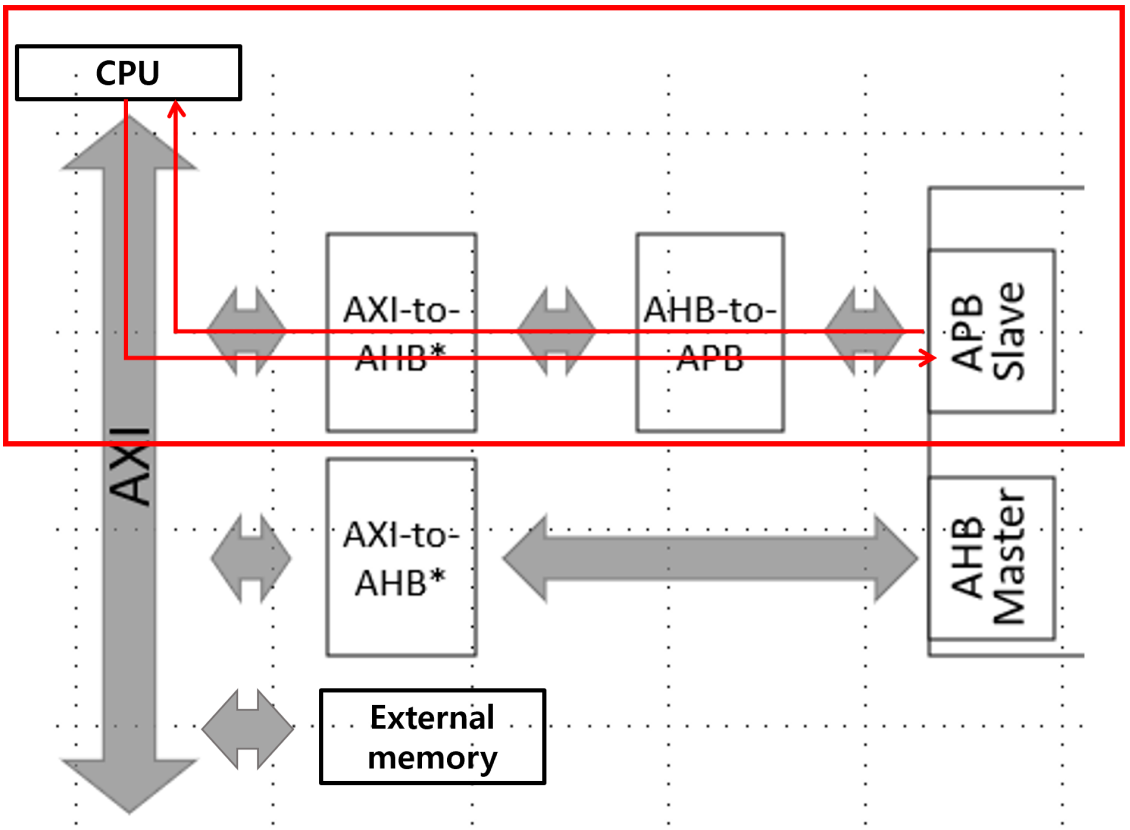
AHB to APB converter RTL 설계

- 목적



(figrue1) System Interconnect Architecture between AXI, AHB, and APB

AMBA Bus protocol의 AXI, AHB, APB간의 통신을 원활하게 하기 위하여 protocol간의 호환성을 보장하는 bridge가 필요하다.

AHB 기반의 아키텍처에서 APB slave를 추가할 때, AXI to AHB bridge를 통해 더 높은 성능을 유지하고, AHB to APB bridge를 사용하여 성능 저하를 최소화할 수 있다.

다음 글은 AHB to APB bridge를 설계하고, 이를 검증하기 위한 테스트 환경 구성 과정을 설명한다.

위 그림과 같이 testbench에서 검증 환경을 구성하고 behavioral simulation을 통해 waveform을 확인하며 검증을 진행하였다.

- **AHB to APB bridge 역할**:

  AHB master에서 APB slave로의 데이터 전송을 조정, AHB signal을 APB signal로 알맞게 변환

*Table 3-2 Transfer size encoding

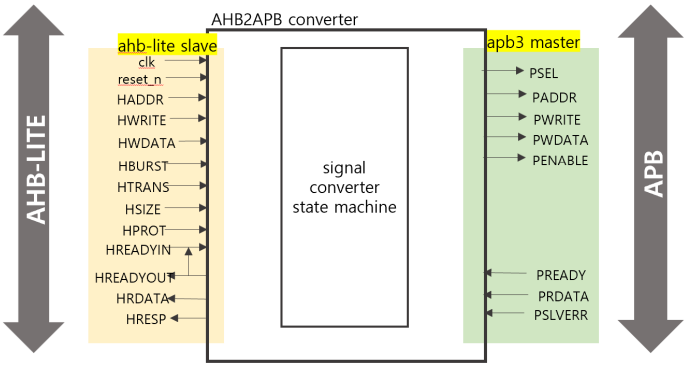
HSIZE[2]	HSIZE[1]	HSIZE[0]	Size (bits)	Description
0	0	0	8	Byte
0	0	1	16	Halfword
0	1	0	32	Word
0	1	1	64	Doubleword
1	0	0	128	4-word line
1	0	1	256	8-word line
1	1	0	512	-
1	1	1	1024	-

**Table 3-3 Burst signal encoding

HBURST[2:0]	Type	Description
b000	SINGLE	Single burst
b001	INCR	Incrementing burst of undefined length
b010	WRAP4	4-beat wrapping burst
b011	INCR4	4-beat incrementing burst
b100	WRAP8	8-beat wrapping burst
b101	INCR8	8-beat incrementing burst
b110	WRAP16	16-beat wrapping burst
b111	INCR16	16-beat incrementing burst

- **설계 조건**:
  - 1 master, 1 slave (1 : 1 transfer)
  - HSIZE = 3'b010 (32bit) 고정
- **테스트 시나리오:**
  - single transfer
  - burst transfer(INCR4, WRAP4 ...)
  - slave delay
  - slave error

