

시스템 버스 블록 설계서

System Bus Block Specifications

Ver.2.0

1994년 1월 20일

기안도

프로세서 연구실

시스템 연구부

한국전자통신연구소

Copyright ©1991•1992•1993•1994 ETRI

이 보고서의 내용을 임의로 전재 또는 복사할 수 없으며, 이 보고서의 내용을
이용 또는 전재할 경우 반드시 한국전자통신연구소의 서면 허락을 먼저 취득
하여야 한다.

한국전자통신연구소

작성자 기안도	승인자 임기욱	문서번호 BLK103-21-2.0	페이지 112						
문서 관리자 김외숙	종합관리자 박진원	년 월 일 1994.1.20.	변경 코드 D	화 일					
종합관리번호		주제분류 TD	등록일자 1994.1. .	FILE/TAPE 번호					
제출 번호	계정번호 3CK2210	구 분 — TD —	활동번호 —	기술보호 기준 1.공지사항 2.특허기술 3.Know-How 기술					
제 목	한글	시스템 버스블록설계서 버전2.0							
	영문	System Bus Block Specifications Ver.2.0							
작성 부서	프로세서연구실 (6120)	출연(위탁) 기관	체신부, 상공부, 과기처						
색 인 어	한글	시스템 버스, 하이파이 플러스 버스, 타이콤							
	영문	system bus, HiPi+Bus, TICOM-III							
요약 (300자이내)	<p>본 문서는 고속중형컴퓨터 (주전산기 III) 시스템 설계서에서 제시한 설계 방향과 목표를 기본으로 하여 하드웨어 서브시스템 중 시스템 버스 블록의 상세한 규격을 기술한 문서이다.</p> <p>시스템 버스 블록은 하드웨어 서브시스템을 구성하는 여러 블록들을 백플레인을 통하여 전기적, 기능적으로 연결시킨다. 그 이터 속도에는 16.5MHz 버스클럭을 사용하고, 다중캐쉬의 캐쉬동 일정 유지 프로토콜을 지원하고, 다중 프로세서 시스템의 인터럽트 전송을 지원한다.</p>								
공동 작성자	심원세, 한종석, 송용호								
검토자	윤석한 / 프로세서연구실								
관련 문서	<p>고속중형컴퓨터 (주전산기 III) 요구사항 정의서 (SYS000-20-2.2, '93.1.)</p> <p>고속중형컴퓨터 (주전산기 III) 시스템 설계서 (SYS000-21-2.3, '93.2.)</p> <p>고속중형컴퓨터 (주전산기 III) 하드웨어 서브시스템 설계서 (SUB100-21-1.1, '93.2.)</p>								
배포처	<p>시스템설계 및 통합 과제, 시스템분석 및 검증 과제, 시스템성능 과제</p> <p>하드웨어설계 과제, 운영체제 과제, 시스템소프트웨어개발 과제</p> <p>참여 4개 기업체 (금성, 대우, 삼성, 현대)</p>								

Contents

목차	v
그림목차	vii
표목차	x
변경이력	xi
1 HiPi+Bus	1
1.1 서론	1
1.1.1 개발 목적 및 내용	1
1.1.2 본 문서의 구성	1
1.1.3 관련문서	1
1.1.4 문서트리	2
1.1.5 참고자료	2
1.2 블록 설계 개념	4
1.2.1 다중 프로세서를 지원하기 위한 고려사항	4
1.2.2 고성능 버스를 위한 고려사항	5
1.3 블록 구조	7
1.3.1 관련 블록 간 구조	7
1.3.2 시스템의 구성	7
1.3.3 주요규격	7
1.3.4 블록의 구조	7
1.3.5 버스신호들	9
1.3.6 문서 작성의 규칙	9
1.3.6.1 신호 이름에 대한 규칙	9
1.3.6.2 신호의 상태에 대한 규칙	11
1.3.6.3 시간 규격 표시에 대한 규칙	12
1.3.6.4 신호의 값을 지정하는 일반적인 규칙	12
2 중재 버스	13
2.1 개요	13
2.2 중재 버스의 신호선	14

2.2.1	어드레스 중재 버스	14
2.2.1.1	Address Bus Request : ABRQ<12..0>*	14
2.2.1.2	Address Bus Arbitration Inhibit : ABINH*	15
2.2.1.3	Write Bus Cycle Inhibit : WRINH*	15
2.2.2	데이터 중재 버스	15
2.2.2.1	Data Bus Request : DBRQ<8..0>*	15
2.2.2.2	Data Bus Arbitration Inhibit : DBINH*	15
2.2.2.3	Priority Change Window : PCW*	16
2.3	중재 방법	17
2.3.1	선형 자가 중재 기법	17
2.3.2	공정성 규칙	18
2.4	어드레스 버스 중재	19
2.5	데이터 버스 중재	20
3	데이터 전송 버스	21
3.1	개요	21
3.2	데이터 전송 버스의 신호선	22
3.2.1	어드레스 버스의 신호선 (signal lines of address bus)	22
3.2.1.1	Address : A<31..4>*	23
3.2.1.2	Address Parity : AP<3..0>*	23
3.2.1.3	Source Identification : SI<7..0>*	23
3.2.1.4	Source Identification Parity : SIP*	23
3.2.1.5	Address Space : AS<2..0>*	23
3.2.1.6	Transfer Types : TT<4..0>*	24
3.2.1.7	Address Space and Transfer Types Parity : STP*	26
3.2.1.8	Byte Enable : BE<15..0>*	26
3.2.1.9	Byte Enable Parity : BEP<1..0>*	26
3.2.1.10	Address Bus Enable : AE*	27
3.2.2	데이터 버스의 신호선 (signal lines of data bus)	27
3.2.2.1	Data : D<127..0>*	27
3.2.2.2	Data Parity : DP<15..0>*	27
3.2.2.3	Destination Identification : DI<7..0>*	28
3.2.2.4	Destination Identification Parity : DIP*	28
3.2.2.5	Data Bus Enable : DE*	28
3.2.3	상태 버스의 신호선 (signal lines of status bus)	28
3.2.3.1	Address Acknowledge : AACK<1..0>*	28
3.2.3.2	Hit On Shared Line : SHD*	28
3.2.3.3	Hit On Dirty Line : DTY*	29
3.2.3.4	Intervention : ITV*	29
3.2.3.5	Snoop No Acknowledge : SNK*	30
3.2.3.6	Hit On Interlocked Region : LCR*	30
3.2.3.7	Spin Queue Order : SPIN<3..0>*	30
3.2.3.8	Data Acknowledge : DACK*	31

3.2.3.9	Cache Data Acknowledge : CDK*	31
3.2.3.10	Busy Status Line : BSY<7..0>*	31
3.3	데이터 전송	33
3.3.1	데이터 전송의 기본 사이클	33
3.3.1.1	어드레스 기본 사이클	33
3.3.1.2	단일 전송 데이터 기본 사이클	35
3.3.1.3	블록 전송 데이터 기본 사이클	37
3.3.1.4	단일 전송 어드레스-데이터 기본 사이클	40
3.3.1.5	블록 전송 어드레스-데이터 기본 사이클	41
3.3.2	전송의 구분	45
3.3.2.1	단일 읽기 전송 모임	45
3.3.2.2	단일 쓰기 전송 모임	45
3.3.2.3	블록 읽기 전송 모임	45
3.3.2.4	블록 쓰기 전송 모임	45
3.4	예외 처리 (Exception Handling)	50
3.4.1	패리티 에러 (Parity Error)	50
3.4.1.1	어드레스 기본 주기에서 패리티 에러가 검출될 경우	50
3.4.1.2	데이터 기본 주기에서 패리티 에러가 검출될 경우	50
3.4.1.3	어드레스 데이터 주기에서 패리티 에러가 검출될 경우	50
3.4.2	Timeout Error	51
3.4.3	Busy 응답	51
3.4.4	기타 에러	51
4	인터럽트 버스	53
4.1	개요	53
4.2	인터럽트 버스의 신호선	54
4.2.1	Interrupt Bus Sync : IBSYNC*	54
4.2.2	Interrupt Bus Data : IBD<7..0>*	54
4.2.3	Interrupt Bus Data Parity : IBDP*	55
4.3	인터럽트의 종류	56
4.3.1	지정 인터럽트	56
4.3.2	중재 인터럽트 1	56
4.3.3	중재 인터럽트 0	57
4.4	벡터의 종류	58
4.4.1	E(Emergency)-Type	58
4.4.2	I(Immediate)-Type	58
4.4.3	N(Notify)-Type	58
4.4.4	Q(Queue)-Type	59
4.5	인터럽트 버스 중재	60
4.5.1	중재 정보	60
4.5.2	부호화된 자가 중재 기법	60
4.6	지정 인터럽트	62
4.7	중재 인터럽트 1	66

4.8 중재 인터럽트 0	73
4.9 예외 처리 (Exception Handling)	74
4.9.1 전송 에러 (Transmission Error)	74
5 유ти리티 버스	75
5.1 개요	75
5.2 유ти리티 버스의 신호선	76
6 시간 규격	79
6.1 개요	79
6.2 버스클럭의 시간규격	80
6.3 중재버스의 시간규격	81
6.3.1 ABRQ<n>*, DBRQ<n>*의 시간규격	81
6.3.2 ABINH*, WRINH*, DBINH*의 시간규격	81
6.4 데이터 전송버스의 시간규격	82
6.5 인터럽트버스의 시간규격	83
6.5.1 IBSYNC* 신호의 시간규격	83
6.5.2 인터럽트 버스 중재 동작의 시간 규격	84
6.5.3 인터럽트 버스 인터럽트 데이터 전송 주기의 시간 규격	84
7 전기적 규격	87
7.1 개요	87
7.2 전원규격	89
7.3 콘넥터의 전기적 규격	89
7.4 BTL 신호의 전기적 규격	90
7.4.1 인터페이스 회로의 용량성 부하	90
7.4.2 인터페이스 회로의 누설 전류	91
7.4.3 구동소자	91
7.4.3.1 구동소자 형태	91
7.4.3.2 정적 부하 전류 (static load current)	91
7.4.3.3 구동소자의 V_{OH}	91
7.4.3.4 구동소자의 V_{OL}	91
7.4.3.5 천이시간 (transition times)	91
7.4.4 수신소자	92
7.4.4.1 임계영역 (threshold)	92
7.4.4.2 잡음제거 (noise rejection)	92
7.4.4.3 입력 클램프 (input clamps)	92
7.5 TTL 신호의 전기적 규격	93
7.5.1 인터페이스 회로의 용량성 부하	93
7.5.2 구동소자의 종류	93
7.6 백플레인의 특성	94
7.6.1 백플레인의 특성임피던스	94
7.6.2 터미네이션	94

8 기계적 규격	95
8.1 개요	95
8.2 백플레인	96
8.3 주보드 (Main Board)	96
8.4 입출력보드 (I/O Board)	96
8.5 파워 바 (Power Bar)와 파워 탭 (Power Tab)	97
8.6 주서브랙 (Main Subrack)	97
8.7 입출력서브랙 (I/O Subrack)	97
8.8 콘넥터	98
A HiPi+Bus 신호선과 핀할당	99
A.1 HiPi+Bus 신호선들	100
A.2 HiPi+Bus 핀할당	101
B 약어 일람표	105

ETRI-CSITL

List of Figures

1.1	문서트리	2
1.2	하드웨어 서브시스템의 블록 간 상호관계	7
1.3	시스템 구성	9
1.4	버스 블록의 구조	11
2.1	선형 자가 중재기의 일례	17
2.2	공정성 중재 규칙	18
3.1	어드레스 기본 사이클	34
3.2	데이터 기본 사이클	36
3.3	블록 전송 데이터 기본 사이클	38
3.4	어드레스 데이터 기본 사이클	40
3.5	블록 전송 어드레스 데이터 기본 사이클	42
3.6	단일 읽기 전송	46
3.7	단일 쓰기 전송	47
3.8	블록 읽기 전송	48
3.9	블록 쓰기 전송	49
4.1	부호화된 자가 중재기의 일례	61
4.2	지정인터럽트의 버스 동작	62
4.3	중재인터럽트1의 버스 동작	68
6.1	버스클럭의 시간규격	80
6.2	중재버스의 시간규격	82
6.3	데이터 전송버스 신호의 시간규격	83
6.4	IBSYNC* 신호의 시간규격	84
6.5	인터럽트 버스 중재 동작의 시간규격	85
6.6	인터럽트 버스의 데이터 전송의 시간규격	86
7.1	BTL 신호선의 전기적 규격	90
7.2	TTL 신호선의 전기적 규격	93
7.3	터미네이션 방법	94
8.1	기계적 장치의 개요	95

ETRI-CSTL

List of Tables

1.1	HiPi+Bus의 규격	8
1.2	신호선들	10
2.1	중재 버스의 규격	13
2.2	중재 버스의 신호들	14
2.3	어드레스 중재 버스의 중재 번호와 슬롯 어드레스	14
2.4	데이터 중재 버스의 중재 번호와 슬롯 어드레스	16
3.1	데이터 전송 버스의 규격 요약	21
3.2	데이터 전송 버스의 신호선 요약	22
3.3	어드레스와 패리티 비트	23
3.4	시발점 식별자	23
3.5	정의된 어드레스 영역	24
3.6	전송 형태	24
3.7	바이트 지정자	26
3.8	바이트 지정자와 패리티 비트	27
3.9	데이터와 데이터 패리티	27
3.10	종착점 식별자	28
3.11	어드레스 응답	29
3.12	SHD*	29
3.13	DTY*	29
3.14	ITV*	30
3.15	SNK*	30
3.16	LCR*	31
3.17	데이터 응답	31
3.18	CDK*	31
4.1	인터럽트 버스의 규격 요약	53
4.2	인터럽트 버스의 신호들	54
4.3	인터럽트 종류들	56
4.4	벡터의 종류	58
4.5	인터럽트 버스 중재에 사용되는 중재 정보	60
4.6	인터럽트 중재 주기에 사용되는 구별자 배정	61

4.7	지정 인터럽트의 단계별 전송 정보	62
4.8	지정 인터럽트 - 단계 0에서 전송되는 정보	63
4.9	지정 인터럽트 단계 0의 처리기 어드레스	63
4.10	지정 인터럽트 - 단계 1에서 전송되는 정보	64
4.11	지정 인터럽트 - 단계 3에서 전송되는 정보의 비트별 의미	65
4.12	중재 인터럽트 1의 단계별 전송 정보	67
4.13	중재 인터럽트 1 - 단계 0에서 전송되는 정보	67
4.14	중재 인터럽트 1의 벡터 형태	69
4.15	중재 인터럽트 1의 모임 주소 각 비트별 의미	69
4.16	중재 인터럽트 1 - 단계 2에서 사용되는 정보	70
4.17	중재 인터럽트 1 - 단계 7에서 사용되는 정보	71
4.18	중재 인터럽트 1 - 단계 12에서 사용되는 정보	71
4.19	중재 인터럽트 1 - 단계 17에서 전송되는 정보	71
4.20	중재 인터럽트 1 - 단계 18에서 전송되는 정보	72
4.21	중재 인터럽트 1 - 단계 19에서 전송되는 정보	72
5.1	유틸리티 버스의 규격 요약	75
5.2	유틸리티 버스의 신호들	76
5.3	슬롯 어드레스와 슬롯 위치	76
5.4	경계주사	77
6.1	버스클럭의 시간규격 요약	80
6.2	중재버스 신호의 시간규격 요약	81
6.3	데이터 전송 버스 신호의 시간규격 요약	83
6.4	IBSYNC* 신호의 시간규격 요약	84
6.5	인터럽트 중재 버스 신호의 시간규격 요약	85
6.6	인터럽트 버스의 인터럽트 데이터 신호의 시간규격 요약	85
7.1	전기적 규격 요약	88
7.2	전원규격 요약	89
7.3	핀의 전기적 규격	89
7.4	BTL 신호의 전기적 규격	90
7.5	TTL 신호의 전기적 규격	93
8.1	백플레인의 기계적 규격	97

변경이력

1. 1993년 2월 12일 : Ver.1.0 작성완료
2. 1993년 2월 22일 : BLK103-21-1.0 문서등록
3. 1993년 8월 16일 : Ver.1.1 작성완료
 - 오자교정
 - BCLK<9..0>*를 BCLK*로 변경
 - DI<7..0>*의 의미를 확장함 (표3.10 참고)
 - WRB 전송형태의 의미를 확장함 (제3.2.1장 Transfer Type 설명 참고)
 - 제7장 전기적 규격에 TTL 관련 규격을 추가함
 - 제8장 기계적 규격의 내용을 변경함
 - 부록을 추가함
4. 1993년 8월 23일 : BLK103-21-1.1 문서등록
5. 1993년 12월 16일 : Ver.2.0 작성완료
 - 오자교정
 - 제3장 데이터전송버스의 내용중 BE<15..0>* 부분 변경
 - 제8장 기계적 규격의 내용중 백플레인 크기와 주보드 크기를 정정함

Chapter 1

HiPi+Bus

1.1 서론

1.1.1 개발 목적 및 내용

공유메모리를 사용하는 다중프로세서 구조를 갖는 시스템에서 프로세서와 메모리 사이의 정보 통로로 사용할 공유버스를 설계하는 것이 본 문서의 목적이다. HiPi+Bus¹는 시스템 버스이며, 이러한 시스템 버스를 설계하는 목표는 다음과 같다.

- 공유메모리 다중프로세서 구조를 지원
공유메모리를 사용하는 다중프로세서 구조에서 프로세서와 메모리를 버스를 이용하여 연결시킬 때 시스템 버스는 데이터 전송 프로토콜, 전송속도, 중재방법, 그리고 인터럽트 등 버스의 각 부분에서 다중프로세서를 지원하기 위한 고려가 필요하다.
- 고성능 컴퓨터 지원
목표로 하는 다중프로세서의 성능이 고성능이기 위해서는 시스템 버스도 이에 어울리는 성능과 기능을 제공해야 하며 이를 위해 데이터 전송 능력, 데이터 전송의 신뢰도, 시스템 구성 능력, 시스템 유지 보수를 위한 기능 등이 고려되어야 한다.

1.1.2 본 문서의 구성

본 문서의 구성은 제 1 장에서 전체적인 구성과 설계 개념을 기술하고, 제 2 장에는 중재버스, 제 3 장에는 데이터 전송 버스, 제 4 장에는 인터럽트 버스, 제 5 장에는 유ти리티 버스, 제 6 장에는 시간 규격, 제 7 장에는 전기적 규격, 제 8 장에는 기계적 규격을 담고 있다.

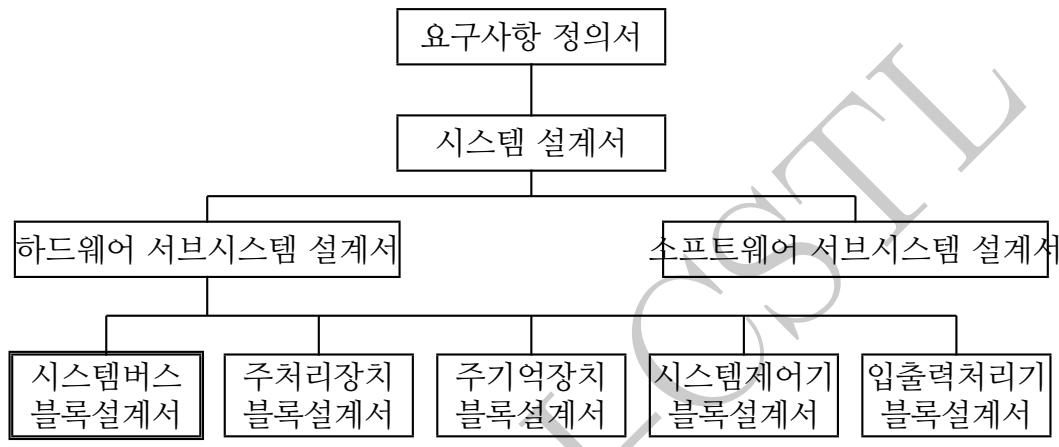
1.1.3 관련문서

- (1) 고속중형컴퓨터 (주전산기 III) 요구사항 정의서, SYS000-20-3.0, 컴퓨터시스템연구실, 시스템공학연구부, 한국전자통신연구소, 1993.

¹Highly Pipelined Plus Bus : HiPi+Bus: /hai-pai-pl'ʌs-bʌs/

- (2) 고속중형컴퓨터 (주전산기 III) 시스템 설계서, SYS000-21-3.0, 컴퓨터시스템연구실, 시스템공학연구부, 한국전자통신연구소, 1993.
- (3) 고속중형컴퓨터 (주전산기 III) 하드웨어 서브시스템 설계서, SUS100-21-2.0, 컴퓨터시스템연구실, 시스템공학연구부, 한국전자통신연구소, 1994.1.

1.1.4 문서트리



1.1.5 참고자료

- (1) 컴퓨터구조연구실, *TICOM 시스템 버스 사용자 지침서 (TD88-6121-133.B)*, 시스템구조연구부, 한국전자통신연구소, 1990.11.
- (2) 컴퓨터구조연구실, *시스템 버스 표준 인터페이스 설계서 Ver.2.0 (TM93-CA-017.B)*, 시스템구조연구부, 한국전자통신연구소, 1993.7.
- (3) 컴퓨터구조연구실, *TTL 회로 설계의 고려사항 (TM92-CA-039)*, 시스템구조연구부, 한국전자통신연구소, 1992.6.26.
- (4) 컴퓨터구조연구실, *HiPi+Bus의 임계시간 (TM92-CA-053)*, 시스템구조연구부, 한국전자통신연구소, 1992.11.16.
- (5) 컴퓨터구조연구실, *BTL 기초조사 (TM93-CA-001.B)*, 시스템구조연구부, 한국전자통신연구소, 1993.2.
- (6) IEEE, *IEEE Standard Test Access Port and Boundary-Scan Architecture*, IEEE std.1149.1, 1990.

- (7) IEEE, *IEEE Standard for Electrical Characteristics of Backplane Transceiver Logic (BTL) Interface Circuits*, IEEE std.1194.1, 1991.
- (8) 컴퓨터구조연구실, *Semaphore Cache (Draft)*, 시스템구조연구부, 한국전자통신연구소, 1993.
- (9) 컴퓨터구조연구실, 인터럽트 관련 모듈 설계서 (*Draft*), 시스템구조연구부, 한국전자통신연구소, 1993.
- (10) 컴퓨터구조연구실, 고속중형컴퓨터 클럭 서브 시스템 (*TM93-CA-020*), 시스템구조연구부, 한국전자통신연구소, 1993.4.

1.2 블록 설계 개념

1.2.1 다중 프로세서를 지원하기 위한 고려사항

- Pended Protocols

Pended 프로토콜은 하나의 데이터 전송을 요청 단계(request phase)와 응답 단계(respond phase)로 나눔으로써 메모리 접근 시간이 버스의 전송 속도(bus bandwidth)에 영향을 주지 않도록 하는 방법이다. 메모리를 접근하는 동안 하나의 프로세서가 계속해서 버스를 점유하는 것이 아니므로, 다수의 데이터 전송 요청이 동시에 발생할 수 있는 다중 프로세서 환경에서는 이 프로토콜이 절대적으로 유리하다.

- Synchronous - 16.5 MHz

Pended 프로토콜은 데이터 전송을 요청과 응답의 두 단계로 나누어 pipeline시킨 것이다. 따라서 이와 같은 프로토콜을 구현하기에는 비동기형 제어 방식보다는 동기형 제어방식이 유리하다. 동기형 제어 방식은 비동기형에 방식에 비해 기술의 발전에 따른 탄력적 성능 향상을 얻기 어렵지만, 신뢰도가 높고 현재의 기술로 비동기형보다 나은 성능을 발휘할 수 있도록 구현할 수 있다는 장점을 가지고 있다. 클럭 주기는 단위 버스 동작이 수행되는 시간을 말하는 것으로 60.6 nsec이다.

- Fairness arbitration

다중 프로세서 시스템의 가장 큰 장점 중의 하나는 여러개의 프로세서가 일을 분담하여 수행함으로써 얻어지는 빠른 응답이라 할 수 있다. 이와 같은 장점을 충분히 발휘하도록 하기 위해서는 모든 프로세서가 동등한 자격을 갖고 매 순간마다 일의 균형있는 분배 (dynamic load balancing)가 이루어져야 한다. 균형 분배는 운영 체제에서 많은 책임을 갖고 있지만, 우선 하드웨어 수준에서의 동등한 자원 공유가 보장되어야 가능하다. 버스는 가장 핵심이 되는 자원이므로 버스 사용의 공정한 중재가 균형 분배의 시발점이라 할 수 있다. HiPi+Bus에는 모든 버스 모듈이 공정하게 버스를 사용할 수 있도록 중재 규칙을 정의하였다.

- Cache Coherency Protocols

버스를 사용하는 모든 다중 프로세서 시스템의 가장 큰 문제는 버스의 포화에 의한 시스템 성능 저하이다. 버스가 포화되면 프로세서 수가 증가하여도 시스템의 성능은 향상되지 않고 오히려 감소되기 때문이다. 이와 같은 문제를 해결하기 위한 방법으로 공유 메모리를 사용하는 다중 프로세서 시스템에서는 각 프로세서가 캐시를 갖게 하여 캐시에 원하는 데이터가 존재하지 않을 경우에 한해서 버스를 사용하게 함으로써 버스의 포화를 방지한다. 그러나 프로세서마다 독립적으로 캐시를 사용하게 되면 캐시상에 존재하는 데이터가 메모리나 다른 캐시의 데이터와 일관성이 깨어지는 문제 (multi-cache consistency problem)가 발생할 수 있으므로 이 문제를 해결하여야 한다. HiPi+Bus에서는 write-back 캐시의 동일성 유지 프로토콜을 지원한다.

- Interlock Protocols

다중처리에서 필수적인 동기화를 지원하기 위한 잠금 프로토콜을 지원한다.

- Interrupt Broadcast and Arbitration

공유 메모리를 사용하는 다중 처리 시스템 환경에 있어서의 모든 프로세서들은 동등한

자격을 갖고 동작한다. 따라서 인터럽트의 요구와 처리에 있어서 단일 프로세서 시스템과는 다른 기능들이 필요하게 된다. 즉, 여러 프로세서가 모두 인터럽트를 처리할 수 있기 때문에 여러 프로세서로 인터럽트를 보내는 방법(broadcast)과 실질적으로 인터럽트를 처리할 프로세서를 선정하는 기능(arbitration)이 필요하다. HiPi+Bus는 이와 같은 인터럽트의 기능들을 제공하고 있으며, 인터럽트를 모든 프로세서들 사이의 비동기적인 통신 수단으로 이용할 수 있도록 일반화시켰다.

1.2.2 고성능 버스를 위한 고려사항

- Bus bandwidth - 264 Mbytes/sec
HiPi+Bus는 264 Mbytes/sec의 지속적인 전송 용량(sustained bus bandwidth)을 갖는다.
- 128 bit data + 32 bit address, NonMultiplexed
HiPi+Bus는 128 비트의 데이터 전송을 위한 신호와 32 비트 어드레스 전송을 위한 신호를 별도로 갖고 있다. 따라서 어드레스 전송을 위해 데이터의 전송선을 사용하지 않기 때문에 매 버스 클럭 주기(60.6 nsec)마다 128 비트의 데이터가 전송될 수 있어서 264 Mbytes/sec의 데이터 전송 속도를 얻을 수 있다. 32 비트의 어드레스 전송선을 갖기 때문에 최대 4 Gbytes의 연속적인 어드레스 영역을 지원한다.
- Block Transfer
캐쉬 라인 크기와 같은 크기인 64바이트 전송을 지원한다.
- All bytes are parity protected
시스템의 신뢰도 향상을 위하여 버스에서는 모든 신호에 에러 방어를 위한 신호를 추가한다. 에러 방어의 대상은 순간적인 에러(transient or soft error)이며 재시도(retry)에 의해 처리한다. 고장으로 인한 계속적인 에러에 대해서는 처리 방법을 버스 규격에서 별도로 정의하지 않는다. 기본적으로 버스의 모든 신호는 바이트 단위의 패리티 비트를 갖는다. 단, 신호의 특성상 바이트 단위로 동시에 구동되지 않는 신호의 경우는 예외로 한다.
- 21 slots
하나의 보드는 하나의 슬롯을 이용하여 버스상에 장착된다. 따라서 슬롯의 수는 버스에 장착될 수 있는 최대 보드의 수를 말하고 이것은 시스템의構성을 제한한다. 21 개의 슬롯은 13 개의 프로세서 보드 (데이터 처리 프로세서와 입출력 프로세서)와 8 개의 메모리 보드가 장착되도록 한다.
- No mechanical switch and jumper
이전에는 버스나 버스 모듈에 다소의 유연성을 제공하기 위하여 기계적인 스위치나 점퍼를 많이 사용되었다. 그러나 기계적 부품은 다른 부품에 비하여 신뢰도가 많이 떨어지기 때문에 많은 시스템 고장의 원인이 되어왔다. 이와 같은 이유로 HiPi+Bus에서는 기계적인 부품을 없애고 소프트웨어 제어가 가능한 레지스터를 사용하여 스위치를 대신한다.

- BTL (Backplane Transceiver Logic)
백플레인 구동에는 BTL 기술을 사용한다.
- I/O connector
백플레인에 입출력 버스와의 연결을 용이하게 하기 위한 콘넥터를 둔다.

1.3 블록 구조

1.3.1 관련 블록 간 구조

시스템 버스 블록(BUS block)은 <그림 1.2>에서 굵은 선으로 표시된 부분에 해당한다. 즉

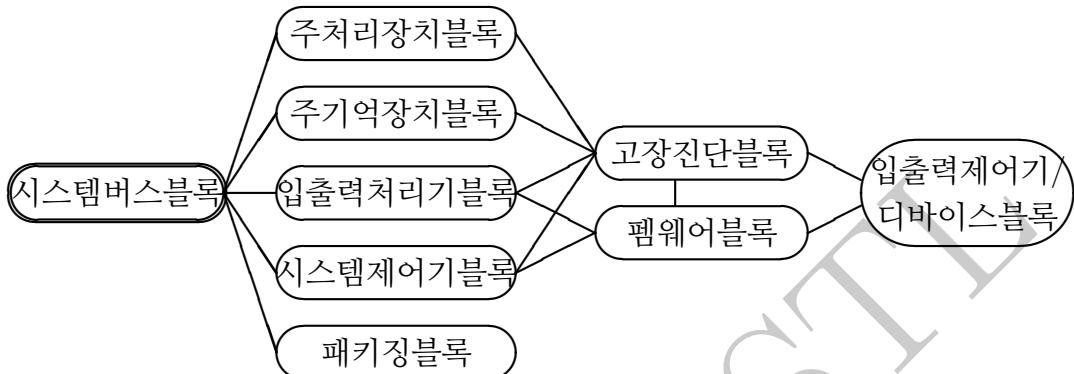


Figure 1.2: 하드웨어 서브시스템의 블록 간 상호관계

하드웨어 서브시스템에서 주 처리 장치 블록(MPU block), 주 기억 장치 블록(MEM block), 입출력 처리기 블록(IOP block), 시스템 제어기 블록(SCM block), 그리고 패키징 블록(PAC block)이 전기적, 기계적, 기능적으로 백플레인에 의해 연결될 때 이 백플레인을 포함한 전기적, 기계적, 기능적 집합체를 시스템 버스 블록이라 한다.

1.3.2 시스템의 구성

공유메모리를 갖는 다중 프로세서 시스템의 시스템 버스인 HiPi+Bus는 프로세스, 메모리, 기타 버스 상에 장착되는 보드들 사이의 정보 교환 통로로써 이용된다.

1.3.3 주요규격

시스템 버스 블록(BUS block)의 주요규격은 <표 1.1>와 같다.

1.3.4 블록의 구조

시스템 버스 블록(BUS block)은 <그림 1.4>와 같이 크게 중재버스(arbitration bus), 데이터 전송버스(dtat transfer bus), 인터럽트 전송버스(interrupt transfer bus), 그리고 유ти리티 버스/utility bus)로 구분된다. 그리고 중재버스는 어드레스 버스 중재버스(address bus arbitration bus)와 데이터 버스 중재버스(data bus arbitration bus)로 구성되고, 데이터 전송버스는 어드레스 버스(address bus), 데이터 버스(data bus), 그리고 상태 버스(status bus)로 구성된다.

