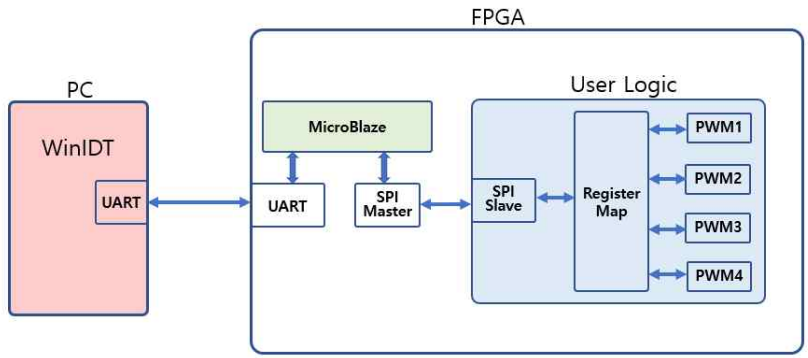
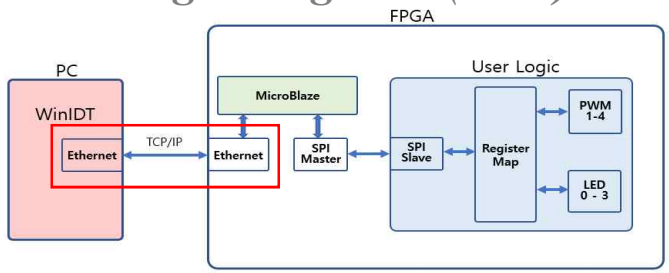
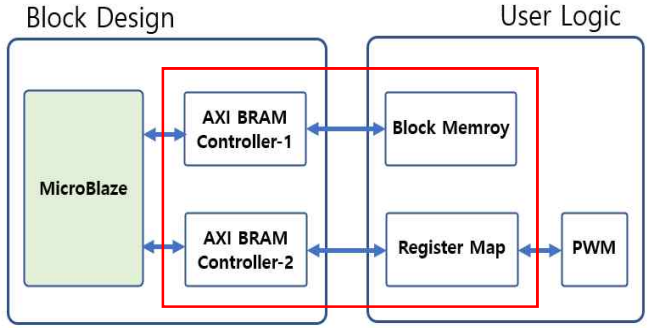
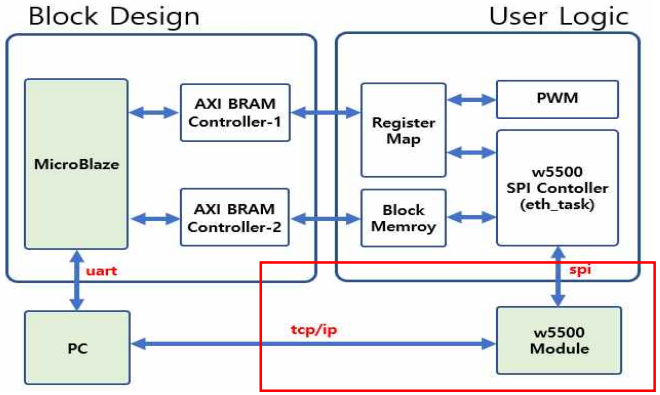
Table of Contents

➢ *SoC 를 위한 Peripheral 설계*









➢ *SoC Peripheral RTC Design Project*

AMBA

(Advanced Microcontroller Bus Architecture)

[AXI (Advanced eXtensible Interface)]

[Reference]

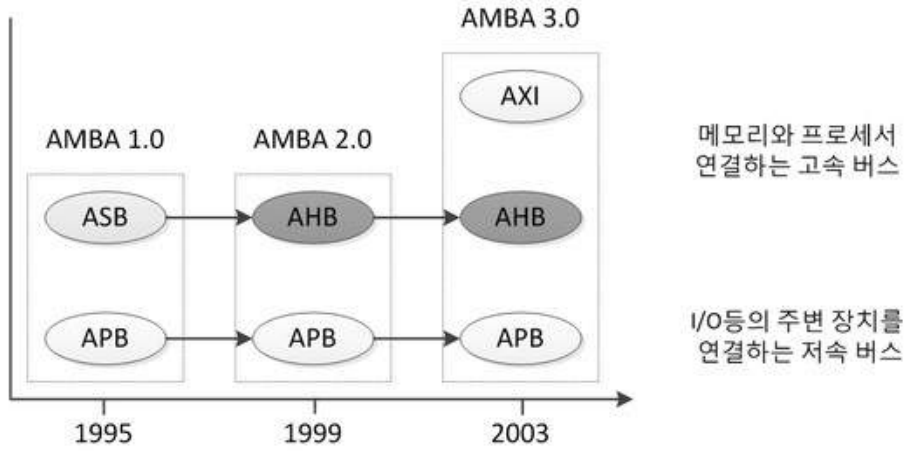
- *ARM, AMBA Specification Rev 2.0,1999.*
- *ARM, AMBA AXI Protocol Specification Rev r0p0, 2003.*
- *http://www.gstitt.ece.ufl.edu/courses/fall15/eel4720_5721/labs/refs/AXI4_specification.pdf*
- *https://en.wikipedia.org/wiki/Advanced_Microcontroller_Bus_Architecture*

AMBA

- ***AMBA (Advanced Microcontroller Bus Architecture)***
  - 영국의 ARM 사가 개발한 시스템용 온 칩[System on Chip(SoC), 칩 내부] 버스 표준 ➔ 컴퓨터를 구입할 때 주로 보게 되는 PCI 와 같은 것들은 장비와 장비 사이에 Data 를 전달하는 Off-chip 버스
  - 온칩 버스를 이용하여 SoC 내부의 각 IP 코어를 연결하고 데이터를 서로 주고 받음
  - 이상적으로, SoC 를 구성하는 모든 업체의 IP 코어를 쉽게 연결할 수 있도록 하는 것이 바람직하지만, 버스의 사양이 통일되지 않으면 상당히 많은 시간과 비용이 소요. 따라서 ARM 사는 온칩 버스의 사양을 정하고 이를 무상 공개하여 IP 코어 인터페이스의 표준화를 촉진
  - FPGA 를 생산하는 Xilinx 에서 ***AXI*** 버스로 ***IP*** 의 버스를 통일하여 실험이나 시제품을 만들 때 하나의 버스 시스템을 이해 하면 되므로 많이 편리

- ***Objective of the AMBA specification*** **(*AMBA Bus*의 목적)**
  - 하나 이상의 CPU, GPU 또는 신호 프로세서를 갖춘 임베디드 마이크로컨트롤러 제품의 ***Right-first-time***(최적의 초기 개발)을 가능하게 함
  - 다양한 ***IC*** 프로세스에서 ***IP*** 코어 , 주변 장치 및 시스템 매크로셀을 재사용할 수 있도록 기술 독립적(***Technology-Independent***)이어야 함
  - 프로세서 독립성을 향상시키기 위한 모듈식 시스템 설계(***Modular System Design***)와 재사용 가능한 주변 장치 및 시스템 IP 라이브러리 개발을 장려합니다.
  - 고성능 및 저전력 온칩 통신을 지원하면서 실리콘 인프라를 최소화

