
글로벌 코히어런스 응답을 예측할 수 있다면 글로벌 코히어런스 응답을 예측하는 단계와,
상기 트랜잭션에 대한 일차 응답 윈도우에서 상기 트랜잭션의 요청자에게 상기 예측된 글로벌 코히어런스 응답을 제공하는 단계를 포함하는
멀티프로세서 시스템에서 캐쉬 코히어런시 유지 방법.

청구항 22.

제 21 항에 있어서,
상기 글로벌 코히어런스 응답을 예측할 수 없다면 상기 트랜잭션에 대한 일차 응답 윈도우에서 상기 트랜잭션의 요청자에게 글로벌 코히어런스 응답의 연기를 나타내는 일차 응답을 전달하는 단계를 더 포함하는 멀티프로세서 시스템에서 캐쉬 코히어런시 유지 방법.

청구항 23.

멀티프로세서 시스템에 있어서,
노드 제어기와,
다수의 마스터 디바이스와,
다수의 양방향 마스터 디바이스 버스를 포함하며,
각각의 상기 양방향 마스터 디바이스 버스는 노드 내의 하나 이상의 마스터 디바이스를 상기 노드 제어기의 포트에 연결하는
멀티프로세서 시스템.

청구항 24.

제 23 항에 있어서,
상기 노드 제어기는
다수의 마스터 디바이스 포트와,
한 쌍의 어드레스 스위치 포트와,
다수의 메모리 서브시스템 포트를 포함하며,
각각의 상기 마스터 디바이스 포트는 마스터 디바이스 버스에 연결되며,
각각의 상기 어드레스 스위치 포트는 한 쌍의 단방향 어드레스 스위치 버스중의 하나에 연결되고, 상기 쌍의 어드레스 스위치 버스중의 하나는 어드레스를 상기 노드 제어기로부터 상기 어드레스 스위치로 전달하며, 상기 쌍의 어드레스 스위치중의 나머지 하나는 어드레스를 상기 어드레스 스위치로부터 상기 노드 제어기로 전달하며,

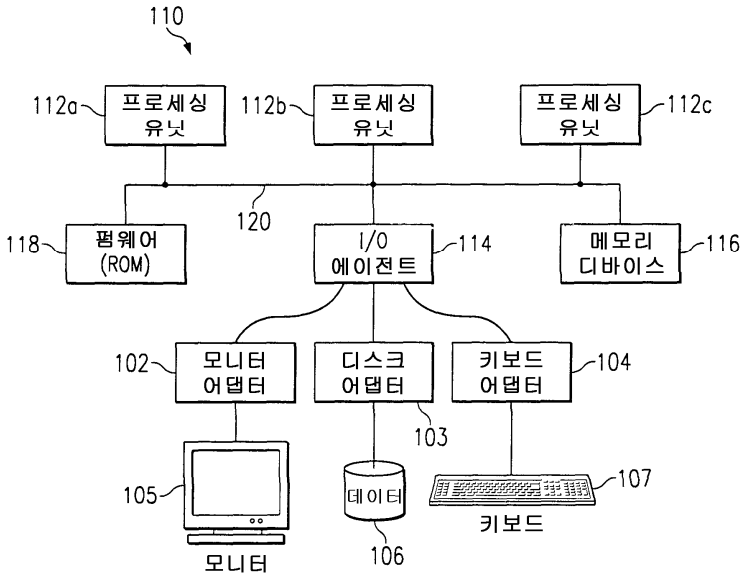
각각의 상기 메모리 서브시스템 포트는 다수의 양방향 메모리 서브시스템 버스들 중의 하나에 연결되고, 각각의 상기 양방향 메모리 서브시스템 버스는 상기 노드 제어기와 다수의 메모리 서브시스템 중의 하나 사이에서 데이터를 전달하는
멀티프로세서 시스템.

청구항 25.

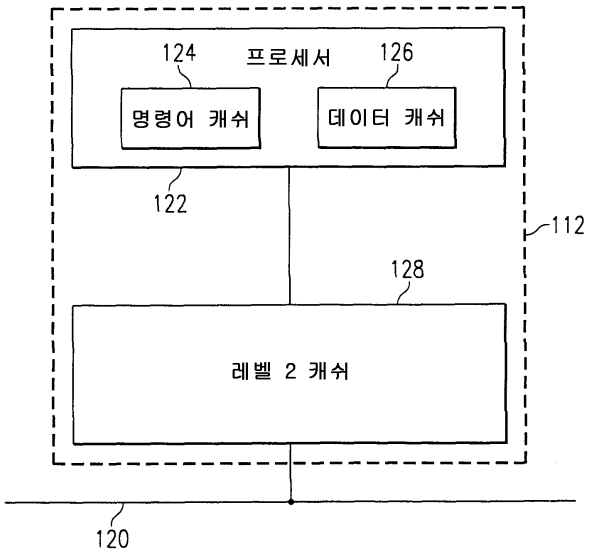
노드 제어기에 있어서,
다수의 마스터 디바이스 포트와,
한 쌍의 어드레스 스위치 포트와,
다수의 메모리 서브시스템 포트를 포함하며,
각각의 상기 마스터 디바이스 포트는 마스터 디바이스 버스에 연결되며,
각각의 상기 어드레스 스위치 포트는 한 쌍의 단방향 어드레스 스위치 버스중의 하나에 연결되고, 상기 쌍의 어드레스 스위치 버스중의 하나는 어드레스를 상기 노드 제어기로부터 상기 어드레스 스위치로 전달하며, 상기 쌍의 어드레스 스위치중의 나머지 하나는 어드레스를 상기 어드레스 스위치로부터 상기 노드 제어기로 전달하며,
각각의 상기 메모리 서브시스템 포트는 다수의 양방향 메모리 서브시스템 버스들 중의 하나에 연결되고, 각각의 상기 양방향 메모리 서브시스템 버스는 상기 노드 제어기와 다수의 메모리 서브시스템 중의 하나 사이에서 데이터를 전달하는
노드 제어기.

