AMBA AXI

2014 – 2015 – 2016 - 2017

Ando Ki, Ph.D.
(adki@future-ds.com)

Agenda

- Addressing options
- Data bus option
- Narrow transfer
- Byte invariance
- Unaligned transfers
- Fixed, incrementing and wrapping burst
- Wrapping bursts
- Responses
- Atomic access
- Atomic instructions
- Atomic lock accesses
- Atomic exclusive access
- Cache support
- Protection
- Write address channel
- Write data channel
- Write response channel
- All together for write
- Read address channel
- Read data channel
- All together for read
- Channel definition
- Channel dependency
- Handshake dependency
- Burst transfers
- Addressing options
- Transaction ordering
- ID scheme

Addressing options

Table 4-1 Burst length encoding

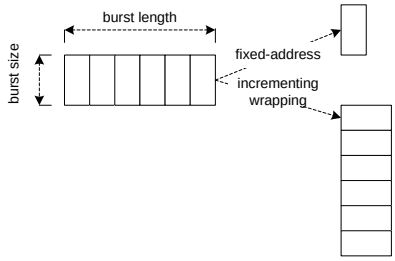
ARLEN[3:0] AWLEN[3:0]	Number of data transfers
b0000	1
b0001	2
b0010	3
. . .	
b1101	14
b1110	15
b1111	16

Table 4-2 Burst size encoding

ARSIZE[2:0] AWSIZE[2:0]	Bytes in transfer
b000	1
b001	2
b010	4
b011	8
b100	16
b101	32
b110	64
b111	128

Table 4-3 Burst type encoding

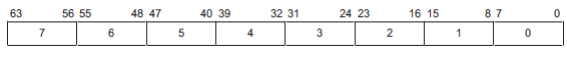
ARBURST[1:0] AWBURST[1:0]	Burst type	Description	Access
b00	FIXED	Fixed-address burst	FIFO-type
b01	INCR	Incrementing-address burst	Normal sequential memory
b10	WRAP	Incrementing-address burst that wraps to a lower address at the wrap boundary	Cache line
b11	Reserved	-	-



- Bursts must not cross 4KB boundaries to prevent them from crossing boundaries between slaves and to limit the size of the address incrementer required within slaves.
  - (note that AHB requires 1Kbyte boundary)
- Rules for wrapping burst
  - the start address must be aligned to the size of the transfer.
  - the length of the burst must be 2, 4, 8, or 16.

Data bus option

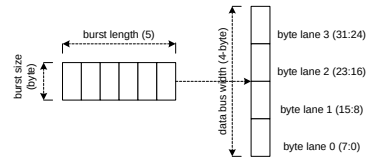
- The write strobe signals, **WSTRB**, enable sparse data transfer on the write data bus. Each write strobe signal corresponds to one byte of the write data bus. When asserted, a write strobe indicates that the corresponding byte lane of the data bus contains valid information to be updated in memory.
- There is one write strobe for each eight bits of the write data bus, so **WSTRB[n]** corresponds to **WDATA[(8 × n) + 7: (8 × n)]**.
- The AXI protocol enables a master to use the low-order address lines to signal an unaligned start address for a burst.
  - The information on the low-order address lines must be consistent with the information contained on the byte lane strobes.



Narrow transfer

- When a master generates a transfer that is narrower than its data bus, the address and control information determine which byte lanes the transfer uses. In incrementing or wrapping bursts, different byte lanes transfer the data on each beat of the burst. In a fixed burst, the address remains constant, and the byte lanes that can be used also remain constant.

- Example
  - the burst has five transfers
  - the starting address is 0
  - each transfer is eight bits
  - the transfers are on a 32-bit bus.

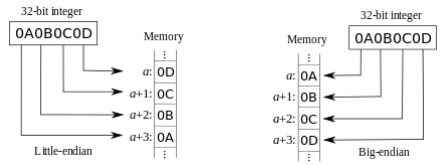


Byte lane used

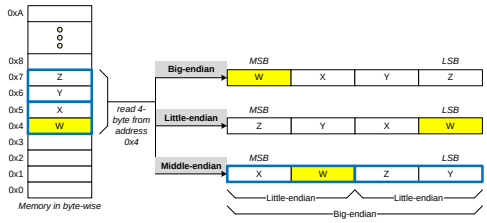
			DATA[7:0]	1st transfer
		DATA[15:8]		2nd transfer
	DATA[23:16]			3rd transfer
DATA[31:24]				4th transfer
			DATA[7:0]	5th transfer

Endianness and endian mapping

- Little–endian v.s. Big-endian



- Little/Big/Middle-endian



https://en.wikipedia.org/wiki/Endianness#Middle-endian

- Address invariance (byte invariance)
  - the address of bytes is always preserved between big and little.
    - the byte address is constant but byte significance is reversed when little- & big-endian are mixed

- Data invariance
  - the relative byte significance is preserved for datum of a particular size.

Invariance

- Address invariance (byte invariance)
  - When 4-byte data is read by big- or little-endian fashion.
    - The 4-byte are stored at the same address (invariant), but its significance varies depending on access size.

