AXI4 Manager Interface

Specification manual

March 11, 2022 (Version 0.8.2)

Main features:

- Configurable data and address bus widths.
- Incremental AXI burst mode (INC) transactions only.
- Full support with interconnect networks and subordinates incompatible with narrow burst transactions.
- Full support to unaligned address requests.
- Automatic burst generation to achieve transfers up to 4096 kB of memory access per request, including communication with two subordinates.
- Expandable internal FIFO memory for read and writes.
- Optional implementation mode and features for traffic injector module.
- Designed in VHDL and rated for simulation. Synthesizable but not tested on FPGA hardware.
- Follows little endian data structure.

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Table of Contents

1 Introduction to the interface

This is a specification manual for the AXI4 Manager interface used on the SafeTI traffic injector project, one of the many Barcelona Supercomputing Center's (BSC) Intellectual Properties (IP). Nonetheless, the interface is also rated for general purpose modules if the communication requirements are met by this interface on AXI4 networks and components.

The AXI4 Manager interface is designed using VHDL and it is contained on the files *axi4_manager.vhd* and *axi4_pkg.vhd* for the design and library of the component respectively.

This document presents the interface and its features on a general purpose implementation on the following section 2 "General purpose interface implementation". Then, the available optimization options specific for a traffic injector module and the configurable parameters of the interface are specified on section 3 "Interface integration".

2 General purpose interface implementation

Figure 1 shows the location of the interface on a general network topology, including the communication ports between the Bus Master (BM) component and the AXI interconnect network.

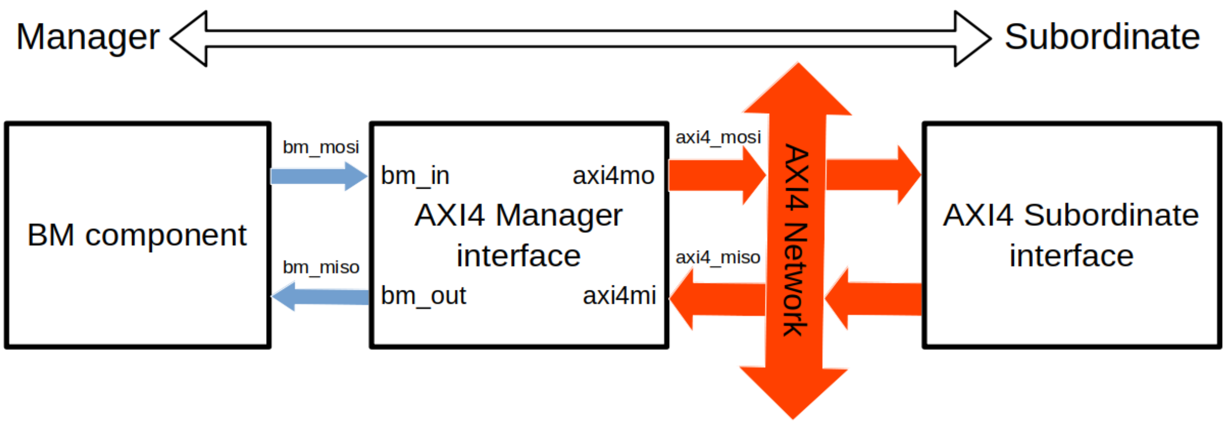

Figure 1: Simplified AXI4 network topology, including the communication ports for general use of the AXI4 Manager interface.

The Manager Input Manager Output (MISO) nomenclature followed for the communication types is that the BM component acts as the Manager of the interface. Thus, the BM component output and input through the *bm_mosi* and *bm_miso* to the AXI4 Manager interface ports *bm_in* and *bm_out* respectively. Meanwhile, the interface uses the *axi4_mosi* as output and the *axi4_miso* as input bus to connect with AXI4 network components.

Since the interface has two sides, the BM and AXI sides, the following sections show the connections of the interface and then explain the communication protocol for properly use.

