HDD(Hardware Description Document)

AXI to APB Bridge Design

차세대반도체학과

우상욱(dndn3515@soongsil.ac.kr)

Revision History

Date	Version	Description	Author
2024/01/03	V0.0	HDD template	Inchul.Song
2024/01/07	V0.1	Intial HDD	Inchul.Song
2026/01/08	V1.0	Rough version	Sangwook.Woo
2026/01/13	V2.0	Rough version modified With Gpt	Sangwook.Woo
2026/01/14	V2.1	목차 수정 및 캡션 추가 보고서 전체 점검	Sangwook.Woo

Table of Contents

1.1. Overview

고성능 SoC의 backbone interconnect는 일반적으로 AXI bus를 기반으로 구성된다. AXI는 파이프라이닝과 burst 전송을 통해 높은 데이터 처리량을 제공하지만, peripheral의 control 및 status register 접근 용도로 사용하기에는 wire 수와 protocol 복잡도 측면에서 부담이 크다.

이러한 이유로 peripheral 제어 및 상태 확인을 위한 용도로는 구조가 단순한 APB bus가 여전히 널리 사용되고 있다. AXI 기반 시스템에서 APB 기반 peripheral을 사용하기 위해서는 AXI protocol을 APB protocol로 변환하는 브리지 로직이 필요하다.

본 문서는 AXI 기반 시스템에서 APB 기반 peripheral 접근을 가능하게 하는 AXI2APB Bridge의 하드웨어 설계에 대한 설명을 목적으로 한다.

1.1.1. Description

AXI와 APB는 SoC 내부에서 서로 다른 목적을 위해 사용되는 대표적인 버스 프로토콜이다. AXI는 고성능 데이터 전송을 위한 버스로, 파이프라이닝과 burst 전송을 통해 높은 처리량을 제공한다. 반면 APB는 구조가 단순하여 제어 레지스터 접근과 저대역폭 주변장치 제어에 적합하다.

본 IP는 이러한 서로 다른 버스 프로토콜을 연결하기 위한 AXI to APB Bridge로서, AXI 기반 시스템에서 APB 기반 주변장치에 접근할 수 있도록 한다. AXI master로부터 전달되는 read 및 write 요청을 수신하여, 이를 APB protocol에 맞는 transaction으로 변환하고 그 결과를 다시 AXI 응답으로 반환하는 역할을 수행한다.

1.1.2. Feature

본 AXI2APB Bridge는 구현 단순성과 동작 안정성을 고려한 구조를 기반으로 설계되었으며, 다음과 같은 하드웨어 동작 특성을 가진다.

1) AXI to APB Transaction 변환 구조

- AXI read 및 write transaction을 APB **single transfer** 단위로 변환하여 처리한다.
- 각 AXI transaction은 APB protocol의 setup phase(PSEL = 1, PENABLE = 0)와 enable phase (PSEL = 1, PENABLE = 1)로 분리되어 순차적으로 수행된다.
- APB transaction 완료 시점에 AXI 측으로 read data 또는 write response가 반환된다.

2) Burst Transaction 처리 방식

- AXI burst transaction을 지원한다.
- Burst 전송은 **각 beat를 독립적인 APB single transfer로 변환**하여 순차 처리한다.
- 모든 beat는 동일한 순서로 처리되며, burst 전송 중 reordering은 발생하지 않는다.
- Burst의 마지막 beat에서는 AXI read channel에서 **RLAST 신호가 생성**된다.
- 본 설계는 다음 기능을 지원하지 않는다.
  - Out-of-order transaction
  - Multiple outstanding transaction
  - AXI ID 기반 transaction 관리

3) APB Wait-state 대응

- APB slave가 wait-state를 요구하는 경우(PREADY = 0),
해당 APB transaction이 완료될 때까지 **enable phase를 유지**한다.
- PREADY가 high로 전환되는 시점에 APB transaction이 완료되며,
이후 AXI response 또는 다음 beat 처리가 진행된다.
- 이를 통해 다양한 APB slave의 동작 지연을 안정적으로 처리할 수 있다.

4) Error Response 처리 방식

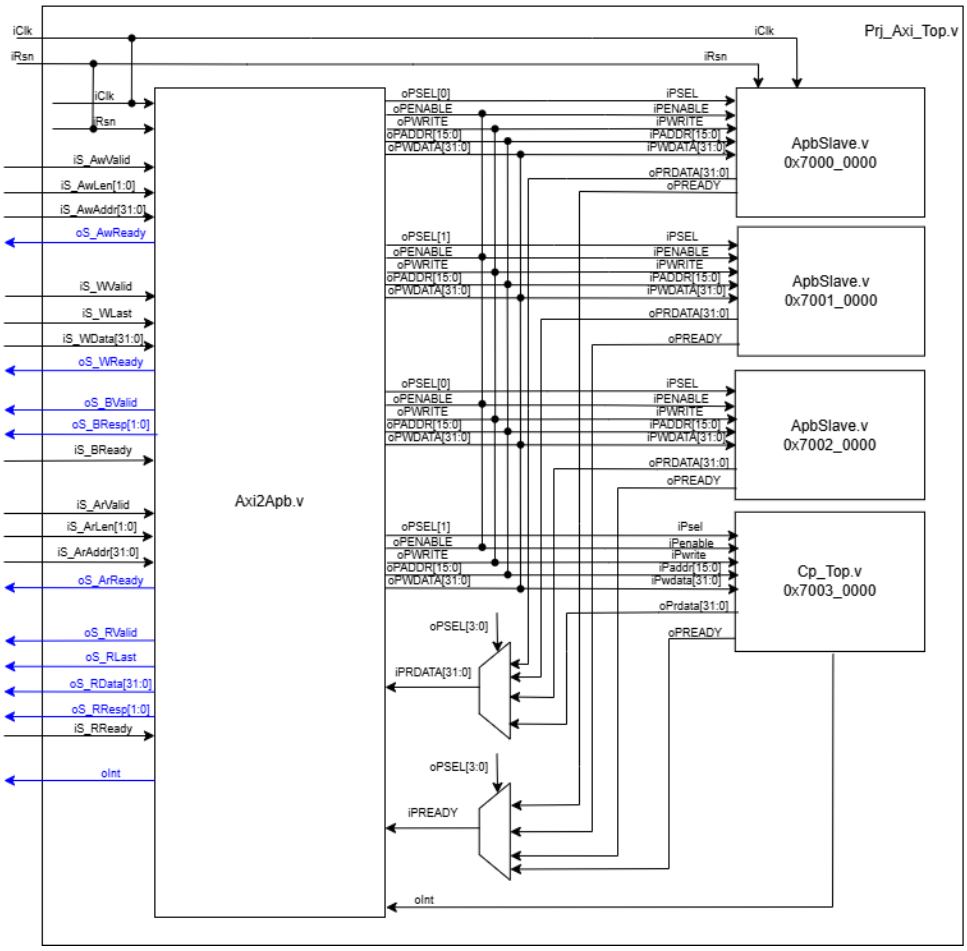
- 지원하지 않는 주소 영역으로 AXI 접근이 발생할 경우, AXI protocol에 따라 error response(BRESP 또는 RRESP)를 반환한다.
- Error response는 APB transaction 이전 단계에서 판단되어 생성된다.
- APB slave error 신호(PSLVERR)는 본 설계에서 고려하지 않는다.