Table 1.1: HiPi+Bus의 규격

중재 특성 (Characteristics of Arbitration)	
중재 규칙	우선순위와 공정성 (Priority and Fairness)
중재 수행 시간	1 Bus Clock (60.6 nsec)
중재 기법	선형자가중재 (Linear Self Arbitration)
중재기의 특성	분산형 중재기 (Distributed Arbiter)
기타	Request Inhibition, Request Timeout 구현
데이터 전송 특성 (Characteristics of Data Transfer)	
프로토콜	펜디드 (Pended)
제어 방식	동기형 (Synchronous)
클럭 속도	16.5 MHz (60.6 nsec)
어드레스/데이터	non-multiplexed
최대 어드레스 영역의 크기	4 Gbytes (32 bits wide)
어드레스 영역의 갯수	8 (3개 정의)
최대 사용 가능한 전송 형태	32 (13개 정의)
버스의 폭 (Bus Width)	128 bits (16 bytes)
데이터의 단위 (Data Unit)	8 bits (1 byte)
전송 가능한 데이터의 크기	16-바이트 이내의 연속된 바이트; 64-바이트
정렬의 제약 (Alignment Restriction)	16 byte boundary; 64 byte boundary
Justification	Straight (Nonjustified)
데이터 전송 속도(Data Transfer Rate)	264 Mbytes/sec (16Bytes × 16.5MHz)
인터럽트 전송 특성 (Characteristics for Interrupt Transfer)	
전송 프로토콜	Message passing protocols
제어 방식	동기형 (Synchronous)
클럭 속도	16.5 MHz (60.6 nsec)
전송 속도	최대 약 1.6 MI/sec (Mega Interrupts per second)
중재 기법	부호화된 자가중재 (Coded Self Arbitration)
중재기 특성	분산형 중재기 (Distributed Arbiter)
에러 방어	Parity Detection
기타	Broadcast and Arbitration
기 타 (Etc.)	
캐시 지원	Write Back Cache Coherency Protocol
동기화 지원 (Synchronization)	Semaphore Cache Protocol
에러 검출 (Error Detection)	바이트 단위의 홀수 패리티(odd parity)
에러 처리 (Error Handling)	재시도 (Retry)
경계조사 (Boundary Scan)	IEEE std.1149.1
구현 기술 (Technology)	BTL (IEEE std.1194.1)
슬롯 수	21 slots/backplane
콘넥터 규격	85×4rows (340pins, 전원제외) 40×4rows (160pins, 전원포함)
공급 전원	+5V, +5V Standby
총 신호수	293 (전원제외, 입출력용 제외)

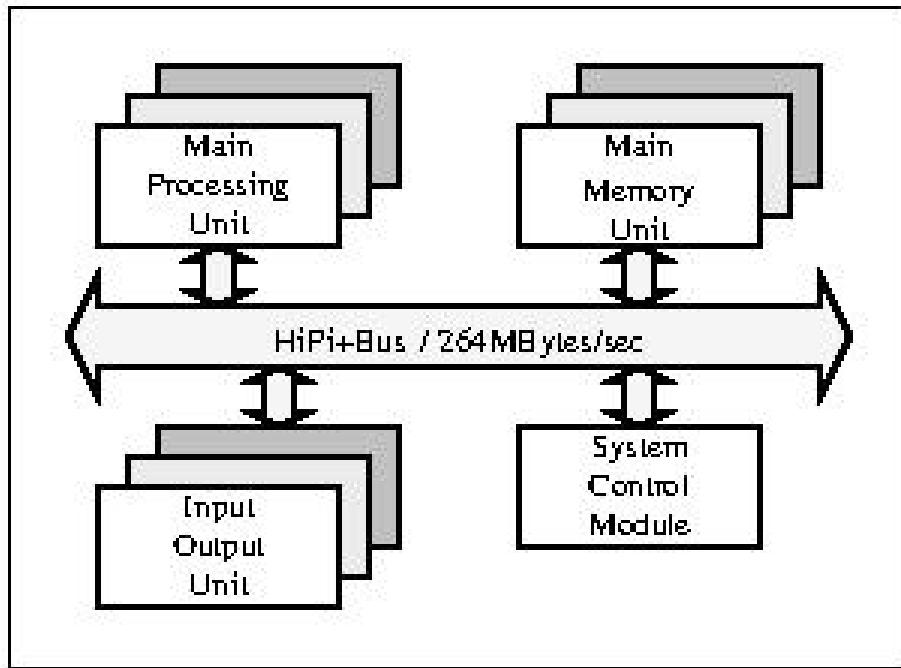


Figure 1.3: 시스템 구성

1.3.5 버스신호들

시스템 버스에서 정의하여 사용하는 버스 신호들은 <표 1.2>와 같다.

1.3.6 문서 작성의 규칙

1.3.6.1 신호 이름에 대한 규칙

- 버스 신호의 Mnemonics은 모두 대문자를 사용한다.
예: AE*, DE*, A<31..4>*
- 버스 신호의 Mnemonics은 신호 이름 중에서 신호 이름을 대표할 수 있는 앞뒤 몇 문자를 사용한다.
예: Bus Clock : BCLK*; Shared Line : SHD*
- 버스 상의 모든 신호의 끝에는 모두 *를 붙힌다. 이는 버스 신호의 전압이 낮은 상태일 때 “참 (true)”이 됨을 의미한다.
예: A<0>*, DTY*
- 동일한 이름을 갖는 여러 비트의 신호는 <>을 사용하여 번호와 크기를 나타낸다.
예: A<31..4>*, A<4>*, D<127..0>*, TT<3>*
- 번호는 큰 쪽을 먼저 쓰는 것을 기본(default)으로 한다.
예: A<31..4>*, D<127..0>*, TT<3..0>*

Table 1.2: 신호선들

Bus	Mnemonic	Size	Name
중재버스	ABRQ<12..0>* ABINH* WRINH* DBRQ<8..0>* DBINH* PCW*	13 1 1 9 1 1	Address Bus Request Address Bus Arbitration Inhibition Write Cycle Inhibition Data Bus Request Data Bus Arbitration Inhibition Priority Change Window
데이터전송버스 (어드레스버스)	A<31..4>* AP<3..0>* SI<7..0>* SIP* AS<2..0>* TT<4..0>* STP* BE<15..0>* BEP<1..0>* AE*	28 4 8 1 3 5 1 16 2 1	Address Address Parity Source Identification Source Identification Parity Address Space Transfer Types Space + Types Parity Byte Enable Byte Enable Parity Address Cycle Enable
데이터전송버스 (데이터버스)	D<127..0>* DP<15..0>* DI<7..0>* DIP* DE*	128 16 8 1 1	Data Data Parity Destination Identification Destination Identification Parity Data Cycle Enable
데이터전송버스 (상태버스)	AACK<1..0>* SHD* DTY* SNK* ITV* LCR* DACK* CDK* SPIN<3..0>* BSY<7..0>*	2 1 1 1 1 1 1 1 4 8	Address Acknowledge Hit on Shared Line Hit on Dirty Line Snoop No Acknowledge Intervention Hit on Interlocked Region Data Acknowledge Cache Data Acknowledge Spin Queue Order Busy Status Line
인터럽트 전송버스	IBSYNC* IBD<7..0>* IBDP*	1 8 1	Interrupt Bus Sync. Interrupt Bus Data Interrupt Bus Data Parity
유틸리티버스	BCLK* RST* SFAIL* GA<4..0>* Tx<4..0>	1 1 1 5 5	Bus Clock System Reset System Fail Geographical Slot Address JTAG Boundary Scan Option
	<i>total</i>	293	

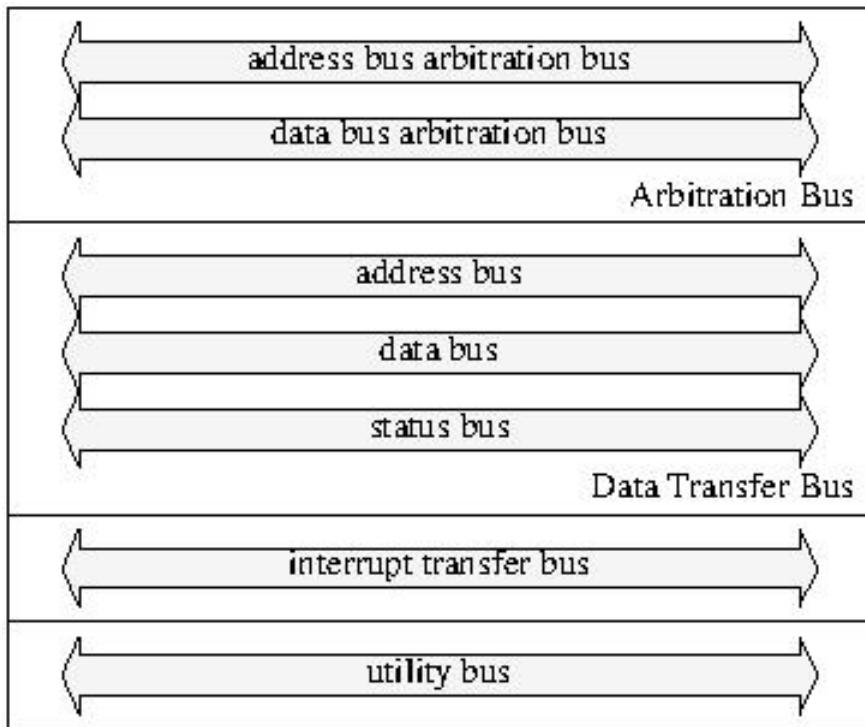


Figure 1.4: 버스 블록의 구조

- 동일 이름을 갖는 여러 비트의 신호중 임의의 한 신호 또는 한 비트를 지칭할 때는 <n> 을 사용한다.
예: A<n>*, D<n>*, TT<n>*
- 동일 이름을 갖는 여러 비트의 신호를 총칭할 때는 비트 폭을 모두 표시하거나 또는 비트 폭 표시가 없는 Mnemonics만으로 표시한다.
예: A<31..4>* 또는 A*; D<127..0>* 또는 D*

1.3.6.2 신호의 상태에 대한 규칙

- 신호의 상태는 신호의 논리적인 값(“참”/“거짓”)을 표현하는 방법을 주로 사용하고, 경우에 따라 버스상에 전압의 높낮음(“high”/“low”)으로 표현하는 방법을 사용한다.
- 신호의 논리적인 값을 표현하기 위하여 “참”, “거짓”, “true”, “false”, T (True의 약자), F (False의 약자), 1, 0을 사용한다. “참”, “true”, T, 1이 같은 값이고, “거짓”, “false”, F, 0이 같은 값이다. 신호의 논리적인 값을 나타낼 때 신호 뒤에 붙는 *는 논리적인 값에 의미를 갖지 않는다.
예: AE* = T, AE = 1, AE* = “true”, AE* = “low”, AE* = L : same state
예: DE* = F, DE = 0, DE* = “false”, DE* = “high”, DE* = H : same state
- 신호의 상태 중 버스 상의 전압 레벨로 표현하기 위하여 “high”, “low”, H, L를 사용한다. 이때 신호의 뒤에 붙는 *는 전압의 레벨을 반대로 하는 역할을 한다.

예: AE* = “high”, AE* = H, AE = L : same state

예: DE* = “low”, DE* = L, DE = H : same state

- 여러개의 신호가 모여서 의미를 갖는 상태를 나타낼 경우 그 신호의 논리적인 값은 숫자로 표현할 수 있다. 이때 값의 표현 방법은 16진수(Hexadecimal)를 기본(default)으로 한다.

16진수(Hexadecimal)를 나타내는 prefix로 ‘0x’를 사용한다.

예: A<31..4>* = 12345, TT<3..0>* = 4, AS<3..0>* = 0x5

- 신호의 상태를 나타낼 때 논리적인 값을 사용하는 것을 기본(default)으로 한다. 단, 신호의 시간적인 규격을 기술할 때는 전압 레벨을 사용한다(신호의 이름에 *의 유무 확인요).

1.3.6.3 시간 규격 표시에 대한 규칙

- 시간 규격은 $nsec$ (10^{-9} 초)를 기본 단위로 한다. 단, 예외의 경우는 별도로 시간 단위를 표시한다.
- 시간 규격에서 사용하는 신호의 타이밍도는 실제 버스에 나타나는 신호를 기준으로 설명함을 원칙으로 하고, 다만 설명의 편의를 위해 구동소자에 관련된 동작을 부가적으로 표시한다.

1.3.6.4 신호의 값을 지정하는 일반적인 규칙

- 신호의 값을 쉽게 구별되도록 지정한다.
 - 예 : TT<4..0>* 신호에서 TT<4>*의 값이 “참(1)”이면 블록에 관련됨을 나타낸다.
 - 예 : TT<4..0>* 신호에서 TT<3>*의 값이 “참(1)”이면 쓰기에 관련됨을 나타낸다.
- 응답 신호의 경우 버스에서의 신호의 값이 기본적으로 오류인 것으로 지정한다.
 - 예 : DACK*의 경우 해당 신호의 값을 구동하지 않으면 버스 상에서 전압이 높은 상태로 유지되므로 자동으로 오류로 값을 갖게 된다.
- 신호의 값 중 일부가 *don't care*인 경우 논리 값이 “0”이 되게 하여 버스에서 높은 전압 값을 유지하도록 한다.

Chapter 2

중재 버스

2.1 개요

중재 버스(arbitration bus : ARB)는 어드레스 중재 버스(address arbitration bus : AARB)와 데이터 중재 버스(data arbitration bus : DARB)로 구성된다.

Table 2.1: 중재 버스의 규격

중재 프로토콜 (Arbitration Protocol)	
Arbitration Rule	Priority, Fairness
Arbitration Time	1 Bus Clock (60.6 nsec)
Arbitration Scheme	Linear Self Arbitration
Arbiter Type	Distributed Arbiter
어드레스 중재 버스 (Address Arbitration Bus)	
Arbitration Rule	Fairness and Priority
Fairness Algorithm	Fairness check and internal pending
Max. No. of Request Modules	13
Etc.	Arbitration Inhibit Function
데이터 중재 버스 (Data Arbitration Bus)	
Arbitration Rule	Fairness and Priority
Fairness Algorithm	Fairness check and internal pending
Starvation Inhibit Algorithm	Time out
Max. No. of ResPonder Modules	9
기타 (Etc.)	
Total No. of Signals	26

2.2 중재 버스의 신호선

중재 버스(ARB)의 신호선은 어드레스 중재 버스(AARB)와 데이터 중재 버스(DARB)로 나뉘어진다. 데이터 전송 버스의 기본 주기 중 어드레스 데이터 주기를 제외하면 어드레스 버스와 데이터 버스는 상호 독립적으로 동작하기 때문에 중재 버스도 분리된다.

Table 2.2: 중재 버스의 신호들

Mnemonic	Size	Name
Address Arbitration Bus		
ABRQ<12..0>*	13	Address Bus Request
ABINH*	1	Address Bus Arbitration Inhibit
WRINH*	1	Write Bus Cycle Inhibit
Data Arbitration Bus		
DBRQ<8..0>*	9	Data Bus Request
DBINH*	1	Data Bus Arbitration Inhibit
PCW*	1	Priority Change Window

2.2.1 어드레스 중재 버스

2.2.1.1 Address Bus Request : ABRQ<12..0>*

RQ가 위치할 수 있는 슬롯마다 하나의 신호가 할당된다. <표 2.3>과 같이 슬롯0부터 슬롯12 까지 번호 순으로 할당되어 있다. 이 신호는 각 슬롯에 위치하는 RQ가 어드레스 버스를 사용하고자 할 때 구동하며, 어드레스 버스 사용허가를 받으면 “거짓”으로 구동한다. 각 슬롯에 꽂히는 RQ는 슬롯 어드레스 GA<4..0>*를 통하여 자신이 사용할 신호선을 알 수 있다. 중재 번호가 높을수록 우선순위가 높다.

Table 2.3: 어드레스 중재 버스의 중재 번호와 슬롯 어드레스

Slot Add	Add Bus Request	priority	Slot Add	Add Bus Request	priority
Slot 0	ABRQ<0>*	lowest	Slot 7	ABRQ<7>*	
Slot 1	ABRQ<1>*		Slot 8	ABRQ<8>*	
Slot 2	ABRQ<2>*		Slot 9	ABRQ<9>*	
Slot 3	ABRQ<3>*		Slot 10	ABRQ<10>*	
Slot 4	ABRQ<4>*		Slot 11	ABRQ<11>*	
Slot 5	ABRQ<5>*		Slot 12	ABRQ<12>*	highest
Slot 6	ABRQ<6>*				

2.2.1.2 Address Bus Arbitration Inhibit : ABINH*

어드레스 버스 중재 동작을 제한하여 어드레스 버스의 사용을 일시적으로 중단시키기 위해 사용되는 신호이다. 하나의 RQ가 이 신호를 “참”으로 구동하면 나머지 RQ들은 이 신호가 “거짓”이 될 때까지 어드레스 버스 중재 요청 신호 구동을 중단하게 된다. RP가 이 신호를 구동한 경우는 모든 RQ들이 이 신호가 “거짓”으로 될 때까지 어드레스 버스 중재 요청 신호 구동을 중단하게 된다. 이 신호는 여러개의 RP가 구동을 할 수 있지만, RQ의 경우는 반드시 하나의 RQ만이 구동하도록 해야 한다. 따라서 RQ가 이 신호를 구동하기 위해서는 먼저 어드레스 버스 중재 동작을 수행하여 버스 사용권을 획득해야만 한다. 보통 동작 상태에서는 미리 정해진 일정한 시간 이상의 구동은 허용되지 않는다.

이 신호는 중재 주기 단위로 제어를 하며, “참”으로 구동되어 있는 경우 진행 중인 중재의 결과는 아무런 의미를 갖지 못한다. 이후 이 신호가 “거짓”이 될 때까지 어드레스 버스 중재 동작이 제한을 받게 된다.

2.2.1.3 Write Bus Cycle Inhibit : WRINH*

이 신호가 사용되는 경우는 다음 두가지 경우이다.

첫째, 블록 전송 중 쓰기 전송인 경우 어드레스 중재의 결과로 어드레스를 구동하는 RQ가 어드레스 구동 시점부터 두번째 데이터 구동 시점까지 이 신호를 구동하여 다른 쓰기 전송과의 데이터 버스 충돌을 방지한다.

둘째, 블록 전송 중 읽기 전송에 대한 블록 전송 데이터 기본주기를 수행할 때 해당 RP가 데이터 버스 중재 주기 시점부터 두번째 데이터 구동 시점까지 이 신호를 구동하여 다른 쓰기 전송과의 데이터 버스 충돌을 방지한다.

이 신호가 “참”으로 구동되어 있는 경우 진행 중인 중재중 쓰기 전송에 관련되는 것은 그 결과를 포기하게 되고 이 신호가 “거짓”이 될 때까지 어드레스 버스 중재 동작이 제한을 받게 된다. 그러나 읽기 전송의 경우는 무관하게 어드레스 버스 중재가 수행된다.

2.2.2 데이터 중재 버스

2.2.2.1 Data Bus Request : DBRQ<8..0>*

RP가 위치할 수 있는 슬롯마다 하나의 신호가 할당된다. 즉, 슬롯10과 슬롯13-20까지 이 신호가 하나씩 할당된다. <표 2.4>는 각 슬롯에 대응되는 신호를 보여주고 있다. 이 신호는 각 슬롯에 위치하는 RP가 데이터 버스를 사용하고자 할 때 구동하며, 데이터 버스 사용허가를 받으면 “거짓”으로 구동한다. 각 슬롯에 꽂히는 RP는 슬롯 어드레스 GA<4..0>*를 통하여 자신이 사용할 신호선을 알 수 있다. 중재 번호가 높을수록 우선순위가 높다.

2.2.2.2 Data Bus Arbitration Inhibit : DBINH*

이 신호가 사용되는 경우는 다음 세가지 경우이다.

첫째, 단일 전송 중 쓰기 전송인 경우 어드레스 중재의 결과로 어드레스를 구동하는 RQ가 어드레스 구동 시점에 이신호를 구동하여 데이터 버스 충돌을 방지한다.

둘째, 블록 전송 중 쓰기 전송인 경우 어드레스 중재의 결과로 어드레스를 구동하는 RQ가 어드레스 구동 시점부터 세번째 데이터 구동 시점까지 이 신호를 구동하여 다른 데이터

Table 2.4: 데이터 중재 버스의 중재 번호와 슬롯 어드레스

Slot Add	Data Bus Request	priority	Slot Add	Data Bus Request	priority
Slot 10	DBRQ<8>*	highest	Slot 17	DBRQ<3>*	
Slot 13	DBRQ<7>*		Slot 18	DBRQ<2>*	
Slot 14	DBRQ<6>*		Slot 19	DBRQ<1>*	
Slot 15	DBRQ<5>*		Slot 20	DBRQ<0>*	lowest
Slot 16	DBRQ<4>*				

기본주기 또는 블록 데이터 기본주기와의 데이터 버스 충돌을 방지한다.

세째, 블록 전송 중 읽기 전송에 대한 블록 전송 데이터 기본주기를 수행할 때 해당 RP가 데이터 구동 시점부터 세번째 데이터 구동 시점까지 이 신호를 구동하여 데이터 버스 충돌을 방지한다.

이 신호가 “참”으로 구동되어 있는 경우 진행 중인 데이터 버스 중재의 결과는 무시하게 되고 이 신호가 “거짓”이 될 때까지 데이터 버스 중재 동작이 제한을 받게 된다.

2.2.2.3 Priority Change Window : PCW*

이 신호는 블록 전송 중 읽기 전송에서 연속된 블록 전송 데이터 기본 사이클이 데이터 버스 중재 때 부터 WRINH* 신호를 구동함으로써 쓰기 전송을 막는 결과를 초래할 수 있는 경우를 방지하기 위해 사용한다.

블록 전송 데이터 기본 사이클을 위한 데이터 버스 중재에서 데이터 버스 사용권을 획득한 경우 데이터 버스사용과 함께 구동을 시작하여 WRINH* 신호를 걸어낼때 같이 걸어낸다. 즉 블록 전송 데이터 기본 사이클을 수행하는 자신이 구동하는 DBINH*, WRINH* 신호가 모두 구동되는 경우 구동한다.

PCW*는 블록 전송 데이터 기본 사이클에서 데이터 중재에서 데이터 버스 사용권을 획득한 보드가 구동하고, PCW* 신호가 구동되어 있는 것을 확인한 다른 보드는 블록 전송 데이터 기본 사이클을 위한 DBRQ<n>*와 WRINH* 신호 구동을 멈추어야 한다. 그리고 PCW*에 의해 중단된 블록 전송 데이터 기본 사이클은 PCW* 신호가 “거짓”이 된 것을 발견하면 재시도 한다.

2.3 중재 방법

2.3.1 선형 자가 중재 기법

선형 자가 중재 기법(linear self arbitration scheme : LSAS)의 특징은 중재에 참가하는 각 중재기에 개별적인 중재 요청선을 할당하며, 이들 요청선들의 집합이 중재 버스를 형성하고, 중재 동작이 각 중재기에서 동일하게 수행되어 스스로 중재를 수행한다는 것이다. 이때 중재의 결과는 우선순위에 의해 결정된다.

<그림 2.1>에 선형 자가 중재 기법에 기초한 중재기(arbiter)의 일례를 나타내었다. 중재는 버

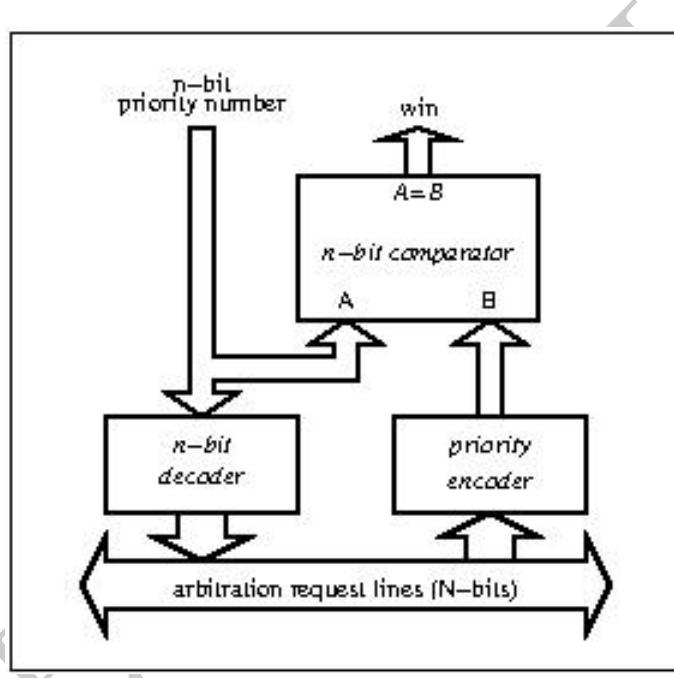


Figure 2.1: 선형 자가 중재기의 일례

스 클럭을 기본단위로 하여 이루어진다. 즉 매 버스 클럭마다 한번의 중재가 이루어진다. 각 중재기에는 n -비트 우선순위(priority number)가 할당된다. n -to-1 멀티플렉서(multiplexer)는 n -비트 우선순위 값을 입력으로 받아 이를 디코딩(decoding)하여 N -비트 폭의 신호선들 중 특정한 하나의 신호선을 구동한다. 이때 n 과 N 은 $n = \lceil \log_2 N \rceil^1$ 의 관계에 있다. 우선순위 엔코더(priority encoder)는 N -비트 중재신호를 입력으로 받아 이들 중 가장 높은 우선순위를 갖는 신호의 번호를 엔코딩하여 출력한다. 비교기(comparator)는 미리 할당된 중재번호와 우선순위 엔코더에서 출력된 값을 비교하여 이들 둘이 같은지 여부를 출력한다. 따라서 비교기에서 출력이 있으면 해당 중재기가 현재 진행 중인 중재동작에서 가장 우선순위가 높은 것이므로 중재에서 이긴 것이 된다.

¹ $[x]$: denotes the least integer that is greater than or equal to x .

2.3.2 공정성 규칙

선형 자가 중재 방법은 기본적으로 우선 순위에 기초한 것이므로 우선순위가 높은 중재기가 상대적으로 자주 중재에서 이길 것이므로 균등한 중재 결과를 보장하기는 어렵다. 이러한 문제를 해결하기 위해서 가능한 균등하게 사용 기회를 제공하도록 하는 공정성 규칙(fairness rule)을 고려한다.

<그림 2.2>에 공정성 규칙을 나타내었다. 중재가 버스 클럭을 기본단위로 이루어지므로 공

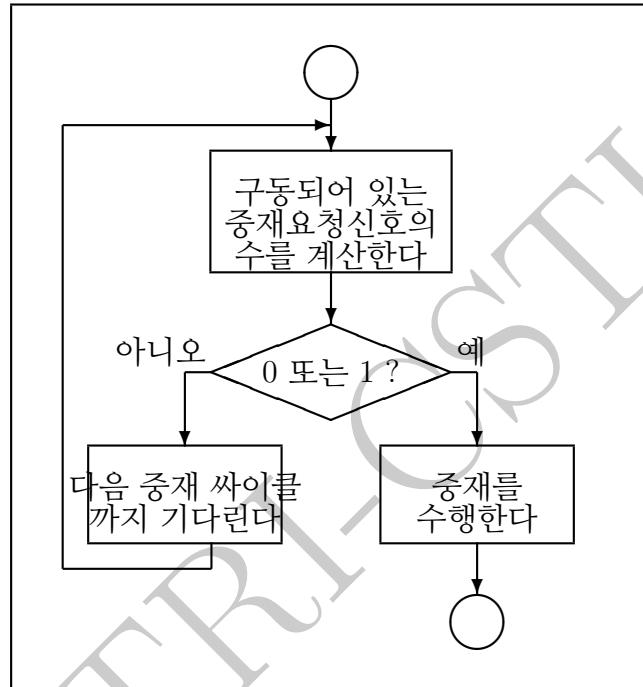


Figure 2.2: 공정성 중재 규칙

정성 규칙도 버스 클럭 단위로 이루어진다. 중재에 참가하기에 앞서 현재 중재에 참가하고 있는 중재기의 갯수를 파악한다. 이것은 중재 버스에 현재 구동되어 있는 중재 요청 신호의 갯수를 알아내는 것으로 가능하다. 그 갯수가 두개 이상이면 현재 중재에 참가 중인 중재기가 두개 이상이라는 것을 의미하고 이어지는 버스 클럭에는 현재 진행 중인 중재에서 이기지 못한 중재 요청이 한개 또는 그 이상 남아 있을 것이 확실하므로 곧 바로 중재에 참가하지 않고 또 다시 공정성 규칙에 따라 점검을 한다. 그러나 만약 그 갯수가 한개 또는 없음이 확인되는 경우는 이어지는 버스 클럭에는 현재 진행 중인 중재에서 이기지 못한 중재기가 없을 것이 확실하므로 곧 바로 중재에 참가한다. 이러한 중재 규칙은 시간적으로 우선하여 발생한 중재 요청은 항상 먼저 처리됨을 보장하게 된다.

2.4 어드레스 버스 중재

어드레스 버스 중재(address bus arbitration)는 어드레스 기본 사이클(address basic cycle)과 어드레스 데이터 기본 사이클(address data basic cycle)에 앞서 항상 수행해야 된다. 어드레스 버스 중재는 선형 자가 중재 기법을 사용하며 공정성 규칙을 지켜야 한다. 그러나 재시도(retry)하는 경우는 공정성 규칙을 무시한다. ABINH* 신호가 구동되어 있는 동안은 중재를 수행하지 않거나 중재의 결과를 무시하여야 한다. WRINH* 신호가 구동되어 있는 동안은 쓰기에 관련된 전송의 중재를 수행하지 않거나 중재의 결과를 무시하여야 한다. 일정시간 이상 메모리 참조에 실패한 경우 ABINH* 신호를 구동할 수 있다.

2.5 데이터 버스 중재

데이터 버스 중재(data bus arbitration)는 데이터 기본 사이클(data basic cycle)에 앞서 항상 수행해야 된다. 데이터 버스 중재는 선형 자가 중재 기법을 사용한다. 데이터 버스 중재는 공정성 규칙을 지켜야 하며 이 공정성 규칙은 시간적으로 우선하여 발생한 요청을 먼저 처리할 수 있도록 한다. DBINH* 신호가 구동되어 있는 동안은 중재를 수행하지 않거나, 중재의 결과를 무시하여야 한다. 이것은 쓰기에 관련된 전송에서 구동하는 데이터와 데이터 버스 중재의 결과에 의해 구동하는 데이터가 충돌하는 것을 방지하기 위해서이다.

일정시간 이상 데이터 중재에 실패하는 경우 ABINH* 신호를 구동할 수 있다.

PCW* 신호가 구동되어 있는 것이 확인 된 경우 확인 된 다음 사이클 부터 PCW* 신호가 걸어진 그 다음 사이클 까지 데이터 버스 중재 동작에 참여하여서는 안된다. 즉 DBRQ<n>* 신호와 WRINH* 신호를 구동하여서는 안된다.

Chapter 3

데이터 전송 버스

3.1 개요

데이터 전송 버스(data transfer bus : DTB)는 데이터의 전송에 직접적으로 관련되는 신호선의 모임이며, 어드레스 버스(address bus : AB), 데이터 버스(data bus : DB), 그리고 상태 버스(status bus : SB)로 구성된다.

Table 3.1: 데이터 전송 버스의 규격 요약

데이터 전송 프로토콜 (Data Transfer Protocols)	
Protocol	Pended
제어 방식	동기형 (Synchronous)
클럭의 속도	16.5 MHz (60.6 nsec)
데이터 전송 속도(Data Bandwidth)	264 Mbytes/sec
어드레스 버스의 특성 (Characteristics of Address Bus)	
최대 어드레스 영역의 크기	4 Gbytes
어드레스 영역의 갯수	8
최대 사용 가능한 전송 형태	32
데이터 버스의 특성 (Characteristics of Data Bus)	
버스의 폭 (Bus Width)	128 bits (16 bytes)
데이터의 단위 (Data Unit)	8 bits (1 byte)
전송 가능한 데이터의 크기	16-바이트 이내의 연속된 바이트; 64-바이트
정렬의 제약 (Alignment Restriction)	16 byte boundary; 64 byte boundary
Justification	Straight (Nonjustified)
에러 방어 (Error Protection)	
에러 검출 (Error Detection)	바이트 단위의 홀수 패리티 (odd parity)
에러의 처리 (Error Handling)	재시도 (Retry)
기 타 (Etc.)	
캐시 지원	Write Back
동기화 지원 (Synchronization)	Semaphore Cache Protocol
총 신호수	244

3.2 데이터 전송 버스의 신호선

Table 3.2: 데이터 전송 버스의 신호선 요약

Mnemonics	size	name
Address Bus		
A<31..4>*	28	Address
AP<3..0>*	4	Address Parity
SI<7..0>*	8	Source Identification
SIP*	1	Source Identification Parity
AS<2..0>*	3	Address Space
TT<4..0>*	5	Transfer Types
STP*	1	Space + Types Parity
BE<15..0>*	16	Byte Enable
BEP<1..0>*	2	Byte Enable Parity
AE*	1	Address Bus Enable
Data Bus		
D<127..0>*	128	Data
DP<15..0>*	16	Data Parity
DI<7..0>*	8	Destination Identification
DIP*	1	Destination Identification Parity
DE*	1	Data Bus Enable
Status Bus		
AACK<1..0>*	2	Address Acknowledge
SHD*	1	Hit on Shared Line
DTY*	1	Hit on Dirty Block
SNK*	1	Snoop No Acknowledge
ITV*	1	Intervention
LCR*	1	Hit On Interlocked Region
SPIN<3..0>*	4	Spin Queue Order
DACK*	1	Data Acknowledge
CDK*	1	Cache Data Acknowledge
BSY<7..0>*	8	Busy Status Line

3.2.1 어드레스 버스의 신호선 (signal lines of address bus)

어드레스 버스는 어드레스, 어드레스 영역, 전송의 종류, 전송하고자 하는 데이터의 크기, 어드레스를 구동하는 모듈의 식별자, 각 신호의 패리티, 그리고 어드레스 버스가 구동되고 있음을 알리는 제어신호로 구성된다.

