AXI3 Protocol VIP Development

A Major Project

Submitted in Partial Fulfilment of the Requirement

for the degree of

Master of technology

In

(VLSI Design)

By

Mevada Nikul
22MECV13

Department of Electronics & Communication Engineering
Institute of Technology, Nirma University

Ahmedabad-382481

December-202

AXI3 Protocol VIP Development

A Major Project

Submitted in Partial Fulfilment of the Requirements

for the degreeof

MASTER OF TECHNOLOGYIN

VLSI DESIGN

By

Mevada Nikul

(22MECV13)

Internal Guide:

Prof. Akash Mecwan

EC Department, Institute of technology Nirma University, Ahmedabad

External Guide

Mr. Mukund Raj Rathore

Verification Engineer

Scaledge India PVT. LTD

Ahmedabad

Department of Electronics & Communication EngineeringInstitute of Technology, Nirma University

Ahmedabad - 382 481

December 2023

Certificate

This is to certify that the Major Project entitled **"AXI3 Protocol VIP Development"** submitted by **Mevada Nikul (22MECV13)**, towards the partial fulfillment of the requirements for the Masters of Technology in VLSI Design, Nirma University, Ahmedabad is the record of work carried out by him under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this Project, to the best of our knowledge haven't been submitted to any other university or institution for award of any degree or diploma.

Date:
Place: Ahmedabad

Pro. Akash Mecwan

Internal Guide,
Institute of Technology,
Nirma University,
Ahmedabad

Dr. Usha Mehta

HOD of Electronics and communication department, Institute of Technology, Nirma University,
Ahmedabad

Dr. Usha Mehta

Professor and Head,
EC Department,
Institute of Technology,
Nirma University, Ahmedabad.

Dr. Rajesh Patel

Director,

Institute of Technology, Nirma University, Ahmedabad.

SCALEDGE

External Certificate

This is to certify that the Major Project Report entitled" **AXI3 Protocol VIP Development"** submitted by **Mevada Nikul (**Roll No. **22MECV13**) as the partial fulfilment of the requirements for the award of the degree of Master of Technology in VLSI Design, Electronics and Communication Engineering, Institute of Technology, Nirma University is the record of work carried out by him under my supervision and guidance. The work submitted in our opinion has reached a level required for being accepted for the examination.

Date:

External Guide

Mukund Raj Rathore

Verification Engineer

Scaledge India PVT. LTD

Statement of Originality

I, **MEVADA NIKUL**, Roll No: **22MECV13**, give undertaking that the MTech thesis entitled " **AXI3 Protocol VIP Development**" submitted by me, towards the partial fulfillment of the requirements for the degree of Master of Technology in **Electronics & Communication Engineering (VLSI Design)** of Institute of Technology, Nirma University, Ahmedabad, contains no material that has been awardedfor any degree or diploma in any university or school in any territory to the best of my knowledge. It is the original work carried out by me and I give assurance that no attemptof plagiarism has been made. It contains no material that is previously published or written, except where reference has been made. I understand that in the event of any similarity found subsequently with any published work or any dissertation work elsewhere; it will result in severe disciplinary action.

Signature of Student

Date:

Place: Ahmedabad

Endorsed by

Pro. Akash Mecwan

(Signature of Guide)

Acknowledgements

It gives me immense pleasure in expressing thanks and profound gratitude to **Pro. Akash Mecwan**, Assistant Professor, EC Department, Institute of Technology, Nirma University for his valuable guidance and continual encouragement throughout this work.The appreciation and continual support he has imparted has been a great motivation to me in reaching a higher goal. His guidance has triggered and nourished my intellectual maturity that I will benefit from, for a long time to come.

It gives me a great pleasure to thank **Dr. Usha Mehta**, PG Coordinator of VLSI Design, EC Department, Institute of Technology, Nirma university for his guidance throughout whole M.Tech and in this project thesis.

It gives me an immense pleasure to thank **Dr. Usha Mehta**, Hon'ble Head of EC Department, Institute of Technology, Nirma University for his kind support and providingbasic infrastructure and healthy research environment.

I would also like to thank my scaledge colleague **Mukund Raj Rathore** (Verification engineer), **Disha Purohit** (Senior Verification engineer) their continuous support and guidance throughout my M.Tech thesis project scaledge.

I would also thank the Institution, all faculty members of Electronics and CommunicationEngineering Department, Nirma University, Ahmedabad for their special attention and suggestions towards the project work.

Mevada Nikul

(22MECV13)

Contents

Abstract

AXI Protocol is a part of ARM Advanced Microcontroller bus architecture (AMBA) Family. Verification of IP very complex due to System on Chip allows the integration of different Intellectual Properties. Advanced eXtensible Interface (AXI) protocol provides high bandwidth, low latency, and can able to operate at high frequencies. And comparing with other AXI provides better efficiency as compare to other AMBA protocols.

Verification of SoC take more time to verify because it is on chip, so developing a reusable Verification IP that allows to reuse this verification environment to verify other SoCs also. The meaning of reusable VIP time taken to verify the SoCs will be greatly reduced. To creating a Verification IP for the AMBA AXI3 (Advanced eXtendsible Interface) protocol by using Universal Verification Methodology (UVM). Here RTL of Master or slave of AXI are not Present here one agent of test bench are work as AXI Master Agent and Second Agent work as AXI Slave of AXI here Required any RTL so Also this concept know as Back-to-Back VIP Development.

AXI3 Protocol Support five independent channels for write, Read or responds channels. In AXI Multiple outstanding transaction and out of order transaction are also cover in verification test plane of axi. Implement test cases scenarios to achieve high functional coverage of axi testbench, ensuring that the AXI VIP effectively exercises the different features and corner cases of the AXI protocol.

Chapter 1

Introduction

- The Because of the AMBA protocol's flexibility, it may be used in a variety of SoC architectures with different size, power, and performance requirements AMBA protocols are widely used open standards, ensuring compatibility across IPs from different providers for SoCs. Because of this interoperability, low-friction integration and IP reuse are possible, resulting in a faster time to market.
- The AMBA AXI protocol provide high-performance and high-frequency for communication between Master and Slave components.
- AXI transactions are typically request-response based, where master initiates a transaction and slave responds according to transection.
- AXI supports multiple master – slave configurations that can contain multiple master – slave configuration.
- AXI protocol have 5 Independent Channels for write read and response.
- All channels has a group of signals required to perform particular operation, write address channel have control, write data information.

1.1 Common Terminologies

- **AW**: AXI Write Address channel.
- **W**: AXI Write Data channel.
- **B**: AXI Write Response channel.
- **AR**: AXI Read Address channel.
- **R**: AXI Read Response channel.

- **Transaction**: The AXI bus handles a complete set of required operations, which can include one or more data transfers.
- **Transfer**: one exchange of information, with valid and ready handshake occurred , handshaking signals available in each channel.
- **Beat**: An individual data transfer within an AXI burst.

- **Aligned:** A data item is considered aligned if its address is divisible by the highest power of 2 that fits its size in bytes. For example, addresses can be aligned with halfwords, words, and doublewords.
- **Unaligned:** These are memory accesses that are not, or might not be, properly aligned to halfwords, words, or doublewords.
- **Outstanding addresses**: This refers to the ability to issue multiple addresses for transactions without waiting for previous transactions to complete. This feature improves system performance by allowing parallel processing of AXI transactions.
- **Out of order transaction:** This means transactions can be completed in a different order than they were started. Faster transactions can finish without waiting for slower ones, improving system performance by reducing delays.
- **Data Interleaving**: Masters producing write data sequence to same slave but data not arriving each clock, interleave to avoid idle cycles on bus.
- **Interleaving Depth:** It is a maximum number of transactions for slave to accept data for interleaved transfers. The default Interleaving depth is 1.

1.2 List of AXI 3.0 Protocol features

- AXI is capable for high-bandwidth and low-latency type applications.
- AXI has separate, independent address and data channels.
- AXI supports unaligned data transfers using byte strobe signals.
- AXI uses burst-based transactions, where only the start address is issued and the remaining addresses are automatically calculated by the slave.
- AXI supports issuing multiple outstanding addresses.
- AXI supports out-of-order transaction completion.
- AXI provides high-frequency operation without the need for complex bridges.

Chapter 2

AXI Channel architecture

2.1 Channels in AXI 3 for Transection:

- axi read address channel.
- axi read data channel.
- axi write address channel.
- axi write data channel.
- axi write response channel.

2.1.1 Write Channel Architecture

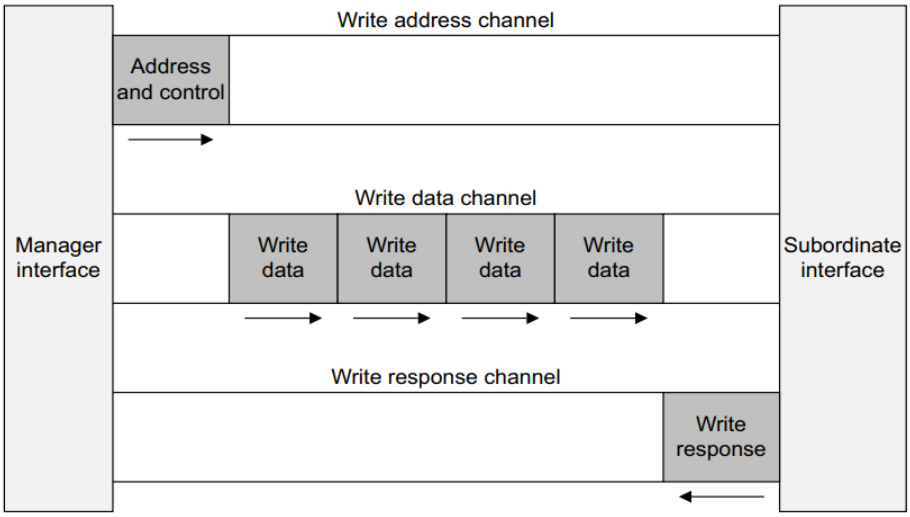


Fig 2.1.1: WR transaction need the WR address and WR data also WR response channels.

2.1.2 W (WRITE) Channel Architecture

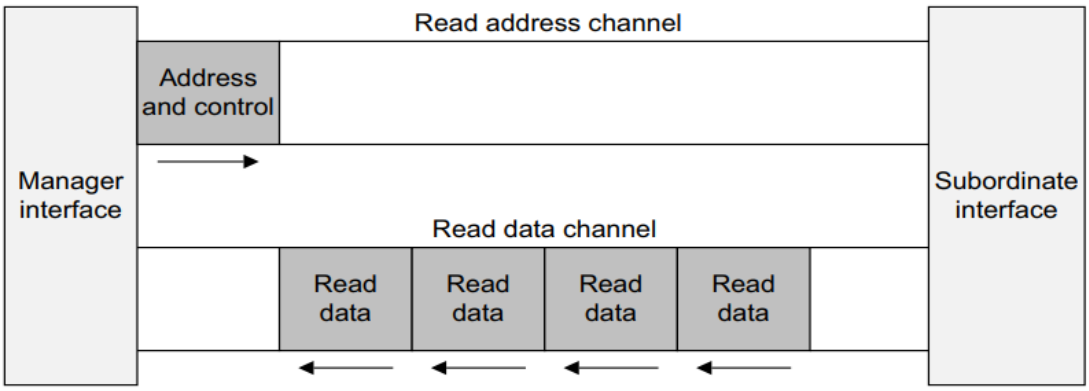


fig 2.1.1 RD transaction need the RD address and RD data channels only.

2.2 AXI 3.0 Channels definition

- AXI has five independent channels, each with a set of information signals and VALID and READY signals that provide a two-way handshake mechanism.
- The master initiates the VALID signal to indicate when valid address, data, and control information are available in the channel.
- AXI includes a WLAST signal to indicate the last data item in a write transaction, RLAST signal to indicate the final data item in a read transaction.
- Read and write transactions each have their own address channels, carrying all the required address and control information for a transaction.

Chapter 3

Signal Descriptions

3.1 AXI 3.0 Global Signals.

Signals	Source	Signal Description
Aclk	Clk signal	All signals of channels sampled on clock edge
ARESETn	Reset signal	Active low, put all signals at its default state

3.2 AXI 3.0 AW (Write address) Channel Signals

Signals	source	Signal Description
AWID	M	Write address ID signal gives identification for the write address channels signal group.
AWADDR	M	write address gives initial address to slave of transfer.
AWLEN	M	Burst length gives number of transfers will happen single transection information to slave.
AWSIZE	M	Burst size, this signal gives information about how many bytes in transfers.
AWBURST	M	Burst type give information about how calculate next address for next transfer in the transaction.
AWLOCK	M	Lock type give information about the atomic access of the transfer like NORMAL, EXCLUSIVE.
AWVALID	M	This signal indicates that the channel contains valid WR_address, WR_control information.
AWREADY	S	Signal indicates that the slave is now ready to accept an WR_address, WR_control information form master side.