5) 제어 구조 및 설계 제약

- Read transaction과 write transaction은 **각각 독립적인 FSM**으로 제어된다.
- 하나의 transaction이 완료된 이후에 다음 transaction을 수신하는 **단일 outstanding transaction 구조**를 사용한다.
- APB는 read와 write를 동시에 처리할 수 없으므로, APB access는 순차적으로 수행된다.

1.2. Overall block

1.2.1. Block diagram



<Figure 1. Overall Diagram>

Figure1은 설계한 전체 diagram이다. AXI master 신호를 받아 multiple slave에게 전달한다. 이번 프로젝트에서는 4개의 Slave로 이루어져 있고 PSEL 신호로 구분된다. PSEL 신호를 통해 Read Data와 Ready 신호를 구분하여 master에 전달한다. 두개의 mux는 PSEL 신호를 활용해서 Slave 4개의 신호를 muxing하여 bridge에 전달한다. Slave의 주소값은 16진수의 5번째 자리로 구분된다. 따라서 PSEL을 4비트로 하고 0001을 기준으로 5번째 자리의 값만큼 shift하면 사용한다.

1.2.2. IO description

Table 1 Input

Name	Bit width	Initial value	Source Block	Description
iClk	1	0	CMU	300MHz clock input
iRsn	1	1	CMU	Active low reset
iS_AWValid	1	0	AXI bus	AXI AW address valid
iS_AwLen	2	N/A	AXI bus	AXI AW Burst Length
iS_AwAddr	32	N/A	AXI bus	AXI AW address
iS_WValid	1	0	AXI bus	AXI W data valid
iS_WLast	1	0	AXI bus	AXI W transfer last signal
iS_WData	32	0	AXI bus	AXI W data
iS_BReady	1	0	AXI bus	AXI B response ready
iS_ArValid	1	0	AXI bus	AXI AR address valid
iS_ArLen	2	N/A	AXI bus	AXI AR Burst Length
iS_ArAddr	32	N/A	AXI bus	AXI AR address
iS_RReady	1	0	AXI bus	AXI R data ready

Table 2 Output

Name	Bit width	Initial value	Destination Block	Description
oS_AwReady	1	N/A	AXI bus	AXI AW ready
oS_WReady	1	0	AXI bus	AXI W slave write done
oS_BValid	1	0	AXI bus	AXI B response valid
oS_BResp	2	0	AXI bus	AXI B response (OK,ERROR)
oS_ArReady	1	0	AXI bus	AXI AR read ready
oS_RValid	1	0	AXI bus	AXI R data valid
oS_RLast	1	0	AXI bus	AXI R burst last signal
oS_RData	32	N/A	AXI bus	AXI R data
oS_RResp	2	0	AXI bus	AXI R response
oInt	1	0	AXI bus	Interrupt signal from APB Slave

1.2.3. Memory list

Table 3 Memory list

Name	Location	Type	Size	Description
InBuf	Cp_Top.InBuf	SP SRAM	512*32	Max packet size: 2kBytes APB data bit width : 32bit, so 512depth
OutBuf	Cp_Top.OutBuf	SP SRAM	512*32	Max packet size: 2kBytes APB data bit width : 32bit, so 512depth

1.2.4. Register map

B	AHB2APB or AXI2APB			
	NAME	Start address	End address	Description
SB	Register Block1	0x7000_0000	0x7000_FFFF	APB Slave1
SB	Register Block2	0x7001_0000	0x7001_FFFF	APB Slave2
SB	Register Block3	0x7002_0000	0x7002_FFFF	APB Slave3
SB	AES Block	0x7000_3000	0x7003_FFFF	AES Encryption only block

<Figure 2. Slave Block 별 Address 범위>

	NAME	Offset	Range	Field	Reset	Access	Description
B	Register Block1	0x7000_0000	0xFFFF				
R	InOutA	0x0000	[31:0]		0x0	R/W	
F			[31:0]	rA	0x0		32bit unsigned Data input
R	InOutB	0x0004	[31:0]		0x0	R/W	
F			[31:0]	rB	0x0		32bit unsigned Data input
R	InOutC	0x0008	[31:0]		0x0	R/W	
F			[31:0]	rC	0x0		32bit unsigned Data input
R	InOutD	0x000C	[31:0]		0x0	R/W	
F			[31:0]	rD	0x0		32bit unsigned Data input

	NAME	Offset	Range	Field	Reset	Access	Description
B	Register Block1	0x7001_0000	0xFFFF				
R	InOutA	0x1000	[31:0]		0x0	R/W	
F			[31:0]	rA	0x0		32bit unsigned Data input
R	InOutB	0x0004	[31:0]		0x0	R/W	
F			[31:0]	rB	0x0		32bit unsigned Data input
R	InOutC	0x0008	[31:0]		0x0	R/W	
F			[31:0]	rC	0x0		32bit unsigned Data input
R	InOutD	0x000C	[31:0]		0x0	R/W	
F			[31:0]	rD	0x0		32bit unsigned Data input