3.2.1.1 Address : A<31..4>*

전송할 데이터의 위치를 알리는 주소이다. 32 비트 중 A<31>*이 가장 높은 자리수이다. A<3..0>* 4개 비트가 없는 대신 16 바이트 이내의 유효 데이터는 BE<15..0>*에 의해 식별된다.

3.2.1.2 Address Parity : AP<3..0>*

어드레스 신호선의 각 바이트에 해당하는 패리티이다. AP<0>*값은 A<3..0>*의 값을 0x0으로 간주한 상태에서 계산된 값이다.

Table 3.3: 어드레스와 패리티 비트

address	address parity	address	address parity
A<31..24>*	AP<3>*	A<23..16>*	AP<2>*
A<15..8>*	AP<1>*	A<7..0>*	AP<0>*

3.2.1.3 Source Identification : SI<7..0>*

어드레스의 구동에 의해서 버스 동작이 시작되며 이 버스 동작은 데이터를 요청하거나 쓰거나 한다. 이때 어드레스 구동과 같은 시간에 해당 SI<7..0>*를 구동하여 버스 동작을 시작한 모듈을 알린다. 비트 5 - 2에 채워지는 슬롯 어드레스는 해당 슬롯의 GA<3..0>* 값이고, 비트 1 - 0에 채워지는 값은 해당 보드 내에서 정의한다.

Table 3.4: 시발점 식별자

field	meaning
SI<7..6>*	<i>reserved</i>
SI<5..2>*	slot address
SI<1..0>*	channel identification

3.2.1.4 Source Identification Parity : SIP*

SI<7..0>*의 패리티 비트이다.

3.2.1.5 Address Space : AS<2..0>*

어드레스 영역을 구분하기 위한 신호이며 3비트의 신호로 8개의 독립적인 어드레스 영역을 사용할 수 있지만 현재 정의하여 사용하는 것은 메모리 영역, 입출력 영역, 그리고 시스템 영역이고 나머지는 확장을 위해 사용을 유보한다. 각 어드레스 영역은 4GBytes의 크기를 갖는다.

Table 3.5: 정의된 어드레스 영역

AS<2..0>*	name	size	application
0x0	memory space	4GBytes	shared memory
0x3	input/output space	4GBytes	memory mapped peripherals
0x7	system space	4GBytes	system configuration registers

3.2.1.6 Transfer Types : TT<4..0>*

전송의 종류를 정의하는 신호이다.

정의된 데이터 전송은 데이터의 이동 방향에 따라 읽기 전송, 쓰기 전송, 그리고 어드레스 단일 전송으로 구분된다. 읽기 전송은 데이터가 RP에서 RQ로 공급되며, 쓰기 전송은 데이터가 RQ에서 RP로 공급된다. 어드레스 단일 전송은 데이터의 이동이 없다.

각 전송은 이동되는 데이터의 크기에 따라 단일 전송과 블록 전송으로 구분된다. 단일 전송의 경우는 16바이트 경계로 정렬된 어드레스(어드레스를 16로 나누어 나머지가 없는 경우)에서 1-바이트, 또는 연속된 16-바이트 이내의 데이터를 BE<15..0>*로 지정하여 전송가능하다. 블록 전송의 경우는 64바이트 경계로 정렬된 어드레스(어드레스를 64로 나누어 나머지가 없는 경우)에서부터 64바이트를 전송가능하다. 이때 BE<15..0>*는 0xFF로 구동한다. 참고로, TT<4>*는 블록 전송을, TT<3>*은 쓰기 전송임을 나타낸다.

Table 3.6: 전송 형태

TT<4..0>*	mnemonics	name	group	affect cache	size
0x00	NRD	normal read	read	yes	1-16
0x08	NWR	normal write	write	yes	1-16
0x07	NCR	non-coherent read	read	no	1-16
0x0F	NCW	non-coherent write	write	no	1-16
0x03	IVD	invalidation		yes	
0x04	ILR	interlock read	read	yes	1-16
0x0C	ILW	interlock write	write	yes	1-16
0x10	BRD	block read	read	yes	64
0x18	BWR	block write	write	yes	64
0x12	CRD	coherent read	read	yes	64
0x11	EXR	exclusive read	read	yes	64
0x19	WRB	writeback	write	yes	64
0x1B	ITW	intervention write	write	yes	64

- Normal Read : NRD

단일 전송에 속하며, 단순히 RP로부터 원하는 데이터를 읽어온다. 이때 해당 데이터는

캐쉬에 저장되지 않는다.

- Normal Write : NWR

단일 전송에 속하며, 단순히 RP로 특정 데이터를 쓴다.

- Non-Coherent Read : NCR

단일 전송에 속하며, 단순히 RP로부터 원하는 데이터를 읽어온다. 캐쉬 동일성 유지 프로토콜과 전혀 관련이 없는 데이터가 읽혀질 때 사용된다. 따라서 시스템 내의 모든 캐쉬는 이 전송에 대하여 아무 것도 고려할 것이 없다.

- Non-Coherent Write : NCW

단일 전송에 속하며, 단순히 RP로 특정 데이터를 쓰는 동작으로 캐쉬 동일성 유지 프로토콜과 전혀 관련이 없는 특정 데이터를 쓸 때 사용한다. 따라서 시스템 내의 모든 캐쉬는 이 전송에 대하여 아무 것도 고려할 것이 없다.

- Invalidiation : IVD

어드레스 단일 전송에 속하며, 캐쉬 동일성 유지 프로토콜을 유지할 책임이 있는 RQ 가 자신의 캐쉬에 있는 유효 데이터를 시스템을 통하여 유일한 유효 데이터로 만들기 위해 캐쉬 동일성 유지 프로토콜을 유지할 책임이 있는 다른 RQ들에게 알릴 때 사용한다. 즉, 다른 캐쉬에 있는 동일 데이터를 무효화시킬 때 사용한다. 이 전송은 어드레스 기본 사이클만으로 수행된다. 특히 IVD의 경우 AACK<1..0>*는 의미를 갖지 않고 대신 SNK*이 사용된다.

- Interlock Read : ILR

단일 전송에 속하며, 잠금동작을 위한 전송으로 이 전송에 의해 잠금이 시작된다. NRD 와 동일한 버스동작으로 수행되지만 이 전송이 완료된 이후는 해당 영역이 잠금상태가 된다.

- Interlock Write : ILW

단일 전송에 속하며, 잠금동작을 위한 전송으로 이 전송에 의해 잠금이 풀린다. NWR와 동일한 버스동작으로 수행되지만 이 전송이 완료된 이후는 해당 영역이 잠금상태에서 풀린다.

- Block Read : BRD

블록 전송에 속하며, 단순히 RP로부터 원하는 데이터를 읽어오며, 이때 해당 데이터는 캐쉬에 저장되지 않는다.

- Block Write : BWR

블록 전송에 속하며, 단순히 RP로 특정 데이터를 쓴다.

- Coherent Read : CRD

블록 전송에 속하며, 캐쉬 동일성 유지 프로토콜을 유지할 책임이 있는 RQ가 데이터를 읽을 때 사용되며, 읽혀진 데이터가 RQ내의 캐쉬에 저장되어 사용된다. 이때 그 데이터가 전체 시스템을 통하여 유일한 유효 데이터일 필요는 없다.

- Exclusive Read : EXR

블록 전송에 속하며, 캐쉬 동일성 유지 프로토콜을 유지할 책임이 있는 RQ가 데이터를 읽을 때 사용되며, 읽혀진 해당 데이터는 전체 시스템을 통하여 유일한 유효 데이터이여야 한다.

- Writeback : WRB

블록 전송에 속하며, 캐쉬 동일성 유지 프로토콜을 유지할 책임이 있는 RQ가 자신의 캐쉬에 있는 유일한 유효 데이터를 메모리로 또는 메모리와 다른 RQ로 전달할 때 사용한다. 이때 DI*가 실제 데이터의 종착지를 결정한다. (주의: RQ에서 RQ로만 데이터가 전달될 때는 ITW를 사용해야만 한다.)

- Intervention Write : ITW

블록 전송에 속하며, 캐쉬 동일성 유지 프로토콜을 유지할 책임이 있는 RQ가 자신의 캐쉬에 있는 유효 데이터를 다른 특정 RQ에 전달할 때 사용한다. 이 전송은 인터벤션이 선행되는 전송이다. 특히 ITW의 경우 AACK<1..0>*는 의미를 갖지 않고, 대신 SNK*가 사용되며, DACK*는 의미를 갖지 않고 CDK*가 사용된다.

3.2.1.7 Address Space and Transfer Types Parity : STP*

AS<2..0>*와 TT<4..0>*를 합한 8 비트에 대한 패리티 비트이다.

3.2.1.8 Byte Enable : BE<15..0>*

단일 전송(NRD, NWR, NCR, NCW, ILR, ILW) 때만 유효하며, 1에서 16까지의 연속된 데이터를 지정하는 신호이다. 16 비트의 신호는 데이터 버스의 각 바이트에 대응한다. 블록 전송의 경우 0xFF로 구동한다.

Table 3.7: 바이트 지정자

data bus	byte enable bit	data bus	byte enable bit
D<127..120>*	BE<15>*	D<63..56>*	BE<7>*
D<119..112>*	BE<14>*	D<55..48>*	BE<6>*
D<111..104>*	BE<13>*	D<47..40>*	BE<5>*
D<103..96>*	BE<12>*	D<39..32>*	BE<4>*
D<95..88>*	BE<11>*	D<31..24>*	BE<3>*
D<87..80>*	BE<10>*	D<23..16>*	BE<2>*
D<79..72>*	BE<9>*	D<15..8>*	BE<1>*
D<71..64>*	BE<8>*	D<7..0>*	BE<0>*

3.2.1.9 Byte Enable Parity : BEP<1..0>*

BE<15..0>*에 대한 패리티 비트이다.

Table 3.8: 바이트 지정자와 패리티 비트

byte enable	byte enable parity	byte enable	byte enable parity
BE<15..8>*	BEP<1>*	BE<7..0>*	BEP<0>*

3.2.1.10 Address Bus Enable : AE*

현재 진행 중인 주기에서 어드레스 버스의 신호들이 유효한 값을 갖는다는 것을 나타낸다.

3.2.2 데이터 버스의 신호선 (signal lines of data bus)

데이터 버스는 데이터, 해당 데이터가 도착해야 하는 모듈(들)의 식별자, 각 신호의 패리티, 그리고 데이터 버스가 구동되고 있음을 알리는 제어신호로 구성된다.

3.2.2.1 Data : D<127..0>*

128 비트 (16 바이트) 폭의 데이터 신호선이다.

3.2.2.2 Data Parity : DP<15..0>*

데이터 신호선의 각 바이트에 해당하는 패리티이다. 16 바이트 이내의 데이터를 전송할 때 BE<15..0>*에 의해 지정된 유효한 바이트에 대응하는 패리티는 반드시 정확한 패리티 값을 가져야하고 나머지는 상관하지 않아도 된다. 그러나 가능한 모든 바이트의 패리티가 정확한 값을 갖는 것이 좋다.

Table 3.9: 데이터와 데이터 패리티

data bus	data parity bit	data bus	data parity bit
D<127..120>*	DP<15>*	D<63..56>*	DP<7>*
D<119..112>*	DP<14>*	D<55..48>*	DP<6>*
D<111..104>*	DP<13>*	D<47..40>*	DP<5>*
D<103..96>*	DP<12>*	D<39..32>*	DP<4>*
D<95..88>*	DP<11>*	D<31..24>*	DP<3>*
D<87..80>*	DP<10>*	D<23..16>*	DP<2>*
D<79..72>*	DP<9>*	D<15..8>*	DP<1>*
D<71..64>*	DP<8>*	D<7..0>*	DP<0>*

3.2.2.3 Destination Identification : DI<7..0>*

데이터 버스에 구동되고 있는 데이터가 실제로 어디로 이동하는 것인지를 알리기 위해 사용된다. 메모리 모듈만 지정되는 경우는 RQ에 의해 메모리 쓰기가 수행될 때 사용하며, 이때 비트 5 - 0의 값은 데이터를 구동하는 RQ를 구별하는 정보이다. 특정 RQ만 지정되는 경우는 메모리 읽기에 대한 데이터 응답이나 캐쉬에서 캐쉬로 데이터 이동이 필요할 때 사용하며, 이때 비트 5 - 0의 값은 데이터를 받을 RQ를 구별하는 정보이다. RQ와 RP가 동시에 지정되는 경우는 비트 7 - 6의 값이 0x3이고 비트 5 - 0의 값은 데이터를 받을 특정 RQ를 구별하는 정보이다.

Table 3.10: 종착점 식별자

DI<7..6>*	DI<5..2>*	DI<1..0>*	meaning
00	<i>reserved</i>	<i>reserved</i>	<i>reserved</i>
01	slot address	channel id.	to RQ only
10	slot address	channel id.	to RP only
11	slot address	channel id.	to RQ and RP

3.2.2.4 Destination Identification Parity : DIP*

DI<7..0>*의 패리티 비트이다.

3.2.2.5 Data Bus Enable : DE*

현재 진행 중인 주기에서 데이터 버스의 신호들이 유효한 값을 갖는다는 것을 나타낸다.

3.2.3 상태 버스의 신호선 (signal lines of status bus)

상태 버스는 어드레스 전달에 대한 응답, 데이터 전달에 대한 응답, 그리고 캐쉬 동일성 유지 프로토콜에 필요한 제어 신호, 잠금에 대한 응답 신호로 구성된다.

3.2.3.1 Address Acknowledge : AAC<1..0>*

어드레스 전달에 대해 어드레스를 받은 모듈이 그 상태를 알리는 신호이다.

3.2.3.2 Hit On Shared Line : SHD*

RQ_1 에 의해 진행되는 버스 동작에 의해 요청되는 데이터의 유효한 내용을 자신의 캐쉬에 갖는 RQ_2 는 SHD* 신호선을 구동하여 RQ_1 으로하여금 요청되는 데이터가 다른 캐쉬에도 있음을 알린다. SHD*는 진행 중인 데이터 전송에 아무런 영향을 주지 않으며 단지 다른 캐쉬에 공유되고 있음을 알리는 신호이다.

Table 3.11: 어드레스 응답

AACK<1..0>*	status	description
00	error	전송된 어드레스 버스의 정보에 오류가 발견됨
01	busy	전송된 어드레스 버스의 정보를 현재 받을 수 없는 상태임
10	lock	요청한 영역이 잠겨 있음
11	no error	전송된 어드레스 버스의 정보를 잘 받았음

Table 3.12: SHD*

SHD*	description
0	-
1	hit on shared line

3.2.3.3 Hit On Dirty Line : DTY*

RQ_1 에 의해 진행되는 버스 동작에 의해 요청되는 데이터의 유효한 값이 메모리에 있지 않고 RQ_2 의 캐쉬에 있을 때, 자신의 캐쉬에 유일한 유효 데이터를 갖는 RQ_2 는 DTY* 신호선을 구동하여 RQ_1 의 진행 중인 데이터 전송을 중지시키고 메모리의 데이터 응답을 방지한다. DTY* 신호가 구동되면 해당 데이터 전송을 수행하는 RQ는 재시도를 해야하고 메모리는 데이터 응답을 하지 않아야 한다. 그리고 DTY* 신호를 구동한 RQ는 자신의 유일한 유효 데이터를 메모리에 반영할 의무를 진다.

Table 3.13: DTY*

DTY*	description
0	-
1	hit on dirty line

3.2.3.4 Intervention : ITV*

RQ_1 에 의해 진행되는 버스 동작에 의해 요청되는 데이터의 유효한 값이 메모리에 있지 않고 RQ_2 의 캐쉬에 있을 때, RQ_2 는 ITV* 신호선을 구동하여 메모리의 데이터 응답을 방지하고 자신이 가지고 있는 유효한 데이터를 RQ_1 에 공급할 수 있다. 그러나 데이터를 요구한 RQ_1 은 버스 동작을 계속 진행해야 한다.

ITV* 신호를 구동한 RQ_2 는 ITW 전송으로 자신의 유일한 유효 데이터를 해당 데이터를 요청한 RQ_1 에게 전달하거나, WRB 전송으로 RQ_1 과 메모리에 동시에 반영할 수 있다. ITV* 신호에 의해 인터벤션 상태에 있는 RQ_1 은 전달받은 데이터에 대한 DACK* 대신 CDK*로써

응답한다. 만약 CDK*로 애러의 발생을 알린 경우, ITV* 신호를 받은 RQ_1 은 요청을 재시도하며, ITV* 신호를 구동했던 RQ_2 는 전송을 다시 시도하지 않는다. 그러나 RQ_1 이 CDK*를 오류로 구동했으므로 재시도 할 것이고, 이것에 의해 다시 RQ_2 가 적절한 동작을 할 수 있다.

이 신호는 캐쉬 동일성 유지 프로토콜을 유지할 책임이 있는 RQ가 현재 요청되는 데이터를 자신의 캐쉬에서 공급하고자 할 때 사용한다. 이 신호가 구동되면 메모리는 데이터를 응답하지 말아야 하며, 이 신호를 구동한 RQ는 반드시 해당 데이터를 공급할 의무를 진다.

Table 3.14: ITV*

ITV*	description
0	-
1	intervention

3.2.3.5 Snoop No Acknowledge : SNK*

캐쉬 동일성 유지 프로토콜을 유지할 책임이 있는 RQ_1 이 현재 진행 중인 버스 동작이 자신의 캐쉬에 영향을 주는 것인데도 불구하고 자신이 이에 합당한 캐쉬 동작을 할 수 없거나 또는 진행 중인 버스 동작에 관련된 자신의 캐쉬 동작에 문제가 발생한 경우 해당 버스 동작을 더 이상 진행되지 못하도록 하기 위해 구동한다. SNK* 신호가 구동되면 해당 데이터 전송을 수행하는 RQ_2 는 재시도를 해야하고 메모리는 데이터 응답을 하지 않아야 한다.

Table 3.15: SNK*

SNK*	description
0	-
1	snoop no acknowledge

3.2.3.6 Hit On Interlocked Region : LCR*

진행되는 버스 동작에 의해 요청되는 영역이 잠겨 있음을 알린다. RQ_1 에 의해 진행되는 잠금동작과 관련되어서 아직 완료되기 이전에 참조하여서는 안되는 영역에 대해 RQ_2 에 의해 참조가 시도되는 경우 RQ_1 은 LCR* 신호를 구동하여 RQ_2 로 하여금 진행하고자 하는 전송이 잠금과 관련된 것임을 알린다.

LCR* 신호가 구동되면 해당 데이터 전송을 수행하는 RQ_2 는 재시도를 해야하고 메모리는 데이터 응답을 하지 않아야 한다.

3.2.3.7 Spin Queue Order : SPIN<3..0>*

스핀(spin) 상태인 세마포 캐쉬(semaphore cache)의 대기 순서.

Table 3.16: LCR*

LCR*	description
0	-
1	hit on interlocked region

3.2.3.8 Data Acknowledge : DACK*

데이터 전달에 대해 데이터를 받은 모듈이 전달된 데이타의 상태를 알리는 신호이다. DACK*는 읽기와 쓰기에 모두 사용된다. 읽기의 경우 RP에서 받은 데이터에 문제가 있을 때 RQ는 DACK*을 error로 구동하고 해당 전송을 처음부터 재시도한다. 쓰기의 경우에도 DACK*을 error로 받으면 해당 전송을 처음부터 재시도한다.

Table 3.17: 데이터 응답

DACK*	status	description
0	error	전송된 데이터 버스의 정보에 오류가 발견됨
1	no error	전송된 데이터 버스의 정보를 잘 받았음

3.2.3.9 Cache Data Acknowledge : CDK*

RQ_1 에서 RQ_2 로 데이터 전송이 이루어지는 ITW 전송에서, 전달되는 데이터에 대한 응답으로 RQ_2 가 구동하고 RQ_1 이 확인하는 신호이다. 그리고 CDK*에서 에러가 구동된 경우, 받는 쪽인 RQ_2 는 요청을 재시도하며, 전송측인 RQ_1 은 ITW 전송을 재시도하지 않는다. (CDK*의 값에 관계없이 RQ_1 은 자발적으로 재시도해서는 안된다. 즉 CDK*가 오류로 구동된 경우에도 RQ_2 가 재시도하기 때문이다.)

Table 3.18: CDK*

CDK*	description
0	error
1	no error

3.2.3.10 Busy Status Line : BSY<7..0>*

보드간 낮은차 인터리빙(low-order interleaving among boards)된 각 RP들은 자신의 상태에 따라 해당 BSY<n>* 신호을 구동할 수 있다. 예로써, 8-way 인터리빙된 8장의 RP가 있다면 자신에게 할당되는 어드레스 영역이 주어진 어드레스를 8로 나누었을 때 나머지에 따라

결정된다 (단 각 RP의 응답 크기에 해당하는 오프셋은 고려한 어드레스를 8로 나눈다). 나머지가 0인 경우 선택되는 RP는 $BSY<0>^*$, 나머지가 1인 경우 선택되는 RP는 $BSY<1>^*$, ..., 나머지가 7인 경우 선택되는 RP는 $BSY<7>^*$ 을 구동함으로써 해당 어드레스에 대한 요청이 RP에 의해 받아들여지지 못하고 $AACK<1..0>^*$ 을 busy로 받게 될 것임을 알리는 것이다.

RQ는 자신이 요청할 어드레스와 관련이 있는 $BSY<n>^*$ 신호가 구동된 경우에는 요청을 유보할 수 있다.

3.3 데이터 전송

버스는 기본적으로 데이터 이동의 통로이며 이러한 데이터 이동은 버스 상에 장착된 보드들 사이에서 이루어진다. 이들 데이터 이동에 참여하는 보드는 특정 시간에 어떤 일을 하느냐에 따라 RQ(ReQuester)와 RP(ResPonder)로 구분한다. RQ는 데이터 이동에 능동적으로 참여하며 데이터 전송을 시작하는 기능을 수행중인 보드이며, RP는 데이터 이동에 수동적으로 참여하며 RQ의 데이터 전송 요청에 응답하는 보드이다. 일반적으로 RQ는 프로세서 보드가 수행하는 기능에 해당하며, RP는 메모리 보드가 수행하는 기능에 해당한다. 그러나 프로세서 보드가 RP로도 동작한다면 RP라고 할 수 있다.

데이터 전송(data transfer)은 버스 동작(bus transaction)으로 수행되며, 버스 동작은 기본 사이클(basic cycle)로 구성된다. 데이터 전송은 데이터 이동의 관점에서 읽기, 쓰기, 그리고 무효화로 구분된다. 읽기는 RQ가 특정 주소의 데이터를 필요로 하여 요청하는 전송이며, 쓰기는 RQ가 특정 주소의 데이터를 공급하는 전송이다. 무효화는 데이터 이동이 없고 특정 주소에 대한 특별한 의미를 전달할 때 사용된다.

버스 동작은 데이터 전송을 위해 진행되는 일련의 동작으로 RQ에 의해 시작되며 기본 사이클로 이루어 진다. 특히 진행되는 버스 동작은 완전하게 종결되는 경우도 있지만 진행 중에 중단되는 경우도 있다. 완전하게 종결되는 경우 데이터 이동이 완벽하게 이루어진 경우이며 중단된 경우 해당 데이터 전송을 위해 처음부터 다시 버스 동작을 시도하여야 한다.

3.3.1 데이터 전송의 기본 사이클

기본 사이클은 분리될 수 없는 일정한 순서로 진행되는 단계(phase)로 구성되며, 각 단계는 한개의 버스 클럭 주기(bus clock period)가 소요된다. 기본 사이클에는 어드레스 기본 사이클(address basic cycle), 단일 전송 데이터 기본 사이클(data basic cycle), 단일 전송 어드레스 데이터 기본 사이클(address data basic cycle), 블록 전송 데이터 기본 사이클(block data basic cycle), 그리고 블록 전송 어드레스 데이터 기본 사이클(block address data basic cycle)이 있다.

어드레스 기본 사이클은 어드레스와 해당 어드레스에 관련된 정보의 이동 또는 전달에 사용된다. 이 어드레스 기본 사이클은 반드시 RQ에 의해 수행된다. 단일 전송 데이터 기본 사이클과 블록 전송 데이터 기본 사이클은 어드레스 기본 사이클에 의해 요청된 데이터의 전달에 사용되며, 반드시 RP에 의해 수행된다. 단일 전송 어드레스 데이터 기본 사이클과 블록 전송 어드레스 데이터 기본 사이클은 어드레스와 해당 어드레스의 데이터를 동시에 이동 또는 전달할 때 사용되며, 반드시 RQ에 의해 수행된다.

3.3.1.1 어드레스 기본 사이클

어드레스 정보(어드레스와 이에 관련된 정보들)를 전달할 때 사용하는 기본 사이클이며 반드시 어드레스 중재 과정을 거친 후에 수행되어야 한다. 이 기본 사이클은 3개의 단계로 구성되며 각 단계는 연속적이다. 어드레스 중재를 수행하여 어드레스 버스 사용권을 획득한 바로 다음 버스 클럭 주기에 어드레스 기본 사이클의 단계 0(phase 0)가 시작된다. RQ는 단계 0에서는 어드레스 정보를 어드레스 버스에 구동한다. 단계 1에서는 각 RP들이 단계 0에 전달된 어드레스 정보를 번역하여 필요한 동작을 준비한다. 이때 RQ는 그냥 기다린다.

단계 2에서는 단계 1에서 결정된 RP가 전달받은 어드레스 정보에 대한 응답(어드레스 응답)을 구동하며, RQ는 이 어드레스 응답을 받아 수행한 어드레스 기본 사이클의 완성 여부를 판단한다.

어드레스 기본 사이클에 의해 수행되는 버스 동작이 읽기인 경우 RQ는 RP로부터 데이터가 전달되어 오기를 기다리게 되고, 선택된 RP는 요청된 데이터를 전달하기 위해 데이터 기본 사이클을 수행해야 된다.

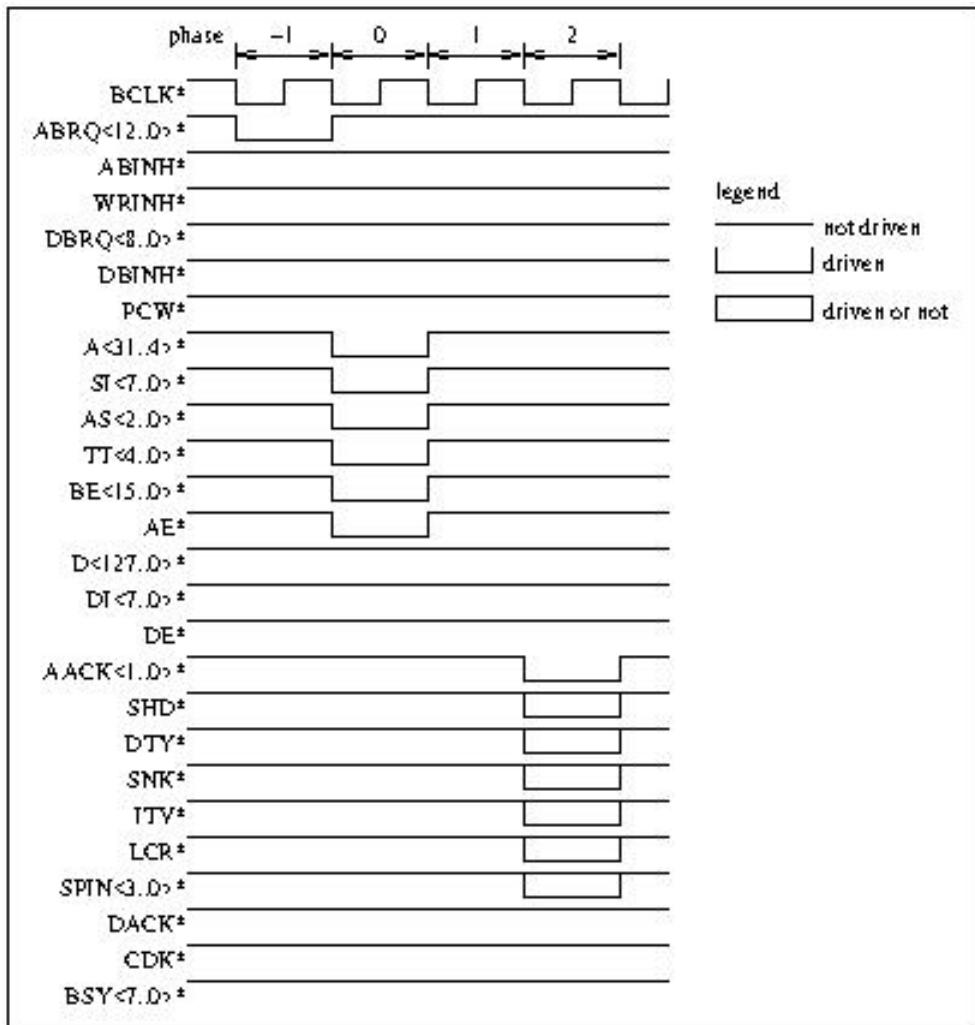


Figure 3.1: 어드레스 기본 사이클

- 어드레스 버스 중재 주기 - 단계 -1 (phase -1)
어드레스 버스에 해당 어드레스를 구동하기에 앞서 수행하는 어드레스 버스 중재 주기.
- 어드레스 기본 주기 - 단계 0 (phase 0)
중재 버스를 통하여 버스 사용허가를 받은 RQ가 한 버스 클럭 주기 동안 버스 상에 어드레스와 관련한 정보를 전송한다. 32 비트의 어드레스, 어드레스 영역, 전송 형태,

바이트 마스크, RQ의 슬롯과 채널 어드레스등과 각 신호의 패리티가 전송된다. 이 주기 동안 사용되는 신호는 A<31..4>*, AP<3..0>*, AS<2..0>*, TT<4..0>*, STP*, BE<15..0>*, BEP<1..0>*, SI<7..0>*, SIP*, AE* 등이다. RP들은 이 단계의 끝 부분에서 어드레스 버스에 실린 신호를 모두 저장한다. 캐쉬를 갖는 RQ들도 이 단계의 끝 부분에서 어드레스 버스에 실린 신호를 모두 저장한다.

- 어드레스 기본 주기 - 단계 1 (phase 1)

이 단계는 버스 상의 신호의 구동은 없다. 단계 0에서 어드레스 버스를 구동했던 RQ는 다음 동작을 준비한다. 단계 0에서 전송된 어드레스 등의 정보를 받은 RP들은 내부적으로 단계 2를 위한 동작이 수행된다. 우선 RP는 각 정보의 패리티 에러 확인을 한다. 패리티 에러가 발견되지 않은 경우 RP들은 저장한 어드레스를 번역하여 전송된 어드레스가 자신의 영역을 지정했는지를 확인한다. 선택된 하나의 RP만 자신의 상태를 점검하여 단계 2에서 전송할 신호의 내용을 준비한다.

캐쉬를 갖는 RQ들은 캐쉬 동일성 유지를 위한 동작을 내부적으로 수행한다.

- 어드레스 기본 주기 - 단계 2 (phase 2)

단계 2에서는 어드레스에 의해 선정된 한 RP가 단계 0에서 전송된 정보들에 대한 응답을 버스 상에 구동한다. 이때 사용되는 신호는 AACK<1..0>*이고, 단계 0에서 어드레스를 전송했던 RQ는 이 정보를 받아서 저장한다. 이 단계 후의 RP와 RQ의 내부적인 동작은 기본 주기에서 규정하지 않는다. 단계 1에서 패리티 에러가 검출된 경우 RP는 신호를 구동하지 않는다.

캐쉬를 갖는 RQ들은 단계 0에서 저장한 어드레스에 대해 캐쉬 동일성 유지 동작을 수행함에 있어서 문제가 발생하면 SNK* 신호를 구동한다. 또는 캐쉬 동일성 유지 동작에 필요한 SHD*, DTY*, ITV* 신호를 구동한다.

3.3.1.2 단일 전송 데이터 기본 사이클

이미 수행된 어드레스 기본 사이클에서 데이터 요청이 발생한 경우 RP가 해당 어드레스의 데이터를 RQ로 전달할 때 사용하며 반드시 데이터 버스중재 과정을 거친 후에 수행된다. 데이터 버스 중재를 수행하여 데이터 버스 사용권을 획득한 바로 다음 버스 클럭 주기에 데이터 기본 사이클이 수행된다.

- 데이터 버스 중재 주기 - 단계 -1 (phase -1)

데이터 버스에 해당 데이터를 구동하기에 앞서 수행하는 데이터 버스 중재 동작.

- 데이터 기본 주기 - 단계 0 (phase 0)

데이터 중재 버스를 통하여 데이터 전송 버스의 사용 허가를 받은 RP가 데이터 전송 버스의 신호들을 구동한다. 구동되는 버스의 신호는 D<127..0>*, DP<15..0>*, DI<7..0>*, DIP*, DE* 등이다. 데이터 신호는 어드레스 기본 주기에 의해 전송된 영역에 해당되는 데이터가 구동되고, 종착점 신호(DI<7..0>*)는 전송하는 데이터를 요청했던 RQ의 시발점 신호(SI<7..0>*)를 구동한다. 어드레스 주기를 통하여 RP로 데이터 요청을 한 RQ들(2개 이상이 될 수 있음)은 데이터 버스의 신호를 저장한다.