3.3 AXI 3.0 W (Write Data) Channel Signals

Signals	Source	Signal Description
WID	M	Write ID signal gives identification for the write address channels signal group.
WDATA	M	Write data.
WSTRB	M	Write strobe signal indicate how many bytes of data are valid in transfer.
WLAST	M	Write last indicate last transfer in transection.
WVALID	M	Write valid, this signal indicates that the channel contains valid write data and strobe information.
WREADY	S	Signal indicates that the slave is now ready to accept an WR data information form master side.

3.4 AXI 3.0 B (Write Response) Channel Signals

Signals	Source	Signal Description
BID	S	This signal gives identification for the WR response channels signal group.
BRESP	S	This signal give status about write transection, transection pass or failed.
BVALID	S	This signal indicates that the channel contains valid WR_response and strobe information.
BREADY	M	This signal indicates that the slave is now ready to accept an WR_response information form slave side.

3.5 AXI 3.0 AR (Read address) Channel Signals

Signals	source	Signal Description
ARID	M	read address ID signal gives identification for the read address channels signal group.
ARADDR	M	read address gives initial address to slave of transfer.
ARLEN	M	Burst length gives number of transfers will happen single transection information to slave.
ARSIZE	M	Burst size, this signal gives information about how many bytes in transfers.
ARBURST	M	Burst type give information about how calculate next address for next transfer in the transaction.
ARLOCK	M	Lock type give information about the atomic access of the transfer like NORMAL, EXCLUSIVE.
ARVALID	M	This signal indicates that the channel contains valid RD_address, RD_control information.
ARREADY	S	Signal indicates that the slave is now ready to accept an RD address, RD control information form master side.

3.6 AXI 3.0 R (Read Data) Channel Signals

Signals	Source	Signal Description
RID	S	read data ID signal gives identification for the read data channels signal group.
RDATA	S	Read data.
RRESP	S	This signal give status about read transection, transection are failed or pass.
RLAST	S	Read last indicate last transfer in transection.
RVALID	S	This signal indicates that the channel contains valid RD_data, RD_control information.
RREADY	S	Signal indicates that the master is now ready to accept an RD_address, RD_control information form slave side.

3.7 AXI Signals Direction

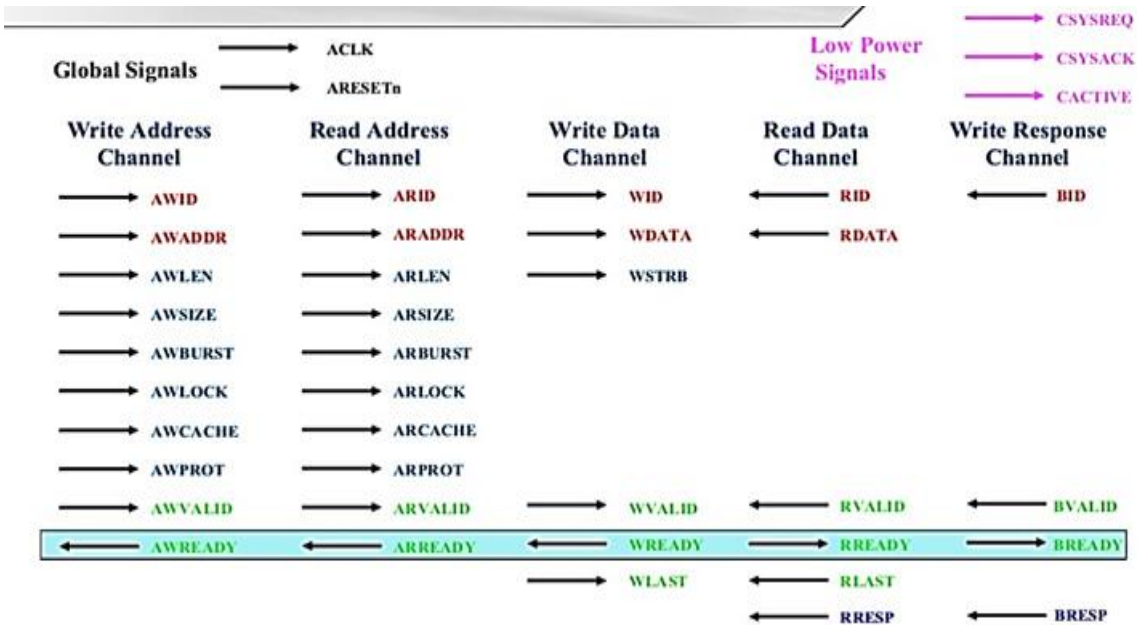


fig 3.7 AXI all signals' Directions

3.8 AXI Master – Slave Configuration

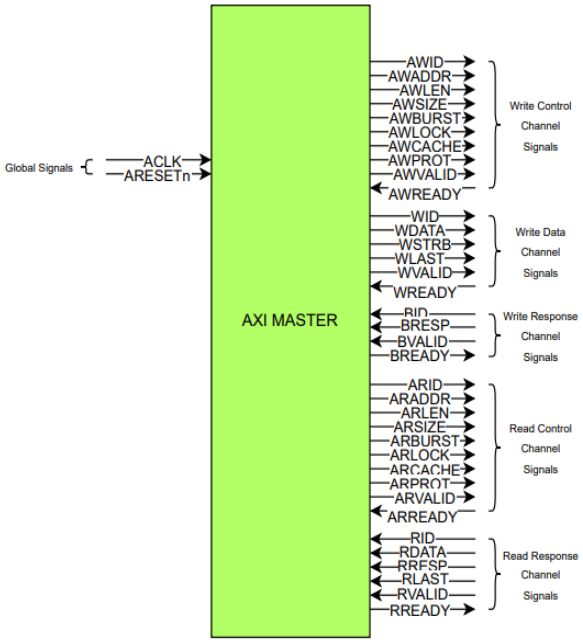


Fig 3.8.1 AXI master

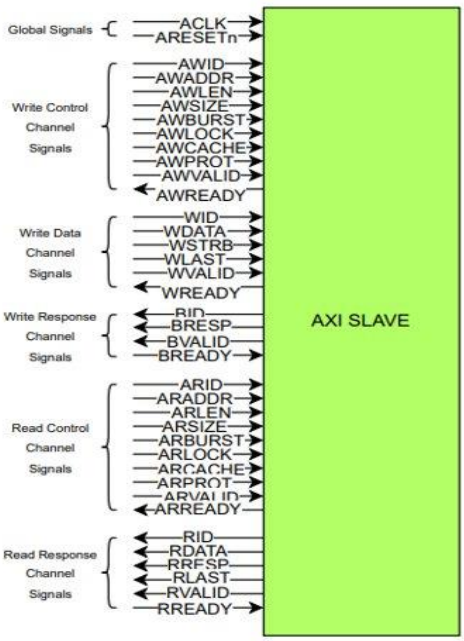


fig 3.8.2 AXI Slave

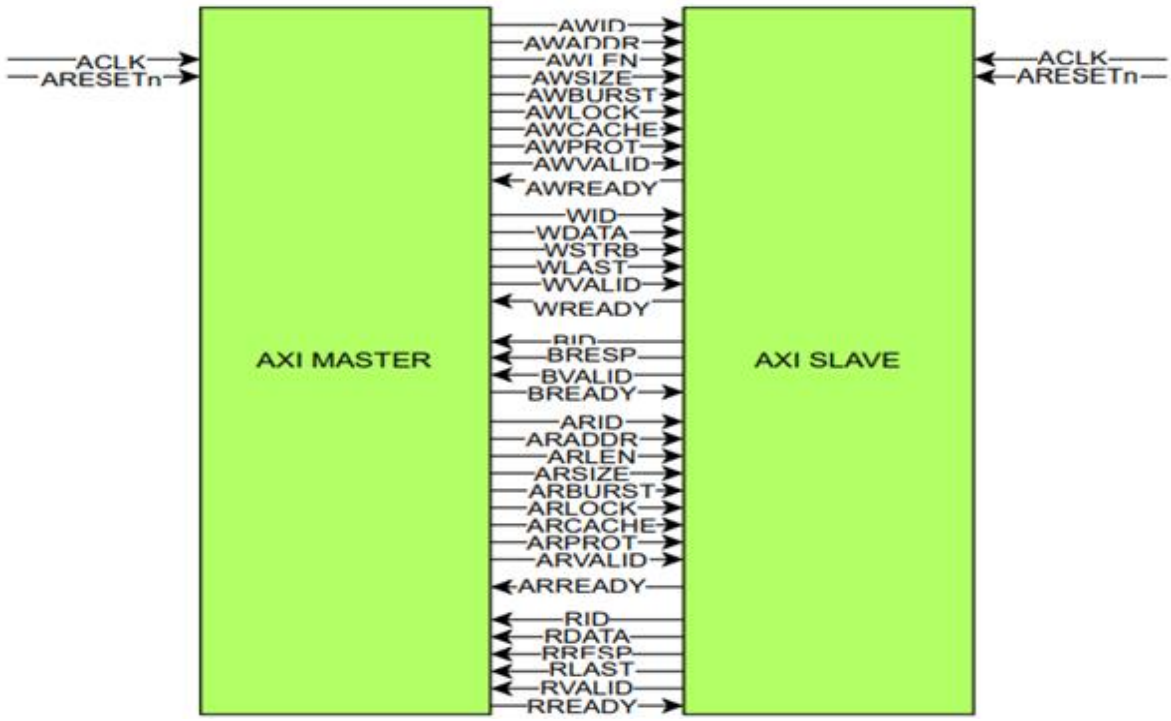


fig 3.8.3 AXI masters Slave configuration

Chapter 4

Handshake Mechanism in AXI

- In AXI, each of the five transaction channels has its own handshaking signals, utilizing the same VALID and READY signals for transferring address, data, and control information.
- The Master produces the VALID signal to indicate that the address, data, or control information is valid.
- The Slave produces the READY signal to indicate that it is ready to accept the information.
- A successful transfer occurs only when both the VALID and READY signals are high.
-

4.1 VALID assert High before READY assert High handshake

- The Master provides information after the P1 clock edge and turns on the VALID signal.
- The Slave activates the READY signal high after the P2 clock edge.
- The Master needs to maintain its information steady until the transfer happens at the P3 clock edge.

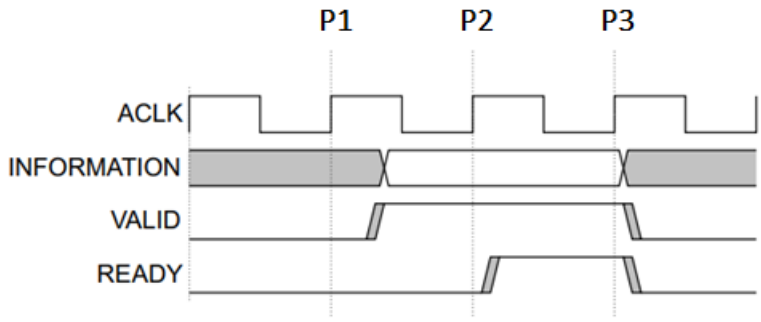


fig 4.1 VALID before READY handshake

- A Master cannot wait until READY is confirmed before activating VALID.
- When VALID is activated, it must stay activated until the handshake happens, ensuring a successful transaction at a rising clock edge when both VALID and READY are activated.

4.2 READY assert High before VALID assert High handshake

- After the P1 clock edge, the Slave activates READY before the address, data, or control information is valid.
- This activation shows that it's ready to receive the information.
- The Master provides the information and activates VALID after the P2 clock edge, leading to the transfer happening at the P3 clock edge.
- In this scenario, the transfer successfully happens within one cycle.

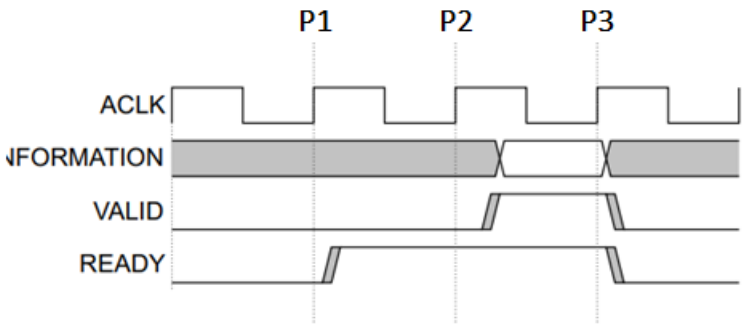


fig 4.2 READY before VALID handshake

4.3 READY assert High before VALID assert High handshake

- After the P1 clock edge, both the Master and Slave are ready to transfer address, data, or control information.
- The transfer takes place at the rising clock edge when both VALID and READY are activated.
- The transfer happens at the P2 clock edge.

