“AXI-Lite Design and Verification using Verilog”

Major Project Report

Submitted in Partial Fulfillment of the
Requirements for the Degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

By

Kashyap Adodariya
(17BEC004)

Department of Electronics and Communication Engineering,
Institute of Technology,
Nirma University,
Ahmedabad 382 481

Nirma University,
Ahmedabad 382 481

CERTIFICATE

This is to certify that the Major Project Report entitled "AXI-Lite Design and Verification" submitted by Mr. Kashyap Adodariya towards the partial fulfillment of the requirements for the award of degree in Bachelor of Technology in the field of Electronics & Communication Engineering of Nirma University is the record of work carried out by him under our supervision and guidance. The work submitted has in our opinion reached a level required for being accepted for examination. The results embodied in this major project work to the best of our knowledge have not been submitted to any other University or Institution for award of any degree or diploma.

Date: 12 May 2021

Industry – Guide
Mr. Hitesh Patel
Senior Verification
Engineer
4, Aryan Park
Complex, Near Shilaj
Railway Crossing,
Thaltej, Ahmedabad.

Institute - Guide
Dr. Usha Mehta
Professor
Institute of Technology,
Nirma University,
Ahmedabad 382 481

Head of Department
Department of Electronics &
Communication Engineering,
Institute of Technology,
Nirma University,
Ahmedabad 382 481

Director
Institute of Technology,
Nirma University,
Ahmedabad 382 481

Undertaking for Originality of the Work

I, Kashyap Adodariya, 17BEC004 , give undertaking that the Major Project entitled "AXI-Lite Design and Verification" submitted by me, towards the partial fulfillment of the requirements for the degree of Bachelor of Technology in Electronics and Communication of Nirma University, Ahmedabad 382 481, is the original work carried out by me and I give assurance that no attempt of plagiarism has been made. I understand that in the event of any similarity found subsequently with any other published work or any project report elsewhere; it will result in severe disciplinary action.

[illegible]

Signature of the Student

Date: 12 May 2021

Place: Ahmedabad

Endorsed by:

(Signature of Guide)

Acknowledgement

A journey is easier when you travel together. Interdependence is certainly more valuable than independence. This project is successful due to the help of many people we would like to thank them for their help, support and time.

We would like to take this opportunity to thank all the people, involved directly or indirectly in our project, who encouraged us and appreciated us throughout the project.

We would like to express our immense gratitude and sincere thanks to our guide Dr. Usha mehta and Dr. Vaishali Dhare for their excellent guidance, invaluable suggestions and continuous encouragement at all the stages of our project. Also, our regards and thanks to Prof. Dipesh Panchal for their support and suggestions.

We would also like to thank Director, Institute of Technology, Nirma University Dr. Dhaval Pujara, Head of Department of Electronics and Communication Engineering for giving us this opportunity to present ourselves and having extreme faith in us to do justice to the work assigned to us.

Kashyap Adodariya (17BEC004)

Abstract

The (AMBA) transport protocols are a bunch of interconnecting info from ARM that normalizes on-chip correspondence elements between completely different utilitarian squares (or IP) for building elite SOC plans. These plans unremarkably have a minimum of one micro chip or microcontrollers aboard a couple of different components internal memory or outside memory connect, DSP, DMA, gas pedals, and different peripherals like USB, UART, PCIe, I2C, etc all incorporated on a solitary chip. The initial introduce learning AMBA conventions is to understand wherever exactly these numerous conventions are utilized, how these developed, and the way all of them fits into a SOC design. the subsequent graph delineates a customary AMBA primarily based SOC arrange that utilizes the AHB or ASB conventions for prime transmission capability interconnect and an APB convention for low transfer speed fringe interconnects.

In this Project, I was implemented a high-performance AXI-LITE Protocol using Verilog. That whole project is divided into two parts, first is AXI-Lite master and slave design. The second part was testbench to evaluate master and slave design components. After that also implement a self-checking testbench to ensure all requirements are full fill for applying different input vectors. For checking different scenarios and test conditions I wrote a shell script for compiling required modules.