2.1 BM side connections

The Table 1 shows all the connections used in the communication between the BM component and the AXI4 Manager interface.

bm_mosi connections		bm_miso connections	
Bus name	Bus purpose	Bus name	Bus purpose
wr_req rd_req	Signals used for request write and read transactions.	wr_req_grant rd_req_grant	Signals used for granting a write and read transaction requests.
wr_addr rd_addr	Bus used to set the initial address on where write to or read from.	wr_error rd_error	Signals used for indicating an error has occurred during an AXI write and read transaction.
wr_size rd_size	Bus used to set the number of bytes to transfer in the transaction (decremented by 1).	wr_done rd_done	Signals used for indicating the completion of a write and read transaction.
wr_data	Data bus used to transfer the data for write transactions.	wr_full	Signal used for indicating no more data can be read from the BM component at the moment.
Other AXI4 Manager interface input signals		rd_data	Data bus used to transfer the read data of read transactions.
rstn	Reset signal on low of the internal FIFOs and registers.	rd_valid	Signal used for indicating the interface drives valid data in the *rd_data* bus to read from.
clk	Clock signal.		

Table 1: Connections used at the BM side of the interface. All signals purpose are considered when the signal is high, unless stated otherwise.

2.2 AXI side connections

The interface features all the connections required by the AMBA AXI specification protocol for AXI4 [1]. However, it does not support all the functionalities available in the protocol. Table 2 shows the connections that are used for communication with the AXI network components. All the connections that are not on Table 2 apply the default values indicated by the protocol on section A9.3 for the outputs and the input connections are ignored.

axi4_mosi		axi4_miso	
Bus name	Bus purpose	Bus name	Bus purpose
aX_id	Bus used to set the identification number of the burst.	aX_ready	Signal used to indicate that the AXI network is ready to read from the aX output signals of the interface for the handshake process of a X transactions.
aX_addr	Bus used to set the initial address of the burst.		
aX_len	Bus used to set the number of beats of the burst (decremented by 1).	w_ready	Signal used to indicate that the AXI network is ready to read from the w_data bus during a write transaction.
aX_size	Bus used to set the transfer size of each beat in the burst.	b_id	Bus used to indicate the identification number the burst is the write response. This must match the aw_id of the handshake process for each particular burst.
aX_burst	Bus used to set the burst mode.		
aX_valid	Signal used to indicate that the aX output signals drive valid data for the handshake process of a X transaction.	b_resp	Bus used to indicate the response of the AXI network component.
w_data	Bus used to drive write data. Only the most significant **dbits** bits of the bus (check section 3) are used.	b_valid	Signal used to indicate the b MISO connections are driving a valid write response.
w_strb	Bus used to set the what byte lanes of *w_data* drive valid data.	r_id	Bus used to indicate the identification number the burst is the write response. This must match the ar_id of the handshake process for each particular burst.
w_last	Signal used to indicate the last beat on a write burst.		

Table 2: Connections used at the AXI side of the interface. All signals purpose are considered when the signal is high. The X letter indicates that the connection is present both on read (r) and write (w) cases.

w_valid	Signal used to indicate the w MOSI signals drive valid data on write transactions.	r_data	Bus used to drive read data. Only the most significant **dbits** bits of the bus (check section 3) are used.
b_ready	Signal used to indicate that the interface is ready to read a write response from the AXI network.	r_last	Signal used to indicate the last beat on a read burst. This interface depends on the correct signaling of *r_last* for the proper operation.
r_ready	Signal used to indicate that the interface is ready to read data from the AXI network.	r_valid	Signal used to indicate the r MISO connections are driving valid control signals and data.

Continuation of Table 2.

2.3 Communication protocol with the interface

The custom protocol used at the BM side of the interface follows two simple steps, a request transaction and then a data transfer.

First, the BM component generates a read or/and write request by setting to high the appropriated request signal while setting the initial address where to read from or write to and the number of bytes of the transaction, or total size, on the corresponding buses. This execution is represented at Figure 2 and Figure 3 for both write and read transaction requests, including the propagation of the request as an AXI handshake signals.

Note that this interface supports unaligned initial address and the encoded size is decremented by one of the total size of the transaction. This is achieved by aligning the address with the AXI data bus width, in a way that the interface access to more data that it is required, but transfers only the data that it has been requested. There's a transaction size limit of 4096 bytes (encoded as 0xFFF), following the memory space allocation for subordinates on the AXI4 specification protocol [1].

The request is granted once the request and grant request signals have been both high for one clock cycle at the same time. After that, the interface will set to low the grant request signal until the completion of the requested transaction and it will ignore any change on the BM control signals (this excludes the *wr_data* bus). The interface sets to high the grant request signal by default, indicating it is prepared to accept any transaction request.