Addr Memroy

	(byte)	(LE)	(BE)
+3	dd	msb	lsb
+2	cc	...	..
+1	bb		
+0	aa	lsb	msb

Access Addr=+0	(LE)	(BE)	
Read 1 byte:	0xaa	0xaa	← address preserved for 0xaa
Read 2 byte:	0xbbaa	0xaabb	
Read 4 byte:	0xddcc_bbaa	0xaabb_ccdd	

Access Addr=+1	(LE)	(BE)	
Read 1 byte:	0xbb	0xbb	← address preserved for 0xbb

Access Addr=+2	(LE)	(BE)	
Read 1 byte:	0xcc	0xcc	← address preserved for 0xcc
Read 2 byte:	0xddcc	0xccdd	

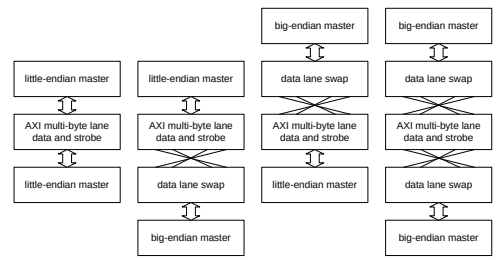
Access Addr=+3	(LE)	(BE)	
Read 1 byte:	0xdd	0xdd	← address preserved for 0xdd

- Data invariance
  - 32-bit data invariance (word invariance)
    - The datum of 32-bit word always the same value independent of endianness.
  - 16-bit data invariance (half-word invariance).
    - The datum of 16-bit word always the same value independent of endianness.
  - This scheme makes it possible to inter-mix big- and little-endian system without any treatment.
    - However, accesses should keep its access size.
      - E.g., 32-bit data invariance only guarantees data-invariance for 32-bit wide data access.

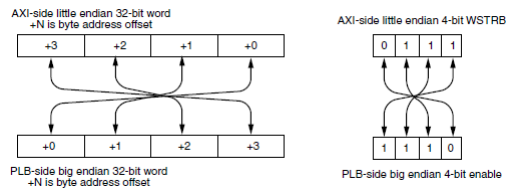
http://stackoverflow.com/questions/21449/types-of-endianness

Byte invariance (Address invariance)

- Byte-invariant endianness means that a byte transfer to a given address passes the eight bits of data on the same data bus wires to the same address location.
  - AXI uses nonjustified-bus with little-endian scheme
  - Most little-endian components can connect directly to a byte-invariant interface.
  - Components that support only big-endian transfers require a conversion function for byte-invariant operation.
    - The conversion function should provide byte-invariant (i.e., address invariant)

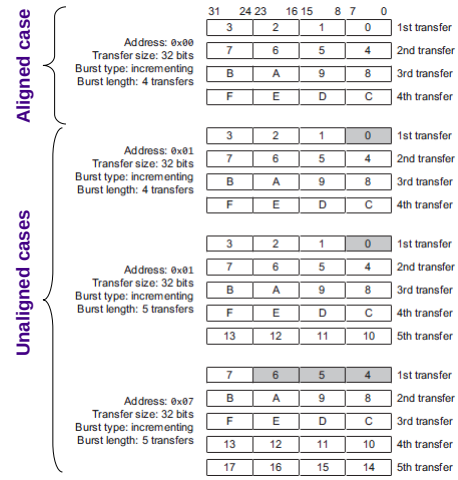


AXI to PLB component, where PLB uses big-endian



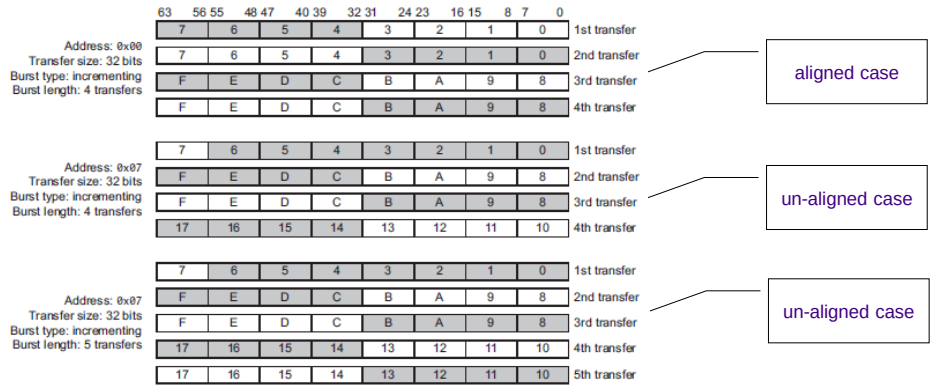
Unaligned transfers

- The AXI protocol uses burst-based addressing, which means that each transaction consists of a number of data transfers. Typically, each data transfer is aligned to the size of the transfer.
  - For example, a 32-bit wide transfer is usually aligned to four-byte boundaries. However, there are times when it is desirable to begin a burst at an unaligned address.
- The AXI protocol enables a master to use the low-order address lines to signal an unaligned start address for a burst. The information on the low-order address lines must be consistent with the information contained on the byte lane strobes.

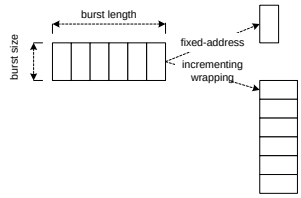


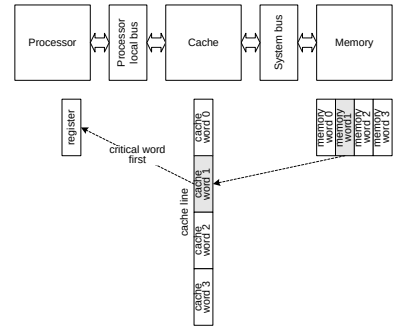
Unaligned transfers

- Aligned and un-aligned word (4-byte) transfer on a 64-bit bus.
  - (note shaded byte-lane is not used)



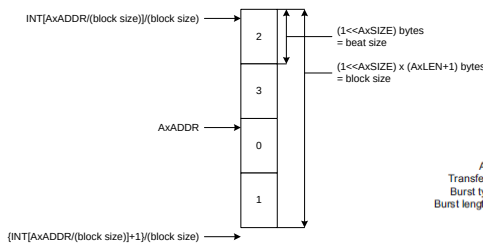
Fixed, incrementing and wrapping burst

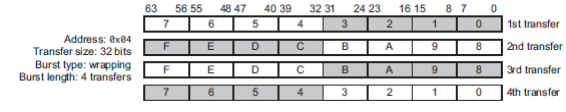




Wrapping burst

- Rules for wrapping burst
  - Bursts must not cross 4KB boundaries
  - the start address must be aligned to the size of the transfer.
  - the length of the burst must be 2, 4, 8, or 16.
- (beat size) can be equal to or smaller than data bus width
- Aligned wrapping word (4-byte) transfers on a 64-bit bus
  - (note shaded byte lane is not used)
  - AxADDR = 0x....4
  - AxSIZE = 3'b010 (4-byte)
  - AxLEN = 4'b011 (4 beats)
  - AxBURST = 2'b10 (wrapping)