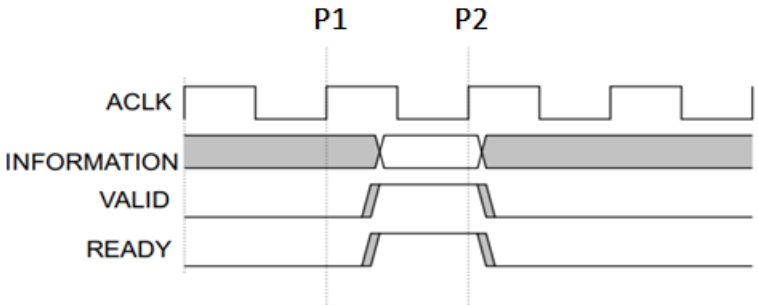


fig 4.3 VALID with READY handshake

4.5 AXI 3.0 channel Dependencies between handshake signals

1.5.1 Read channel dependencies

- The slave can wait for ARVALID to be activated before it activates ARREADY.
- The slave must wait for both ARVALID and ARREADY to be activated before it begins to return read data by activating RVALID.

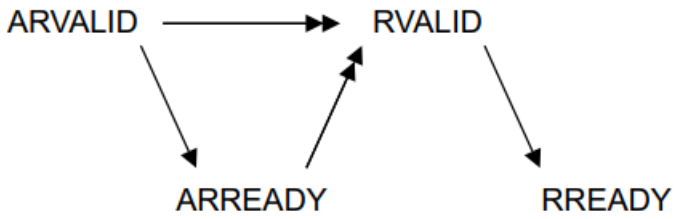


fig 4.5.1 dependencies in read channels handshake

1.5.2 Write channel dependencies

- The master should not wait for the slave to activate AWREADY or WREADY before activating AWVALID or WVALID.
- The slave can wait for AWVALID or WVALID, or both, before activating AWREADY.
- The slave can wait for AWVALID or WVALID, or both, before activating WREADY.
- The slave must wait for both WVALID and WREADY to be activated before activating BVALID.

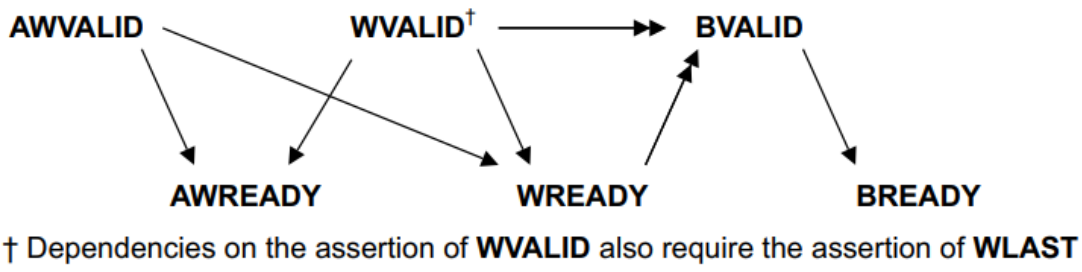


fig 4.5.2 dependencies in write channels handshake

4.6 Address structure

4.6.1 AW/AR Burst length

- ARLEN[7:0] indicates the burst length for read transfers.
- AWLEN[7:0] indicates the burst length for write transfers.
- The burst_length for AXI3 is calculated as: Burst Length = AxLEN[3:0] + 1

AXI follows specific rules for burst usage:

- For wrapping bursts, the burst length must be 2, 4, 8, or 16.
- In AXI3, a burst cannot cross a 4KB address boundary.

4.6.2 AW/AR Burst size

- ARSIZE[2:0] is used for read transfers.
- AWSIZE[2:0] is used for write transfers.

In the AXI specification, AxSIZE represents either ARSIZE or AWSIZE.

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

Fig 4.6.2 write read Brust size

"If the AXI bus is wider than the transfer size, the AXI interface must decide, based on the transfer address, which byte lanes of the data bus to utilize for each transfer."

4.6.3 Burst type

- ARBURST[1:0] is used for read transfers.
- AWBURST[1:0] is used for write transfers.

AxBURST[1:0]	Burst type
0b00	FIXED
0b01	INCR
0b10	WRAP
0b11	Reserved

Fig 4.6.3 Brust type

1. Fixed burst:

- In a fixed burst type, the address stays consistent for every transfer within the burst.
- This type is used for repetitive accesses to the same location, like when loading or emptying a peripheral FIFO application.

1. **Incrementing burst:**

- In an incrementing burst, the address for each transfer within the burst increases in increments of the previous transfer address, with the slave calculating the next address.
- The increment value of the address depends on the transfer size.

- For example, in a burst with a size of four bytes, the address for each transfer is calculated as the previous address plus four.

2. Wrapping burst:

- A wrapping burst resembles an incrementing burst, where the address for each transfer increases based on the previous transfer address.
- In a wrap type burst, the address wraps around to a lower address once it reaches a wrap boundary.
- The wrap boundary is determined by multiplying the size of each transfer in the burst by the total number of transfers in the burst.

4.7 Example of Calculate Address with INC, FIXED, WRAP

4.7.1 Burst type – INC

AWADDR = 32'h1000_0000

WLEN = 5

AWBURST = INCR

AWSIZE = 2

Decoding:

AWLEN = 5 (**burstlen** = 6)

AWBURST = INCR (Data write in incremental address locations

AWSIZE = 2 (2^AWIZE = 4 byte per Transfer)

Total Byte in Transection = burstlen*byte per transfer = 6 * 4

24 byte

Data Write Happen from 32'h1000_0000 to 32'h1000_0017

4.7.2 Burst type – FIXED

AWADDR = 32'h1000_F000

AWLEN = 4

AWBURST = FIXED

AWSIZE = 2

Decoding:

AWLEN = 4 (**burstlen** = 5)

AWBURST = FIXED (Data write at fixed address only)

AWSIZE = 2 (2^AWIZE = 4 byte per Transfer)

Total Byte in Transection = burstlen*byte per transfer = 5 * 4 =

20 byte

All Data Write Happen at 32'h1000_F000

4.7.3 Burst type – WRAP

AWADDR = 32'h1000_1010
AWLEN = 5
AWBURST = WRAP
AWSIZE = 2

Decoding:

AWLEN = 5 (**burstlen** = 6)
AWBURST = WRAP (Data write in wrapping manner)
AWSIZE = 2 (2^AWIZE = 4 byte per Transfer)
Total Byte in Transection = burstlen*byte per transfer = 6 * 4 = **24 byte**

WRAP boundaries calculations:

AWADDR % (Total Byte in Transection) = reminder

32'h1000_0010 % 24 = 8

Lower wrap address = AWADDR – 8 = 32'h1000_0008

Upper wrap address = Lower wrap address + 23 = 32’h1000_001F

Write data in memory:

1st Transefer write at **address = 32’h1000_0010**

2st Transefer write at **address = 32’h1000_0014**

3st Transefer write at **address = 32’h1000_0018**

4st Transefer write at **address = 32’h1000_001C**

5st Transefer write at **address = 32’h1000_0008 (wrap to lower addr)**

6st Transefer write at **address = 32’h1000_000C**

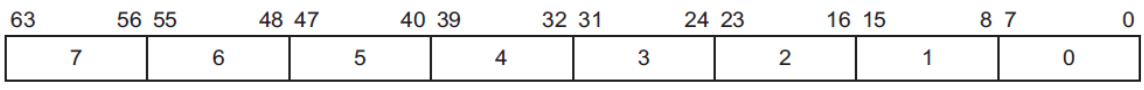
4.8 Data read and write structure

4.8.1 Write strobe

- Strobe value indicates how many bytes of data are valid in transfers. a write strobe indicates that the corresponding byte_lane of the data bus contains valid information to be updated in memory.
- The size of WSTRB varies according to the data width.
- In this AXI VIP, the write bus is 32 bits wide, so the strobe is 4 bits wide. Each bit of the strobe signal represents a valid byte in the write transfer.

WSTRB[n] corresponds to the byte lane in WDATA from bit position

(8 × n) + 7 to (8 × n).



4.8 Narrow_Transfers

- If a master initiates a transfer narrower than its data bus, WSTRB specifies which byte lanes are valid for the transfer.
- In incrementing or wrapping bursts, different byte lanes transmit data for each transfer within the burst.
- In a fixed burst, where the address remains constant, the byte lanes used also remain constant.

Example 1:

- The burst consists of five transfers.
- Starting address: 0.
- Each transfer is 8 bits wide.
- Transfers occur 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

Fig 4.8.1 Narrow_transfer

Example 2:

- The burst comprises three transfers.
- Starting address: 4.
- Each transfer is 32 bits wide.
- Transfers occur on a 64-bit bus.

Byte lane used

DATA[63:32]		1st transfer
	DATA[31:0]	2nd transfer
DATA[63:32]		3rd transfer

Fig 4.8.2 Narrow_transfer

4.9 Narrow Transfers Examples

4.9.1 Narrow INC (Without strobe Signal)

AWADDR = 32'h100 (**256**)
AWLEN = 4
AWBURST = INCR
AWSIZE = 1

Decoding:

AWLEN = 4 (**burstlen** = 5)
AWSIZE = 1 (2^AWIZE = 2 byte per Transfer)
Here burst size is smaller then WDATA BUS size (Let assume BUS size 64 - bit and Transfer size 16 - bit)

Total Byte in Transection = burstlen*byte per transfer = 5 * 2 = **10 byte**

Write data in memory:

1st Transefer:

WDATA = 64'h10203040_50607080 (Total 8 byte How many byte valid ?)
80,**70** write on address of **100** and **101 (Remain Byte Ignore)**

2st Transefer:

WDATA = 64'h10203040_50607080 (Total 8 byte How many byte valid ?)
80,**70** write on address of **100** and **101**

3st Transefer:

WDATA = 64'h11223344_55667788
Offset = 32'h101 % 8 = 2
66,**55** write on address of **102** and **103**

4st Transefer:

WDATA = 64'h11223344_55667788
Offset = 32'h106 % 8 = 6
22,**11** write on address of **106** and **107**

5st Transefer:

WDATA = 64'h10203040_50607080
Offset = 32'h108 % 8 = 0
80,70 write on address of **108** and **109**

4.9.2 Narrow (With Strobe)

AWADDR = 32'h100 (**256**)
AWLEN = 4
AWBURST = INCR
AWSIZE = 2

Decoding:

AWLEN = 4 (**burstlen** = 5)
AWSIZE = 2 (2^AWIZE = 4 byte per Transfer)

Here burst size is smaller then WDATA BUS size (Let assume BUS size 64 - bit and Transfer size 16 - bit)

Total Byte in Transection = burstlen*byte per transfer = 5 * 4 = **20 byte**

Write data in memory:

1st Transefer:

WDATA = 64'h10203040_50607080
WSTRB = 8'b0000_1111
80, **70, 60, 50** write on address of **100**, **101, 102, 103 (Remain Byte Ignore)**

2st Transefer:

WDATA = 64'h11223344_55667788
WSTRB = 8'b1111_0000
44, 33, 22, 11 write on address of **104**, **105, 106, 107**

3st Transefer:

WDATA = 64'h10203040_50607080
WSTRB = 8'b0000_1111
80, 70, 60, 50 write on address of **108, 109, 10A, 10B**

4st Transefer:

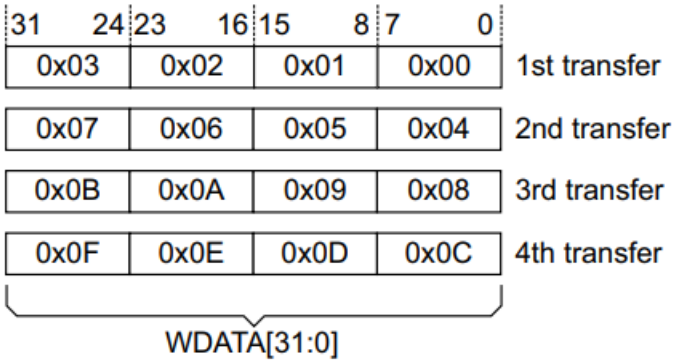
WDATA = 64'h11223344_55667788
WSTRB = 8'b1111_0000
44, 33, 22, 11 write on address of **10C, 10D, 10E, 10F**

4st Transefer:

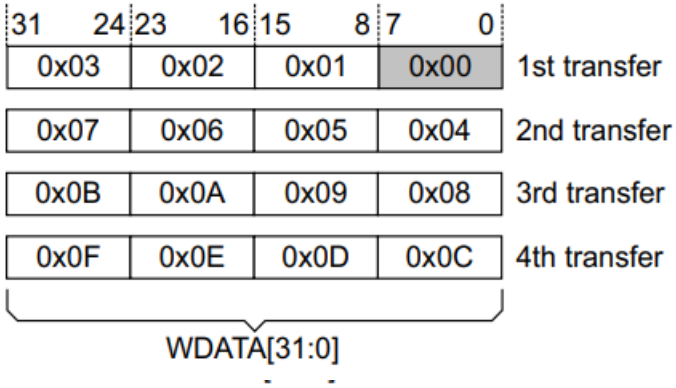
WDATA = 64'h10203040_50607080
WSTRB = 8'b1111_0000
80, 70, 60, 50 write on address of **110, 111, 112, 113**