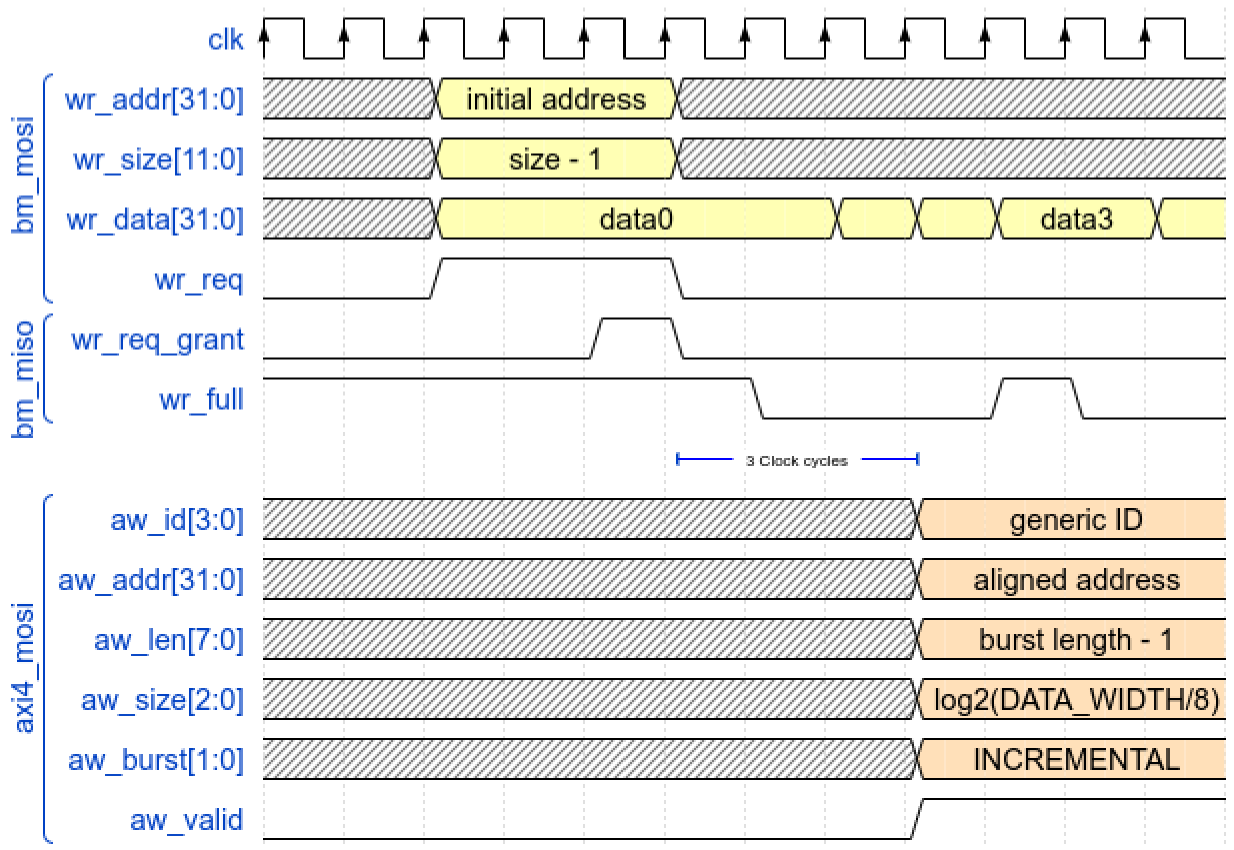

Figure 2: BM request grant and generation of the signals for an AXI handshake on a write transaction. Using default bus widths and taking into account only the valid wr_data bytes.