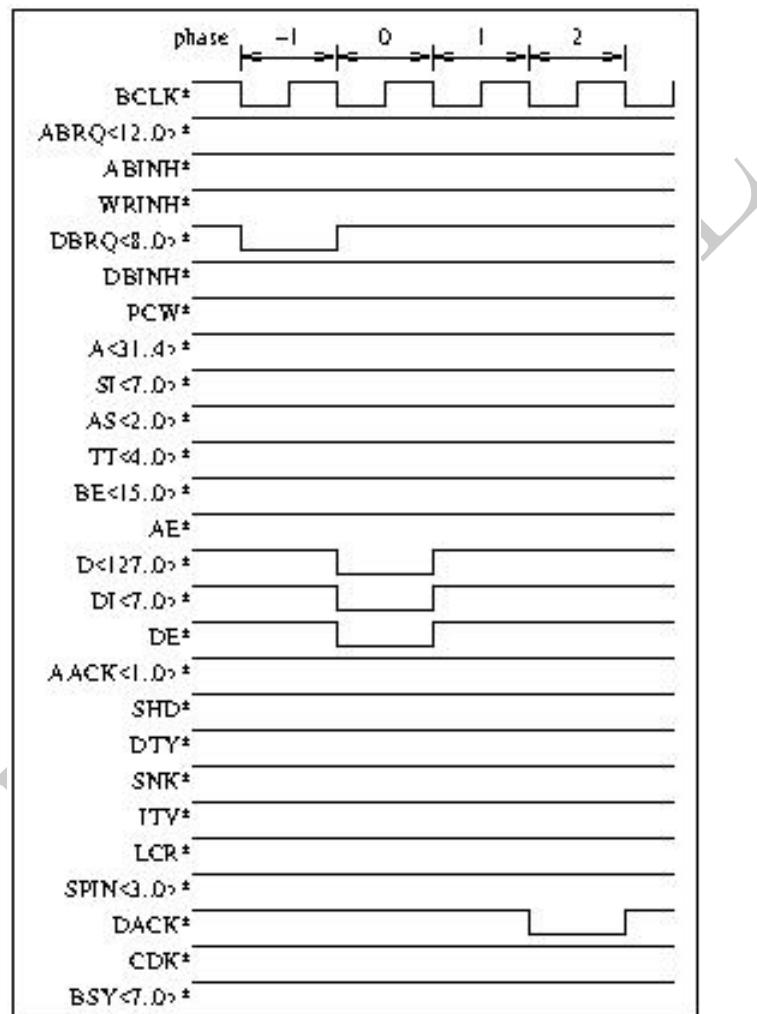


Figure 3.2: 데이터 기본 사이클

- 데이터 기본 주기 - 단계 1 (phase 1)

단계 0에서 데이터 버스를 구동했던 RP는 다음 동작을 준비한다. 단계 0에서 전송된 데이터 등의 정보를 받은 RQ는 내부적으로 단계 2를 위한 동작이 수행된다. 우선 RQ들은 내부적으로 전송된 정보들의 패리티 에러의 확인을 한다. 에러가 발견되지 않으면, 종착점 신호를 자신의 슬롯 어드레스와 비교하여 전송된 데이터가 자신이 요청한 것인지를 확인하게 된다. 자신의 데이터가 아니면 데이터의 저장을 무효화한다.

- 데이터 기본 주기 - 단계 2 (phase 2)

단계 1에서 전송된 정보 중에 패리티 에러가 발견되면 DACK*를 구동하지 않고, 패리티 에러가 없으면 DACK*를 구동하여 RP로부터 전송된 데이터를 RQ가 잘 받았음을 알린다. 특히 해당 데이터에서 패리티 에러가 발견된 경우 RQ는 그 데이터 전송을 있게 한 어드레스 전송까지 무효화하고, 어드레스 기본 주기부터 재시도한다.

3.3.1.3 블록 전송 데이터 기본 사이클

이미 수행된 어드레스 기본 사이클에서 데이터 요청이 있는 경우 RP가 해당 어드레스의 데이터를 RQ로 전달할 때 사용하며 반드시 데이터 중재 과정을 거친 후에 수행된다. 데이터 중재를 수행하여 데이터 버스 사용권을 획득한 바로 다음 버스 클럭 주기에 데이터 기본 사이클이 수행된다.

- 블록 전송 데이터 버스 중재 주기 - 단계 -1 (phase -1)

데이터 버스에 해당 데이터를 구동하기에 앞서 수행하는 데이터 버스 중재 동작.

블록 전송 어드레스 데이터 기본 주기의 데이터 구동과 충돌을 방지하기 위해 WRINH* 신호를 구동한다.

- 블록 전송 데이터 기본 주기 - 단계 0 (phase 0)

데이터 중재 버스를 통하여 데이터 전송 버스의 사용 허가를 받은 RP가 데이터 전송 버스의 신호들을 구동한다. 구동되는 버스의 신호는 D<127..0>*, DP<15..0>*, DI<7..0>*, DIP*, DE* 등이다. 데이터 신호는 어드레스 기본 주기에 의해 전송된 영역에 해당되는 데이터가 구동되고, 종착점 신호(DI<7..0>*)는 전송하는 데이터를 요청했던 RQ의 시발점 신호(SI<7..0>*)를 구동한다. 어드레스 주기를 통하여 RP로 데이터 요청을 RQ는 데이터 버스의 신호를 저장한다.

블록 전송 어드레스 데이터 기본 주기의 데이터 구동에 충돌을 방지하기 위해 WRINH* 신호를 구동한다. 다른 블록 전송 데이터 기본 주기의 데이터 구동과 충돌을 방지하기 위해 DBINH* 신호를 구동한다.

- 블록 전송 데이터 기본 주기 - 단계 1 (phase 1)

데이터 전송 버스의 신호들을 구동한다. 구동되는 버스의 신호는 D<127..0>*, DP<15..0>*, DI<7..0>*, DIP*, DE* 등이다. 데이터 신호는 어드레스 기본 주기에 의해 전송된 영역에 해당되는 데이터가 구동되고, 종착점 신호(DI<7..0>*)는 전송하는 데이터를 요청했던 RQ의 시발점 신호(SI<7..0>*)를 구동한다. 어드레스 주기를 통하여 RP로 데이터를 요청한 RQ는 데이터 버스의 신호를 저장한다.

단계 0에서 데이터 버스를 구동했던 RP는 다음 동작을 준비한다. 단계 0에서 전송된 데이터 등의 정보를 받은 RP들은 내부적으로 단계 2를 위한 동작을 수행한다. 우선 RQ

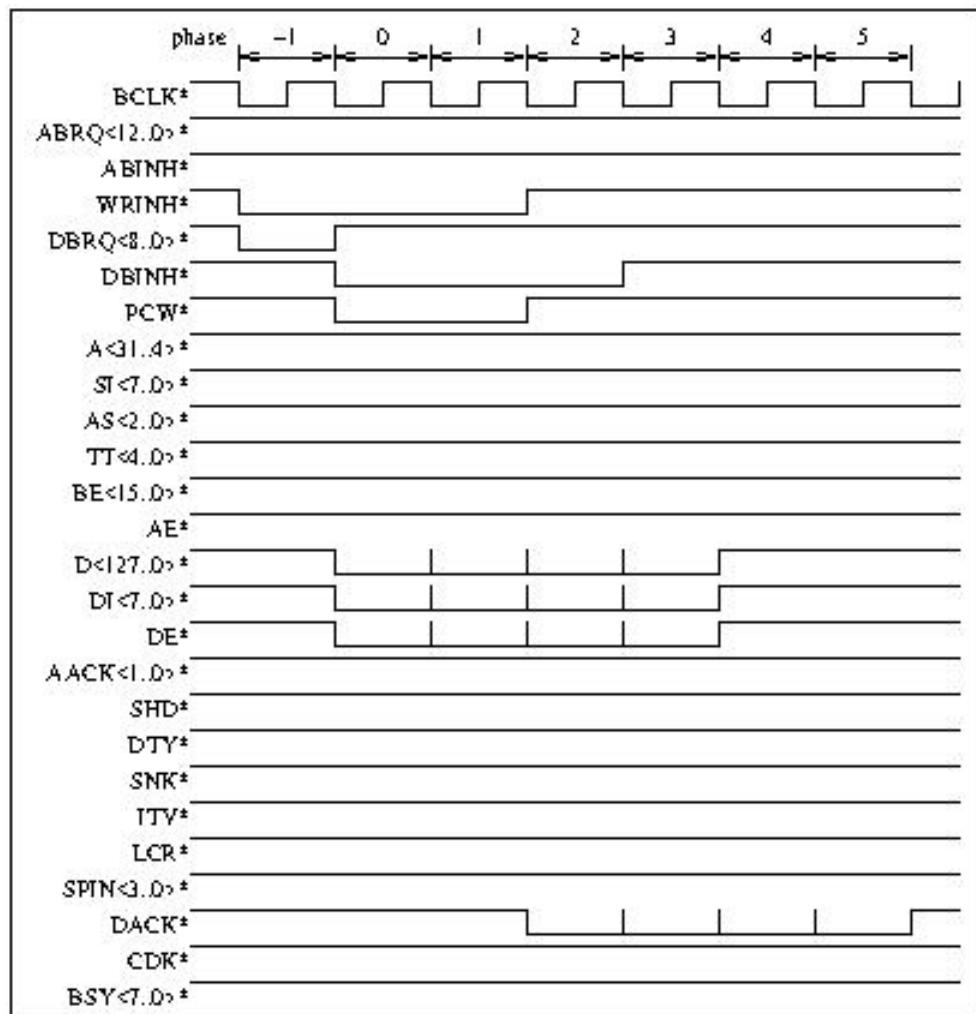


Figure 3.3: 블록 전송 데이터 기본 사이클

들은 내부적으로 전송된 정보들의 패리티 에러의 유무를 확인한다. 에러가 발견되지 않으면, 종착점 신호를 자신의 슬롯 어드레스와 비교하여 전송된 데이터가 자신이 요청한 것인지를 확인하게 된다. 자신의 데이터가 아니면 데이터의 저장을 무효화한다. 블록 전송 어드레스 데이터 기본 주기의 데이터 구동에 충돌을 방지하기 위해 WRINH* 신호를 구동한다. 다른 블록 전송 데이터 기본 주기의 데이터 구동과 충돌을 방지하기 위해 DBINH* 신호를 구동한다.

- **블록 전송 데이터 기본 주기 - 단계 2 (phase 2)**

데이터 전송 버스의 신호들을 구동한다. 구동되는 버스의 신호는 D<127..0>*, DP<15..0>*, DI<7..0>*, DIP*, DE* 등이다. 데이터 신호는 어드레스 기본 주기에 의해 전송된 영역에 해당되는 데이터가 구동되고, 종착점 신호(DI<7..0>*)는 전송하는 데이터를 요청했던 RQ의 시발점 신호(SI<7..0>*)를 구동한다. 어드레스 주기를 통하여 RP로 데이터를 요청한 RQ는 데이터 버스의 신호를 저장한다.

단계 0에서 전송된 정보 중에 패리티 에러가 발견되면 DACK*를 구동하지 않고, 패리티 에러가 없으면 DACK*를 구동하여 RP로부터 전송된 데이터를 RQ가 잘 받았음을 알린다. 특히 해당 데이터에서 패리티 에러가 발견된 경우 RQ는 어드레스 기본 주기 부터 재시도 한다.

다른 블록 전송 데이터 기본 주기의 데이터 구동과 충돌을 방지하기 위해 DBINH* 신호를 구동한다.

- **블록 전송 데이터 기본 주기 - 단계 3 (phase 3)**

데이터 전송 버스의 신호들을 구동한다. 구동되는 버스의 신호는 D<127..0>*, DP<15..0>*, DI<7..0>*, DIP*, DE* 등이다. 데이터 신호는 어드레스 기본 주기에 의해 전송된 영역에 해당되는 데이터가 구동되고, 종착점 신호(DI<7..0>*)는 전송하는 데이터를 요청했던 RQ의 시발점 신호(SI<7..0>*)를 구동한다. 어드레스 주기를 통하여 RP로 데이터를 요청한 RQ는 데이터 버스의 신호를 저장한다.

단계 1에서 전송된 정보 중에 패리티 에러가 발견되면 DACK*를 구동하지 않고, 패리티 에러가 없으면 DACK*를 구동하여 RP로부터 전송된 데이터를 RQ가 잘 받았음을 알린다. 특히 해당 데이터에서 패리티 에러가 발견된 경우 RQ는 그 데이터 전송을 있게 한 어드레스 전송까지 무효화하고, 어드레스 기본 주기부터 재시도한다.

- **블록 전송 데이터 기본 주기 - 단계 4 (phase 4)**

단계 2에서 전송된 정보 중에 패리티 에러가 발견되면 DACK*를 구동하지 않고, 패리티 에러가 없으면 DACK*를 구동하여 RP로부터 전송된 데이터를 RQ가 잘 받았음을 알린다. 특히 해당 데이터에서 패리티 에러가 발견된 경우 RQ는 어드레스 기본 주기 부터 재시도한다.

- **블록 전송 데이터 기본 주기 - 단계 5 (phase 5)**

단계 3에서 전송된 정보 중에 패리티 에러가 발견되면 DACK*를 구동하지 않고, 패리티 에러가 없으면 DACK*를 구동하여 RP로부터 전송된 데이터를 RQ가 잘 받았음을 알린다. 특히 해당 데이터에서 패리티 에러가 발견된 경우 RQ는 어드레스 기본 주기 부터 재시도한다.

3.3.1.4 단일 전송 어드레스-데이터 기본 사이클

어드레스 정보(어드레스와 이에 관련된 정보들)와 데이터 정보(데이터와 이에 관련된 정보들)를 동시에 전달할 때 사용하는 기본 사이클이며 반드시 어드레스 중재 과정을 거친 후에 수행되어어야 한다. 이 기본 사이클은 4개의 단계로 구성되며 각 단계는 연속적이다. 어드레스 중재를 수행하여 어드레스 버스 사용권을 획득한 바로 다음 버스 클럭 주기에 어드레스-데이터 기본 사이클의 단계 0(phase 0)가 시작된다. RQ는 단계 0에서 어드레스 정보를 어드레스 버스에 구동한다. 단계 1에서는 각 RP들이 단계 0에 전달된 어드레스 정보를 번역하여 필요한 동작을 준비한다. 이때 RQ는 데이터 정보를 데이터 버스에 구동한다. 단계 2에서는 단계 1에서 결정된 RP가 전달받은 어드레스 정보에 대한 응답(어드레스 응답)을 구동하며, 동시에 단계 1에서 전달된 데이터를 점검한다. 단계 3에서는 RP가 단계 1에서 전달받은 데이터에 대한 데이터 응답을 구동한다. RQ는 단계 2에 전달받은 어드레스 응답과 단계 3에 전달받은 데이터 응답에 의해 수행한 어드레스-데이터 기본 사이클의 완성 여부를 판단한다.

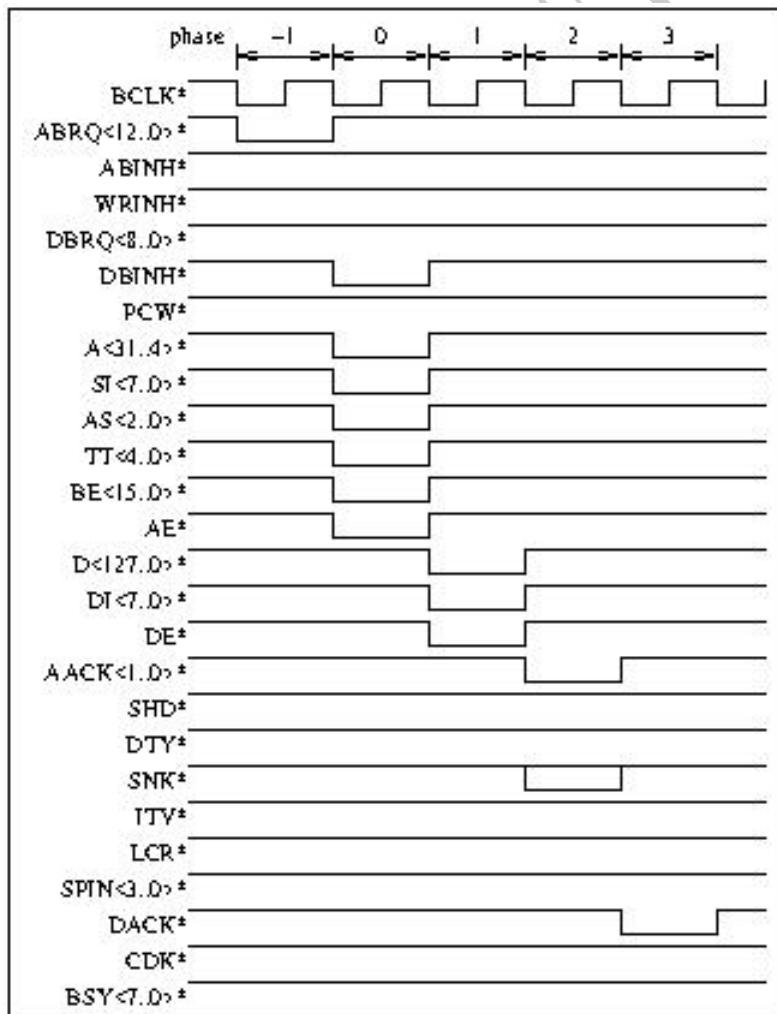


Figure 3.4: 어드레스 데이터 기본 사이클

- 단일 전송 어드레스 버스 중재 주기 - 단계 -1 (phase -1)
어드레스 버스에 해당 어드레스를 구동하기에 앞서 수행하는 어드레스 버스 중재 동작.
- 단일 전송 어드레스 데이터 기본 주기 - 단계 0 (phase 0)
중재 버스를 통하여 버스 사용허가를 받은 RQ가 한 버스 클럭 주기 동안 버스 상에 어드레스와 관련한 정보를 전송한다. 32 비트의 어드레스, 어드레스 영역, 전송 형태, 바이트 마스크, RQ의 슬롯과 채널 어드레스 등과 각 신호의 패리티가 전송된다. 이 주기 동안 사용되는 신호는 A<31..4>*, AP<3..0>*, AS<2..0>*, TT<4..0>*, STP*, BE<15..0>*, BEP<1..0>*, SI<7..0>*, SIP*, AE* 등이다. RP들은 이 단계의 끝 부분에서 어드레스 버스에 실린 신호를 모두 저장한다.
캐쉬를 갖는 RQ들도 이 단계의 끝 부분에서 어드레스 버스에 실린 신호를 모두 저장한다.
데이터 기본주기와의 데이터 충돌을 방지하기 위해 DBINH* 신호를 구동한다.
- 단일 전송 어드레스 데이터 기본 주기 - 단계 1 (phase 1)
단계 0를 구동한 RQ가 계속해서 데이터 버스를 구동한다. 이 단계에서 구동되는 신호 선은 D<127..0>*, DP<15..0>*, DI<7..0>*, DIP*, DE* 등이다. 데이터 기본 주기와는 달리 별도의 데이터 중재 과정이 없다.
RP들은 단계 0에서 저장한 어드레스 버스의 정보들의 에러 확인과 번역을 수행한다.
또한 단계 1의 부분에서는 RQ로부터 오는 데이터를 저장한다.
캐쉬를 갖는 RQ들은 캐쉬 동일성 유지를 위한 동작을 내부적으로 수행한다.
- 단일 전송 어드레스 데이터 기본 주기 - 단계 2 (phase 2)
단계 2에서는 어드레스에 의해 선정된 한 RP가 단계 0에서 전송된 정보들에 대한 응답을 버스 상에 구동하고 단계 1에서 받은 데이터의 패리티 에러를 확인한다. 이때 사용되는 신호는 AAC<1..0>*이고, 만약 단계 0에서 전송된 정보에 에러가 검출될 경우 신호 구동과 데이터 패리티 확인을 수행하지 않는다.
캐쉬를 갖는 RQ들이 단계 0에서 저장한 어드레스에 대한 캐쉬 동일성 유지 동작을 수행함에 있어서 문제가 발생하면 SNK* 신호를 구동한다. 단계 0를 수행했던 RQ는 어드레스 상태 신호를 저장한다.
- 단일 전송 어드레스 데이터 기본 주기 - 단계 3 (phase 3)
단계 1에서 전송된 데이터를 받은 RP가 RQ로 응답하는 단계이다. 데이터의 에러 유무를 보고한다. 이때 사용하는 신호선은 DACK*이다. 단계 2의 응답에서 어드레스를 정상적으로 접수하지 못했다는 반응이 있었을 경우 이 단계의 응답은 의미가 없다.
RQ는 데이터 응답을 저장한다.

3.3.1.5 블록 전송 어드레스-데이터 기본 사이클

어드레스 정보(어드레스와 이에 관련된 정보들)와 데이터 정보(데이터와 이에 관련된 정보들)를 동시에 전달할 때 사용하는 기본 사이클이며 반드시 어드레스 중재 과정을 거친 후에 수행되어야 한다. 이 기본 사이클은 4개의 단계로 구성되며 각 단계는 연속적이다. 어드레스

중재를 수행하여 어드레스 버스 사용권을 획득한 바로 다음 버스 클럭 주기에 어드레스-데이터 기본 사이클의 단계 0(phase 0)가 시작된다. RQ는 단계 0에서는 어드레스 정보를 어드레스 버스에 구동한다. 단계 1에서는 각 RP들이 단계 0에 전달된 어드레스 정보를 번역하여 필요한 동작을 준비한다. 이때 RQ는 데이터 정보를 데이터 버스에 구동한다. 단계 2에서는 단계 0에서 결정된 RP가 전달받은 어드레스 정보에 대한 응답(어드레스 응답)을 구동하며, 동시에 단계 0에서 전달된 데이터를 점검한다. 단계 3에서는 RP가 단계 1에서 전달받은 데이터에 대한 데이터 응답을 구동한다. RQ는 단계 2에 전달받은 어드레스 응답과 단계 3에 전달받은 데이터 응답에 의해 수행한 어드레스-데이터 기본 사이클의 완성 여부를 판단한다.

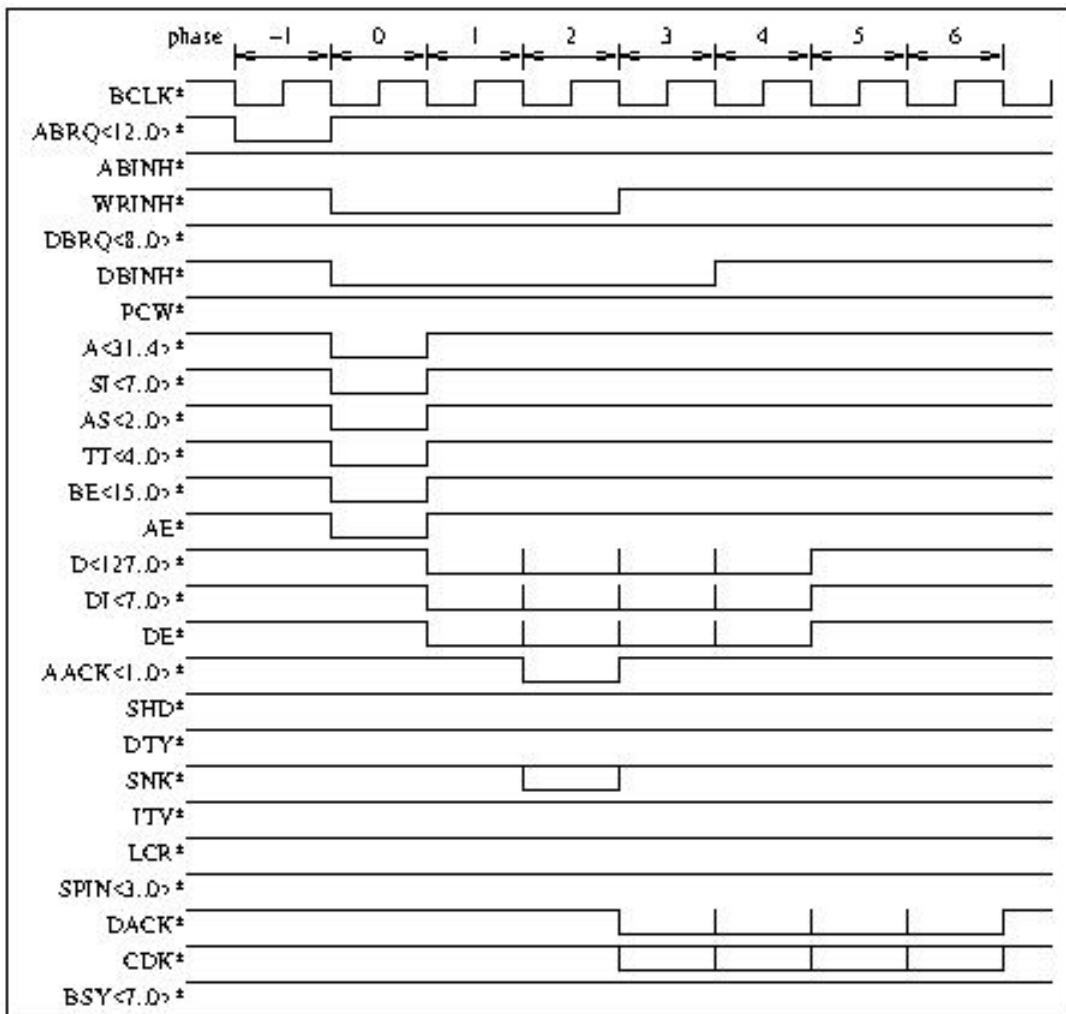


Figure 3.5: 블록 전송 어드레스 데이터 기본 사이클

- 블록 전송 어드레스 버스 중재 주기 - 단계 -1 (phase -1)
어드레스 버스에 해당 어드레스를 구동하기에 앞서 수행하는 어드레스 버스 중재 동작.

- 블록 전송 어드레스 데이터 기본 주기 - 단계 0 (phase 0)

중재 버스를 통하여 버스 사용허가를 받은 RQ가 한 버스 클럭 주기 동안 버스 상에 어드레스와 관련한 정보를 전송한다. 32 비트의 어드레스, 어드레스 영역, 전송 형태, 바이트 마스크, RQ의 슬롯과 채널 어드레스 등과 각 신호의 패리티가 전송된다. 이 주기 동안 사용되는 신호는 A<31..4>*, AP<3..0>*, AS<2..0>*, TT<4..0>*, STP*, BE<15..0>*, BEP<1..0>*, SI<7..0>*, SIP*, AE* 등이다. RP들은 이 단계의 끝 부분에서 어드레스 버스에 실린 신호를 모두 저장한다.

캐쉬를 갖는 RQ들도 이 단계의 끝 부분에서 어드레스 버스에 실린 신호를 모두 저장한다.

다른 블록 전송 쓰기와의 데이터 충돌을 방지하기 위해 WRINH* 신호를 구동한다. 다른 전송의 데이터와 충돌을 방지하기 위해 DBINH* 신호를 구동한다.

- 블록 전송 어드레스 데이터 기본 주기 - 단계 1 (phase 1)

단계 0를 구동한 RQ가 계속해서 데이터 버스를 구동한다. 이 단계에서 구동되는 신호 선은 D<127..0>*, DP<15..0>*, DI<7..0>*, DIP*, DE* 등이다. 데이터 기본 주기와는 달리 별도의 데이터 중재 과정이 없다.

RP들은 단계 0에서 저장한 어드레스 버스의 정보들의 예러 확인과 번역을 수행한다. 또한 단계 1의 부분에서는 RQ로부터 오는 데이터를 저장한다.

캐쉬를 갖는 RQ들은 캐쉬 동일성 유지를 위한 동작을 내부적으로 수행한다.

다른 블록 전송 쓰기와의 데이터 충돌을 방지하기 위해 WRINH* 신호를 구동한다.

다른 전송의 데이터와 충돌을 방지하기 위해 DBINH* 신호를 구동한다.

- 블록 전송 어드레스 데이터 기본 주기 - 단계 2 (phase 2)

단계 2에서는 어드레스에 의해 선정된 한 RP가 단계 0에서 전송된 정보들에 대한 응답을 버스 상에 구동하고 단계 1에서 받은 데이터의 패리티 예러를 확인한다. 이때 사용되는 신호는 AAC<1..0>*이고, 만약 단계 0에서 전송된 정보에 예러가 검출될 경우 신호 구동과 데이터 패리티 확인을 수행하지 않는다.

단계 0를 구동한 RQ가 계속해서 데이터 버스를 구동한다. 이 단계에서 구동되는 신호 선은 D<127..0>*, DP<15..0>*, DI<7..0>*, DIP*, DE* 등이다. 데이터 기본 주기와는 달리 별도의 데이터 중재 과정이 없다.

캐쉬를 갖는 RQ들이 단계 0에서 저장한 어드레스에 대한 캐쉬 동일성 유지 동작을 수행함에 있어서 문제가 발생하면 SNK* 신호를 구동한다. 단계 0를 수행했던 RQ는 어드레스 상태 신호를 저장한다.

다른 블록 전송 어드레스 데이터 기본주기와의 데이터 충돌을 방지하기 위해 WRINH* 신호를 구동한다. 다른 전송의 데이터와 충돌을 방지하기 위해 DBINH* 신호를 구동한다.

- 블록 전송 어드레스 데이터 기본 주기 - 단계 3 (phase 3)

단계 1에서 전송된 데이터를 받은 RP가 RQ로 응답하는 단계이다. 데이터의 예러 유무를 보고한다. 이때 사용하는 신호선은 DACK*이다. 그러나 현재 진행 중인 전송이 ITW인 경우 DACK*는 의미가 없고 대신 CDK*가 사용된다. 단계 2의 응답에서 어드레스를 정상적으로 접수하지 못했다는 반응이 있었을 경우 이 단계의 응답은 의미가 없다.

단계 0를 구동한 RQ가 계속해서 데이터 버스를 구동한다. 이 단계에서 구동되는 신호선은 D<127..0>*, DP<15..0>*, DI<7..0>*, DIP*, DE* 등이다. 데이터 기본 주기와는 달리 별도의 데이터 중재 과정이 없다.

단계 0에 구동된 어드레스에 의해 선택된 RP는 단계 1에 구동된 데이터에 대한 데이터 응답 DACK*을 구동하고, RQ는 해당 데이터 응답을 저장한다. 그러나 현재 진행중인 전송이 ITW인 경우 DACK*는 의미가 없고 대신 CDK*가 사용된다.

다른 전송의 데이터와 충돌을 방지하기 위해 DBINH* 신호를 구동한다.

- 블록 전송 어드레스 데이터 기본 주기 - 단계 4 (phase 4)

단계 2에서 전송된 데이터를 받은 RP가 RQ로 응답하는 단계이다. 데이터의 에러 유무를 보고한다. 이때 사용하는 신호선은 DACK*이다. 그러나 현재 진행 중인 전송이 ITW인 경우 DACK*는 의미가 없고 대신 CDK*가 사용된다. 단계 2의 응답에서 어드레스를 정상적으로 접수하지 못했다는 반응이 있었을 경우 이 단계의 응답은 의미가 없다.

단계 0를 구동한 RQ가 계속해서 데이터 버스를 구동한다. 이 단계에서 구동되는 신호선은 D<127..0>*, DP<15..0>*, DI<7..0>*, DIP*, DE* 등이다. 데이터 기본 주기와는 달리 별도의 데이터 중재 과정이 없다.

단계 0에 구동된 어드레스에 의해 선택된 RP는 단계 2에 구동된 데이터에 대한 데이터 응답을 구동하고, RQ는 해당 데이터 응답을 저장한다.

- 블록 전송 어드레스 데이터 기본 주기 - 단계 5 (phase 5)

단계 3에서 전송된 데이터를 받은 RP가 RQ로 응답하는 단계이다. 데이터의 에러 유무를 보고한다. 이때 사용하는 신호선은 DACK*이다. 그러나 현재 진행 중인 전송이 ITW인 경우 DACK*는 의미가 없고 대신 CDK*가 사용된다. 단계 2의 응답에서 어드레스를 정상적으로 접수하지 못했다는 반응이 있었을 경우 이 단계의 응답은 의미가 없다.

단계 0에 구동된 어드레스에 의해 선택된 RP는 단계 3에 구동된 데이터에 대한 데이터 응답을 구동하고, RQ는 해당 데이터 응답을 저장한다.

- 블록 전송 어드레스 데이터 기본 주기 - 단계 6 (phase 6)

단계 4에서 전송된 데이터를 받은 RP가 RQ로 응답하는 단계이다. 데이터의 에러 유무를 보고한다. 이때 사용하는 신호선은 DACK*이다. 그러나 현재 진행 중인 전송이 ITW인 경우 DACK*는 의미가 없고 대신 CDK*가 사용된다. 단계 2의 응답에서 어드레스를 정상적으로 접수하지 못했다는 반응이 있었을 경우 이 단계의 응답은 의미가 없다.

단계 0에 구동된 어드레스에 의해 선택된 RP는 단계 4에 구동된 데이터에 대한 데이터 응답을 구동하고, RQ는 해당 데이터 응답을 저장한다.