	NAME	Offset	Range	Field	Reset	Access	Description
B	Register Block1	0x7002_0000	0xFFFF				
R	InOutA	0x0000	[31:0]		0x0	R/W	
F			[31:0]	rA	0x0		32bit unsigned Data input
R	InOutB	0x0004	[31:0]		0x0	R/W	
F			[31:0]	rB	0x0		32bit unsigned Data input
R	InOutC	0x0008	[31:0]		0x0	R/W	
F			[31:0]	rC	0x0		32bit unsigned Data input
R	InOutD	0x000C	[31:0]		0x0	R/W	
F			[31:0]	rD	0x0		32bit unsigned Data input

<Figure 3. Slave1~Slave3 Register Map>

	NAME	Offset	Range	Field	Reset	Access	Description
B	AES(Cipher only)	0x7003_0000	0xFFFF				
S	Common						
R	StartCmd	0x0000	[0:0]		0x0	W	NatHw start command
F			[0:0]	StartCmd	0x0		Set to 1 by SW : NatHw start
R	DataByteSize	0x0004	[31:0]		0x0	R/W	Input data byte size (1~2048 value)
F			[11:0]	DataWdSize	0x0		
S	Control						
R	AesKey_31_0	0x2000	[31:0]		0x0	R/W	128bit AES key value
F			[31:0]	AesKey_31_0	0x0		AES key[31:0]
R	AesKey_63_32	0x2004	[31:0]		0x0	R/W	128bit AES key value
F			[31:0]	AesKey_63_32	0x0		AES key[63:32]
R	Aeskey_95_64	0x2008	[31:0]		0x0	R/W	128bit AES key value
F			[31:0]	AesKey_95_64	0x0		AES key[95:64]
R	AesKey_127_96	0x200C	[31:0]		0x0	R/W	128bit AES key value
F			[31:0]	AesKey_127_96	0x0		AES key[127:96]
~~R~~	~~AesDirection~~	~~0x2010~~	~~[0:0]~~		~~0x0~~	~~R/W~~	~~AES direction~~
~~F~~			~~[0:0]~~	~~AesDirection~~	~~0x0~~		~~1'b1: Encyption, 1'b0: Decryption~~
S	InBuf						
R	InBufAddr_0	0x4000	[31:0]		0x0	W	AES InBuf (2kB : 512depth * 4Bytes)
F			[31:0]	InBufAddr_0	0x0		InBuf address 0 data write
R	InBufAddr_1	0x4004	[31:0]		0x0	W	
F			[31:0]	InBufAddr_1	0x0		InBuf address 1 data write
R	_	_	[31:0]		0x0	W	_
F			[31:0]	_	0x0		_
R	InBufAddr_511	0x47FC	[31:0]		0x0	W	
F			[31:0]	_	0x0		InBuf address 511 data write
S	OutBuf						
R	OutBuf1_Addr_0	0x6000	[31:0]		0x0	R	AES OutBuf #1 (2kB : 512depth * 4Bytes)
F			[31:0]	OutBuf1_Addr_0	0x0		OutBuf #1 address 0 data read
R	OutBuf1_Addr_1	0x6004	[31:0]		0x0	R	
F			[31:0]	OutBuf1_Addr_1	0x0		OutBuf #1 address 1 data read
R	_	_	[31:0]		0x0	R	_
F			[31:0]	_	0x0		_
R	OutBuf1_Addr_511	0x67FC	[31:0]		0x0	R	_
F			[31:0]	OutBuf1_Addr_511	0x0		OutBuf #1 address 511 data read
S	Int						
R	IntEnable	0xA000	[0:0]		0x0	R/W	AES interrupt enable register
F			[0]	IntEnable	0x0		Set to 1 by SW : AES interrupt pending register Enable
R	IntPending	0xA004	[0:0]		0x0	R/W	AES interrupt pending register
F			[0]	IntPending	0x0		When interrupt enable is On and AES operation is done, HW write 1 to make Interrupt, If SW reads 1 and then writes 1 to clear
R	IntMask	0xA008	[0:0]		0x0	R/W	AES interrupt mask register
F			[0]	IntMask	0x0		Set to 1 by SW : AES interrupt transfer to interrupt output
F			[31:0]	OutBuf1_Addr_1	0x0		OutBuf #1 address 1 data read
R	_	_	[31:0]		0x0	R	_
F			[31:0]	_	0x0		_
R	OutBuf1_Addr_511	0x67FC	[31:0]		0x0	R	_
F			[31:0]	OutBuf1_Addr_511	0x0		OutBuf #1 address 511 data read
S	Int						
R	IntEnable	0xA000	[0:0]		0x0	R/W	AES interrupt enable register
F			[0]	IntEnable	0x0		Set to 1 by SW : AES interrupt pending register Enable
R	IntPending	0xA004	[0:0]		0x0	R/W	AES interrupt pending register
F			[0]	IntPending	0x0		When interrupt enable is On and AES operation is done, HW write 1 to make Interrupt, If SW reads 1 and then writes 1 to clear
R	IntMask	0xA008	[0:0]		0x0	R/W	AES interrupt mask register
F			[0]	IntMask	0x0		Set to 1 by SW : AES interrupt transfer to interrupt output