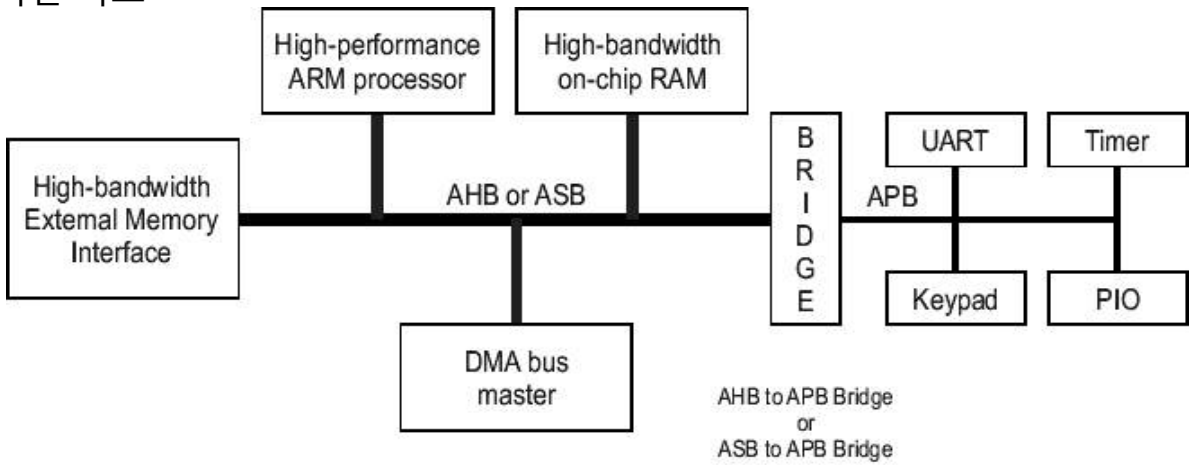
AMBA



- ***AMBA Bus* 종류**
  - *Advanced High-performance Bus* (AHB)
  - *Advanced System Bus* (ASB)
  - *Advanced Peripheral Bus* (APB)

- ***AMBA 1.0***
  - AMBA 의 첫 번째 버전인 AMBA 1.0 은 1995 년에 공개
  - 1.0 에서는 ***ASB (Advanced System Bus)***와 ***APB (Advanced Peripheral Bus)***라는 두 버스 사양이 정의
  - ***ASB*** 는 프로세서와 메모리 및 고성능 장치를 연결하는 버스,
  - ***APB*** 는 I/O 등의 주변 장치를 연결하는 버스

- ***AMBA Bus***
  - ***APB***는 ***higher bandwidth main system***과 ***lower bandwidth*** 디바이스를 연결하는 2차버스의 역할을 담당



AMBA AHB

* High performance
* Pipelined operation
* Multiple bus masters
* Burst transfers
* Split transactions

AMBA ASB

* High performance
* Pipelined operation
* Multiple bus masters

AMBA APB

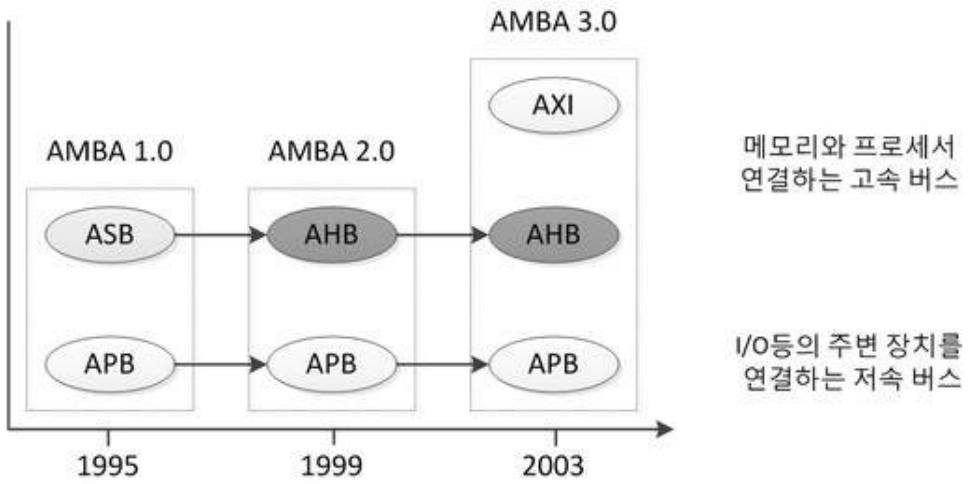
* Low power
* Latched address and control
* Simple interface
* Suitable for many peripherals

AMBA

- 프로세서와 메모리 등을 연결하는 버스 규격
  - 1995 년에 발표된 ASB (Advanced System Bus) 규격에서 ***1999*** 년에는 **고성능** ***AHB (Advanced High-performance Bus)*** 규격으로 대체되었으며, ***2003*** 년에는 고성능 시스템 ***LSI(Large Scale Integration)*** 용 ***AXI (Advanced eXtensible Interface)*** 가 추가

- ***AMBA 2.0***
  - AMBA 의 두 번째 버전인 AMBA 2.0 은 1999 년에 공개
  - 이 사양은 새롭게 ***AHB (Advanced High-performance Bus)***를 정의
  - **AHB 는 SoC 용**으로 **ASB 보다 진보된 기능**을 가지고 있음 ➔ 구체적으로 설명하면 합성을 쉽게 하기 위하여 상승에지 만의 타이밍을 이용하도록 하고, Tri-State 없이 설계하도록 변경

- ***AMBA 3.0***
  - AMBA 의 세 번째 버전인 AMBA 3.0 은 2003년 6월에 공개
  - 이 사양은 AMBA 2.0 AHB 와 APB외에 ***AXI (Advanced eXtensible Interface)***라는 새로운 버스가 추가되었고. AXI 는 고성능 SoC 의 요구에 부응하기 위해 **AHB 사양 보다 높은 성능**이 필요한 시스템의 메인 버스를 상정하고 개발 된 인터페이스



AMBA

➢ 용어 정의

- ***Bus cycle***
  - ✓ AHB, APB → rising edge to rising edge
  - ✓ ASB → falling edge to falling edge
- ***Bus transfer***
  - ✓ AHB, ASB → 하나이상의 버스 사이클을 요구하는 Data read/Write. 지정된 slave로부터 completion응답을 이 이루어 질때까지. ***ASB***에서의 **전송사이즈**는 ***byte(8bit), halfword(16bit), word(32bit),*** AHB는 ASB에서 지원되는 사이즈를 지원하며 64bit, 128bit를 추가적으로 지원.
  - ✓ APB → 항상 two-bus cycle을 요구.
- ***Burst Operation***
  - ✓ Bus master에 의해 초기화되는 하나 이상의 데이터(데이터 묶음) 전송동작. APB에서는 지원되지 않음,

- ***AHB master*** : 한번에 하나의 master만이 bus를 사용할 수 있음.
- ***AHB slave*** : 슬레이브는 주어진 어드레스 범위내에서 read/write동작을 수행함. Slave는 데이터 전송에 대한 success, failure, waiting상태에 대한 정보를 master에게 돌려 주어야 함.
- ***AHB arbiter*** : arbiter는 한번에 하나의 master만이 bus를 사용하도록 권한을 부여함. 하나의 AHB는 하나의 arbiter를 가짐.
- ***AHB decoder*** : decoder는 전송하고자 하는 slave의 어드레스를 디코딩. 하나의 decoder가 모든 AHB상의 slave들을 디코딩 함.

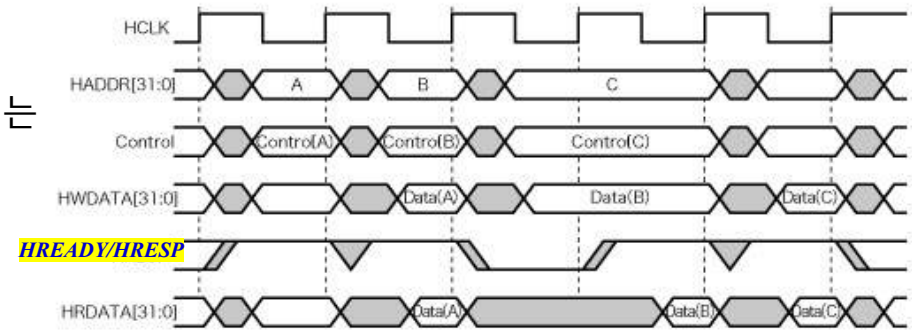
AMBA → AXI(1)

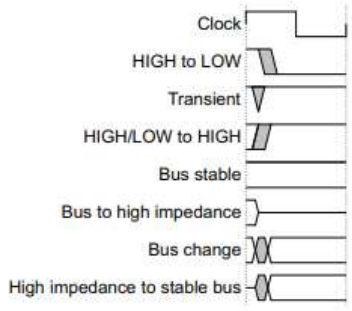
- ***AXI*** 사양 → AHB 의 신호와 타이밍은 일반적인 CPU 의 버스 인터페이스와 비슷
  → 반면 AXI 사양은 성능을 높이기 위해 기존과는 상당히 다른 구조