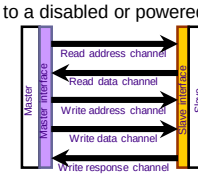


Responses

- BRESP[1:0] for write response of write transaction
  - a single response is signaled for the entire burst, and not for each data transfer within the
  - burst.
- RRESP[1:0] for read data of read transaction
  - Each transfer has its own response.

RRESP[1:0] BRESP[1:0]	Response	Meaning
b00	OKAY	Normal access okay indicates if a normal access has been successful. Can also indicate an exclusive access failure.
b01	EXOKAY	Exclusive access okay indicates that either the read or write portion of an exclusive access has been successful.
b10	SLVERR	Slave error is used when the access has reached the slave successfully, but the slave wishes to return an error condition to the originating master.
b11	DECERR	Decode error is generated typically by an interconnect component to indicate that there is no slave at the transaction address.

- The OKAY response indicates:
  - the success of a normal access
  - the failure of an exclusive access
  - an exclusive access to a slave that does not support exclusive access.
- The EXOKAY response indicates the success of an exclusive access
- The DECERR response
  - Interconnect responds for accesses to un-mapped locations.
- The SLVERR response includes
  - FIFO/buffer overrun or under-run condition
  - unsupported transfer size attempted
  - write access attempted to read-only location
  - timeout condition in the slave
  - access attempted to an address where no registers are present
  - access attempted to a disabled or powered-down function.



Atomic access

- Locked access (AXI 3 only)
  - AXI slave guarantees that there will be no accesses with different transaction ID between locked read and locked write from the same transaction ID.
- Exclusive access (AXI 3 and AXI 4)
  - AXI slave reports if there are any write accesses with different transaction ID between exclusive read and exclusive write from the same transaction ID.

Table 6-1 Atomic access encoding

ARLOCK[1:0] AWLOCK[1:0]	Access type
b00	Normal access
b01	Exclusive access
b10	Locked access
b11	Reserved

RRESP[1:0] BRESP[1:0]	Response	Meaning
b00	OKAY	Normal access okay indicates if a normal access has been successful. Can also indicate an exclusive access failure.
b01	EXOKAY	Exclusive access okay indicates that either the read or write portion of an exclusive access has been successful.
b10	SLVERR	Slave error is used when the access has reached the slave successfully, but the slave wishes to return an error condition to the originating master.
b11	DECERR	Decode error is generated typically by an interconnect component to indicate that there is no slave at the transaction address.

Exclusive ERROR

Exclusive OK

Atomics instructions

- Examples of atomic operations
  - test and set
  - atomic increment
  - atomic exchange register and memory location, i.e., swap
  - compare and swap
- Atomic increment case
  - simple approach

```
ldr r0, [r1]
add r0, r0, #1
str r0, [r1]
```

  - What happen when more than one try

```
atomic_inc:
    ldrex r0, [r1]
    add r0, r0, #1
    strex r2, r0, r1
    cmp r2, #0
    bne atomic_inc
```

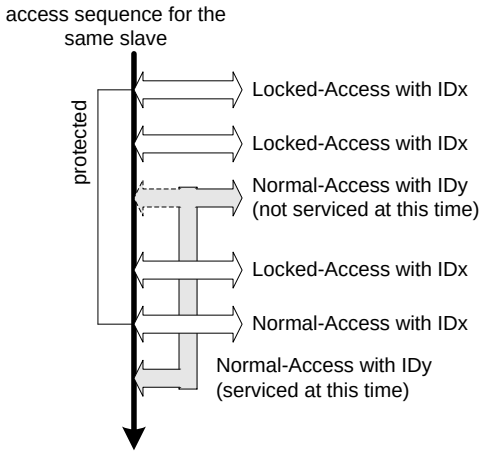
    - "strex r0, r1, [addr]": 'r0' will have '0' when succeeded

Programs	Shared programs
Higher-level API	Locks, Semaphoeres, Monitors, ..
Hardware	load/store, disable interrupt, test&set, Compare&swap

Atomics instructions

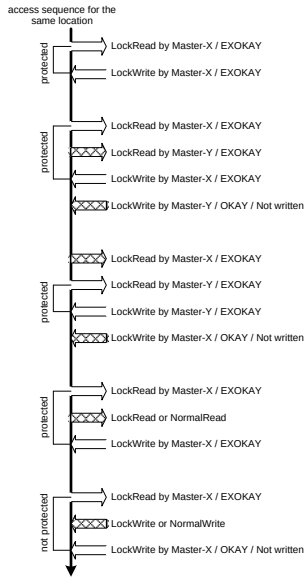
- ARM
  - SWP
  - LDREX & STREX
- Intel
- Sparc
- MIPS
- Motorola

Atomic lock accesses



- When a master starts a locked sequence of either read or write transactions it must ensure that it has no other outstanding transactions waiting to complete. Any transaction with ARLOCK[1:0] or AWLOCK[1:0] set to indicate a locked sequence forces the interconnect to lock the following transaction. Therefore, a locked sequence must always complete with a final transaction that does not have ARLOCK[1:0] or AWLOCK[1:0] set to indicate a locked access. This final transaction is included in the locked sequence and effectively removes the lock.