도면

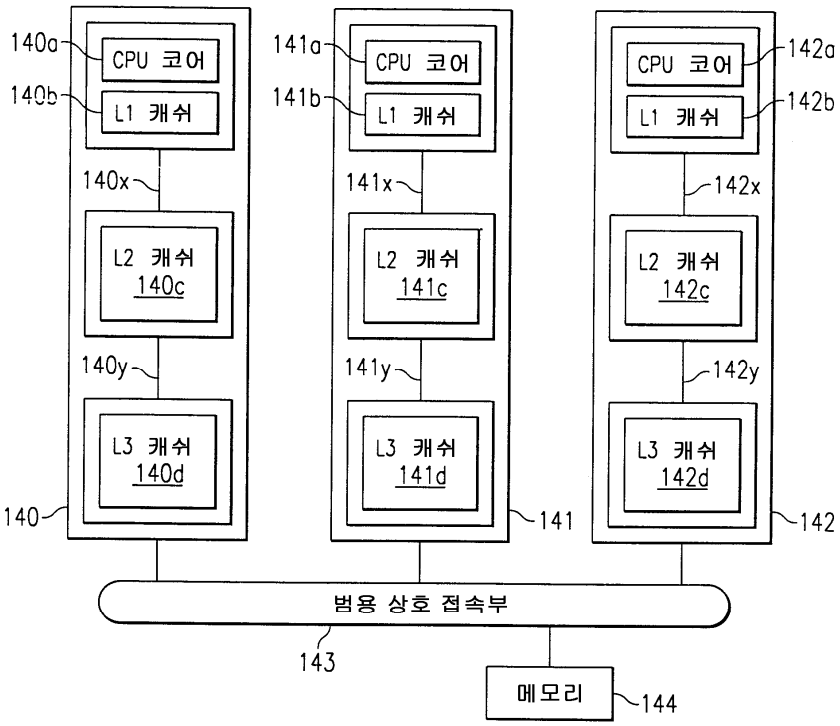
도면1



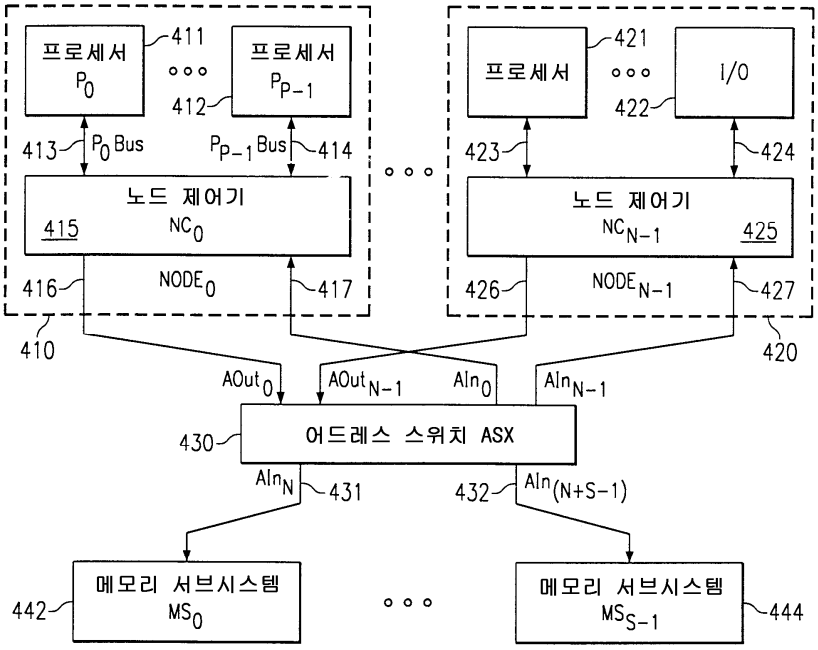
도면2



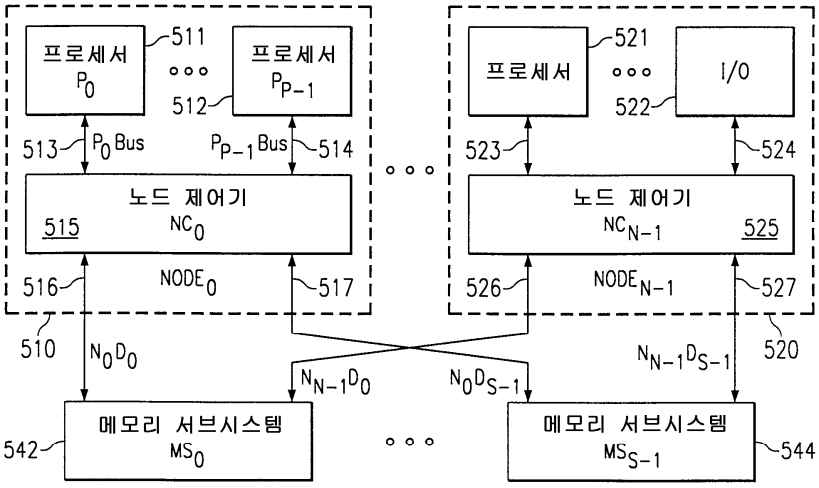
도면3


140a
CPU 코어
140b
L1 캐쉬
140x
L2 캐쉬
140c
140y
L3 캐쉬
140d
140
141a
CPU 코어
141b
L1 캐쉬
141x
L2 캐쉬
141c
141y
L3 캐쉬
141d
141
142a
CPU 코어
142b
L1 캐쉬
142x
L2 캐쉬
142c
142y
L3 캐쉬
142d
142
범용 상호 접속부
143
메모리
144