  - ***AXI*** 에서 도입된 '**채널**' 이라는 개념
    - **AXI 와 AHB 의 가장 큰 차이는 '채널'**이라는 개념의 도입
    - 채널은 지금까지의 일반적인 버스 구조에는 없었던 개념[버스라는 개념 자체도 정확한 정의가 있는 것은 아님. CPU 외부 신호 군 전체를 하나의 버스라고 칭하는 것도 있고, 여러 선으로 구성되어 다양한 정보를 전송하는 것을 버스라고 부르는 경우도 있음]
    - '***AXI*** 가 기존과 어떻게 다른가?' 를 알아보기 위해 *AHB* 와 같은 기존의 "버스"에 대해서 먼저 파악 필요

  - ***AHB*** 와 같은 기존의 "버스" 파악(1/3)
    - ***AHB*** **버스는 각각의 정보를 나르는 역할만 함**(예를 들어 ADDR 은 주소만, WDATA 는 쓰기 자료, RDATA 는 읽기 자료만 이동)
    - 이러한 버스 사이의 동작 타이밍은 문서에 정의되어 있으며, 이것들을 따라서 설계.
    - 하나의 전송을 위해 각 버스(HADDR 과 HWDATA)가 함께 작동해야 하며, 독립적으로 작동할 수 없음
    - 이러한 버스는 단지 정보를 버스 위에 실어 놓기만 하고, 정보를 외부 디바이스에 건네주는 순서는, 다른 버스나 제어 신호와의 타이밍에 의해 결정 → 순서 그림과 같음

[그림] ***AHB*** 의 타이밍 다이어그램

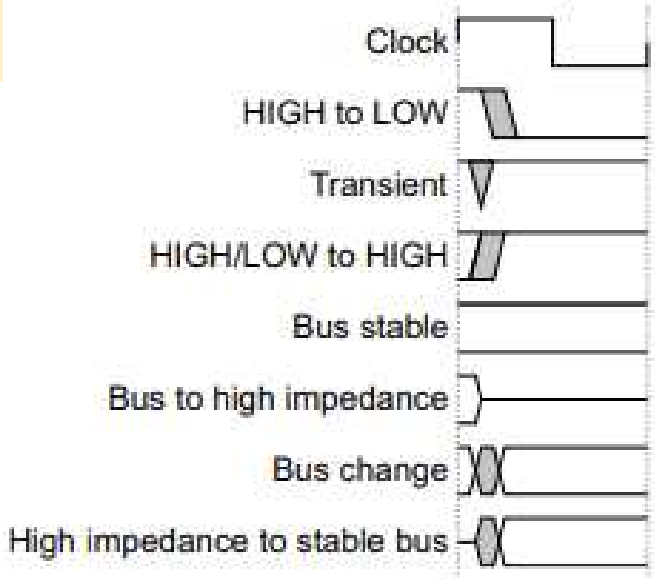




AMBA → AXI(1)

➢ ***Timing Diagrams*** & ***Signals*** & ***Numbers*** [참고]

- *The figure named Key to timing diagram conventions explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.*
- *Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.*
- *Timing diagrams sometimes show single-bit signals as HIGH and LOW at the same time and they look similar to the bus change shown in Key to timing diagram conventions. If a timing diagram shows a single-bit signal in this way then its value does not affect the accompanying description.*



Key to timing diagram conventions

Signals

The signal conventions are:

Signal level — The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means:
- HIGH for active-HIGH signals
- LOW for active-LOW signals.

Lower-case n — At the start or end of a signal name denotes an active-LOW signal.

Numbers

Numbers are normally written in decimal. Binary numbers are preceded by `0b`, and hexadecimal numbers by `0x`. Both are written in a `monospace` font.

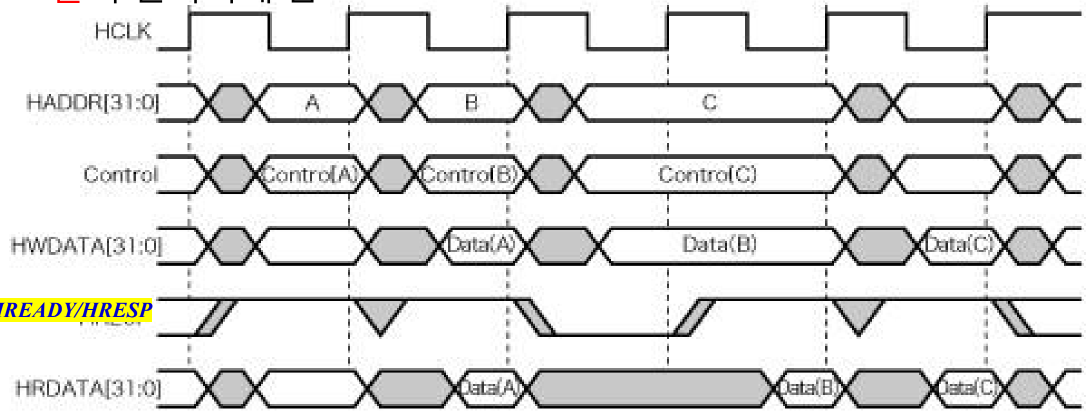
AMBA → AXI(1)

❖ *AHB* 와 같은 기존의 "버스" 파악 (2/3)

- **주소(*HADDR*)와 제어(*Control*)신호를 송신**해, **그에 대한 데이터를 송신(*write*)**하고, **응답(*read*)하는 순서**로 진행.

➔ 하지만, 속도가 낮은 디바이스가 데이터 준비에 시간이 걸리면 그 동안 주소버스는 다음을 진행하지 않고 대기하게 되는 문제점 : 이로 인해 버스 효율이 떨어지게 됨

[그림] *AHB* 의 타이밍 다이어그램



1) **1 사이클의 동안**, 마스터(CPU)가 주소버스(***HADDR***)에 주소를 싣고, 제어 신호(***Control***)에는

2) 이에 해당하는 **트랜잭션(*transaction*)**의 정보를 실음

✓ **명령의 시작부터 명령 처리 종료까지의 한 묶음의 처리**를 **트랜잭션(transaction)**라고 부름. 예를 들면 **읽기 처리의 경우, 읽기 명령을 시작하고 나서 데이터를 읽어낼 때 까지**를 가리킴

3) [**데이터 읽기의 경우**] **다음 사이클**에 슬레이브(***Slave***, 디바이스)가 읽기 데이터버스(***HRDATA***)에 데이터를 싣고, 결과를 응답 신호(***HRESP***)에 실음

4) [**데이터 쓰기의 경우**] 마스터가 쓰고자 하는 데이터를 데이터버스(***HWDATA***)에 싣고, 슬레이브가 결과를 응답 신호(***HRESP***)에 실음

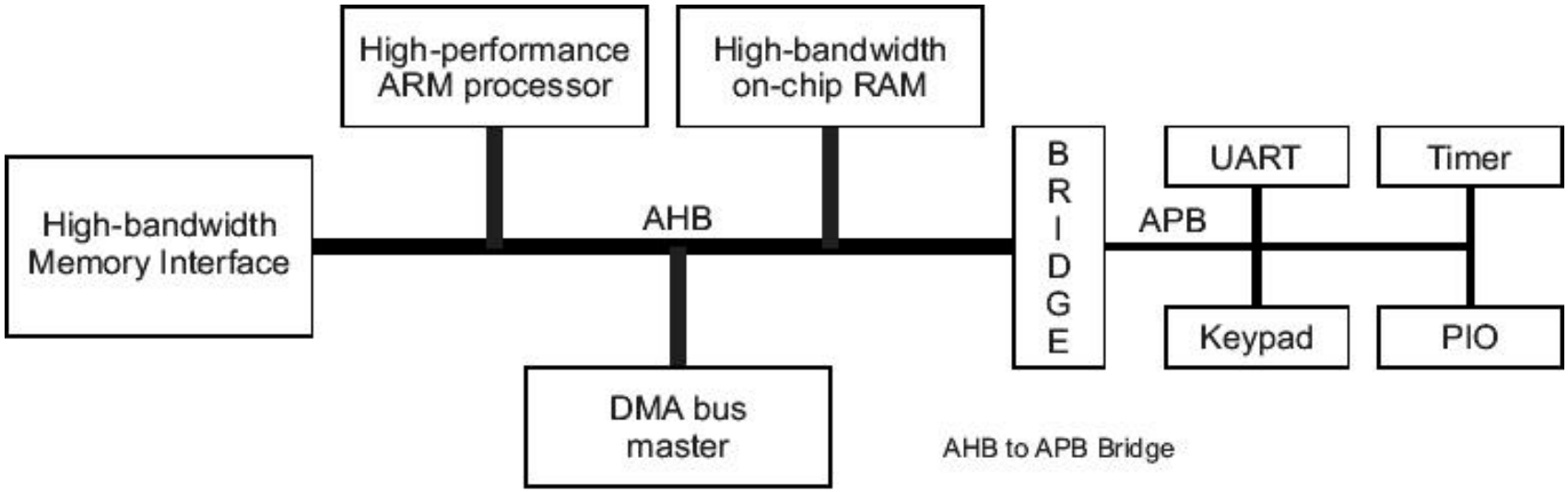
- 이 방식(***AHB***)은 **각 버스에 신호가 실리는 차례나 타이밍, 버스 상호의 관계에 의해서 데이터의 전송 처리가 진행**