<Figure 4. Cp_Top Register Map>

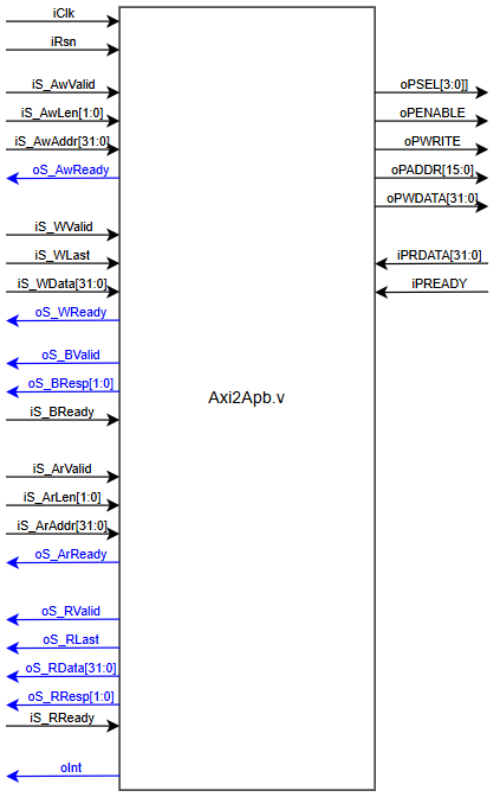
1.3. Sub block description

본 장에서는 AXI2APB Bridge 내부를 구성하는 sub block들의 계층 구조와 각 sub block의 기능 및 동작을 설명한다. AXI2APB Bridge는 AXI transaction을 APB transaction으로 변환하기 위한 제어 로직을 중심으로 구성되어 있으며, read path와 write path를 분리한 구조를 가진다.

각 sub block은 AXI Slave Interface로부터 전달되는 요청을 처리하여 APB Master Interface 신호를 생성하고, APB transaction 완료 이후 AXI response를 생성하는 역할을 수행한다. 내부 제어는 FSM 기반으로 구현되어 있으며, 단일 outstanding transaction 구조를 따른다.

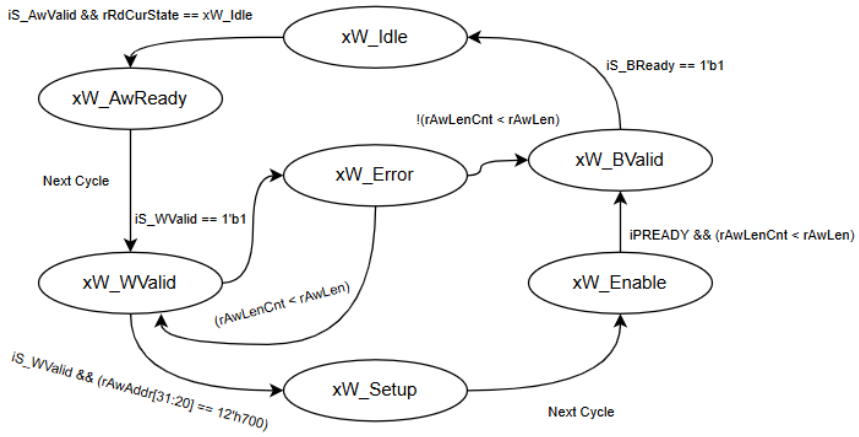
1.3.1. Sub block #1

이번 프로젝트에서 메인이 되는 AXI to APB Conversion Block이다. 이 모듈은 AXI master로부터 전달되는 read 및 write transaction을 수신하여, 이를 APB protocol에 맞는 신호로 변환하는 핵심 제어 블록이다. 본 sub block은 AXI read path와 write path를 각각 독립적으로 처리하며, 내부 FSM을 통해 transaction의 순차적인 진행을 제어한다.



<Figure 5. Aix2Apb Diagram>

FSM 기반 컨트롤 로직을 구성하였다. 사용된 FSM은 두개로 Write FSM과 Read FSM이다.

