# AMBA AXI Infrastructure

Version 0 Revision 2

July 18, 2018 (March 1, 2016)

Ando Ki, Ph.D.
andoki@gmail.com / adki@future-ds.com

## Copyright notice



## License notice



## Abstract

This document addresses architecture of AMBA AXI.

**Table of Contents**

# 1 AMBA Buses

AMBA 1.0 consisting of ASB (Advance System Bus) and APB (Advanced Peripheral Bus) was introduced in 1995. AMBA 2.0 consisting of ASB, APB, and AHB (AMBA High-Performance Bus) was introduced in 1999.

In 2003, AXI (Advance eXtensible Interface) was introduced as part of AMBA 3.0 and it is also called AXI3. In 2009, AXI4 was introduced, where new features are added on AXI3 and some features are deprecated from AXI3.

Along with AHB and AXI, APB is also evolved. AMBA 2 APB called APB2 is fairly the same as the first APB defined in AMBA 1.0. AMBA 3 APB called APB3 has wait and error reporting features. AMBA 4 APB called APB4 has protection and partial access features.

**Table 1: AMBA Buses**

| AXI4 | AXI3 | AHB |
|---|---|---|
| • an extension of AMBA 3 AXI<br>• Burst length up to 256<br>• Quality-of-service<br>• Removal of lock transaction<br>• Removal of write interleaving | • channel architecture<br>• registers slices<br>• one address for burst up to 16<br>• multiple outstanding bursts<br>• out of order completion<br>• data interleaving<br>• low-power interface | • burst transfers<br>• pipelined operation<br>• split transactions<br>• single-cycle bus master handover<br>• single-clock edge operation<br>• multiple bus masters (up to 16)<br>• two uni-directional 32-bit data bus for read and write |
| **APB4** | **APB3** | **APB2** |
| • an extension of AMBA 3 APB<br>• transaction protection (normal-privileged, secure-nonsecure, data-instruction)<br>• Sparse data transfer (partial access) | • an extension of AMBA 2 APB<br>• wait state supported<br>• error response supported | • low power<br>• latched address and control<br>• simple interface<br>• suitable for many peripherals |

# 2 Overview of AMBA AXI

AMBA AXI provides information flow channels between maters and slaves as shown in Figure 1, where address/control-information flows from master to slave while data/response-information flows master to slave or vise versa depending on read or write transaction. Information driven by master should be routed to a specific slave so that AMBA AXI uses multiplex, which consists of address decoder and arbiter. Information driven by slave should return to a

specific master that initiated the transaction so that AMBA AXI uses multiplex, which directs the specific master by using transaction identification. There is also a default slave that takes care of transactions not binding to any slaves within system.

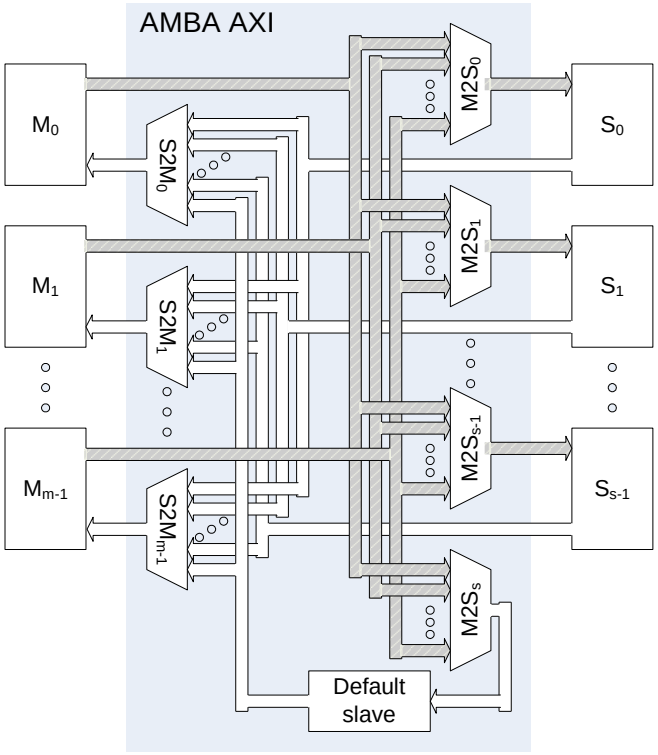



**Figure 1: Structure of AMBA AXI**

The followings are AXI protocol in short.

- Channel-based protocol
- 5 independent channels
- Single-clock edge operation: ACLK
- Two-way flow-control mechanism for each channel
- Valid/ready handshake mechanism
- Separate address/control and data phases
- Registers slices
- Easy to add register stages since each AXI channel transfers information in only one direction
- Burst-based protocol
- One address for burst
- Variable-length burst
- 1 to 16 data transfers per burst for AXI3; up to 256 for AXI4
- Multiple outstanding bursts
- Out-of-order transaction completion
- Data interleaving

- Lock and exclusive for atomic access in AXI3; only exclusive for atomic in AXI4

## 2.1 Five independent channels

There are five independent channels for each AXI interface as shown in Figure 2, where ‘write address channel’ carries address/control information for write transaction, ‘write data channel’ moves data for write, and ‘write response channel’ returns result of write transaction, while ‘read address channel’ carries address/control information for read transaction and ‘read data channel’ returns data/status of read transaction.