4.10 Unaligned transfers

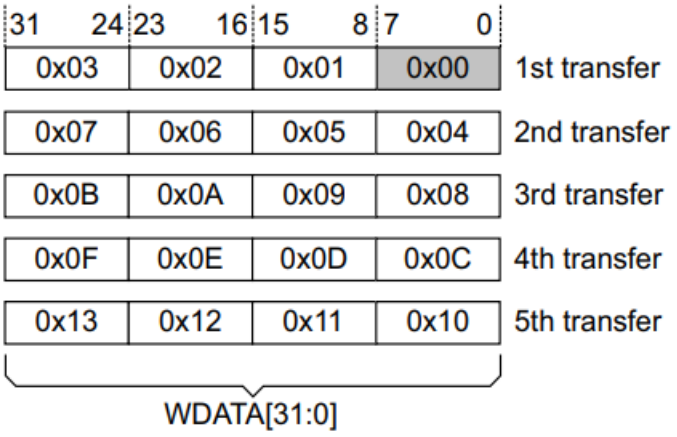
Address: 0x00
Transfer size: 32-bits
Burst type: incrementing
Burst length: 4 transfers



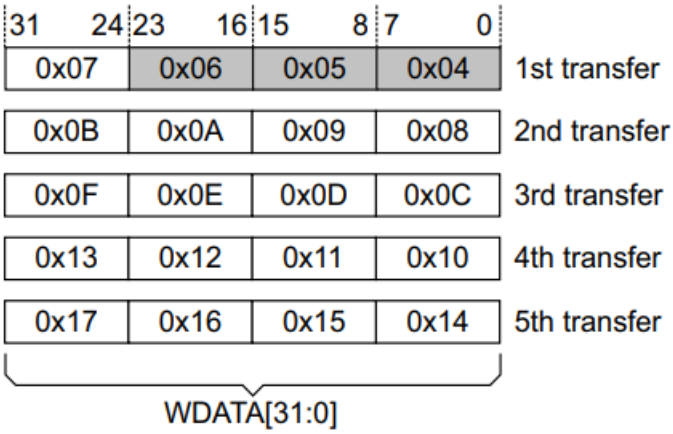
Address: 0x01
Transfer size: 32-bits
Burst type: incrementing
Burst length: 4 transfers



Address: 0x01
Transfer size: 32-bits
Burst type: incrementing
Burst length: 5 transfers



Address: 0x07
Transfer size: 32-bits
Burst type: incrementing
Burst length: 5 transfers



4.11 AXI 3.0 Write and Read response.

The AXI protocol offers response signalling for both read and write transactions:

- For read transactions, the slave responds with the read data and response information through the same read data channel.
- For write transactions, the response information is conveyed through a separate write response channel.

Response signalling:

- RRESP[1:0] for read transfers.
- BRESP[1:0] for write transfers.

RRESP[1:0] BRESP[1:0]	Response
0b00	OKAY
0b01	EXOKAY
0b10	SLVERR
0b11	DECERR

Fig 4.11 write and read response type

OKAY:

- Indicates success of normal access, showing that a regular access has been completed without issues.
- It can also indicate a failed exclusive access.

EXOKAY:

- Signifies exclusive access success, indicating that either the write and read part of an exclusive access has been successfully completed.

SLVERR:

- Indicates a slave error, used when the access reaches the slave successfully, but the slave needs to return an error condition to the master.
- Slave error can occur in cases like FIFO or buffer overrun/underrun, unsupported transfer size, attempting write access to a read-only location, timeout conditions, or accessing a disabled or powered-down function.

DECERR:

- Decode Generated error, usually issued by an interconnect component, indicating that there's no slave at the transaction address.
- If the interconnect cannot decode a slave access successfully, it must return DECERR. This may involve routing the access to a default slave, which then returns the DECERR response.
- The AXI protocol mandates that all data transfers for a transaction be completed, even if an error condition arises. Any component providing a DECERR response must adhere to this requirement.

Chapter 5

Atomic Accesses

- Atomic accesses are employed in configurations with multiple masters and slaves.
- The ARLOCK[1:0] or AWLOCK[1:0] signal indicates exclusive and locked access information.

There are three types of accesses:

1. Normal Access
2. Exclusive Access
3. Locked Access

- By default, value of ARLOCK & AWLOCK is **0** means Normal Access.

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

Fig 4.11 write and read Access type

1.1 Exclusive Access

- Exclusive access is used when we need specific access to one of the addresses of a slave.
- This mechanism allows the slave to remain accessible to other masters.

- ARLOCK[1:0] or AWLOCK[1:0] signal selects exclusive access, and RRESP[1:0] or BRESP[1:0] signal indicates the success or failure of the exclusive access with responses like OKAY or EXOKAY.

The process for an exclusive access:

i. The first master initiates an exclusive read operation on a slave address location.

ii. Later, the master tries to complete the exclusive operation by performing an exclusive write operation to the same address location.

iii. The exclusive write access of the master is determined:

- Successful if no other master has modified that location between the read and write accesses.
- Failed if another master has modified that location between the read and write accesses. In this case, the address location is not updated, and an OKAY response is generated to indicate the failure of exclusive access.

1.2 Locked Access

- The arbiter within the interconnect enforces the restriction for access.
- In a locked access, another master cannot access the same slave while the slave is busy.
- Any transaction with ARLOCK[1:0] or AWLOCK[1:0] set to indicate a locked accsess compels the interconnect to lock the transaction.
- Therefore, a locked sequence must always finish with a final transaction that does not have ARLOCK[1:0] or AWLOCK[1:0] set to indicate a locked access. This final transaction effectively removes the lock.
- When a master initiates a locked sequence of read or write transactions, it must ensure it has no other outstanding transactions waiting to complete.
- When completing a locked sequence, a master must ensure that all previous locked transactions are finished before issuing the final unlocking transaction.
- The master must ensure that all requested writes or reads within a locked sequence have the same ARID or AWID value.

5.3 AXI 3.0 Data interleaving.

- The write data interleaving feature allows a slave interface to accept interleaved write data from different transactions.
- slave specifies a write data interleaving depth, indicating if it can accept interleaved write data from masters with different transactions.
- This depth represents the number of different addresses pending in the slave interface for which write data can be supplied.
- By default, the write data interleaving depth of any interface is only one.

Restrictions for data interleaving:

- A master cannot interleave the write data of different transactions that have the same AWID.
- The order in which a slave receives the first transfer of each transaction must match the order in which it receives the addresses for the transactions.

Chapter 6

Wave Forms

6.1 Simple write

- **AWLEN** = 3
- **AWSIZE** = Halfword/ 16 bits/ 2 bytes
- **AWBURST** = Fixed

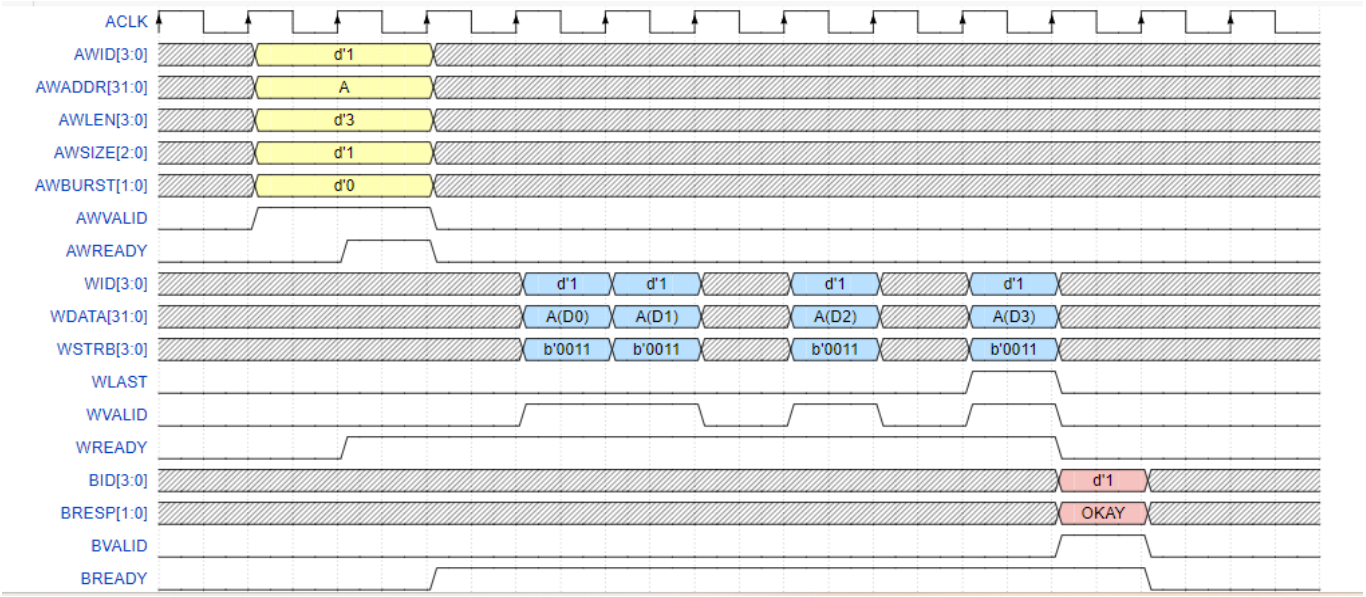


Fig 6.1 simple one write transection waveform.