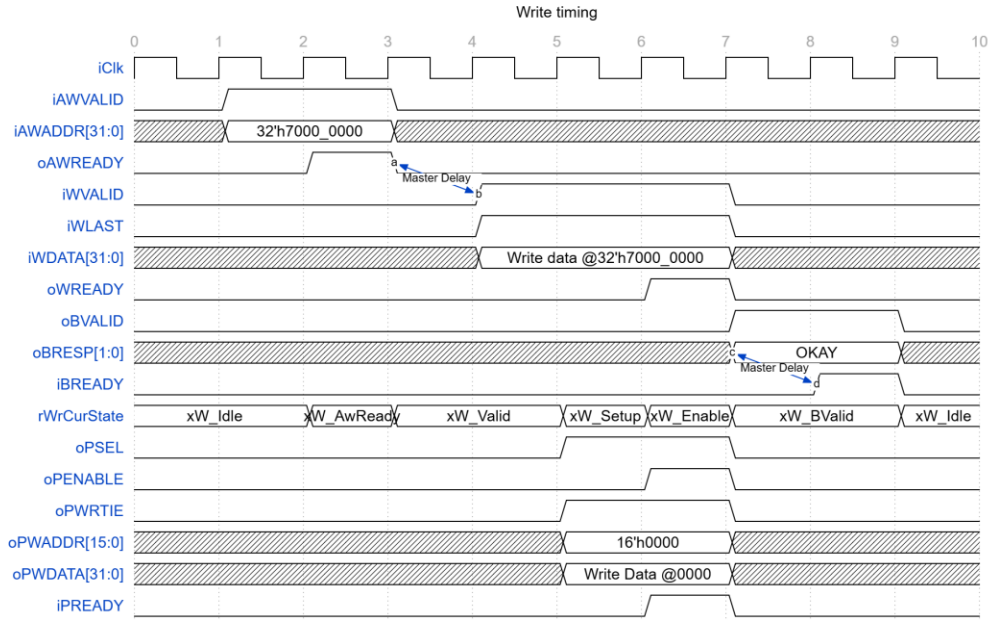


<Figure 6. Axi2Apb의 Write FSM>

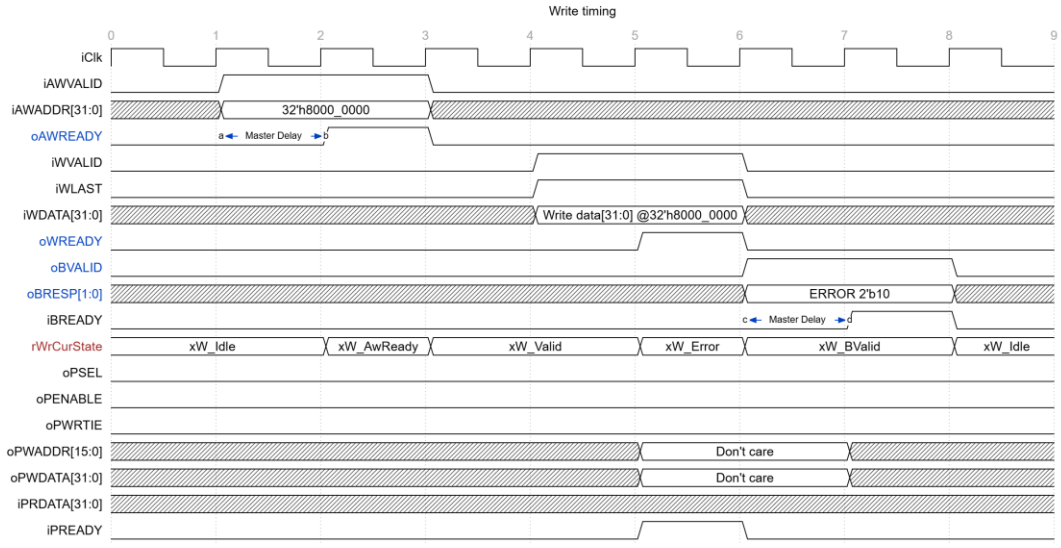
AXI write transaction 은 AW channel 과 W channel 의 valid/ready handshake 로 시작한다. AWVALID/AWREADY handshake 완료 시 write address(AWADDR) 및 burst length(AWLEN)가 내부에 latch 되며, WVALID/WREADY handshake 를 통해 write data(WDATA)가 latch 된다. Burst write 의 경우 WLAST 로 마지막 beat 를 판단한다.

AXI write 요청이 수락되면 APB write transaction 으로 변환된다.

- **Setup phase**: PSEL 을 assert 하고 PADDR/PWRITE/PWDATA 를 설정한다. (PENABLE=0)
- **Enable phase**: 다음 cycle 에 PENABLE 을 assert 한다. (PSEL=1, PENABLE=1)
- **Wait-state 처리**: PREADY=0 인 동안 enable phase 를 유지하여 APB slave 가 준비될 때까지 대기한다.
- **Complete**: PREADY=1 이 되는 cycle 에 APB write 를 완료로 판단하고, AXI write response channel 에서 BVALID 를 assert 한다. 정상 접근 시 BRESP=OKAY, unsupported address 접근 시 BRESP=ERROR 를 반환한다.

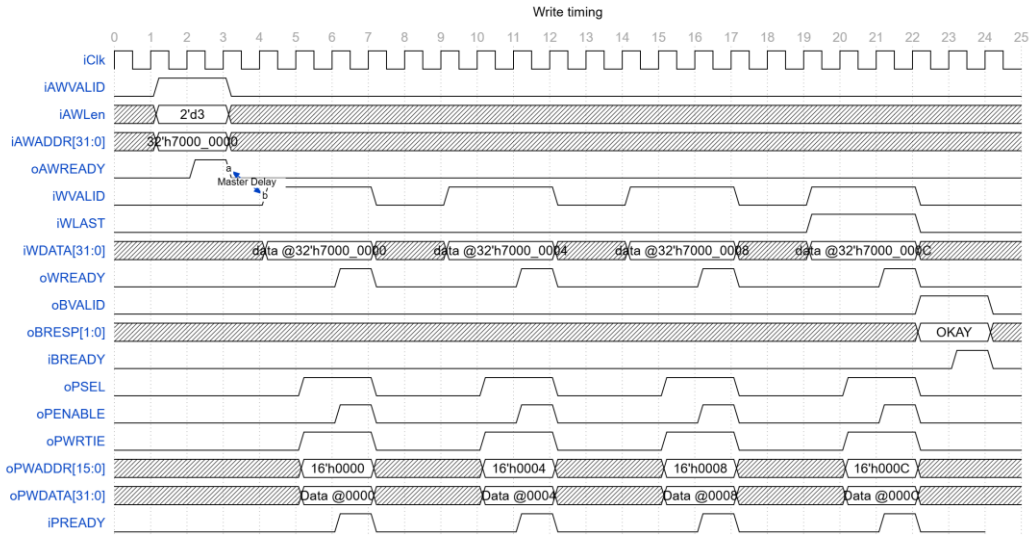


<Figure 7. Axi2Apb 의 Single beat Write Timing>



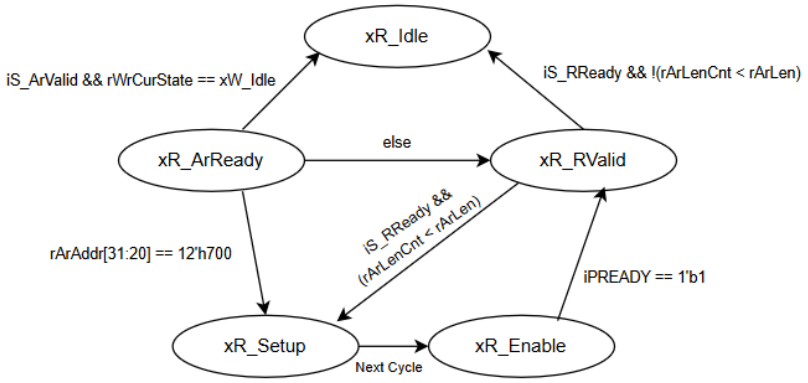
<Figure 8. Axi2Apb 의 Write error Timing>

잘못된 address에 접근 했을시에 동작이다. 이 경우 setup,enable state에는 점프하지 않고 WValid, Error를 통하고 Bvalid state에서 error를 응답한다.



<Figure 9. Axi2Apb 의 4-burst Write Timing>

Busrt 동작의 경우 single과 유사하다. AwLen을 통해 burst 회수를 약속하고 내부 counter를 통해 address를 증가 시켜 WVALID와 WDATA를 받아 write 동작을 수행한다.