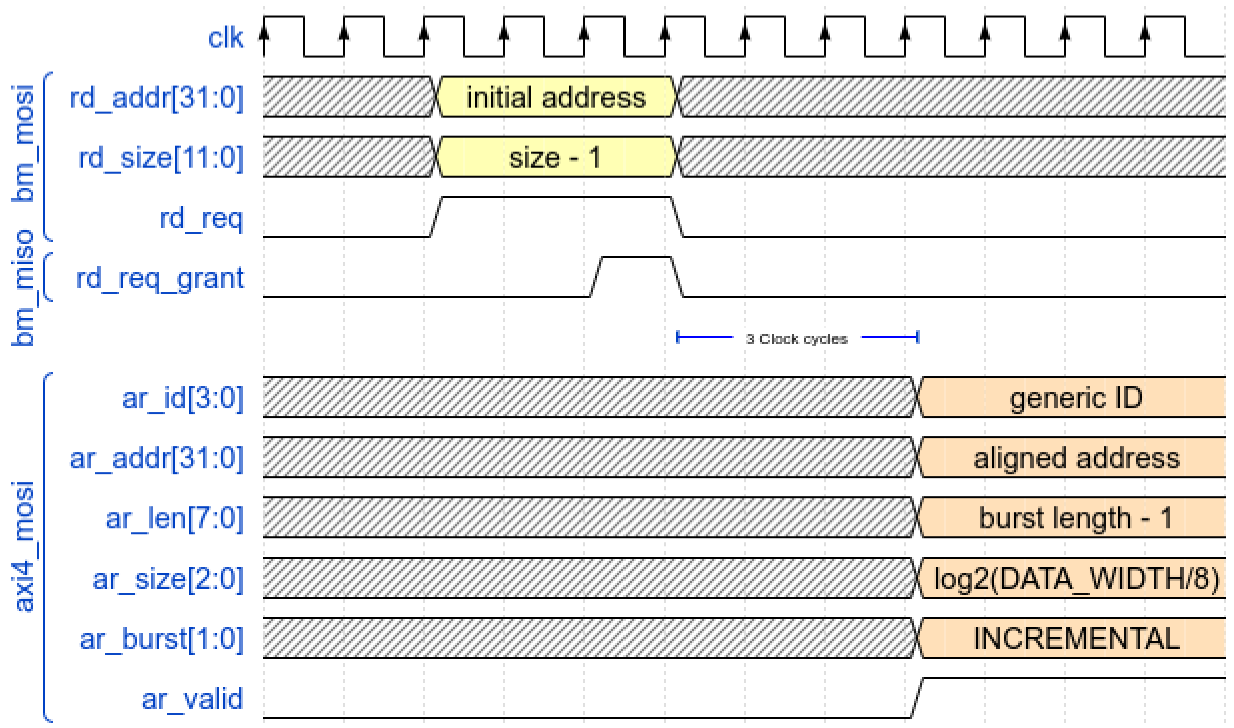

Figure 3: BM request grant and generation of the signals for an AXI handshake on a read transaction. Using default bus widths.

Second, the BM data transfer starts. In case of write transactions, the interface will set to low *wr_full* (high by default) to indicate it will read from the *wr_data* bus on each rising clock edge. Thus, the BM component must maintain the data when *wr_data* is high and continue with the next word to write when *wr_data* is low for each clock cycle, as Figure 2 show. The interface will write only a number of bytes equal to the total size set during the request, prioritizing LSB over MSB byte positions on the *wr_data* bus.

For example, if *wr_size* is 0x004 (5 bytes) during the transaction request with a *wr_data* bus width of 4 bytes, the interface will read from the *wr_data* bus each rising clock edge that the wr_full flag is low, but it only will use the first word and the least significative byte of the second word in the AXI write burst. The signal *wr_full* is only set to high when the transaction ends or when the internal write FIFO is full.

Meanwhile, in case of read transactions, the interface will set to high the *rd_valid* when the *rd_data* bus drives valid data to be read by the BM component. For each rising clock edge that the *rd_valid* signal is high, the *rd_data* bus must be read. There is no pause reading feature in this interface design from the BM side.

After the transaction is completed on both sides, the interface sets to high for one clock cycle the *wr_done* or *rd_done* signal depending on the type of the transaction it has been completed, which can be seen on Figures 4 and 5. Furthermore, during, after the completion of the transaction or even when no transaction has been made, *wr_error* and *rd_error* transmits the registered AXI error flags if the identification number *r_id* or *b_id* matches with the set on the interface *axi_id*. These are the registered signal propagation of *r_resp* and *b_resp* respectively for read and write transactions.