6.2 Simple read

- **ARLEN** = 2
- **ARSIZE** = Halfword/ 16 bits/ 2 bytes
- **ARBURST** = Fixed

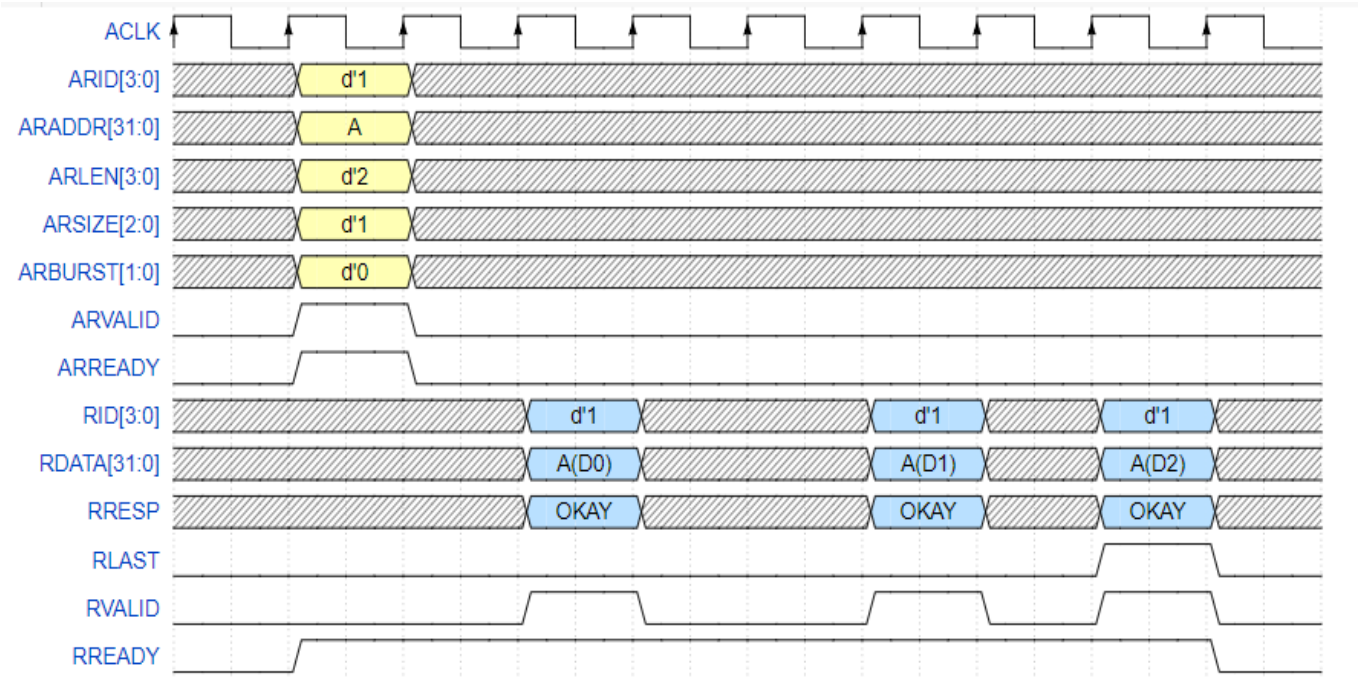


Fig 6.2 simple one read transection waveform

6.3 Burst based transfers

Transaction 1

- **AWLEN** = 3
- **AWSIZE** = Word/ 32 bits/ 4 bytes
- **AWBURST** = INCR

Transaction 2

- **AWLEN** = 1
- **AWSIZE** = Word/ 32 bits/ 4 bytes
- **AWBURST** = INCR

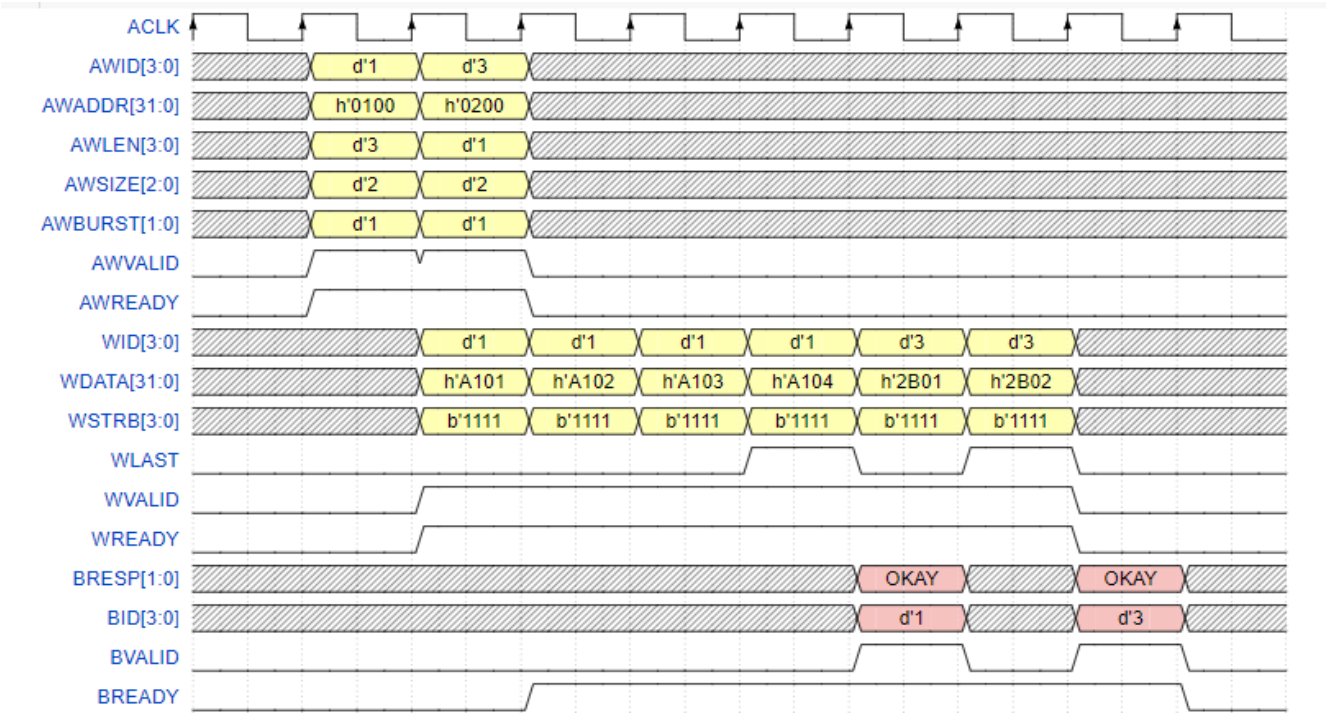


Fig 6.3 two write burst transection waveform

6.4 Interleaving transfers

Transaction 1

- **AWLEN** = 2
- **AWSIZE** = Word/ 32 bits
- **AWBURST** = Fixed

Transaction 2

- **AWLEN** = 1
- **AWSIZE** = Word/ 32 bits
- **AWBURST** = Fixed

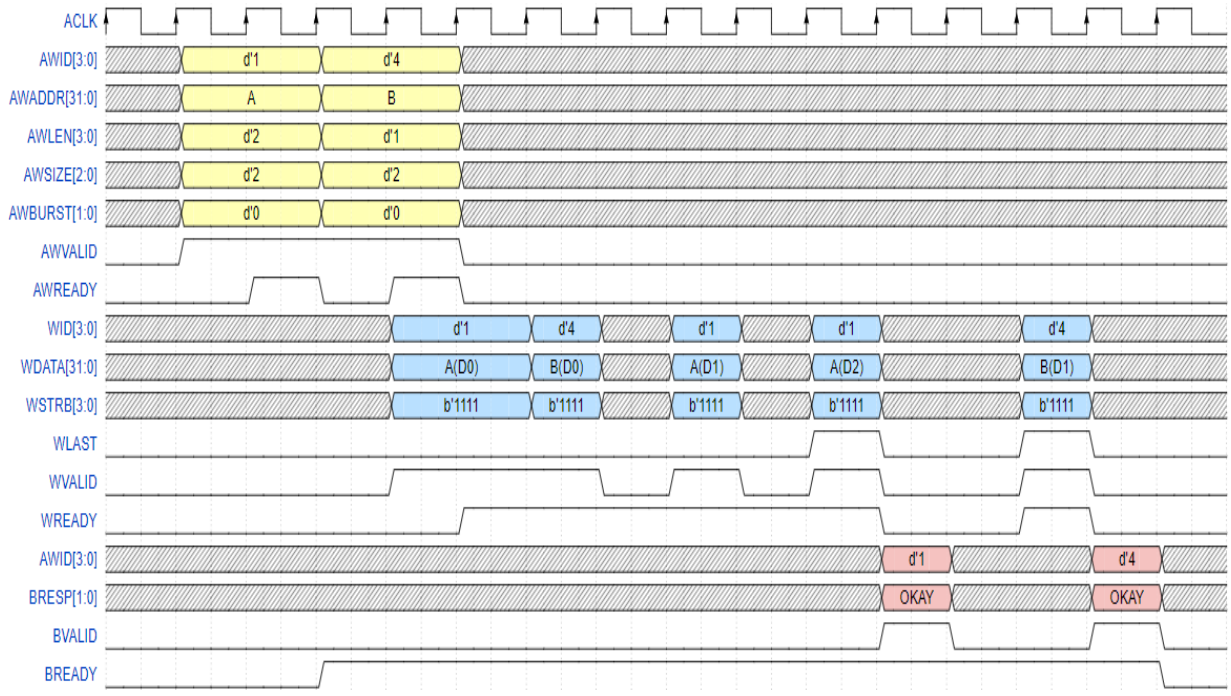


Fig 6.4 two write interleaving transection.

6.5 Out of order transfers

Transaction 1

- **ARLEN** = 1
- **ARSIZE** = Word/ 32 bits
- **ARBURST** = Fixed

Transaction 2

- **ARLEN** = 3
- **ARSIZE** = Word/ 32 bits
- **ARBURST** = Fixed

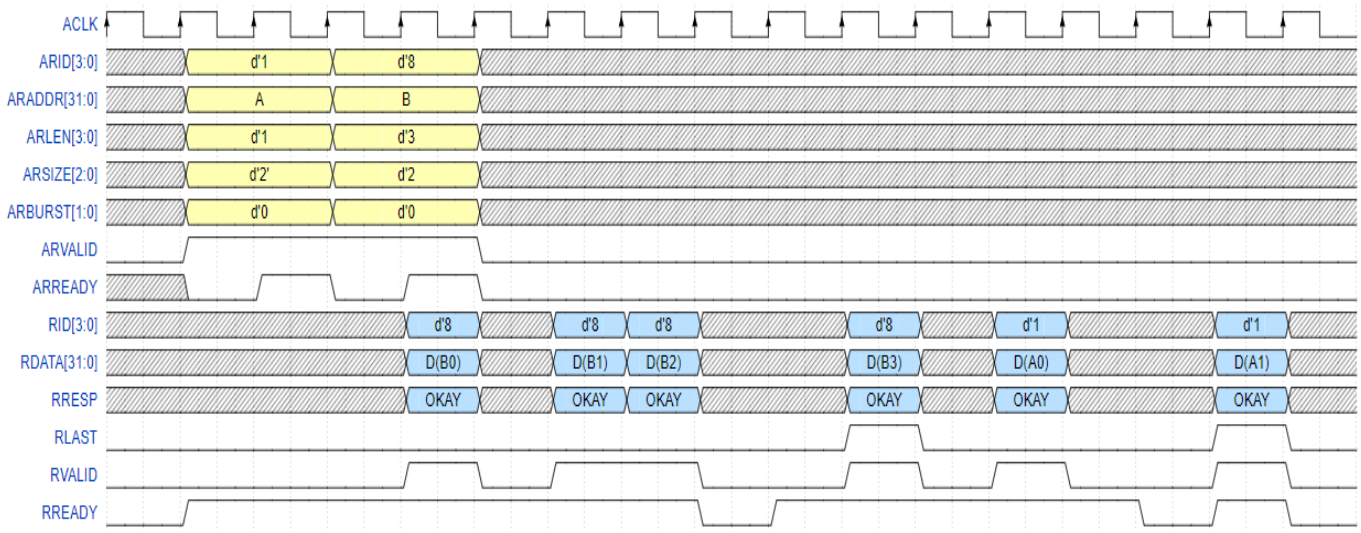


Fig 6.5 two out of order write transection.

Chapter 7

AXI VIP Verification Environment

7.1 Verification Steps.

1. Listing down features from AXI spec sheet.
2. Listing down scenario according features.
3. Developed Test plan.
4. Developed Functional Coverage Plan.
5. Implement Test-bench architecture.
6. Component coding of Test-bench.
7. Implement and Pass Sanity test case.
8. bring up Sanity test case.
9. Implement remain test cases according to Test Plan.
10. Pass regression with all AXI test cases and debugging.
11. Implement functional coverage and generating coverage results.
12. Analyse coverage results if not meet with expectation, then modify test case to get maximum coverage.

7.2 UVM Test bench Architecture.

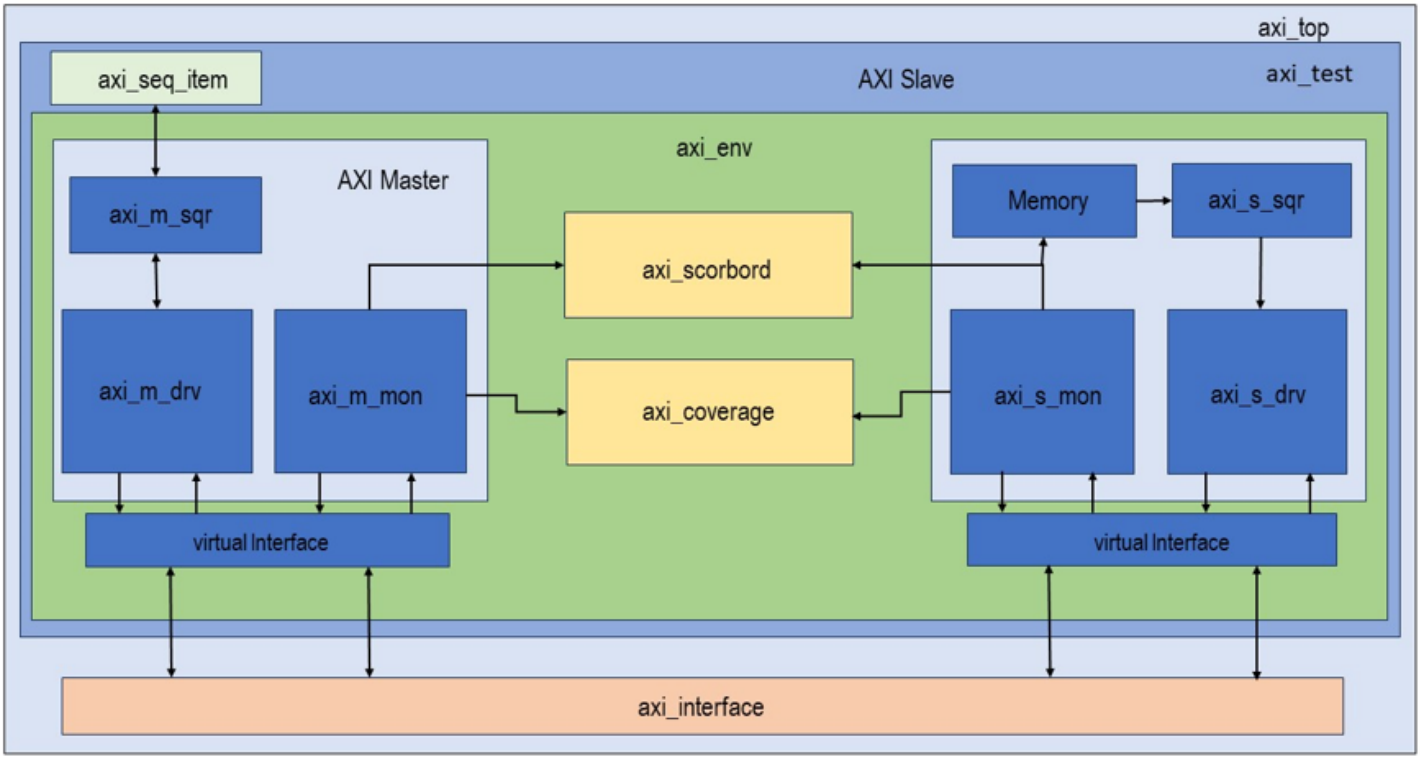


Fig 7.2 UVM Testbench Architecture.