- 따라서, 버스 간의 동작 타이밍은 엄밀하게 정의되어 있고 버스 신호만 독립적으로 수행되지는 않음

AMBA → AXI(1)

❖ *AHB* 와 같은 기존의 "버스“ 파악 (3/3)

- ***A typical AMBA AHB-based system***



AMBA Advanced High-performance Bus (AHB)
* High performance
* Pipelined operation
* Burst transfers
* Multiple bus masters
* Split transactions

AMBA Advanced Peripheral Bus (APB)
* Low power
* Latched address and control
* Simple interface
* Suitable for many peripherals

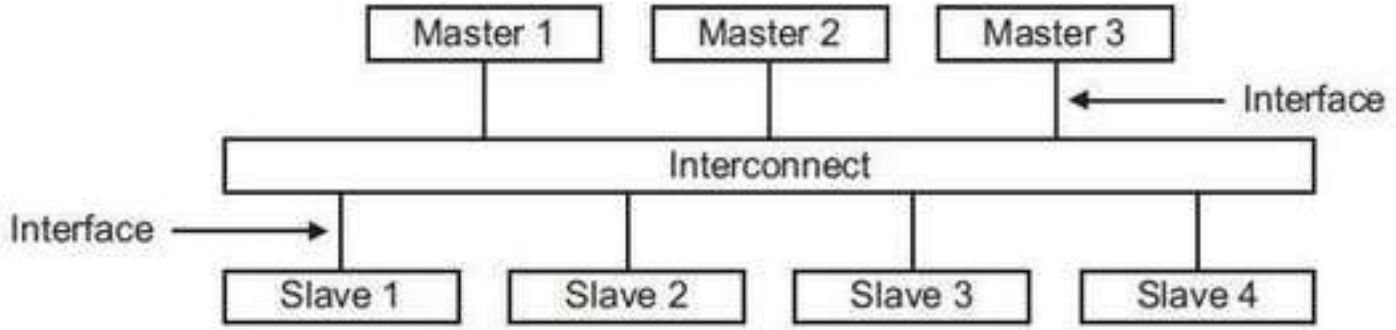
AMBA → AXI(1)

❖ *AXI(Advanced eXtensible Interface)*

- 반면 AXI 에서는 다음과 같은 신호로 구성

➔ **(1) 읽기 주소·채널 (2) 읽기 데이터·채널 (3) 쓰기 주소·채널 (4) 쓰기 데이터·채널 (5) 쓰기 응답 채널**

- 종래 「**xxx 버스**」라고 칭하고 있던 것이 「**xxx 채널**」이 되어 있음
- ***AXI*** 는 **각 채널마다 독립적으로 정보를 전송** ➔ 예를 들어, **AHB 에서는 "주소를 전송하면 다음 사이클에서 해당 데이터를 반드시 전달 해야 한다" 등의 제약**이 있지만, ***AXI*** 에서는 채널 간 단계에 대해서 제약이 없음 ➔ 따라서 **다른 채널의 시기에 관계없이 정보를 계속 전송할 수 있음**
- 이처럼 AXI 에서는 버스 사이가 독립적으로 동작하며 정보 스트림을 전송 할 수 있어, "채널"이라고 부름
- ***AXI* 사양**은 **인터페이스 사양**을 제시한 것이지, 버스 사양에 대한 것은 아님 ➔ 따라서 **상호연결(*Interconnect*) 부분은 사양에서 제외**(그림 참고)



[그림] AXI 사양 구조

- 따라서, **상호연결(*Interconnect*)부분**은 문서에서 정의하고 있지 않음(AHB 의 사양서에는 멀티플렉서로 연결 구성이 실려 있지만, 나중에 고안된 멀티 레이어 AHB 는 상호연결 구현 방식을 사용할 수 있도록 되어져 있음). 상호연결 형태로는 다음과 같은 것들을 생각할 수 있음

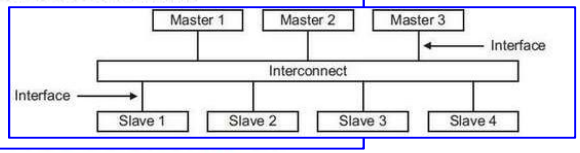
➔ 1. 포인트 투 포인트 2. 싱글 레이어 3. 멀티 레이어 4. 주소는 싱글 레이어 5. 데이터는 멀티 레이어

AMBA → AXI(2)

A1.3 AXI Architecture

The AXI protocol is burst-based and defines the following independent transaction channels:

- read address
- read data
- write address
- write data
- write response.



❖ *AXI* 의 주요 특징(1/5)

1) **채널 구조**와 **레지스터 슬라이스**

- **마스터와 슬레이브** 간은 **읽기 주소 채널**, **읽기 데이터 채널**, **쓰기 주소 채널**, **쓰기 데이터 채널**, **쓰기 응답 채널**이라는 **다섯 가지 채널**로 연결 (그림)
- AHB 사양과 같이 규정된 시간에 마스터에서 주소 또는 데이터 신호가 일방적으로 출력되는 것이 아니라. **읽기 동작**에서 **마스터와 슬레이브**가 **읽기 주소 채널**과 **읽기 데이터 채널**을 통해 **데이터를 전송**
- **쓰기**는 **쓰기 주소 채널**, **쓰기 데이터 채널** 및 쓰기 **응답 채널**을 통해 **데이터를 전송**

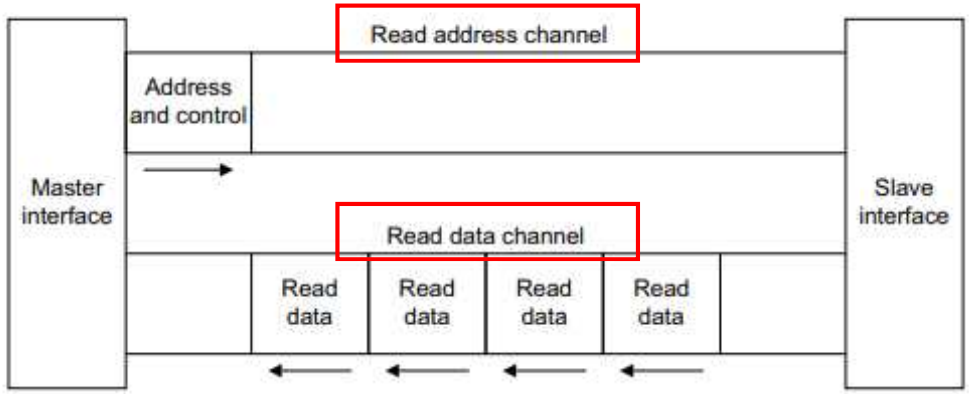
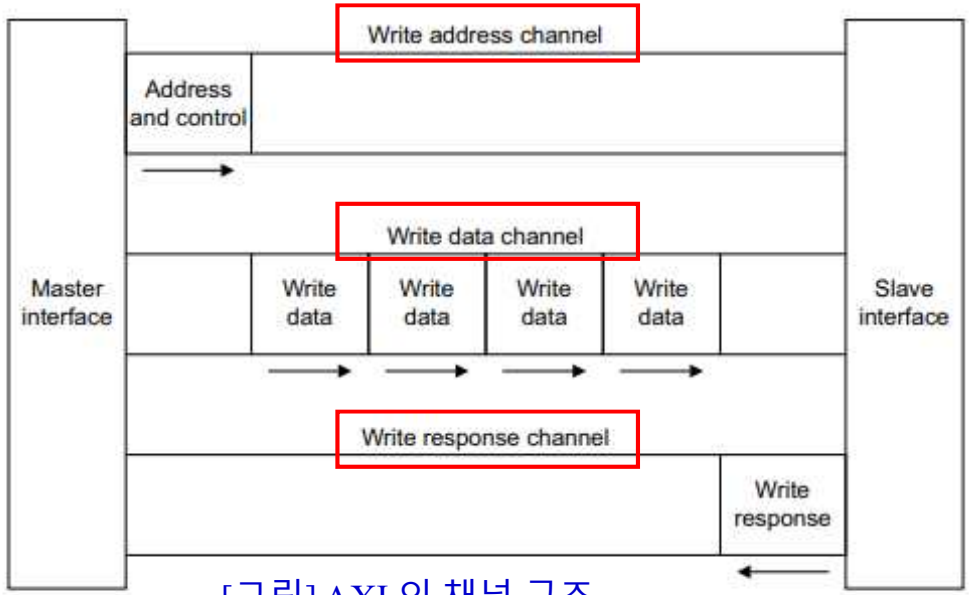