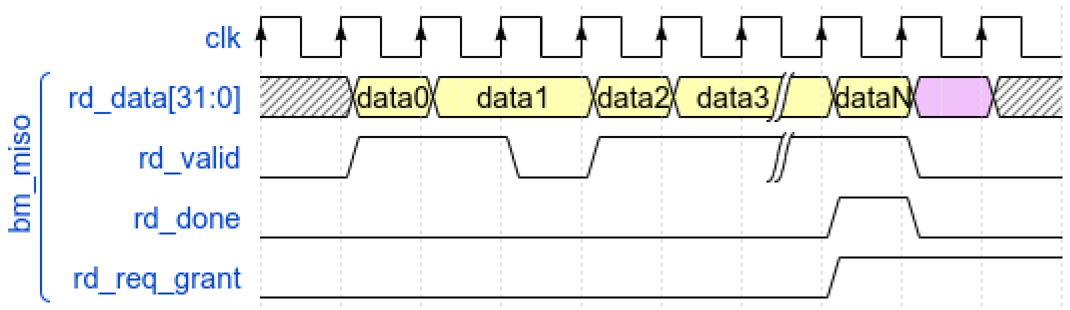

Figure 4: Timing of a BM transfer during a read transaction. Pink slots must be ignored by other components, since the valid flag is low.

In respect of the AXI side, the Manager interface follows the AXI4 protocol from the AMBA AXI specification protocol [1] by ARM. It is recommended to read the original source, even though the basics of the protocol are explained next.

The AXI4 communication protocol includes 5 channels; AW, AR, W, R and B. These are used for two main steps, the handshake (AW and AR channels) and data/response transfer (W, R and B channels) in form of a burst.

The handshake process is where the Manager interface sets the characteristics of the burst. These are signaled by the ar and aw MOSI connections for requesting a read or a write burst respectively. When both valid and ready signals are high for one clock cycle, the handshake process is completed and the data burst transfer can proceed, as Figure 5 and 6 shows for write and read bursts.

During the data transfer, the pair of valid/ready signals are used to transfer each beat. Since the burst mode applied is incremental (INC), it is expected that the subordinate will increment the address access in steps of the number of bytes in the AXI data bus width.

An important aspect about the protocol is that the AXI data bus width fixes the address alignment, in such way that the same data slot is accessed for a number of addresses equal to the number of bytes in the AXI data bus width. For this, the AXI4 protocol allows for unaligned access. However, since an unaligned data transfer will access whole data slots, this Manager interface features a smart re-arrangement of the data to only deliver the requested read data and only write the requested write data, the former using the *w_strb* bus.

If the transaction surpasses the 4 kB boundary (different subordinates) or more that one burst is required to complete the transaction (only possible when AXI data bus width is lower than 128 bits due to the maximum number of beats is 256 in the AXI4 protocol), this Manager interface will automatically generate new bursts updating the *aX_addr* and *aX_len*. However, the interface will always generate bursts that use the whole AXI data bus width.