1. axi_sequences

- Sequences are constructed to offer input to designs and check the capabilities of tests and verification IP.
- These sequences provide control over randomization of transactions and create scenarios for test cases.
- VIP includes various sequences tailored for different test cases, such as axi_out_of_order_sqn and axi_interleaving_sqn.

2. axi_master_driver

- basically, driver drive packet level transection to pin level (Interface).
- VIP have axi master driver which contain 3 driving logic of AW, AR,W,B channel logic.

3. axi_slave_driver

- VIP have slave driver to drive transection from axi memory component, its contain R,B channel driving logic.

4. axi_master_sequencer

- The role of sequencer is routes a sequence to the driver and driver and sequencer are communicate using TLM port.
- Here in VIP master driver routes master sequences to master driver.

5. axi_slave_sequencer

- The role of slave sequencer same as master sequencer, slave sequencer routes transection from axi memory component to slave driver.

6. axi_master_monitor

- Monitor is passive component used to sample/ capture signals information form interface and translate in packet format.
- In axi vip role of master monitor is sample slave signals from interface (B, R channel Information).

7. axi_slave_monitor

- The role of slave monitor same as master monitor, sample transection from master interface (AW,W,AR channels information).

8. axi_master_agent

- The role of agent is a container that holds and connects the driver, monitor, and sequencer instances.
- In axi vip role of master agent hold instance of master driver, master monitor, master sequencer and TLM ports connection.

9. axi_slave_agent

- The role of slave agent is same as master agent, in axi vip role of slave agent hold instance of slave driver, slave sequencer, slave monitor and TLM ports connections.

10. axi_scoreboard

- Scoreboard it a component that checking the expected results against the actual results, scoreboard role in axi vip is master agent transection and slave agent transection compare and give result pass or failed.

11. axi_memory

- axi memory hold write transection and generate response according to transection.

7.2 UVM Test bench components hierarchy

```
# UVM_INFO @ 0: reporter [UVMTOP] UVM testbench topology:
# ----------------------------------------------------------------------------
# Name                       Type                                  Size  Value
# ----------------------------------------------------------------------------
# uvm_test_top               axi_wr_rd_wrap_test                   -     @471
#   env                      axi_env                               -     @478
#     cvg                    axi_coverage                          -     @506
#       analysis_imp         uvm_analysis_imp                      -     @513
#     m_age                  axi_m_age                             -     @485
#       m_drv                axi_m_drv                             -     @523
#         rsp_port           uvm_analysis_port                     -     @538
#         seq_item_port      uvm_seq_item_pull_port                -     @530
#       m_mon                axi_m_mon                             -     @655
#         mmon_ap            uvm_analysis_port                     -     @669
#       m_sqr                axi_m_sqr                             -     @546
#         rsp_export         uvm_analysis_export                   -     @553
#         seq_item_export    uvm_seq_item_pull_imp                 -     @647
#         arbitration_queue  array                                 0     -
#         lock_queue         array                                 0     -
#         num_last_reqs      integral                              32    'd1
#         num_last_rsps      integral                              32    'd1
#     s_age                  axi_s_age                             -     @492
#       amem                 axi_mem                               -     @828
#         mem_ip             uvm_analysis_imp                      -     @835
#       s_drv                axi_s_drv                             -     @689
#         rsp_port           uvm_analysis_port                     -     @704
#         seq_item_port      uvm_seq_item_pull_port                -     @696
#       s_mon                axi_s_mon                             -     @821
#         smon_ap            uvm_analysis_port                     -     @845
#       s_sqr                axi_s_sqr                             -     @712
#         rsp_export         uvm_analysis_export                   -     @719
#         seq_item_export    uvm_seq_item_pull_imp                 -     @813
#         arbitration_queue  array                                 0     -
#         lock_queue         array                                 0     -
#         num_last_reqs      integral                              32    'd1
#         num_last_rsps      integral                              32    'd1
#     sb                     axi_sb                                -     @499
#       mip                  uvm_analysis_imp_axi_master_monitor   -     @869
#       sip                  uvm_analysis_imp_axi_slave_monitor    -     @877
# ----------------------------------------------------------------------------
```

Fig 7.2 UVM Testbench components hierarchy.

AXI VIP UVM Testbench

Slave components

AXI VIP UVM Testbench

Master components

Chapter 8

AXI VIP Features Result

8.1 AXI Fixed Brust type.

8.1.1 AXI Fixed Brust Write transection.

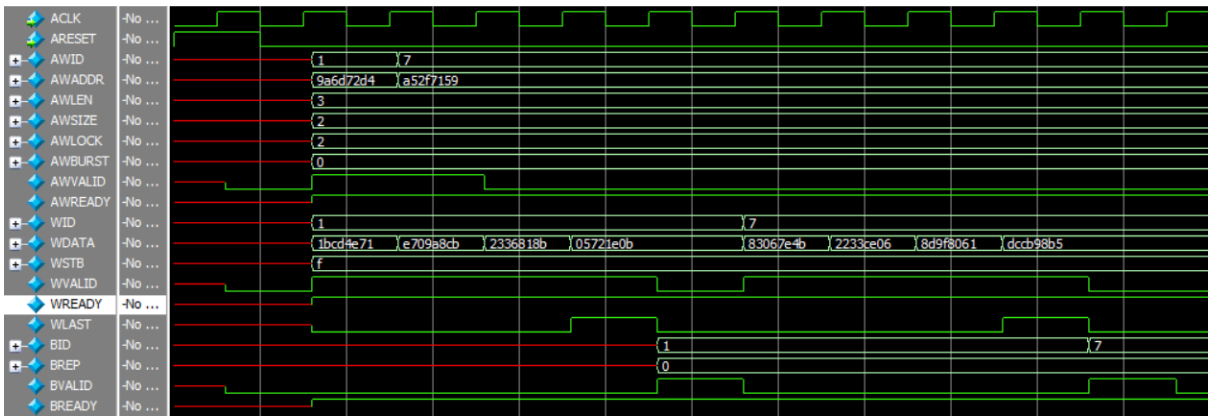


Fig 8.1.1 Write FIXED Request.

- Master Initiate transection with NORMAL Access and FIXED Burst type.
- Slave response as OKAY to indicate success of NORMAL Access.

8.1.2 AXI Fixed Brust Read transection..

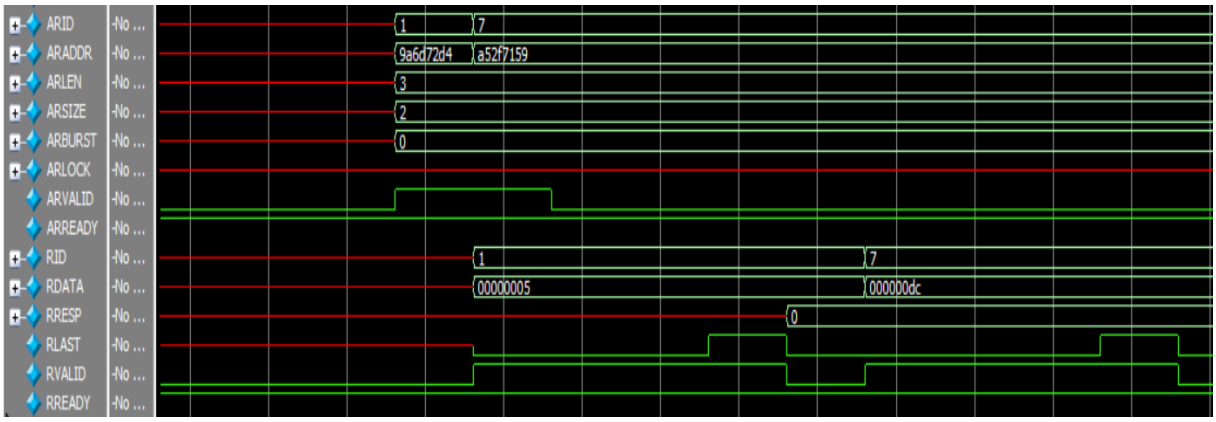


Fig 8.1.2 Read FIXED Request.

- Slave response with RDATA and RRESPONSE with NORMAL Access and FIXED Burst type.
- Slave response as OKAY to indicate success of NORMAL Access.

8.2 AXI INCR Brust type.

8.2.1 AXI INCR Brust Write transection.

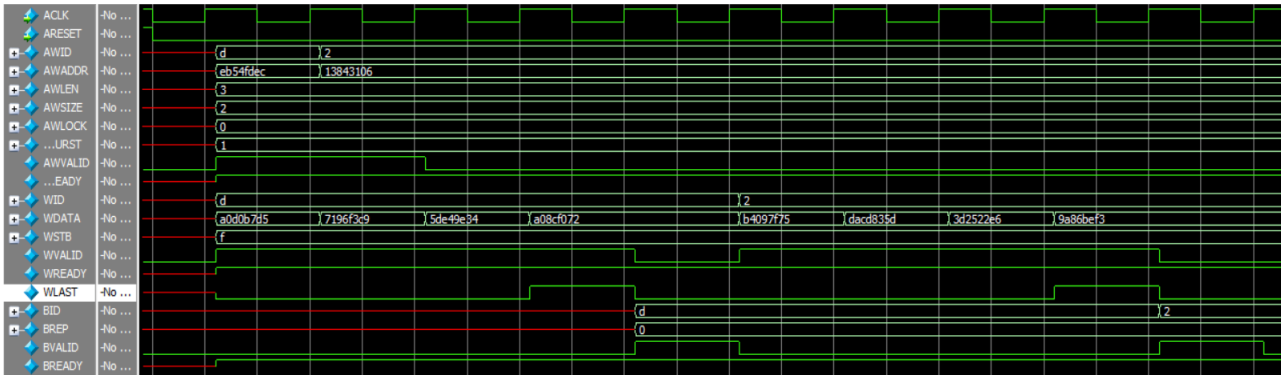


Fig 8.2.1 Write INCR Request.

- Master Initiate transection with NORMAL Access and INCR Burst type.
- Slave response as OKAY to indicate success of NORMAL Access.

8.2.2 AXI INCR Brust Read transection.

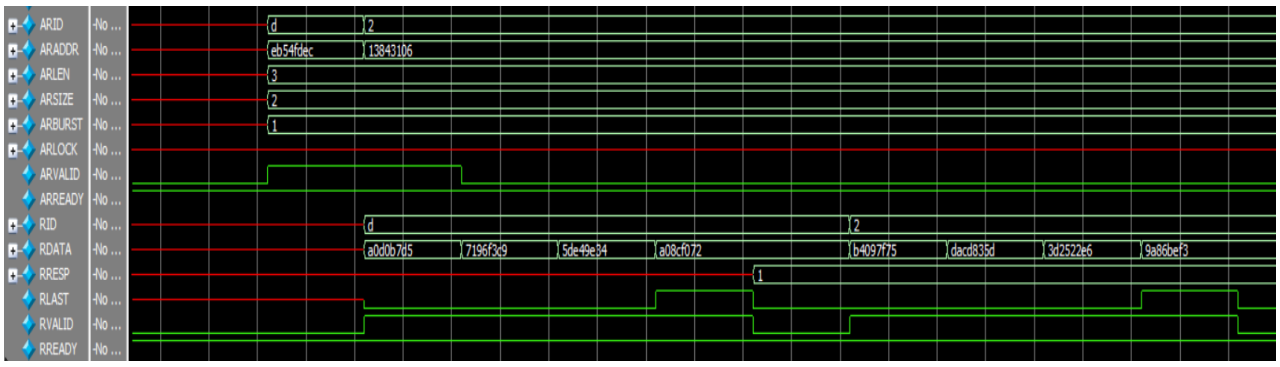


Fig 8.2.2 Read INCR Request.

- Slave response with RDATA and RRESPONSE with NORMAL Access and INCR Burst type.
- Slave response as OKAY to indicate success of NORMAL Access.

8.3 AXI WRAP Brust type.

8.3.1 AXI WRAP Brust Write transection.

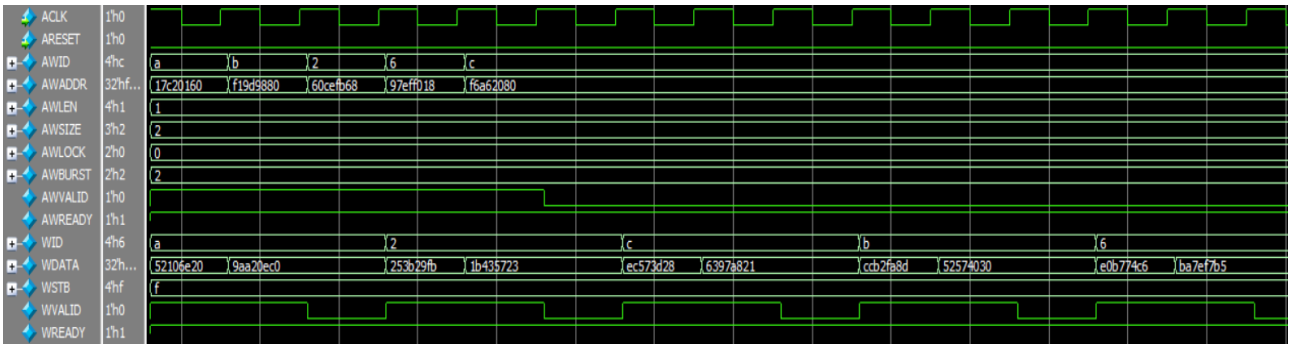


Fig 8.3.1 Write WRAP Request.