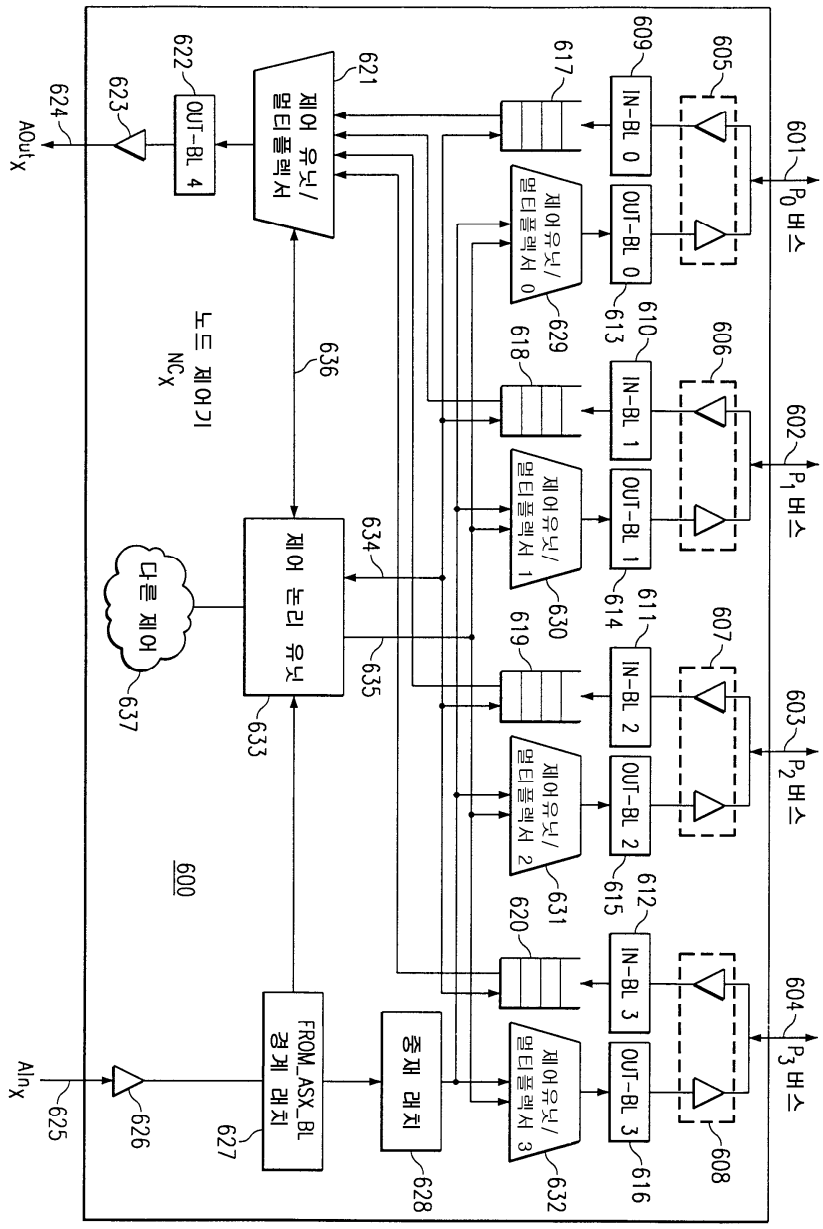
도면4


프로세서
P0
411
프로세서
PP-1
412
413
P0 Bus
PP-1 Bus
414
노드 제어기
415
NC0
416
NODE0
417
410
프로세서
421
I/O
422
423
424
노드 제어기
NCN-1
425
426
NODEN-1
427
420
AOut0
AOutN-1
AIn0
AInN-1
430
어드레스 스위치 ASX
AInN
431
432
AIn(N+S-1)
메모리 서브시스템
MS0
442
메모리 서브시스템
MSS-1
444