INDEX

Chapter No.	Title	Page No.
	Acknowledgement	**4**
	Abstract	**5**
	Index	**6**
	List of Figures	**7**
	Nomenclature	**8**
1	**Introduction**	**9**
	1.1 Basic idea	9
	1.2 Motivation	10
	1.3 Objective	11
	1.4 Theory	11
	1.5 Scope of the project	13
2	**Literature Review**	**15**
	2.1 Proposal	15
3	**Software Design**	**18**
4	**Results**	**20**
5	**Conclusions**	**23**
	References (as per IEEE format)	**24**
	Appendix	

LIST OF FIGURES

Figure No.	Title	Page No.
1.1	Channel connections between master and slave	9
1.2	Evolution of AMBA architectures	12
1.3	Different types of signals	13
1.4	Read transfers between read data and read address channels	14
1.5	Write transfers between write data and write address channels	14
2.1	Name of signals used	15
2.2	Read Transactions	15
2.3	Write Transactions	17
3.1	Data flow in signals	18
3.2	Test-plan for verification of master and slave connection as well as individual functionality	19
4.1	Address write channel with required signals	20
4.2	Data write channel simulation	20
4.3	Write Response channel simulation	21
4.4	Master and slave connect back to back	21
4.5	Self-checking test-bench results	22

NOMENCLATURE

Subscripts

clk	Clock
rst	Reset

Abbreviations

DMA	Direct Memory Access
AXI	Advanced eXtensible Interface
IP	Intellectual Property
APB	Advanced Peripheral Bus
AHB	Advanced High Performance Bus
DUT	Device Under Test
QoS	Quality of Service
SoC	System on Chip

Chapter 1: Introduction

The quintessence of the AXI convention is that it gives a structure to how various squares inside each chip speak with one another. It offers a strategy prior to anything is sent, so the correspondence is clear and continuous. That way, various parts can converse with one another without stepping on one another.

1.1 Basic idea

The methodology for the AXI convention is as per the following:

- Master and slave must "handshake" to affirm substantial signs
- Transmission of management signal ought to be in independent stages
- Separate channels for transmission of signals
- Ceaseless exchange might be cultivated through burst-type correspondence

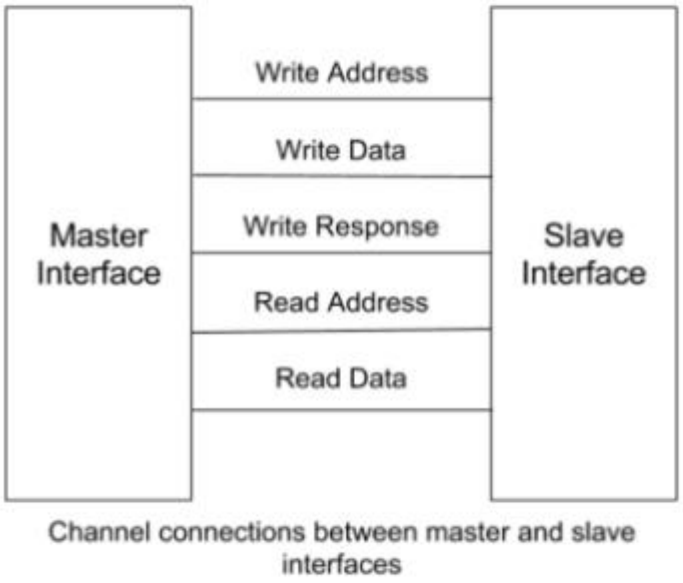


Fig 1.1 : Connection between Master and Slave

1.2 Motivation

More and more quantity of functional blocks (IP) incorporating into SOC plans, the usual transport conventions (AHB/ASB) began hitting impediments sooner and in the year 2003 , the new update of AMBA 3 acquainted a point with point availability convention—AXI (Advanced Extensible Interface). Further in 2010, a changed form in positive direction was introduced—AXI 4. The below chart displays this development of conventions alongside the SOC configuration patterns in industry.