AHB-LITE to APB converter block diagram / signal mapping

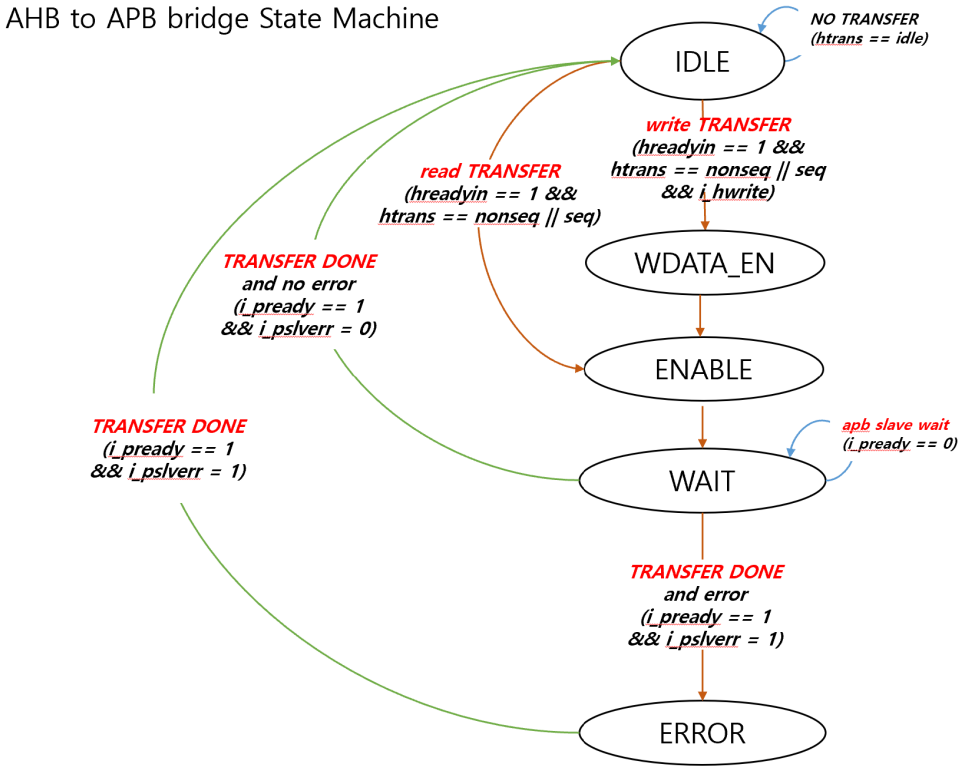


AHB to APB Signal Mapping

AHB-LITE		APB3
HADDR	→	PADDR
HWRITE	→	PWRITE
HWDATA	→	PWDATA
HREADYOUT	←	PREADY
HRDATA	←	PRDATA
HRESP	←	PSLVERR

다음 그림은 설계한 AHB-LITE to APB converter의 block diagram과 ahb와 apb signal 간의 매핑 표이다.

AHB-LITE to APB converter state machine



State description

IDLE:

- If ***htrans*** is ‘idle’, maintain the ‘IDLE state’.
- If ***htrans*** is ‘nonseq’ or ‘seq’, and ***hready*** is ‘1’, the transfer starts, initiating the address phase of AHB.
  During this, if it is a write transfer(i_hwrite==1), move to the ‘WDATA_EN state to wait for ***hwdata***.
  If it is a read transfer, generate the signals for the setup phase of APB, such as ***psel***, ***paddr***, and ***pwrite***,
  then move to the ‘ENABLE state’.

WDATA_EN:

- Generate the signals for the setup phase of APB, such as ***psel***, ***paddr***, and ***pwrite***, ***pwdata*** then move to the ‘ENABLE state’.

ENABLE:

- Activate the ***penable*** signal required for the APB access phase, then move to the ‘WAIT state’.

WAIT:

- If ***pready*** is ‘0’, the access phase of APB is still ongoing, so remain in the ‘WAIT state’.
- If ***pready*** is ‘1’:
  - If ***pslverr*** is ‘0’, the access phase of APB has successfully completed.
    Generate the ***hresp*** = 0 and ***hrdata***(if it's a read transfer) signals for AHB, then return to the ‘IDLE state’.
  - If ***pslverr*** is ‘1’, an error occurred during the APB access phase. Transition to the ‘ERROR state’
    (note: in the case of AHB, **handle the error over 2 clock cycles. On the first clock cycle, generate ***hresp*** = 1 and ***hreadyout*** = 0 signals).

** Table 5-2 Transfer response

	HREADY	
HRESP	0	1
0	Transfer pending	Successful transfer completed
1	ERROR response, first cycle	ERROR response, second cycle

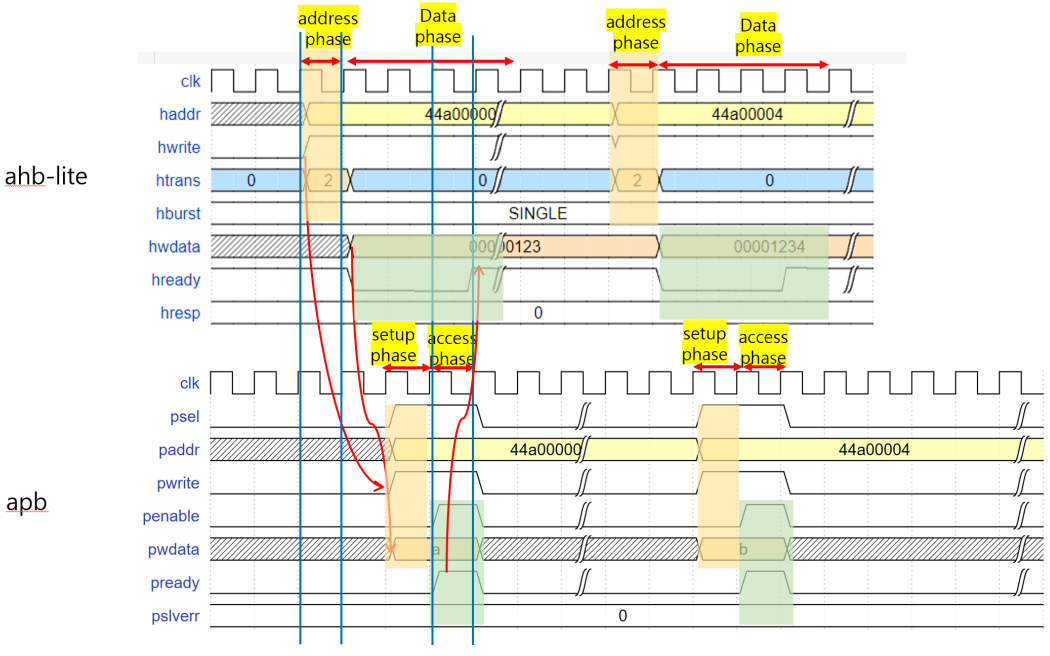
ERROR:

- Over 2 clock cycles, generate ***hresp*** = 1 and ***hreadyout*** = 1 signals to complete error handling. After this, return to the ‘IDLE state’.

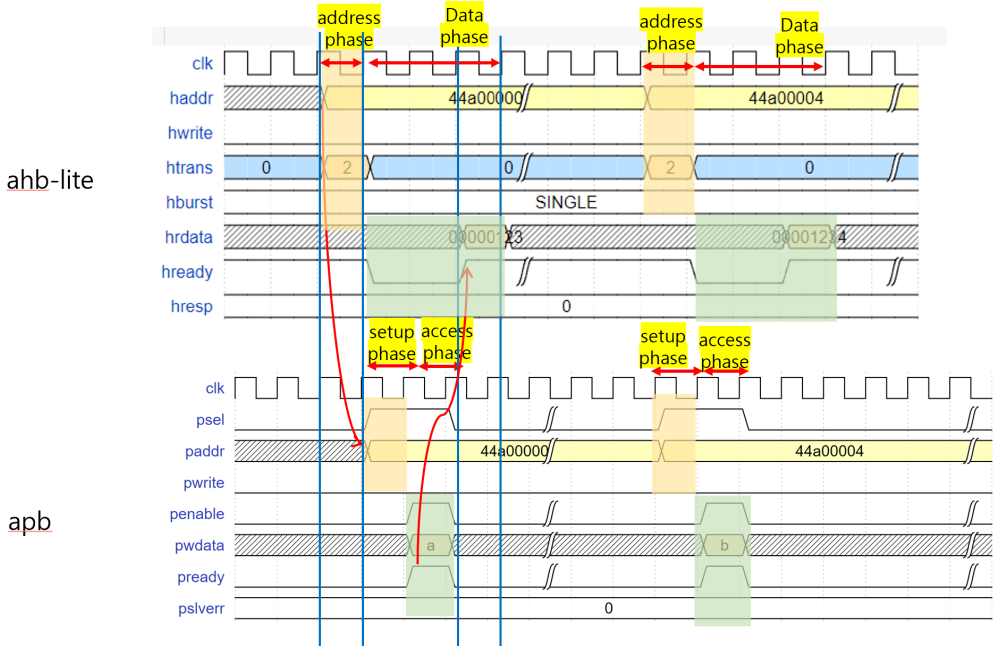
다음은 설계한 ahb to apb converter의 state machine과 state description이다.