Atomic exclusive access



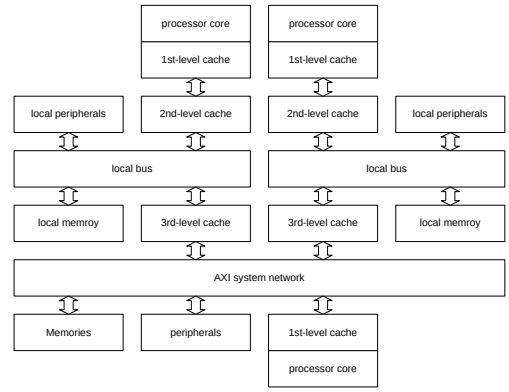
- The basic process for an exclusive access is:
  - 1. A master performs an exclusive read from an address location.
  - 2. At some later time, the master attempts to complete the exclusive operation by performing an exclusive write to the same address location.
  - 3. The exclusive write access of the master is signaled as:
    - • Successful if no other master has written to that location between the read and write accesses. (EXOKAY)
    - • Failed if another master has written to that location between the read and write accesses. In this case the address location is not updated. (OKAY)

Cache support

- These signals provide additional information about how the transaction can be processed.
  - B ([0], Bufferable), C ([1], Cacheable), RA ([2], Read Allocate), WA ([3], Write Allocate)

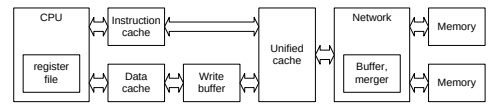
ARCACHE[3:0]
AWCACHE[3:0]

WA	RA	C	B	Transaction attributes
0	0	0	0	Noncacheable and nonbufferable
0	0	0	1	Bufferable only
0	0	1	0	Cacheable, but do not allocate
0	0	1	1	Cacheable and bufferable, but do not allocate
0	1	0	0	Reserved
0	1	0	1	Reserved
0	1	1	0	Cacheable write-through, allocate on reads only
0	1	1	1	Cacheable write-back, allocate on reads only
1	0	0	0	Reserved
1	0	0	1	Reserved
1	0	1	0	Cacheable write-through, allocate on writes only
1	0	1	1	Cacheable write-back, allocate on writes only
1	1	0	0	Reserved
1	1	0	1	Reserved
1	1	1	0	Cacheable write-through, allocate on both reads and writes
1	1	1	1	Cacheable write-back, allocate on both reads and writes



Cache support

- Bufferable
  - write delay can be an arbitrary one
- Cacheable → Modifiable (AXI4)
  - Read: prefetch or read cache is possible
  - Write: write merging is possible
- Read allocate
  - If read miss, fetch the data to cache.
- Write allocate
  - If write miss, fetch the data to cache, and then write to the cache (and through the memory)



Signal	AXI4 definition	Description
ARCACHE[3]	Other Allocate	When asserted HIGH, the transaction must be looked up in a cache because it could have been allocated in the cache by another transaction, either a write transaction or a transaction from another master. The transaction must also be looked up in a cache if **ARCACHE**[2] is asserted HIGH. When deasserted LOW, if **ARCACHE**[2] is also deasserted LOW, then the transaction does not need to be looked up in a cache.
ARCACHE[2]	Allocate	When asserted HIGH, the transaction must be looked up in a cache because it could have been allocated. The transaction must also be looked up in a cache if **ARCACHE**[3] is asserted HIGH. When deasserted LOW, if **ARCACHE**[3] is also deasserted LOW, then the transaction does not need to be looked up in a cache. When asserted HIGH, it is recommended that this transaction is allocated in the cache for performance reasons.
ARCACHE[1]	Modifiable	When asserted HIGH, the characteristics of the transaction can be modified and a larger quantity of read data can be fetched than is required. When deasserted LOW the characteristics of the transaction must not be modified.
ARCACHE[0]	Bufferable	When **ARCACHE**[3:1] = b000, this bit has no effect. When **ARCACHE**[3:1] = b001, if this bit is deasserted LOW, the read data must be obtained from the final destination. If this bit is asserted HIGH, the read data can be obtained from the final destination or from a write that is progressing to the final destination. When either **ARCACHE**[3] is asserted HIGH, or **ARCACHE**[2] is asserted HIGH, this bit can be used to distinguish between Write Through and Write Back memory types.

Signal	AXI4 definition	Description
AWCACHE[3]	Allocate	When asserted HIGH, the transaction must be looked up in a cache because it could have been previously allocated. The transaction must also be looked up in a cache if **AWCACHE**[2] is asserted HIGH. When deasserted LOW, if **AWCACHE**[2] is also deasserted LOW, then the transaction does not need to be looked up in a cache and the transaction must propagate to the final destination. When asserted HIGH, it is recommended that this transaction is allocated in the cache for performance reasons.
AWCACHE[2]	Other Allocate	When asserted HIGH, the transaction must be looked up in a cache because it could have been previously allocated in the cache by another transaction, either a read transaction or a transaction from another master. The transaction must also be looked up in a cache if **AWCACHE**[3] is asserted HIGH. When deasserted LOW, if **AWCACHE**[3] is also deasserted LOW, then the transaction does not need to be looked up in a cache and the transaction must propagate to the final destination.
AWCACHE[1]	Modifiable	When asserted HIGH, the characteristics of the transaction can be modified and writes can be merged. When deasserted LOW, the characteristics of the transaction must not be modified.
AWCACHE[0]	Bufferable	When deasserted LOW, if both of **AWCACHE**[3:2] are deasserted LOW, the write response must be given from the final destination. When asserted HIGH, if both of **AWCACHE**[3:2] are deasserted LOW, the write response can be given from an intermediate point, but the write transaction is required to be made visible at the final destination in a timely manner. When deasserted LOW, if either of **AWCACHE**[3:2] is asserted HIGH, the write response can be given from an intermediate point, but the write transaction is required to be made visible at the final destination in a timely manner. When asserted HIGH, if either of **AWCACHE**[3:2] is asserted HIGH, the write response can be given from an intermediate point. The write transaction is not required to be made visible at the final destination.