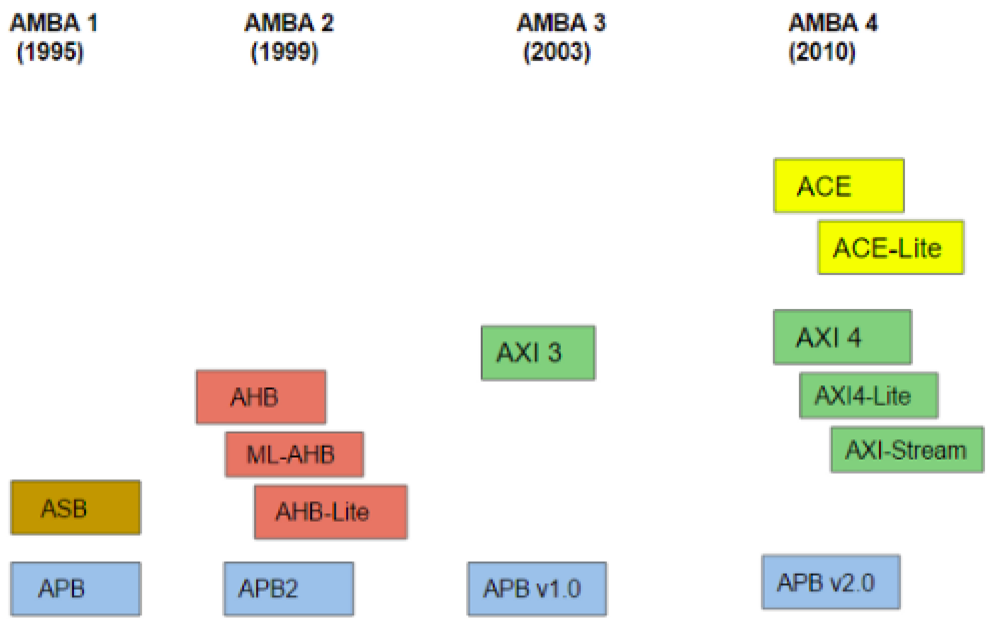


Fig 1.2 Evolution of AMBA architecture

The convention utilized by numerous SoC today is AXI, and is important for the ARM AMBA particular. It is particularly pervasive in Xilinx's Zynq gadgets, giving the interface between the preparing framework and programmable rationale segments of the chip.

1.3 Objective

The determinations of the convention are very basic, and are summed up underneath:

- Prior to transmission of any control signal/address/information, both expert and slave should expand their "hand" for a handshake by means of prepared and legitimate signs.
- To design an AXI-lite master and slave.
- Separate stages exist for transmission of control signal/address and information. All five channel design.
- Avoid signal dependence with other channels.
- Try to implement memory-based slave for write and read.
- Check basic functionality in waveforms.

1.4 Theory

The convention essentially creates the standards about how various modules on a chip speak with one another, essentially it requires a handshake-like technique prior to all transmissions. Having a convention, for example, this permits a genuine "framework" as opposed to an "assortment" of modules to be set up as the convention interfaces and gives a viable medium to move information between the current segments on the chip.

To go more inside and out, the interface works by setting up correspondence among master and slave gadgets. Among these two gadgets (or more if utilizing an AXI Core IP) exists five separate channels: Read Address, Write Address, Read Data, Write Data, and Write Response. Each channel has its own special signals just as comparable signs existing among every one of the five. The substantial and prepared signs exist for each channel as they take into consideration the handshake cycle to happen for each channel. For communicating any sign (address/information/reaction/and so forth) the significant channel source gives a functioning legitimate sign and a similar channel's objective should give a functioning prepared sign. After the two signs are dynamic, transmission might happen from that channel. As expressed over, the transmission of control signals/address and information are done in discrete stages, and subsequently a location should consistently be moved between gadgets before the handshake interaction can happen for the comparing data move. On account of composing data, the response channel is utilized toward the fulfillment of the data move.

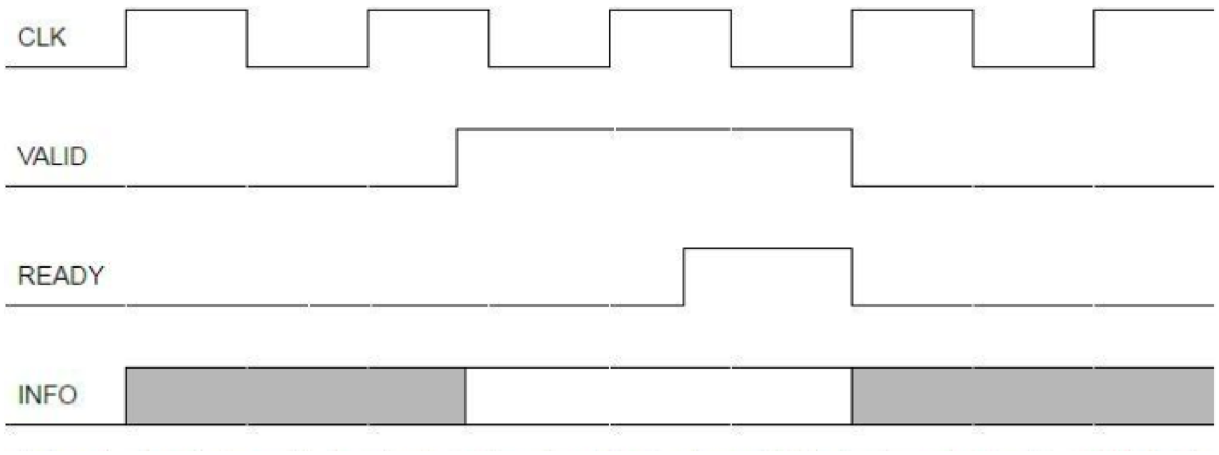


Fig 1.3 : Different types of signals

There it is. The convention is that simple! Obviously there are extra alternatives that the convention gives that up the intricacy fairly, like burst move, QoS, Protections, and others. These alternatives are essentially additional signs existing on the various channels that take into account extra usefulness, for general use nonetheless, the above portrayal conveys the idea on how this interface by and large functions.