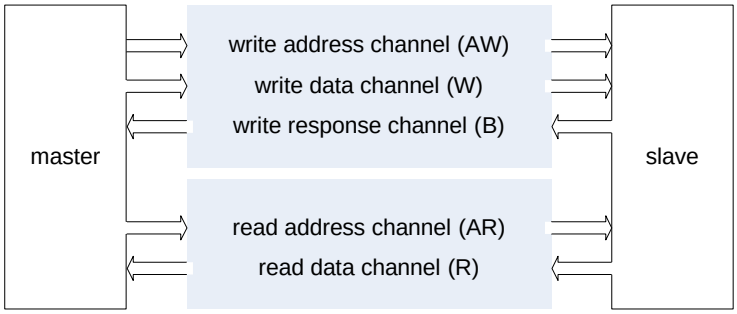



**Figure 2: Five-channel architecture**

## 2.2 Two-way VALID/READY handshake

Each channel uses two-way VALID/READY handshake protocol in order to control information flow.

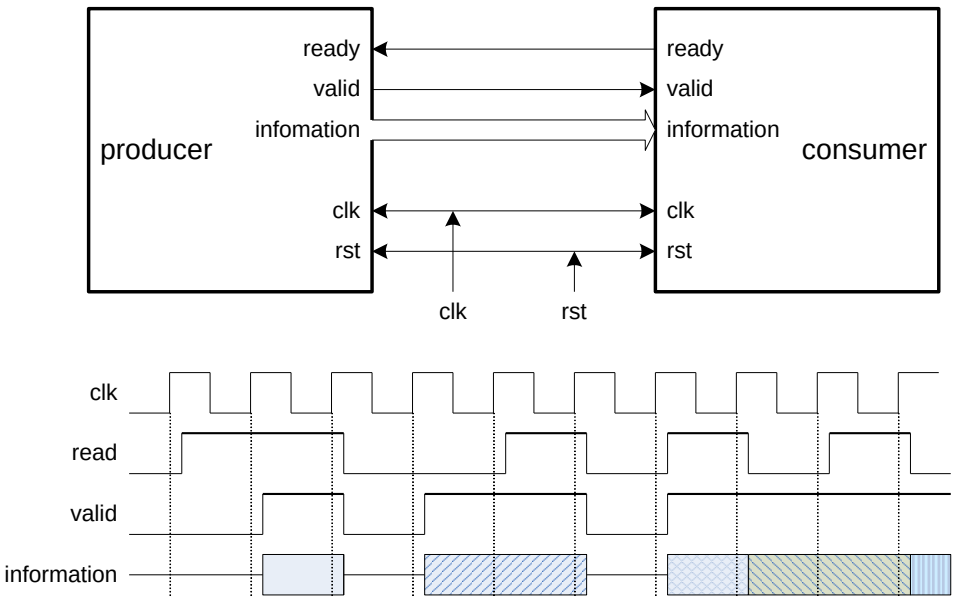



**Figure 3: Two-way VALID/READY handshake**

As a short of dual-ready handshake protocol, ready-valid handshake protocol uses two signals in order to control data movement between two blocks, where one is producer  and the other is consumer in terms of data. For write transaction, master is producer of address/data and slave is consumer, while slave is producer of response and master is consumer.

Handshake signals:

- Ready (rdy): the consumer is ready to accept data
- Valid (vld):the data is now valid

The data moves from producer to consumer whenever both ‘VALID’ and ‘READY’ are high at the rising edge of CLOCK.

## 2.3 Write transaction

Each write transaction, i.e., a write burst consists of three information transferring, which are write address, write data, and write response transfers.