Figure A1-1 Channel architecture of reads

Figure A1-2 shows how a write transaction uses the write address, write data, and write response channels.



[그림] AXI 의 채널 구조

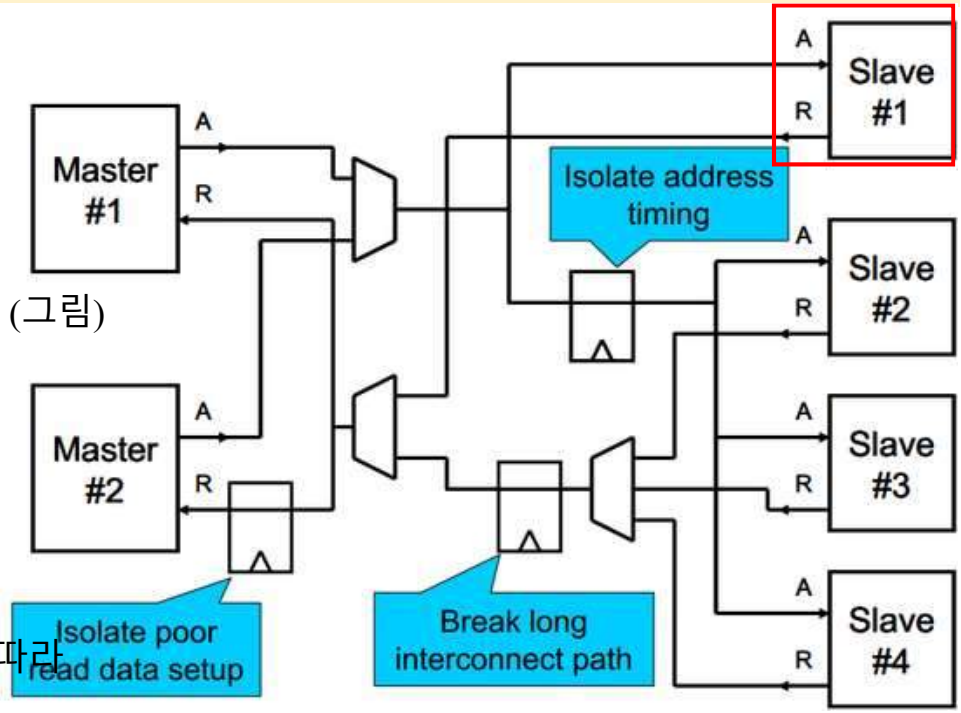
Figure A1-2 Channel architecture of writes

AMBA → AXI(2)

❖ AXI 의 주요 특징(2/5)

1) **채널 구조**와 **레지스터 슬라이스**

- 각 채널의 신호는 기본적으로 모두 같은 방향이므로 채널의 경로 (path)에 Register 를 쉽게 삽입할 수 있음 (그림)
- 채널에 레지스터를 추가할 수 있다는 것은 타이밍이 중요한 설계에 큰 이점을 제공
- 버스가 고속 클럭에 동기화하여 작동할 경우 버스의 타이밍 설계는 매우 어려워지지만 신호선의 경로를 따라 레지스터를 삽입하는 것으로, 1 클럭 신호가 움직이는 범위가 줄어 배선 길이가 짧아, 따라서 버스의 동작 주파수를 높일 수 있음



[그림] 채널 레지스터 삽입 (레지스터 슬라이스)

- 레지스터 슬라이스를 사용하여 전체 버스의 동작 주파수를 저하시키지 않고 버스 타이밍 설계를 할 수 있음

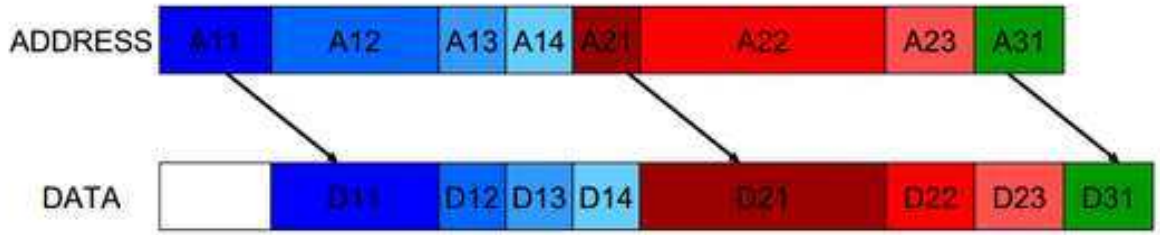
그러나 레지스터를 삽입 한 위치에 데이터 전송을 하기 위해서는 1 클럭의 시간이 더 필요 ➔ 고속 데이터 전송이 필요한 장치는 레지스터를 사용하지 않는 위치에 연결 필요성 (예; 이 그림의 Slave #1 에 연결)

AMBA → AXI(2)

❖ AXI 의 주요 특징(3/5)

2) 버스트(*Burst*) 기반 트랜잭션

- AXI 사양의 트랜잭션은 버스트(묶음) 전송을 전제(물론 단일 데이터 전송도 할 수도 있음)
- 고성능 시스템에서는 대부분의 경우 CPU 에 캐시 메모리가 탑재되어 있으며, 메인 메모리와의 데이터 전송은 캐시 메모리를 채우거나 메모리를 업데이트할 때 버스트 전송이 대부분을 차지
- 또한 지속적인 스트리밍 데이터는 일반적으로 일정량의 버스트 전송을 반복 → 따라서 버스트 전송을 사용할 경우 트랜잭션 시스템의 성능 향상을 기대
- AHB 에서는 버스트 전송시에 버스의 마스터가 데이터 주소를 하나씩 생성하는 방식을 사용 (그림 참조)
- AXI 에서는 버스트로 전송되는 데이터의 주소는 첫 번째 주소만 보내고 후속 데이터의 주소 생성은 슬레이브가 책임을 짐 → 따라서 주소 채널의 전송량은 다른 채널에 비해 데이터의 양을 현격하게 줄일 수 있음

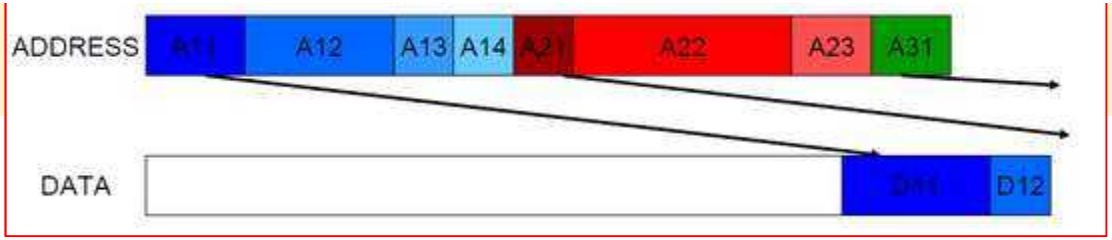


[그림] AXI 에서 버스트 전송 데이터의 주소는 선두만 지정

AMBA → AXI(2)