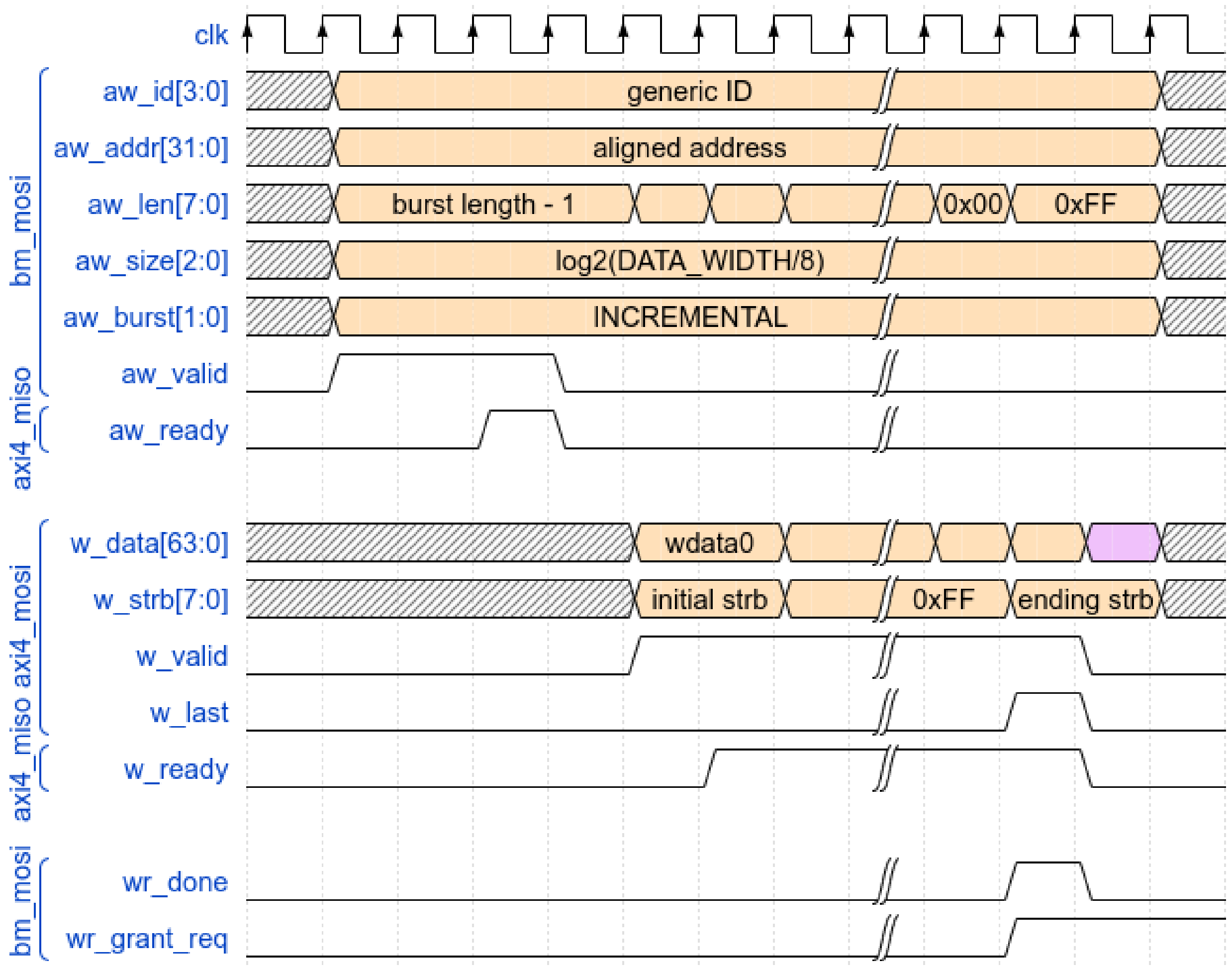

Figure 5: AXI handshake and data transfer timing for a write transaction (considering null write data). Pink slots must be ignored by other components, since the valid flag is low.