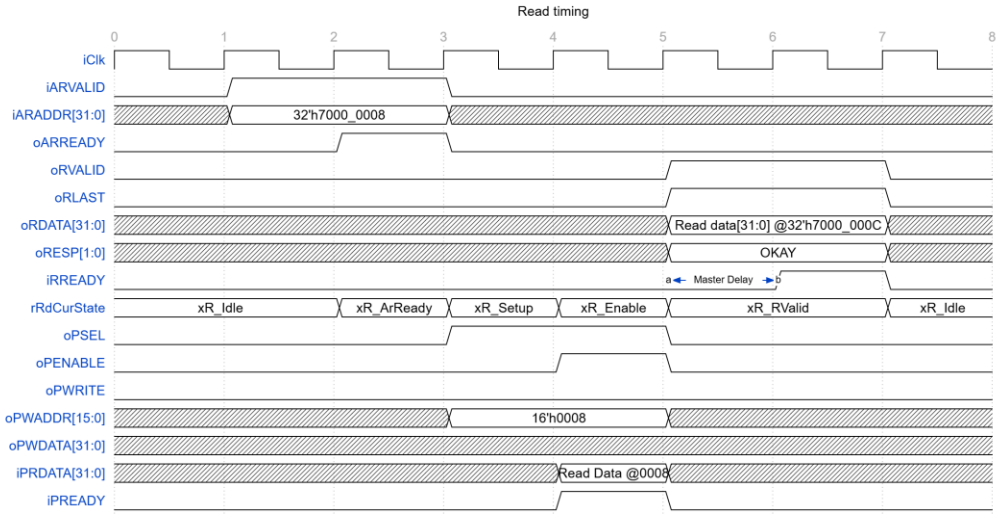
<Figure 10. Axi2Apb 의 Read FSM>

AXI read transaction은 ARVALID/ARREADY handshake로 시작한다.
Handshake 완료 시 read address(ARADDR) 및 burst length(ARLEN)가 내부에 latch된다.

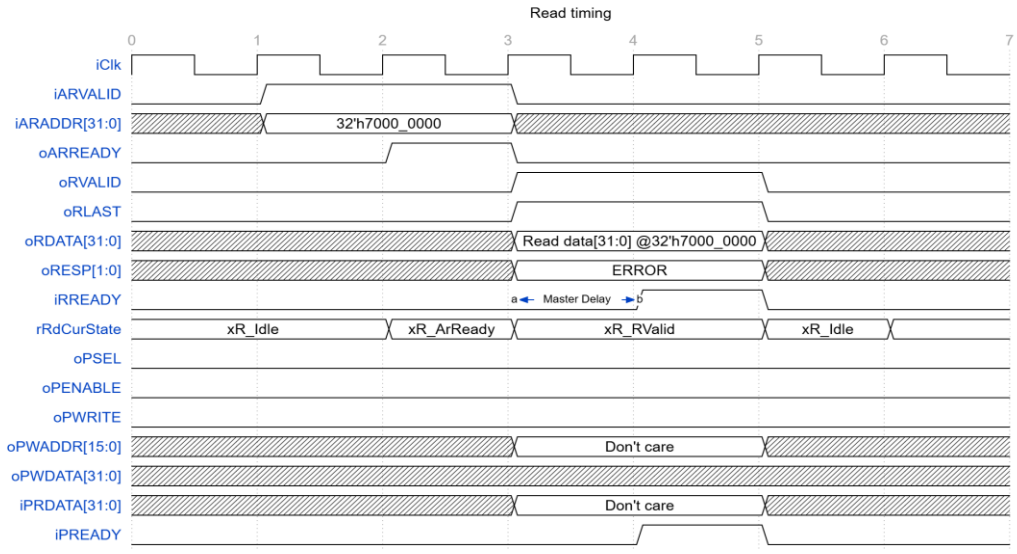
AXI read 요청이 수락되면 APB read transaction으로 변환된다.

- **Setup phase**: PSEL을 assert하고 PADDR를 설정하며 PWRITE=0으로 설정한다. (PENABLE=0)
- **Enable phase**: 다음 cycle에 PENABLE을 assert한다.
- **Wait-state 처리**: PREADY=0이면 enable phase를 유지한다.

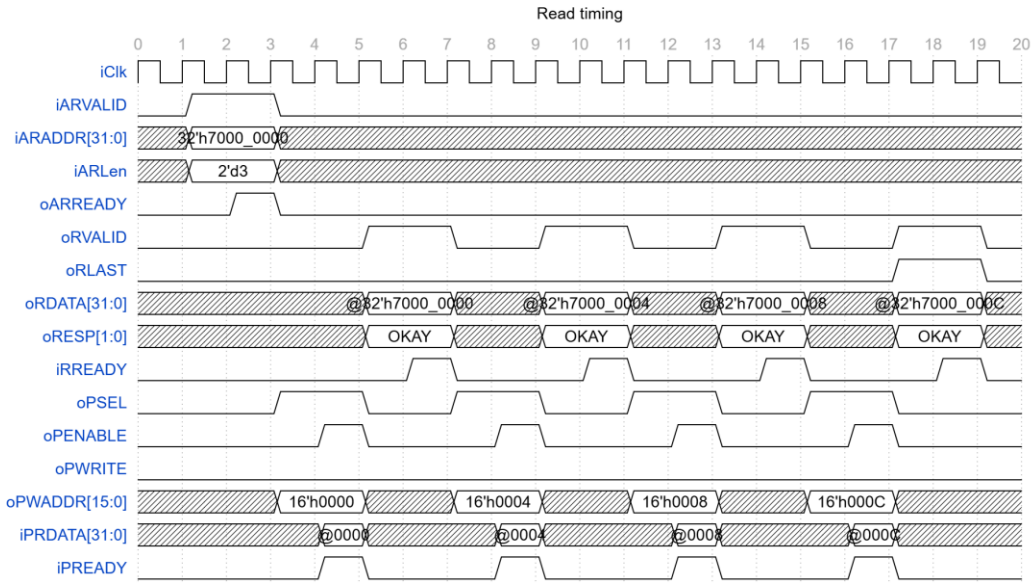
• **Complete 및 반환**: PREADY=1이 되는 cycle에 APB read data를 수집하고, AXI read data channel에서 RVALID와 함께 RDATA를 반환한다. Burst read의 마지막 beat에서는 RLAST를 assert한다. 정상 접근 시 RRESP=OKAY, unsupported address 접근 시 RRESP=ERROR를 반환한다.



