AMBA™ Specification (AHB)

By GH Lee ver. 1.2

Seoul National Univ.

Design Automation Lab.

http://poppy.snu.ac.kr

Hynix Project Document ver. 2002-01

Chapter 1 Introduction to the AMBA Buses

1.1 Overview of the AMBA specification

1.2 A typical AMBA-based microcontroller

1.3 Introducing the AMBA AHB

1.1 Overview of the AMBA specification

Advanced Microcontroller Bus Architecture (AMBA)는 embedded microcontroller의 설계를 위한 on-chip communication의 표준안이 되고 있다.

AMBA의 Specification에는 3가지가 있다.

The Advanced High-performance Bus (AHB)

The Advanced System Bus (ASB)

The Advanced Peripheral Bus (APB)

1.2 A typical AMBA-based microcontroller

AMBA-based microcontroller는 일반적으로 AHB나 ASB를 backbone bus로 사용한다. 여기에는 ARM과 같은 CPU나 DMA등이 연결될 수 있다.

또한 peripheral macrocell communication을 위해서 APB가 붙을 수 있는데 이것은 ASB나 AHB의 local bus처럼 여겨지며 Bridge를 사용하여 연결된다.

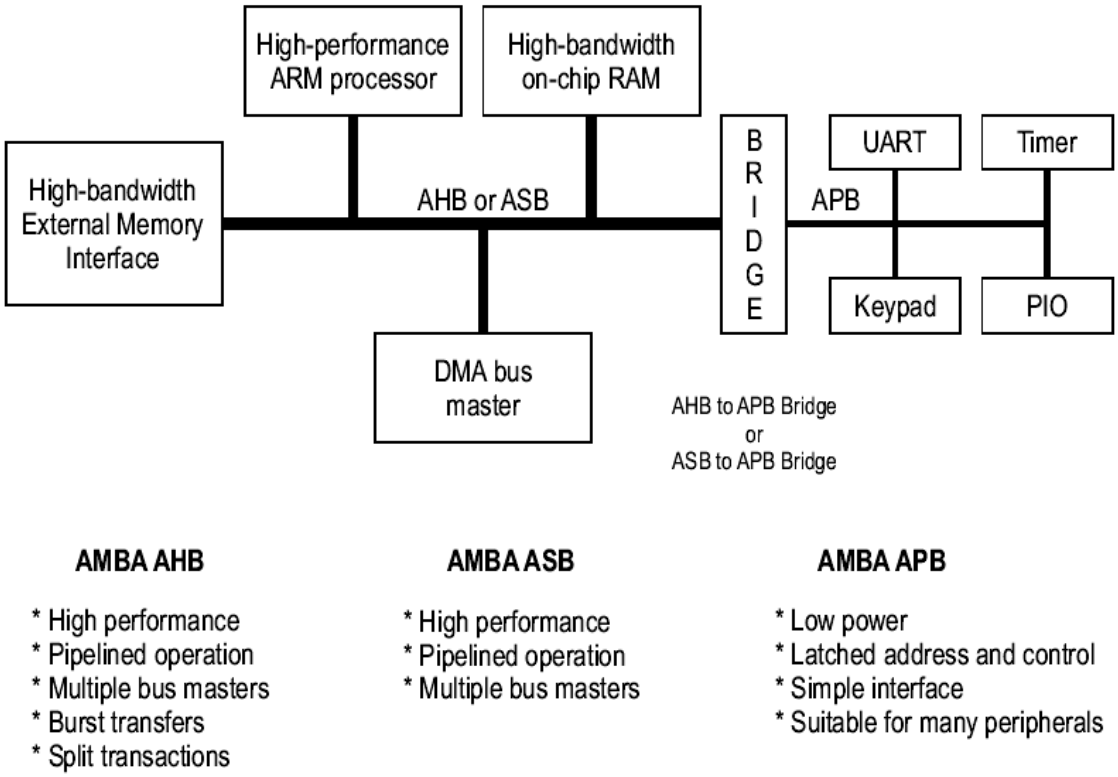

Figure 1-1 typical AMBA system

이 중에서 가장 널리 사용되고 있는 AMBA AHB에 대해서 다음 장에 걸쳐서 자세히 알아보고자 한다.

1.3 Introducing the AMBA AHB

AHB는 High-performance를 위한 AMBA bus의 새로운 제안이다. 이를 위한 AMBA AHB의 특성은 다음과 같다.

Burst transfer

Split transaction

Single-cycle bus master handover

Single-clock edge operation

Non-tristate implementation (separate WDATA and RDATA)

Wider data bus configurations (64/128 bits)

AMBA AHB에는 하나 또는 그 이상의 bus master가 연결된다. 이러한 것들에는 DMA나 DSP같은 것들이 연결 될 수 있다.

External memory interface나 APB bridge등은 AMBA AHB의 slave로 연결 될 수 있다. 그러나 low-bandwidth를 갖는 peripheral 장치들은 APB를 통해서 연결이 되어야 할 것이다.

AMBA AHB system을 구성하는 요소는 다음과 같다.

구성요소	설명
AHB Master	Bus Master는 Address나 control signal들을 내보냄으로 read나 write의 operation을 할 수 있도록 해 주는 장치이다. 한번에 하나의 Master만이 전송을 가능하게 한다.
AHB Slave	Bus Slave는 주어진 address-space안에서 read와 write를 가능하게 해주는 장치이다. Slave는 ready등의 signal을 통해서 master로 하여금 기다리게 하거나 전송이 잘 못되었음을 알린다.
AHB arbiter	Bus Arbiter는 한번에 오직 하나의 Master가 선택되도록 하는 역할을 한다. 고유의 priority algorithm을 가지고 이러한 arbitration을 하게 되는데, AHB에는 오직 하나의 arbiter가 존재하게 된다.
AHB decoder	AHB decoder의 역할은 Master로 나오는 Address의 상위 비트를 가지고서 적절한 slave를 선택해 주는 것이다. AHB에는 역시 하나의 Decoder가 존재한다.

Chapter 2 AMBA Signals

2.1 AMBA signal names

AMBA의 signal들에는 항상 첫 글자에 특정한 letter가 사용된다. 또한 n이라는 문자가 사용되는 것은 active LOW라는 의미를 가지고 있다. 그 이외의 signal들은 모두 대문자로 표현되며 active HIGH의 의미를 갖는다.

2.1.1 AHB signal prefixes

H AHB signal임을 의미한다

2.1.2 ASB signal prefixes

B ASB signal임을 의미한다

A ASB의 arbiter와 master간에 사용되는 signal임을 의미한다

D ASB의 decoder에 사용되는 signal임을 의미한다

2.1.3 APB signal prefixes

P APB signal임을 의미한다.

예를 들어서 HREADY라 함은 AHB의 HIGH에서 동작하는 signal임을 의미하고, BnRES라 함은 ASB의 LOW에서 동작하는 reset signal임을 알 수 있다.

2.2 AMBA AHB signal list

이번 장에서는 AHB에 사용되는 signal들의 대략적인 설명을 볼 수 있다. 이 후에 나오는 장에서 더욱 자세히 살펴 볼 수 있을 것이다.

Signal 이름을 잘 살펴보면 항상 앞에 H라는 letter가 붙어 있음을 확인 할 수 있다. 이것은 AHB의 signal임을 의미하는 것이다.

Table 2-1 AMBA AHB Signals

Name	Source	Description
HCLK	Clock	AHB의 모든 clock은 이 signal에 동기화 되어 있으며 항상 rising edge에서 동작한다
HRESETn	Reset	유일하게 LOW에서 동작하는 signal로서 AHB의 reset역할을 한다.
HADDR[31:0]	Master	32-bits address bus
HTRANS[1:0]	Master	NonSequential, Sequential, Idle, Busy등의 transfer의 성격을 나타낸다.
HWRITE	Master	HIGH일 때는 write이고, LOW일 때는 read를 의미한다.
HSIZE[2:0]	Master	Transfer의 data size를 의미한다. 보통 byte(8-bits), halfword(16-bits), word(32-bits)등으로 나타나는데 AHB에서는 최고 1024 bits까지 지원한다.
HBURST[2:0]	Master	어떤 burst mode인지를 알려주는 역할을 한다. 4, 8, 16또는 1개의 burst를 지원하며, incrementing또는 wrapping으로 나누어 진다.
HPROT[3:0]	Master	추가적인 정보로서 현재의 transfer가 opcode fetch인지 data access인지 등을 알려주어 전송을 보호하는 역할을 한다. 이것 외에도 cacheable, bufferable등의 정보도 알려준다.
HWDATA[31:0]	Master	Write operation일 때 사용되며, 최소의 data bus size는 32 bits를 추천한다.
HSELx	Decoder	각각의 Slave는 고유의 Select 신호를 가지게 되며 이것은 Decoder를 통하여 선택이 된다. Transfer동안에 어떠한 slave가 선택 되어질 것인가를 알려주는 signal이다.

Table 2-1 AMBA AHB Signals (continued)

Name	Source	Description
HRDATA[31:0]	Slave	Slave로부터 master에게 전해지는 data를 받는 signal로서 32 bits를 추천한다.
HREADY	Slave	HIGH일 때에는 전송이 제대로 이루어 졌음을 의미하고 LOW일 때에는 transfer를 extend하기를 원한다는 것을 의미한다.
HRESP[1:0]	Slave	Transfer에 대한 추가적인 정보이다. OKAY, ERROR, RETRY, SPLIT으로 구분된다.

AMBA는 여러 개의 Master를 연결해서 사용하기 때문에 Arbitration과정이 필요하게 된다. 이를 위해서 사용되는 signal들이 Table 2-2에 나와있다. 여기서 suffix x가 의미하는 바는 X라는 module로부터 나오는 signal임을 의미한다. 예를 들어 Master0으로부터 나오는 HBUSREQx는 HBUSREQ0으로 표현될 것이다.

Table 2-2 AMBA AHB Arbiter Signal Descriptions

Name	Source	Description
HBUSREQx	Master	Bus Request 신호는 bus Master가 bus에 대한 access를 요청할 때 사용된다. 각각의 bus Master는 각각 HBUSREQx라는 interface를 가지게 되며, Arbiter는 최고 16개의 Master를 다룰 수 있다. 즉HBUSREQ[15:0]까자 사용할 수 있다.
HLOCKx	Master	Lock 신호는 bus Request 신호와 동시에 인가된다. HLOCKx 신호는 master가 burst trans를 하려고 할 때, 중간에 다른 Master에게 bus의 소유를 넘기지 않을 것을 위해서 사용된다. Locked transfer의 첫번째 전송이 시작되면 Arbiter는 현재의 전송을 끝마치기 전에는 다른 Master에게 bus의 사용권을 넘겨주지 않을 것이다.
HGRANTx	Arbiter	Grant 신호는 arbiter에 의해서 발생된다. 이것은 가장 높은 priority를 가진 master에게만 HIGH를 준다. 그렇지만 Grant를 얻었다고 해서 바로 전송을 시작할 수 있는 것은 아니다. HREADY가 HIGH일 때에만 비로소 전송을 시작할 수 있게 된다.
HMASTER[3:0]	Arbiter	현재 어떤 Maser가 선택 되어져 있는 가는 HMASTER[3:0] 신호를 이용해서 알 수 있다. 이것은 master들이 연결되는 MUX에 사용 되어진다. Maser number는 또한 SPLIT-capable slave에 의해서도 주어진다. 이것은 Master로 하여금 SPLIT 전송을 마무리 할 수 있도록 하는데 사용된다.
HMASTLOCK	Arbiter	HMASTLOCK 신호는 현재 전송하는 것이 locked transfer인지를 알려준다. 이것은 address와 control signal과 동시에 나오게 된다.
HSPLIT[15:0]	Slave (SPLIT-capable)	16bit의 Split bus는 SPLIT-capable slave에 의해서 주어진다. 이를 통해서 SPLIT 전송을 마무리 할 수 있도록 하는데 사용된다.

Chapter 3 AMBA AHB

3.1 About the AMBA AHB

AHB는 High-performance를 위한 AMBA bus의 새로운 제안이다. 이를 위한 AMBA AHB의 특성은 다음과 같다.

Burst transfer

Split transaction

Single-cycle bus master handover

Single-clock edge operation

Non-tristate implementation (separate WDATA and RDATA)

Wider data bus configurations (64/128 bits)

3.2 Bus interconnection

AMBA의 design에는 Multiplexor, arbiter, decoder가 있어 여러 개의 Master와 Slave를 중개해 주는 역할을 한다. Figure 3-1에 이에 대한 그림이 나와 있다.

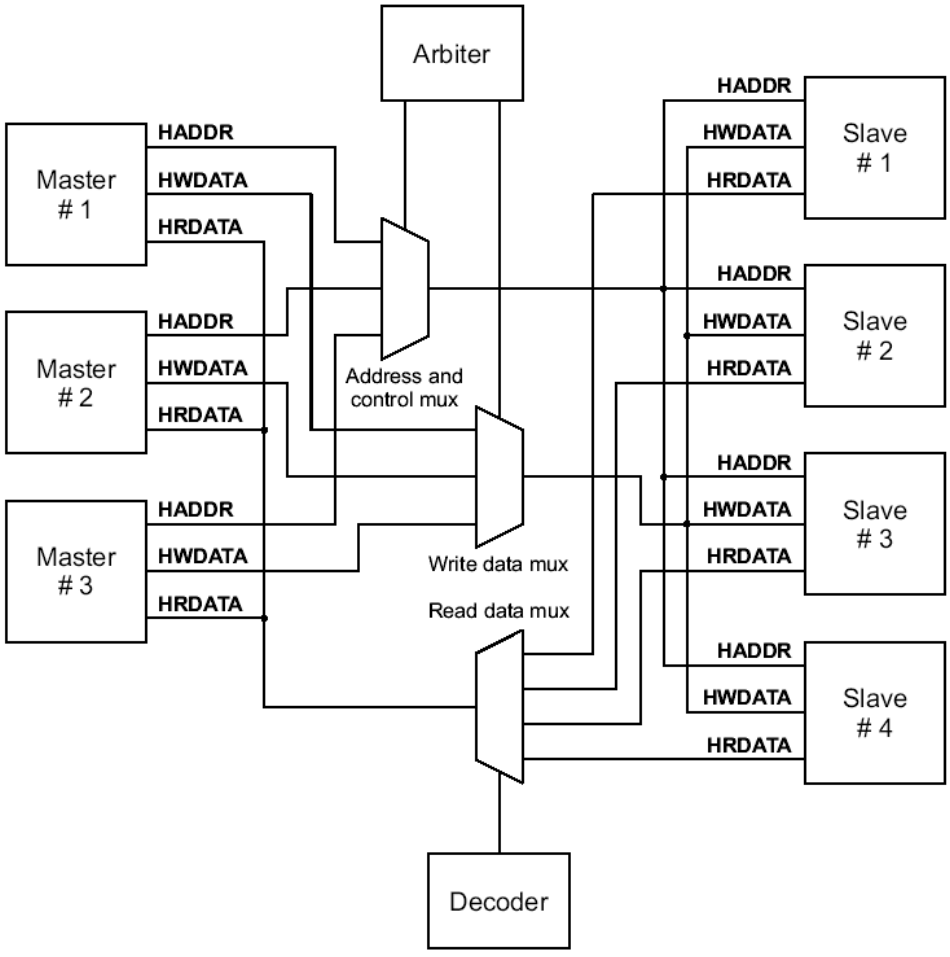

Figure. 3-1 Multiplexor interconnection

3.3 Basic transfer

AHB의 transfer에는 2가지 중요한 사실이 있다.