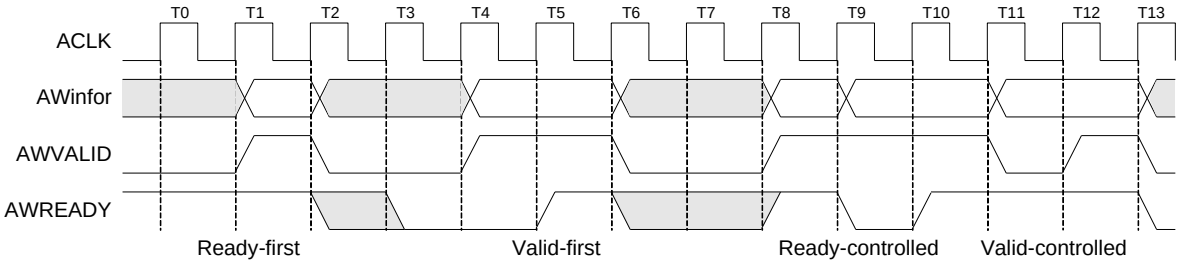



**Figure 4: Write address transferring case**

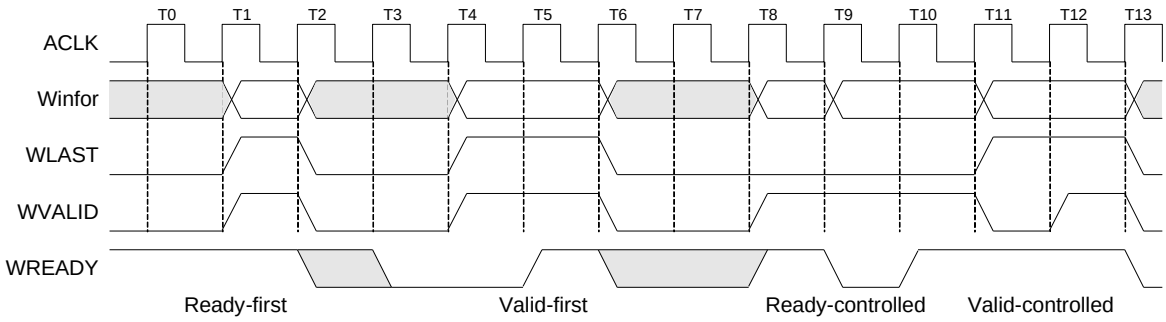



**Figure 5: Write data transferring case**

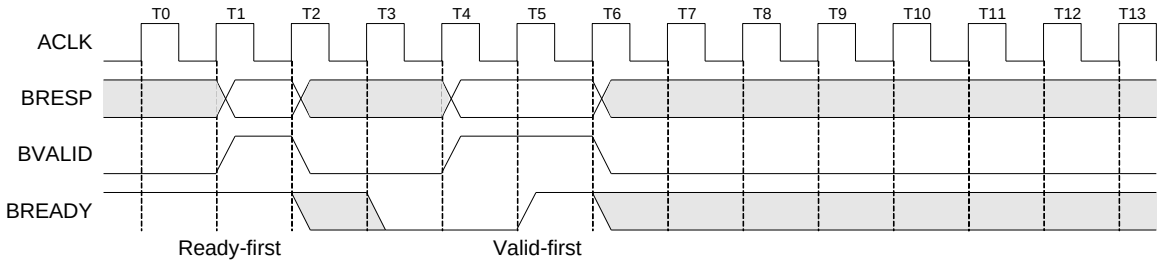



**Figure 6: Write response transferring case**

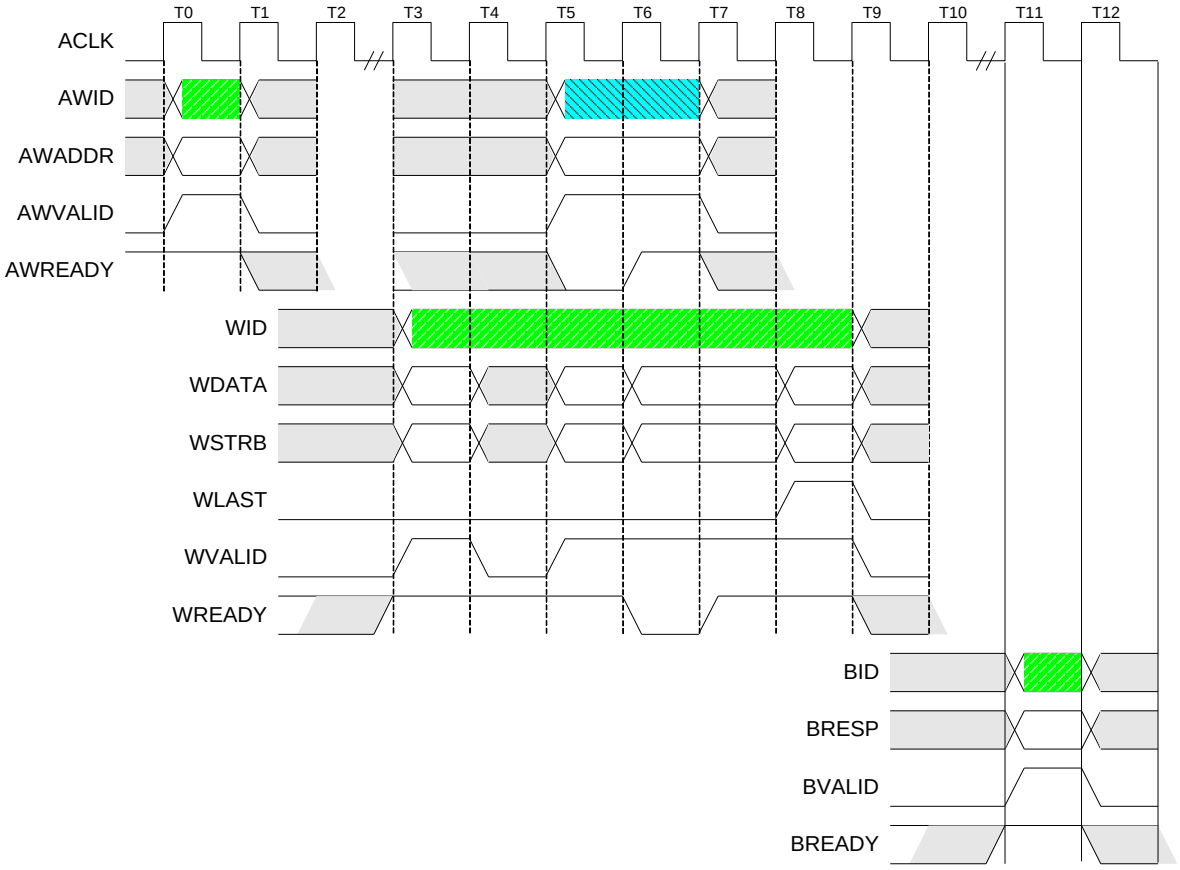



**Figure 7: An example of write**

## 2.4 Read transaction

Each read transaction, i.e., a read burst consists of two information transferring, which are read address and read data/response transfers.