- Master Initiate transection with NORMAL Access and WRAP Burst type.
- Slave response as OKAY to indicate success of NORMAL Access.

8.3.2 AXI WRAP Brust Read transection.

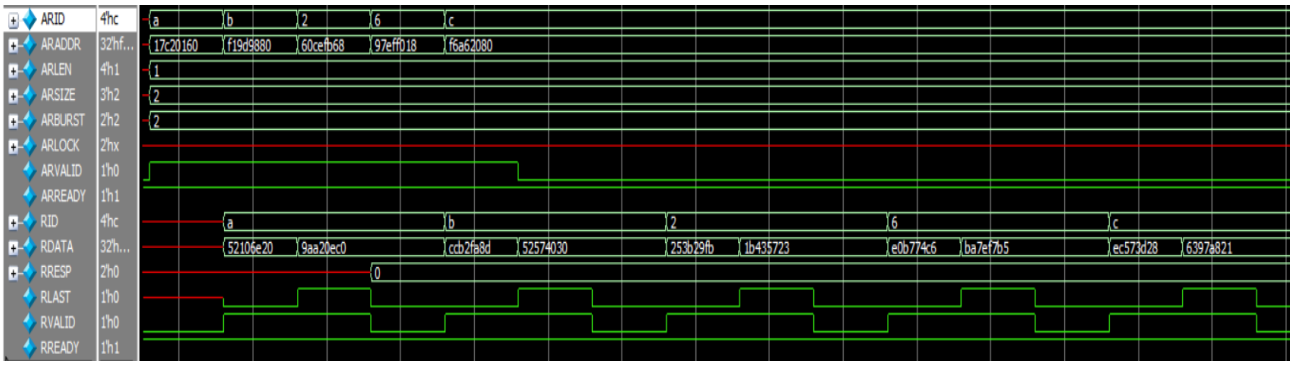


Fig 8.3.2 Write WRAP Request.

- Slave response with RDATA and RRESPONSE with NORMAL Access and WRAP Burst type.
- Slave response as OKAY to indicate success of NORMAL Access.

8.4 AXI out of order response.

8.4.1 AXI INCR Brust, out_of_order feature Write transection.

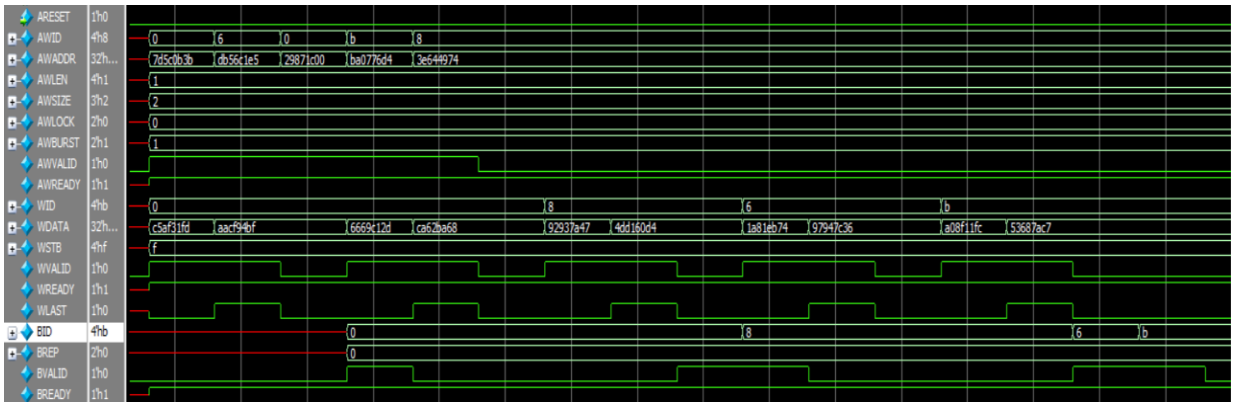


Fig 8.4.1 out of order response.

- Master Initiate transection with NORMAL Access, INCR Burst type and out of order response number which is 3 here, Number of Transactions are 5.
- Slave response as OKAY to indicate success of NORMAL Access but in out of order.

8.5 AXI Interleaving Transection.

8.5.1 AXI INCR Brust, Interleaving feature Write transection.

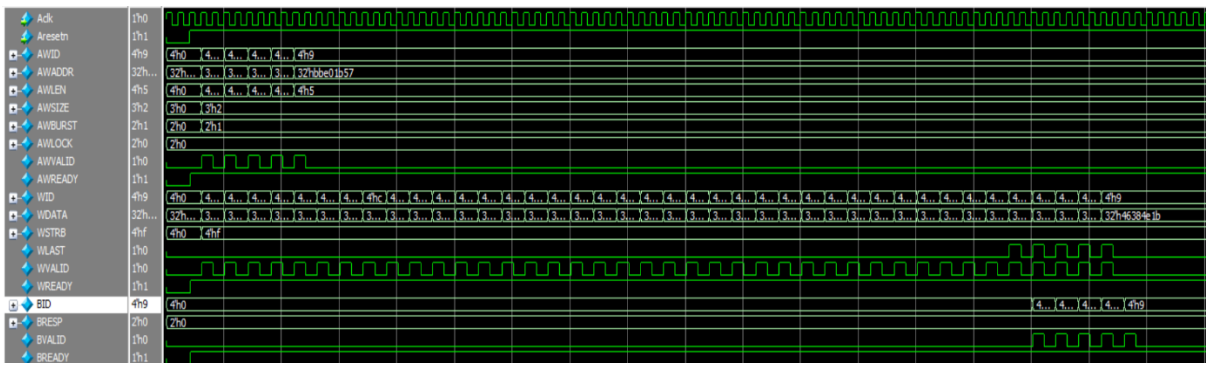


Fig 8.5.1 Interleaving transection.

- Master Initiate transection, Interleaving transection with interleaving Depth = 3, NORMAL Access, INCR Burst type.
- In write request transfers of different transection are overlapping.
- First transfer of each transection happen first then it can interleave.
- Slave response as OKAY to indicate success of NORMAL Access.

8.6 AXI outstanding addresses.

8.6.1 AXI INCR Brust, outstanding feature Write transection.

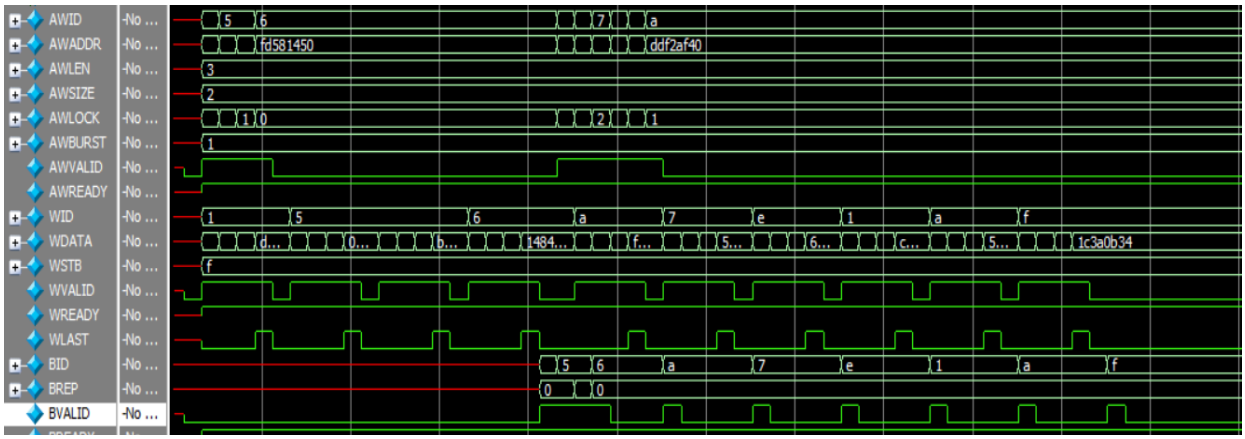


Fig 8.6.1 outstanding address.

- Master Initiate transection with NORMAL Access, WRAP Burst type with NUMBER_OF_OUTSTANDING_ADDR = 3.
- Slave response as OKAY to indicate success of NORMAL Access in order.

8.7 AXI exclusive Access.

8.7.1 AXI Exclusive Access Read Modified Write transection.

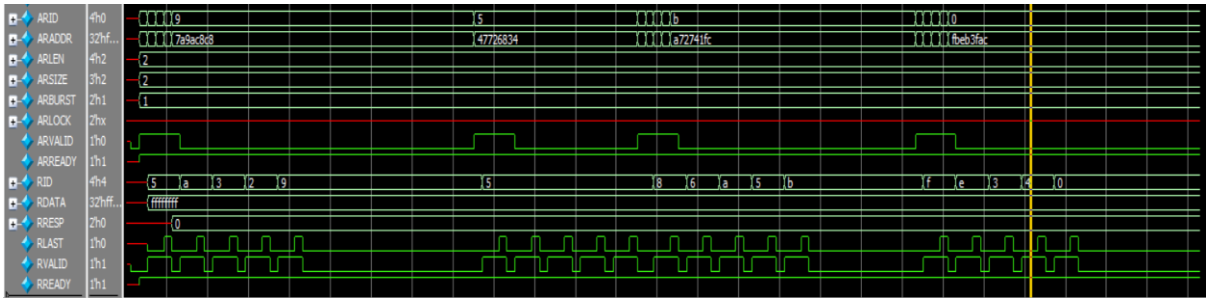


Fig 8.7.1 Exclusive read modified write transection.

- Master Initiate First Read transection with EXLUSIVE Access, WRAP Burst type.
- Slave assign one EXLUSIVE address for particular Master.
- Slave response as EXOKAY to indicate success of Exclusive Access in order.

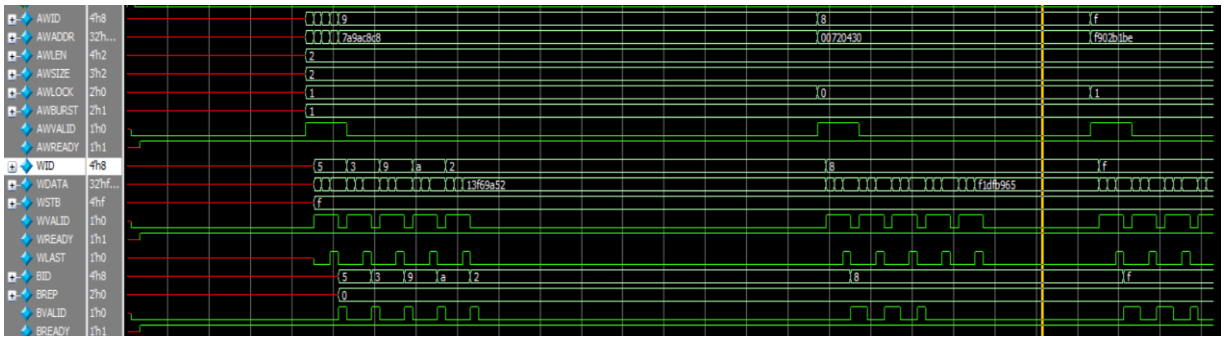


Fig 8.7.2 Exclusive read modified write transection.

- Master Initiate Second transection with EXLUSIVE Access, WRAP Burst type with write request.
- If master write happened with same on exclusive address, then slave give EXOKAY response, indicate success of Exclusive access otherwise give OKAY response to indicate fail of Exclusive access.

8.8 AXI Narrow Transection.

8.8.1 AXI Narrow transfer feature Write transection.

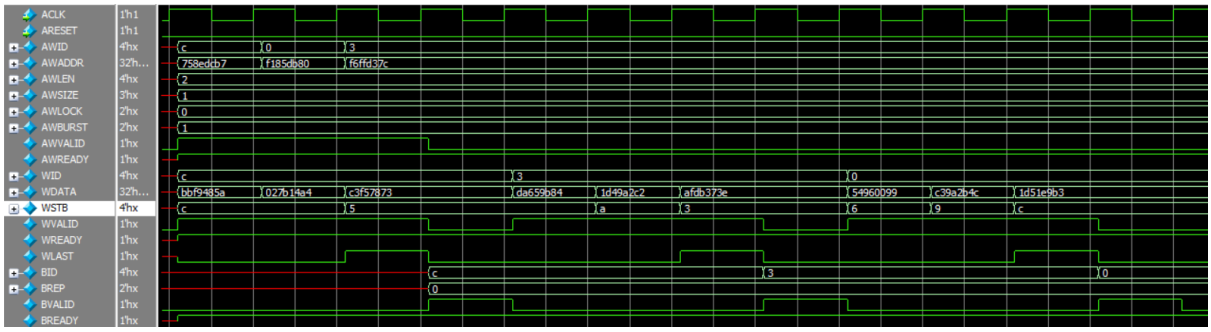


Fig 8.9.1 Exclusive read modified write transection.