Address phase에 있어서 오직 single cycle만으로 구성된다.

Data phase에서는 HREADY signal을 사용하여 몇 cycle동안의 데이터의 extend가 가능하다.

Figure 3-2에 가장 간단한 transfer에 대한 그림이 나와있다. 이 그림에는 HREADY에 의한 wait는 나와 있지 않다.

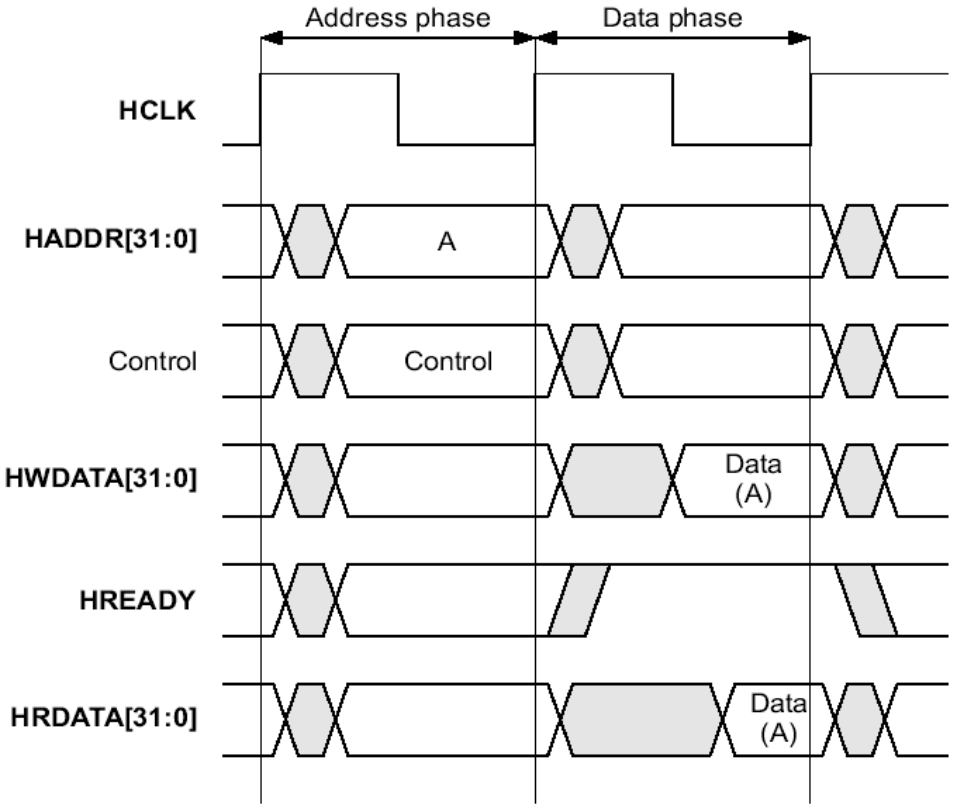

Figure. 3-2 Simple transfer

Master는 항상 HCLK의 rising edge에서 address와 control signal을 전송하게 된다. 그 다음의 rising edge에서 slave는 address와 control signal을 sampling하게 되고 그 다음의 rising edge에서 해당 data를 받는다.

서로 다른 clock period를 갖는 개체 사이에서 전송을 하기 위해서 Slave는 HREADY signal

을 사용하여 data를 extend시키는데 이에 대한 그림이 Figure 3-3에 나와있다.

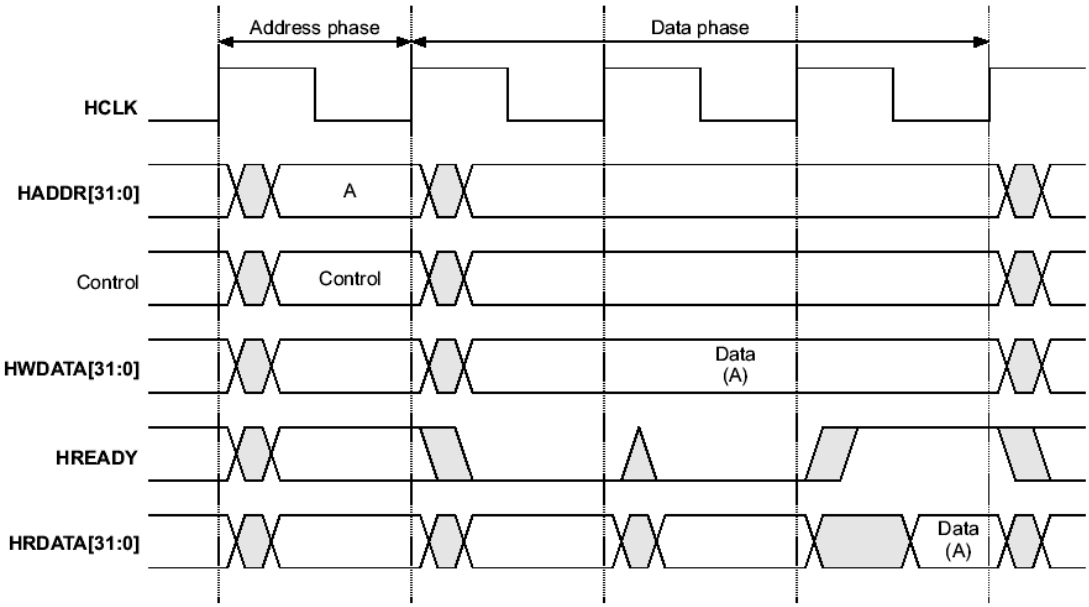

Figure 3-3 Transfer with wait states

여기에서 주의 할 것은 write transfer일 경우에는 data phase의 전체 cycle에 걸쳐서 HWDATA에 data를 담게 되는데, read transfer일 경우에는 HREADY가 HIGH가 되는 부분에서만 HRDATA의 data가 유효하다는 것이다.

서로 상관이 없는 address들의 전송에 대한 그림이 Figure 3-4에 나와있다. 여기서는 A, B, & C의 Address에 대한 전송을 보여주고 있다.

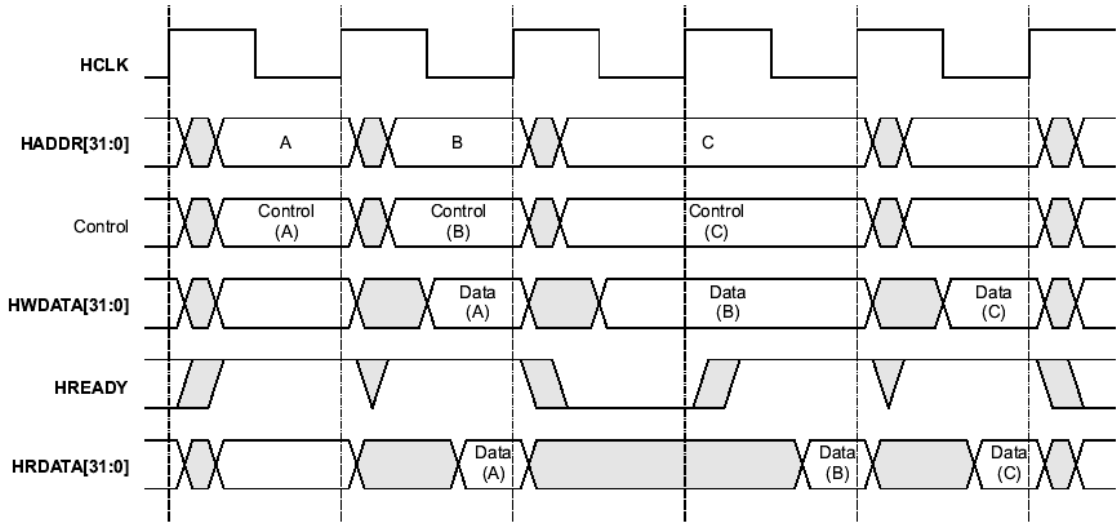

Figure. 3-4 Multiple transfer

A와 C는 zero wait state로 전송이 이루어 졌고, B는 one wait state의 전송이 이루어 졌음을 볼 수 있다.

B의 data를 받는데 사용되는 cycle이 2cycle이 됨에 따라서 C의 Address를 전송하는 것도 2cycle이 된 것을 확인 할 수 있다.

AHB는 이와 같이 pipe-lined transfer가 가능하다.

3.4 Transfer type

모든 transfer에는 HTRANS[1:0]이라는 신호를 받으면서 진행된다. 이에 대한 설명이 Table 3-1에 나와있다.

Table 3-1 Transfer type encoding

HTRANS[1:0]	Type	Description
00	IDLE	아무런 전송이 필요하지 않음을 의미한다. 즉 해당 master에게 grant가 이루어 졌지만 아무런 전송을 하지 않으려 할 때에 IDLE transfer를 하게 된다. Slave는 zero wait state의 OKAY response를 하며 transfer는 slave에 의해서 무시되어진다.
01	BUSY	BUSY transfer는 burst transfer의 중간에 IDLE cycle을 넣으려고 할 때 나타난다. 즉 master가 계속하여 burst transfer를 하려고 하지만 즉시 다음 전송이 이루어 질 수 없는 상황에서 BUSY을 나타낸다. BUSY의 cycle에서는 현재의 모든 control signal들이 다음 cycle로 전달되며 BUSY 상태의 전송은 slave에 의해서 모두 무시 되어야 한다. Slave는 IDLE 상태와 같이 zero wait state의 OKAY response를 해야 한다.
10	NONSEQ	Burst transfer의 첫번째 전송을 의미하거나 single transfer를 의미한다. 또한 현재 Address와 control signal들은 전의 transfer와 상관이 없음을 알려준다.
11	SEQ	Burst transfer의 나머지는 모두 SEQUENTIAL로 나타나며 이것은 전의 address와 control이 현재의 transfer와 상관이 있다는 것을 알려준다. 그리하여 현재의 address는 전의 address에서 transfer size 만큼이 더해져서 나타나게 된다.

Figure 3-5에 몇 가지 transfer type이 사용된 예를 보여준다.

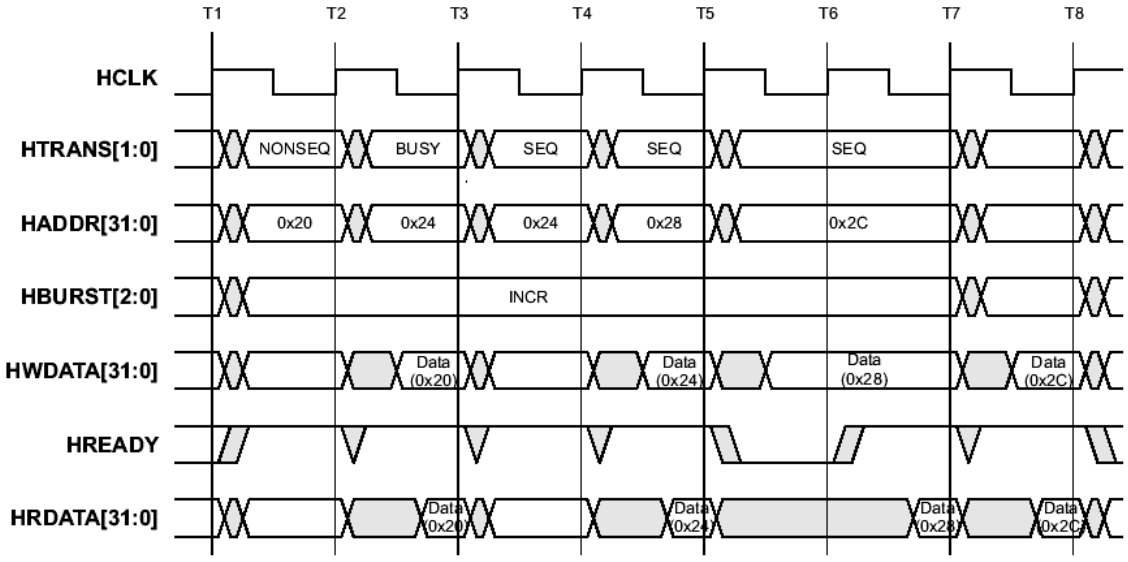

Figure. 3-5 Transfer type example

위의 그림에서 첫 번째 전송은 NONSEQ로 시작한 것을 볼 수 있다.
Master가 전송을 바로 할 수 없어서 BUSY transfer를 통하여 그 다음의 전송을 한 cycle delay 시킨 것을 볼 수 있다.
세 번째 전송이 되려고 할 때 이번엔 slave쪽에서 받을 수 없어서 HREADY를 통해 한 cycle의 delay를 시킨 것을 볼 수 있다.
마지막 전송은 zero wait state로 마무리 되었다.

3.5 Burst operation

4, 8, 또는 16개의 burst transfer가 AMBA AHB protocol에서 정의 된다. 또한 1개의 전송도 가능하며, incrementing이나 wrapping burst mode가 지원된다.

Incrementing burst는 address가 순차적으로 burst size에 따라서 커지는 것을 의미한다. 그러므로 다음의 address는 전의 address에서 size만큼 더해 주면 된다.

Wrapping burst에서는 size*beat 만큼의 address boundary가 설정이 된다. 그러므로 Address가 incrementing burst와 같이 증가하다가 boundary에 이르면 다시 밑의 boundary에서 시작하게 된다.

Table 3-2에 8가지 가능한 burst mode에 대해서 설명해 놓았다.

Table 3-2 Burst signal encoding

HBURST[2:0]	Type	Description
000	SINGLE	Single transfer
001	INCR	Incrementing burst of unspecified length
010	WRAP4	4-beat wrapping burst
011	INCR4	4-beat incrementing burst
100	WRAP8	8-beat wrapping burst
101	INCR8	8-beat incrementing burst
110	WRAP16	16-beat wrapping burst
111	INCR16	16-beat incrementing burst

Bursts는 1kB의 address boundary를 넘지 못한다. 그러므로 master가 fixed length로 incrementing burst를 할 때 boundary를 넘지 않도록 하는 것은 중요하다.

Incrementing burst는 어떠한 길이의 transfer라도 1kB의 boundary를 넘지 않는 선에서 얼마든지 가능하다.

3.5.1 Early burst termination

상황에 따라서는 burst의 마지막 전송을 끝내기 전에 전송이 termination되는 경우가 있다. HTRANS signal을 통해서 slave는 burst transfer가 종료 되었는지에 대한 것을 판단 할 수 있다.

중간에 전송이 종료 되었다면 master는 나머지 전송을 위해서 나머지 개수에 맞는 새로운 burst를 시작해야 할 것이다.

Figure 3-6에서부터 Figure 3-10에 이르기까지 각 burst mode에 따른 example을 보여주고 있다.