<Figure 11. Axi2Apb 의 Single-beat Read Timing>



<Figure 12. Axi2Apb 의 Single-beat Read Error Timing>

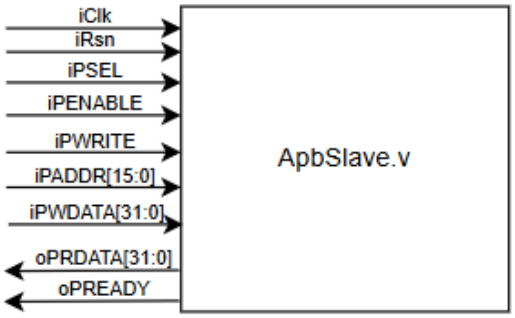


<Figure 13. Axi2Apb 의 4-burst Read Timing>

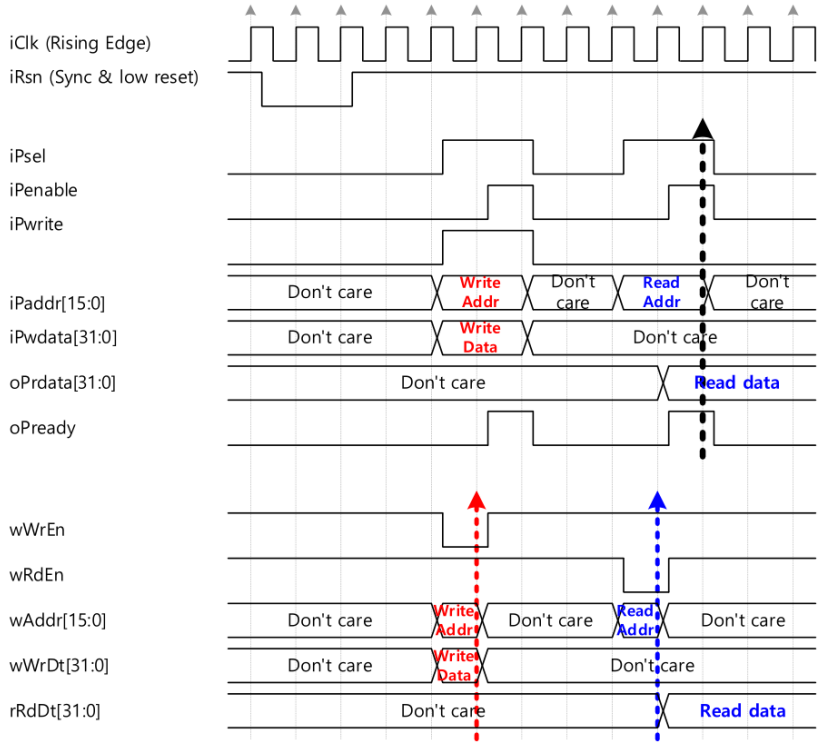
AXI burst transaction을 지원하되, burst를 구성하는 각 beat를 **독립적인 APB single transfer**로 순차 변환하여 처리한다. beat 간 reordering은 발생하지 않으며 순서대로 처리된다. out-of-order transaction, multiple outstanding transaction, AXI ID 기반 관리 기능은 지원하지 않는다. Write burst의 경우 WLAST를 기준으로 burst 종료를 판단하고, burst 완료 이후 BVALID를 발생시킨다. 또한 Write와 Read 모두 slave error는 고려하지 않고 Write와 Read의 동시 접근은 없다고 가정한다.

1.3.2. Sub block #2

ApbSlave는 R/W 32 bit register 4개가 내부에 있고 apb 신호에 맞춰 동작한다.