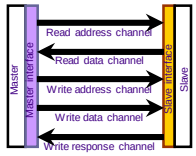
Protection

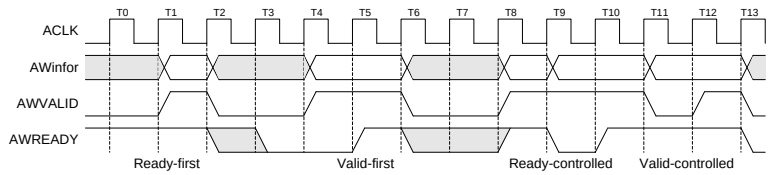
Table 5-2 Protection encoding

ARPROT[2:0] AWPROT[2:0]	Protection level
[0]	1 = privileged access 0 = normal access
[1]	1 = nonsecure access 0 = secure access
[2]	1 = instruction access 0 = data access

- To support complex system designs, it is often necessary for both the interconnect and other devices in the system to provide protection against illegal transactions. The **AWPROT** or **ARPROT** signal gives three levels of access protection:

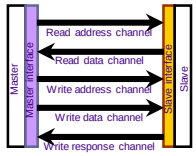
Write address channel

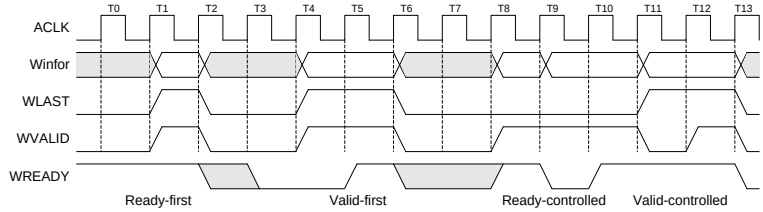




- The master can assert the AWVALID only when it drives valid information (i.e., address and control).
- The AWVALID and AWinfor must remain asserted until the salve accepts these (i.e., AWREADY is asserted).
- The AWREADY can be either high or low by default.
  - But, high is recommended in order to reduce transfer latency.
- Do not wait for AWREADY before asserting AWVALID in order to prevent a deadlock case.

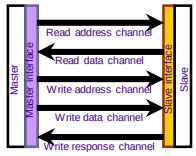
Write data channel

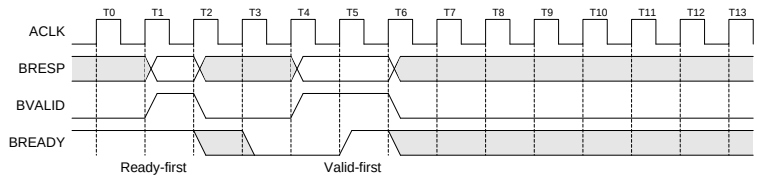




- The master can assert the WVALID only when it drives valid information (i.e., data and control).
- The WVALID and Winfor must remain asserted until the salve accepts these (i.e., WREADY is asserted).
- The master must assert WLAST when it drives the final write transfer in the burst.
- The WREADY can be either high or low by default.
  - But, high is recommended in order to reduce transfer latency.
- Do not wait for WREADY before asserting WVALID in order to prevent a deadlock case.

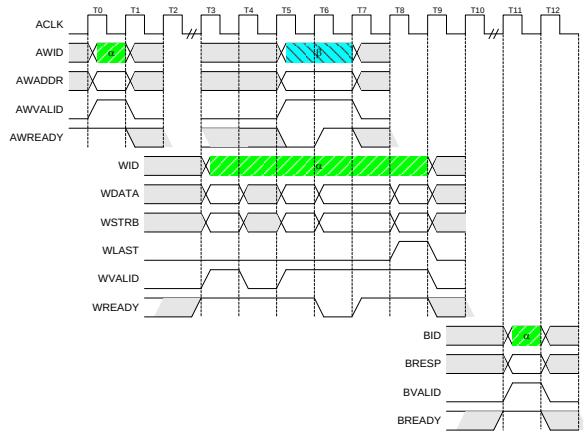
Write response channel



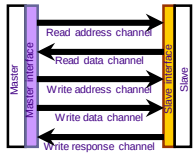


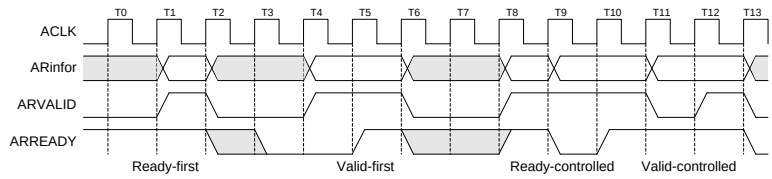
- The salve can assert the BVALID only when it drives a valid write response.
- The BVALID must remain asserted until the master accepts the write response (i.e., BREADY is asserted).
- The BREADY can be either high or low by default.
  - But, high is recommended in order to reduce transfer latency.
- Do not wait for BREADY before asserting BVALID in order to prevent a deadlock case.

All together for write



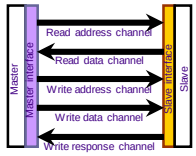
Read address channel

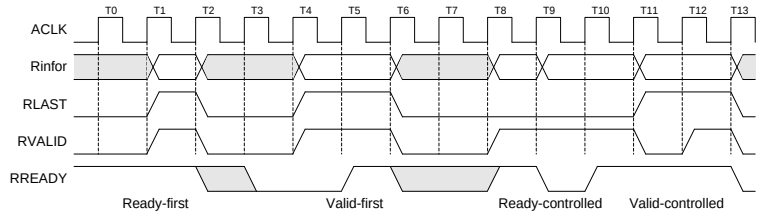




- The master can assert the ARVALID only when it drives valid information (i.e., address and control).
- The ARVALID and ARinfor must remain asserted until the salve accepts these (i.e., ARREADY is asserted).
- The ARREADY can be either high or low by default.
  - But, high is recommended in order to reduce transfer latency.
- Do not wait for ARREADY before asserting ARVALID in order to prevent a deadlock case.