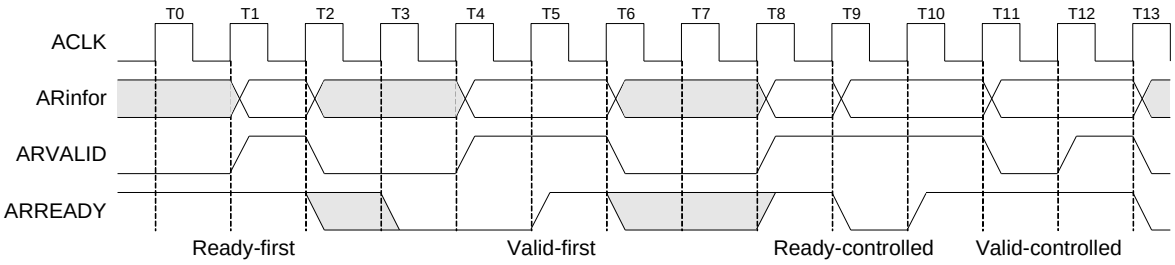



**Figure 8: Read address transferring case**

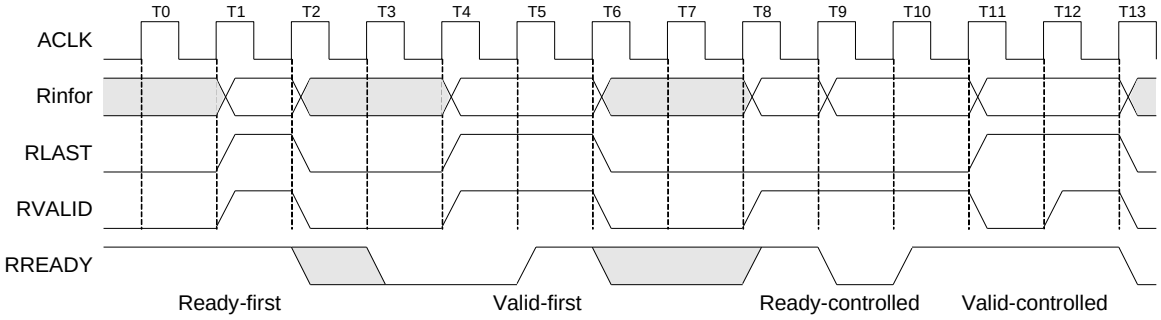