3.3.2 전송의 구분

정의된 전송의 종류는 NRD, NWR, NCR, NCW, IVD, ILR, ILW, BRD, BWR, CRD, EXR, WRB, ITW로 13 가지이지만 실제 데이터가 이동되는 것은 IVD를 제외한 12 가지 전송형태이다. 그리고 데이터 이동이 수반되는 전송형태는 데이터의 실제 이동 방향에 따라 읽기 전송 모임(read transfer class)과 쓰기 전송 모임(write transfer class)으로 구분할 수 있다. 읽기 전송 모임은 데이터가 RP로부터 RQ로 이동하는 경우로써 NRD, NCR, CRD, EXR, BRD, 그리고 ILR가 여기에 속한다. 쓰기 전송 모임은 데이터가 RQ에서 RP로 이동하는 경우로써 NWR, NCW, WRB, BWR, 그리고 ILW가 여기에 속한다. 특히 ITW의 경우는 쓰기 전송과 비슷하지만 데이터는 RQ에서 RQ로 이동한다. 그리고 이동하는 데이터의 크기에 따라 단일 전송과 블록 전송으로 구분되며, 단일 전송의 경우 16바이트 이내의 연속된 데이터 전송에 사용되고, 블록 전송의 경우 64바이트 데이터 전송에 사용된다. 단일 전송에는 NRD, NWR, NCR, NCW, ILR, ILW가 있고, 블록 전송에는 BRD, BWR, CRD, EXR, WRB, ITW가 있다.

3.3.2.1 단일 읽기 전송 모임

단일 전송 읽기 전송은 단일 전송 어드레스 기본 사이클과 단일 전송 데이터 기본 사이클로 구성된다. 데이터가 필요한 RQ가 어드레스 기본 사이클을 수행하고, 어드레스 기본 사이클에 구동된 어드레스에 의해 선택된 RP는 단일 전송 데이터 기본 사이클을 수행하여 요청된 데이터를 전송하게 된다. <그림 3.6>는 읽기 전송 모임의 진행 순서를 나타낸다.

3.3.2.2 단일 쓰기 전송 모임

단일 전송 쓰기 전송은 단일 전송 어드레스 데이터 기본 사이클로 구성된다. 특정 RQ가 특정 RP로 데이터를 공급하는 전송이다. <그림 3.7>는 쓰기 전송 모임의 진행 순서를 나타낸다.

3.3.2.3 블록 읽기 전송 모임

블록 전송 읽기 전송은 블록 전송 어드레스 기본 사이클과 블록 전송 데이터 기본 사이클로 구성된다. 데이터가 필요한 RQ가 어드레스 기본 사이클을 수행하고, 어드레스 기본 사이클에 구동된 어드레스에 의해 선택된 RP는 블록 전송 데이터 기본 사이클을 수행하여 요청된 데이터를 전송하게 된다.

3.3.2.4 블록 쓰기 전송 모임

블록 전송 쓰기 전송은 블록 전송 어드레스 데이터 기본 사이클로 구성된다. 특정 RQ에서 특정 RP로, 혹은 특정 RQ에서 다른 특정 RQ로 데이터를 공급하는 전송이다.

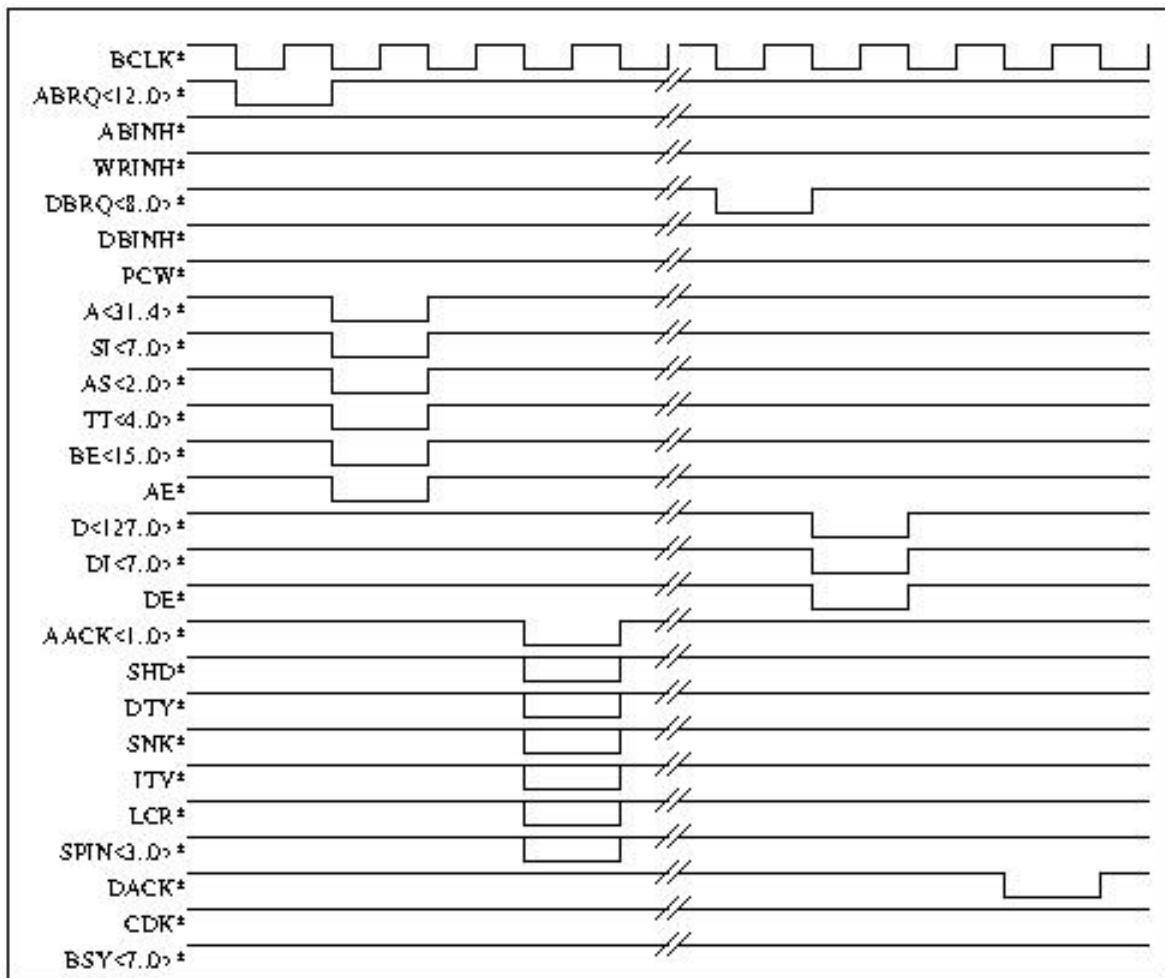


Figure 3.6: 단일 읽기 전송

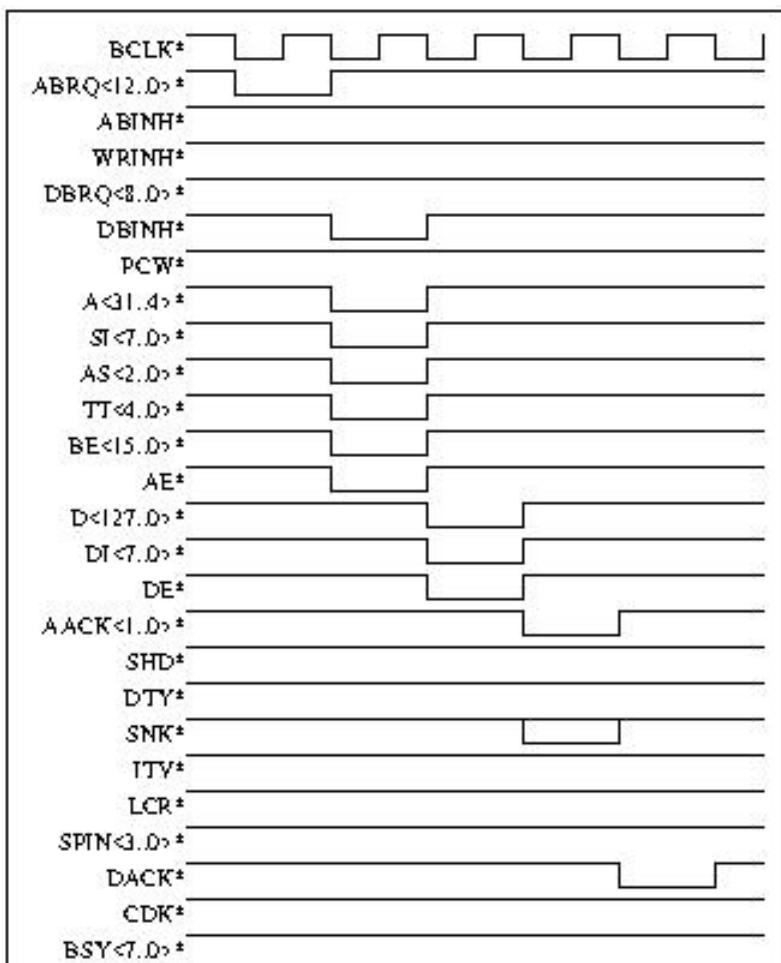


Figure 3.7: 단일 쓰기 전송

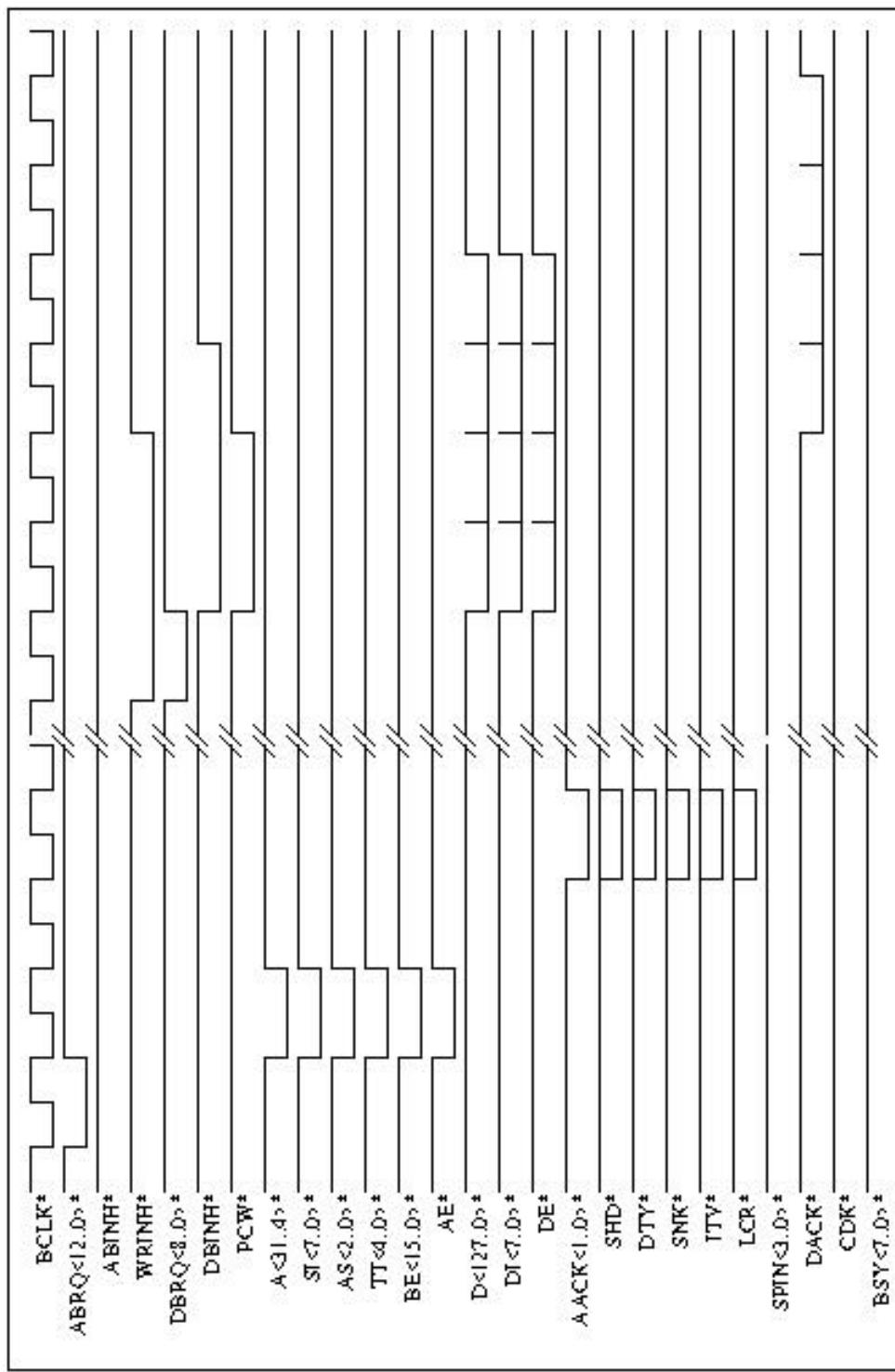


Figure 3.8: 블록 읽기 전송

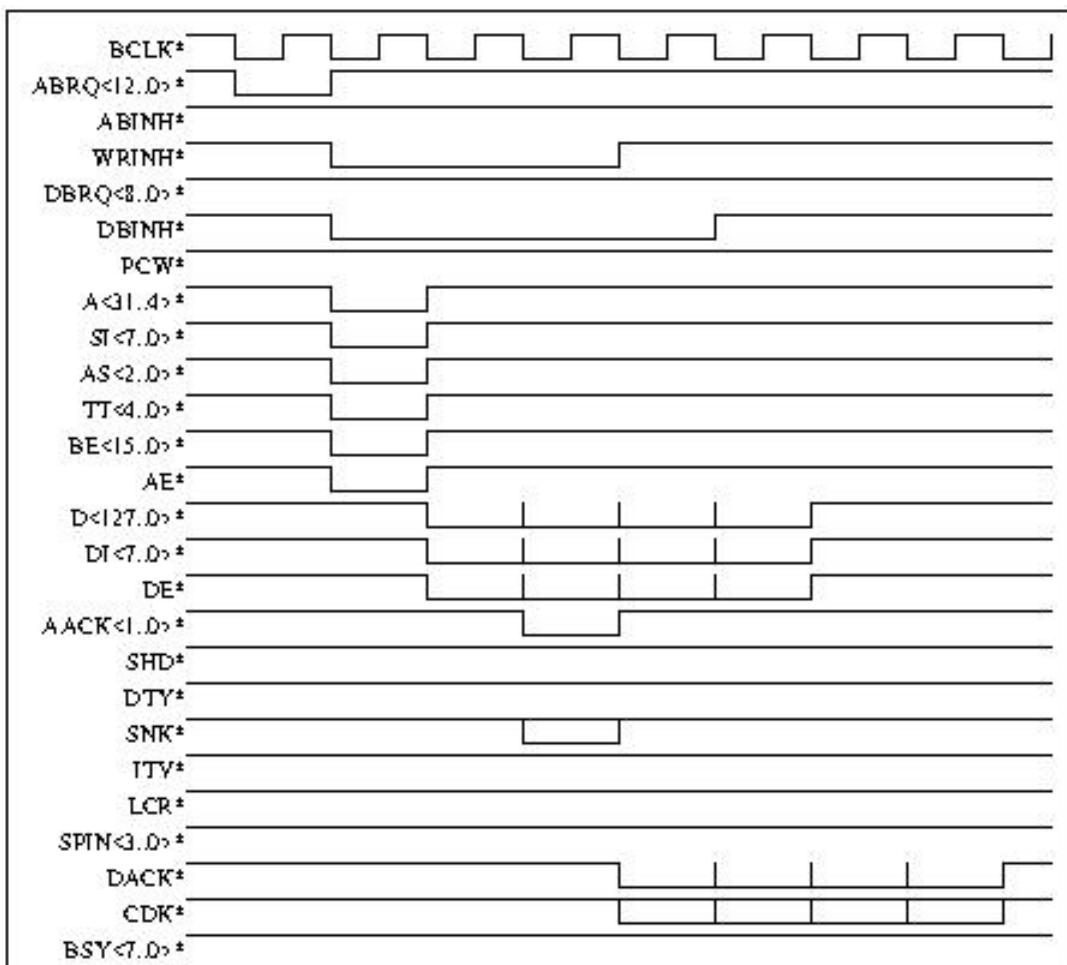


Figure 3.9: 블록 쓰기 전송

3.4 예외 처리 (Exception Handling)

예외 처리란 지금까지 정의한 동작 규정 중에서 정상적으로 동작이 완료되지 않고 에러가 발생했거나 특별한 상황이 발생했을 때, 이의 처리를 말한다.

3.4.1 패리티 에러 (Parity Error)

패리티는 데이터 전송 버스의 신호 전송에 있어서 에러의 발생을 검출하기 위하여 사용한다. 패리티 에러의 검출은 기본 주기 단위로 이루어지고, 패리티 에러의 처리는 전송 형태 단위의 재시도 (Retry)를 원칙으로 한다. 즉 데이터 전송 중 패리티 에러는 기본 주기의 끝 단계에서 상태 버스를 통하여 보고되거나 자체적으로 검출되고, 이의 처리는 그 기본 주기를 포함하는 전송 형태 단위로 재시도한다.

데이터 전송 중의 에러의 발생은 전송 형태 별로 약간씩 처리 방법이 다를 수 있는데 에러 처리의 관점에서 볼 때 기본 주기로 나눌 수 있다.

3.4.1.1 어드레스 기본 주기에서 패리티 에러가 검출될 경우

RQ는 상태 버스를 통하여 RQ로 에러의 검출을 알리고 ($AACK<1..0>^* = 0$) 받은 어드레스 버스의 정보를 무시한다. 어드레스에 패리티 에러가 검출되어 어떤 RP가 선택되었는지를 알 수 없을 경우는 응답하지 않는다. 즉 $AACK<1..0>^*$ 를 구동하지 않는다. (응답 신호를 구동하지 않으면 버스 상의 신호는 negative logic이기 때문에 RQ는 $AACK<1..0>^* = 0$ 인 것으로 받게 된다.) RQ는 에러의 검출을 받으면 동일한 어드레스 기본 주기를 다시 시도한다 (재시도). 만약 연속하여 두번 에러 응답을 받을 경우는 해당 프로세서로 데이터 전송 버스의 에러를 알린다. 그 후의 처리는 프로세서의 버스 에러 루틴의 기능에 따른다.

3.4.1.2 데이터 기본 주기에서 패리티 에러가 검출될 경우

RQ는 RP로 부터 받은 데이터에서 패리티를 확인하여 에러 여부를 알 수 있다. 패리티 에러가 발생하면, 받은 데이터를 무시하고 어드레스 주기 부터 다시 수행한다. 따라서 이 경우는 RP는 데이터 전송 중에 발생한 에러 처리에 대하여 관여되지 않는다. 만약 재시도한 주기에서 또다시 에러가 발생할 경우는 해당 프로세서로 데이터 전송 버스의 에러를 알린다. 그 후의 처리는 프로세서의 버스 에러 루틴의 기능에 따른다.

블록 전송의 경우 전송되는 특성 데이터에 오류가 발생해도 일단 데이터 기본 주기를 끝까지 계속하고 다시 어드레스 주기 부터 재시도한다.

3.4.1.3 어드레스 데이터 주기에서 패리티 에러가 검출될 경우

RQ는 RP로 부터 오는 상태 버스에 의해 어드레스 버스와 데이터 버스의 전송 정보에 대한 에러 보고를 받는다 ($AACK<1..0>^* = 0$, $DACK^* = 0$). RP가 에러를 검출하면 오류를 RQ로 보내고 어드레스 데이터 주기를 끝까지 수행한다. RQ는 에러를 보고 받으면, 수행 중인 어드레스 데이터 기본 주기의 수행을 끝까지 마치고 처음부터 재시도를 한다. 만약 재시도한 주기에서 또다시 에러가 발생할 경우는 해당 프로세서로 데이터 전송 버스의 에러를 알린다. 그 후의 처리는 프로세서의 버스 에러 루틴의 기능에 따른다.

3.4.2 Timeout Error

Timeout 에러는 어드레스 주기의 전송을 마치고 데이터를 기다리는 RQ가 일정한 시간이 지난 후에도 데이터가 오지 않을 경우 무한히 기다리는 것을 방지하기 위한 에러이다. 어드레스 버스의 정보를 받은 RP가 해당 어드레스의 데이터를 접근하다가 문제가 발생하여 데이터를 응답하지 않을 경우 발생된다. RQ가 어드레스 주기를 수행시키고 난 후 기다리는 시간은 조정이 가능하도록 구현하며, 최소한 데이터 접근 시간, 데이터 버스 사용상 지연 시간 등을 더한 것 보다 커야한다.

에러의 처리는 어드레스 주기 부터 재시도를 하며, 똑같은 에러가 발생될 경우는 패리티 에러의 경우와 같다.

3.4.3 Busy 응답

Busy 응답이란 어드레스 전송 주기에서 전송된 어드레스 버스의 정보를 처리할 수 없을 경우 발생한다 ($AACK<1..0>^* = 1$). 예를 들면, 메모리 제어기가 이전에 전송되어 온 요청을 처리하기 위하여 새로 도착한 요청을 처리할 수 없을 경우를 말한다. 따라서 Busy 응답은 에러의 발생을 말하는 것이 아니며, 어드레스 전송은 제대로 되었으나 RP의 상태가 지금 당장 이를 처리하지 못한다는 것을 의미한다.

RQ는 이와같은 응답을 받았을 경우 중재 주기에서 공정성 규칙을 무시하고 중재 부터 재시도한다. 그러나 경우에 따라 오랜 동안 Busy 응답만을 받고 메모리 보드 참조에 실패할 경우 ABINH* 신호를 이용할 수 있다. 그래도 계속해서 같은 응답을 받을 경우는 RP에 이상이 발생한 것으로 간주하여 프로세서로 RP의 에러를 알린다. 그 후의 처리는 프로세서의 RP 에러 루틴의 기능에 따른다.

3.4.4 기타 에러

기타 에러는 전송 규격에 정의되지 않은 신호의 조합을 발견했을 때와 그밖의 에러를 말한다. 모든 에러는 일단 순간적인 외부의 잡음으로 간주하고 재시도를 한다. 연속한 에러가 검출될 경우는 프로세서로 에러의 보고를 한다.

ETRI-CSTL

Chapter 4

인터럽트 버스

4.1 개요

인터럽트 버스는 버스 모듈 사이의 비동기적인 신호 교환(asynchronous communication)의 통로이며 프로세서 사이의 통신(interprocessor communication)을 지원한다.

Table 4.1: 인터럽트 버스의 규격 요약

인터럽트 전송 프로토콜 (Interrupt Transfer Protocols)	
Protocol	Message passing protocols
제어 방식	동기형 (Synchronous)
클럭의 속도	16.5 MHz (60.6 nsec)
버스 사용을 위한 중재 (Arbitration)	
중재 규칙	Priority
중재 수행 시간	5 Bus Clock (303.0 nsec)
중재 기법	Coded Self Arbitration Scheme
중재기의 특성	분산형 중재기
	성능 (Performance)
전송속도	약 1.6 MDI/sec (Mega Direct Interrupts per second) 약 0.6 MAI/sec (Mega Arbit. Interrupts per second)
에러 방어 (Error Protection)	
에러 검출 (Error Detection)	바이트 단위의 홀수 패리티(odd parity)
에러의 처리 (Error Handling)	재시도 (Retry)
기 타 (Etc.)	
총 신호수	10

4.2 인터럽트 버스의 신호선

Table 4.2: 인터럽트 버스의 신호들

Mnemonic	Size	Name
IBSYNC*	1	Interrupt Bus Sync.
IBD<7..0>*	8	Interrupt Bus Data
IBDP*	1	Interrupt Bus Data Parity

4.2.1 Interrupt Bus Sync : IBSYNC*

본 버스의 모든 동작은 시분할 방식으로 진행되기 때문에 필요한 신호이다. 이 신호는 인터럽트를 전송하고 있는 모듈이 구동하는 신호로써, 이제 인터럽트 전송이 끝나므로 다음 인터럽트 전송을 원하는 모듈은 중재를 준비하라는 의미를 갖는 신호이다. 인터럽트를 전송하고 있는 모듈은 이 신호를 계속 “참 (낮은 전압)”으로 구동하며, 인터럽트 전송 마지막 주기에서 이 신호를 “거짓 (높은 전압)”으로 한다. 인터럽트를 전송하고자 기다리고 있는 모듈들은 이 신호를 관찰하여 “거짓”이 되면 중재를 수행하게 된다. 중재를 수행하는 모듈도 이 신호를 “참”으로 구동한다. 중재 동작 후 버스 사용허가를 받은 모듈은 자신이 인터럽트 전송을 마칠 때까지 “참”으로 구동하게 되고, 나머지 모듈은 구동을 멈추고 전송이 끝나서 이 신호가 “거짓”이 될 때까지 기다리게 된다. 따라서 인터럽트를 전송하고자 하는 모듈이 없을 때는 이 신호의 상태는 “거짓”이 된다.

4.2.2 Interrupt Bus Data : IBD<7..0>*

이 신호는 크게 두가지 목적을 갖는데, 첫째는 인터럽트에 관한 데이터 전송 통로이고, 둘째는 버스 중재를 위한 제어 신호이다. 따라서 이 신호는 인터럽트 전송 주기에 따라 다른 방법으로 동작한다. 이 신호는 인터럽트 버스 상의 모든 모듈이 입출력으로 사용할 수 있고, 동시에 두개 이상의 모듈이 구동할 수 있으며 중재 기능을 구현할 수 있도록 wired-OR 신호선을 사용한다.

데이터 전송 통로로 이용될 경우는 동기형 프로토콜의 일반적인 데이터 전송 방법을 사용한다. 인터럽트 전송 주기에 따라 전송 방향이 결정되고, 데이터를 전송하는 모듈은 한개가 되며, 한개 이상의 모듈로 입력이 될 수 있다. 인터럽트의 종류 (Class), 요청기의 주소 (Source Address), 벡터 (Vector), 응답 (Acknowledge) 등이 전송된다.

인터럽트 버스의 중재 동작 중에는 요청기들 사이의 중재와 처리기 사이의 중재가 있다. 요청기 사이의 중재는 같은 주기에 두개 이상의 요청기가 인터럽트를 전송하고자 할 경우 인터럽트 전송의 시작 전에 이루어지며, 처리기 사이의 중재는 인터럽트가 여러개의 처리기로 보내지는 중재 인터럽트 주기 중에 인터럽트 처리를 맡을 한 처리기를 선정하는 과정에 수행된다. 중재 동작에서 이 신호들로 출력되는 정보는 인터럽트의 종류와 슬롯 번호등이 되며, 각 신호는 중재 버스와 같이 각 비트 별로 우선 순위 비교를 위해 사용된다.

4.2.3 Interrupt Bus Data Parity : IBDP*

인터럽트 버스의 신뢰도 향상을 위하여 필요한 신호로써 IBD<7..0>*의 패리티로 동작된다. 중재동작 때는 패리티 사용하지 않고, 데이터 전송동작 때는 해당 데이터의 패리티로 동작한다.

4.3 인터럽트의 종류

인터럽트의 종류(interrupt class)는 인터럽트의 전송 방법에 따라 구분된다. 인터럽트의 종류는 인터럽트 처리기를 요청기가 선정하여 인터럽트를 전송하는 지정 인터럽트(direct interrupt)와 인터럽트 처리기를 요청기들 사이의 중재를 통하여 선정되도록 하는 중재 인터럽트(arbitration interrupt)로 크게 분류할 수 있다. 중재 인터럽트는 전송 정보의 양에 따라 중재 인터럽트 1, 중재 인터럽트 0로 구분된다. 3 종류의 인터럽트는 인터럽트 전송 순서에 있어서 우선순위가 있으며, 지정 인터럽트가 높은 우선순위를 가지며, 중재 인터럽트 1, 그리고 중재 인터럽트 0의 순서로 우선순위가 낮아진다. 인터럽트 종류는 인터럽트 버스 중재(interrupt bus arbitration)와 인터럽트 전송의 첫번째 단계(phase 0)에 Class 필드에 의해 정의된다. Class 필드의 값은 <표 4.3>와 같다.

Table 4.3: 인터럽트 종류들

interrupt class	interrupt name	num. of phases	priority
3	direct interrupt	5	high
2	arbitration interrupt 1	22	
1	arbitration interrupt 0	reserved	low
0	no interrupt	0	

4.3.1 지정 인터럽트

지정 인터럽트란 인터럽트 요청기가 인터럽트를 받을 처리기의 주소를 지정하여 인터럽트를 전송하는 것을 말한다. 일반적으로 지정 인터럽트는 시스템의 제어(시험, 고장 진단 등)와 그리고 실시간 응용등에 사용될 수 있다. 전송 순서에 있어서 가장 높은 우선순위를 갖는다. 4 가지 벡터 형태(Q-, N-, I-, E-type)를 모두 전송할 수 있다.

처리기를 지정하는데 있어서 두 가지의 방법이 있는데, 하나의 인터럽트 처리기만을 지정하는 경우와 동시에 시스템 전체에 있는 모든 처리기를 지정하는 경우(broadcast)가 있다. 전자의 경우는 각각 처리기에 한정된 정보의 전달을 위해 사용될 수 있고, 후자는 시스템 전체에 위급한 상황등의 정보 전달을 위해 사용할 수 있다. 지정 인터럽트의 경우는 여러 개의 처리기에 인터럽트가 전달되어도 중재 인터럽트와 같은 처리기 사이의 중재 활동이 필요 없다.

지정 인터럽트는 5 개의 단계로 구성이 되고 각 단계는 한개의 인터럽트 버스 기본 주기가 소요된다.

4.3.2 중재 인터럽트 1

중재 인터럽트 1(arbitration interrupt 1)은 다중 프로세서 환경의 효과적인 인터럽트 분배를 구현하기 위해 필요한 인터럽트 전송 방법의 한 형태이다. 이 방법은 인터럽트의 동적인 분배를 실현할 수 있으며, 분배 과정에서 프로세서의 불필요한 Context Switching를 막을 수 있도록 구현되어 있다. 다중 프로세서 운영 체제 환경에서 입출력의 제어와 프로세서 사이의

대화 수단 등으로 사용할 수 있다.

중재 인터럽트 1은 지정 인터럽트와는 달리 처리기를 지정하지 않고(프로세서의 모임만 지정), 한개 이상의 처리기에 인터럽트를 동시에 전달한 후, 처리기들끼리 중재를 하여 인터럽트를 접수할 처리기를 하나만 선정하도록 한다. 중재 과정에서 처리기들은 현재 자신의 상태(상위 프로세서 등의 상태 포함)에 따라 우선 순위를 결정하게 되는, 그 순간에 인터럽트를 처리하기에 가장 적절한 것이 중재의 최종 승자가 되도록 구현되어 있다.

중재 인터럽터 1은 22 개의 단계로 구성되어 있고 각 단계는 한개의 인터럽트 버스 기본 주기를 사용한다.

4.3.3 중재 인터럽트 0

중재 인터럽트 0(arbitration interrupt 0)은 인터럽트 버스의 기능성 확장을 위하여 마련된 것으로 현재 규격은 유보한다.

4.4 벡터의 종류

인터럽트 벡터는 전송된 인터럽트의 중심이 되는 정보로써 8 비트와 16 비트 크기가 있다. 인터럽트 벡터가 담고 있는 의미는 인터럽트 발생 원인과 요청하는 동작이 무엇인지를 나타내게 된다. 따라서 각 벡터는 전 시스템에서 유일한 의미를 갖게 된다. 그와 같은 벡터는 그들이 갖고 있는 특성에 따라 4 가지로 분류되는데, 전송된 정보 중에 2 비트의 Type 필드에 의해 지정된다. <표 4.4>은 각 형태의 벡터의 특성을 기술하고 있다. Type 필드의 값이

Table 4.4: 벡터의 종류

vector type	name	vector size	description	priority	mask	class
3	E-type	8bit	emergency int.	high	non-maskable	DI
2	I-type	8bit	immediate funciton		no pending maskable	DI
1	N-type	16bit	notify int.		no pending maskable	DI or AI
0	Q-type	16bit	queue int.	low	pending maskable	DI or AI

클수록 처리의 우선 순위가 높다.

4.4.1 E(Emergency)-Type

긴급한 상황이 발생했을 경우 사용하는 인터럽트이다. 지정 인터럽트에서만 사용할 수 있으므로 8 비트의 벡터를 갖는다. 마스크시킬 수 없으며, 해당 프로세서로 가장 높은 순위의 인터럽트로 전달해야 한다.

4.4.2 I(Immediate)-Type

이 형태의 벡터는 인터럽트라기 보다는 인터럽트 버스를 통하여 단순한 기능을 구현하기 위한 목적으로 사용한다. 지정 인터럽트에서만 사용할 수 있으므로 8 비트의 벡터를 갖는다. 대부분 처리기가 독자적으로 수행할 수 있는 단순한 기능을 구현하기 때문에 해당 프로세서로 인터럽트를 전달하지 않을 수 있다. 마스크시킬 수 있으며, 마스크되어 있는 상태에서 인터럽트가 전송될 경우 그 인터럽트는 무시된다. 시스템의 시험이나 고장 진단의 목적으로 이용될 수 있다.

4.4.3 N(Notify)-Type

Q-Type과 함께 다중 처리 운영 체제가 동작 중에 주로 사용될 수 있는 벡터들이다. 지정 인터럽트와 중재 인터럽트를 통하여 사용할 수 있다. 따라서 16 비트의 벡터 크기를 갖는다.