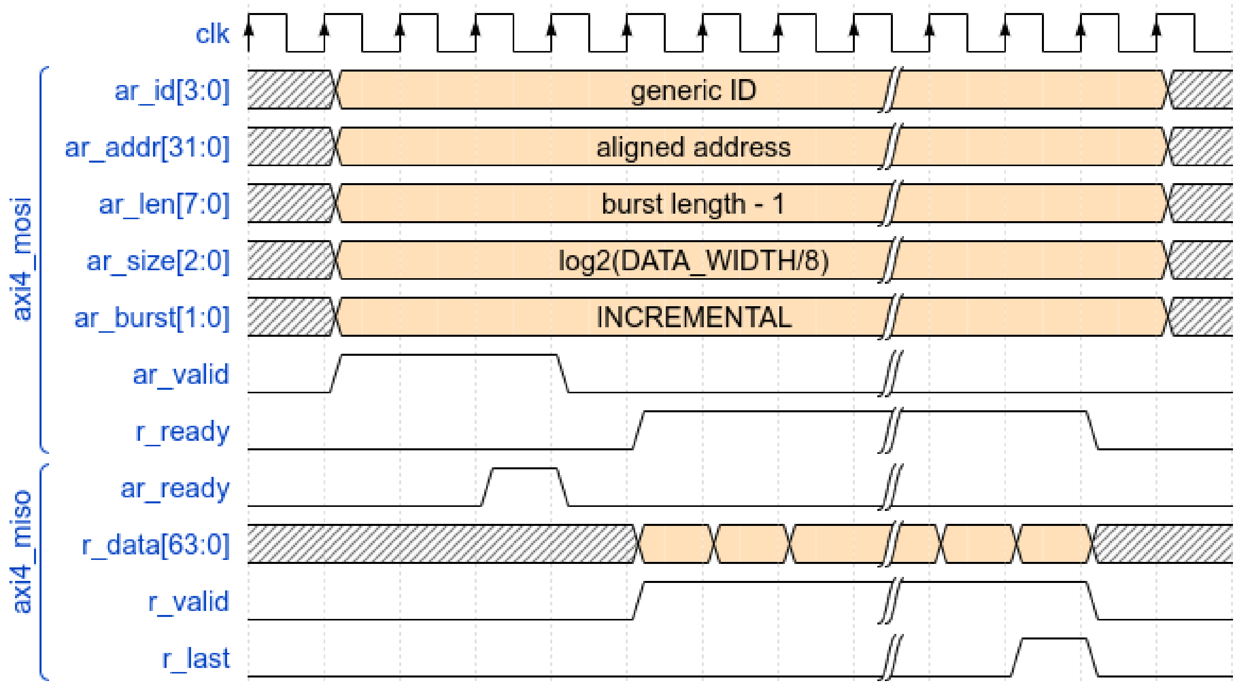

Figure 6: AXI handshake and data transfer timing for a read transaction.

3 Interface integration

The AXI4 Manager interface includes a number of parametrizable variables that allow to adapt the interface to the characteristics of the AXI network and the BM component. These are listed on Table 3.

Variable name	File location	Type	Allowed range	Default value
ID_R_WIDTH	*axi4_pkg.vhd*	integer	1 to 32	1
ID_W_WIDTH	*axi4_pkg.vhd*	integer	1 to 32	1
ADDR_WIDTH	*axi4_pkg.vhd*	integer	12 to 32	1
DATA_WIDTH	*axi4_pkg.vhd*	integer	8 to 1024*	64
axi_id	*axi4_manager.vhd*	integer	0 to 2^{ID_WIDTH}-1**	0
dbits	*axi4_manager.vhd*	integer	8 to DATA_WIDTH*	32
rd_n_fifo_regs	*axi4_manager.vhd*	integer	2 to 256*	4
wr_n_fifo_regs	*axi4_manager.vhd*	integer	2 to 256*	4
ASYNC_RST	*axi4_manager.vhd*	boolean	TRUE or FALSE	FALSE
Injector_implementation	*axi4_manager.vhd*	boolean	TRUE or FALSE	FALSE

*Table 3: Configurable parameters of the AXI4 Manager interface. *: Only power of two's are accepted. Other values will result in problematic behavior.**: ID_WIDTH makes reference to the highest in value between ID_R_WIDTH and ID_W_WIDTH.*

The parameters located on the *axi4_pkg.vhd* file will take effect on all instantiations of the interface and are specific to the AXI network characteristics, which must match. These follow the name convention indicated by the AXI4 protocol, section E1.19 [1].

Meanwhile, the parameters located on the *axi4_manager.vhd* can be modified at the instantiation of the interface, which are used as follows:

- axi_id: This is the index used at every handshake process on the AXI side. Even though the AXI4 protocol allows for this value to be dynamic, this interface doesn't support this feature in particular. Instead, it allows to set a number to allow tracking the transactions made by the interface in particular.

- dbits: Both *rd_data* and *wr_data* buses have a bus width of 1024 bits. However, the actual use of these buses depend on the data bus width of the BM component, that is configured by the variable dbits. Thus, only MSB dbits bit range are able to drive data on the *rd_data* and *wr_data* buses, as Figure 7 shows. It's expected that unused parts of the interface due to implementation options are to be optimized by the synthesis tool. Due to a design constrain, dbits must be lower or equal to the AXI data bus width DATA_WIDTH in order to ensure the correct function of the interface.

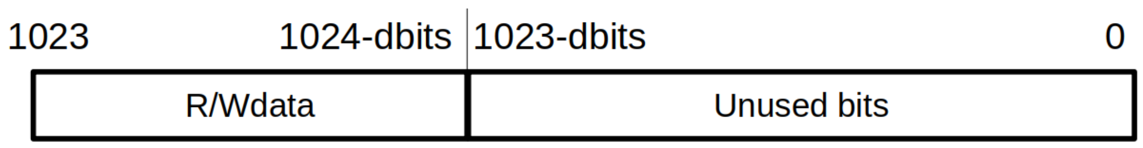

Figure 7: Bit range where data is driven on wr_data and rd_data buses of the BM side connections.

- rd_n_fifo_regs and wr_n_fifo_regs: The interface includes a number of FIFO registers to allow a continuous flux of data transfers. Thus, this two variables manage the number of registers on the FIFO used for read and write transactions respectively.