**Figure 9: Read data/response transferring case**

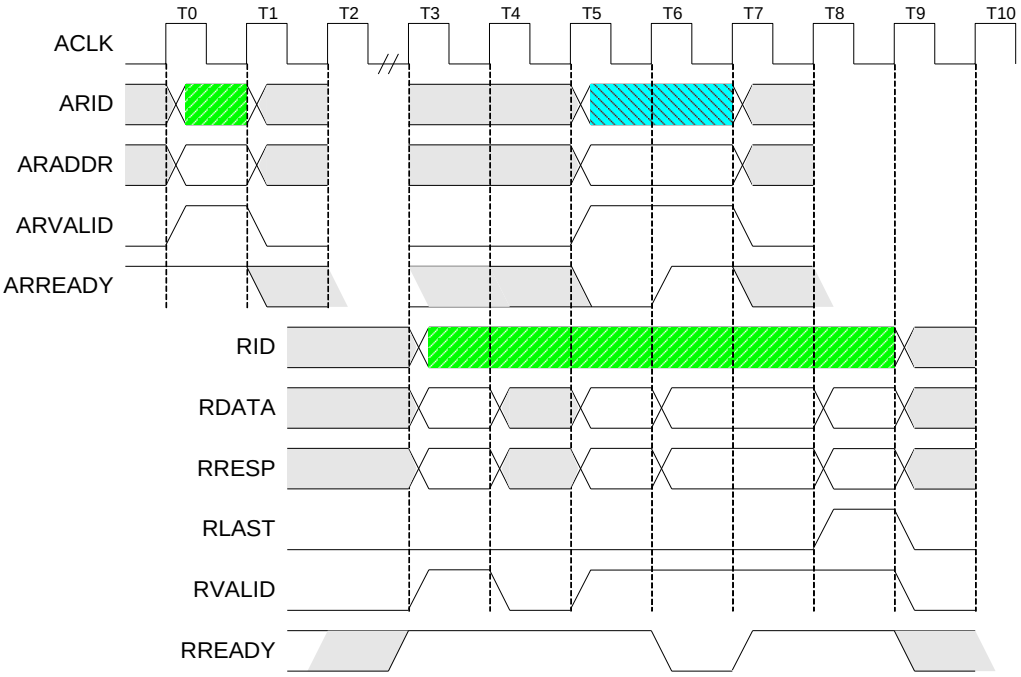
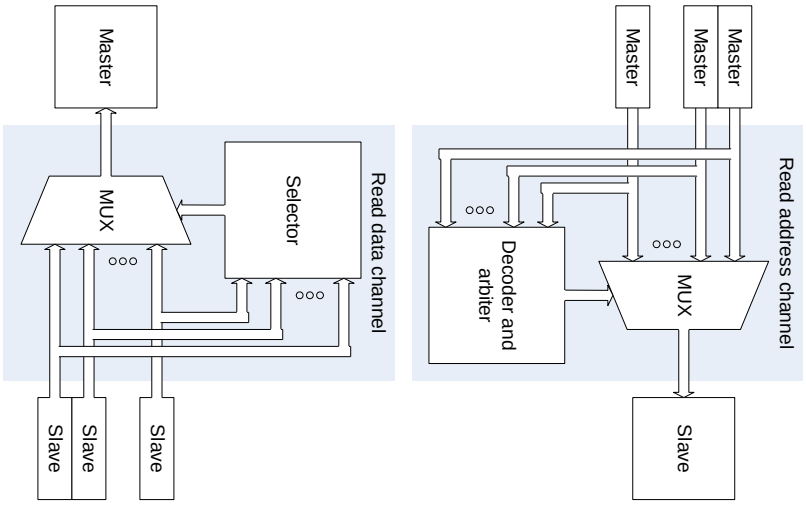
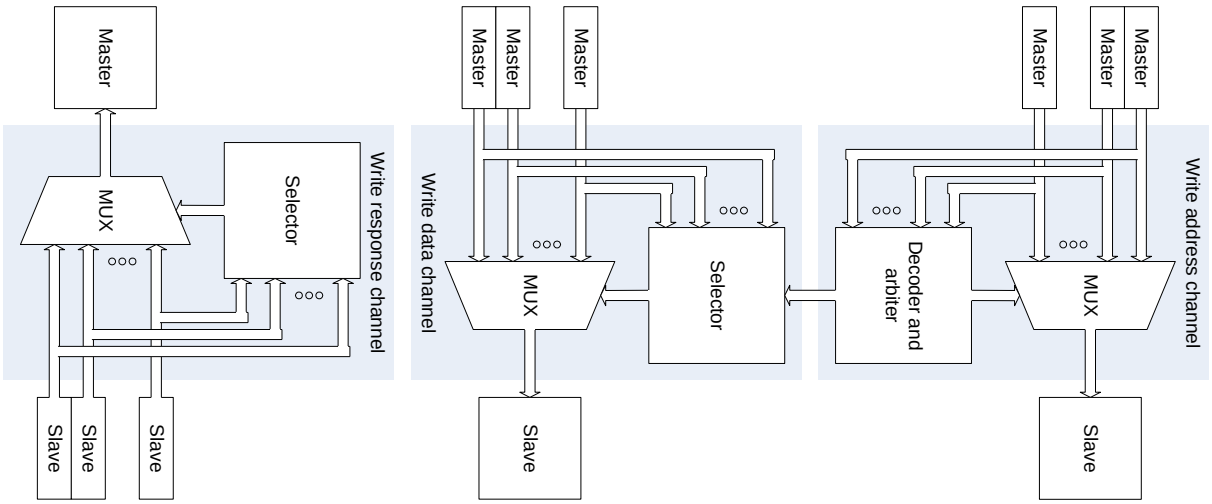