AHB-LITE signal, APB3 signal 변환 예상

single transfer : 2 write transfer



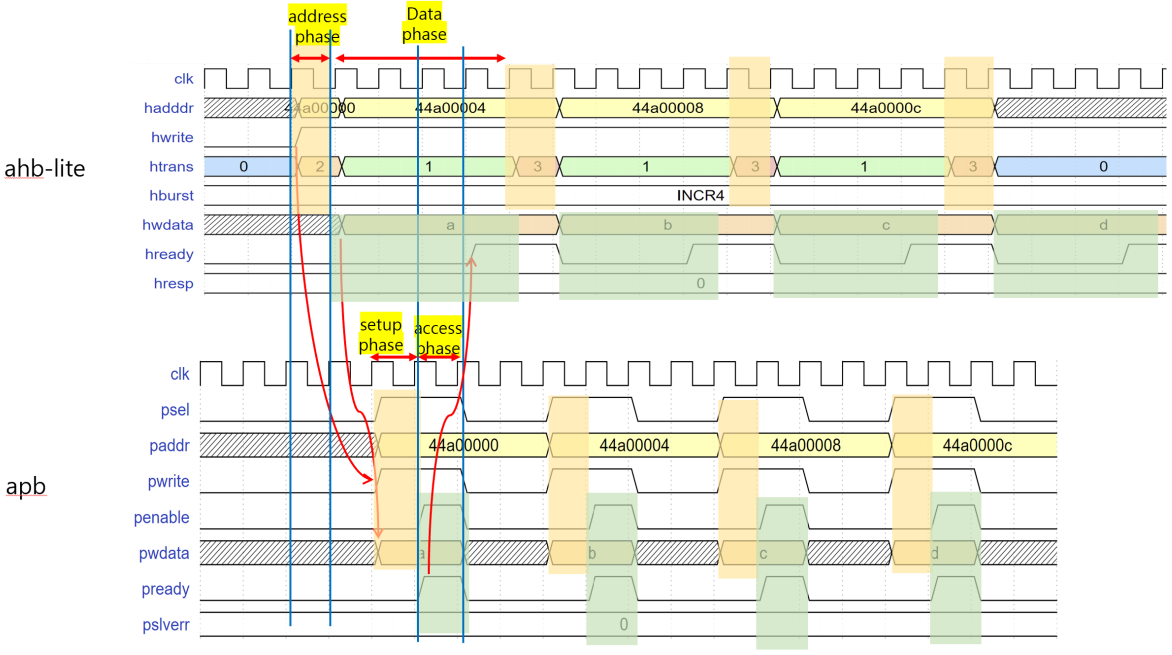
single transfer : 2 read transfer



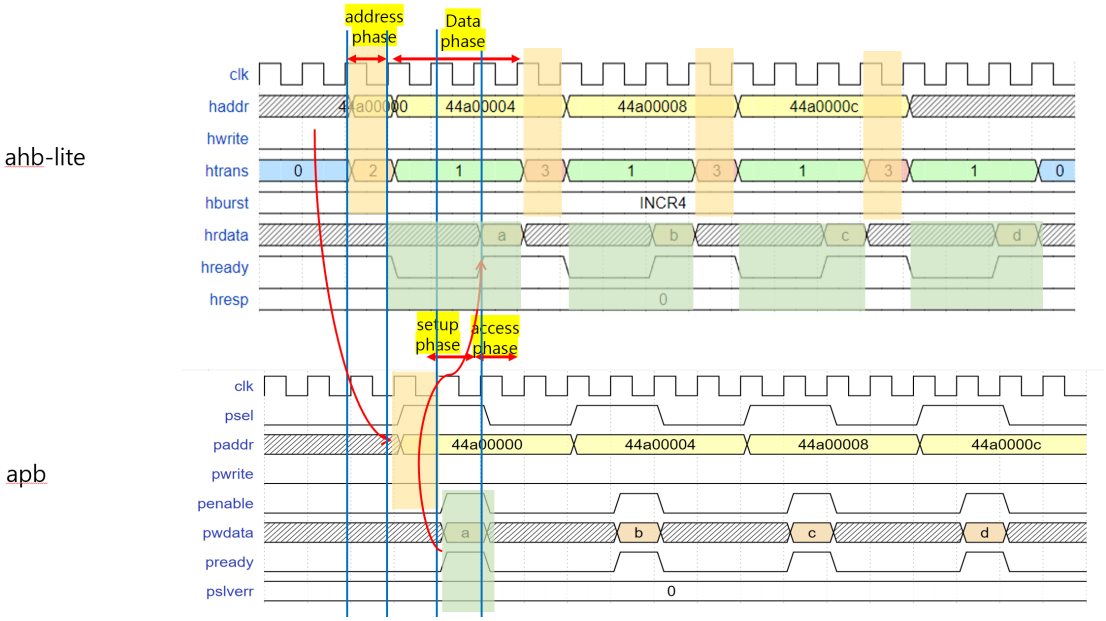
Signal Waveforms for Single Transfer in AHB-Lite and APB

위 그림은 single transfer에서 2 Write transfer / 2 Read transfer를 수행했을 경우 ahb-lite의 signal이 apb의 signal로 변환되는 과정을 예상한 그림이다.(no delay, no error)

Burst transfer : INCR4 write transfer



Burst transfer : INCR4 read transfer



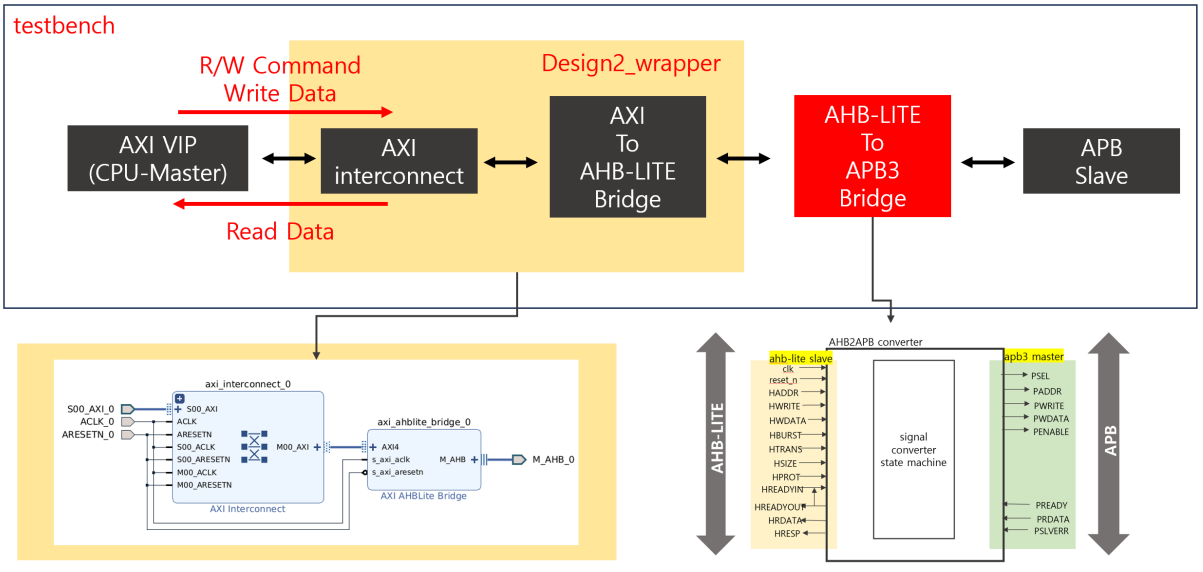
Signal Waveforms for Burst Transfer(INCR4) in AHB-Lite and APB

위 그림은 burst transfer에서 INCR4 수행

ahb-lite의 signal이 apb의 signal로 변환되는 과정을 예측한 그림이다.(no delay, no error)

behavioral simulation

AHB to APB bridge RTL 설계 및 검증 환경



Testbench(top) block diagram

검증 환경testbench에서 위 그림과 같이 검증 환경을 구성하였다.