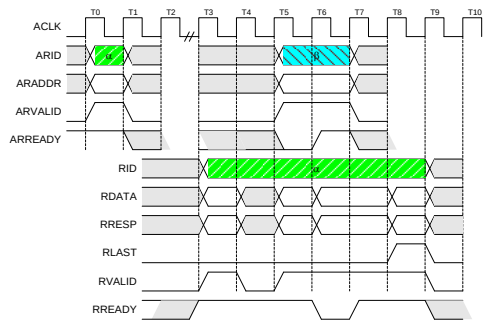
Read data channel





- The slave can assert the RVALID only when it drives valid information (i.e., data and control).
- The RVALID and Rinfor must remain asserted until the master accepts these (i.e., RREADY is asserted).
- The slave must assert RLAST when it drives the final write transfer in the burst.
- The RREADY can be either high or low by default.
  - But, high is recommended in order to reduce transfer latency.
- Do not wait for RREADY before asserting RVALID in order to prevent a deadlock case.

All together for read

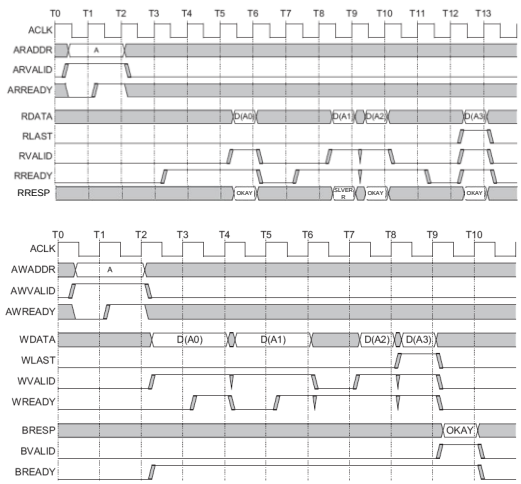


Channel definition

- Read and write address channel
  - variable-length bursts, 1 to 16 data transfers pre burst
  - bursts with a transfer size of 8-1024 bits (1 to 128 bytes)
  - wrapping, incrementing, and non-incrementing bursts
  - atomic operations, using exclusive or locked accesses
  - system-level caching and buffering control
  - secure and privileged access
- Read data channel
  - data bus, can be 8, 16, 32, 64, 128, 256, 512, 1024 bits wide
  - a read response indicating the completion status of the read transaction
- Write data channel
  - the data bus, can be 8, 16, 32, 64, 128, 256, 512, 1024 bits wide
  - one byte lane strobe for every eight data bits
  - always treated as buffered, so that the master can perform write transactions without slave acknowledgement of previous write transactions
- Write response channel
  - all write transactions use completion signaling
  - The completing signal occurs once for each burst, not for each individual data transfer within the burst

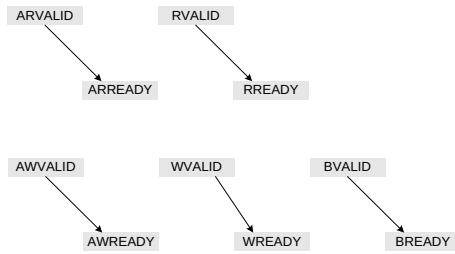
Channel dependency

- The slave should start read data sequence after the read address sequence.
  - 슬레이브는 읽기 주소를 받은 후에 읽은 데이터를 전송한다. (당연, 주소 없이 읽기는 불가능)
  - Read data must always follows the address to which the data related.
- The master should start write address sequence before write data sequence.
  - 마스터는 쓰기 데이터를 보내기에 앞서 쓰기 주소를 보낸다. 단, 슬레이브 입장에서는 쓰기 데이터가 쓰기 주소보다 먼저 도착할 수도 있다.
  - However, the write data can appear at the interface before the write address that related to it due to register stages of the write address.
- The slave should start write response sequence after write data sequence.
  - 슬레이브는 연속 쓰기에 대해 한번만 응답하는데, 이때 응답은 데이터 전송을 모두 받은 후 한다. (AXI3)
  - A write response must always follow the last write transfer in the write transaction to which the write response related.
  - In addition, the AXI4 protocol requires that the write response for all transactions must not be given until the clock cycle after address acceptance. (슬레이브 입장에서 쓰기 주소가 쓰기 데이터 보다 나중에 오기도 하므로)



Handshake dependency

- In order to avoid deadlock
  - The VALID signal must not be dependent on the READY signals
    - The VALID signal must not wait for any READY signal before driving it.
  - The READY signal can wait for assertion of the VALID signal.

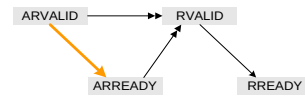


- Guess what will happen.
  - Master waits for READY before driving VALID for something.
  - While, slave also waits for VALID before driving READY for something.

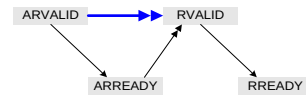
•The single-headed arrow points to signal that can be asserted before or after the previous signal is asserted. (A→B: B를 구동할 수 있다, A를 기다리지 않고)

Handshake dependency

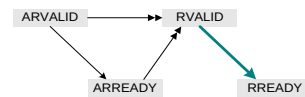
- The slave can wait for ARVALID to be asserted before it asserts ARREADY.
  - Meaning that ARREADY can be driven prior to ARVALID.



- The slave must wait for ARVALID and ARREADY to be asserted before it starts to return read data by asserting RVALID.