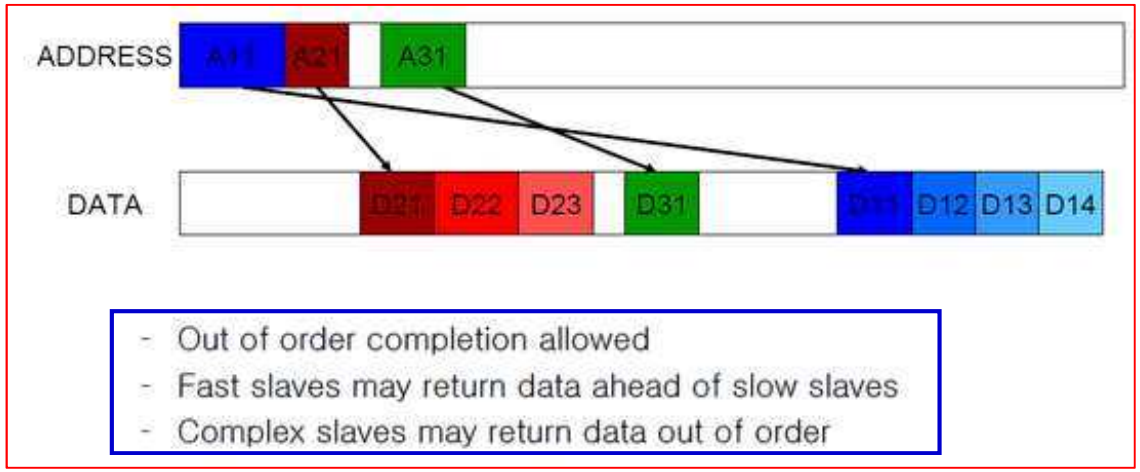
❖ AXI 의 주요 특징(4/5)

3) 여러 주소 발행



[그림] With AHB ➔ If one slave is very slow, all data is held up

- 버스의 효율적인 이용이라는 관점에서 보면 저속 슬레이브 장치의 존재는 매우 신경이 쓰이는 문제
- 기존의 버스는 하나의 트랜잭션이 종료한 후 다음을 수행하므로, 일단 저속 장치에 액세스 해 버리면, 그 트랜잭션이 끝날 때까지 전체 버스가 점령되어 버리는 문제점



- AXI 사양은 먼저 발행 한 주소의 트랜잭션이 종료되기 전에 다음 주소를 여러 게시할 수 있음(여러 주소 발행).
- 발행된 주소에 해당하는 트랜잭션은 아웃 오브 오더 (***out-of-order***; 순서와 상관없이)에 완료할 수 있음(즉, 발행한 주소의 순서와 관계없이 트랜잭션을 종료할 수 있음)

AMBA → AXI(2)

❖ AXI 의 주요 특징(5/5) → skip

4) 제어 정보 추가

- 고성능 시스템에서 요구하는 2 차 캐시를 연결하거나 프로세서에 추가된 보안 기능을 지원하기 위해 AXI 사양은 다음 신호를 정의

☞ 시스템 캐시를 연결하는 신호 ☞ 보안 액세스를 나타내는 신호

5) 단독 엑세스 추가

- 멀티 프로세서 구성의 경우, 프로세서 간의 동기화 동작을 실현하기 위해 세마포어 등이 이용
- 세마포어를 실현하기 위해서는 세마포어 처리 동안 다른 프로세서 처리가 끼어들지 않도록 배타적 연산을 실시
- AHB 는 이를 특정 버스 마스터가 버스를 점유하는 버스 잠금으로 구현하였고, 그 동안 다른 버스 마스터는 버스를 사용할 수 없도록 함
- AXI 는 해당 주소에 단독으로 액세스하는 방법을 취하고 버스가 점령되지 않도록 되어 있음

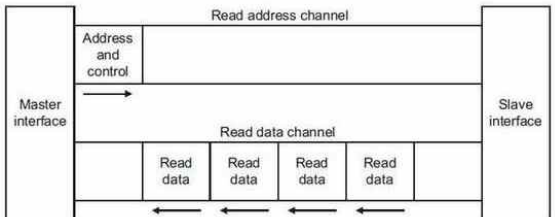
6) 바이트 레인 스트로브 신호

- 데이터 버스 폭보다 폭이 좁은 데이터를 전송하는 경우 어떤 바이트 레인에 유효한 데이터가 있는지를 나타내는 신호 (바이트 레인 스트로브 신호)가 추가

7) 저소비 전력화

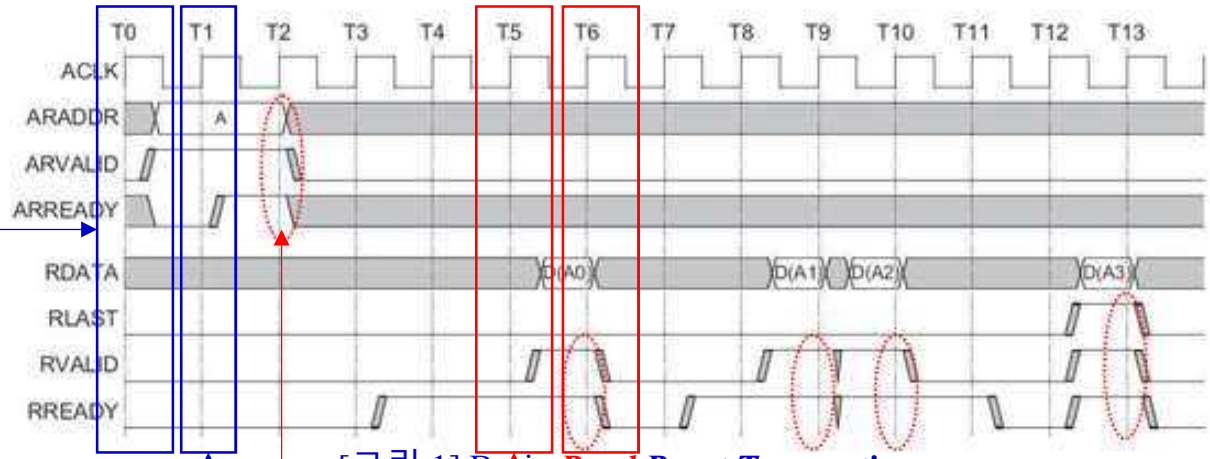
- 소비 전력 최소화 위해 AXI 사양은 주변 회로 클럭을 정지하기 위한 인터페이스 신호 및 프로토콜 사양이 정의

AMBA → AXI(3)



➢ 기본 데이터 전송 프로토콜(1/4)

❖ AXI 사양의 기본 **데이터 전송** 프로토콜을 사용하여 읽기와 쓰기에 대한 4-Word(=8byte=64bit) 버스트 전송 트랜잭션을 예로 설명



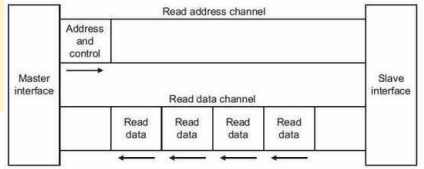
[그림 1] Basic ***Read Burst Transaction***

1) 읽기 트랜잭션(1) [그림 1. 참고]

1. **Master 의 주소채널**은 **전송 시작주소**와 **전송에 대한 제어 정보** (버스트 길이 데이터 폭 전송 유형 등)를 싣고, ARVALID 신호를 Assert(그림1의 시간 T0) → 슬레이브는 주소 채널의 정보를 받을 수 있으므로, ARREADY 을 Assert 하고 응답(그림 1의 시간 T1) → 주소 채널의 정보는 ARVAILD 과 ARREADY 가 Assert 되는 시점에 전송 (그림 1의 시간 T2)