지정 인터럽트와 중재 인터럽트가 서로 벡터의 길이가 다르기 때문에 지정 인터럽트로 이 벡터를 전송할 경우는 낮은 자리의 한 바이트가 생략된 것으로 처리된다. 이 형태 특징은 우선 순위에 있어서 Q-Type보다 높고 인터럽트가 마스크될 경우 접수되지 않고 (No Pending) 요청기로 접수되지 않음을 알린다. 처리가 지연된 상태에서 새로운 인터럽트가 전송되면 이전에 도착한 인터럽트는 무시되며, 이 경우 예외로 처리되지 않는다. 즉, 인터럽트가 처리되지 않은 상태에서 다시 이 형태의 인터럽트가 전달되면 그전에 전달되었던 인터럽트는 무시되고 새로운 인터럽트를 처리하게 된다.

4.4.4 Q(Queue)-Type

N-Type보다 낮은 우선 순위를 갖고 한번 전송된 인터럽트는 반드시 처리되어야 한다. 따라서 인터럽트가 마스크될 경우는 처리가 지연(Pending)된다. 지연된 상태에서 새로운 인터럽트가 전달되면 처리기의 저장 능력에 따라 Queue에 쌓아두거나, Queue가 부족할 경우는 요청기로 전달된 인터럽트를 받을 수 없음을 알린다 N-Type과 같이 운영 체제에서 16 비트의 벡터를 정의하여 사용할 수 있다. 지정 인터럽트로 전달할 경우는 낮은 자리의 8 비트가 생략된 것으로 가정한다.

4.5 인터럽트 버스 중재

인터럽트 버스 중재(interrupt bus arbitration)는 인터럽트의 전송을 원하는 인터럽트 요청기들이 동시에 여러개가 존재할 수 있기 때문에 이를 충돌없이 처리하기 위해 필요한 동작이다. 따라서 인터럽트 버스 상의 인터럽트 전송은 항상 요청기에 의한 중재 주기부터 시작된다. 인터럽트 요청기는 진행 중인 인터럽트의 마지막 단계에서 IBSYNC* 구동을 멈춤으로써 다음 주기가 중재 주기가 될 수 있음을 알린다. 이때 인터럽트를 보낼 요청기가 있으면 중재 주기가 시작되고, 중재 방법은 8 비트의 부호화된 자가 중재 기법(coded self arbitration scheme)을 사용한다. 중재 주기에서 우선 순위 경쟁을 위하여 사용하는 정보는 인터럽트 종류(interrupt class), 요청기의 구별자(id address)이며 항상 우선 순위 방법을 사용한다. 중재에서는 패리티를 사용하지 않는다. 인터럽트 중재 주기는 5개 버스 클럭 주기동안 수행된다.

4.5.1 중재 정보

중재 주기에서 각 요청기가 사용하는 중재 정보는 <표 4.5>와 같다. 비트 7부터 비트 0가

Table 4.5: 인터럽트 버스 중재에 사용되는 중재 정보

bit field	information
7, 6	interrupt class
5, 4, 3, 2	ida<5..2>
1, 0	ida<1..0>

IBD<7..0>*의 신호에 각각 대응되어 구동되고, 중재 주기에서는 패리티 비트는 사용하지 않는다. 비트 0부터 5까지는 <표 4.6>에 나타낸 요청기 구별자가 되고, 비트6, 7은 <표 4.3>에 나타낸 인터럽트의 종류를 나타내는데, 그 숫자가 클수록 중재에 있어서 높은 우선 순위를 갖는다. 비트 6, 7은 인터럽트의 종류에 따른 우선 순위 정보이외에 인터럽트 요청의 존재 유무(0일 경우 No interrupt)를 나타낸다.

<표 4.5>의 비트 0부터 5까지는 요청기 구별자를 나타내는데, 요청기 구별자는 <표 4.6>과 같다. 요청기가 위치하는 슬롯 어드레스에 따라 요청기 구별자(ida<5..0>)가 결정되는데 슬롯0에서 12 까지는 4개씩의 요청기가 배정될 수 있고, 슬롯13 부터 슬롯20 까지는 정의를 유보한다. 슬롯10은 시스템 콘트롤러가 삽입되는 곳으로 요청기 구별자가 4 개 더 배정된다. 슬롯10번의 경우 시스템 제어에 관련된 인터럽트를 수행할 경우는 가장 높은 우선 순위를 사용하고, 일반적인 입출력 동작을 할 경우는 자신의 보통의 요청기 구별자를 사용할 수 있도록 하기 위함이다.

4.5.2 부호화된 자가 중재 기법

부호화된 자가 중재 기법(coded self arbitration scheme : CSAS)은 wired-OR 버스의 특성을 이용한 중재 방법으로 버스를 요청하는 모든 모듈에 중재기(distributed arbiter)를 갖고, 각

Table 4.6: 인터럽트 중재 주기에 사용되는 구별자 배정

Slot ID	Interrupt Requester Identification Address	
	ida<5..2>	ida<1..0>
0 - 9	GA<3..0>	<i>reserved</i>
10	GA<3..0> 0x15	<i>reserved</i> <i>reserved</i>
11 - 12	GA<3..0>	<i>reserved</i>
13 - 20		<i>undefined</i>

중재기에 고유의 번호(인터럽트 버스에서는 슬롯 어드레스)를 코드화하여 할당한다. 버스는 wired-OR 특성을 갖기 때문에 동시에 여러 모듈이 버스에 신호를 구동하였을 경우 low 값만 나타나게 된다. 중재가 진행되는 방법은, 버스 모듈들은 동시에 중재정보(중재번호)를 구동하고 모든 신호가 버스 상에서 안정된 후 자신이 구동한 중재번호가 버스에 나타나면 중재에서 이진것이고, 그러하지 않을 경우 신호의 구동을 멈추고 다음 중재 주기까지 기다리게 된다. CSAS 방법을 사용하려면 버스의 신호선은 중재를 원하는 모듈 수에 \log_2 를 취한 수¹ 만큼이 필요하고, wired-OR 기능이 있는 신호선을 사용해야 하며, 중재 시간은 신호의 수에 비례하게 된다. <그림 4.1>는 CSAS을 이용한 중재기의 일례이다.

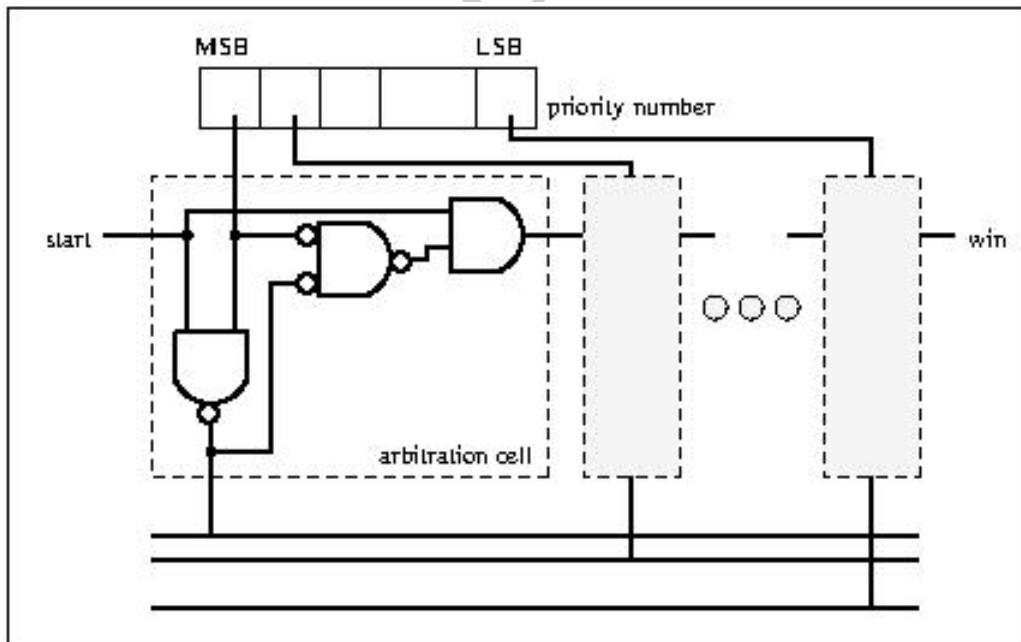


Figure 4.1: 부호화된 자가 중재기의 일례

¹ $n = \lceil \log_2 N \rceil$; n은 필요한 버스의 신호선 수, N은 모듈의 수; $\lceil x \rceil$ denotes the least integer that is greater than or equal to x.

4.6 지정 인터럽트

지정 인터럽트(direct interrupt)란 인터럽트 요청기가 인터럽트를 받을 처리기의 주소를 지정하여 전송하는 것을 의미한다. 일반적으로 지정 인터럽트는 시험 및 고장 진단을 위한 시스템의 제어 (시험, 고장 진단 등), 에러의 보고 그리고 실시간 응용 등에 사용될 수 있다. 지정 인터럽트는 인터럽트 중재가 끝난 바로 다음 주기에서 시작되고, 5 개의 단계 (Phase)로 구성된다. 각 단계는 버스 클럭 주기 동안 수행된다. 5 단계는 중재 주기를 거쳐 사용 허가를 받은 요청기가 인터럽트 처리기의 주소, 요청기의 주소, 인터럽트의 종류 (벡터)를 전송하는 3 단계와 지정된 처리기가 받은 정보에 대한 응답을 준비하고 응답을 보내는 2 단계로 구성된다. <표 4.7>는 지정 인터럽트의 단계 별 전송되는 정보와 보내는 측과 받는 측 등을 보여주고 있다. 그리고 <그림 4.2>는 실제 인터럽트 버스상에서의 동작을 보이고 있다.

Table 4.7: 지정 인터럽트의 단계별 전송 정보

phase	transfer information	from	to
-5	interrupt class + slot id + id in slot	IR	IR's
-4	interrupt class + slot id + id in slot	IR	IR's
-3	interrupt class + slot id + id in slot	IR	IR's
-2	interrupt class + slot id + id in slot	IR	IR's
-1	interrupt class + slot id + id in slot	IR	IR's
0	interrupt class + destination address	IR	IH's
1	vector type + source address	IR	IH
2	vector	IR	IH
3	dummy	-	-
4	acknowledge	IH	IR

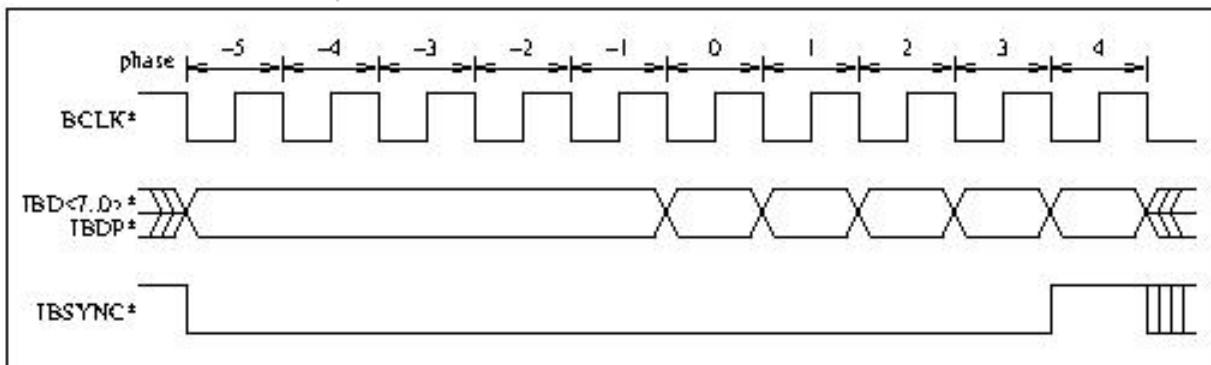


Figure 4.2: 지정인터럽트의 버스 동작

지정 인터럽트 - 단계 -5 — 단계 -1

인터럽트 버스를 사용하기에 앞서 수행하는 중재 주기이다. 인터럽트 요청기는 현재 진행 중인 단계가 지정 인터럽트를 보내기에 앞서 수행하는 중재 주기임을 알리기 위해 IBSYNC* 신호를 구동한다.

지정 인터럽트 - 단계 0

이 단계에 전송되는 정보는 종류 정보와 인터럽트를 받는 측(처리기)의 주소가 전송된다. 인터럽트 요청기는 현재 인터럽트 전송이 진행되고 있음을 알리기 위해 IBSYNC* 신호를 구동한다. 비트별 전송 정보는 <표 4.8>과 같다. 종류 정보는 지정된 처리기가 저장하게 되

Table 4.8: 지정 인터럽트 - 단계 0에서 전송되는 정보

bit field	information
7, 6	interrupt class
5, 4, 3, 2	slot id
1, 0	id in slot

는데, 받는 측은 이 값에 따라 다음부터 진행될 단계에 대해서 준비할 수 있게 된다. 종류 필드의 값에 따른 의미는 <표 4.3>와 같다.

이 단계에서 전송되는 비트 0에서 비트 5까지의 값은 다음 주기부터 전송되는 인터럽트를 받을 처리기의 주소를 나타내는데, 이때 지정된 처리기는 요청기에서 오는 인터럽트 정보를 받고, 해당 동작을 수행하게 된다. 비트 0부터 비트 5까지의 값에 대한 처리기의 할당은 <표 4.9>과 같다. Slot ID가 15일 경우는 모든 처리기가 지정된다. 모든 처리기가 지정될 경우, 다음 주기부터 전송되는 정보를 모든 처리기는 동시에 저장하게 된다.

Table 4.9: 지정 인터럽트 단계 0의 처리기 어드레스

Slot ID	ID in Slot	Interrupt Requester
0 - 12	x	Slot 0 - Slot 12
13 - 14	x	reserved
15	x	All Handler

지정 인터럽트 - 단계 1

이 단계에서는 인터럽트 요청기가 처리기에게 벡터의 종류와 자신의 어드레스를 전송한다. 인터럽트 요청기는 현재 인터럽트 전송이 진행 되고 있음을 알리기 위해 IBSYNC* 신호를 구동한다. 처리기의 주소는 6 비트가 사용되고, 상위의 2 비트는 벡터의 종류에 대한 정보를

Table 4.10: 지정 인터럽트 - 단계 1에서 전송되는 정보

bit field	information
7, 6	vector type
5, 4, 3, 2	slot id
1, 0	id in slot

같이 전송한다. <표 4.10>는 각 비트별 전송 정보를 나타내고 있다. 지정 인터럽트에 의해 전송될 수 있는 벡터의 종류는 <표 4.4>과 같다. 모든 형태의 벡터를 전송할 수 있으나, 최대 전송할 수 있는 벡터의 크기가 8 비트로 한정되어 있기 때문에 16 비트의 벡터를 갖는 경우는 하위 8 비트를 0으로 간주한다. 지정 인터럽트에 의해 전송되는 벡터는 주로 E-와 I-Type이 되며, 시스템의 시험이나 진단을 위한 목적으로 사용된다. E-Type은 마스크가 불가능하기 때문에 도착한 인터럽트는 항상 해당 프로세서로 전달하고, I-Type의 경우 마스크되어 있으면 도착한 인터럽트를 무시한다. Q-와 N-Type은 중재 인터럽트와 동일하게 동작한다 (중재 인터럽트 1 참조).

지정 인터럽트 - 단계 2

이 단계에서는 요청기가 처리기로 인터럽트 벡터(Interrupt Vector)를 보낸다. 인터럽트 요청기는 현재 인터럽트 전송이 진행되고 있음을 알리기 위해 IBSYNC* 신호를 구동한다. 인터럽트 벡터는 요청기가 실질적으로 처리기에 원하는 행동이 무엇인지를 담고 있는 것이다. 인터럽트 벡터는 그들이 갖고 있는 의미에 따라 입출력 동작 제어를 위한 것, 각 슬롯에서 발생된 에러의 보고를 위한 것, 각 슬롯에서 보고된 에러의 처리를 위하여 시스템 콘트롤러에서 보내는 제어를 위한 것, 끝으로 프로세서들 사이에 그들이 수행하고 있는 프로세서 제어나 대화를 위한 정보의 교환을 위한 것 등으로 분리할 수 있다. 8 비트의 벡터 값에 따른 의미는 4.4에서 설명한 바와 같다.

지정 인터럽트 - 단계 3

인터럽트 요청기는 현재 인터럽트 전송이 진행되고 있음을 알리기 위해 IBSYNC* 신호를 구동한다. 인터럽트를 받은 처리기가 요청기에게 응답을 보내기 위해 이제 까지 받은 정보를 점검하여 다음 단계에서 요청기로 보낼 응답을 준비한다.

지정 인터럽트 - 단계 4

인터럽트 요청기는 현재 진행 중인 단계가 지정 인터럽트의 마지막임을 알리기 위해 이제 까지 구동하고 있던 IBSYNC* 신호를 걷어낸다. 지정 인터럽트의 최종 단계로, 인터럽트를 받은 처리기가 요청기에게 응답을 보내는 일을 수행한다. 응답 단계에서는 진행된 인터럽트 전송에 에러 발생 여부와 인터럽트 처리기의 상태 등을 전달한다. <표 4.11>는 전송되는 8 비트의 응답 정보의 각 비트별 의미를 나타내고 있다. 에러가 발생했을 때 해당 비트의 값은

“거짓”이 된다. 따라서 정확한 순서와 의미로 인터럽트가 전송된 경우는 각 비트의 값이 “참”이 되도록 하여야만 한다.

Table 4.11: 지정 인터럽트 - 단계 3에서 전송되는 정보의 비트별 의미

bit field	information
7	sequence error
6	parity error
5	buffer full error
4	unacceptable error
3	<i>reserved</i>
2	<i>reserved</i>
1	<i>reserved</i>
0	overwritten error

- 비트 7은 인터럽트 전송 주기 진행에의 에러가 발생했을 경우에 “거짓”이 되는 비트로 전송 도중 (DI의 phase4이나 AI-1의 phase21 제외)에 IBSYNC* 신호가 거짓이 되었을 경우가 해당된다.
- 비트 6은 단계 0, 1 혹은 2에서 전송된 데이터에 패리티 에러가 검출되었을 때 “거짓”이 된다.
- 비트 5는 벡터가 저장될 곳이 다른 인터럽트에 의해 가득찼을 경우에 “거짓”이 된다.
- 비트4는 전송된 벡터를 저장할 수 없는 처리기나 그밖에 정의되지 않은 에러에 의하여 벡터를 접수할 수 없는 경우에 “거짓”이 된다.
- 비트 0은 처리기에 현재 처리가 지연된 인터럽트(pended interrupt)가 존재했는데 새로운 인터럽트가 발생하여 이전 것을 지워지고 새 인터럽트가 저장됐을 경우 “거짓”이 된다.

각 비트는 정의된 상황이 발생했을 경우 “거짓”이 되도록 함에 따라 처리기가 아무 응답을 안할 경우도 에러로 처리된다. 또한 두 개 이상의 처리기가 응답을 해야하는 경우(broadcast)에는 한개만이라도 에러가 없다고 응답이 오면 에러가 발생하지 않은 것으로 간주된다.

4.7 중재 인터럽트 1

중재 인터럽트 1(arbitration interrupt 1)은 다중 프로세서 환경의 효과적인 인터럽트 분배를 구현하기 위해 필요한 인터럽트 전송 방법의 한 형태이다. 이 방법은 인터럽트의 동적인 분배를 실현할 수 있으며, 분배 과정에서 프로세서의 불필요한 Context Switching을 막을 수 있도록 구현되어 있다. 다중 프로세서 운영 체제 환경에서 입출력의 제어와 프로세서 사이의 대화 수단 등으로 사용할 수 있다.

중재 인터럽트 1은 지정 인터럽트와는 달리 처리기를 지정하지 않고(프로세서의 모임만 지정), 한개 이상의 처리기에 인터럽트를 동시에 전달한 후, 처리기들끼리 중재를 하여 인터럽트를 접수할 처리기를 하나만 선정하도록 한다. 중재 과정에서 처리기들은 현재 자신의 상태(상위 프로세서 등의 상태 포함)에 따라 우선 순위 경쟁을 하게 되는데, 그 순간에 인터럽트를 처리하기에 가장 적절한 것이 선정된다.

중재 인터럽트 1도 다른 인터럽트와 같이 인터럽트 버스 사용에 관한 중재 주기가 끝난 바로 다음 주기부터 요청기에 의하여 시작되고, 22 개의 단계로 구성된다. 첫 단계는 인터럽트 종류, 벡터의 종류 그리고 처리기의 모임을 지정하는 정보를 전송하고, 단계1에서는 단계0에서 전달된 정보를 처리기들이 처리하는 시간이고, 단계2부터 단계16까지는 처리기들 사이의 중재가 진행된다., 단계17에서는 요청기의 어드레스를 전송하고, 단계18과 단계19에서는 16 비트의 벡터를 전송하고, 단계20에서는 응답기가 이제까지 받은 정보에 대한 점검을 하고, 끝으로 단계21에 처리기는 요청기로 받은 인터럽트에 대한 응답을 보낸다.

<표 4.12>는 중재 인터럽트 1의 각 단계 별 전송되는 정보와 방향등을 나타내고 있다. 표에서 처리기에 복수로 표현된 것은 여러개의 처리기가 동작에 참여한다는 것을 의미한다. 그리고 <그림 4.3>는 실제 인터럽트 버스상에서의 동작을 보이고 있다.

중재 인터럽트 1 - 단계 -5 – 단계 -1

인터럽트 버스를 사용하기에 앞서 수행하는 중재 주기이다. 인터럽트 요청기는 현재 진행중인 단계가 중재 인터럽트를 보내기에 앞서 수행하는 중재 주기임을 알리기 위해 IBSYNC* 신호를 구동한다.

중재 인터럽트 1 - 단계 0

인터럽트 요청기는 현재 중재 인터럽트 전송이 진행되고 있음을 알리기 위해 IBSYNC* 신호를 구동한다. 단계 0(phase 0)에서 전송되는 정보는 <표 4.13>과 같이 인터럽트 종류(Class), 벡터의 형태 (Type)와 처리기들의 모임 주소 (Group ID)가 전송된다. 인터럽트 종류 필드는 중재 인터럽트 1을 나타내는 “2”가 전송된다. 벡터의 종류에 관한 정보가 단계 0에서 전송되는 것이 지정 인터럽트와 다른점으로 볼 수 있다. 벡터의 종류를 단계 0에서 전송하는 이유는 단계 2에서 16까지 진행되는 중재에서 이 정보가 필요하기 때문이다. 중재 인터럽트에서 사용할 수 있는 형태는 <표 4.14>와 같다. 중재 인터럽트의 주된 이용은 다중 처리 운영 체제가 동작하는데 있어서 프로세서들 사이의 동적인 인터럽트의 분배를 위한 것이다. 따라서 벡터 종류도 그와 같은 응용과 밀접한 관계를 갖고 있다.

- N-Type은 notify라는 말이 의미하는 바와 같이 임의의 순간에 발생한 어떤 사건을 처리 기로 통고하는 의미를 지닌다. 따라서 그 인터럽트는 시간과 관계를 갖고 있기 때문에

Table 4.12: 중재 인터럽트 1의 단계별 전송 정보

phase	transfer information	from	to
-5	Interrupt Class + slot id + id in slot	IR	IR's
-4	Interrupt Class + slot id + id in slot	IR	IR's
-3	Interrupt Class + slot id + id in slot	IR	IR's
-2	Interrupt Class + slot id + id in slot	IR	IR's
-1	Interrupt Class + slot id + id in slot	IR	IR's
0	Interrupt Class + Type + Destination Group ID	IR	IH's
1	dummy	-	-
2	invers num. of Interrupts in Handler	IH's	IH's
3	invers num. of Interrupts in Handler	IH's	IH's
4	invers num. of Interrupts in Handler	IH's	IH's
5	invers num. of Interrupts in Handler	IH's	IH's
6	invers num. of Interrupts in Handler	IH's	IH's
7	invers Priority of Process	IH's	IH's
8	invers Priority of Process	IH's	IH's
9	invers Priority of Process	IH's	IH's
10	invers Priority of Process	IH's	IH's
11	invers Priority of Process	IH's	IH's
12	Slot Address of Handler	IH's	IH's
13	Slot Address of Handler	IH's	IH's
14	Slot Address of Handler	IH's	IH's
15	Slot Address of Handler	IH's	IH's
16	Slot Address of Handler	IH's	IH's
17	Source Address	IR	IH
18	Vector 1	IR	IH
19	Vector 0	IR	IH
20	dummy	-	-
21	Acknowledge	IH	IR

Table 4.13: 중재 인터럽트 1 - 단계 0에서 전송되는 정보

bit field	information
7, 6	2
5, 4	vector type
3, 2, 1, 0	group id

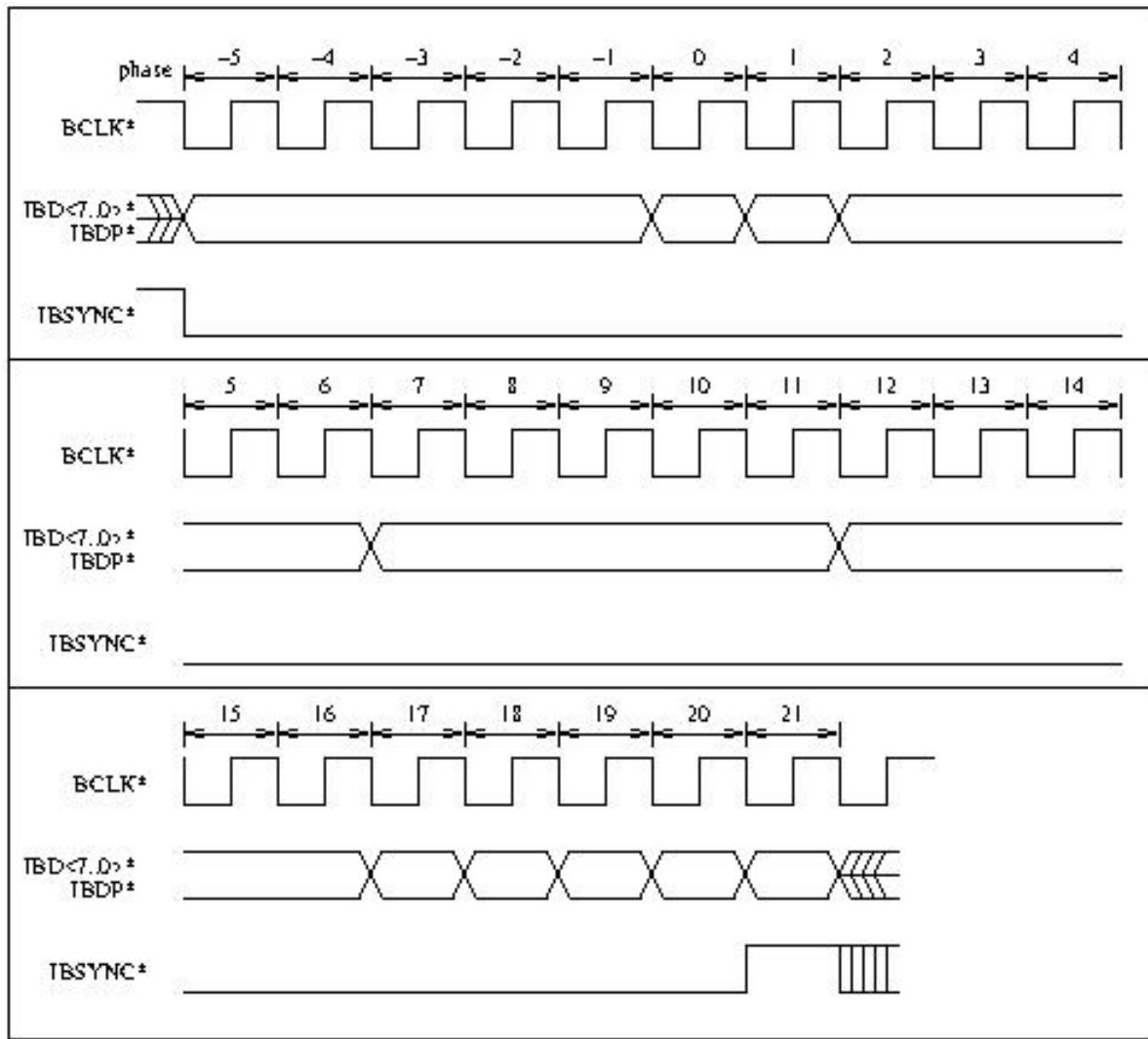


Figure 4.3: 중재인터럽트1의 버스 동작

시간이 지나면 그 의미를 상실할 수 있다. 또한 인터럽트가 처리되지 않은 상태에서 또 다시 그 인터럽트가 발생할 경우는 이와 같은 인터럽트에 해당되는 것은 4.4.3에서 설명한 바와 같다.

- Q-Type은 한번 전달된 인터럽트는 처리될 때까지 Queue에 저장된다. 즉 한번 발생된 인터럽트는 무시될 수 없고, 처리가 지연될 수 있으나 무시될 수 없다. 입출력 인터럽트와 같이 대부분의 인터럽트가 이 형태에 해당된다. 따라서 인터럽트를 저장할 장소가 없어 인터럽트가 지연될 경우에 새로운 인터럽트가 도착하면 요청기로 버퍼에 여유가 없음을 알린다. 처리기에 Queue(FIFO Buffer)를 사용하여 구현하지 않는 경우(오직 한개의 인터럽트만 저장할 수 있음)는 요청기에서 재시도(retry)를 통해서 Queue의 기능을 대신할 수 있다.

모임 주소(Group ID)는 처리기들과 연결된 프로세서들을 서로 기능적으로 분류한 주소를

Table 4.14: 중재 인터럽트 1의 벡터 형태

type	name	description
0	Q-type	queue interrupt
1	N-type	notify interrupt
2	AI-1 type-2	<i>reserved</i>
3	AI-1 type-3	<i>reserved</i>

말한다. 본 인터럽트 버스에서는 <표 4.15>와 같이 4 종류의 모임을 갖을 수 있고 현재 정의된 주소는 3 개이다. 모임 주소는 비트별로 할당되어 따라서 동시에 2 개 이상의 모임을 선택할 수 있다. <표 4.15>는 모임 주소 각 비트가 지정하는 처리기(해당 프로세서)를 보여주고 있다.

Table 4.15: 중재 인터럽트 1의 모임 주소 각 비트별 의미

group id bit field	group name
bit 3	system controllers
bit 2	input-output processors
bit 1	<i>reserved</i>
bit 0	general purpose processors

- 비트3은 시스템 콘트롤러 그룹을 지정하며, 시스템 제어를 할 수 있는 처리기들이 지정된다.
- 비트2는 입출력 처리를 맡고 있는 처리기들이 지정되며, 입출력 프로세서와 시스템 콘트롤러 안에 입출력 처리 기능을 수행하는 부분에 해당되는 처리기가 지정된다.
- 비트0는 일반적인 데이터 처리를 할 수 있는 처리기(프로세서)가 지정된다.
- 비트1은 현재 지정되지 않은 것으로 실시간 처리나 그밖의 응용을 위하여 유보하고 있다.

각 비트는 어느 조합으로도 동시에 지정이 가능함으로써 3 비트를 동시에 구동하면 시스템 안의 모든 처리기로 중재 인터럽트를 보내게 된다.

중재 인터럽트 1 - 단계 1

인터럽트 요청기는 현재 중재 인터럽트 전송이 진행되고 있음을 알리기 위해 IBSYNC* 신호를 구동한다. 인터럽트 처리기들은 단계 0(phase 0)에서 전송된 정보를 처리한다.

중재 인터럽트 1 - 단계 2 — 단계 6

인터럽트 요청기는 현재 중재 인터럽트 전송이 진행되고 있음을 알리기 위해 IBSYNC* 신호를 구동한다. 단계 2 (phase 2)부터는 단계 0에서 지정된 처리기들 사이에 수행되는 중재 동작의 첫 단계이다. 처리기들 사이의 중재는 3단계로 구성되는데, <표 4.12>의 단계 2에서 단계 16까지가 해당된다. 지정된 모임 안에 해당되는 처리기들은 각 단계마다 우선 순위 경쟁을 수행하고, 수행한 단계에서 선정된 것들만 계속되는 중재 주기를 수행한다. 결과적으로 단계 16을 마친 후에는 하나의 처리기만 선정되게 된다. 따라서 단계별로 우선 순위가 적용되어 단계 2가 가장 높은 우선 순위 경쟁이 된다.

이 단계에서 처리기들이 우선 순위 경쟁을 위해 사용하는 정보는 처리기 안에 현재 있는 인터럽트 갯수의 역수와 인터럽트 마스크 비트의 상태 (마스크 되어 있을 경우 0으로 버스에 구동한다)이다. 이와 같은 중재 정보가 의미하는 것은 마스크가 되어 있지 않을수록, 그리고 처리가 지연되고 있는 인터럽트(Pended Interrupt)가 적은 처리기일수록 새로운 인터럽트를 접수하는데 있어서 높은 우선 순위를 갖는다는 것이다. 이때 마스크와 인터럽트 갯수는 단계 0에서 지정한 벡터 형태에 관한 것이 된다. <표 4.16>은 중재 정보로 사용되는 각 데이터의 비트별 할당 상황이다.