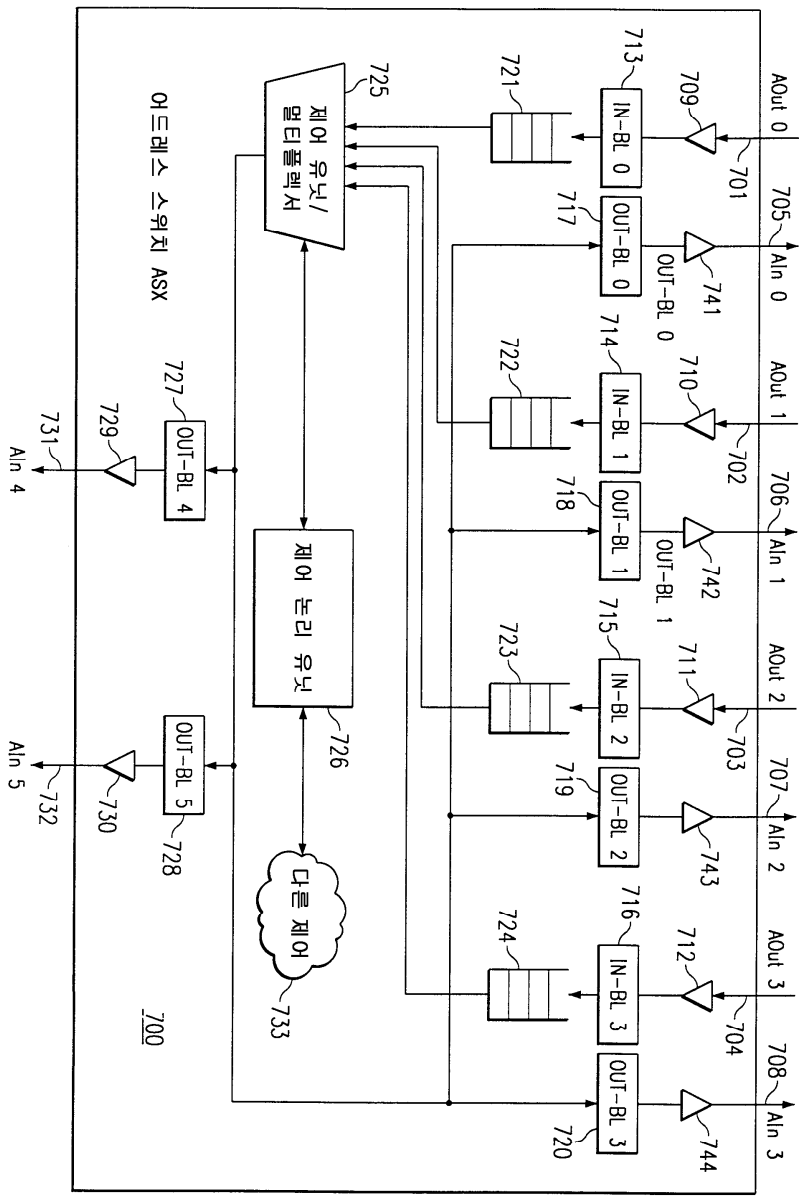
도면5


프로세서 P_0
511
프로세서 P_{P-1}
512
513
P_0 Bus
P_{P-1} Bus
514
노드 제어기 NC_0
515
$NODE_0$
516
517
510
N_0D_0
$N_{N-1}D_0$
N_0D_{S-1}
$N_{N-1}D_{S-1}$
520
542
메모리 서브시스템 MS_0
메모리 서브시스템 MS_{S-1}
544
프로세서
521
I/O
522
523
524
노드 제어기 NC_{N-1}
525
$NODE_{N-1}$
526
527

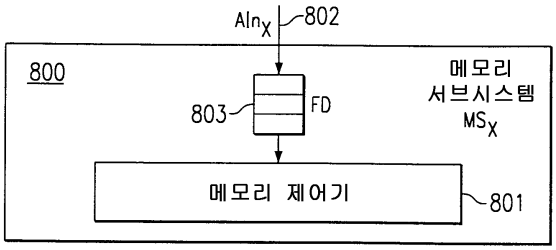
도면6


600
노드 제어기 NC_X
제어 유닛/ 멀티플렉서
621
OUT-BL 4
622
623
624
$AOut_X$
636
제어 논리 유닛
633
634
635
다른 제어
637
FROM_ASX_BL 경계 래치
627
626
625
AIn_X
중재 래치
628
IN-BL 0
609
OUT-BL 0
613
제어유닛/ 멀티플렉서 0
629
617
IN-BL 1
610
OUT-BL 1
614
제어유닛/ 멀티플렉서 1
630
618
IN-BL 2
611
OUT-BL 2
615
제어유닛/ 멀티플렉서 2
631
619
IN-BL 3
612
OUT-BL 3
616
제어유닛/ 멀티플렉서 3
632
620
605
606
607
608
601 P_0 버스
602 P_1 버스
603 P_2 버스
604 P_3 버스

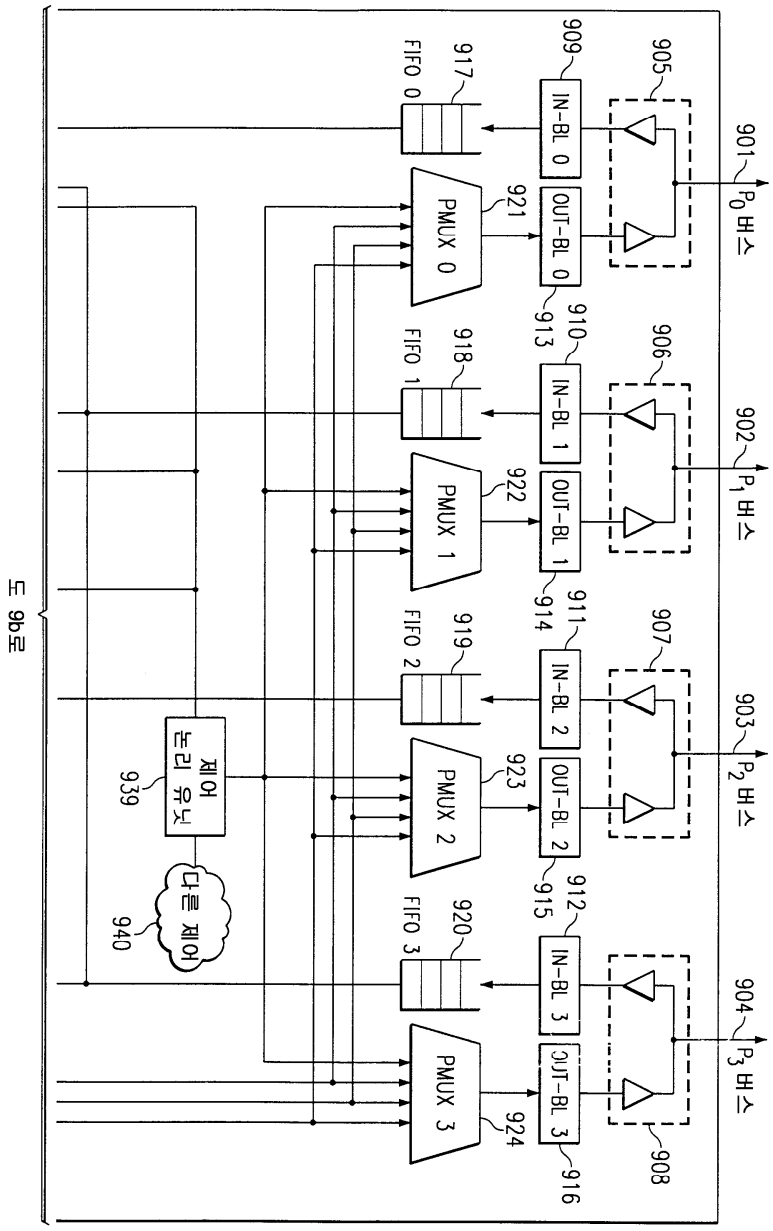
도면7


어드레스 스위치 ASX
700
제어 유닛/
멀티플렉서
725
제어 논리 유닛
726
다른 제어
733
IN-BL 0
IN-BL 1
IN-BL 2
IN-BL 3
OUT-BL 0
OUT-BL 1
OUT-BL 2
OUT-BL 3
OUT-BL 4
OUT-BL 5
721
722
723
724
717
718
719
720
727
728
729
730
731
732
AOut 0
AOut 1
AOut 2
AOut 3
AIn 0
AIn 1
AIn 2
AIn 3
AIn 4
AIn 5
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
741
742
743
744