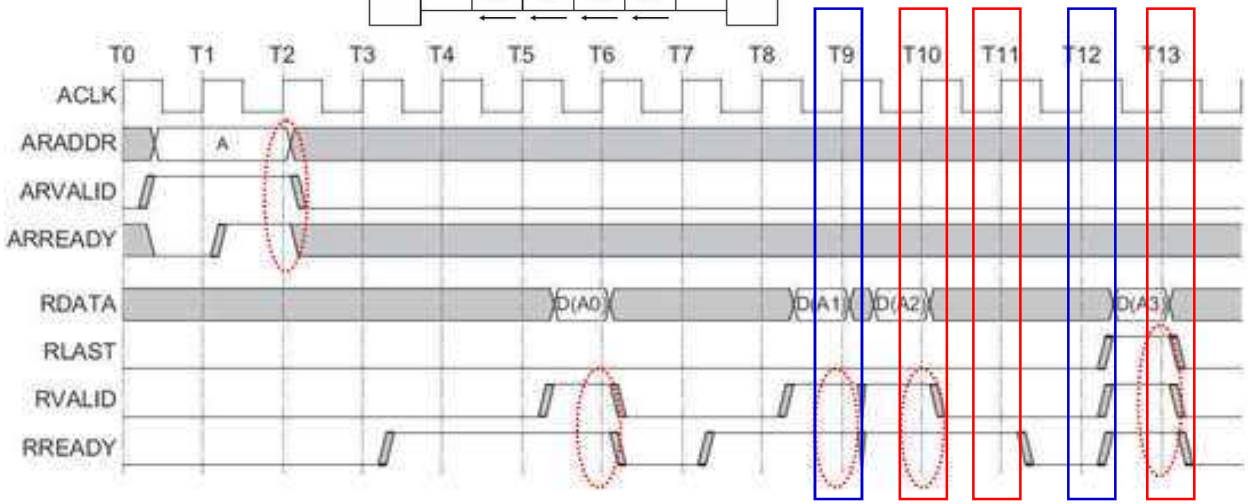
2. 시간 T5 에서 슬레이브의 읽기 데이터가 준비되어 슬레이브는 읽기 채널 RDATA 에 첫 번째 읽을 데이터를 실어 RVALID 신호를 Assert → 마스터는 데이터를 받을 준비가 되어 있었기 때문에 RREADY 이 Assert 되고, RVALID 과 RREADY 가 함께 Assert 되는 시간 T6 에 데이터가 전송됨.

AMBA → AXI(3)



➢ 기본 데이터 전송 프로토콜(2/4)

❖ AXI 사양의 기본 데이터 전송 프로토콜을 사용하여 읽기와 쓰기에 대한 4-Word 버스트 전송 트랜잭션을 예로 설명



[그림 1] Basic ***Read Burst Transaction***

1) 읽기 트랜잭션(2) [그림 1. 참고]

3. 시간 T9 에서 읽기 데이터가 준비되어 있는 슬레이브는 읽기 채널 RDATA 두 번째 읽을 데이터를 실어 RVALID 신호를 Assert ➔ 마스터는 데이터를 받을 준비가 있었기 때문에 RREADY 이 Assert 되고, RVALID 과 RREADY 가 함께 Assert 되는 시간 T10 에서 데이터가 전송.

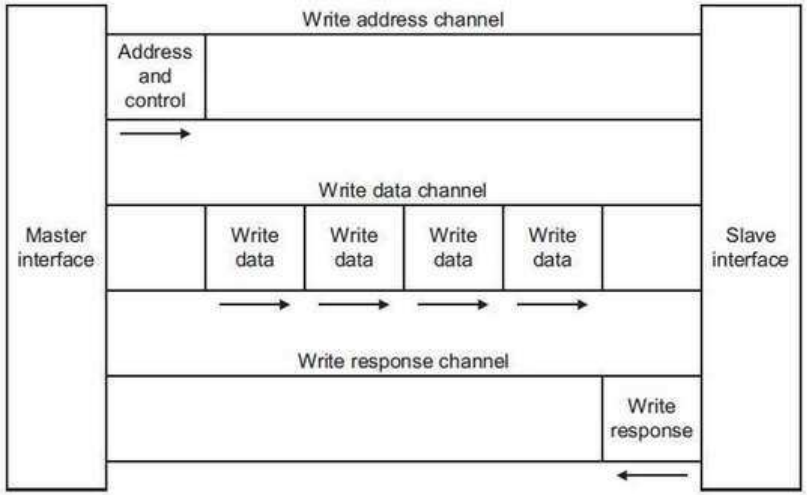
4. 같은 T10 에서 슬레이브는 다음 데이터가 준비되어 있기 때문에 읽기 채널 RDATA 세 번째 읽을 데이터를 실어 RVALID 신호를 Assert ➔ 마스터는 데이터를 받을 준비가 있었기 때문에 RREADY 이 Assert 되고, RVALID 과 RREADY 가 함께 Assert 되는 시간 T11 에서 데이터가 전송.

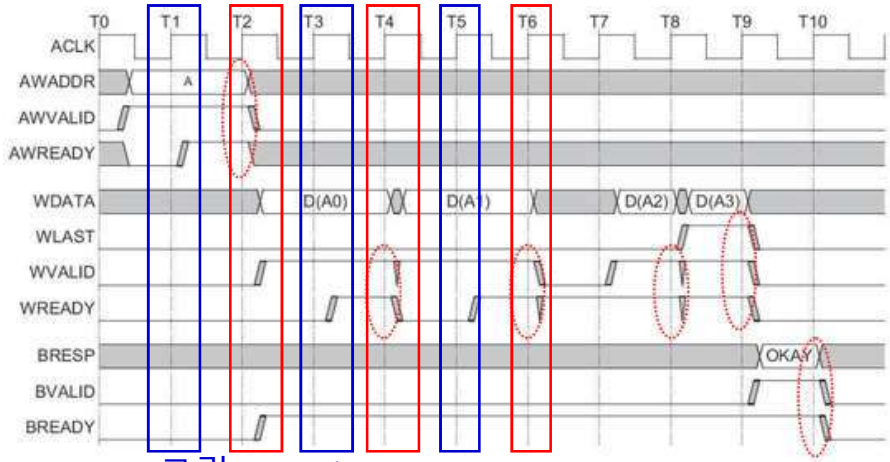
5. 시간 T12 에서 읽기 데이터가 준비되어 있는 슬레이브는 읽기 채널 RDATA 에 4 번째 읽을 데이터를 실어 RVALID 신호를 Assert ➔ 마지막 데이터이므로 RLAST가 Assert. 마스터는 데이터를 받을 준비가 있었기 때문에 RREADY 이 Assert 되고, RVALID 과 RREADY 가 함께 Assert 되는 시간 T13 에서 데이터가 전송

AMBA → AXI(3)

➢ 기본 데이터 전송 프로토콜(3/4)

❖ AXI 사양의 기본 데이터 전송 프로토콜을 사용하여 읽기와 쓰기에 대한 4-Word 버스트 전송 트랜잭션을 예로 설명





[그림 2] Basic ***Write Burst Transaction***

2) 쓰기 트랜잭션(1) [그림 2. 참고]

1. 마스터는 주소채널을 통해 전송 시작주소와 전송에 대한 제어 정보 (버스트 길이, 데이터 폭 전송, 유형 등)을 싣고 AWVALID 신호를 Assert (그림 2의 시간 T0). 슬레이브가 주소채널의 정보를 받을 수 있으므로, AWREADY 을 Assert 하고 응답합니다 (그림 2의시간 T1). 주소채널의 정보는 AWVAILD 와 AWREADY 가 Assert 되는 그림 9(b)의 시간 T2 에 전송