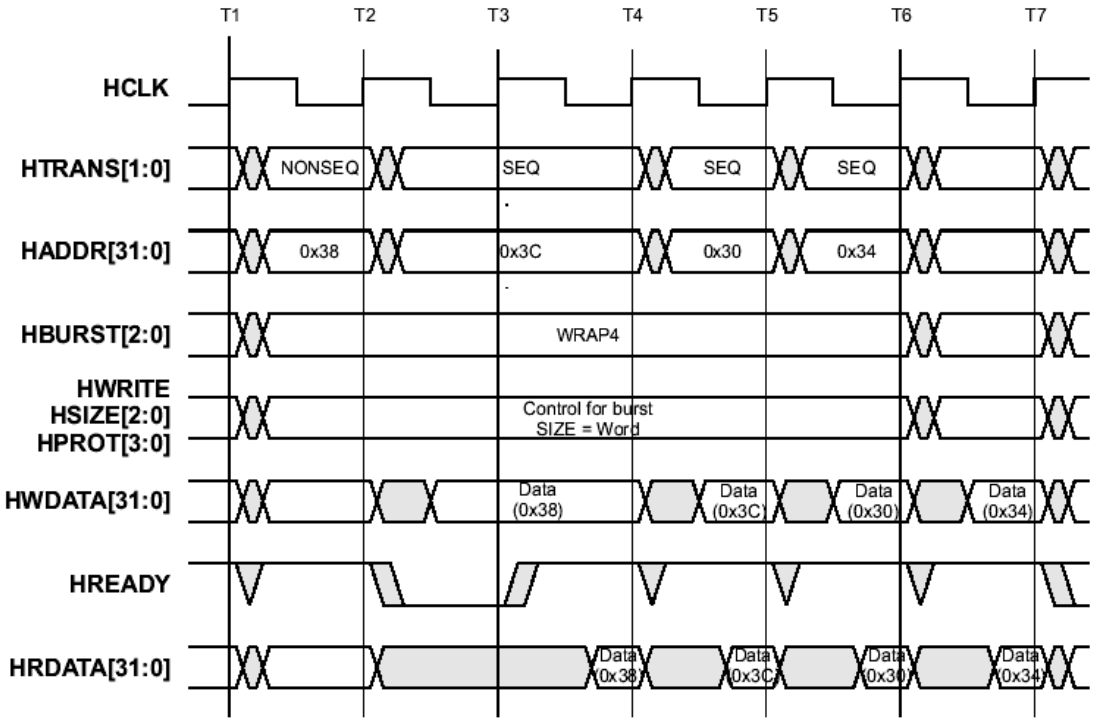

Figure 3-6 Four-beat wrapping burst

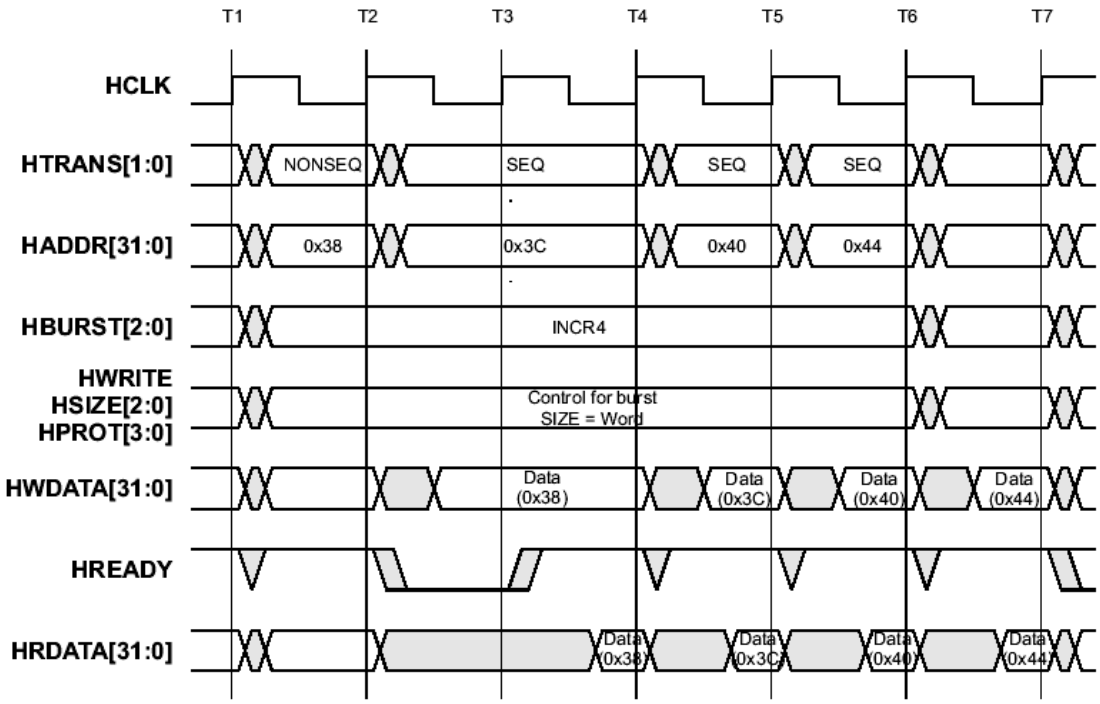

Figure 3-7 Four-beat incrementing burst

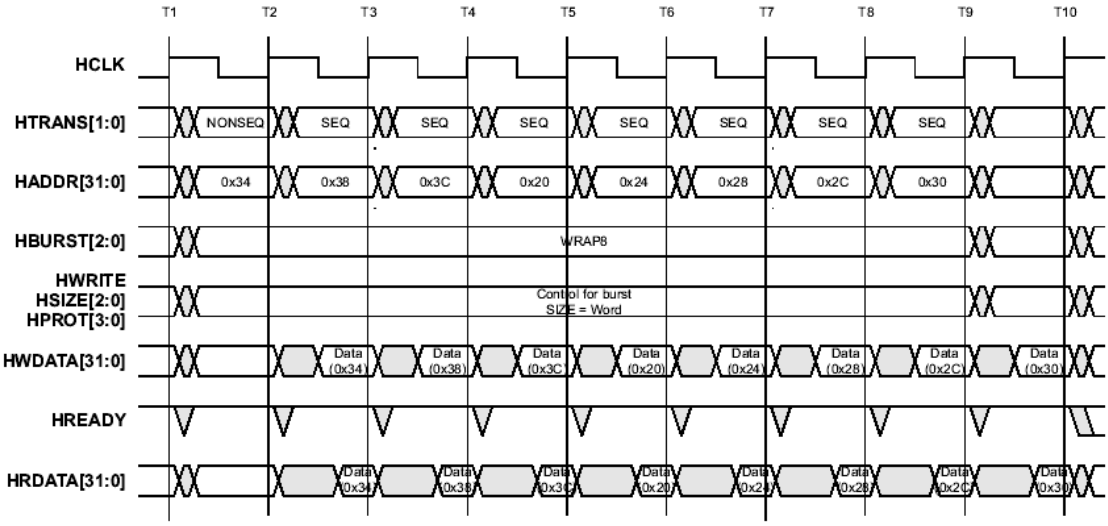

Figure 3-8 Eight-beat wrapping burst

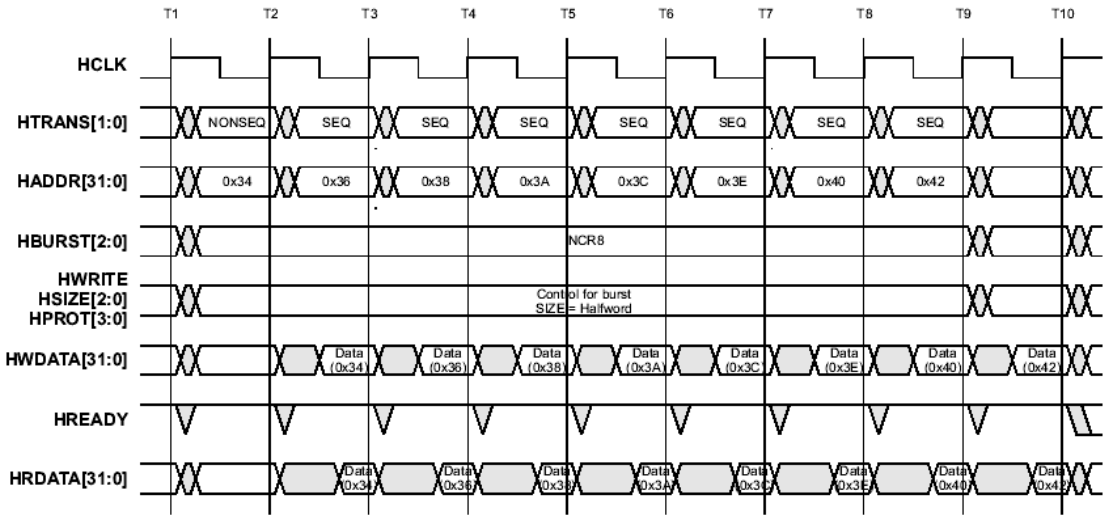

Figure 3-9 Eight-beat incrementing burst

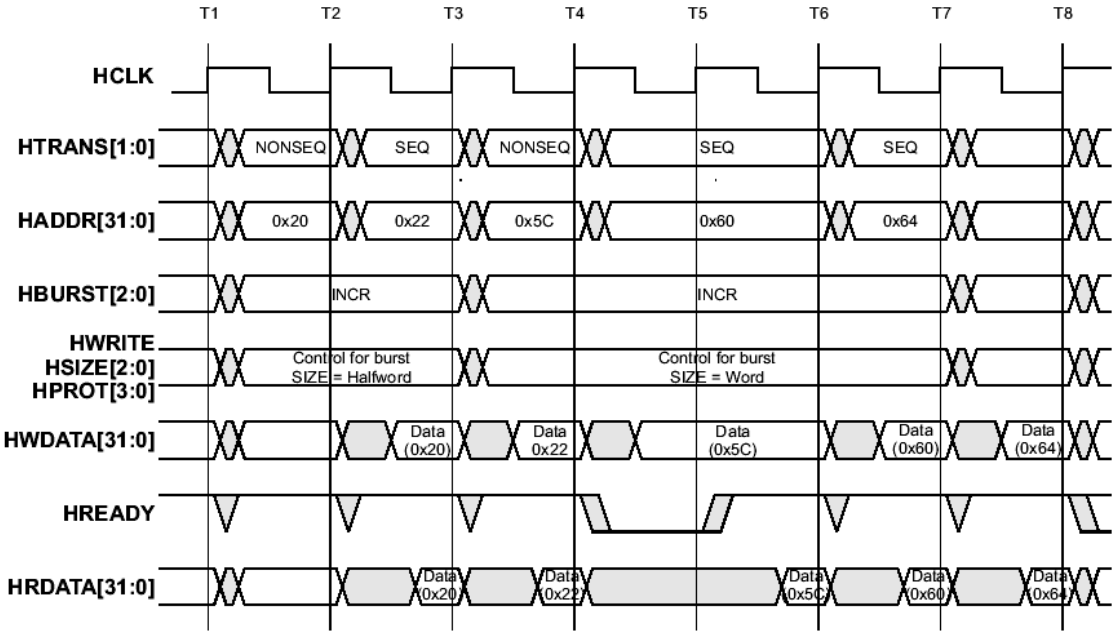

Figure 3-10 Undefined-length bursts

3.6 Control signals

burst type외에도 transfer를 이루기 위해서 몇 가지 control signal들이 사용된다. Transfer의 방향을 설정해 주는 HWRITE와 transfer size를 알려주는 HSIZE[1:0]과 protection control을 위해서 사용되는 HPROT[3:0]이 그것이다.

3.6.1 Transfer direction

HWRITE가 HIGH일 때, 이 signal은 HWDATA[31:0]을 사용하여 write하는 과정임을 알려준다. HWRITE가 LOW일 때는 HRDATA[31:0]을 사용하여 read하는 과정임을 알려준다.

3.6.2 Transfer size

HSIZE[2:0]은 table 3-3과 같은 transfer size를 갖는 transfer임을 알려주는 signal이다.

Table 3-3 Size encoding

HSIZE[2:0]	Size	Description
000	8 bits	Byte
001	16 bits	Half word
010	32 bits	Word
011	64 bits	
100	128 bits	4-word line
101	256 bits	8-word line
110	512 bits	
111	1024 bits	

3.6.3 Protection control

HPROT[3:0]은 opcode fetch인지 data access인지 또는 privileged mode access인지 user mode인지 등을 알려주기 위한 것이다.

이에 대한 설명이 Table 3-4에 나와있다.

Table 3-4 Protection signal encoding

HPROT[3] cacheable	HPROT[2] bufferable	HPROT[1] privileged	HPROT[0] Data/opcode	Description
–	–	–	0	Opcode fetch
–	–	–	1	Data access
–	–	0	–	User access
–	–	1	–	Privileged access
–	0	–	–	Not bufferable
–	1	–	–	Bufferable
0	–	–	–	Not cacheable
1	–	–	–	cacheable

그러나 이러한 protection mode에도 불구하고 HPROT의 사용을 하지 않는 것을 권장한다. 그러므로 slave는 HPROT의 정보를 사용할 필요는 없다.

3.7 Address decoding

각각의 slave에 붙어있는 HSELx의 signal을 enable해주기 위한 장치가 decoder이다. Select signal은 high-order address signals을 사용하여 구현된다.

주의하여야 할 것은 slave는 HREADY가 HIGH일 때에만 HSELx의 signal을 바꾸어야 한다는 것이다. 경우에 따라서는 HREADY가 LOW일 때 HSELx signal을 바꾸어도 되지만 이 경우에는 Slave쪽에서 마지막 전송이 끝날 때에만 선택되는 slave가 바뀌어야 한다.

하나의 slave가 가질 수 있는 address의 boundary는 1kB이다. Bus master는 1kB의 boundary를 넘어서 burst를 할 수 없도록 design되어 있기 때문에 이러한 설정이 가능하다.

이 경우에도 default slave가 있어야 하는데 master가 잘못된 영역의 address에 쓰려고 하는 경우에 이 default slave가 선택이 되어야 하며 선택된 slave는 SEQUENTIAL이나 NONSEQUENTIAL일 경우에는 ERROR response를 IDLE이나 BUSY일 경우에는 OKAY response를 내보내 주어야 한다.

Figure 3-11에 address를 decoding하는 과정을 보여주고 있다.