- Master Initiate write transection with strobe value, WRAP Burst type and Normal Access.
- Slave accept data only which is valid only, strobe give information about how many bytes are valid in data.
- Slave response as OKAY to indicate success of NORMAL Access in order.

8.8.2 AXI Narrow transfer feature Read transection.

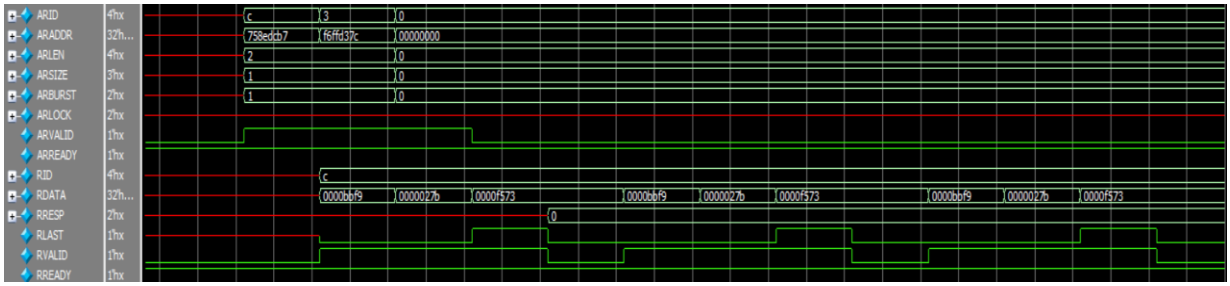


Fig 8.9.1 Exclusive read modified write transection.

- Master Initiate read transection, WRAP Burst type and Normal Access.
- Slave response as OKAY to indicate success of NORMAL Access in order.

8.10 Coverage Report.

Covergroups

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/axi_pkg/axi_coverage		99.40%							
TYPE AXI_WRITE_CG		99.10%	100	99.10%		✓	auto(1)		
CVP AXI_WRITE_CG::WR_ADDR_CP		100.00%	100	100.00...		✓			
CVP AXI_WRITE_CG::WR_ID_CP		100.00%	100	100.00...		✓			
CVP AXI_WRITE_CG::WR_BRUST_SIZE_CP		100.00%	100	100.00...		✓			
CVP AXI_WRITE_CG::WR_BRUST_TYPE_CP		100.00%	100	100.00...		✓			
CVP AXI_WRITE_CG::WR_BRUST_LEN_WRAP_CP		100.00%	100	100.00...		✓			
CVP AXI_WRITE_CG::WR_BRUST_LEN_INCR_CP		100.00%	100	100.00...		✓			
CVP AXI_WRITE_CG::WR_BRUST_LEN_FIXED_CP		93.75%	100	93.75%		✓			
TYPE AXI_READ_CG		99.10%	100	99.10%		✓	auto(1)		
CVP AXI_READ_CG::RD_ADDR_CP		100.00%	100	100.00...		✓			
CVP AXI_READ_CG::RD_ID_CP		100.00%	100	100.00...		✓			
CVP AXI_READ_CG::RD_BRUST_SIZE_CP		100.00%	100	100.00...		✓			
CVP AXI_READ_CG::RD_BRUST_TYPE_CP		100.00%	100	100.00...		✓			
CVP AXI_READ_CG::RD_BRUST_LEN_WRAP_CP		100.00%	100	100.00...		✓			
CVP AXI_READ_CG::WR_BRUST_LEN_INCR_CP		100.00%	100	100.00...		✓			
CVP AXI_READ_CG::WR_BRUST_LEN_FIXED_CP		93.75%	100	93.75%		✓			
TYPE AXI_TRANSECTION_CG		100.00%	100	100.00...		✓	auto(0)		
CVP AXI_TRANSECTION_CG::WR_RD_TRANSECTION_CP		100.00%	100	100.00...		✓			
INST \/axi_pkg::axi_coverage::AXI_TRANSECTION_CG		100.00%	100	100.00...		✓			0

- Functional coverage is process to measure of how much functionalities/features of the design have been covered by the tests.

1. **AXI WRITE Cover group.**

   - Cover point for write address.
   - Cover point for write ID.
   - Cover point for write burst type.
   - Cover point for write burst size.
   - Cover point for write supported len for FIXED burst type.
   - Cover point for write supported len for INCR burst type.
   - Cover point for write supported len for WRAP burst type.

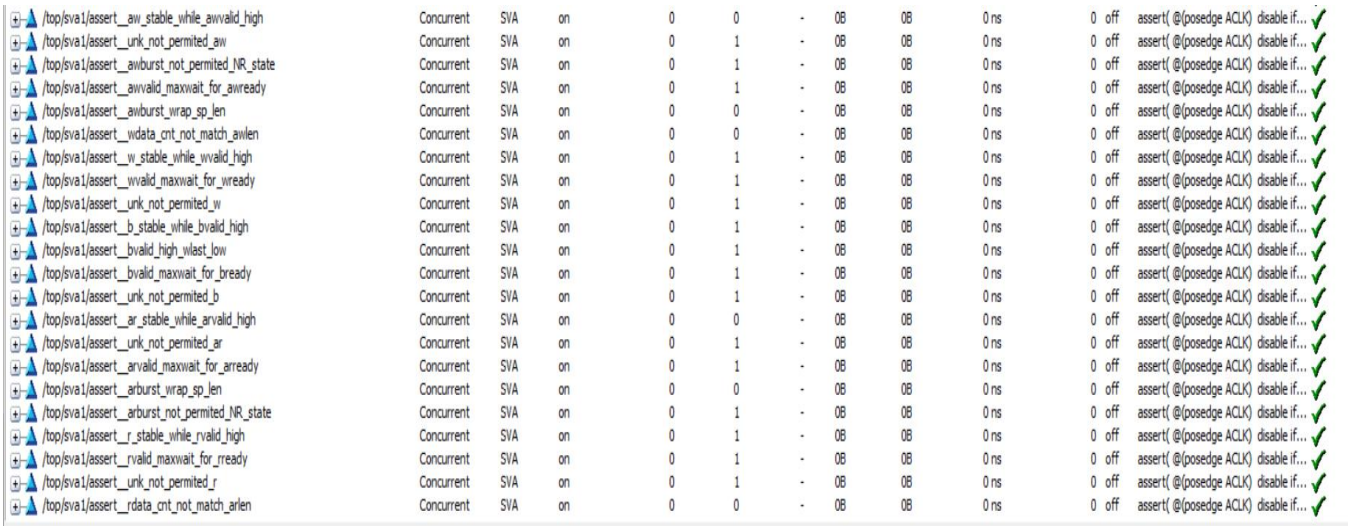
2. **AXI READ Cover group.**

   - Cover point for read address.
   - Cover point for read ID.
   - Cover point for read burst type.
   - Cover point for read burst size.
   - Cover point for read supported len for FIXED burst type.
   - Cover point for read supported len for INCR burst type.
   - Cover point for read supported len for WRAP burst type.

3. **AXI Transection cover Group**

   - Cover point for consecutive write requests.
   - Cover point for a write request followed by a read request.
   - Cover point for consecutive read requests.
   - Cover point for a read request followed by a write request.

8.11 Assertion Report.



/top/sva1/assert__aw_stable_while_awvalid_high	Concurrent	SVA	on	0	0	-	0B	0B	0 ns	0 off	assert(@(posedge ACLK) disable if...
/top/sva1/assert__unk_not_permited_aw	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ACLK) disable if...
/top/sva1/assert__awburst_not_permited_NR_state	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ACLK) disable if...
/top/sva1/assert__awvalid_maxwait_for_awready	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ACLK) disable if...
/top/sva1/assert__awburst_wrap_sp_len	Concurrent	SVA	on	0	0	-	0B	0B	0 ns	0 off	assert(@(posedge ACLK) disable if...
/top/sva1/assert__wdata_cnt_not_match_awlen	Concurrent	SVA	on	0	0	-	0B	0B	0 ns	0 off	assert(@(posedge ACLK) disable if...
/top/sva1/assert__w_stable_while_wvalid_high	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ACLK) disable if...
/top/sva1/assert__wvalid_maxwait_for_wready	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ACLK) disable if...
/top/sva1/assert__unk_not_permited_w	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ACLK) disable if...
/top/sva1/assert__b_stable_while_bvalid_high	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ACLK) disable if...
/top/sva1/assert__bvalid_high_wlast_low	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ACLK) disable if...
/top/sva1/assert__bvalid_maxwait_for_bready	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ACLK) disable if...
/top/sva1/assert__unk_not_permited_b	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ACLK) disable if...
/top/sva1/assert__ar_stable_while_arvalid_high	Concurrent	SVA	on	0	0	-	0B	0B	0 ns	0 off	assert(@(posedge ACLK) disable if...
/top/sva1/assert__unk_not_permited_ar	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ACLK) disable if...
/top/sva1/assert__arvalid_maxwait_for_arready	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ACLK) disable if...
/top/sva1/assert__arburst_wrap_sp_len	Concurrent	SVA	on	0	0	-	0B	0B	0 ns	0 off	assert(@(posedge ACLK) disable if...
/top/sva1/assert__arburst_not_permited_NR_state	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ACLK) disable if...
/top/sva1/assert__r_stable_while_rvalid_high	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ACLK) disable if...
/top/sva1/assert__rvalid_maxwait_for_rready	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ACLK) disable if...
/top/sva1/assert__unk_not_permited_r	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ACLK) disable if...
/top/sva1/assert__rdata_cnt_not_match_arlen	Concurrent	SVA	on	0	0	-	0B	0B	0 ns	0 off	assert(@(posedge ACLK) disable if...

- Assertion is process to bound a design during simulation.
- In AXI VIP development write assertion for all axi channels.

8.12 Tools Used during AXI VIP Development.

1. Synopsys VCS for simulation and coverage.
2. Synopsys Verdi for wave form analyse.
3. Questa sim VSIM for simulation and waveform analyse.

Chapter 9

Conclusion.

- AMBA AXI- 3.0 protocols back-to-back VPI is developed successfully and it can be configured and inserted into a test bench for verifying a design with the help of different test cases.
- Now days complexity of RTL and SoCs are going to increase, verification of complex RTL and SoCs are going to very difficult to handle and it take a lot of time, so there is a need a develop standard verification environment so we can reuse to verify other standard IPs and it take less time to verify, Now most of SoCs uses AXI Protocol based on chip communication
- This Project work is carried out by designing AXI Master and AXI Slave with the help of System Verilog, Universal Verification Methodology (UVM) developed Verification IP(VIP) for AXI protocol.
- AXI 3.0 VIP is Back-to-Back VIP that mean, there are no any RTL are Present AXI Master and AXI slave are integrated back-to-back and communicate with the help of request and response based.
- This project verifies features of AXI which include FIXED Burst type, INCR Burst type, WAP Brust type, out of order transection completion, multiple outstanding address, interleaving transfers of different transection, narrow transfer.
- All the components and objects in the verification environment are developed using UVM which helps to reduce time to develop testbench and reduce verification time.
- 99.40 % of functional coverage was achieved by this project with regression pass to all of the test cases of AXI VIP.

Chapter 10

Reference

1. AMBA AXI and ACE Protocol Specification AXI3, AXI4, and AXI4-Lite ACE and ACE-Lite
2. Mahesh G, Shaktivel SM. Functional Verification of the Axi2Ocp Bridge using System Verilog and effective bus utilization calculation for AMBA AXI 3.0 Protocol. IEEE Sponsored 2nd International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS). 2015. 2. Mahesh G, Shaktivel SM. Verification of Memory Transactions in AXI Protocol using System Verilog approach. IEEE ICCSP Conference. 2015.
3. Naidu KJ, Srikanth M. Design and Verification of Slave block in Ethernet Management Interface using UVM. Indian Journal of Science and Technology. 2016 Feb; 9(5):1-7.
4. Universal Verification Methodology (UVM) 1.1 User'Guide, May,2011. 9. [9] UVM Cookbook, Mentor Graphics, Sept, 20
5. Mahesh G, Shaktivel SM. Verification of Memory Transactions in AXI Protocol using System Verilog approach. IEEE ICCSP Conference. 2015.
6. Chen CH, Iu JC, Huang II. A Synthesizable AXI Protocol Checker for SoC Integration. IEEE Transl, ISOCC. 2010; 8:103-6
7. AMBA AXI-4 Specification, Copyright ARM Limited. http://www.gstitt.ece.ufl.edu/courses/eel4720_5721/labs/refs/AXI4_specification.pdf.
8. https://developer.arm.com/documentation/102202/latest/AXI-protocol-overview

- **plagiarism report**