By working with the master and slave gadgets, the AXI convention works across five tends to that incorporate peruse and compose address, peruse and compose information, and compose reaction. Since each channel has its own one of a kind sign, it can send the handshake reaction continuously so it tends to be gotten and placed into request. That way, the channel that has need will be reacted to first, etc. The source should give a legitimate sign and one that gets an appropriate reaction from the beneficiary.

By having the transmission acted in independent stages, it permits the exchange of data to be acted in a precise way. This implies that a handshake or arrangement is arrived from the outset, at that point the data is moved from the source to the beneficiary. Also, that is the way the AXI convention attempts to move data between various sources without obstruction.

With such countless gadgets utilizing the AXI convention, it makes it simpler to interface various gadgets into a similar center since need is set up. By finding out about how the interface functions, you can set the needs for clear correspondence to happen.

1.5 Scope of the Project

The AXI convention is burst-based and characterizes the accompanying free exchange channels:

• Read address

• Read data

• Write address

• Write data

• Write response.

An address channel conveys control data that portrays the idea of the information to be moved. The data is moved between master and slave utilizing for the same reason:

• A write data channel to move information from the expert to the slave. In a compose exchange, the slave utilizes the compose reaction channel to flag the fulfillment of the exchange to the master.

• A read data channel to move data from the slave to the expert.

The AXI convention:

• Licenses address data to be given in front of the real data transfer.

• upholds numerous exceptional exchanges

• upholds faulty fruition of exchanges.