  Incrementing rd_n_fifo_regs will increase the performance of the interface in reads by finishing the AXI burst earlier, freeing the AXI subordinate, since more data can be cached. On write transactions, incrementing wr_n_fifo_regs any further may improve the BM data throughput on cases where the AXI side cannot sustain a stable number of beats, but it depends on the ratio of data widths between both BM and AXI sides.

- ASYNC_RST: This flag allows for asynchronous reset of the interface when enabled.

- Injector_implementation: This flag allows to enable the optimizations specific for the traffic injector module. Further information can be read on section 4.

This concludes all what there's to know in order to use the interface for general purpose communication with an AXI4 compatible network or component. The following section 4 introduces special features to accommodate compatible traffic injector modules with the BM protocol of this interface.

4 Interface optimization for traffic injector modules

This AXI4 Manager interface features various optimizations to increase the control over the transactions that are executed on an AXI network by setting the Injector_implementation flag to TRUE. In particular, the use of this mode allows to bypass the data bottleneck due to dbits being many times lower than the AXI data bus width DATA_WIDTH, increasing the transaction throughput to the maximum allowed by the AXI network. Furthermore, this mode saves resources on part of the logic of write transactions as Figure 8 shows.

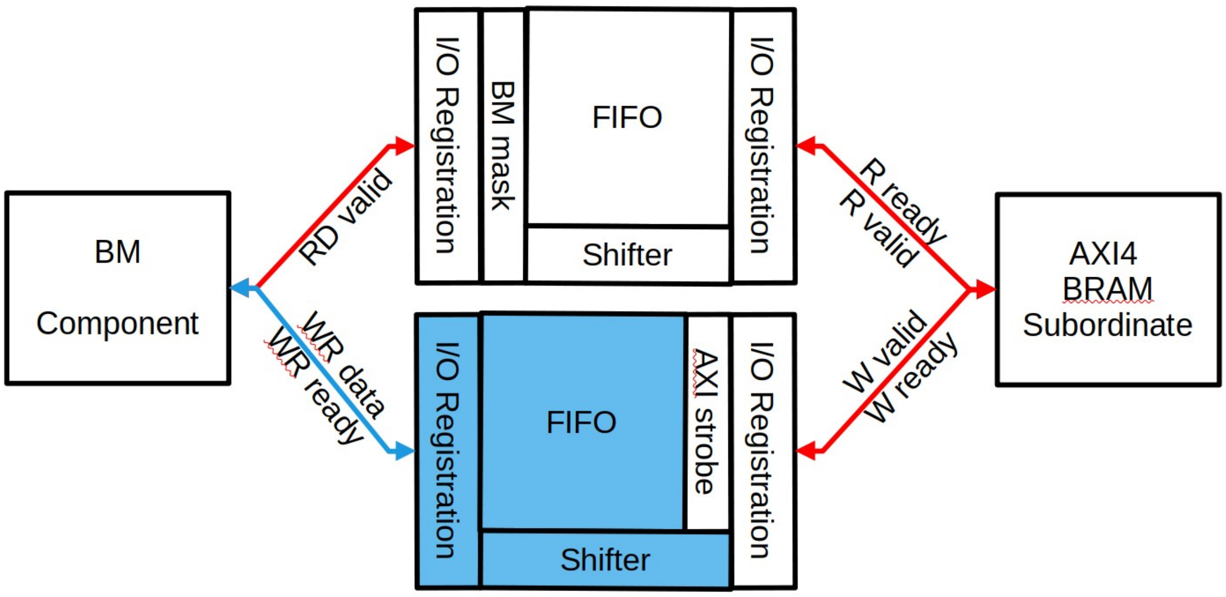

Figure 8: Data infrastructure for read and write transactions. Marked in blue the components saved when the Injector_implementation is set to true.

The bypass, however, makes that all byte in write transactions use full zeros and read data is discarded if the input *bm_in_bypass_rd* is set to high. This way, the injector is able to read from the AXI memory space by setting *bm_in_bypass_rd* to low on the load of the injection program and obtain high transaction throughput by setting it to high during the execution of the injector program. On the case where the the Injector_implementation mode is set to FALSE, the input port *bm_in_bypass_rd* will not be read, even though it is recommended to input a constant low.

References

[1] “AMBA AXI and ACE Protocol Specification” (IHI 0022H.c), 26 Januray 2021 by ARM.