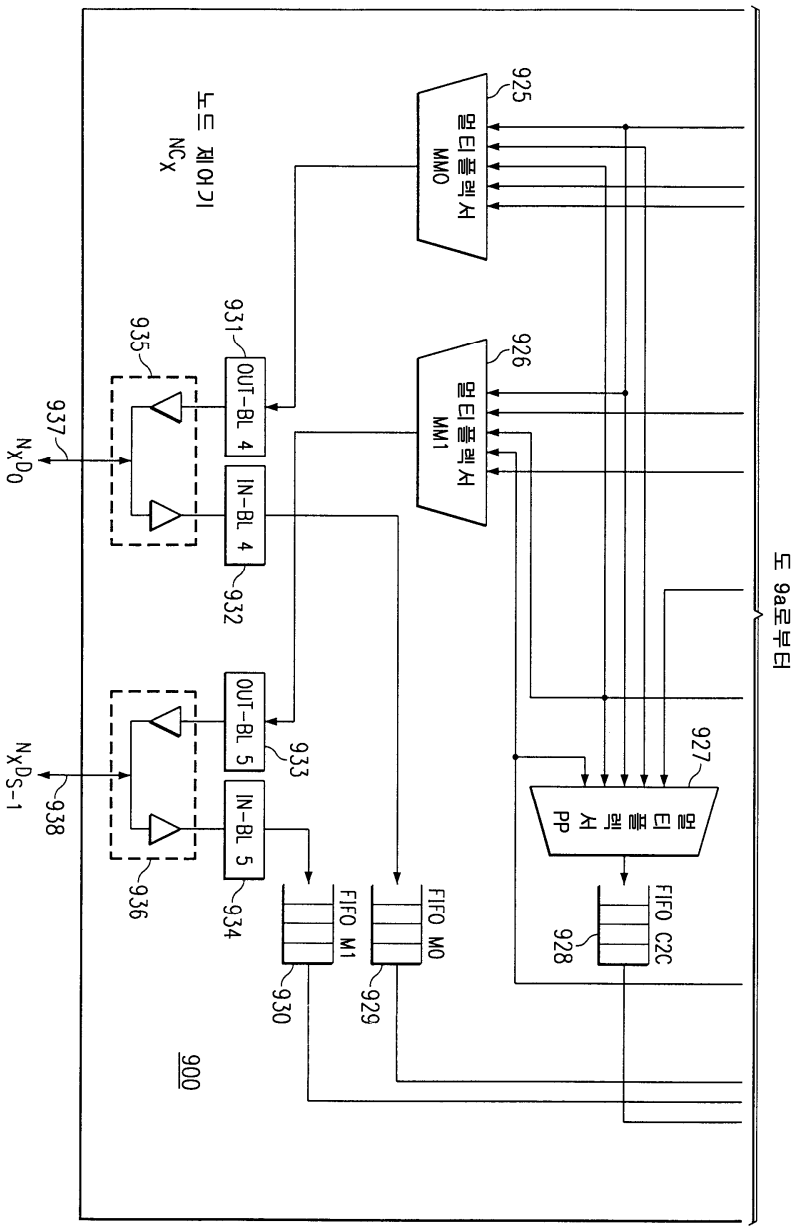
도면8


AIn$_X$
802
800
803
FD
메모리
서브시스템
MS$_X$
메모리 제어기
801

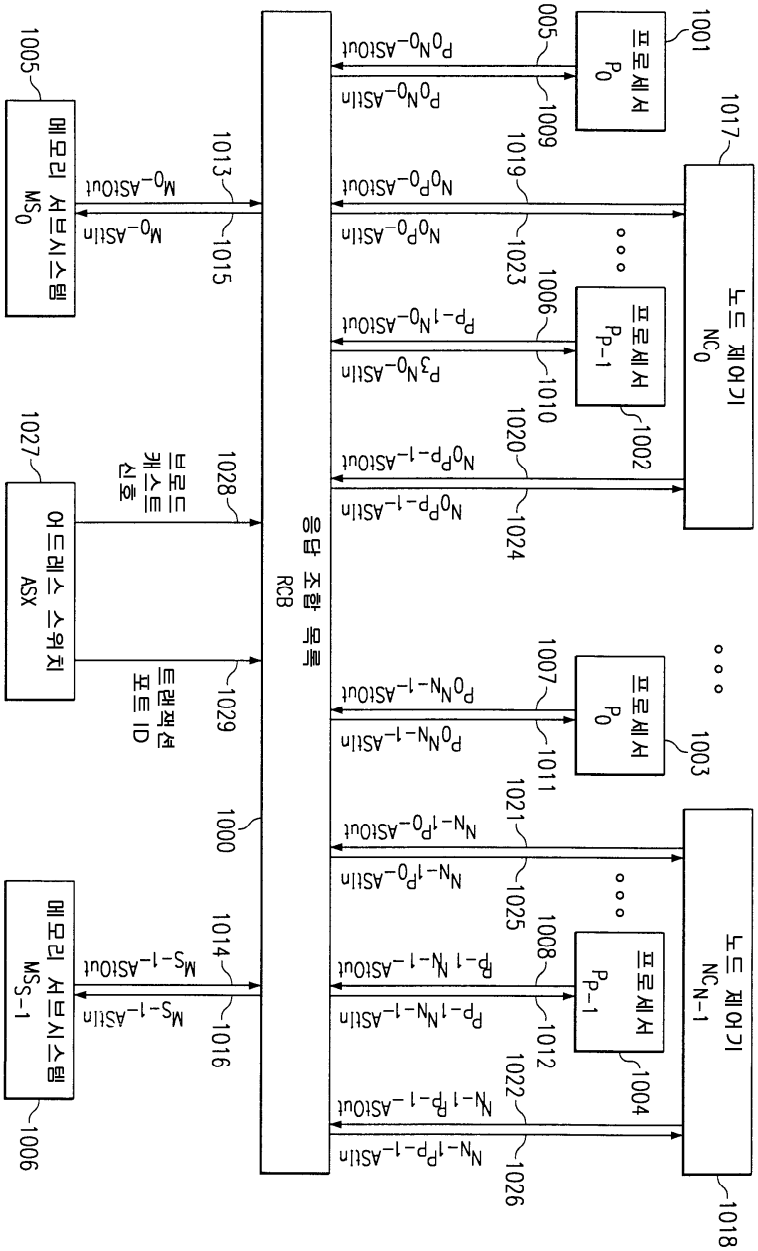
도면9a


도 9b로
901
P_0 버스
902
P_1 버스
903
P_2 버스
904
P_3 버스
905
906
907
908
909
IN-BL 0
910
IN-BL 1
911
IN-BL 2
912
IN-BL 3
913
OUT-BL 0
914
OUT-BL 1
915
OUT-BL 2
916
OUT-BL 3
917
FIFO 0