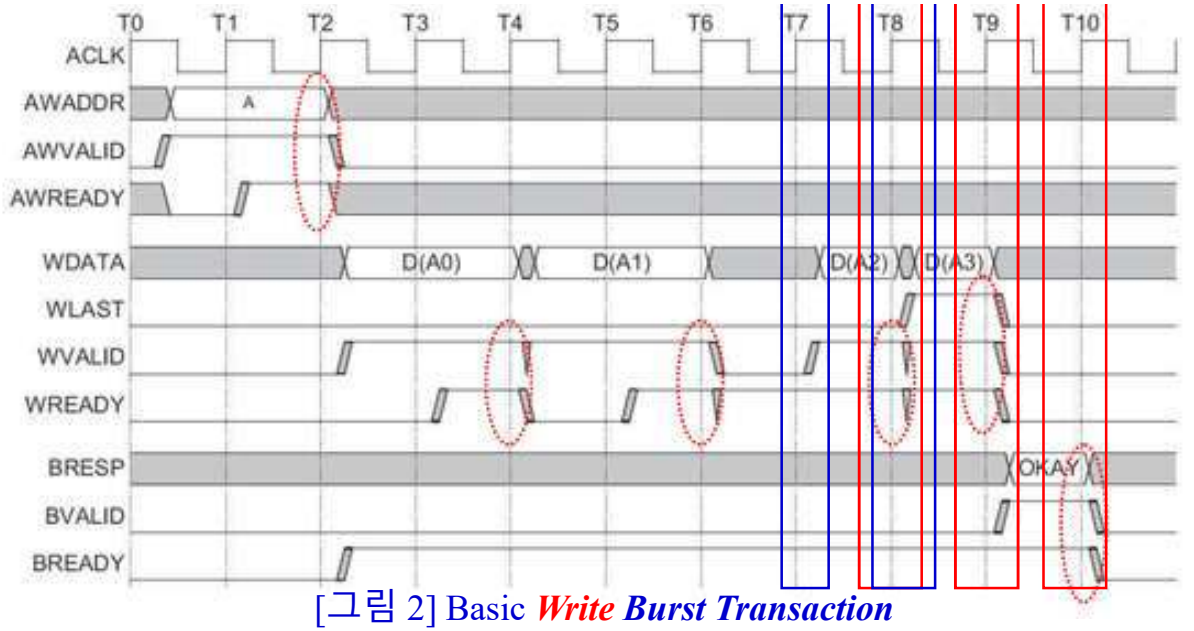
2. T2 이후 쓰기채널은 WDATA 에 첫째 쓰기 데이터를 실어 WVALID 신호를 Assert 합니다. 슬레이브는 데이터를 받을 준비가 되어 있기 때문에, 시간 T3 에서 WREADY 을 Assert 합니다. WVALID 과 WREADY 가 함께 Assert 되는 시간 T4 에 데이터가 전송

3. 마스터는 다음 데이터를 준비할 수 있었으므로, T4 에서 쓰기채널 WDATA 에 두 번째 쓰기 데이터를 실어 WVALID 신호를 Assert 합니다. 슬레이브는 데이터를 받을 준비가 되어 있기 때문에 시간 T5 에서 WREADY 을 Assert 합니다. WVALID 과 WREADY 가 함께 Assert 되는 시간 T6 에서 데이터가 전송

AMBA → AXI(3)

- 기본 데이터 전송 프로토콜(4/4)

❖ AXI 사양의 기본 데이터 전송 프로토콜을 사용하여 읽기와 쓰기에 대한 4-Word 버스트 전송 트랜잭션을 예로 설명

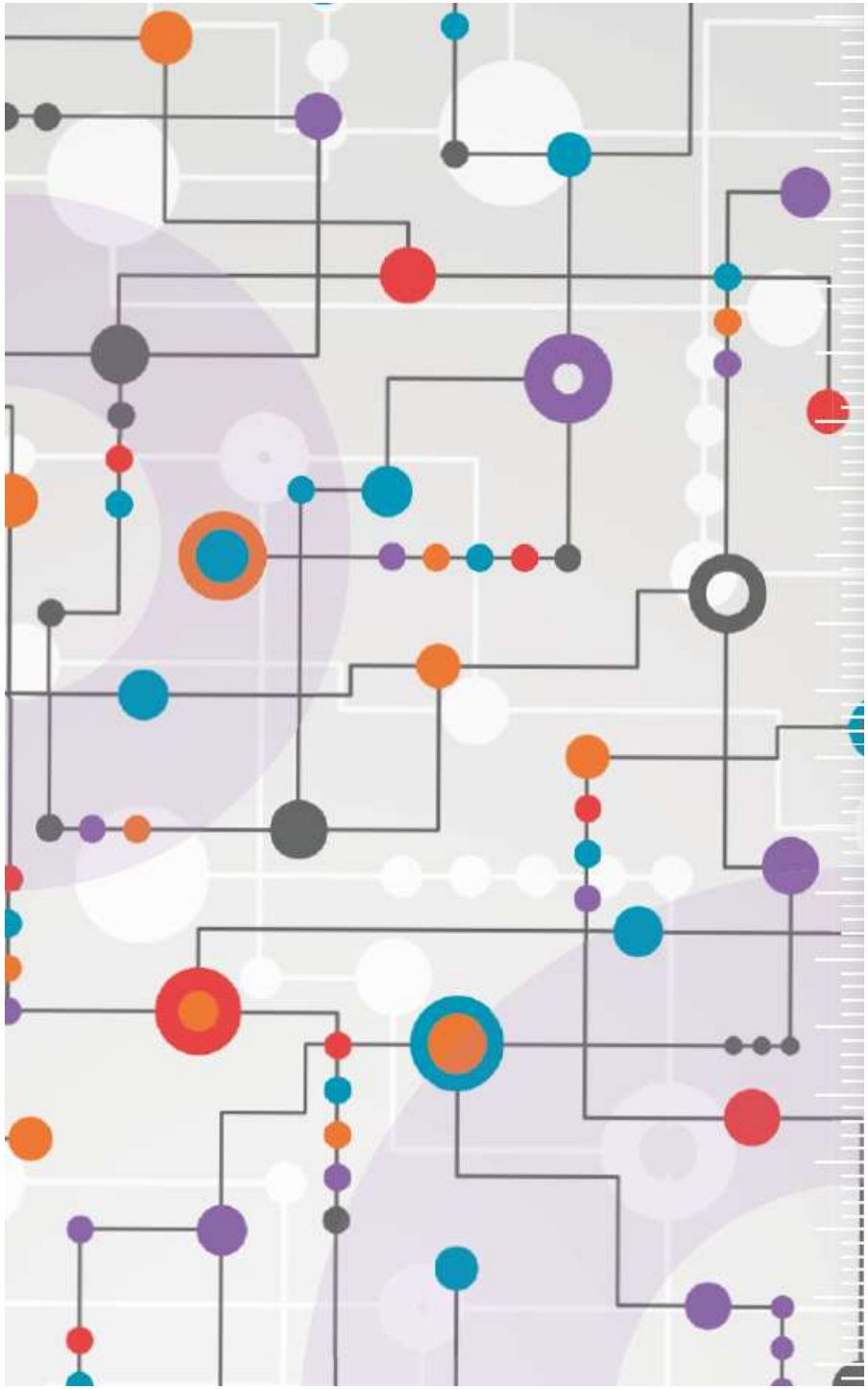


[그림 2] Basic ***Write Burst Transaction***

2) 쓰기 트랜잭션(2) [그림 2. 참고]

4. 마스터는 다음 데이터를 준비할 수 있었으므로, 쓰기 채널 WDATA 에 3 번째 기록 데이터를 실어 WVALID 신호를 Assert 합니다 (그림 2의 시간 T7). 슬레이브는 데이터를 받을 준비가 되어 있었기 때문에 WREADY 이 Assert 하고, WVALID 과 WREADY 가 함께 Assert 되는 시간 T8 에 데이터가 전송 ➔ 마스터는 동일한 T8 에 다음 데이터의 준비했기 때문에, 쓰기 채널 WDATA 에 4 번째 기록 데이터를 실어 WVALID 신호를 Assert 합니다. 전송 마지막 데이터이므로 WLAST 가 Assert 됩니다. WVALID 과 WREADY 가 함께 Assert 되는 시간 T9 에 데이터가 전송됩니다

5. 같은 T9 에서 슬레이브가 쓰기 데이터를 받았는지 확인하고, 쓰기 응답 채널에 응답 상태를 싣고 BVALID 을 Assert 합니다. 마스터는 응답 상태를 받을 준비가 되어 있었기 때문에 BREADY 가 Assert 되고, BVALID 와 BREADY 가 함께 Assert 되는 시간 T10 에서 BRESP 가 전송



수고하셨습니다.