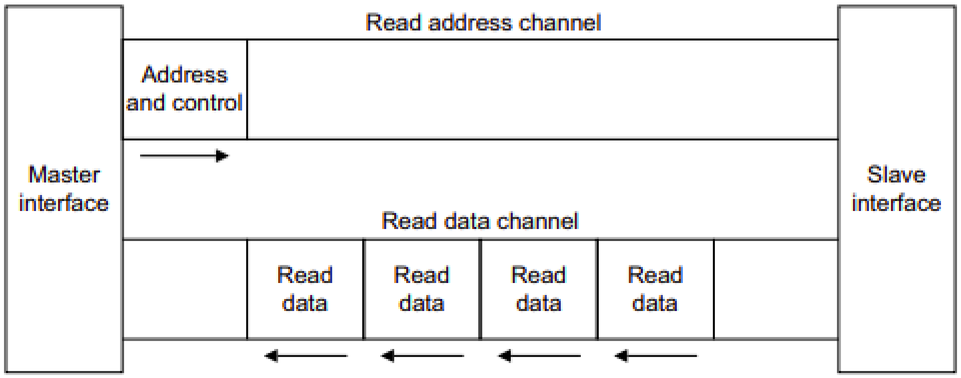


Fig 1.4: Read transfers between read data and read address channels

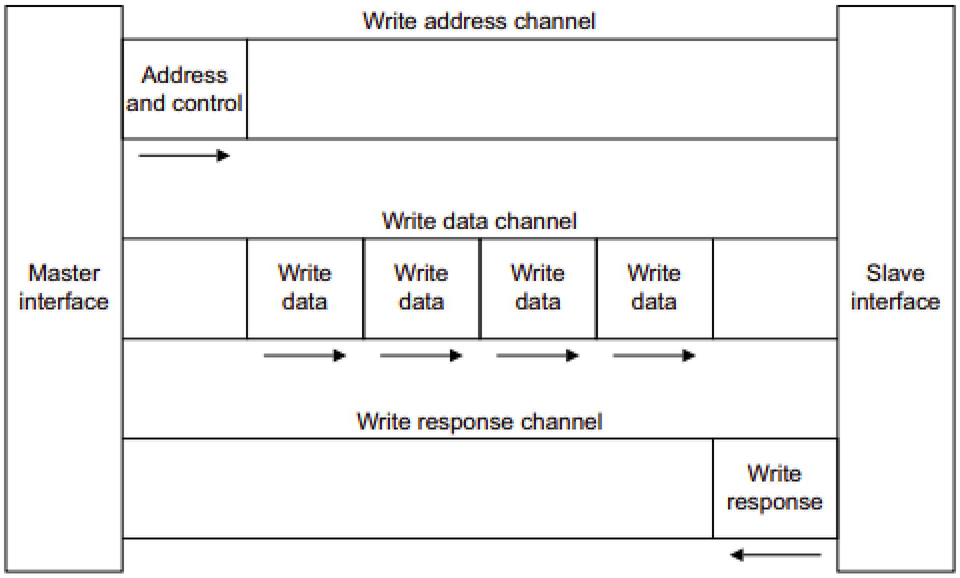


Fig 1.5: Write transfers between write data and write address channels

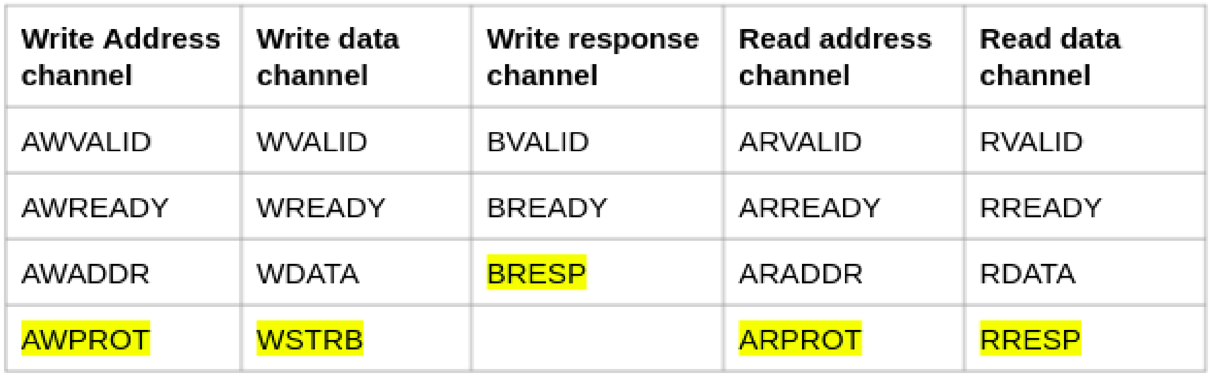
Every one of the autonomous channels comprises a bunch of data signals and VALID and READY signals that give a two-way handshake component. The data source utilizes the VALID sign to show when legitimate address, information or control data is accessible on the channel. The objective uses the READY sign to show when it can acknowledge the data. Both the read information channel and the compose information channel additionally incorporate a LAST sign to demonstrate the exchange of the last information thing in an exchange.

Chapter 2

In this chapter, I discussed our proposal based on what we do in the AXI-Lite protocol. Apart from that we also do some assumptions based on AXI-Lite guided [2].

2.1 Proposal

For implementing AXI-Lite, we used all five channel basic functionality that was referred to below signals.



Write Address channel	Write data channel	Write response channel	Read address channel	Read data channel
AWVALID	WVALID	BVALID	ARVALID	RVALID
AWREADY	WREADY	BREADY	ARREADY	RREADY
AWADDR	WDATA	BRESP	ARADDR	RDATA
AWPROT	WSTRB		ARPROT	RRESP

Figure 2.1: Signals Used. The yellow color shows that not used signal

Various AXI signals were related to every one of the five channels. To see how a to interconnect handles these signs, a more intensive glance at a basic AXI exchange is required which will be depicted subsequently.

Read Transactions

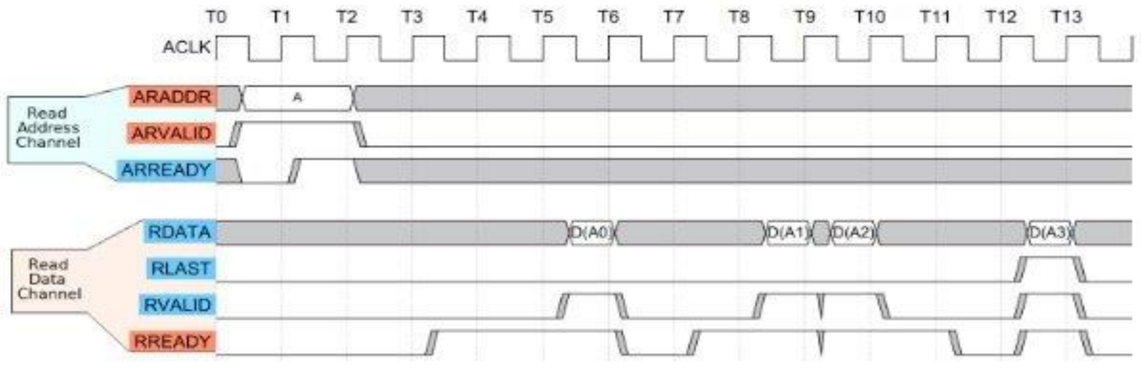


Figure 2.2: Show Read transaction. Here avoid RLAST signal

The above figure2.2 shows a planning outline of a reading exchange between an AXI expert and slave. Note that a portion of the signs has been precluded for lucidity. The consumed orange addresses an expert controlled sign, while the blue is a slave-controlled sign.