918
FIFO 1
919
FIFO 2
920
FIFO 3
921
PMUX 0
922
PMUX 1
923
PMUX 2
924
PMUX 3
939
제어 논리 유닛
940
다른 제어

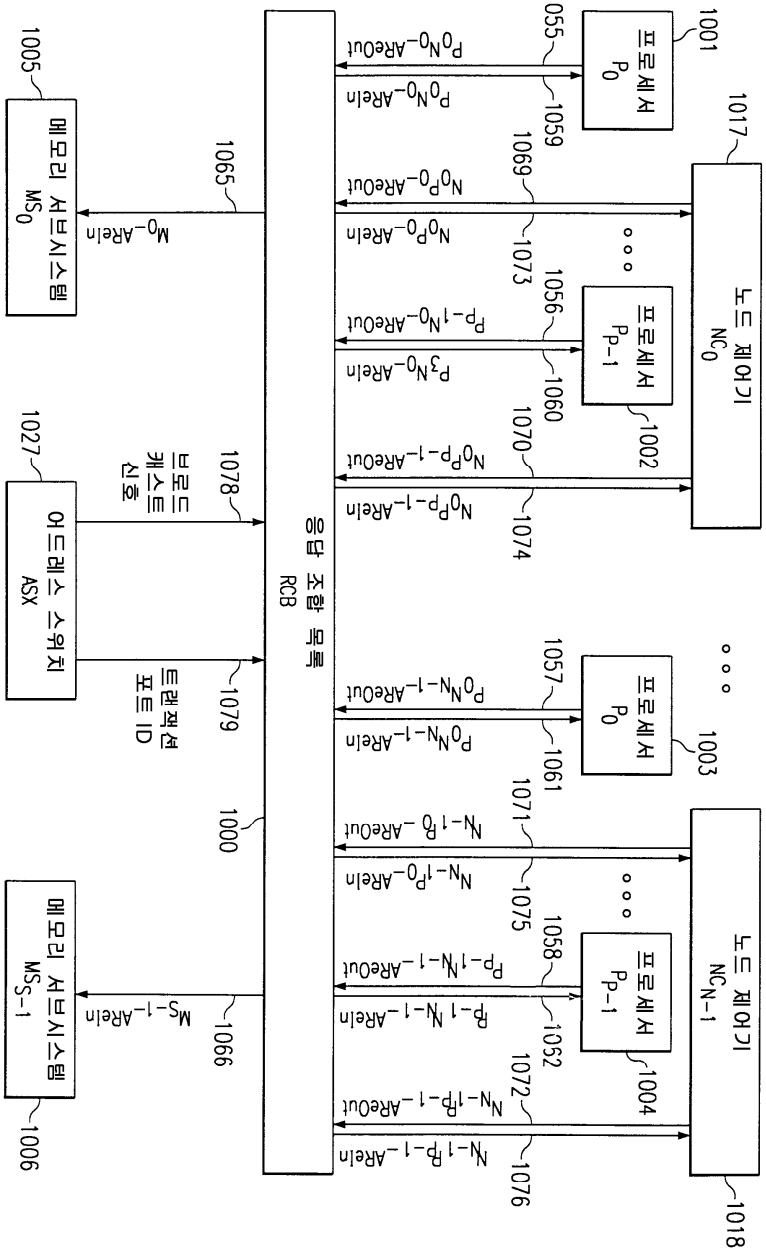
도면9b


도 9a로부터
900
925
멀티플렉서 MM0
926
멀티플렉서 MM1
927
멀티플렉서 PP
928
FIFO C2C
929
FIFO M0
930
FIFO M1
931
OUT-BL 4
932
IN-BL 4
933
OUT-BL 5
934
IN-BL 5
935
936
937
938
노드 제어기 NCx
NxD0
NxDS-1