‘AXI verification’, ‘AXI interconnect’, ‘AXI to AHB-LITE bridge’는 Xilinx ip를 사용하였으며, AXI VIP로 test bench에서 burst trasnfer를 생성하여 설계한 AHB to APB bridge, APB slave를 검증하였다

(Test 1)

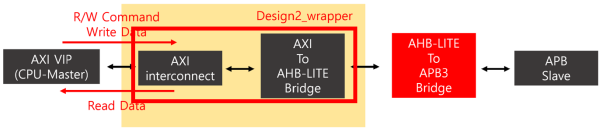
AXI LITE ↔ AHB LITE ↔ APB3

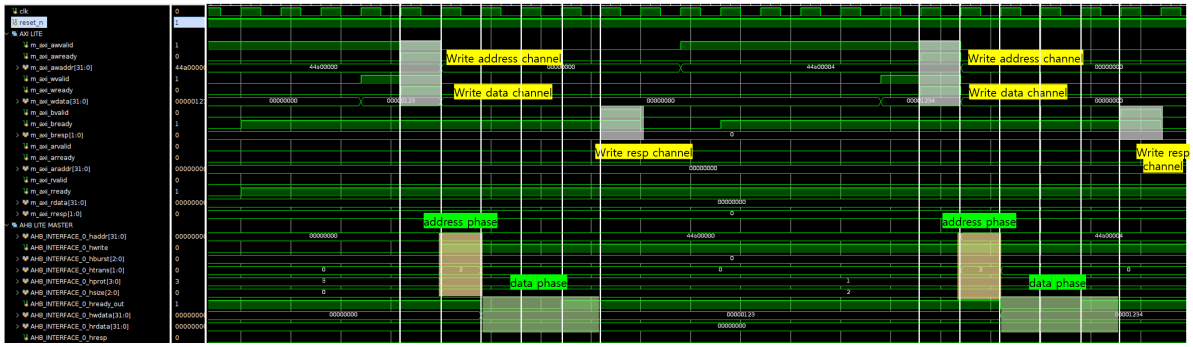
single transfer : 2 write transfer / 2 read transfer

```
slv_base_addr = 32'h44A0_0000;

vip_mst_agent = new("master vip agent", u_axi_lite_vip.inst.IF);
vip_mst_agent.start_master();
vip_mst_agent.AXI4LITE_WRITE_BURST(32'h44A0_0000,0,32'h0123,axi_resp);
vip_mst_agent.AXI4LITE_WRITE_BURST(32'h44A0_0004,0,32'h1234,axi_resp);
vip_mst_agent.AXI4LITE_READ_BURST(32'h44A0_0000,0,resp_data,axi_resp);
vip_mst_agent.AXI4LITE_READ_BURST(32'h44A0_0004,0,resp_data,axi_resp);
$stop;
```

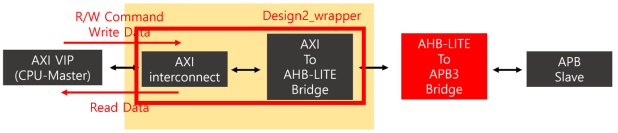
위와 같이 testbench에서 single transfer를 생성하여 검증하였다.

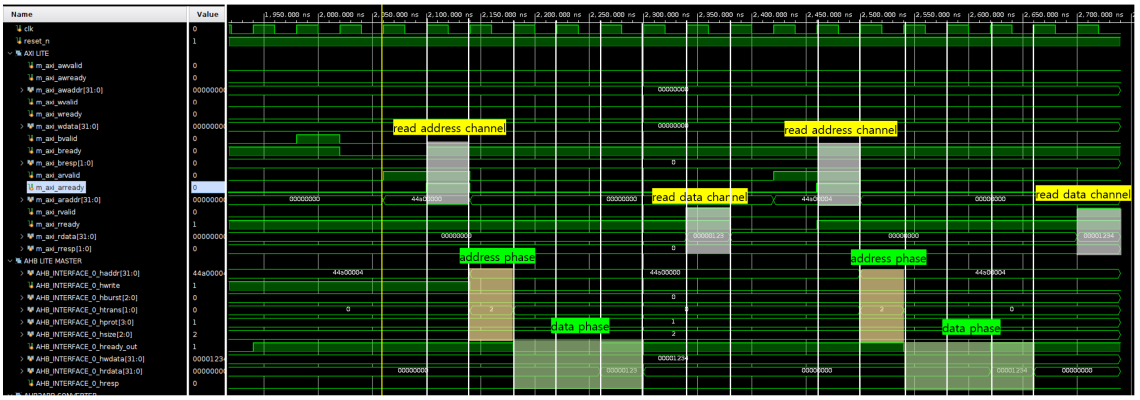
single transfer
2 write transfer
AXI LITE <-> AHB LITE





single transfer
2 read transfer
AXI LITE <-> AHB LITE

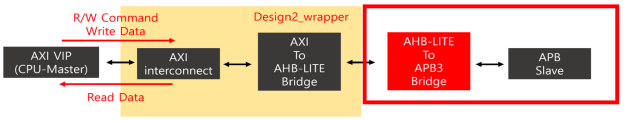


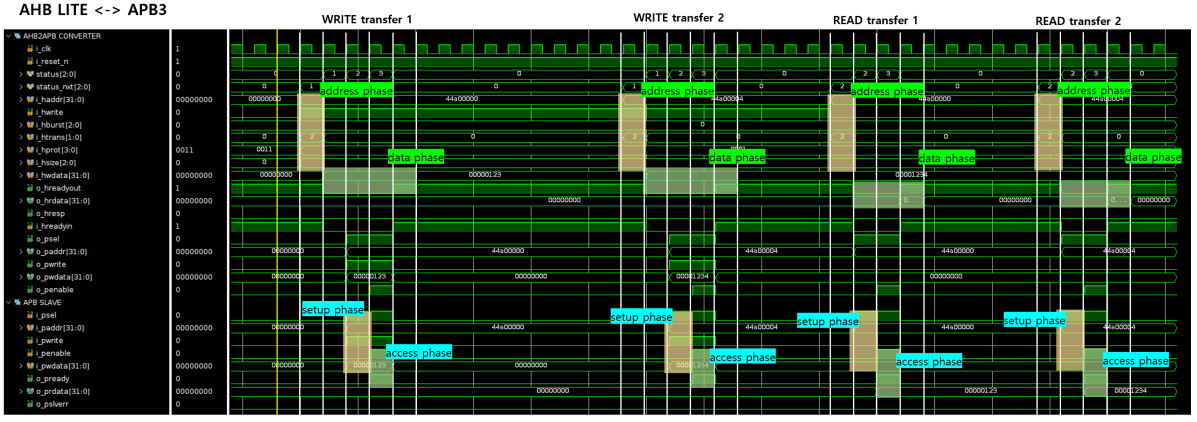


위 그림은 AXI LITE ↔ AHB LITE bridge의 signal 변환을 나타내는 wave form이다.