To get the exchange going, the expert places the slave's location on the ARADDR line and affirms that there is a substantial location (ARVALID). Following time T1, the slave states the prepared sign (ARREADY). Recollect the wellspring of information affirms the substantial sign when data is free, while the recipient declares the prepared sign when it can burn-through that data. For an exchange to happen both READY and VALID should be attested. The entirety of this occurs on the read address channel, with the location move finishing on the rising edge of time T2.

From here, the remainder of the exchange happens on the read information channel. At the point when the expert is prepared for information it affirms its RREADY signal. The slave at that point places information on the RDATA line and declares that there is substantial information (RVALID). For this situation, the slave is the source and the expert is the recipient.

Review that VALID and READY can be stated in any request inasmuch as VALID doesn't rely upon READY. This read addresses a solitary burst exchange comprised of 4 beats or information moves. Notice the slave declares RLAST when the last beat is moved.

Write Transactions

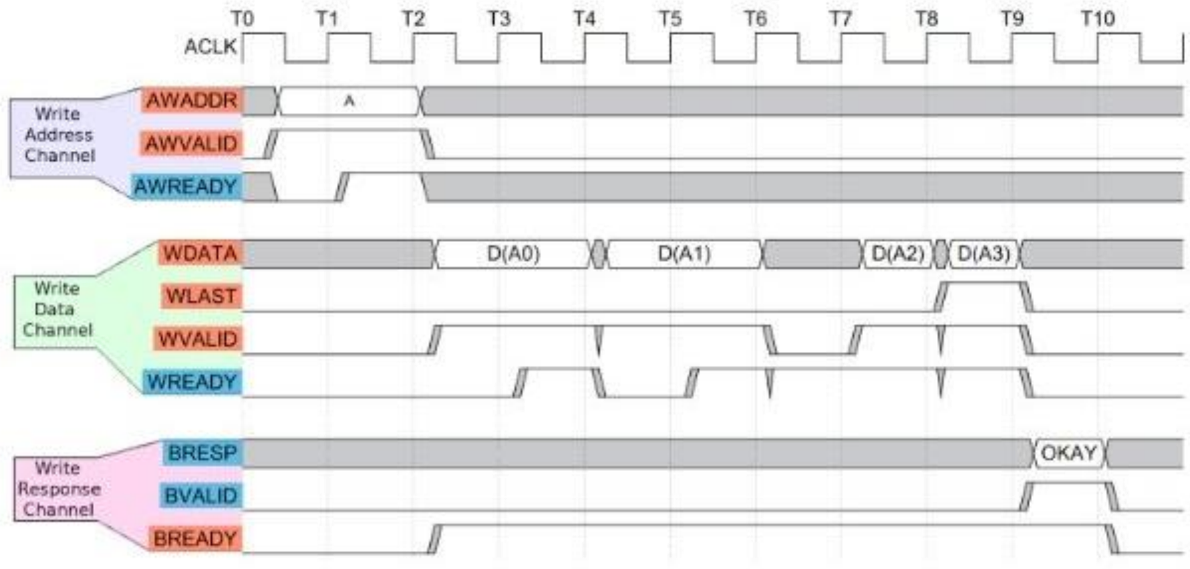


Figure2.3: shows a planning chart of an AXI write exchange

The above figure 2.3 shows a planning chart of an AXI write exchange. The tending to stage is like a read. An expert places a location on the AWADDR line and attests a substantial sign. The slave affirms that it's prepared to get the location and the location is moved.

Then, on the Write Data Channel, the expert spots information on the transport and affirms the legitimate sign (WVALID). At the point when the slave is prepared, it attests WREADY and information move starts. This exchange is again 4 beats for a solitary burst.

The expert attests the WLAST when the last beat of information has been moved. As opposed to peruses, composes incorporate a Write Response Channel where the slave can affirm that the compose exchange has finished effectively.

Chapter 3: Software Design

This chapter mainly discusses the implementation approach and verification Test Plan.