**Figure 10: An example of read**

## 3 Read channels of AXI

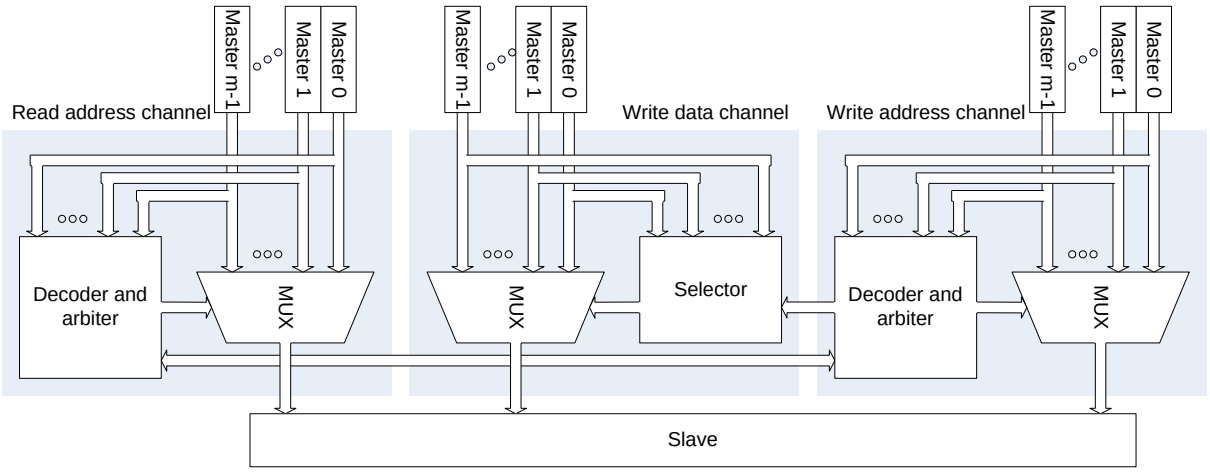



## 4 Write channels of AXI



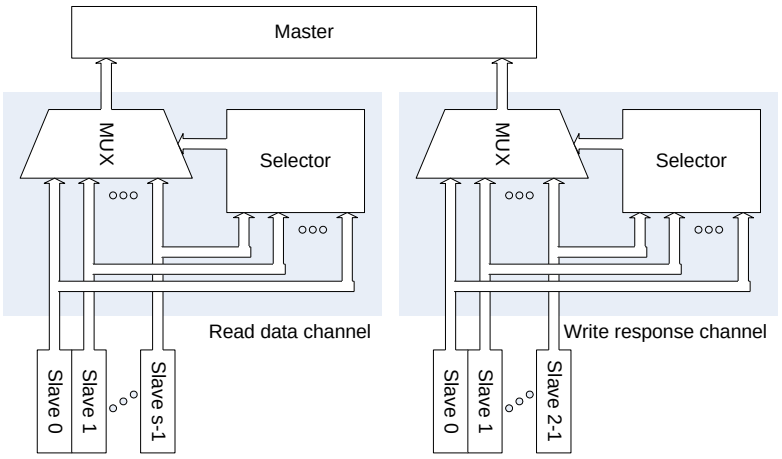

## 5 Forward channels: write-address, write-data and read-address

Forward channels are dedicated to a specific slave and handles transactions driven by multiple masters. Write-address and read-address channel use address (AWADDR, ARADDR) to select a specific slave, while write-data channel uses information given by write-address channel.




## 6 Backward channels: write-response and read-data

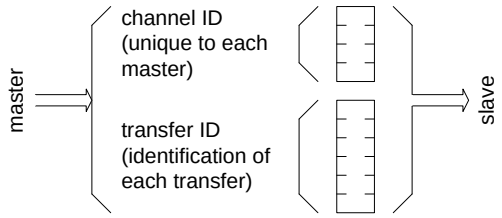
Backward channels are dedicated to a specific master and handles transactions driven by multiple slaves. Write-response and read-data channels use tags (BID and RID) to select the master that is waiting for the transaction.

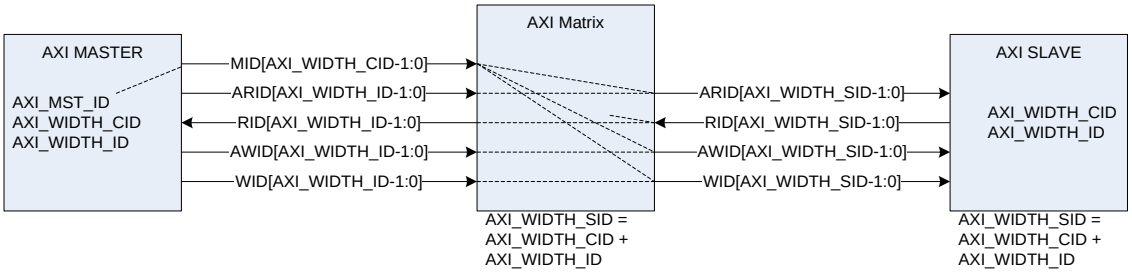



## 7 ID scheme

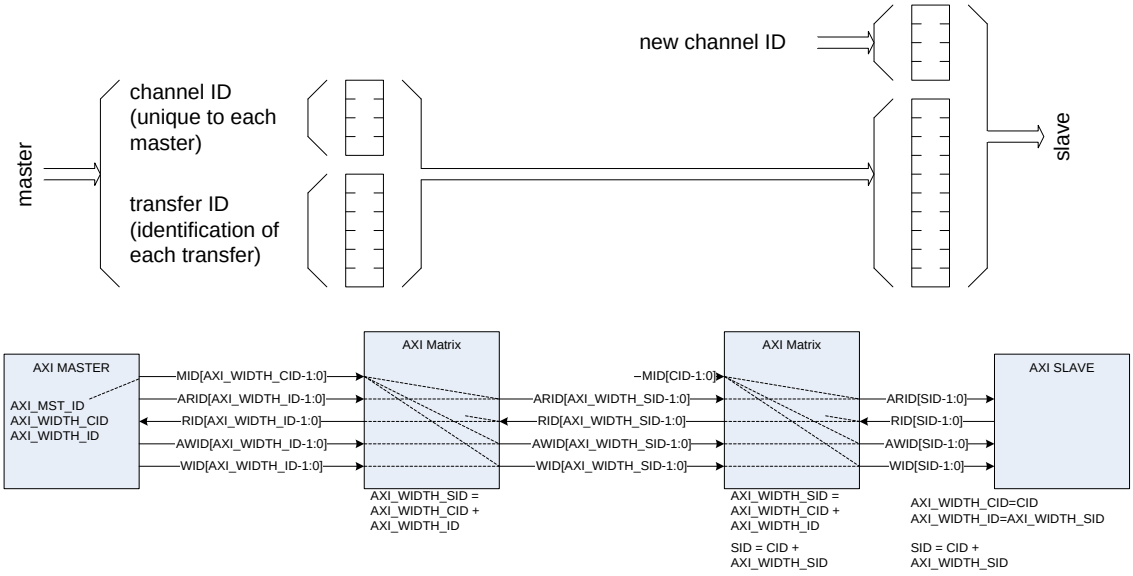
Each master will have a unique ID called master id and generates unique number called transfer id for each transfer. Those two id (master id and transfer id) form an AXI ID (including ARID, RID, AWID, WID, RID) by concatenating[1] and will be fed in to AXI matrix.