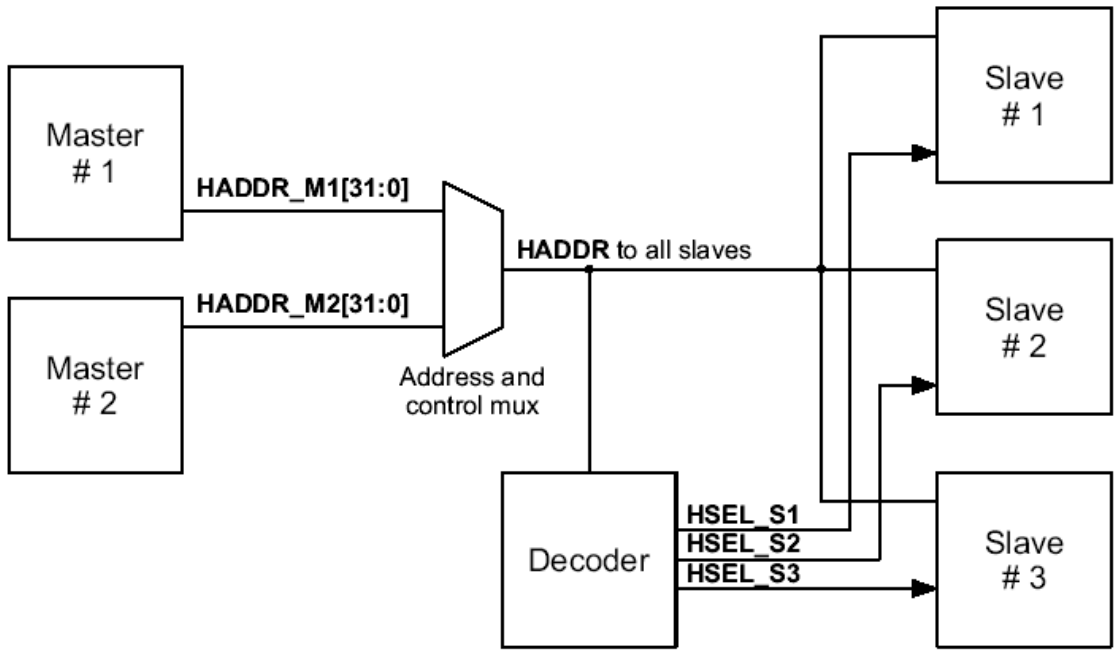

Figure 3-11 Slave select signals

3.8 Slave transfer responses

일단 master가 transfer를 시작하면 slave는 이 transfer가 어떻게 진행되어야 할지를 결정해 주어야 한다.

Slave가 선택이 되어지면 slave는 HREADY와 HRESP[1:0]을 통하여 transfer에 대한 response를 하게 된다.

Slave가 몇 가지 방법으로 transfer를 마무리 할 수 있다.
즉시 transfer를 완성한다.
한 개 또는 그 이상의 wait state를 준 후에 transfer를 완성한다.
현재의 전송이 잘못 되었음을 알리고 transfer를 끝낸다.
다른 transfer를 위해서 현재의 transfer를 다음으로 미룬다.

3.8.1 Transfer done

HREADY를 통해서 AHB는 transfer를 extend시키거나 끝을 알릴 수 있다. 그러나 extend되는 것에는 미리 예측 할 수 있는 Maximum이 있어서 그 이상을 넘어가면서 HREADY를 LOW로 지속 할 수는 없다. 권장되는 maximum wait states는 16이다.

3.8.2 Transfer response

HREADY가 HIGH일 때와 OKAY response는 transfer가 정상적으로 이루어 졌음을 의미한다. 그리고 ERROR response는 ROM에다 쓰려고 하는 등 잘못된 address에 데이터를 access 하려고 할 때 나타난다.

SPLIT과 RETRY는 transfer를 delay하기 위한 것이다. 이에 대한 자세한 설명은 다음 장에 있다.

Table 3-5에 HRESP[1:0]에 대한 설명이 나와 있다.

Table 3-5 Response encoding

HRESP[1:0]	Type	Description
00	OKAY	OKAY response와 HREADY가 HIGH이면 정상적으로 전송이 이루어 졌음을 의미한다. 또한 OKAY를 통하여 추가적인 cycle을 삽입하기 위하여 사용되기도 한다.
01	ERROR	Master에게 transfer가 Error가 발생하였고 더 이상 같은 영역에 transfer를 계속 할 수 없음을 알린다. Error condition을 위해서 2-cycle이 필요하다.
10	RETRY	Transfer가 아직 성공적으로 끝나지 않았음을 알리고 transfer를 다시 할 것을 알리는 response이다. Retry condition을 위해서 2-cycle이 필요하다.
11	SPLIT	Transfer가 아직 성공적으로 끝나지 않았음을 알리고 다른 transfer를 원하는 master에게 그 소유권을 넘겨주기 위한 response이다. Split condition을 위해서 2-cycle이 필요하다.

3.8.3 Two-cycle response

오직 OKAY response만이 single cycle동안에 이루어지고 다른 response는 모두 two-cycle 동안에 이루어 진다. HREADY가 LOW인 동안에 ERROR, RETRY 또는 SPLIT의 response가 나타나고 HREADY가 HIGH로 바뀌는 순간에 transfer를 끝낸다.

ERROR이나 RETRY SPLIT의 response를 나타내기 위해서 2 cycle 이상이 소요될 경우에는 OKAY response와 HREADY를 LOW로 함으로서 2 cycle 이상의 시간을 벌 수 있다.

이러한 two-cycle response가 필요한 이유는 pipelined 구조 때문이다. 즉 Master가 미리 fetch해 놓았던 다음 address를 취소하고 다시 새로운 address를 fetch하기 위해서 master에게 시간을 주기 위한 이유이다.

RETRY나 SPLIT의 경우에는 무조건 다음 전송이 취소가 되어야 하지만 경우에 따라서 ERROR는 전송을 취소할 필요는 없다.

Figure 3-12은 RETRY operation에 대한 그림이다.

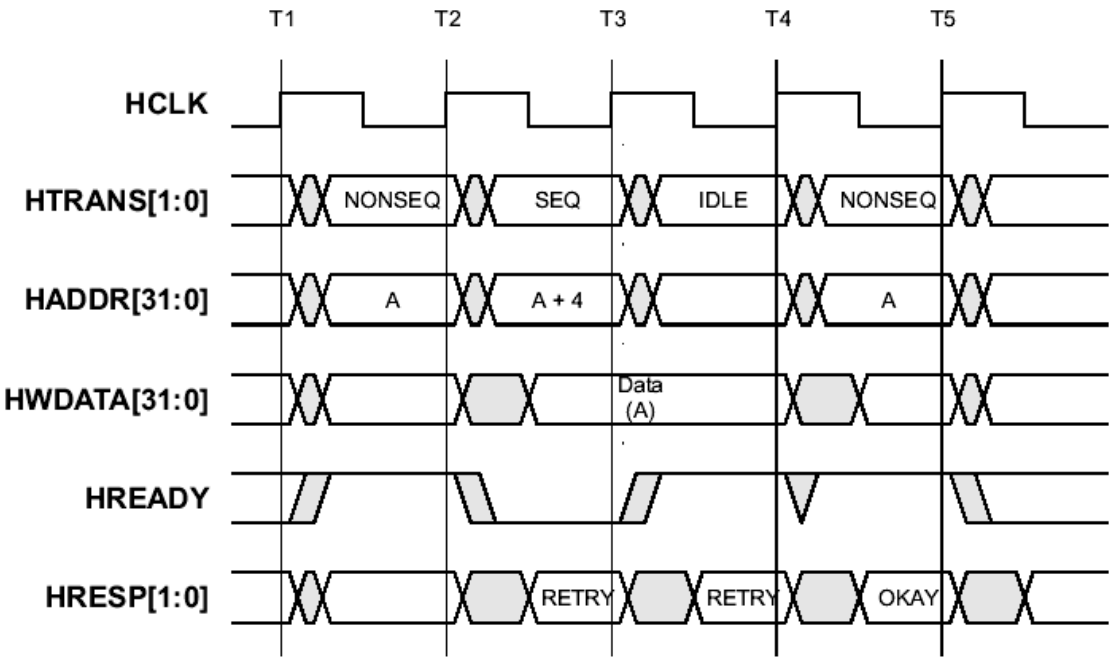

Figure 3-12 Transfer with retry response

위의 그림에서 다음과 같은 일이 일어난다.

Master가 address A로부터 전송을 시작한다.

Response를 받기 전에 master는 미리 A+4의 address를 bus에 올려 놓는다.

Slave쪽에서는 address A의 전송을 제대로 하지 못했기 때문에 RETRY를 보내고 A+4의 address를 취소하고 address A부터 다시 시작할 것을 요구한다.

3.8.4 Error response

Slave로부터 ERROR response를 받았을 때 master가 항상 전송을 취소해야 하는 것은 아니다. Master에 따라서 전송을 계속 해도 되고 취소 해도 된다.

Figure 3-13은 Error response를 내보내기 전에 OKAY와 LOW HREADY를 통해서 한 cycle만큼의 delay를 주는 것을 볼 수 있다. 그리고서 two-cycle의 ERROR response 신호를 보낸다.

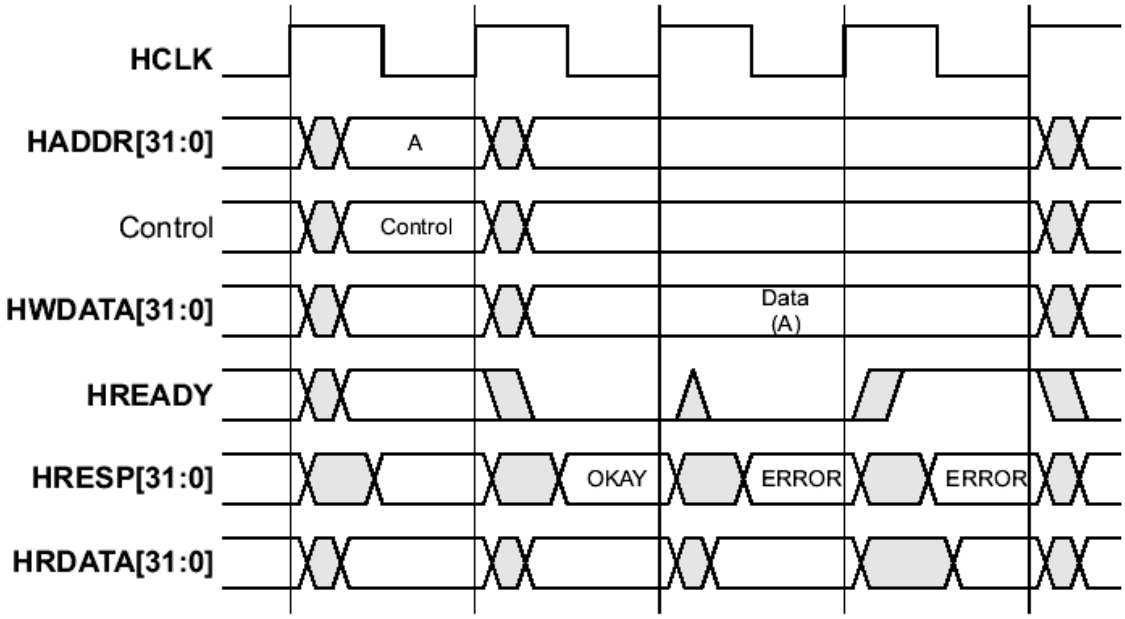

Figure. 3-13 Error response

3.9.5 Split and Retry response

SPLIT이나 RETRY response는 bus에 데이터를 계속하여 받을 수 없을 때 나타나는 일련의 작업이다. 두 response 모두 전송을 끝마칠 수 있는 환경을 주기 때문에 priority가 더 높은 master에게 bus에 대한 권리를 넘길 수 있다.

SPLIT과 RETRY의 차이점은 이러한 response가 발생한 후에 발생하는 일에서 차이가 난다는 것이다.
RETRY의 경우 arbiter는 보통 때 사용되었던 일반적인 priority algorithm을 그대로 사용할 것이다.
SPLIT의 경우에는 split난 것을 제외한 것을 가지고 priority algorithm을 적용하게 된다. 그러므로 더 priority가 낮은 master에게 bus를 사용할 권리를 줄 수 있게 된다.

Master의 입장에서 보면 RETRY나 SPLIT의 경우 모두다 같은 방식으로 response를 다루게 된다. 즉 전송이 완전히 마무리가 될 때까지 계속 Request를 하거나 ERROR로 인하여 전송을 취소하게 될 것이다.

3.9 Data buses

AHB는 endian에 대한 고려가 전혀 없다. 그러므로 master와 slave간에 스스로 endian에 대한 것을 맞추어 주어야 한다.

Endian에 관한 설명은 table 3-6과 3-7에 걸쳐 나와있다.

Table 3-6 Active byte lanes for a 32-bit little-endian data bus

Transfer size	Address offset	DATA [31:24]	DATA [23:16]	DATA [15:8]	DATA [7:0]
Word	0	✓	✓	✓	✓
Halfword	0	-	-	✓	✓
Halfword	2	✓	✓	-	-
Byte	0	-	-	-	✓
Byte	1	-	-	✓	-
Byte	2	-	✓	-	-
Byte	3	✓	-	-	-

Table 3-7 Active byte lanes for a 32-bit big-endian data bus

Transfer size	Address offset	DATA [31:24]	DATA [23:16]	DATA [15:8]	DATA [7:0]
Word	0	✓	✓	✓	✓
Halfword	0	✓	✓	-	-
Halfword	2	-	-	✓	✓
Byte	0	✓	-	-	-
Byte	1	-	✓	-	-
Byte	2	-	-	✓	-
Byte	3	-	-	-	✓

3.10 Arbitration

AMBA system에 있어서 Arbiter의 역할은 어떤 Master가 Bus에 대한 Access를 얻는가를 설정해 주는 것이다. 이를 위해서, 모든 Master는 REQUEST/GRANT interface를 가지고 있어야 하며, Arbiter에게 Request를 했을 때, Arbiter는 고유의 Priority scheme을 가지고, 어떤 Master가 가장 높은 Priority를 갖는 가를 계산해서 그 해당 Master에게 Grant Signal을 보낸다.

또한 각각의 Master는 HLOCKx signal을 갖는데 이것은 Master가 bus의 독점적인 access를 하려고 할 때 발생된다. 즉 현재 Grant된 Master의 LOCK이 enable상태일 때에는 다른 Priority가 높은 Master가 Bus를 요구했을지라도 현재의 Master의 전송이 끝날 때 까지는 Bus를 넘겨주지 않는다.

3.10.1 Signal Description

Arbiter에 사용되는 signal에 대한 간략한 설명이 Table 3-8에 나와있다. 이는 Table 2-2에도 설명된 바 있다.

Table 3-8 AMBA AHB Arbiter Signal Description

Name	Source	Description
HBUSREQx	Master	Bus Request 신호는 bus Master가bus에 대한 access를 요청할 때 사용된다. 각각의 bus Master는 각각 HBUSREQx라는 interface를 가지게 되며, Arbiter는 최고 16개의 Master를 다룰 수 있다. 즉 HBUSREQ[15:0]까자 사용할 수 있다.
HLOCKx	Master	Lock 신호는 bus Request신호와 동시에 인가된다. HLOCKx 신호는 master가burst trans를 하려고 할 때, 중간에 다른 Master에게 bus의 소유를 넘기지 않을 것을 위해서 사용된다. Locked transfer의 첫 번째 전송이 시작되면 Arbiter는 현재의 전송을 끝마치기 전에는 다른 Master에게 bus의 사용권을 넘겨주지 않을 것이다.
HGRANTx	Arbiter	Grant 신호는 arbiter에 의해서 발생된다. 이것은 가장 높은 priority를 가진 master에게만 HIGH를 준다. 그렇지만 Grant를 얻었다고 해서 바로 전송을 시작할 수 있는 것은 아니다. HREADY가HIGH일 때에만 비로소 전송을 시작 할 수 있게 된다.