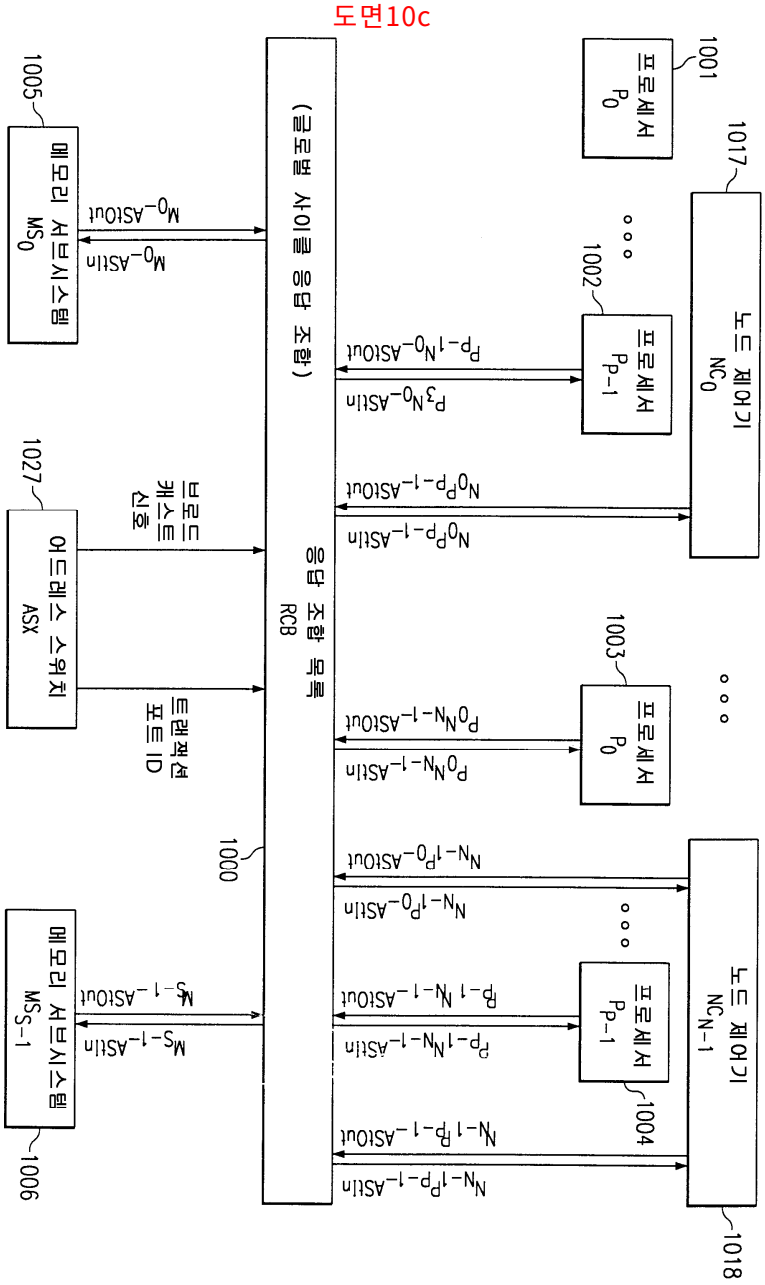
도면10a


프로세서 P_0
프로세서 P_{P-1}
노드 제어기 NC_0
프로세서 P_0
프로세서 P_{P-1}
노드 제어기 NC_{N-1}
응답 조합 목록 RCB
메모리 서브시스템 MS_0
메모리 서브시스템 MS_{S-1}
어드레스 스위치 ASX
브로드캐스트 신호
트랜잭션 포트 ID
P_0N_0-ASIIn
P_0N_0-ASIOut
N_0P_0-ASIIn
N_0P_0-ASIOut
$P_{P-1}N_0$-ASIIn
$P_{P-1}N_0$-ASIOut
N_0P_{P-1}-ASIIn
N_0P_{P-1}-ASIOut
P_0N_{N-1}-ASIIn
P_0N_{N-1}-ASIOut
$N_{N-1}P_0$-ASIIn
$N_{N-1}P_0$-ASIOut
$P_{P-1}N_{N-1}$-ASIIn
$P_{P-1}N_{N-1}$-ASIOut
$N_{N-1}P_{P-1}$-ASIIn
$N_{N-1}P_{P-1}$-ASIOut
M_0-ASIIn
M_0-ASIOut
M_{S-1}-ASIIn
M_{S-1}-ASIOut
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029