[1]For example, higher 'WIDTH_CID' will be master id and lower 'WIDTH_ID' will be transfer id.




For the case of multiple AXI bus matrix, new master id will be added at the higher part of AXI ID.




## 8 Atomic access

The atomic-lock access should consist of a read followed by a write, where the first lock-read should be issued with ARLOCK[1] is ‘1’ and the following lock-write should be issued with ARLOCK[1] is ‘0’. The AXI switch should ensure that only the master that initiates locked-read is allowed access to the slave region until an unlocked transfer from the same master completes. To do so, this AXI switch only allows locked-write from the same master after locked-read by maintaining master ID.

The atomic-exclusive access is not handled within AXI switch and simply is passed to the designated slave.

# 9 AMBA AXI modules

## 9.1 AXI switch

AXI switch (axi_switch_mxs collectively) supports m masters and s slaves. It supports atomic-lock access.

### 9.1.1 Master to slave mux

Master to slave mux (axi_mtos_m*x*, collectively) handles forward channels. Each slave including default slave will be bounded to one mux.

### 9.1.2 Slave to master mux

Slave to master mux (axi_stom_s*x*, collectively) handles backward channels. Each master will be bounded to one mux.

### 9.1.3 Arbiters

Each master to slave mux has an arbiter (axi_arbiter_mtos_m*x*) internally and each slave to master mux has an arbiter (axi_arbiter_stom_s*x*) internally.

The master to slave arbiter ensures locked read/write pair no to be interrupted by other transactions from different masters.

### 9.1.4 Default slave

Default slave (axi_slave_default) is embedded in the AXI switch and handles accesses that do not belong to any AXI slaves.

It is recommended to use this default-slave for the un-used slave port of AXI switch, which will respond ‘DECERR’ (i.e., xRESP[1:0] == 2’b11).

## 9.2 AXI to APB bus bridge

AXI to APB bus bridge (axi_to_apb) converts AXI protocol to APB protocol. It interfaces different clock domains, i.e., ACLK and PCLK.

## 9.3 AXI to AHB bus bridge

## 9.4 AHB to AXI bus bridge

## 9.5 AXI slave supporting block RAM

AXI slave supporting BRAM (Block RAM) provides memory space. Its address and data bit widths are configurable. It does not support atomic-exclusive access and multiple out-standing requests.

### 9.6 AXI DMA

### 9.7 AXI BFM

AXI BFM[3] (Bus Functional Model, trx_axi) provides communication channel between the host computer and AMBA AXI. It only supports Dynalith iCON-USB and iCON-Express and Future Design Systems CON-FMC.

## 10 Verification

### 10.1 Functional verification

'axi_master' and 'axi_slave' models are prepared in order to verify AMBA AXI.

#### 10.1.1 AXI master model

'axi_master' module generates a series of transactions, in which 'axi_master_read()' and 'axi_master_write()' tasks are used.

- Macros
  - AMBA_AXI4: AMBA AXI4 is enabled when it is defined.
    - AxLEN[7:0] instead of AxLEN[3:0]
    - AxLOCK instead of AxLOCK[1:0]
    - AxQOS
    - AxREGION
- Parameters
  - MST_ID: It identifies master and should be 'WIDTH_CID' bits wide.
  - WIDTH_CID: It specifies higher bit width of AWID, WID, BID, ARID, and RID for channel ID. It usually contails 'MST_ID'.
  - WIDTH_ID: It specifies lower bit width of AWID, WID, BID, ARID, and RID for transfer ID. Each new transfer should have unique ID.
  - WIDTH_AD: It specifies bit width of AWADDR and ARADDR.
  - WIDTH_DA: It specifies bit width of WDATA and RDATA.

```
module axi_master #(parameter MST_ID   =0        // Master ID
                             , WIDTH_CID=4     // channel ID (i.e., master id)
                             , WIDTH_ID =4       // ID width in bits
                             , WIDTH_AD =32       // address width
                             , WIDTH_DA =32)      // data width
( Global ports, Write-address ports, Write-data ports, Write-response ports, Read-address ports, Read-data ports);
```

‘axi_master_read()’ task generates a read transaction.

- addr: It specifies starting address of a burst.
- bnum: It specifies byte-number to read at a beat and it can be 1 to 16.
- bleng: It specifies the number of beat in a burst and it can be 1 to 16 for AXI3 and 1 to 256 for AXI4.
- burst: It specifies burst type and uses the same coding value of AXI specification.
- lock: It specifies type of atomic access and uses the same coding value of AXI specification.
- delay: It makes this task use a random delay for AWVALID, WVALID, BREADY, ARVALID, and RREADY.

```
task axi_master_read;
    input [WIDTH_AD-1:0] addr ;
    input [15:0]        bnum ; // num of byte
    input [15:0]        bleng; // burst length: 1, 2, ...
    input [ 1:0]        burst; // 0:fixed, 1:incr, 2:wrap
    input [ 1:0]        lock ; // 2'b10:lock, 2'b01:excl
    input               delay; // 0:don't use delay
```

‘axi_master_write()’ task generates a write transaction.