Table 3-8 AMBA AHB Arbiter Signal Description (continued)

Name	Source	Description
HMASTER[3:0]	Arbiter	현재 어떤 Maser가 선택 되어져 있는 가는 HMASTER[3:0]신호를 이용해서 알 수 있다. 이것은 master들이 연결되는 MUX에 사용 되어진다. Maser number는 또한 SPLIT-capable slave에 의해서도 주어진다. 이것은 Master로 하여금 SPLIT 전송을 마무리 할 수 있도록 하는데 사용된다.
HMASTLOCK	Arbiter	HMASTLOCK 신호는 현재 전송하는 것이 locked transfer인지를 알려준다. 이것은 address와control signal과 동시에 나오게 된다.
HSPLIT[15:0]	Slave (SPLIT-capable)	16bit의Split bus는SPLIT-capable slave에 의해서 주어진다. 이를 통해서 SPLIT 전송을 마무리 할 수 있도록 하는데 사용된다.

3.10.2 Requesting Bus Access

Arbiter는 Request 신호를 매번 cycle의 rising edge에서 sampling하게 된다. 그 때마다 내부 priority algorithm을 가지고서 다음 grant를 결정하게 된다.

보통의 경우에 있어서 Arbiter는burst가 끝났을 때에 grant신호를 바꾸게 된다. 그렇지만 어떤 경우에 있어서는 burst를 갑자기 종료하고 다른 priority가 높은 master에게 access를 넘기기도 한다.

이와 같은 이유에서 원래는 burst를 위해서 한번만 Request를 하면 되지만, Re-Request를 해야 하는 경우가 발생하게 된다. 즉 중간에 burst가 종료되었으면, 그 나머지를 전송하기 위해서 Master는 다시 Request를 해야 한다. 예를 들어 처음에 8개의 전송을 하다가 4개까지 전송하고 중단 되었다면 4개짜리 burst나 1개짜리 burst를 4번 요청하는 등의 방법으로 새로운 burst를 시작해야 한다.

만약 아무런 Request가 없는 경우에도 Grant의 어떠한 bit는 항상 HIGH로 setting을 해주어야 하는데 이를 위해서 Default Master가 필요하다. 이 때는 Request를 하지 않는 경우이므로 Master의 HTRANS 값은 IDLE상태를 나타내야 한다.

3.10.3 Granting Bus Access

HREADY가 HIGH일 때 HMASTER[3:0] 값이 바뀌면서 전송을 시작하게 될 것이다. 이에 대한 그림이 Figure 3-14에 나와있다.

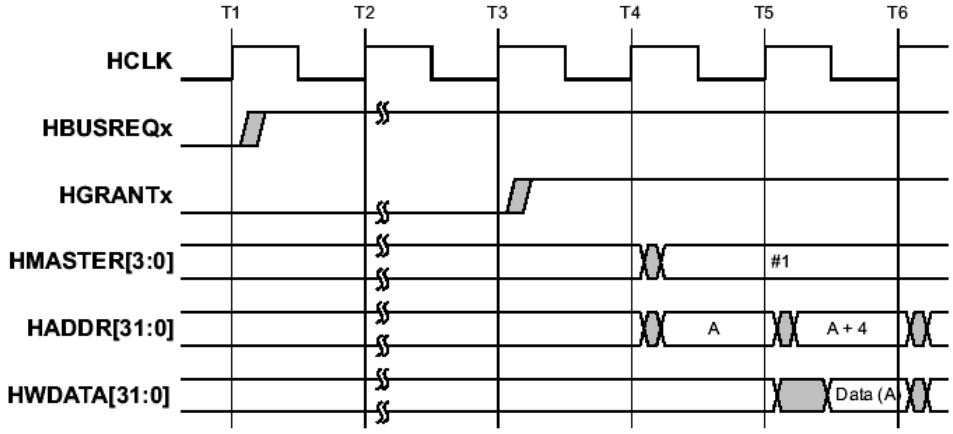

Figure. 3-14 Grant access with no wait states

Figure 3-15는 bus를 handover하는 과정에서 wait states가 사용된 그림이다.

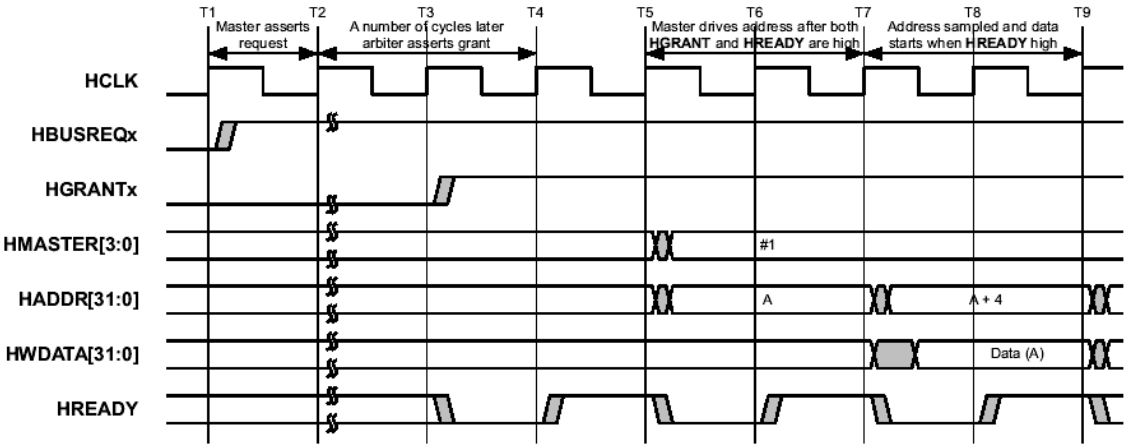

Figure. 3-15 Grant access with wait states

Data Bus의 소유권은 Address Bus의 소유에 따르게 된다. 즉, Burst의 마지막 Address를 전송하고 난 후에 다른 Master로 Grant가 넘어 갔을지라도 그 마지막 Address에 따른 Data는 HREADY가 HIGH일 때 까지, 계속 bus를 소유하게 된다. Figure 3-16는 이러한 Bus의 소유권을 주고 받는 과정을 보여준다.

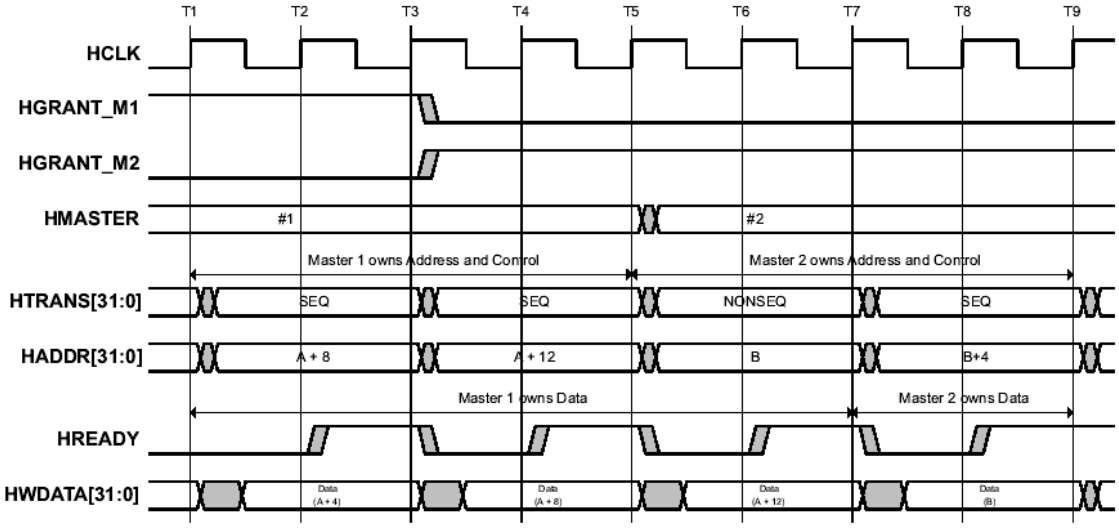

Figure. 3-16 Data bus ownership

Figure 3-17은 burst transfer의 마지막에서 bus를 넘겨주는 과정을 보여준다.

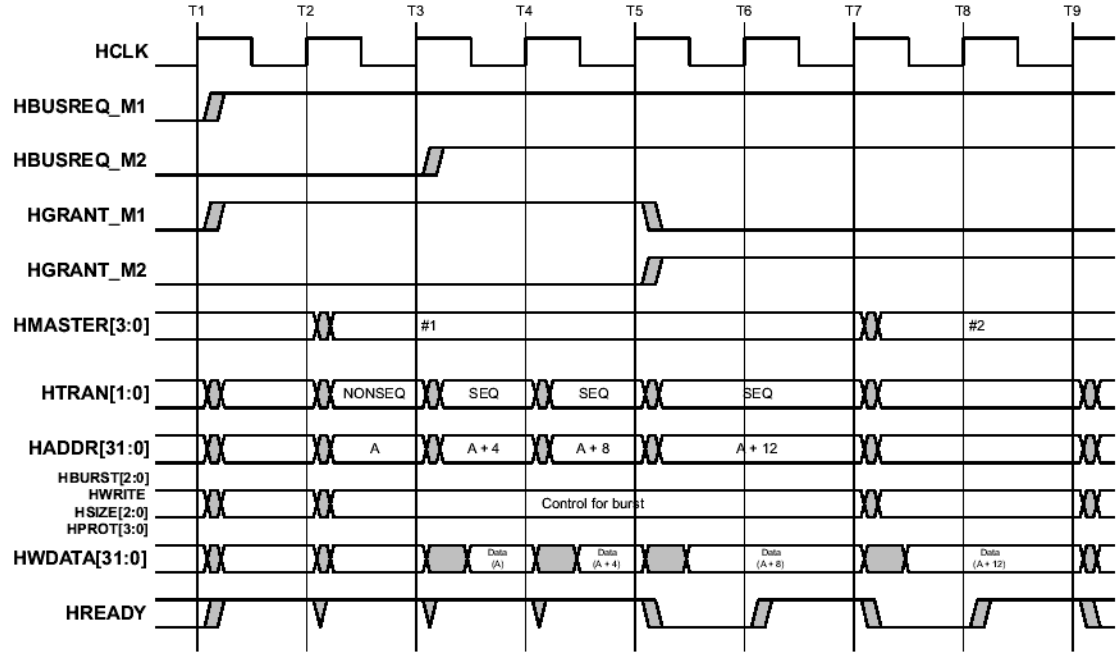

Figure 3-17 Handover after burst

Arbiter는 HGRANTx 신호를 penultimate (one before last) Address가 sample되었을 때 바꾸게 된다.

Figure 3-18는 이에 따른 Address의 Multiplexing과정을 보여주는 그림이다. delay 되는 HMASTER[3:0]의 값이 MUX의 write data control 신호로 사용된다.

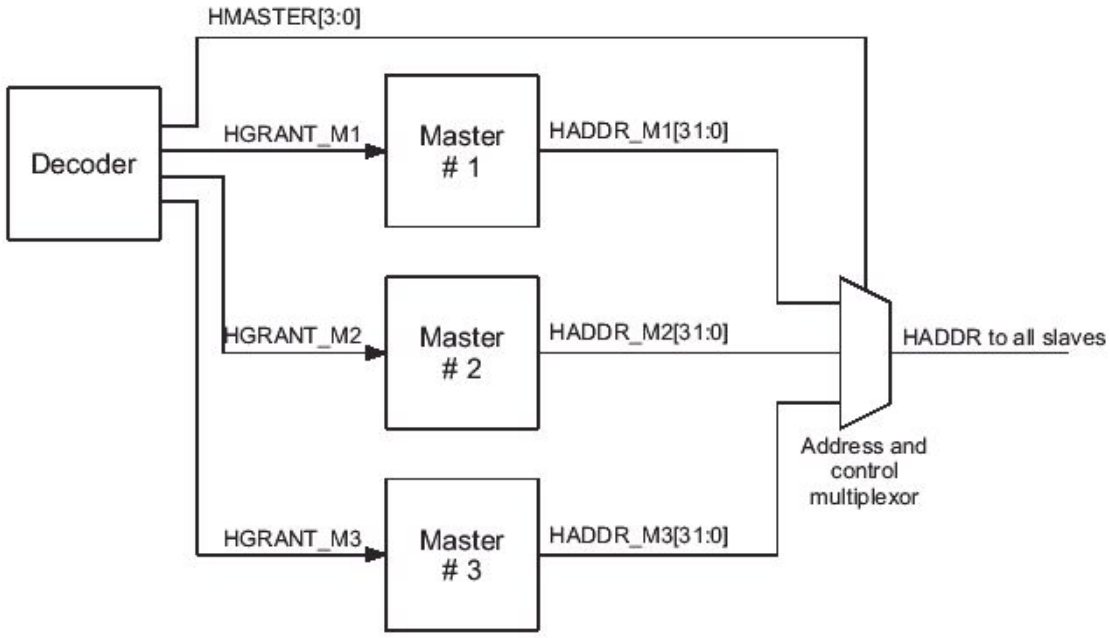

Figure. 3-18 Bus master grant signals

3.11 Split transfers

Transfer가 일어나고 있는 동안에 slave는 현재의 transfer를 나중으로 미룰 것인가에 대한 결정을 하게 된다. 이에 대한 결과로 나타나는 것이 Split이다. Split에 대한 내용은 4장에서 자세히 설명되어 질 것이다.

Figure 3-19에 SPLIT에 대한 sequence를 보여주고 있다.

1. The master starts the transfer in an identical way to any other transfer and issues address and control information
2. If the slave is able to provide data immediately it may do so. If the slave decides that it may take a number of cycles to obtain the data it gives a SPLIT transfer response.

   During every transfer the arbiter broadcasts a number, or tag, showing which master is using the bus. The slave must record this number, to use it to restart the transfer at a later time.
3. The arbiter grants other masters use of the bus and the action of the SPLIT response allows bus master handover to occur. If all other masters have also received a SPLIT response then the default master is granted.
4. When the slave is ready to complete the transfer it asserts the appropriate bit of the **HSPLITx** bus to the arbiter to indicate which master should be regranted access to the bus.
5. The arbiter observes the **HSPLITx** signals on every cycle, and when any bit of **HSPLITx** is asserted the arbiter restores the priority of the appropriate master.
6. Eventually the arbiter will grant the master so it can re-attempt the transfer. This may not occur immediately if a higher priority master is using the bus.
7. When the transfer eventually takes place the slave finishes with an OKAY transfer response.

Figure 3-19 basic stages of a SPLIT transaction

3.11.1 Bus handover with split transfers

전송을 하다가 SPLIT이 발생하면 IDLE의 cycle을 소비한 후에 다른 master에게 bus를 사용할 권리를 넘겨주게 된다.