Table 4.16: 중재 인터럽트 1 - 단계 2에서 사용되는 정보

bit field	information
7	reverse mask bit
6	<i>reserved</i>
5, 4, 3, 2, 1, 0	reverse number of interrupts in handler

중재 인터럽트 1 - 단계 7 — 단계 11

인터럽트 요청기는 현재 중재 인터럽트 전송이 진행되고 있음을 알리기 위해 IBSYNC* 신호를 구동한다. 단계 7(phase 7)에서는 단계 2의 우선 순위 경쟁에서 이긴 것만 중재를 계속 수행하게 된다. 이 단계에서는 현재 각 처리기에 관련된 프로세서가 수행하고 있는 작업의 우선 순위 (작업의 우선 순위는 운영체제에서 정의한 관리 단위의 수행상 경쟁을 말함. UNIX의 경우는 process priority가 이에 해당됨)를 사용하여 중재를 한다. 따라서 새로운 작업을 시작할 때(Context Switching)나 작업의 우선 순위가 변화할 때는 그 우선 순위(Process Priority)를 인터럽트 처리기에 알려주어야 한다. 처리기가 사용할 수 있는 있는 우선 순위는 최대 8 비트이다. 높은 우선 순위를 갖는 작업을 처리하고 있는 프로세서일 경우 이 단계에서 출력되는 정보는 적은 값이 된다.

중재 인터럽트 1 - 단계 12 — 단계 16

인터럽트 요청기는 현재 중재 인터럽트 전송이 진행되고 있음을 알리기 위해 IBSYNC* 신호를 구동한다. 단계 12에서도 앞에서와 같이 단계 7까지 우선 순위 경쟁에서 이긴 것들

Table 4.17: 중재 인터럽트 1 - 단계 7에서 사용되는 정보

bit field	information
7, 6, 5, 4, 3, 2, 1, 0	inverse priority of process

사이에서만 수행한다. 이 단계에서 사용하는 경쟁 정보는 그 처리기가 현재 위치하는 슬롯 어드레스를 이용한다. 따라서 단계 7까지는 2 개 이상의 처리기가 우선 순위 경쟁에서 이길 수 있으나, 이 과정을 마치고 나면 항상 1 개의 처리기만이 선정되게 된다. 각 슬롯 어드레스에 대한 위치 배정은 <표 4.18>과 같다.

Table 4.18: 중재 인터럽트 1 - 단계 12에서 사용되는 정보

bit field	information
7, 6	0
5, 4, 3, 2	slot id
1, 0	id in slot

중재 인터럽트 1 - 단계 17

인터럽트 요청기는 현재 중재 인터럽트 전송이 진행되고 있음을 알리기 위해 IBSYNC* 신호를 구동한다. 이 단계에서는 요청기가 선정된 하나의 처리기로 자신의 주소를 전송한다. 전송되는 정보의 각 비트별 할당은 지정 인터럽트의 단계 1과 같은데, 단지 비트 6과 7을 사용하지 않은 점이 다르다. 이단계에서 사용하는 정보는 <표 4.19>과 같다.

Table 4.19: 중재 인터럽트 1 - 단계 17에서 전송되는 정보

bit field	information
7, 6	0
5, 4, 3, 2	slot id
1, 0	id in slot

중재 인터럽트 1 - 단계 18

인터럽트 요청기는 현재 중재 인터럽트 전송이 진행되고 있음을 알리기 위해 IBSYNC* 신호를 구동한다. 단계 18에서는 요청기가 처리기로 16 비트의 인터럽트 벡터 중 상위 8 비트를 전송한다. 인터럽트 벡터의 의미는 4.4에서 설명한 바와 동일하다.

Table 4.20: 중재 인터럽트 1 - 단계 18에서 전송되는 정보

bit field	information
7, 6, 5, 4, 3, 2, 1, 0	most significant byte of interrupt vector

중재 인터럽트 1 - 단계 19

인터럽트 요청기는 현재 중재 인터럽트 전송이 진행되고 있음을 알리기 위해 IBSYNC* 신호를 구동한다. 단계 19에서는 요청기가 처리기로 16 비트의 인터럽트 벡터 중 하위 8 비트를 전송한다. 인터럽트 벡터의 의미는 4.4에서 설명한 바와 동일하다.

Table 4.21: 중재 인터럽트 1 - 단계 19에서 전송되는 정보

bit field	information
7, 6, 5, 4, 3, 2, 1, 0	lest significant byte of interrupt vector

중재 인터럽트 1 - 단계 20

인터럽트 요청기는 현재 중재 인터럽트 전송이 진행되고 있음을 알리기 위해 IBSYNC* 신호를 구동한다. 인터럽트 요청기는 이제까지 받은 정보에 대해 점검하고 다음 단계에 사용할 응답을 준비한다.

중재 인터럽트 1 - 단계 21

인터럽트 요청기는 현재 진행 중인 단계가 중재 인터럽트의 마지막임을 알리기 위해 이제까지 구동하고 있던 IBSYNC* 신호를 걷어낸다. 단계 7에서는 인터럽트 처리기가 인터럽트를 보낸 요청기로 응답을 보내는 단계이다. 이 단계에서 전송되는 신호의 의미는 지정 인터럽트의 경우와 동일하며, <표 4.11>과 같다.

4.8 중재 인터럽트 0

중재 인터럽트 0(arbitration interrupt 0)은 현재 16 단계만 확정되어 있고 앞으로의 확장을 위하여 규격 결정을 유보하고 있다.

4.9 예외 처리 (Exception Handling)

본 절에서는 인터럽트 버스 상에서 동작 중에 앞에서 정의되지 않은 상태가 발생했을 때 처리하는 방법에 대해 기술한다.

4.9.1 전송 에러 (Transmission Error)

인터럽트 전송 주기 안에서 처리기가 패리티 에러를 검출한 경우를 말한다. 처리기는 응답 단계를 통하여 요청기로 전송 중에 에러가 발생했음을 통고하고, 받은 인터럽트는 등록하지 않는다. 요청기는 전송 에러를 받으면 중재 주기 부터 다시 인터럽트 전송을 재시도 (Retry) 한다. 재시도한 전송 주기에서 다시 동일한 에러를 검출할 경우는 고장이 발생했음을 시스템 콘트롤러에 전달하고 그 이후의 동작은 버스 규격에서 제시하지 않는다.

Chapter 5

유틸리티 버스

5.1 개요

유틸리티 버스(utility bus)는 버스의 기능을 수행하기 위해 기본적으로 필요한 신호과 버스의 부가적인 기능을 구현하기 위해 추가된 신호들의 집합이다.

Table 5.1: 유틸리티 버스의 규격 요약

버스 클럭 (Bus Clock)	
클럭의 속도	16.5 MHz (60.6 nsec)
버스 제어를 위한 신호	
슬롯 위치 식별용	Geographical Slot Address
시스템 제어 용	System Fail, Reset
경계주사 용	JTAG Boundary Scan
기 타 (Etc.)	
총 신호수	13

5.2 유틸리티 버스의 신호선

Table 5.2: 유틸리티 버스의 신호들

Mnemonic	Size	Name
BCLK*	1	Bus Clock
GA<4..0>*	5	Geographical Slot Address
SFAIL*	1	System Fail
RST*	1	Reset
Tx<4..0>	5	JTAG Boundary Scan Option

Bus Clock : BCLK*

버스 동작의 기준 시간이 되는 신호이다. 백플레인에서 공급된다. 각 슬롯에 도착되는 신호의 시간 차이 발생을 최소화하기 위하여 두개의 슬롯씩 별도의 신호를 두어 클럭을 공급한다. 단 슬롯 10번은 슬롯 8, 슬롯 9와 같이 연결된다.

Geographical Slot Address : GA<4..0>*

슬롯 어드레스는 슬롯의 위치를 나타내는 5 비트의 신호들로써 슬롯에 삽입된 보드는 이 신호를 통하여 자신의 위치를 알 수 있게 된다. 슬롯의 어드레스는 주서브랙(main sub-rack)의 전면에서 볼 때 가장 왼쪽이 슬롯 0가 되고, 순서적으로 할당되어 가장 오른쪽이 슬롯 20이 된다. 각 슬롯마다 독립적으로 자신의 위치를 나타내는 값을 버스 콘넥터에 제공해야 하므로 다른 버스의 신호와 달리 모든 슬롯을 연결시키는 신호가 아니다. <표 5.3>은 보드가 꽂히는 쪽(백플레인의 전면)에서 바라본 슬롯의 위치와 슬롯 어드레스 값의 대응관계를 보여 주고 있다. 각 슬롯에 꽂히는 보드들은 초기화 과정에서 이 슬롯 어드레스를 읽어서 중재 번호,

Table 5.3: 슬롯 어드레스와 슬롯 위치

Slot No.	0	1	2	3	4	5	6	7	8	...	13	14	15	16	17	18	19	20
GA<4>*	0	0	0	0	0	0	0	0	0	...	0	0	0	1	1	1	1	1
GA<3>*	0	0	0	0	0	0	0	1	...	1	1	1	0	0	0	0	0	0
GA<2>*	0	0	0	0	1	1	1	0	...	1	1	1	0	0	0	0	0	1
GA<1>*	0	0	1	1	0	0	1	1	0	...	0	1	1	0	0	1	1	0
GA<0>*	0	1	0	1	0	1	0	1	0	...	1	0	1	0	1	0	1	0

인터럽트 버스의 인터럽트 요청기와 처리기의 주소, 그리고 기타 자신을 나타내는 번호(ID)로 사용한다.

System Reset : RST*

시스템 전원을 올린 후 시스템 내의 모든 하드웨어의 초기화를 위하여 사용하며, 또한 시스템의 운영중 복구가 불가능한 에러가 발생했을 경우에도 하드웨어의 초기화를 위하여 사용한다. 시스템 콘트롤러에 의해서 구동되고 다른 모든 슬롯에 위치하는 모듈은 입력 신호로 사용된다. 이 신호가 구동될 경우 각 모듈은 수행 중인 작업을 중단하고 자신이 갖고 있는 모든 자원을 초기화하여야 한다. 이 신호의 시간 규격은 최소한 100 msec 동안 안정한 값을 구동하여야 한다는 것이다.

System Fail : SFAIL*

시스템의 고장 신호는 국부적으로 복구가 불가능한 시스템의 고장이 발생했을 때, 시스템 콘트롤러에 고장의 발생을 알리는 신호로 사용된다. 보통 복구가 불가능한 에러는 동시에 여러 슬롯에서 발생할 수 있기 때문에 두개 이상의 모듈이 동시에 구동이 가능하다.

Boundary Scan Option : Tx<4..0>

JTAG (Joint Test Action Group) 경계주사(boundary scan) 기능을 위한 핀이다. 정확한 기능과 규격은 IEEE Std.1149.1을 따르며 여기에서는 <표 5.4>과 같이 사용핀을 정의한다.

Table 5.4: 경계주사

	define	description
Tx<4>	TRST*	Test Reset Input
Tx<3>	TDO	Test Data Out
Tx<2>	TDI	Test Data Input
Tx<1>	TMS	Test Mode Select Input
Tx<0>	TCK	Test Clock Input

ETRI-CSTL

Chapter 6

시간 규격

6.1 개요

시간규격에서는 백플레인에 구동되는 신호의 시간적 제약을 규정한다. 모든 시간규격은 백플레인에 나타나는 실제 신호를 기준으로 표시하며, 표시의 편의를 위해 토템폴 신호, 트라이스테이트 신호 그리고 오픈콜렉터 신호를 따로 구분하지 않고, 여러 신호가 모여서 동작하는 신호와 단일 신호를 따로 구분하지 않는다. 각 시간규격은 버스의 부하에 관계없이 최소값과 최대값의 범위를 넘지 않아야한다.

전달지연시간(propagation time)은 구동소자의 출력단 신호가 변하기 시작하여 백플레인에 나타나는 신호가 백플레인의 모든 곳에서 전기적규격에서 규정하는 V_{OL} 이하로 안정될 때까지의 시간으로 규정한다.¹

¹BTL의 경우 90%에서 10%로의 하강시간이 최대 10nsec이고, 백플레인의 슬롯이 0.8inch 간격으로 21개 있을 때 5nsec/ft 이내의 전달 지연을 보장하면 신호가 백플레인을 최장으로 전달된다해도 10nsec 이내의 지연이 보장되므로 HiPi+Bus에서는 버스 전달 지연(bus propagation time)을 최대 20nsec로 규정한다.

6.2 버스클럭의 시간규격

버스 클럭 신호는 본 시스템 버스의 모든 부분의 제어에 있어서 기준이 되는 신호이다. 클럭은 슬롯 10에서 공급되고 다른 슬롯에 위치한 모든 보드들은 이 신호를 받아서 제어 정보로 사용한다.

Table 6.1: 버스클럭의 시간규격 요약

timing parameter	name	min	typical	max
TP_0^{BCLK}	BCLK* cycle time	59	60	61
TP_1^{BCLK}	BCLK* high pulse width	-	30	-
TP_2^{BCLK}	BCLK* low pulse width	-	30	-
TP_3^{BCLK}	BCLK* falling transition time	3	-	5

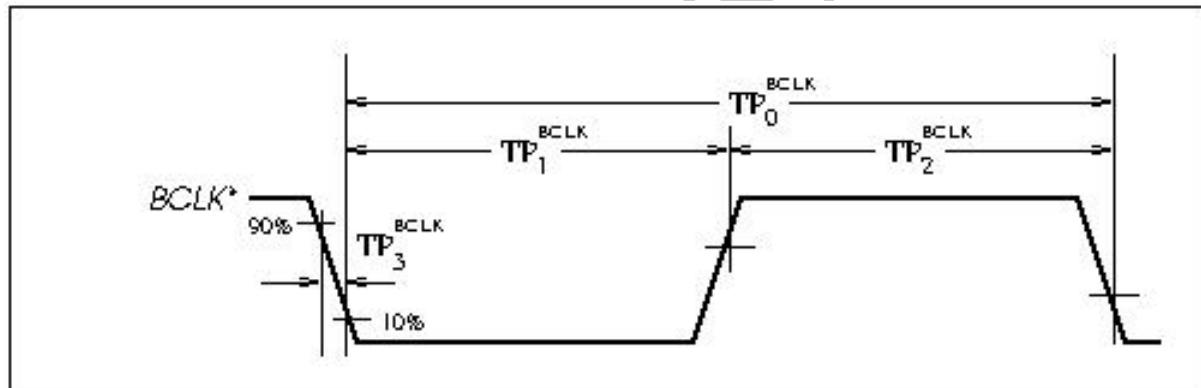


Figure 6.1: 버스클럭의 시간규격

버스클럭의 속도(clock frequency)는 16.5 MHz로 60.6 nsec의 주기(clock period, clock cycle time))를 갖고 정확한 제어를 위하여 백플레인에 나타나는 버스클럭 신호의 하강점(falling edge)만을 사용하도록 규정하고 시간적 허용오차는 $\pm 1\text{nsec}$ 이내로 규정한다. 클럭의 Duty Cycle은 50% ($TP_1^{BCLK} \div TP_0^{BCLK}$)로 규정한다. 버스클럭 신호가 90%에서 10%로 전이하는 시간인 하강점 전이시간(falling transition time)를 3 nsec 이상 5 nsec 이내로 규정한다.

6.3 중재버스의 시간규격

6.3.1 ABRQ<n>*, DBRQ<n>*의 시간규격

중재에 참가하기 위해서는 중재에 참가하고자 하는 버스클럭 주기에 해당 버스클럭의 백플레이인의 하강점에서 TP_1^{ARB} 시간 이후부터 TP_0^{ARB} 동안 해당 중재요청신호를 백플레이인에 안정되게 구동하여야 한다. 이때 중재요청신호의 전달지연시간은 TP_2^{ARB} 로 규정한다. 중재 주기에서 중재를 수행한 결과 버스 사용권을 획득한 중재기는 자신의 중재요청신호의 구동을 이어지는 중재에 여향을 미치지 않도록 충분히 빨리 신호구동을 멈추어야 하고 이 때 TP_3^{ARB} 를 보장하여야 한다. 나머지 중재기들은 중재요청신호를 계속 구동하여 이어진 중재주기로 중재동작을 계속하게 된다.

Table 6.2: 중재버스 신호의 시간규격 요약

timing parameter	name	min	typical	max
TP_0^{BCLK}	BCLK* cycle time	59	60	61
TP_0^{ARB}	arbitration time	-	50	-
TP_1^{ARB}	pre-time	-	-	10
TP_2^{ARB}	bus propagation time	-	-	20
TP_3^{ARB}	off time	10	-	15
TP_0^{INH}	ABINH time	25	30	35
TP_1^{INH}	bus propagation time	-	-	20
TP_2^{INH}	off time	10	-	15

6.3.2 ABINH*, WRINH*, DBINH*의 시간규격

이 신호는 중재요청신호의 구동을 제한하는데 사용하며 TP_0^{INH} 시간동안 안정된 신호로 유지되어야 한다. 연속해서 여러 버스 사이클 동안 구동하는 경우, TP_1^{INH} 시간은 무시하고 처음 구동 사이클에서 지킨다. 연속해서 여러 버스 사이클 동안 구동하는 경우, TP_2^{INH} 시간은 무시하고 마지막 구동 사이클에서 지킨다.

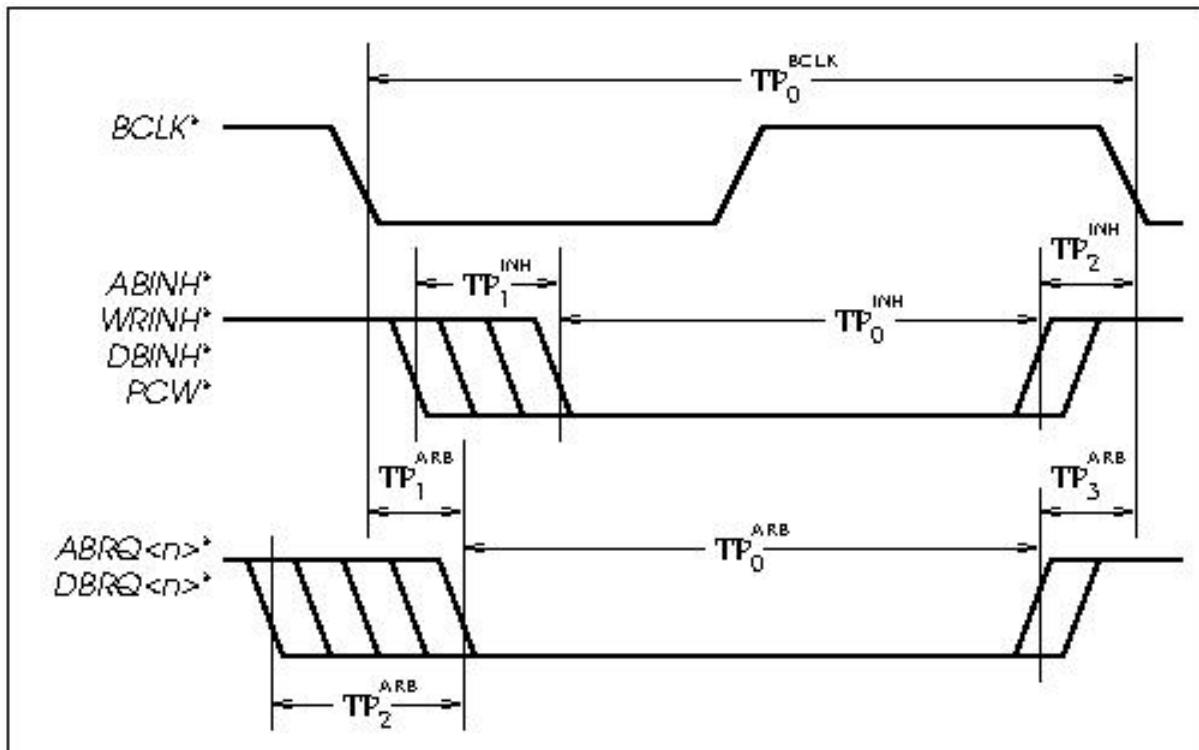


Figure 6.2: 중재버스의 시간규격

6.4 데이터 전송버스의 시간규격

데이터 전송 버스는 동기형 제어 방식을 사용함에 따라 모든 신호는 한개 클럭 주기 동안에 버스 상에서 구동되고 감지하게 된다. 따라서 모든 신호는 데이터 전송 프로토콜에서 규정된대로 자신에게 할당된 주기에 언제나 한개 버스 클럭 동안만 구동을 하게된다.

데이터 전송버스의 모든 정보는 백플레인에 나타나는 버스클럭 신호의 하강점으로부터 TP_0^{DTB} (DTB signal latch time)후에 데이터 전송버스 정보를 안정되게 래치할 수 있도록 보장해야 한다. 이를 위하여 셋업시간(setup time)으로 TP_2^{DTB} 를 홀드시간(hold time)으로 TP_3^{DTB} 를 반드시 보장해야 한다. 그리고 이어지는 버스클럭에 구동될 신호와의 충돌을 방지하기 위해 TP_4^{DTB} 를 보장해야 한다.

Table 6.3: 데이터 전송 버스 신호의 시간규격 요약

timing parameter	name	min	typical	max
TP_0^{BCLK}	BCLK* cycle time	59	60	61
TP_0^{DTB}	DTB signal latch point	38	40	42
TP_1^{DTB}	bus propagation time	-	-	20
TP_2^{DTB}	setup time	10	-	-
TP_3^{DTB}	hold time	5	-	-
TP_4^{DTB}	guard time	10	-	-

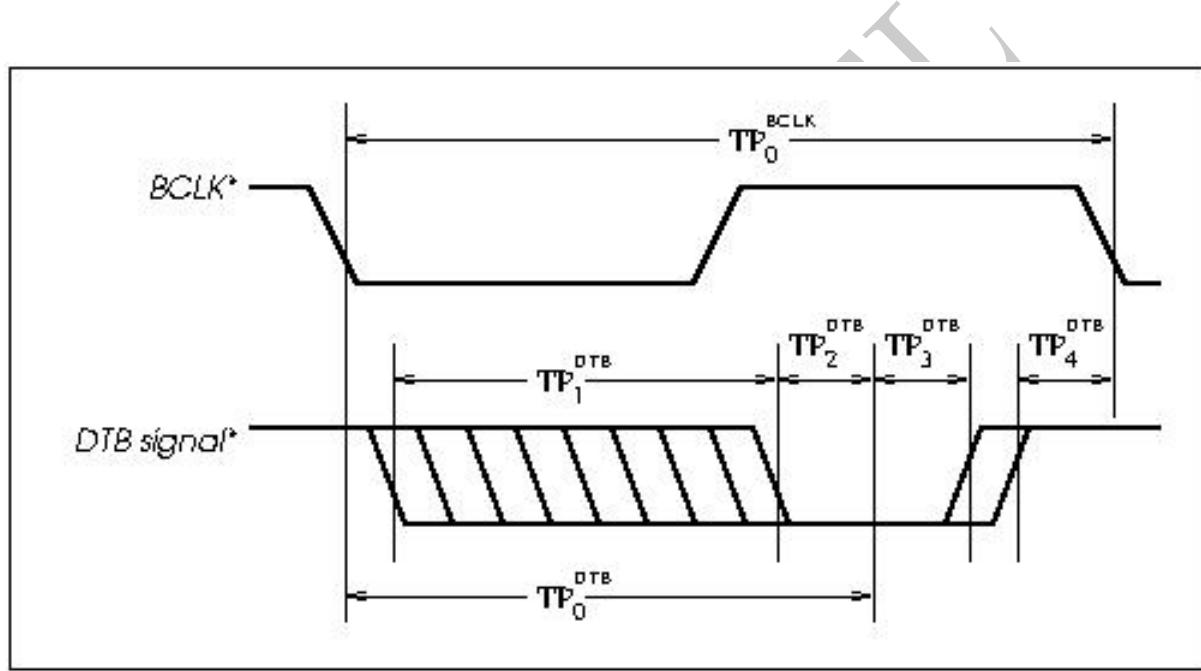


Figure 6.3: 데이터 전송버스 신호의 시간규격

6.5 인터럽트버스의 시간규격

6.5.1 IBSYNC* 신호의 시간규격

IBSYNC* 신호는 백플레인에 나타나는 버스클럭의 하강점에서 TP_0^{IBSYNC} 후에 안정된 신호를 래치할 수 있도록 셋업시간으로 TP_2^{IBSYNC} 을 홀드시간으로 TP_3^{IBSYNC} 를 보장해야 한다. IBSYNC* 신호를 “거짓(high voltage level)”로 하기 위해서는 구동을 멈추어야 하는데 이 때 다음 신호에 영향을 주지 못하도록 TP_4^{IBSYNC} 을 보장해야 한다. 여러 버스 사이를 동안 구동하는 경우, TP_1^{IBSYNC} 은 처음 사이클에서, TP_4^{IBSYNC} 은 마지막 사이클에서만 지키면 된다.

Table 6.4: IBSYNC* 신호의 시간규격 요약

timing parameter	name	min	typical	max
TP_0^{BCLK}	BCLK* cycle time	59	60	61
TP_0^{IBSYNC}	IBSYNC* latch point	38	40	42
TP_1^{IBSYNC}	bus propagation time	-	-	20
TP_2^{IBSYNC}	IBSYNC* setup time	10	-	-
TP_3^{IBSYNC}	IBSYNC* hold time	5	-	-
TP_4^{IBSYNC}	IBSYNC* guard time	10	-	-

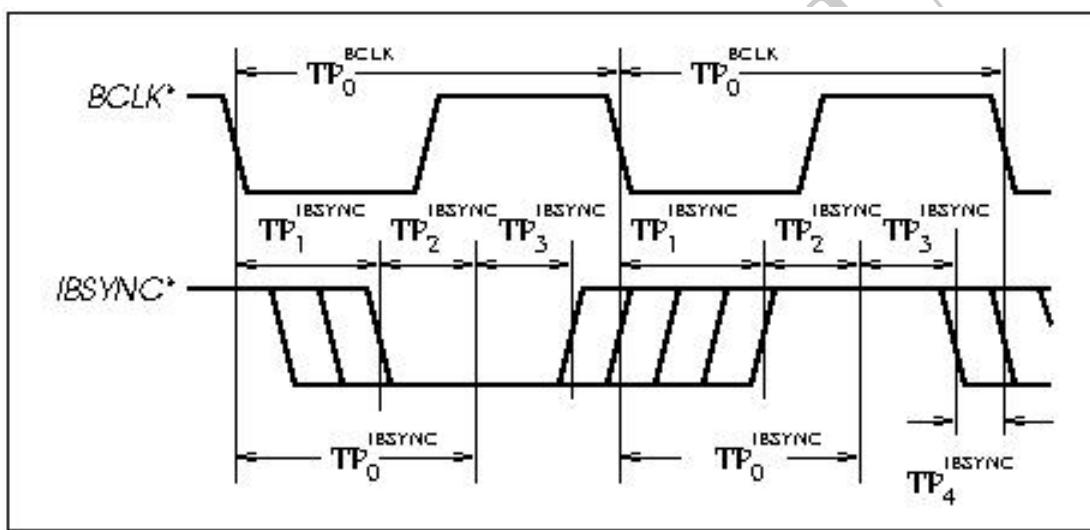


Figure 6.4: IBSYNC* 신호의 시간규격

6.5.2 인터럽트 버스 중재 동작의 시간 규격

중재 주기는 버스 클럭 (BCLK*)의 하강점부터 시작되며, $TP_0^{IBARB} + TP_1^{IBARB}$ 동안 신호를 구동해야 한다. 특히 중재결과는 TP_0^{IBARB} 시점에 래치할 수 있도록 안정된 신호를 보장해야 한다.

6.5.3 인터럽트 버스 인터럽트 데이터 전송 주기의 시간 규격

인터럽트 버스를 통하여 전송하고자 하는 데이터의 신호는 백플레인에 나타나는 버스클럭의 하강점에서 TP_0^{IBD} 후에 안정된 신호를 래치할 수 있도록 셋업시간으로 TP_2^{IBD} 을 허드시간으로 TP_3^{IBD} 를 보장하여야 한다. 그리고 앞과 뒤 신호에 영향을 주지 않도록 TP_4^{IBSYNC} 을 보장하여야 한다.

Table 6.5: 인터럽트 중재 버스 신호의 시간규격 요약

timing parameter	name	min	typical	max
TP_0^{BCLK}	BCLK* cycle time	59	60	61
TP_0^{IBARB}	arbitration time	240	242	244
TP_1^{IBARB}	post time	20	30	40
TP_2^{IBARB}	guard time	10	-	-

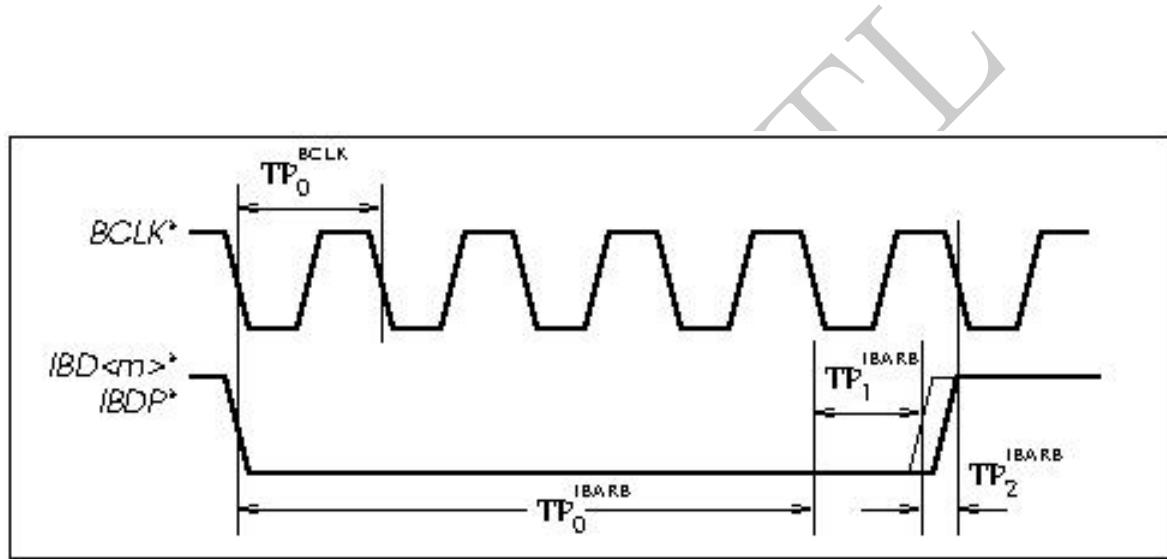


Figure 6.5: 인터럽트 버스 중재 동작의 시간규격

Table 6.6: 인터럽트 버스의 인터럽트 데이터 신호의 시간규격 요약

timing parameter	name	min	typical	max
TP_0^{BCLK}	BCLK* cycle time	59	60	61
TP_0^{IBD}	IBD<n>* signal latch point	38	40	42
TP_1^{IBD}	bus propagation time	-	-	20
TP_2^{IBD}	setup time	10	-	-
TP_3^{IBD}	hold time	5	-	-
TP_4^{IBD}	guard time	10	-	-

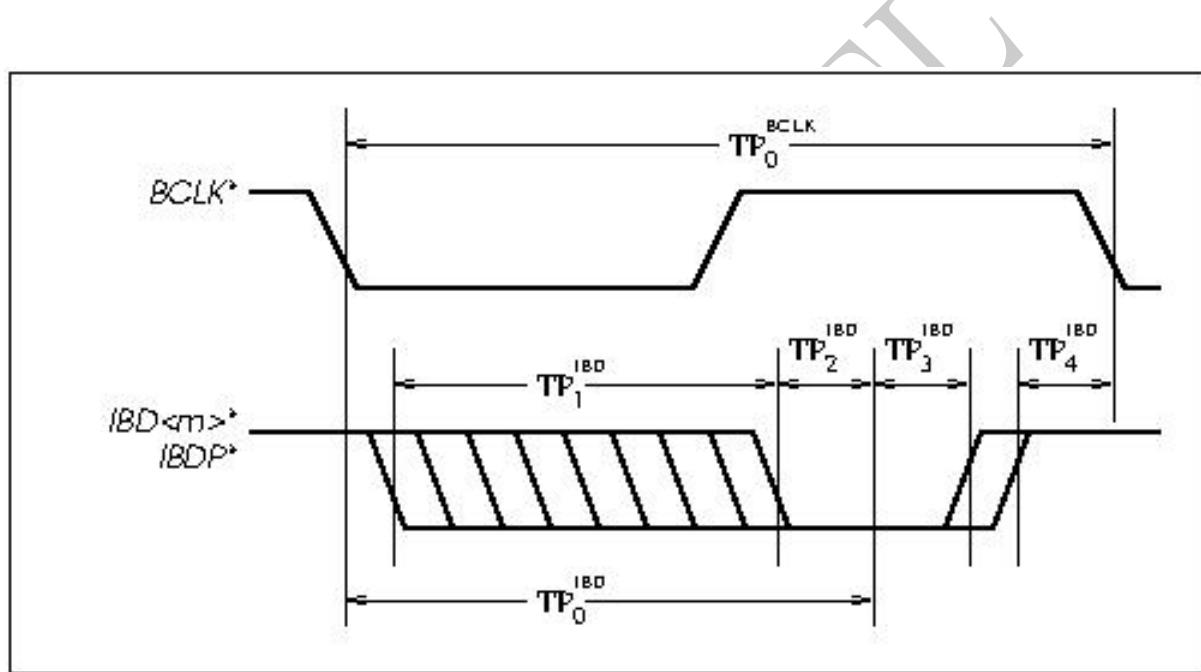


Figure 6.6: 인터럽트 버스의 데이터 전송의 시간규격

Chapter 7

전기적 규격

7.1 개요

전기적 규격에서는 버스에서 공급되는 전원에 대한 전원규격, 전달되는 신호의 전기적 규격을 규정한다.