- addr: It specifies starting address of a burst.
- bnum: It specifies byte-number to read at a beat and it can be 1 to 16.
- bleng: It specifies the number of beat in a burst and it can be 1 to 16 for AXI3 and 1 to 256 for AXI4.
- burst: It specifies burst type and uses the same coding value of AXI specification.
- lock: It specifies type of atomic access and uses the same coding value of AXI specification.
- delay: It makes this task use a random delay for AWVALID, WVALID, BREADY, ARVALID, and RREADY.

```
task axi_master_write;
    input [WIDTH_AD-1:0] addr ;
    input [15:0]        bnum ; // num of byte
    input [15:0]        bleng; // burst length: 1, 2, ...
    input [ 1:0]        burst; // 0:fixed, 1:incr, 2:wrap
    input [ 1:0]        lock ; // 2'b10:lock, 2'b01:excl
    input               delay; // 0:don't use delay
```

#### 10.1.2 AXI slave model

‘axi_slave’ module handles a series of transactions.

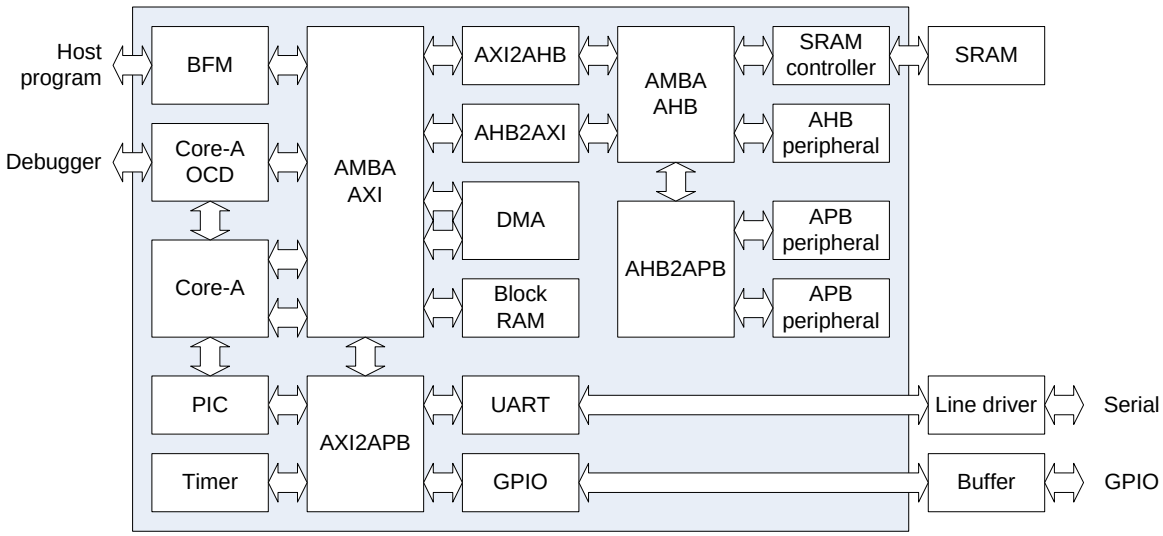
- Macros

- AMBA_AXI4: AMBA AXI4 is enabled when it is defined.
  - AxLEN[7:0] instead of AxLEN[3:0]
  - AxLOCK instead of AxLOCK[1:0]
  - AxQOS
  - AxREGION

- Parameters
  - WIDTH_CID: WIDTH_CID: It specifies higher bit width of AWID, WID, BID, ARID, and RID for channel ID. It usually contains 'MST_ID' and will be used to determine master to get the return data.
  - WIDTH_ID: It specifies lower bit width of AWID, WID, BID, ARID, and RID for transfer ID. Each new transfer should have unique ID.[2]
  - WIDTH_AD: It specifies bit width of AWADDR and ARADDR.
  - WIDTH_DA: It specifies bit width of WDATA and RDATA.
  - ADDR_LENGTH: It specifies the range of memory in the slave.

```
module axi_slave #(parameter WIDTH_CID=4        // Channel ID width in bits
                           , WIDTH_ID=4         // ID width in bits
                           , WIDTH_SID=WIDTH_CID+WIDTH_ID
                           , WIDTH_AD=32        // address width
                           , WIDTH_DA=32        // data width
                           , ADDR_LENGTH=12)
(Global ports, Write-address ports, Write-data ports, Write-response ports, Read-address ports, Read-data ports);
```

## 10.2 FPGA-based verification




[2] Each transfer will have 'WIDTH_SID' bits of ID, in which higher 'WIDTH_CID' bits will be used to determine master port by AXI bus matrix when the transfer returns and lower 'WIDTH_ID' bits will be passed to the master.

## 11 Summary

This document addresses architecture of AMBA AXI.

## 12 References

[1] ARM, AMBA AXI Protocol Specification (Version 2.0), 2010.
[2] ARM, AMBA Specification (Rev 2.0), 1999.
[3] Future Design Systems, TRX_AXI: AMBA AXI Transactor for GPIF2MST, FDS-TD-2018-04-008.

## 13 Wish list

- ☐

## 14 Revision history

- ☐ 2018.07.19: Documentation re-written by Ando Ki (adki@future-ds.com)

-- End of Document --