Figure 3-20은 SPLIT이 일어나는 sequence를 보여주고 있다.

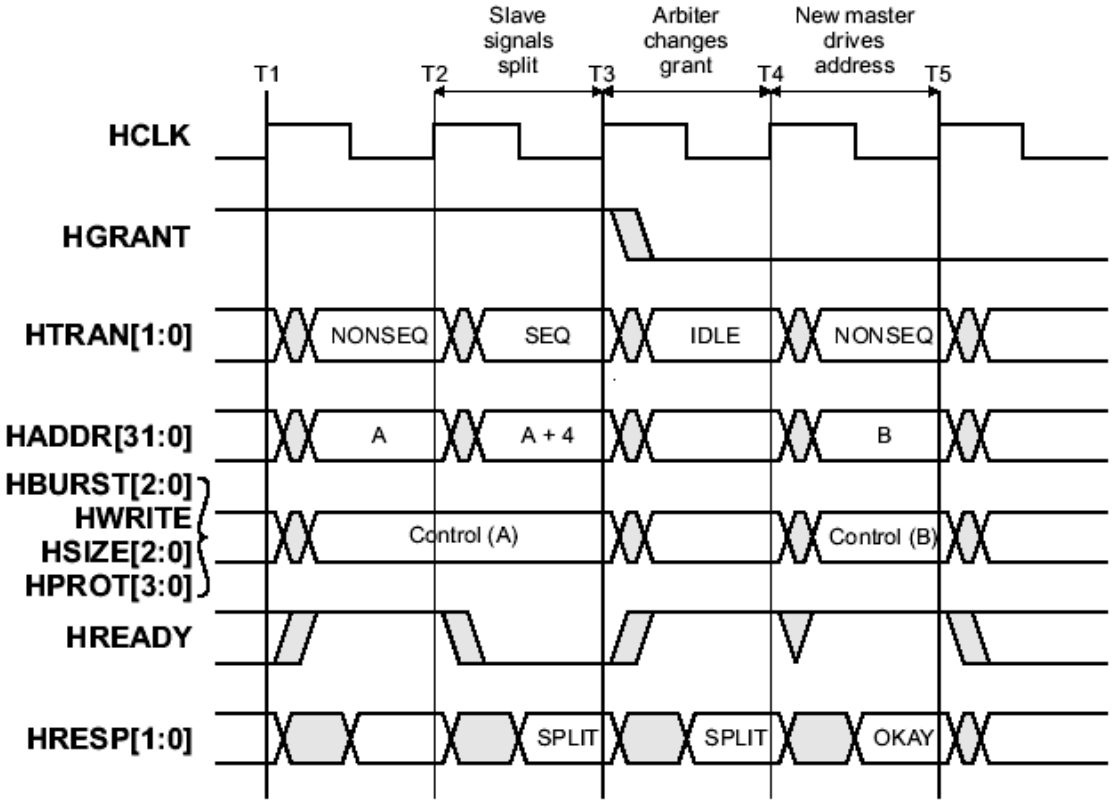

Figure 3-20 Handover after split transfer

3.12 Reset

HRESETn 신호는 AHB에 있어서 유일한 active LOW signal이다. Reset은 언제나 발생할 수 있지만 AHB가 알게 되는 것은 HCLK의 rising edge에서 sampling하게 된다.

Reset이 되는 동안에 HTRANS[1:0]은 IDLE을 나타내어야 하며 address와 control signal들은 valid level에 있어야 한다.

3.13 AHB bus interface

AHB bus의 interface에 대한 그림이 Figure 3-21부터 3-24에 걸쳐 나와있다.

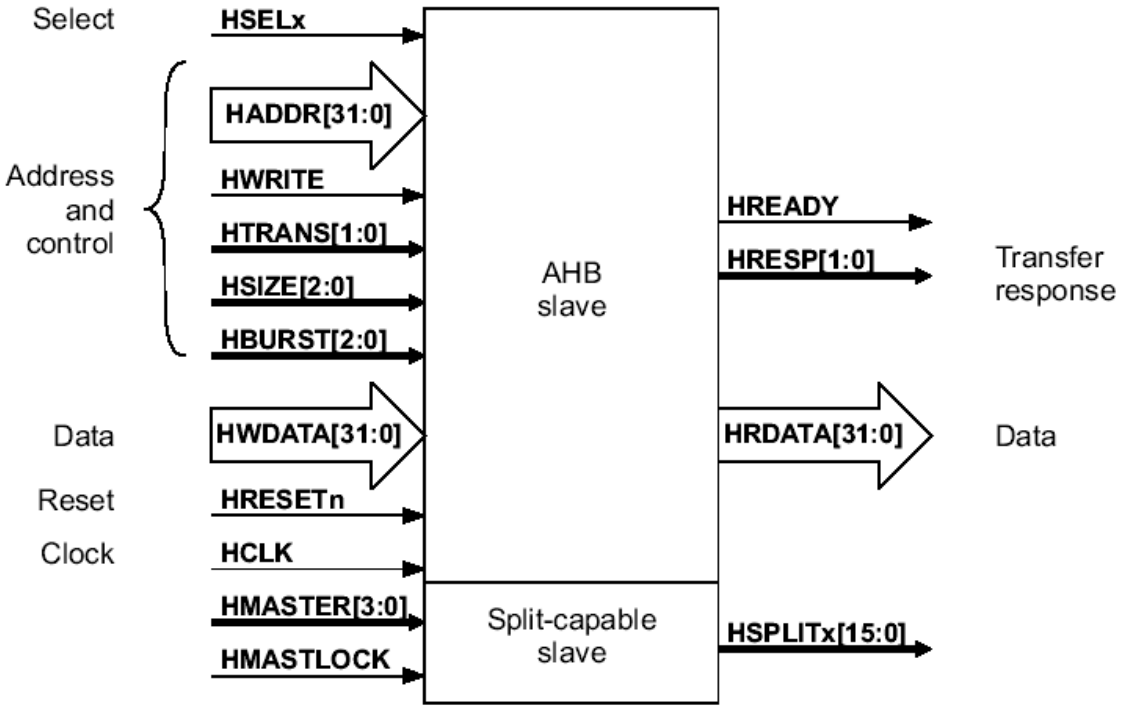

Figure 3-21 AHB bus slave interface

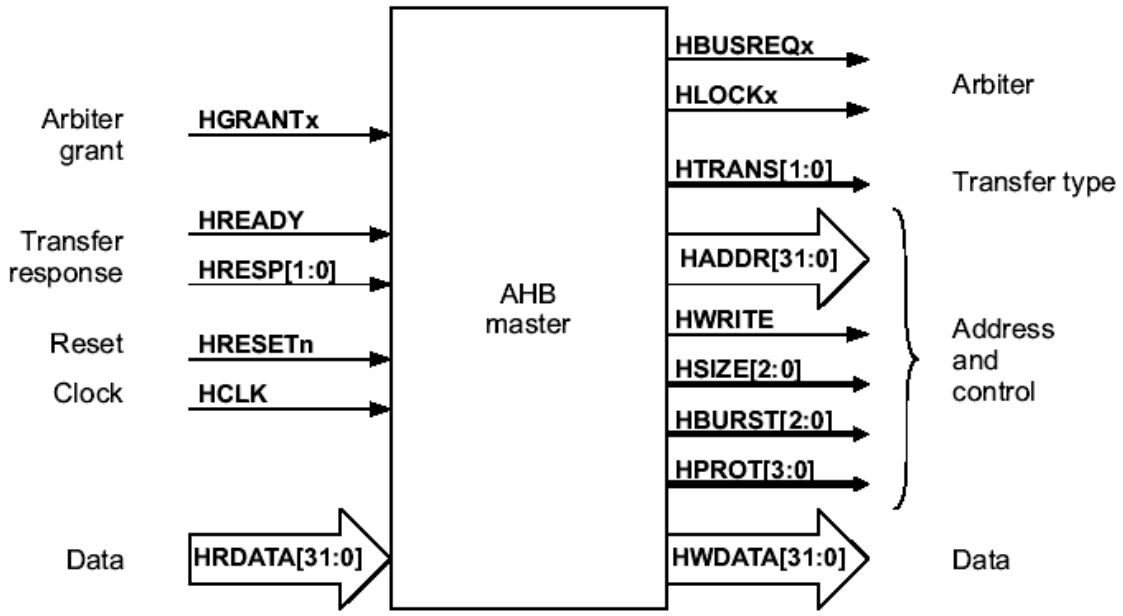

Figure 3-22 AHB bus master interface

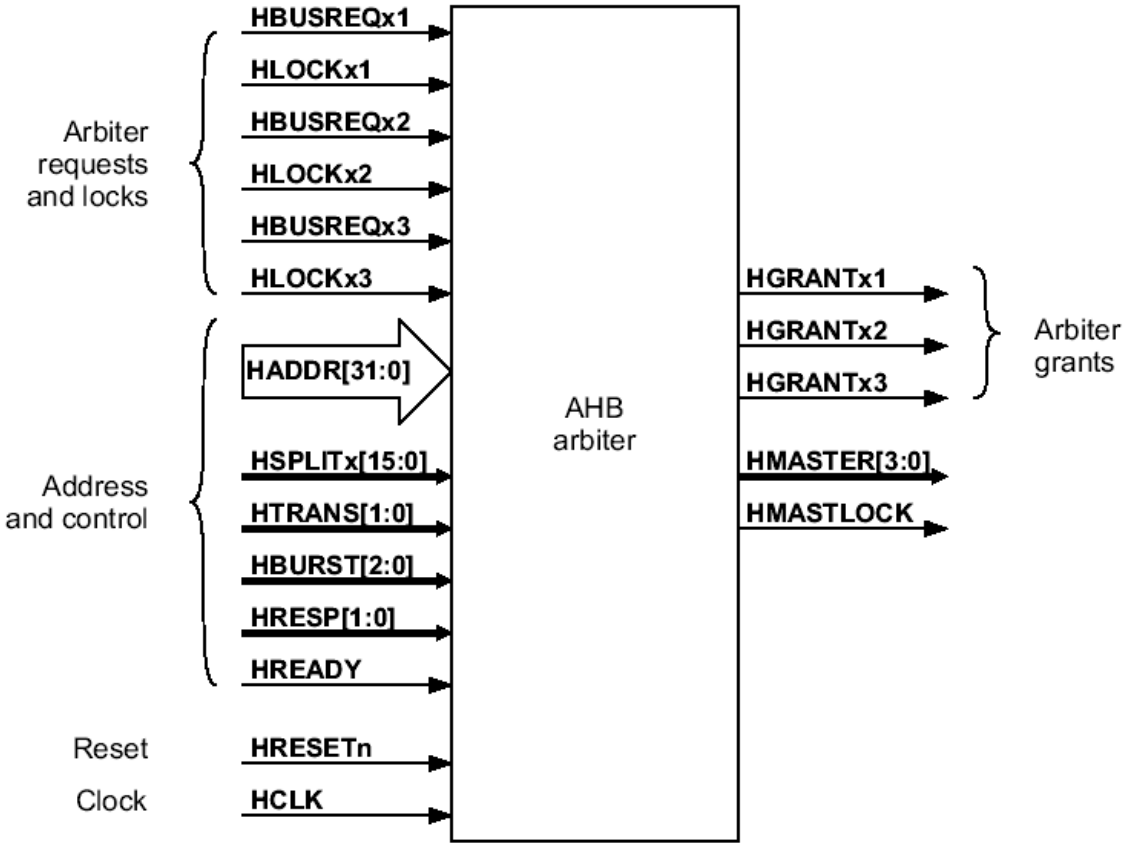

Figure 3-23 AHB bus arbiter interface

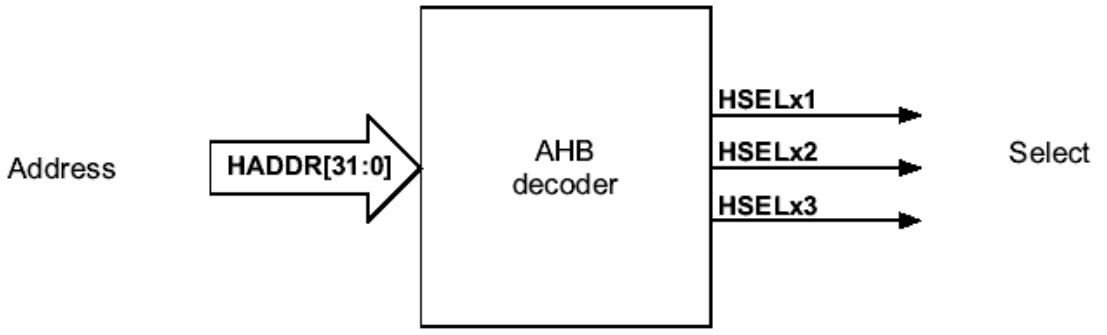

Figure 3-24 AHB bus decoder interface

Chapter 4 Design of the AMBA AHB

4.1 Design Environment

AMBA AHB의 VHDL coding을 위해서 Synopsis의 Cyclone Simulator를 사용하여 Simulation하였다.

4.2 Used Signal list for Arbiter

Arbiter의 Interface는 3.16에서 설명하였다. AMBA AHB Arbiter를 구현하기 위해서는 몇 가지 Constant value, Signal 그리고 Function의 정의가 필요하다.

4.2.1 Prefix

AMBA bus는 1개부터 최대 16개의 Master를 가질 수 있다. 어떻게 System을 구성하는 가에 따라서 Master의 개수는 항상 달라질 수 있을 것이다. 이를 위해서 처음에 몇 가지 상수 값을 정의 하게 된다. Table 4-1에 Define되는 상수 값들을 나타내었다.

Table 4-1 Defined constants

Constant Name	Description
Data_bus_size	Data bus의 Size이다.
Address_bus_size	Address bus의 Size이다.
Number_internal_bus	Internal bus의 숫자이다. 이것은 AMBA bus등을 몇 개나 사용할지에 대한 값을 정의하게 된다. Bus의 개수가 여러 개인 경우에는 Partition등을 통해서 더욱 최적화된 성능을 얻을 수 있다. 그러나 여기서 사용되는 internal bus는 1개를 기준으로 하였다.
Number_master	AMBA AHB에 붙는 Master의 개수이다.
Number_slave	AMBA AHB에 붙는 Slave의 개수이다. Slave의 decoding은 Address의 상위 bits를 통하여 이루어 진다. 이러한 이유에서 Slave로서 붙을 수 있는 개수는 Address에 따라서 달라지게 되는데, 만약 어떤 Master가 32bits의 Address bus를 사용하고, 상위 16bits를 Decoding으로 사용하게 된다면, 최대 65536개의 Slave를 연결할 수 있게 된다.

4.2.2 Functions

AMBA AHB의 구현을 위해서 두 가지 함수를 구현하였다. 그 중에 하나가 REQUEST되는 Master의 개수를 Return해주는 함수(CHECK_REQNUM)이고, REQUEST된 것 중 가장 Priority가 가장 높은 것을 계산하여 그에 해당하는 Master number를 Return해주는 함수(CHECK_PRIORITY)이다.