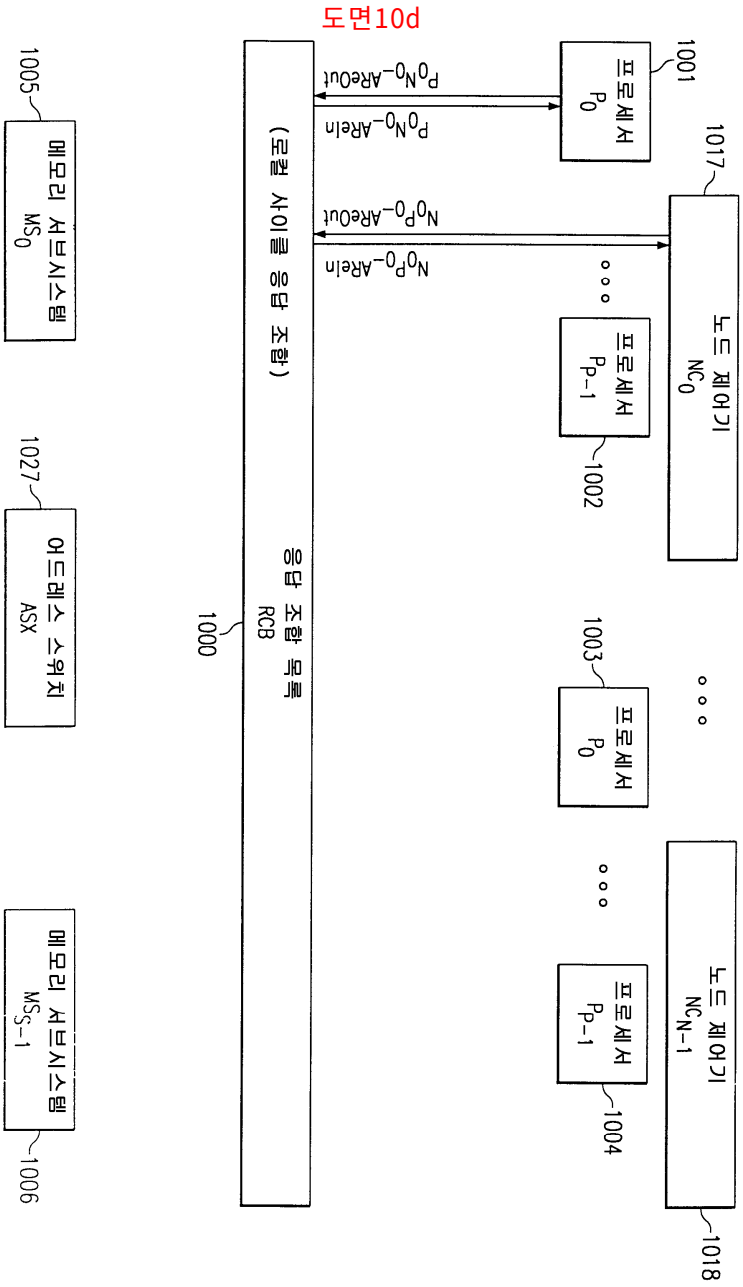
도면10b


1001
프로세서 P_0
1017
노드 제어기 NC_0
1055
P_0N_0-AReOut
1059
P_0N_0-AReIn
1069
N_0P_0-AReOut
1073
N_0P_0-AReIn
1056
$P_{P-1}N_0$-AReOut
1060
P_3N_0-AReIn
1002
프로세서 P_{P-1}
1070
N_0P_{P-1}-AReOut
1074
N_0P_{P-1}-AReIn
1057
P_0N_{N-1}-AReOut
1061
P_0N_{N-1}-AReIn
1003
프로세서 P_0
1071
$N_{N-1}P_0$-AReOut
1075
$N_{N-1}P_0$-AReIn
1058
$P_{P-1}N_{N-1}$-AReOut
1052
$P_{P-1}N_{N-1}$-AReIn
1004
프로세서 P_{P-1}
1072
$N_{N-1}P_{P-1}$-AReOut
1076
$N_{N-1}P_{P-1}$-AReIn
1018
노드 제어기 NC_{N-1}
1005
메모리 서브시스템 MS_0
1065
M_0-AReIn
1027
어드레스 스위치 ASX
1078
브로드캐스트 신호
1079
트랜잭션 포트 ID
1000
응답 조합 목록 RCB
1066
M_{S-1}-AReIn
1006
메모리 서브시스템 MS_{S-1}

도면10c


(글로벌 사이클 응답 조합)
응답 조합 목록
RCB
1000
프로세서
P_0
1001
노드 제어기
NC_0
1017
프로세서
P_{P-1}
1002
프로세서
P_0
1003
프로세서
P_{P-1}
1004
노드 제어기
NC_{N-1}
1018
메모리 서브시스템
MS_0
1005
메모리 서브시스템
MS_{S-1}
1006
어드레스 스위치
ASX
1027
브로드캐스트 신호
트랜잭션 포트 ID
M_0-AStIn
M_0-AStOut
$P_{P-1}N_0$-AStOut
P_3N_0-AStIn
N_0P_{P-1}-AStOut
N_0P_{P-1}-AStIn
P_0N_{N-1}-AStOut
P_0N_{N-1}-AStIn
$N_{N-1}P_0$-AStOut
$N_{N-1}P_0$-AStIn
$P_{P-1}N_{N-1}$-AStOut
$P_{P-1}N_{N-1}$-AStIn
$N_{N-1}P_{P-1}$-AStOut
$N_{N-1}P_{P-1}$-AStIn
M_{S-1}-AStOut
M_{S-1}-AStIn

도면10d


1005
메모리 서브시스템
MS_0
1027
어드레스 스위치
ASX
1006
메모리 서브시스템
MS_{S-1}
1000
응답 조합 목록
RCB
(로컬 사이클 응답 조합)
1001
프로세서
P_0
1002
프로세서
P_{P-1}
1017
노드 제어기
NC_0
1003
프로세서
P_0
1004
프로세서
P_{P-1}
1018
노드 제어기
NC_{N-1}
P_0N_0-ARelOut
P_0N_0-ARelIn
N_0P_0-ARelOut
N_0P_0-ARelIn

도면11

트랜잭션의 상태

상태	사이클 수	상태 시작	상태 종료	위치/동작
1a	1	시스템내 제 1 사이클	시스템내 제 1 사이클	경계 래치 IN_BL 에 래치
1b	트랜잭션의 형태에 의존	시스템내 제 2 사이클	트랜잭션에 대한 Primary ARespOut 의 마지막 사이클	노드 제어기내에 큐됨
2a	미정	상태 1b 만료 이후의 사이클	트랜잭션이 FROM_ASX_BL 에 수신되기 전의 사이클	노드 제어기내에 큐되고 글로벌 스누프 동안 선택
2b	1	트랜잭션이 FROM_ASX_BL 에 상주하는 동안의 사이클	트랜잭션이 FROM_ASX_BL 에 상주하는 동안의 사이클	글로벌 스누프 동안 경계 래치 FROM_ASX_BL 에 래치
3	일정 수의 사이클	상태 2b 만료 이후의 사이클	노드 제어기에 의해 검출된 트랜잭션의 Global ARespIn	글로벌 스누프 동안 마스터 디바이스에게 브로드캐스트
4	트랜잭션의 형태에 의존	상태 3 만료 이후의 사이클	요청자에게 데이터 전송이 시작하기 이전 Read 트랜잭션 사이클 --- 시스템에 대하여 트랜잭션의 Non-Read 트랜잭션 완료	트랜잭션의 실행
5	Read 트랜잭션 및 전송될 데이터량에 의존	상태 4 만료 이후의 사이클	시스템에 대하여 트랜잭션의 완료	트랜잭션의 실행

도면12a

DClaim 상태 :	1a	1b	2a	2b	3	4
DClaim (로컬)	Primary AStat=Retry Primary AResp=X	Primary AStat=X Primary AResp=Retry	Global AStat=X Global AResp=Retry	--- ---	Global AStat=X Global AResp=X	Global AStat=X Global AResp=X
Read (스누프됨)	Global AStat=X Global AResp=X	Global AStat=X Global AResp=X	Global AStat=X Global AResp=Retry	--- ---	Global AStat=X Global AResp=Retry	Global AStat=X Global AResp=Retry

도면12b

Read 상태 :	1a	1b	2a	2b	3	4
Read (로컬)	Primary AStat=Retry Primary AResp=X	Primary AStat=X Primary AResp=Retry	Global AStat=X Global AResp=X	--- ---	Global AStat=X Global AResp=X	Global AStat=X Global AResp=X
DClaim (스누프됨)	Global AStat=X Global AResp=X	Global AStat=X Global AResp=X	Global AStat=X Global AResp=Retry	--- ---	Global AStat=X Global AResp=Retry	Global AStat=X Global AResp=Retry