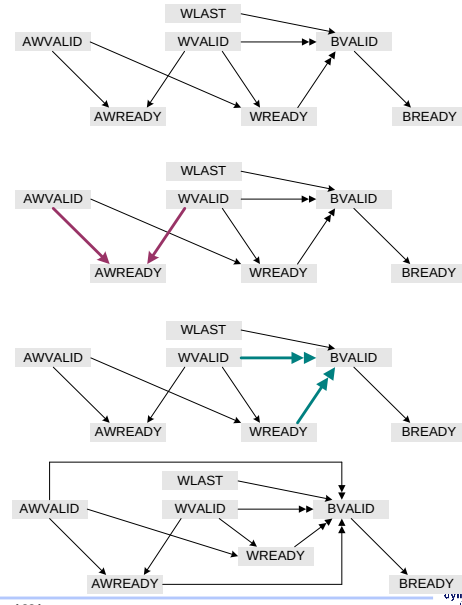
- The master can wait for RVALID to be asserted before it asserts RREADY.
  - Meaning that RREADY can be driven prior to RVALID.
  - The VALID signal must not wait for any READY signal before driving it.



- The single-headed arrow points to signal that can be asserted before or after the previous signal is asserted. (A→B: B를 구동할 수 있다, A를 기다리지 않고)
- The double-headed arrow points to signal that must be asserted only after assertion of the previous signal. (A=>>B: B를 구동하기에 앞서 A를 기다려야 한다.

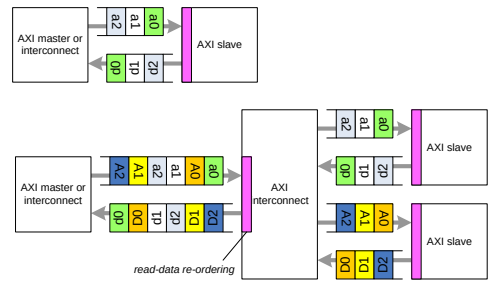
Handshake dependency

- The master must not wait for the slave to assert AWREADY or WREADY before asserting AWVALID or WVALID.
- The slave can wait for AWVALID or WVALID, or both, before AWREADY.
- The slave can wait for AWVALID or WVALID, or both, before asserting WREADY.
- The slave must wait for both WVALID and WREADY to be asserted before asserting BVALID.
- In addition, the AXI4 protocol requires that the write response for all transactions must not be given until the clock cycle after address acceptance.



Ordering

- Write request and data: The write data can appear at an interface before the write address that relates to it.
  - 쓰기의 경우, 데이터가 주소보다 먼저 도착할 수 있다.
- Read data must always follow the address to which the data relates.
  - 읽기 데이터는 반드시 해당 데이터의 주소 뒤에 온다.
- A write response must always follow the last write transfer in the write transaction to which the write response relates.
  - 쓰기 응답은 마지막 쓰기 전송 후에 온다.
- The data for a sequence of read transaction with the same ARID value must be returned in order (같은 ARID를 갖는 읽기 데이터의 경우)
  - 같은 슬레이브에서 여러 읽기 데이터가 공급될 경우, 슬레이브는 읽기 주소를 받은 순서대로 읽기 데이터를 공급한다.
  - 다른 슬레리브에서 오는 읽기 데이터는, 인터콘넥터에서 마스터에서 전달된 순서대로 공급해야 한다.

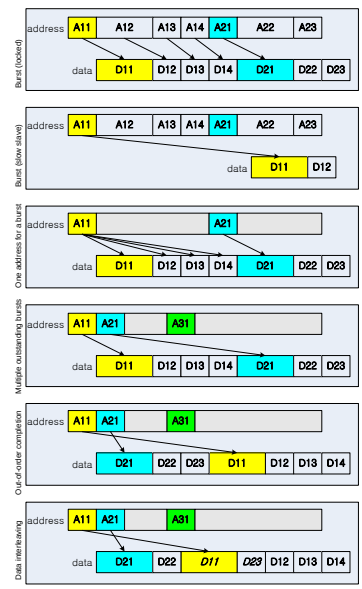


Ordering

- Write data with different AWIDs follow their address order.
- Responses to multiple writes with different IDs can be out-of-order from address order.
- Write interleaving
  - Interleaving rule
    - Data with different ID can be interleaved.
    - The order within a single burst is maintained
    - The order of first data needs to be the same with that of request
  - AXI4 does not support data interleaving
    - As a result, WID is not used.
    - But, all data-for-write should follow the same order of address arriving.

Burst transfers

- AHB burst (locked)
  - Address and data are locked together
  - Single pipeline stage
  - HREADY controls intervals of address and data
- AHB burst (slow slave)
  - If one slave is very slow, all data is held up.
- AXI one address for burst
  - One Address for entire burst
- AXI multiple outstanding bursts
  - One Address for entire burst
  - Allows multiple outstanding addresses
- AXI out-of-order completion
  - Each transaction has an ID attached
  - Transactions with the same ID must be ordered
  - Requires bus-level monitoring to ensure correct ordering on each ID
  - Masters can issue multiple ordered addresses
  - Fast slaves may return data ahead of slow slaves
  - Complex slaves may return data out of order
- AXI data interleaving
  - Returned data can even be interleaved
  - Gives maximum use of data bus
  - Data within a burst is always in order