(1) CHECK_REQNUM

이 function은 HREQUEST[15:0]를 입력으로 받아서 request하는 Master의 개수를 Return 해주는 함수이다. 계산하는 방법은 비교적 간단하다. 우선 reqnum이라는 변수를 만들고 초기 값을 0으로 setting 한다. HREQUEST[15:0] 신호를 받아서 하나하나의 비트를 살펴보고 비트 값이 1일 때에 reqnum을 하나씩 증가시켜주면 된다.

예를 들어 HREQUEST[15:0]이 0x0301라고 할 때 이에 대한 return값

0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1

Figure 4-1 reqnum을 구하기 위한 Example

reqnum = 3 이 될 것이다. 이에 대한 algorithm을 보면 다음과 같다.

```
int reqnum = 0;
for i=0 to Last bit
        if REQUEST(i) == 1
                reqnum ++;
        end if;
end for;
```

Figure 4-2 Request Number 계산을 위한 Algorithm

(2) CHECK_PRIORITY

Bus를 Request하는 Master가 많을 경우에는 가장 큰 priority를 가진 master에게 grant 신호를 내보내 주어야 한다. 이를 계산 하기 위한 함수가 CHECK_PRIORITY이다. 여기서는 입력으로 받은 HBUSREQ[15:0] 중에서 낮은 bit의 master가 priority가 높은 것으로 가정하게 된다. 이를 구현하기 위한 Algorithm은 다음과 같다.

```
int sel = 0;
for i=last bit to 0
        if REQUEST(i) == 1
                sel = i;
        end if;
end for;
```

Figure 4-3 Max Priority Master 계산을 위한 Algorithm

4.2.3 Used Signal Description

Arbiter의 구현을 위해서 Table 4-2에 6개의 Signal을 정의 하였다. 또한 Arbiter의 FSM을 구현하기 위한 state에 대한 설명이 Table 4-3에 있다.

Table 4-2 Defined Signals

Signal Name	Description
ReqNum	Integer형으로서 0에서 master_number-1의 값을 가질 수 있다. HBUSREQ[15:0]가 변할 때 마다 예전에 설명한 바 있는 CHECK_REQNUM 함수를 사용하여 그 결과를 ReqNum이라는 Signal에 저장하게 된다.
PriSel	ReqNum과 같은 integer형을 갖는다. 현재 state가 REQ_GRAN, m0 또는 TRANS_SPLIT 일 때에만 CHECK_PRIOITY 함수를 사용하여 Priority가 가장 높은 Master의 Number를 PriSel이라는 Signal에 저장하게 된다.
Master_Sig	4bit std_logic_vector signal이다. State가 FIRST_TRANS 또는 IDLE 일 때에만 Update된다. Update될 때에는 PriSel 값을 4 bits vector로 변환하여 Master_sig에 저장하게 된다.
MastLock_Sig	Master_Sig와 같이 FIRST_TRANS 또는 IDLE 일 때에만 Update된다. Update될 때에는 HLOCK[15:0]의 vector들 중에서 PriSel에 해당하는 vector의 bit 값을 MastLock_Sig에 저장하게 된다.
Burst_Cnt	Burst mode에서 Count를 세기 위해서 쓰이는 Signal이다. Burst transfer를 할 때에는 1개, 4개, 8개 또는 16개의 data를 한번의 Grant 동안에 보낼 수 있어야 한다. 이를 위해서는 현재 몇 개의 Transfer가 이루어 졌는지에 대한 count가 필요한데, Burst mode에 따라서 count되는 수의 초기값이 달라져야 한다. 이를 위해서 사용하는 Signal이 Burst_Cnt이다. HBURST[2:0]의 값에 따라서 1, 4, 8 또는 16의 값을 가지게 된다.

이 외에도 **Grant_Variable**이라는 Variable도 정의하게 된다. 이 값은 16 bits vector로서 **PriSel**에 해당하는 bit만 1을 갖고 그 이외의 bit 들은 0을 갖는다.
이 값은 **PriSel**이 update 될 때 같이 update되며 **HGRANT[15:0]**를 update하는데 사용된다.

Table 4-3 States of the AMBA Arbiter FSM

State Name	Description
IDLE	Default master가 select되는 상태이다. 아무런 Request가 없을 때는 항상 IDLE 상태가 된다. 또한 RESETn이 LOW일 때에도 IDLE 상태가 된다. ReqNum의 값을 살펴보고 이 값이 0보다 큰 경우 REQ_GRANT state로 넘어가게 된다.
SETUP	HREQUEST에 대해서 Priority 계산하는데 필요한 cycle을 위한 state 이다. 여기서는 단순히 lower bit의 Request가 priority가 높은 것으로 가정하였기 때문에 이를 위한 계산은 한 사이클이면 충분하다. 이 상태에서 만약 ReqNum이 0이 된다면 IDLE상태로 가고, 그 이외에는 Enable state로 갈 것이다. PriSel을 계산하는 state이기도 하다.
ENABLE	HGRANT 신호를 주는 state이다. HREADY를 check하여 HREADY가 HIGH가 되면 XFER state로 넘어가게 된다. master_sig과 mastlock_sig을 계산하는 state이기도 하다.
XFER	모든 Trans의 첫 번째 Trans는 항상 이 State를 통하게 되어있다. 첫 번째 Trans를 한 후에는 HBURST[2:0] signal에 따라서 m14, m6, m2 또는 Setup state로 옮겨가게 된다.
m14-m2	Burst transfer에서 count를 위한 states이다. HREADY='1' 이고, HRESP="00", HTRANS(1)='1' 일 때 하나 밑의 숫자를 갖는 state로 넘어가게 된다. 만약 RESP(1)='1'일 때는 즉 SPLIT이거나 RETRY인 경우에는 다음 state는 setup이 될 것이다. 이것은 mX state에서 모두 동일하다.
m1	Burst의 마지막에서 2번째의 Trans이다. 마지막 Data를 전송하려고 할 때 HGRANT의 값이 update될 수 있다. 그러므로 PriSel을 update하기 위한 계산은 m1 state에서 이루어져야 할 것이다.
m0	Burst transfer에서 마지막 Data를 전송하고 있음을 의미한다. ReqNum이 0일 경우 IDLE 상태로 가고, 그 이외의 경우에는 master_sig과 mastlock_sig을 계산하고 Xfer state로 옮겨가서 새로운 trans를 시작하게 한다.

4.3 Arbiter FSM and VHDL coding

Figure 4-4는 Arbiter의 구현을 위해서 사용된 FSM을 그린 것이다.

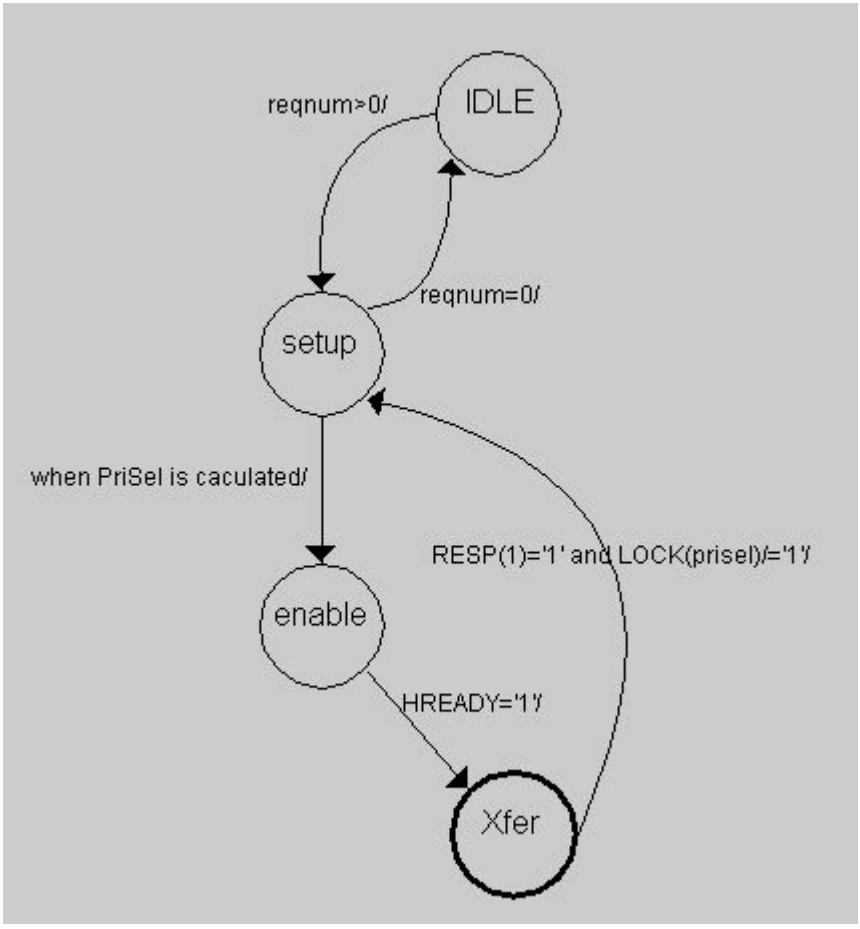

Figure 4-4 Arbiter의 FSM 일부

Figure 4-4는 위에서 설명된 각각의 state를 전부 표현 한 것은 아니다. 이 FSM은 굵은 선으로 그려진 Xfer라는 state는 Figure 4-5와 같은 m14~m0에 이르는 state들과 연결되는 FSM을 갖는다.

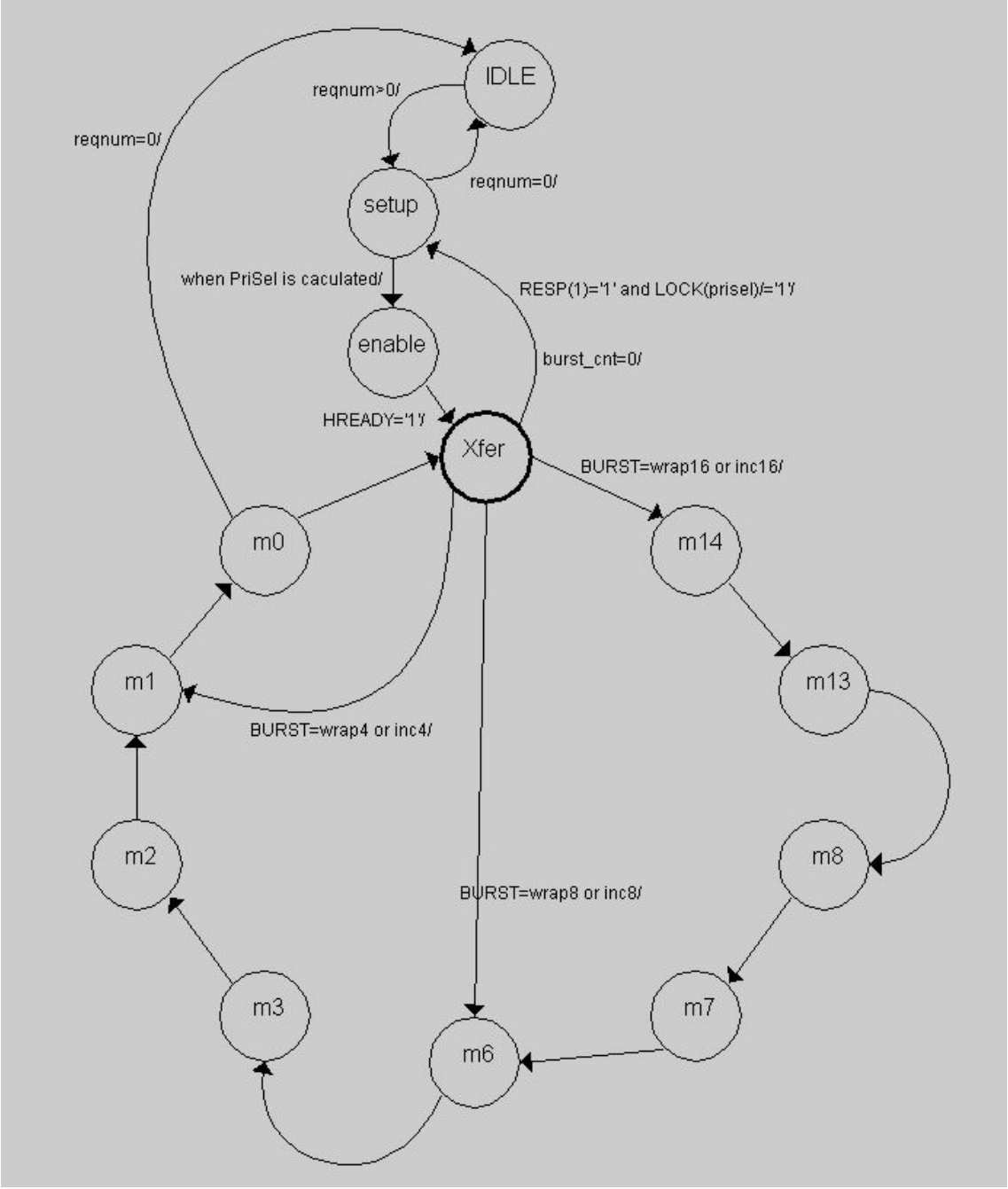

Figure. 4-5 Arbiter의 전체 FSM

Figure 4-5는 전체 state를 모두 보여주는 FSM이다. 여기서 돌아가는 화살표는 그 내부적으로*m12, m11, m10*등의 state가 생략 되었음을 의미한다. 또한 여기서의 Xfer밑의 모든 화살표는

HREADY='1' and HRESP[1:0]="00" and HTRANS(1)='1'
위의 식을 만족 해야만 다음 상태로 옮겨 갈 수 있다.

여기서 주의 할 것은 Count의 하나하나에 해당하는 *m14*에서 *m1*에서는 *IDLE*상태로 갈 수 없다는 것이다. Count가 다 될 때까지 즉, 모든 Burst transfer의 과정이 끝나기 전에는 HGRANT[15:0]의 값을 바꾸지 않음을 의미한다. 그러나 HRESP[1:0]에 따라서 *Setup*의 state로는 언제든지 빠져 나갈 수 있으며 이 때에는 상황에 따라서 HGRANT[15:0]의 값이 바뀔 수 있다.

HRESP[1:0]에 따른 Response는 2 cycle response임을 볼 수 있다.
Trans가 시작되는 때부터는 즉, Figure 4-5의 Xfer부터 *m14, m13, … m0*까지의 모든 state에서는 HRESP[1:0]의 지배를 받게 된다.

4.3.1 OKAY RESP

Figure 4-5의 FIRST_TRANS부터 시작하는 TRANS에서는 위에서 설명한 바와 같이,

HREADY='1' and HRESP[1:0]="00" and HTRANS(1)='1'
의 식을 만족해야만 Transition이 일어나게 된다. 여기서 HRESP[1:0]="00"에 해당하는 것이 바로 OKAY Resp를 받았을 경우이다. OKAY Resp를 받았다 하더라도 HREADY가 '1'이 아니거나 HTRANS(1)이 '1'이 아닌 경우 (idle or busy)에는 항상 그 자리에 멈추어 있게 된다.