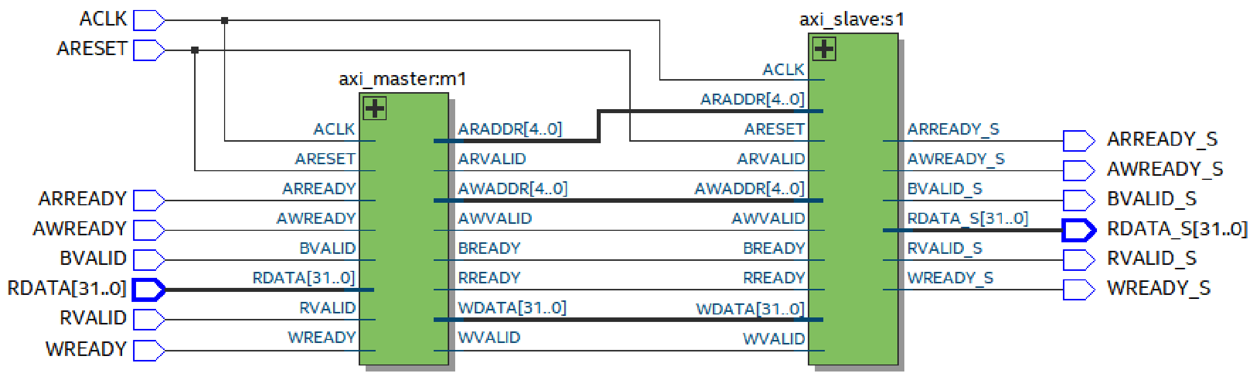


Figure 3.1: Flow of signal. Back-to-back connection master and slave.

The above figure shows that back-to-back connection of master and slave design modules in that ACLK and ARESET are common for both modules. Axi_master:m1 having all the required logic for update the channel based on the availability of information. Axi_slave:s1 module having the logic of accepting channel information based on handshaking signals. so that in the slave module take care of all dependencies which followed a handshaking signal. Apart from that slaves also respond to all valid signals based on slave current status. Slave have also one small mapped memory to store incoming data and replay when read channel requested.

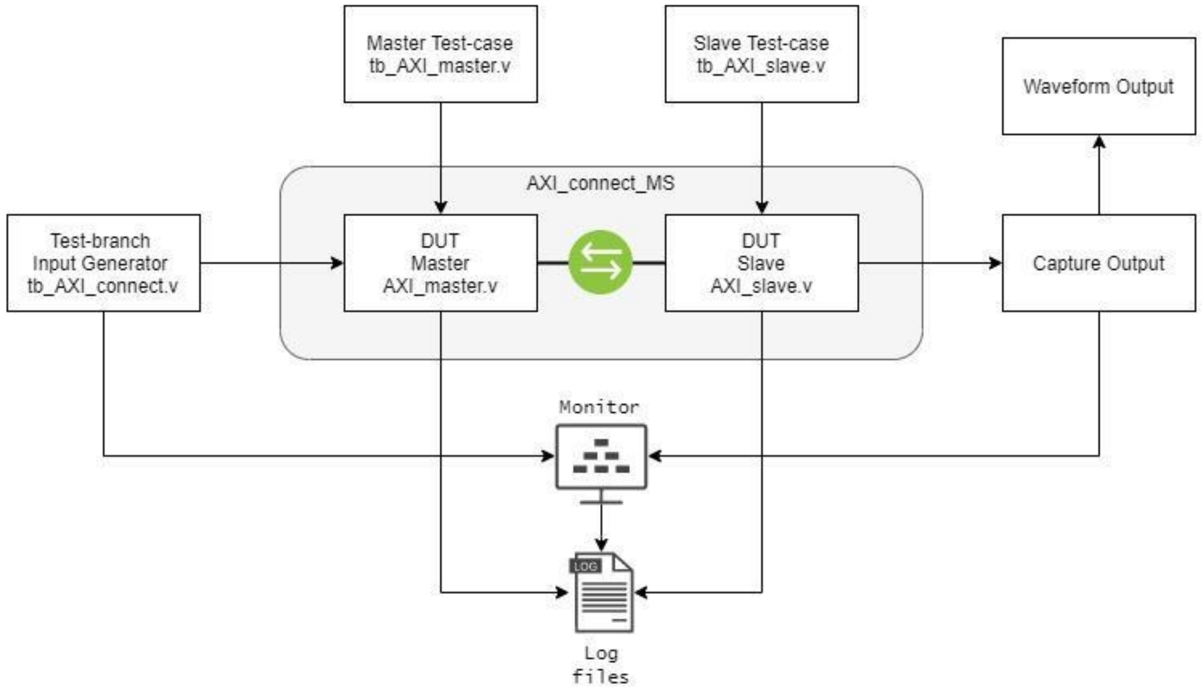


Figure 3.2: Test-plan for verification of master and slave connection as well as individual functionality

The above figure shows the test plan of verification. In that first, we try to implement both DUT modules. After that DUT attach with master test case and slave test case respectively which contains test cases and all required input vectors. Test-cased pass by DUT then results come in form of log files and waveform. If both DUT match required then connect both DUT in AXI_connect_MS, apply test-bench input generator, and capture output. Here output also comes into two forms one is waveforms and another one is log files. Here we implement a self-checking test-bench so that the monitor checks both input and required output. If not match that log file display error message, else log file display test-case pass massage.