Table 7.1: 전기적 규격 요약

전원 분배 (Power Distribution)	
전원의 종류 및 최대 허용 용량	+5V (30 A/slot), +5V STB (1 A/slot)
콘넥터 핀 당 최대 허용 전류	2 A
BTL 신호의 전기적 특성 (Characteristics of BTL Electrical Signal)	
최고 전압치 (Steady-state High Level)	1.9 V 이상 2.1 V 이하
최저 전압치 (Steady-state Low Level)	1.1 V 이하 0.0 V 이상
BTL 버스 드라이버의 특성 (Characteristics of BTL Bus Driver)	
기술 (Technology)	BTL (Backplane Transceiver Logic)
전류구동능력 (current driving capability)	80 mA @ V_{OL}
TTL 신호의 전기적 특성 (Characteristics of TTL Electrical Signal)	
최고 전압치 (Steady-state High Level)	2.5 V 이상
최저 전압치 (Steady-state Low Level)	0.6 V 이하
TTL 버스 드라이버의 특성 (Characteristics of TTL Bus Driver)	
기술 (Technology)	Advanced Schottky TTL
전류구동능력 (current driving capability)	64 mA @ V_{OL}
백플레인의 특성 (Characteristics of Backplane)	
특성 임피던스 (버스클럭/일반신호선)	50/80 Ω (홀이 없는 상태)
핀/슬롯 당 최대 허용 용량성 부하 (BTL 신호선)	15 pF
핀/슬롯 당 최대 허용 용량성 부하 (TTL 신호선)	25 pF
버스 터미네이터의 특성 (Characteristics of Bus Terminator)	
구성 소자	Resister, Capacitor
설치 위치	백프레인의 양끝

7.2 전원규격

백플레인에서 공급해야 되는 주전원으로는 5V TTL 소자를 위한 직류 +5V와 주전원과 무관하게 항상 동작할 필요가 있는 회로를 위해 백플레인에서 공급하는 직류 +5V STB 전원이 있다.

보드 내부 소자의 전원으로 공급되는 PGND와 이것과는 별도로 BTL 버스 구동소자의 기준 전압으로 SGND가 있다.

Table 7.2: 전원규격 요약

mnemonic	description	allowed variation	ripple/noise below 10MHz	maximum current per slot
+5V	+5V DC power	+0.25V/-0.125V	50mA	30A
+5V STB	+5V DC standby	+0.25V/-0.125V	50mA	1A
SGND	signal ground reference			
PGND	power ground reference			

7.3 콘넥터의 전기적 규격

백플레인에 사용하는 콘넥터는 핀과 소켓 형태를 갖는 것을 사용하는데 전원규격에서 정의한 규격을 만족하기 위해서는 <표 7.3>와 같은 규격을 유지해야된다.

Table 7.3: 핀의 전기적 규격

핀당 최대 허용 전류	2A
핀당 접촉저항	$20m\Omega$
최대 허용 전압	직류 200V
절연저항	$5000 M\Omega$

7.4 BTL 신호의 전기적 규격

BTL의 정확한 규격은 IEEE Std.1194.1을 따른다.

BTL 신호의 전기적 규격은 <표 7.4>와 같다. 신호의 참조전압을 GND 0V라 하고, 터미네이

Table 7.4: BTL 신호의 전기적 규격

V_t	typical	2.0V
V_{OH}	min	1.9V
V_{IH}	min	1.62V
V_{IL}	max	1.47V
V_{OL}	max	1.1V
V_{OL}	min	0.75V
GND		0.0V

션에 의한 전압이 V_t 이고, 신호가 구동되지 않은 상태에서의 신호선의 전압은 V_{OH}^{min} 이상이 되어야 하고, 신호선에 신호를 구동할 때 구동소자는 해당 신호선의 전압을 V_{OL}^{max} 와 V_{OL}^{min} 사이가 되도록 해야 한다. 수신소자의 경우는 V_{IH}^{min} 이상이 되는 전압에 대해서는 높은 전압으로, V_{IL}^{max} 이하의 전압에 대해서는 낮은 전압으로 받아들여야 한다. 이것을 그림으로 나타내면 <그림 7.1>과 같다.

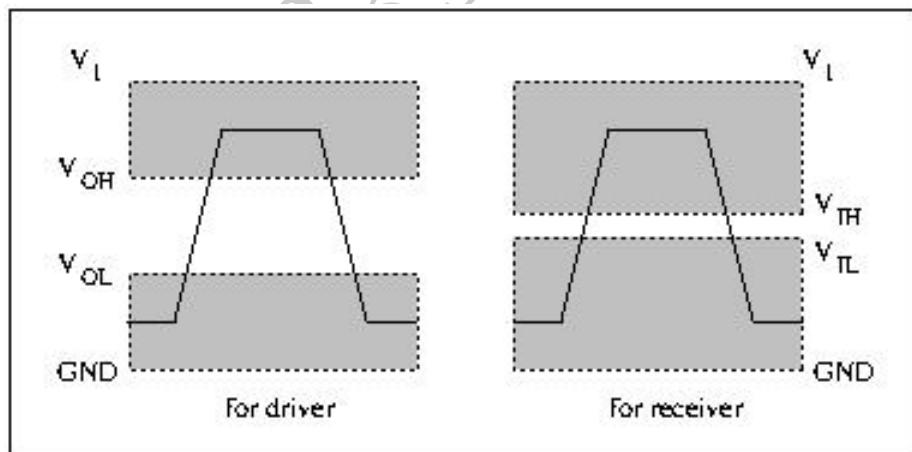


Figure 7.1: BTL 신호선의 전기적 규격

7.4.1 인터페이스 회로의 용량성 부하

BTL 버스에 연결되는 수신소자(receiver), 구동소자(driver), 수신/구동소자(transceiver)의 각 출력단의 용량성 부하는 5pF보다 크지 않아야 한다.

7.4.2 인터페이스 회로의 누설 전류

백플레인 신호선의 전압이 V_{OL} 영역에 있을 때, 해당 신호선에 연결된 인터페이스 소자 중 신호를 구동하지 않는 소자의 역방향 누설 전류(negative leakage current)가 $-250\mu A$ 보다 작아야 한다. 즉 최대 역방향 누설 전류(maximum negative leakage current)가 $-250\mu A$ 이다. 백플레인 신호선의 전압이 V_{OH} 영역에 있을 때, 해당 신호선에 연결된 인터페이스 소자는 제 7.4.3.3장에서 규정하는 최대 순방향 누설 전류(maximum positive leakage current)를 보장해야 한다.

7.4.3 구동소자

7.4.3.1 구동소자 형태

구동소자는 낮은 전압이 참이어야만 하고, wired-OR 기능이 있어야만 한다.

7.4.3.2 정적 부하 전류 (static load current)

백플레인 신호선의 전압이 V_{OL} 일 때 구동 소자는 80mA의 전류 흡수능력(current sinking capability)이 있어야 한다.

7.4.3.3 구동소자의 V_{OH}

V_{OH} 는 누설 전류, 터미네이션 전압, 부하저항, 그리고 백플레인에 연결된 소자의 수에 의해 결정된다. V_{OH}^{min} 는 1.9V이며, 이것은 식 (7.1)에 의해 정의된다. V_{OH}^{max} 는 누설 전류가 없는 상황에서 터미네이션 전압 V_t 가 된다.

$$V_{OH}^{min} = V_t - \frac{n \times I_l}{2} \times R_t \quad (7.1)$$

V_t : the termination voltage

I_l : the maximum positive leakage current

n : the number of devices on the bus (assuming all have the same I_l)

R_t : the termination resistor

7.4.3.4 구동소자의 V_{OL}

구동소자의 V_{OL} 은 전류 흡수가 80mA일 때 0.75V에서 1.1V 사이를 유지해야만 한다.

7.4.3.5 천이시간 (transition times)

10%에서 90%로의 상승 시간과 90%에서 10%로의 하강 시간은 10nsec보다 크지 않아야 한다. 그리고 잡음을 최소화하기 위해서는 구동소자가 출력 천이 시간을 3nsec보다 작지 않게 해야 한다.

7.4.4 수신소자

7.4.4.1 임계영역 (threshold)

백플레인 신호선에 연결된 모든 수신소자의 수신 임계영역 전압(receiver threshold voltage)은 염격히 지켜져야 한다. 1.47V 이하인 경우는 낮은 전압으로, 1.62V 이상이면 높은 전압으로 규정한다. 앞에서 규정한 좁은 임계영역전압을 보장함으로써 높은 잡음 허용한계를 얻을 수 있다.

7.4.4.2 잡음제거 (noise rejection)

잡음의 영향을 최소화 하기 위해 3nsec 미만의 펄스를 제어하는 필터를 수신측에 허용한다.

7.4.4.3 입력 클램프 (input clamps)

과도한 역저압에 의한 영향을 방지하기 위해 1.5V에 적어도 12mA의 클램핑 회로가 수신소자의 입력단에 있어야 한다.

7.5 TTL 신호의 전기적 규격

TTL 신호의 전기적 규격은 <표 7.5>와 같다. 신호의 참조전압을 GND 0V라 하고, 터미네이

Table 7.5: TTL 신호의 전기적 규격

V_t	typical	2.5V
V_{OH}	min	2.0V
V_{IH}	max	1.8V
V_{IL}	min	0.8V
V_{OL}	max	0.6V
GND		0.0V

션에 의한 전압이 V_t 이고, 신호가 구동되지 않은 상태에서의 신호선의 전압은 V_{OH}^{min} 이상이 되어야 하고, 신호선에 신호를 구동할 때 구동소자는 해당 신호선의 전압을 V_{OL}^{max} 와 V_{OL}^{min} 사이가 되도록 해야 한다. 수신소자의 경우는 V_{IH}^{min} 이상이 되는 전압에 대해서는 높은 전압으로, V_{IL}^{max} 이하의 전압에 대해서는 낮은 전압으로 받아들여야 한다. 이것을 그림으로 나타내면 <그림 7.2>과 같다.

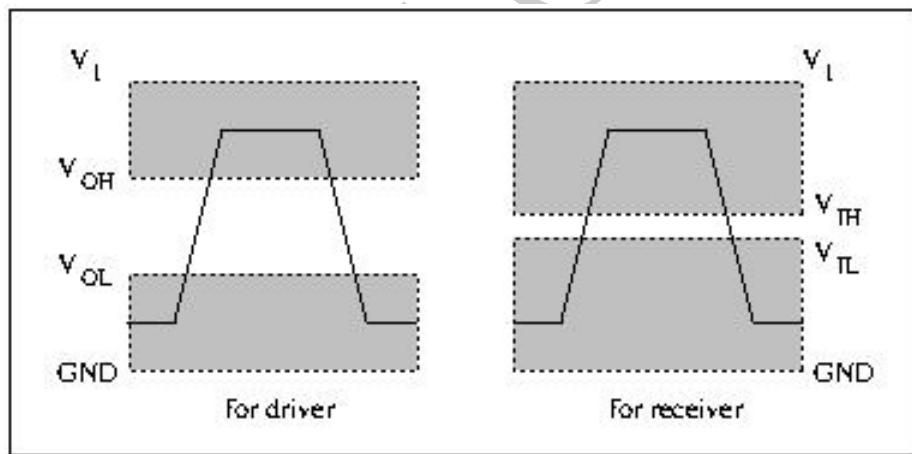


Figure 7.2: TTL 신호선의 전기적 규격

7.5.1 인터페이스 회로의 용량성 부하

TTL 버스에 연결되는 수신소자(receiver), 구동소자(driver), 수신/구동소자(transceiver)의 각 출력단의 용량성 부하는 15pF보다 크지 않아야 한다.

7.5.2 구동소자의 종류

TTL 구동소자의 종류는 totem-pol, three-state, 그리고 open-collector 등 세 가지가 있다.

7.6. 백플레인의 특성

7.6.1 백플레인의 특성임피던스

백플레인의 버스클럭 신호선의 특성임피던스는 50Ω 을 유지해야 하고, 나머지 백플레인 신호선의 특성임피던스는 콘넥터를 끊기 위한 훈이 없는 상태에서 80Ω 이어야 하고, 훈과 콘넥터를 연결하였을 때 60Ω 이상이어야 하고, 모든 슬롯에 보드가 끊힌 상태에서 20Ω 이상을 유지해야 한다.

7.6.2 터미네이션

백플레인에는 두 가지 터미네이션 방법을 사용한다. 버스클럭의 경우 [그림 7.3.a]과 같이 직렬터미네이션 방법을 사용하고, 나머지 신호선들은 [그림 7.3.b]과 같이 병렬터미네이션 방법을 사용한다.

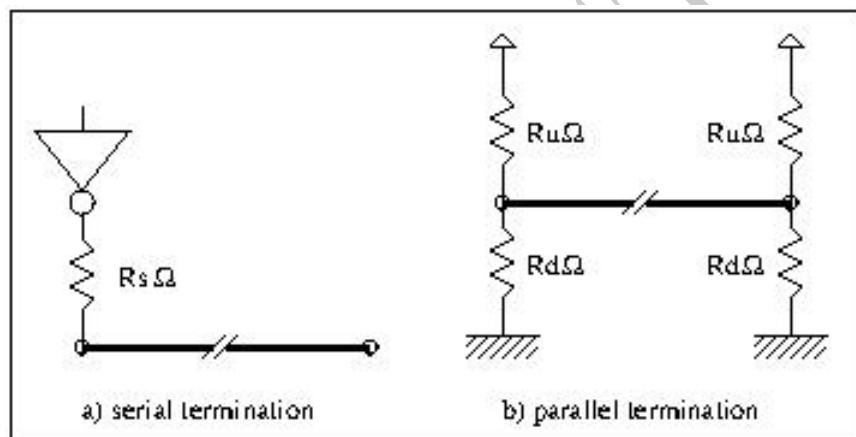


Figure 7.3: 터미네이션 방법

Chapter 8

기계적 구조

8.1 개요

버스의 기계적 구조는 백플레인(backplane), 백플레인에 꽂혀 사용될 보드(board), 백플레인과 보드를 지지하는 서브랙(subrack), 각 보드에서 필요로 하는 주전원 전력을 균등하게 공급하도록 하는 파워 바(power bar), 그리고 백플레인과 보드를 연결하는 콘넥터의 구조를 정의한다.

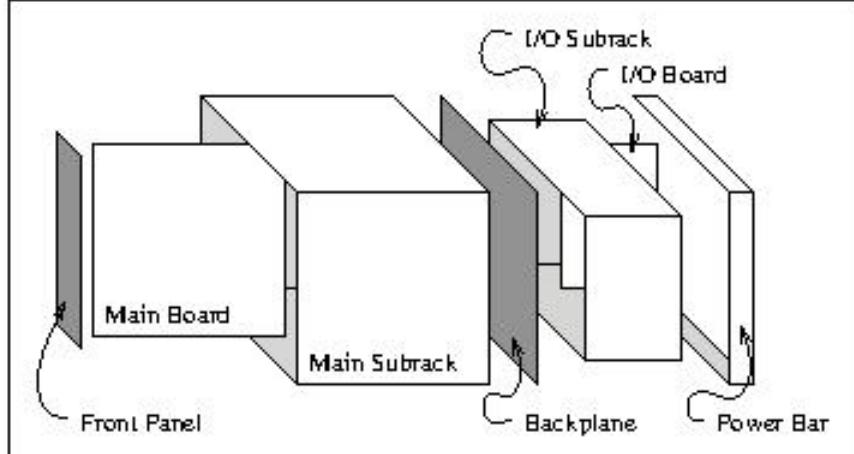


Figure 8.1: 기계적 장치의 개요

- front panel (전면판넬) : HiPi+Bus main board의 전면에 설치되는 전면 판넬이다.
- main board (주보드) : HiPi+Bus 백플레인에 꽂히게 될 보드이다.
- main subrack (주서브랙) : main board가 백플레인의 슬롯에 꽂힐때 제대로 꽂히도록 지지하는 기구물이다.
- backplane (백플레인) : HiPi+Bus 백플레인이다.

- I/O subrack (입출력서브랙) : I/O board가 백플레인의 뒷쪽 슬롯에 꽂힐 때 제대로 꽂히도록 지지하는 기구물이다.
- I/O board (입출력보드) : HiPi+Bus 백플레인 뒷쪽 슬롯에 꽂하게 될 보드로 I/O bus 와의 연결을 용이하게 하기 위한 보드이다.
- power bar (파워바) : HiPi+Bus 백플레인의 뒷쪽에 설치되어 백플레인을 통해 백플레인에 꽂히는 모든 보드에 전원을 공급하고 분배하는 장치이다.

8.2 백플레인

백플레인의 크기는 $525.78\text{mm} \times 486.3\text{mm} \times 4\text{mm}$ (width \times height \times thickness)이다. 백플레인에는 보드가 꽂힐 슬롯, 파워바가 연결되어 전원이 공급될 접점, 주서브렉과 입출력서브렉이 고정될 지지영역등이 있다. 그리고 버스클럭 발생회로가 있고, 터미에이션 저항이 설치될 영역이 있다. 백플레인에는 보드가 꽂힐 21개 슬롯(슬롯 0 - 20)이 있고, 각 슬롯에는 보드 와의 연결을 위한 백플레인 콘넥터(간접형 숫 콘넥터)가 설치된다. 백플레인을 전면에서 볼 때 가장 왼쪽부터 슬롯 0번(J0), 그 다음이 슬롯 1번(J1) 순이며 가장 오른쪽이 슬롯 20번(J20)이다. 각 슬롯 간의 간격은 0.8 inch이며, 따라서 슬롯 0번에서 슬롯 20번까지의 거리는 16 inch(406.4 mm)이다. 백플레인은 슬롯에 꽂힌 보드들 사이의 신호 전송의 통로를 제공하고 또한 슬롯을 통하여 각 보드에 전원을 공급한다. 각 보드에 공급될 +5V 전원은 파워 바(Power Bar)를 통하여 전원 공급 장치로부터 공급받으며 이 파워 바와의 연결을 위해 백플레인의 위 아래에 각각 20개씩의 접점(+5V용 20개, GND용 20개)이 있다. 또한 +5V STB 전원용 파워 탭(Power Tab) 설치를 위한 접점이 있다. 백플레인은 주서브렉의 뒷면에 설치되며, 그 상세한 규격은 도면으로 첨부한다.

8.3 주보드 (Main Board)

주보드는 백플레인 앞쪽에서 주서브렉안으로 꽂힌다. 주보드의 크기는 $400\text{mm} \times 455.6\text{mm} \times 2\text{mm}$ 이다. 보드에는 내부 신호선들과 백플레인의 신호선들을 연결하기 위한 보드 콘넥터(board connector, 암 콘넥터) 설치를 위한 영역과, 서브렉에 설치할 때의 가이드 레일(guide rail)을 위한 여유 영역과 전면 판넬(front panel) 설치를 위한 여유 영역이 있고, 이들을 제외한 모든 영역에 부품이 위치할 수 있다. 보드의 상세한 규격은 도면으로 첨부한다.

8.4 입출력보드 (I/O Board)

입출력보드는 백플레인 뒷쪽에서 입출력서브렉안으로 꽂힌다. 입출력보드의 크기는 $127\text{mm} \times 362.2\text{mm} \times 2\text{mm}$ 이다. 보드에는 내부 신호선들과 백플레인의 신호선들을 연결하기 위한 입출력 보드 콘넥터 (I/O board connector, 암 콘넥터) 설치를 위한 영역과, 서브렉에 설치할 때의 가이드 레일(guide rail)을 위한 여유 영역이 있고, 이들을 제외한 모든 영역에 부품이 위치할 수 있다. 보드의 상세한 규격은 도면으로 첨부한다.

Table 8.1: 백플레인의 기계적 규격

외관상 규격	
Size	525.78mm×486.3mm×4mm
Number of Slot	21 (J0 - J20)
Interval between neighbor slots	0.8 inch
Interval from J0 to J20	16 inch
Degree of PCB layer	
Signal	4
Clock	1
+5	2
SGND	1
PGND	2
Total	10
전원 공급용 접점수	
+5V	20
GND	20
+5V Standby	1

8.5 파워 바 (Power Bar)와 파워 탭 (Power Tab)

파워 바(Power Bar)는 백플레인에 꽂히는 21개 보드에 주전원인 +5V, GND 전원을 균등하게 분배되도록 하기 위해 전원 장치로부터 백플레인 사이에 위치하는 전원 분배용 도체 기구물이며, 그 상세한 규격은 도면으로 첨부한다.

백플레인에는 +5V STB 전원을 연결하기 위한 접점이 있고 이 접점에 설치되는 것이 파워 탭이다.

8.6 주서브랙 (Main Subrack)

백플레인이 고정되고, 백플레인에 꽂힐 주보드들을 지지하기 위한 주서브랙에는 보드가 꽂힐 때 용이함을 위해 가이드 레일(guide rail)이 설치된다. 서브랙의 상세한 규격은 도면으로 첨부한다.

8.7 입출력서브랙 (I/O Subrack)

백플레인이 뒤쪽에 고정되고, 백플레인에 꽂힐 입출력보드들을 지지하기 위한 입출력서브랙에는 보드가 꽂힐 때 용이함을 위해 가이드 레일(guide rail)이 설치된다. 서브랙의 상세한 규격은 도면으로 첨부한다.

8.8 콘넥터

시스템 버스 백플레인과 보드를 연결하기 위해 간접형 콘넥터(Indirect Connector)를 사용한다. 콘넥터는 340핀(85핀 × 4줄)(전원 핀을 따로) 주콘넥터와 160핀(40핀 × 4줄)(전원 핀 포함) 입출력 콘넥터로 백플레인 쪽에는 슛콘넥터(버스 콘넥터), 보드 쪽에는 암콘넥터(보드 콘넥터)를 사용한다. 특히, 버스 콘넥터는 press-fit 형태의 콘넥터를 사용하며, 각슬롯의 아래쪽 160핀용 콘넥터는 백플에인 뒤쪽에서 보드가 설치 될 수 있도록 다리가 긴 콘넥터를 사용한다.

Appendix A

HiPi+Bus 신호선과 핀할당

ETRI-CSTL

A.1 HiPi+Bus 신호선들

Bus	Mnemonic	Size	Name
중재버스	ABREQ<12..0>*	13	Address Bus Request
	ABINH*	1	Address Bus Arbitration Inhibition
	WRINH*	1	Write Cycle Inhibition
	DBREQ<8..0>*	9	Data Bus Request
	DBINH*	1	Data Bus Arbitration Inhibition
	PCW*	1	Priority Change Window
데이터전송버스 (어드레스버스)	A<31..4>*	28	Address
	AP<3..0>*	4	Address Parity
	SI<7..0>*	8	Source Identification
	SIP*	1	Source Identification Parity
	AS<2..0>*	3	Address Space
	TT<4..0>*	5	Transfer Types
	STP*	1	Space + Types Parity
	BE<15..0>*	16	Byte Enable
	BEP<1..0>*	2	Byte Enable Parity
	AE*	1	Address Cycle Enable
데이터전송버스 (데이터버스)	D<127..0>*	128	Data
	DP<15..0>*	16	Data Parity
	DI<7..0>*	8	Destination Identification
	DIP*	1	Destination Identification Parity
	DE*	1	Data Cycle Enable
데이터전송버스 (상태버스)	AACK<1..0>*	2	Address Acknowledge
	SHD*	1	Hit on Shared Line
	DTY*	1	Hit on Dirty Line
	SNK*	1	Snoop No Acknowledge
	ITV*	1	Intervention
	LCR*	1	Hit on Interlocked Region
	DACK*	1	Data Acknowledge
	CDK*	1	Cache Data Acknowledge
	SPIN<3..0>*	4	Spin lock list number
인터럽트 전송버스	BSY<7..0>*	8	Busy Status Line
	IBSYNC*	1	Interrupt Bus Sync.
	IBD<7..0>*	8	Interrupt Bus Data
유틸리티버스	IBDP*	1	Interrupt Bus Data Parity
	BCLK*	1	Bus Clock
	RST*	1	System Reset
	SFAIL*	1	System Fail
	GA<4..0>*	5	Geographical Slot Address
	Tx<4..0>	5	Boundary Scan Option
	<i>total</i>	293	

A.2 HiPi+Bus 핀할당

	A	B	C	D
1	-	-	+5V STB	+5V STB
2	RST*	SFAIL*	-	-
3	GA<0>*	GA<1>*	GA<2>*	GA<3>*
4	GA<4>*	-	-	Tx<0>
5	Tx<1>	Tx<2>	Tx<3>	Tx<4>
6	-	-	-	-
7	ABRQ<0>*	ABRQ<1>*	ABRQ<2>*	ABRQ<3>*
8	ABRQ<4>*	ABRQ<5>*	ABRQ<6>*	ABRQ<7>*
9	ABRQ<8>*	ABRQ<9>*	ABRQ<10>*	ABRQ<11>*
10	ABRQ<12>*	ABINH*	WRINH*	-
11	DBRQ<0>*	DBRQ<1>*	DBRQ<2>*	DBRQ<3>*
12	DBRQ<4>*	DBRQ<5>*	DBRQ<6>*	DBRQ<7>*
13	DBRQ<8>*	DBINH*	PCW*	-
14	-	-	-	-
15	AE*	-	-	-
16	A<4>*	A<5>*	A<6>*	A<7>*
17	AP<0>*	A<8>*	A<9>*	A<10>*
18	A<11>*	A<12>*	A<13>*	A<14>*
19	A<15>*	AP<1>*	A<16>*	A<17>*
20	A<18>*	A<19>*	A<20>*	A<21>*
21	A<22>*	A<23>*	AP<2>*	A<24>*
22	A<25>*	A<26>*	A<27>*	A<28>*
23	A<29>*	A<30>*	A<31>*	AP<3>*
24	SI<0>*	SI<1>*	SI<2>*	SI<3>*
25	SI<4>*	SI<5>*	SI<6>*	SI<7>*
26	SIP*	AS<0>*	AS<1>*	AS<2>*
27	TT<0>*	TT<1>*	TT<2>*	TT<3>*
28	TT<4>*	STP*	BE<0>*	BE<1>*
29	BE<2>*	BE<3>*	BE<4>*	BE<5>*
30	BE<6>*	BE<7>*	BEP<0>*	BE<8>*

	A	B	C	D
31	BE<9>*	BE<10>*	BE<11>*	BE<12>*
32	BE<13>*	BE<14>*	BE<15>*	BEP<1>*
33	-	-	-	-
34	AACK<0>*	AACK<1>*	SHD*	DTY*
35	SNK*	ITV*	LCR*	-
36	DACK*	CDK*	-	-
37	SPIN<0>*	SPIN<1>*	SPIN<2>*	SPIN<3>*
38	BSY<0>*	BSY<1>*	BSY<2>*	BSY<3>*
39	BSY<4>*	BSY<5>*	BSY<6>*	BSY<7>*
40	-	BCLK*	-	-
41	DE*	-	-	-
42	D<0>*	D<1>*	D<2>*	D<3>*
43	D<4>*	D<5>*	D<6>*	D<7>*
44	DP<0>*	D<8>*	D<9>*	D<10>*
45	D<11>*	D<12>*	D<13>*	D<14>*
46	D<15>*	DP<1>*	D<16>*	D<17>*
47	D<18>*	D<19>*	D<20>*	D<21>*
48	D<22>*	D<23>*	DP<2>*	D<24>*
49	D<25>*	D<26>*	D<27>*	D<28>*
50	D<29>*	D<30>*	D<31>*	DP<3>*
51	D<32>*	D<33>*	D<34>*	D<35>*
52	D<36>*	D<37>*	D<38>*	D<39>*
53	DP<4>*	D<40>*	D<41>*	D<42>*
54	D<43>*	D<44>*	D<45>*	D<46>*
55	D<47>*	DP<5>*	D<48>*	D<49>*
56	D<50>*	D<51>*	D<52>*	D<53>*
57	D<54>*	D<55>*	DP<6>*	D<56>*
58	D<57>*	D<58>*	D<59>*	D<60>*
59	D<61>*	D<62>*	D<63>*	DP<7>*
60	D<64>*	D<65>*	D<66>*	D<67>*

	A	B	C	D
61	D<68>*	D<69>*	D<70>*	D<71>*
62	DP<8>*	D<72>*	D<73>*	D<74>*
63	D<75>*	D<76>*	D<77>*	D<78>*
64	D<79>*	DP<9>*	D<80>*	D<81>*
65	D<82>*	D<83>*	D<84>*	D<85>*
66	D<86>*	D<87>*	DP<10>*	D<88>*
67	D<89>*	D<90>*	D<91>*	D<92>*
68	D<93>*	D<94>*	D<95>*	DP<11>*
69	D<96>*	D<97>*	D<98>*	D<99>*
70	D<100>*	D<101>*	D<102>*	D<103>*
71	DP<12>*	DP<104>*	D<105>*	D<106>*
72	D<107>*	D<108>*	D<109>*	D<110>*
73	D<111>*	DP<13>*	D<112>*	D<113>*
74	D<114>*	D<115>*	D<116>*	D<117>*
75	D<118>*	D<119>*	DP<14>*	D<120>*
76	D<121>*	D<122>*	D<123>*	D<124>*
77	D<125>*	D<126>*	D<127>*	DP<15>*
78	DI<0>*	DI<1>*	DI<2>*	DI<3>*
79	DI<4>*	DI<5>*	DI<6>*	DI<7>*
80	DIP*	-	-	-
81	-	-	-	-
82	IBD<0>*	IBD<1>*	IBD<2>*	IBD<3>*
83	IBD<4>*	IBD<5>*	IBD<6>*	IBD<7>*
84	IBDP*	IBSYNC*	-	-
85	-	-	-	-

- 주의 1 : 미정의 핀은 이웃 콘넥터 핀과 연결되어 있다.

	A	B	C	D
1	Vcc	Vcc	Vcc	Vcc
2	-	-	-	-
3	-	-	-	-
4	-	-	-	-
5	-	-	-	-
6	-	-	-	-
7	-	-	-	-
8	-	-	-	-
9	-	-	-	-
10	-	-	-	-
11	-	-	-	-
12	-	-	-	-
13	-	-	-	-
14	-	-	-	-
15	-	-	-	-
16	-	-	-	-
17	-	-	-	-
18	-	-	-	-
19	-	-	-	-
20	-	-	-	-
21	-	-	-	-
22	-	-	-	-
23	-	-	-	-
24	-	-	-	-
25	-	-	-	-
26	-	-	-	-
27	-	-	-	-
28	-	-	-	-
29	-	-	-	-
30	-	-	-	-
31	-	-	-	-
32	-	-	-	-
33	-	-	-	-
34	-	-	-	-
35	-	-	-	-
36	-	-	-	-
37	-	-	-	-
38	-	-	-	-
39	-	-	-	-
40	GND	GND	GND	GND

- 주의 1 : 미정의 핀은 이웃 콘넥터 핀과 연결되지 않는다.

Appendix B

약어 일람표

A Address	CONT Controller
AACK Address Acknowledge	D Data
AAR Address Arbitration	DACK Data Acknowledge
AARB Address Arbitration Bus	DAR Data Arbitration
ABINH Address Bus Arbitration Inhibition	DARB Data Arbitration Bus
ABRQ Address Bus Request	DB Data Bus
AB Address Bus	DBINH Data Bus Arbitration Inhibition
AE Address Cycle Enable	DBRQ Data Bus Request
AP Address Parity	DE Data Cycle Enable
ARB Arbitration Bus	DI Destination Identification
AS Address Space	DIP Destination Identification Parity
AS-TTL Advanced Schottky TTL	DP Data Parity
BCLK Bus Clock	DTB Data Transfer Bus
BE Byte Enable	DTY Hit on Dirty Line
BEP Byte Enable Parity	FAST-TTL Fairchild AS TTL
BSY Busy Status Line	FF Flip Flop
BTL Backplane Transceiver Logic	GA Geographical Slot Address
CDK Cache Data Acknowledge	HiPi-Bus Highly Pipelined Bus
	HiPi+Bus Highly Pipelined Plus Bus

APPENDIX B. 약어 일람표

IB	Interrupt Bus	RP	ResPonder
IBD	Interrupt Bus Data	RQ	ReQuester
IBDP	Interrupt Bus Data Parity	RST	System Reset
IBSYNC	Interrupt Bus Sync.	SB	Status Bus
IEEE	The Institute of Electrical and Electronics Engineers	SCM	System Controller Module
IOP	Input Output Processor	SFAIL	System Fail
ITV	Intervention	SHD	Hit on Shared Line
I/O	Input Output	SI	Source Identification
JTAG	Joint Test Action Group	SIP	Source Identification Parity
LCR	Hit on Interlocked Region	SNK	Snoop No Acknowledge
MAI	Mega Arbitration Interrupts per second	SPIN	Spin Queue Order
MDI	Mega Direct Interrupts per second	STP	Space + Types Parity
MEM	Main Memory Unit	TPG	Timing Pulse Generator
MI	Mega Interrupts per second	TT	Transfer Types
MPU	Main Processing Unit	TTL	Transistor Transistor Logic
PAC	Packaging	Tx	JTAG Boundary Scan Option
PCW	Priority Change Window	UB	Utility Bus
		WRINH	Write Cycle Inhibition