4.3.2 RETRY RESP

Figure 4-5의 Xfer, *m14, m13, …, m0*의 모든 state에서 HRESP[1:0]가 RETRY인 경우에는 Setup state에서 다시 TRANS를 시작하게 된다. 물론 Setup으로 옮겨 가면 PriSel 값이 update될 것이다.

4.3.3 SPLIT RESP

Figure 4-5의 Xfer, *m14, m13, …, m0*의 모든 state에서 HRESP[1:0]가 SPLIT인 경우에는 현재의 GRANT[15:0]의 정보가 현재 Slave의 SPLIT[15:0]으로 넘어가게 된다. 그리고 Arbiter에는 REQUEST[15:0]과 SPLIT[15:0]을 XOR시킨 결과가 입력으로 들어가게 된다. 이에 대한 Example이 Figure 4-6에 나와 있다.

Figure. 4-6 Split과정에 의한 REQUEST의 조정

Figure 4-6 에서 **HREQUEST[15:0]**가 0x0301이면 **HGRANT[15:0]**는 0x0001이 될 것이다. 즉 첫번째 master가 TRANS를 주도하게 될 것이다. 그러나 이때 Split이 발생 된다면 현재의 **HGRANT[15:0]**값이 **SPLIT[15:0]**에 저장 된다. 그리고 이 값은 Arbiter의 Priority를 정하는 입력에 **HREQUEST[15:0]** 값과 XOR되어 들어가게 된다.

이와 같을 경우 Master(0)가 아무리 Request를 한다 해도 Slave가 나중에 받아 들이겠다는 Split을 보내기 때문에 Arbiter는 Request를 하지 않은 것으로 여기게 된다. 그러므로 priority가 낮은 다른 Master에게 bus를 사용 할 수 있는 기회를 주는 것이다.

이와 같은 Split이 일어난 후에는 arbiter의 입력으로 들어가는 Request가 달라지게 되므로 다른 RESP와 달리 다른 Master에게 Grant를 주게 되므로 전체적인 latency에 좋은 영향을 주게 될 것이다. 이때에는 새롭게 Xfer부터 다른 Master가 Trans를 시작하게 된다.

그러나 Split이 발생 했다고 해서 master가 항상 bus의 access를 잃는 것은 아니다. 만약 Locked transfer를 하고 있는 중이었다면, 아무리 Split이 나더라도 bus의 대한 소유권을 그대로 가지고 있을 것이다.

4.3.4 ERROR RESP

Figure 4-5의 Xfer, *m14, m13, …, m0*의 모든 state에서 **HRESP[1:0]**가 ERROR인 경우에는 더 이상 해당 Slave에 전송을 할 수 없음을 의미한다. 그러나 현재의 코딩에서는 Slave에서 전송 자체를 무시를 하면서 Master는 전송을 계속 유지하는 방식을 채택하였다.

4.4 Other components

Arbiter이외에도 AMBA를 구성하기 위한 component는 더 있다. Decoder와 Master쪽의 Multiplexor, Slave쪽의 Multiplexor가 그것이다.

4.4.1 Decoder

Decoder는 Address를 받아서 그 상위 bits를 사용하여 Decoding을 하게 된다. 여기서 사용되는 방법은 Figure 4-7에 나와 있다.

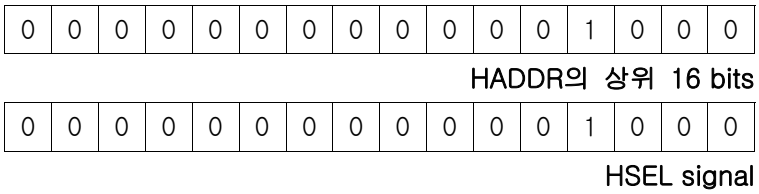

Figure. 4-7 Decoding Example

Figure 4-7에서 사용되는 방법은 상위 bits를 그대로 HSELx에 전해주는 것이다. 즉 LSB부터 slave의 번호를 붙일 때, 4번째 slave가 Select될 것이다.

Figure 4-7에 나와 있는 것과 같이 사용할 경우 몇 가지 제약이 있어야 한다. Master의 Address Upper bits는 항상 16 bit중 하나만 1로 set되어야 하며, 최대 16개의 Slave를 가질 수 있게 된다. 또한 Slave의 Address width는 16비트 이하로 제한된다.

4.4.2 Master Multiplexor

Master Multiplexor는 여러 개의 Master 또는 Slave가 내보내는 Signal 중에서 Grant 또는 Sel 되는 것의 Signal만을 선택하기 위한 장치이다. 주의 할 것은 몇 개의 Master가 사용되느냐, 몇 개의 Slave가 사용되느냐에 따라서 MUX의 bus크기가 달라져야 한다는 것이다.

이를 위해서 4.2.1에서 설명하였던 것과 같은 Constant를 Define하게 된다.

Table 4-1 Defined constants for Master MUX

Constant Name	Source	Description
M_TransBusSet	Master	Trans가 2 bits 이므로 2 bits 크기의 bus set을 구성하게 된다. 만약 3개의 Master가 쓰인다면 2 bits 의 3개의 Array를 Define하는 셈이다.
M_AddrBusSet	Master	Address_bus_size 크기로 number_master 개수 만큼의 Array를 Define 한다.
M_WriteSigSet	Master	1 bit 크기로 number_master 개수 만큼의 Array를 Define 한다.
M_SizeBusSet	Master	3 bits 크기로 number_master 개수 만큼의 Array를 Define 한다.
M_BurstBusSet	Master	3 bits 크기로 number_master 개수 만큼의 Array를 Define 한다.
M_ProtBusSet	Master	4 bits 크기로 number_master 개수 만큼의 Array를 Define 한다.
M_WdataBusSet	Master	Data_bus_size 크기로 number_master 개수 만큼의 Array를 Define 한다.
M_ReadySigSet	Slave	1 bits 크기로 number_master 개수 만큼의 Array를 Define 한다.
M_RespBusSet	Slave	2 bits 크기로 number_master 개수 만큼의 Array를 Define 한다.
M_RdataBusSet	Slave	Data_bus_size 크기로 number_master 개수 만큼의 Array를 Define 한다.

이와 같이 Define된 Set들은 Master Multiplexor의 IN 또는 OUT의 type에 사용된다. 또한 MUX를 control 하기 위한 signal이 필요하게 되는데 이 것은 Arbiter가 내보내는 신호 중에 HMASTER[3:0]을 이용하면 된다.

Master의 신호를 Slave에 전해주기 위한 것은 HMASTER[3:0]에 해당하는 array를 받아들여서 MUX의 출력으로 보내주면 된다. 예를 들어, trans_o라는 OUT에 trans_i IN의 Array중 하나를 연결해 주어야 할 때, trans_i의 array중 HMASTER[3:0]에 해당하는 bit vector를 trans_o에 보내주면 된다.

다음의 경우를 보자. 현재 HMASTER[3:0]이 “0010”일 경우 이 것은 2에 해당하는 Master가 Grant 되었음을 의미한다. 그렇다면 trans_o에 trans_i(2)를 넣어 주는 것이 MUX의 역할이다.

반대로 Slave로부터 오는 신호를 Master에게 전해주는 과정은 Select된 Slave의 Signal들을 그냥 각각의 Master에게 모두 연결해 주기만 하면 된다. HGRANTx에 의해서 grant가 안된 Master들은 넘어오는 signal을 모두 무시하고, grant된 Master만이 AMBA Bus로부터 정보

를 받아들일 것이다.

여기서 주의할 것은, Data bus는 이전의 Master에게 ownership이 있다는 것이다. 예를 들어 현재 Master0이 Grant 되었다고 하자. Master0가 마지막 전송을 끝내어서 이제 Master1에게 Grant가 넘어간다고 하더라도 Master0의 마지막 Data에 대해서는 Master1이 첫번째 Address를 전송하는 동안에 전송이 이루어져야 할 것이다.

그러므로 wdata_i 는 한 cycle delay된 HMASTER[3:0]의 값이 array의 입력으로 들어가야 할 것이며 update되는 시기는 입력으로 들어오는 ready가 '1'이 되는 시기 즉 마지막 data가 성공적으로 전송이 되었을 때가 되어야 할 것이다.

4.4.3 Slave Multiplexor

Slave Multiplexor는 여러 개의 Slave 또는 Master가 내보내는 Signal 중에서 Sel 또는 Grant 되는 것의 Signal만을 선택하기 위한 장치이다. 주의 할 것은 몇 개의 Master가 사용되느냐, 몇 개의 Slave가 사용되느냐에 따라서 MUX의 bus크기가 달라져야 한다는 것이다.

이를 위해서 4.2.1에서 설명하였던 것과 같은 Constant를 Define하게 된다.

Table 4-2 Defined constants for Slave MUX

Constant Name	Source	Description
S_ReadySigSet	Slave	1 bits 크기로 number_slave 개수 만큼의 Array를 Define 한다.
S_RespBusSet	Slave	2 bits 크기로 number_slave 개수 만큼의 Array를 Define 한다.
S_RdataBusSet	Slave	Data_bus_size 크기로 number_slave 개수 만큼의 Array를 Define 한다.
S_SplitBusSet	Slave	16 bits 크기로 number_slave 개수 만큼의 Array를 Define 한다.
S_AddrBusSet	Master	Address_bus_size 크기로 number_slave 개수 만큼의 Array를 Define 한다.
S_WriteSigSet	Master	1 bit 크기로 number_slave 개수 만큼의 Array를 Define 한다.
S_TrsnsBusSet	Master	2 bits 크기로 number_slave 개수 만큼의 Array를 Define 한다.
S_SizeBusSet	Master	3 bits 크기로 number_slave 개수 만큼의 Array를 Define 한다.
S_BurstBusSet	Master	3 bits 크기로 number_slave 개수 만큼의 Array를 Define 한다.
S_WdataBusSet	Master	Data_bus_size 크기로 number_slave 개수 만큼의 Array를 Define 한다.
S_MasterBusSet	Master	4 bits 크기로 number_slave 개수 만큼의 Array를 Define 한다.
S_MastlockSigSet	Master	1 bit 크기로 number_slave 개수 만큼의 Array를 Define 한다.

이와 같이 Define된 Set들은 Slave Multiplexor의 IN 또는 OUT의 type에 사용된다. 또한 MUX를 control 하기 위한 signal이 필요하게 되는데 이 것은 Decoder가 내보내는 신호 중에 HSELx를 이용하면 된다.

Slave쪽의 MUX는 이전에 설명된 바 있는 Master MUX와 비슷하게 구현이 된다. 차이가 있다면 delay되는 bus가 Master MUX는 오직 wdata였던 반면에 Slave MUX는 rdata, ready, resp, split의 4개가 모두 delay되어야 한다는 것이다.

4.5 Connection of the all components

최상위 schematic인 AMBA에는 그래서 이전에 설명 되었던 4개의 component를 연결하면 된다.

Figure 4-8은 이에 대한 전체 schematic을 보여주는 그림이다.

Figure 4-8에서 왼쪽의 선들은 모두 Master에게 연결되는 pin이고 오른쪽의 선들은 Slave에 연결되는 pin이다. 그림의 밑에 HRESETn 과 HCLK도 살펴 볼 수 있다.

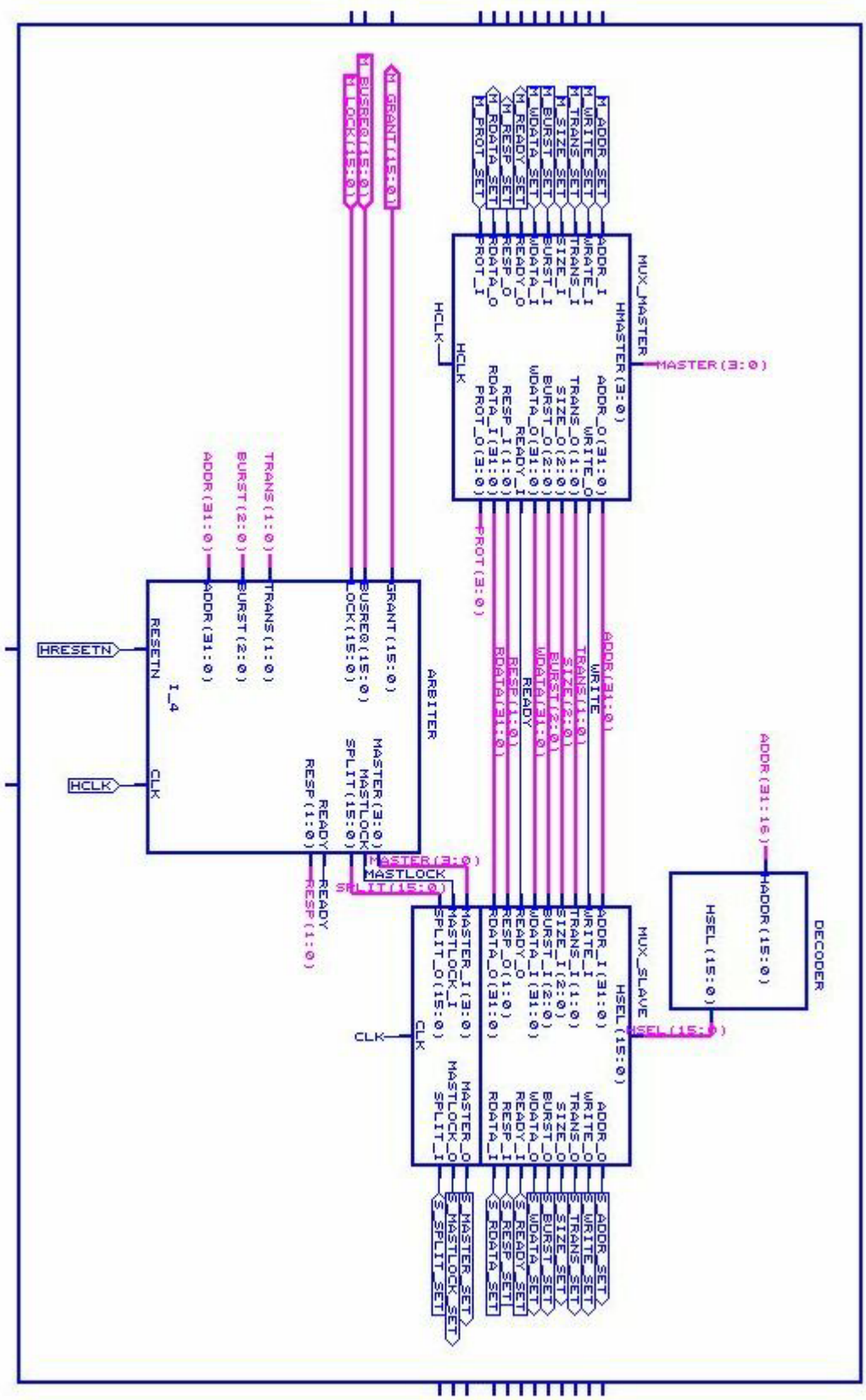

Figure. 4-8 AMBA schematic