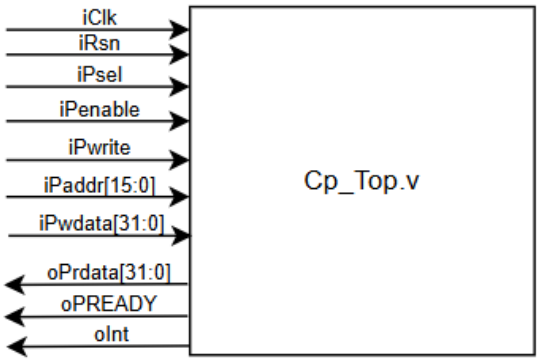


<Figure 14. ApbSlave Block Diagram>

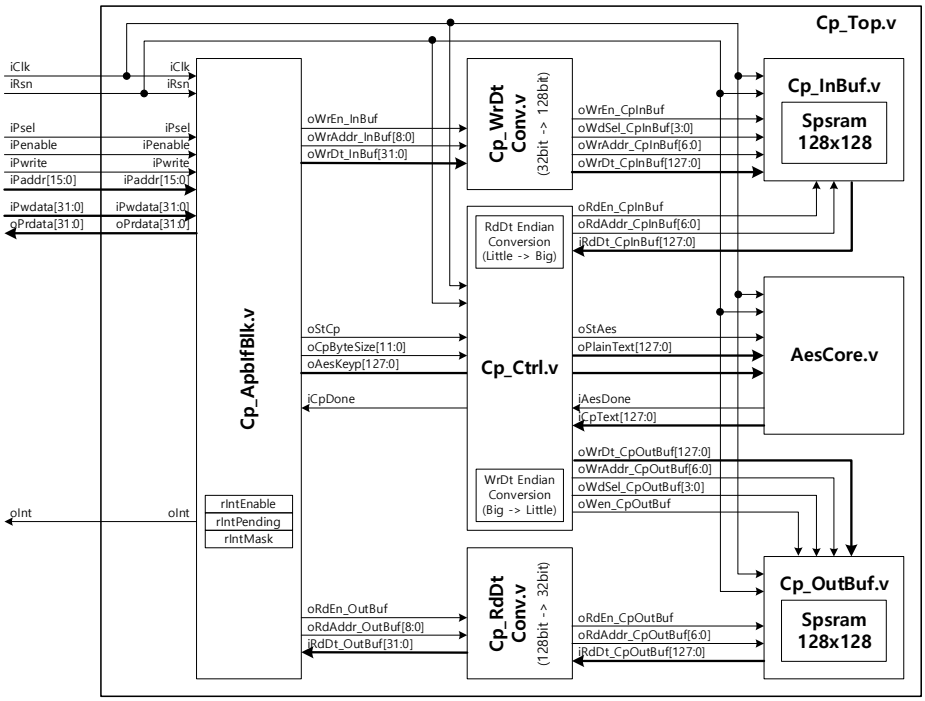


<Figure 15. Apb Slave 의 R/W timing diagram>

1.3.3. Sub block #3



<Figure 16. Cp_Top Block Diagram>



<Figure 17. Cp_Top 의 Overall Block Diagram>

본 sub block은 AES-128 암호화를 수행하는 연산 블록으로, APB 기반 register/memory-mapped interface를 통해 제어된다. AXI2APB bridge를 통해 전달된 APB transaction을 수신하여 입력 데이터/제어 레지스터를 설정하고, 암호화 연산 결과를 내부 저장공간에 저장한다. 암호화 연산이 완료되면 interrupt를 발생시켜 상위(master)로 완료 상태를 알린다.

본 HDD는 AXI2APB bridge 설계(AXI↔APB 변환 및 응답 처리)를 중심으로 작성되었으므로, AES 연산 블록의 내부 암호화 로직 및 상세 register 정의는 별도 문서를 참조한다.

1.4. Further work

본 AXI2APB Bridge는 구현 단순성과 검증 용이성을 우선하여 설계되었으며, 안정적인 AXI–APB 변환 동작을 목표로 기본 기능 위주로 구현되었다. 그러나 AXI protocol이 제공하는 full feature를 고려할 때, 본 설계에는 몇 가지 구조적 제약이 존재하며, 향후 다음과 같은 방향으로 확장이 가능하다.

(1) Outstanding 및 Out-of-order Transaction 지원

현재 설계는 **단일 outstanding transaction 구조**를 기반으로 하며, 하나의 transaction 또는 burst beat가 완료된 이후에 다음 요청을 처리한다. 이로 인해 AXI protocol이 지원하는 multiple outstanding transaction 및 out-of-order transaction 기능은 활용되지 못한다.

향후에는 AXI ID(AWID/ARID)를 활용하여 여러 transaction을 동시에 관리하고, ID 기반 response 반환을 지원하는 구조로 확장할 수 있다. 이를 위해 transaction 정보를 저장하기 위한 request queue 및 reorder buffer 도입이 필요하며, 이를 통해 AXI 측 throughput을 향상시킬 수 있을 것으로 기대된다.

(2) Burst 처리 성능 개선

현재 burst transaction은 각 beat를 독립적인 APB single transfer로 변환하여 순차 처리한다. APB는 구조적으로 burst 전송을 지원하지 않으므로, APB access latency가 긴 경우 AXI burst 전체 처리 시간이 증가하는 한계가 존재한다.

이를 개선하기 위해, AXI 측 burst 요청을 내부 queue에 저장하고 APB access를 비동기적으로 처리하는 방식의 **작업 큐 기반 구조**를 도입할 수 있다. 이러한 구조를 통해 AXI–APB 간 throughput mismatch를 완화하고, AXI master의 stall을 줄일 수 있다.

(3) Read/Write 동시 처리 구조 개선

AXI protocol은 read channel과 write channel이 독립적으로 동작할 수 있으나, APB protocol은 구조적으로 read와 write를 동시에 수행할 수 없다. 본 설계에서는 이러한 APB 특성을 고려하여 read/write를 순차적으로 처리한다.

향후에는 read/write 요청을 내부 buffer에 분리 저장하고, APB access는 arbitration을 통해 순차 수행하되 AXI side에서는 동시 요청 수신이 가능하도록 구조를 확장할 수 있다. 이를 통해 AXI interface 측의 병렬성을 보다 효과적으로 활용할 수 있다.

(4) Clock Domain 확장 및 Timing 개선

현재 설계는 AXI와 APB가 동일 clock domain에서 동작하는 구조를 가정한다. 그러나 실제 SoC 환경에서는 AXI와 APB가 서로 다른 clock domain에서 동작하는 경우가 많다.

향후에는 **asynchronous FIFO**를 도입하여 AXI–APB 간 clock domain crossing(CDC)을 지원하는 구조로 확장할 수 있다. 이를 통해 높은 AXI clock frequency 환경에서도 안정적인 동작이 가능하도록 개선할 수 있다.

(5) Error Handling 확장

현재 설계는 unsupported address 접근에 대해서만 error response를 반환하며, APB slave error 신호(PSLVERR)는 고려하지 않는다. 향후에는 PSLVERR를 반영하여 보다 정교한 error reporting을 지원하고, error 발생 원인을 구분할 수 있도록 확장할 수 있다.

1.5. Appendix

AMBA AXI Protocol Specification (ARM IHI 0022)

AXI protocol의 channel 구조, handshake 방식, burst 및 response 정의를 참고하여 AXI interface 동작을 설계하였다.

AMBA APB Protocol Specification v2.0 (ARM IHI 0024C)

APB protocol의 setup/access phase, wait-state 처리 및 read/write transfer 동작을 참고하여 APB interface를 구현하였다.

AXI bus protocol 이해 및 AXI interface를 갖는 HW Block 설계 및 검증 _11주차 pdf

AXI2APB 및 AHB2APB 변환 구조, FSM 설계 방법, timing diagram 작성 방식을 참고하였다.