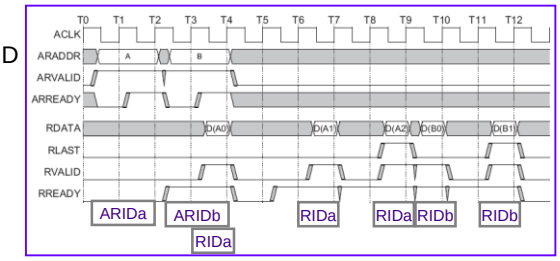


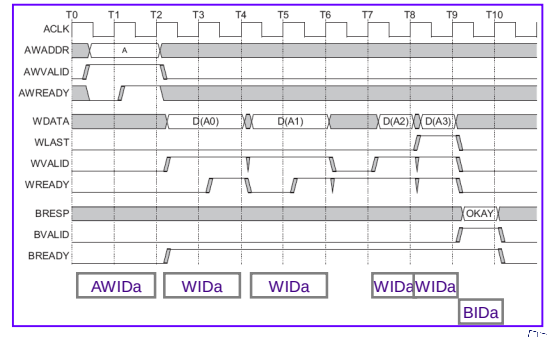
Addressing options

- AXI is burst-based, i.e., all transaction can be seen as a burst that consists of a number of transfers.
- The address of a burst is the address of the first byte in the transfer.
  - The slave should calculate the address of subsequent transfers in the burst.
- The burst address must not cross 4Kbyte boundary.
  - not allowed crossing boundaries between slaves
    - A burst is only destined for a single slave.
- No early termination allowed
  - Master can disable further writes by de-asserting all the strobes of the remaining transfers.
  - Master can discard further reads, but the remaining transfers should be completed.
    - Be careful to discard a read-sensitive device such as a FIFO.
- Additional limitations for AXI4
  - Burst longer than 16 are only supported for the INCR burst type.
    - WRAP & FIXED burst types can be up to 16 burst length.
  - Exclusive access are not permitted to use a burst length greater than 16.

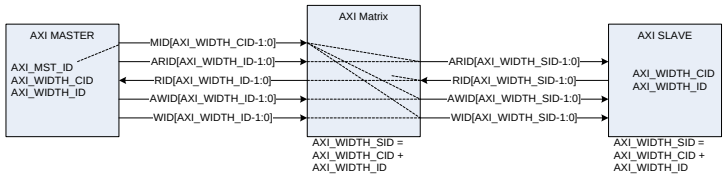
Transaction ordering

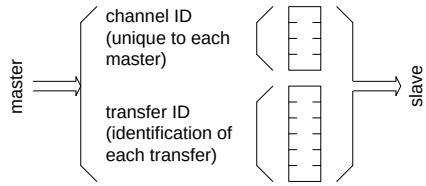
- For each channel, each information is tagged with transaction identification (ID tag) in order to find corresponding information.
  - Write transaction: AWID[n:0], WID[n:0], BID[n:0]
  - Read transaction: ARID[n:0], RID[n:0]
  - Where n is implementation-specific
- Multi-master system should use additional ID tag to ensure that ID from all masters are unique.
- Multiple virtual masters can be possible by adopting sub-field tag ID.
  - A port of master interface can act as a multiple master.





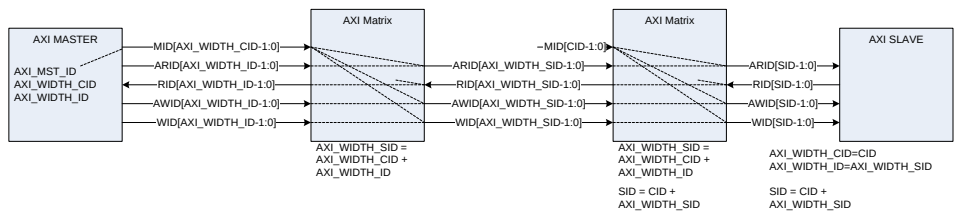
ID scheme (1/3)

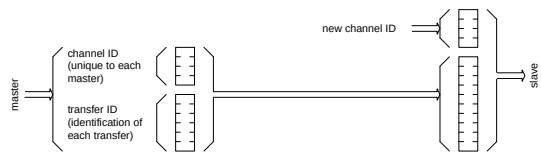




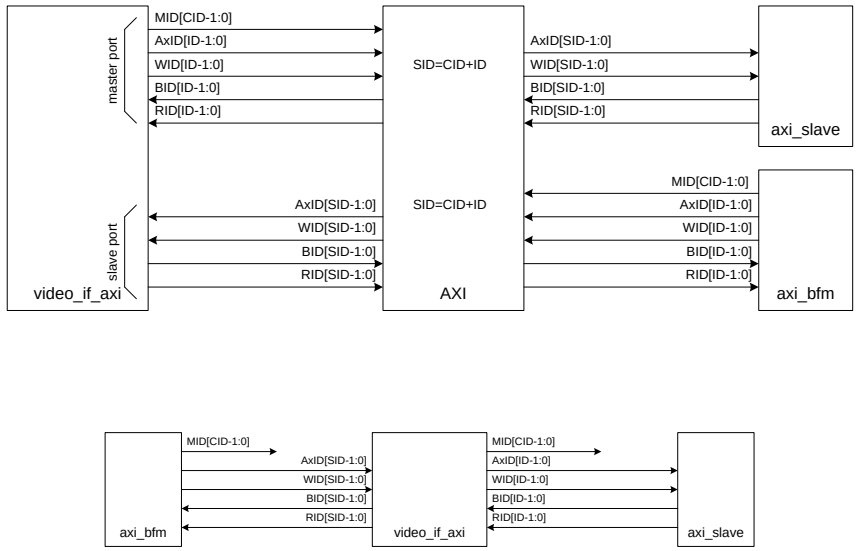
- ID (AxID, WID, BID, RID) is used to determine return path.
  - channel ID identifies which master
  - transfer ID identifies which transfer

ID scheme (2/3)





ID scheme (2/3)



References

- AMBA® AXI Protocol v1.0 Specification, IHI 0022B, ARM Limited, 2004.
- AMBA® AXI Protocol Version: 2.0 Specification, IHI 0022C (ID030510), ARM Limited, 2010.
- AMBA® AXI and ACE Protocol Specification, IHI 0022D (ID102711), ARM Limited, 2011. (AXI3, AXI3, and AXI4-Lite, ACE and ACE-Lite)
- AMBA® 4 AXI4-Stream Protocol Version: 1.0 Specification, IHI 0051A (ID030510), ARM Limited, 2010.