AXI LITE로 생성한 single transfer가 AHB LITE signal로 올바르게 변환되는 것을 확인하였다.

single transfer
2 write transfer / 2read transfer
AHB LITE <-> APB3





위 그림은 설계한 AHB LITE to APB converter와 APB slave의 입출력을 나타내는 waveform이다.

single transfer가 converter로 입력되어 APB signal로 올바르게 변환되는 것을 확인하였으며, 동시에 APB slave의 동작을 함께 검증하였다.

(Test 2)

AXI4 ↔ AHB LITE ↔ APB3

Burst transfer : WRAP4

slave error → o
(테스트를 위해 임의로 특정 address에 접근했을 때 slave에서 error signal을 발생하도록 설계)

slave delay → X

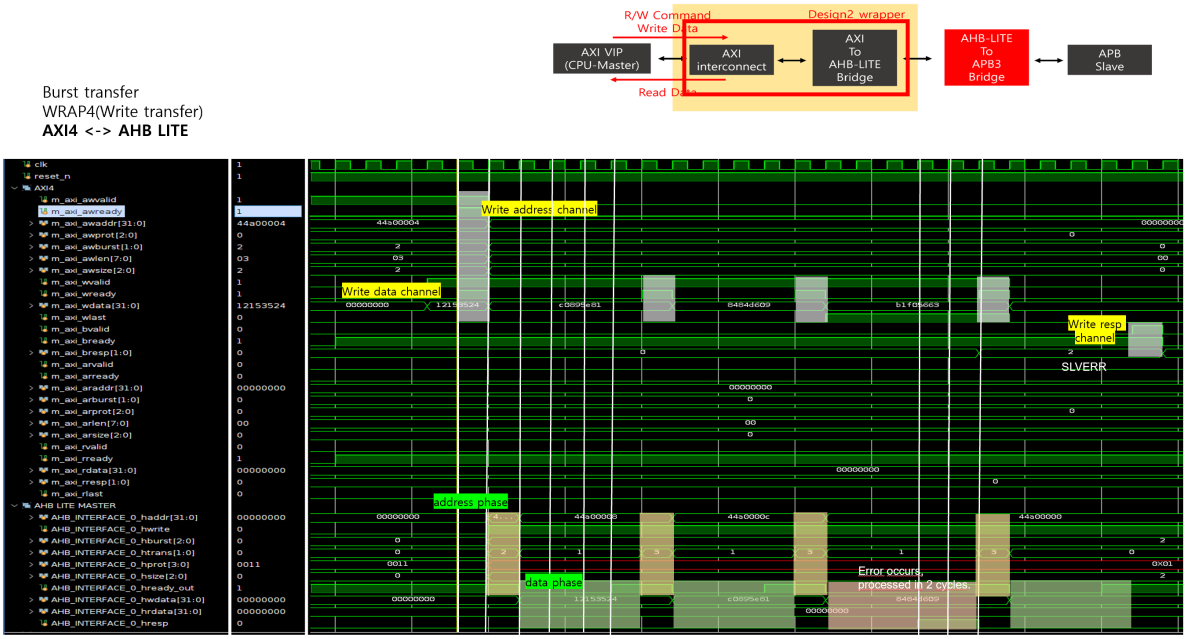
test bench
Generate burst transfer

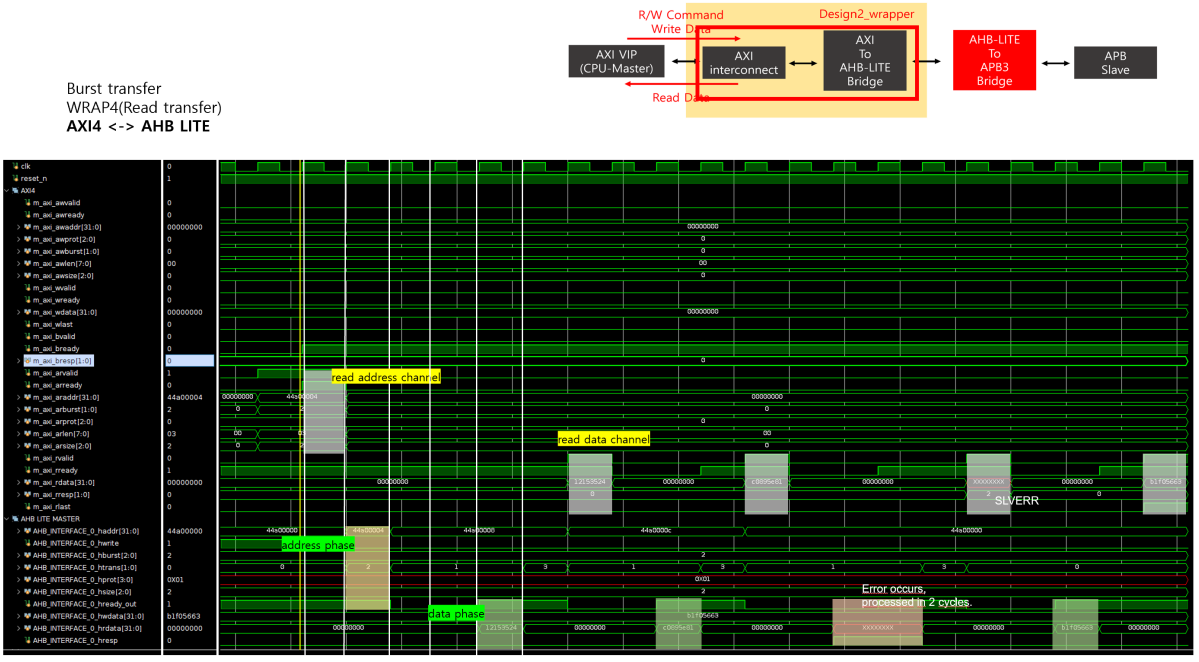
```
axi4_size   = XIL_AXI_SIZE_4BYTE;
axi4_burst  = XIL_AXI_BURST_TYPE_WRAP;
axi4_len    = 3;
axi4_lock   = XIL_AXI_ALOCK_NOLOCK;
axi4_cache  = 0;
axi4_prot   = 3'b000;
axi4_region = 0;
axi4_qos    = 0;
axi4_awuser = 0;
axi4_aruser = 0;
axi4_wuser  = 0;
axi4_ruser  = 0;

slv_base_addr = 32'h44A0_0004;

vip_mst_agent = new("master vip agent", u_axi_vip.inst.IF);

vip_mst_agent.start_master();

foreach (axi4_data[i]) begin
    axi4_data[i] = $random;
end

vip_mst_agent.AXI4_WRITE_BURST(
    0,              //ID
    slv_base_addr,  //ADDR
    axi4_len,       //Burst Length
    axi4_size,      //Data Size
    axi4_burst,     //Burst type
    axi4_lock,      //Lock
    axi4_cache,     //Cache
    axi4_prot,      //Prot
    axi4_region,    //Region
    axi4_qos,       //Qos
    axi4_awuser,    //AWUSER
    {axi4_data[3], axi4_data[2], axi4_data[1], axi4_data[0]},
    axi4_wuser,     //WUSER
    axi_bresp        //RESP
);

vip_mst_agent.AXI4_READ_BURST(
    0,              //ID
    slv_base_addr,  //ADDR
    axi4_len,       //Burst Length
    axi4_size,      //Data Size
    axi4_burst,     //Burst type
    axi4_lock,      //Lock
    axi4_cache,     //Cache
    axi4_prot,      //Prot
    axi4_region,    //Region
    axi4_qos,       //Qos
    axi4_awuser,    //AWUSER
    resp_data,      //WDATA
    axi_rresp,      //RESP
    axi4_ruser     //RUSER
);

$stop;
```

write burst transfer

read burst transfer

위와 같이 testbench에서 burst transfer인 wrap4 write / read transfer를 생성하여 검증하였다.

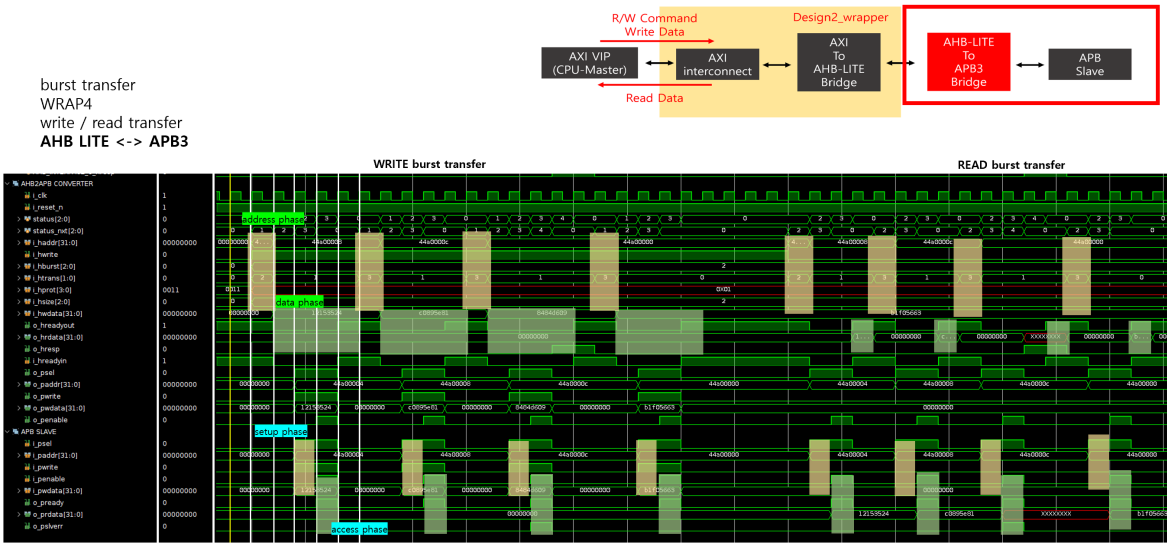




위 그림은 AXI4 ↔ AHB LITE bridge의 signal 변환을 나타내는 wave form이다.

AXI4로 생성한 burst transfer가 AHB LITE signal로 올바르게 변환되는 것을 확인하였다.

또한, slave error 발생 시 2 cycle의 hresp 응답을 확인하였다.



위 그림은 설계한 AHB LITE to APB converter와 APB slave의 입출력을 나타내는 waveform이다.

burst transfer가 converter로 입력되어 APB signal로 올바르게 변환되는 것을 확인하였다.

apb bus는 burst를 지원하지 않으므로, wrap4 일 경우, apb에서 하나의 transaction이 끝나기를 기다렸다가(pready) 다음 transaction을 진행하도록 설계하였다.

(Test 3)

AXI4 ↔ AHB LITE ↔ APB3

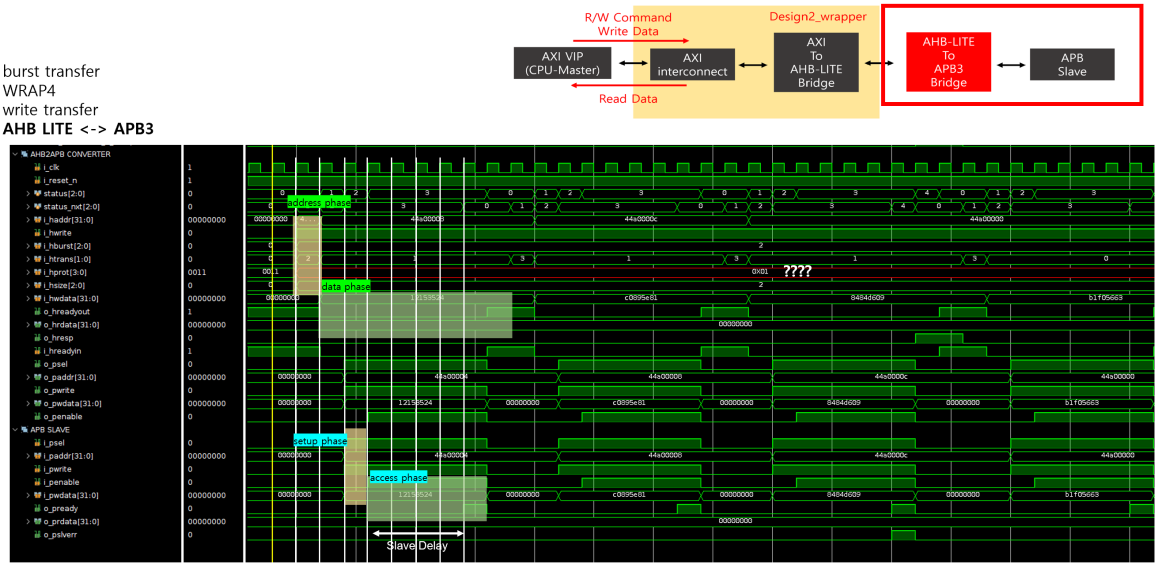
Burst transfer : WRAP4

slave error → o
(테스트를 위해 임의로 특정 address에 접근했을 때 slave에서 error signal을 발생하도록 설계)

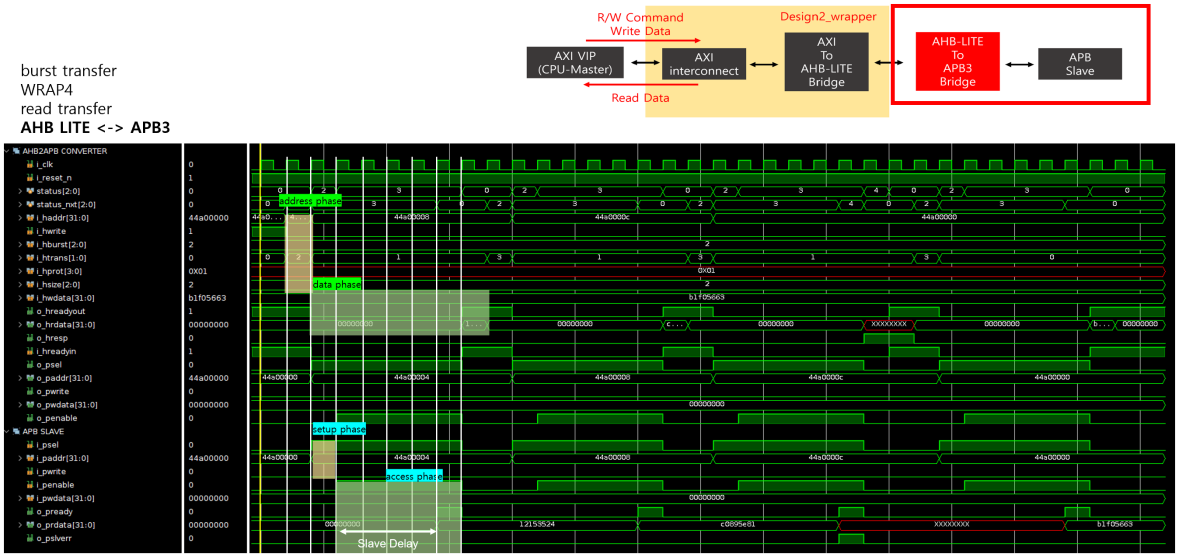
slave delay → o

(임의로 slave 처리 cycle을 추가하여 slave 지연으로 인해 ready 신호가 늦어지는 경우를 테스트)

burst transfer
WRAP4
write transfer
AHB LITE <-> APB3



burst transfer
WRAP4
read transfer
AHB LITE <-> APB3



동일하게 burst trasfer에서 slave delay가 발생했을 경우, 설계한 AHB LITE to APB converter의 입출력 waveform을 확인하였다.

****hprot가 0x01?

Table 3-1: **AXI4 Protection And Cache Support Translation to AHB-Lite Protection**

AXI4 Protection and Cache Support		AHB-Lite Protection Support	Description
s_axi_awcache [3:0]/ s_axi_ arcache[3:0]	s_axi_awprot[2:0]/ s_axi_arprot[2:0]	m_ahb_hprot[3:0]	
xxxx	xx1	0x1x	Privileged on AXI4 is translated to the equivalent privileged in AHB-Lite
xxxx	xx0	0x0x	Normal access on AXI4 is mapped to the equivalent user access on AHB
xxxx	0xx	0xx1	Data access on AXI4 is translated to the equivalent Data access in AHB-Lite
xxxx	1xx	0xx0	Instruction access on AXI4 is mapped to the equivalent Opcode fetch on AHB
00x1	xxx	01xx	Bufferable on AXI4 is translated to the equivalent Bufferable in AHB-Lite
00x0	xxx	00xx	Non-bufferable on AXI4 is translated to the equivalent Non-bufferable in AHB-Lite
xxxx	xxx	0xxx	There is no cacheable in AXI4. Always transferring Non-cacheable on AHB-Lite

axi4 awprot, aprot = 3'b000으로 입력 시 hprot = 4'b0x01

참조

AMBA_AHB-Lite_SPEC.pdf

ARM_AMBA3_APB.pdf

AXI4_specification.pdf

axi-vip-en-us-1.1 spec.pdf