Chapter 4: Results

1. Address Write Channel

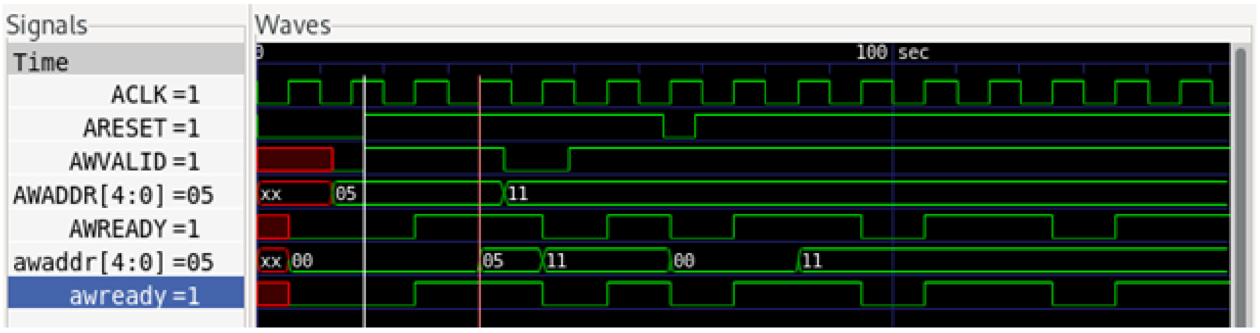


Figure 4.1: Address write channel with required signals

figure 4.1 shows that write address channel simulation on axi_slave module. In that first, we give ACLK and ARESET signals as per specification. Note In this simulation upper case related to slave input or output port and lower case monitor signal which is the latch inside slave. First, we provide AWADDR and AWVALID signals as input and depend on input AWREADY asserted high. After one clock cycle awaddr accepts that input address and latch into the slave as shown in fig.4.1 When the reset was low then awaddr was also goes starting address in this case zero

2. Data Write Channel

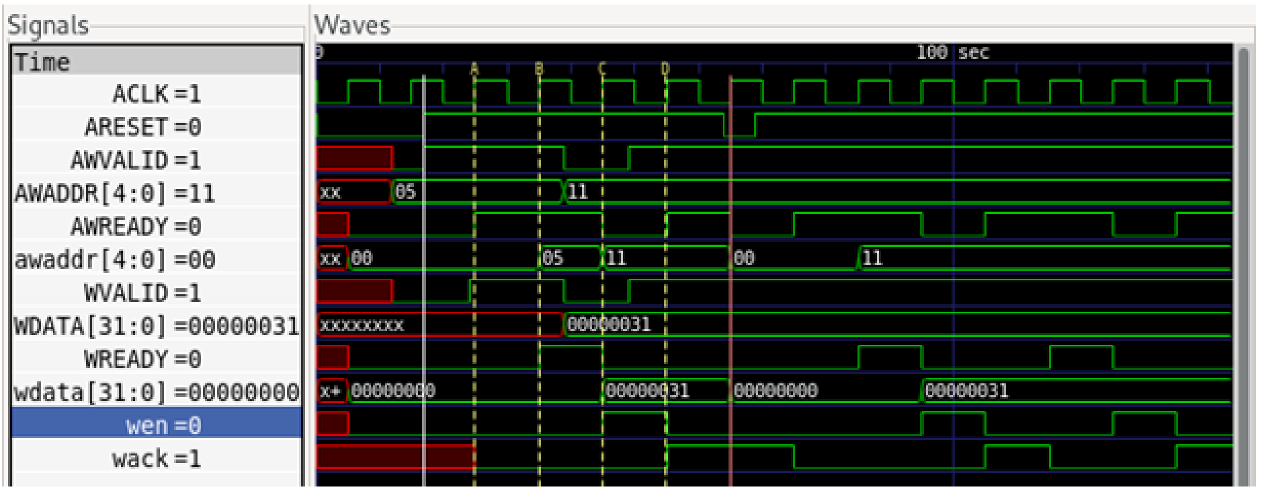


Figure 4.2: Data write channel simulation

Above fig. 4.2, show the simulation of data writes channel at a slave. In that first perform write address and put some address on slave latch. In the first case, the address should be 0x05 which shows with a white marker. At marker B 0x05 latch into a slave. In axi don't have share bus terminology that's why along with the write address we put write data. For that first slave input, WVALID becomes asserted one and WDATA put some value here 0x31. A marker, A WVALID becomes asserting high. After the very next clock cycle, WREADY also becomes asserting high at marker B. As soon as WREADY becomes high next clock cycle wdata latch data at marker C. When data latch into slave wen signal also asserted high. This signal uses to lock data and provide acknowledgment for memory. After the next clock cycle data store in memory of the array and wack is asserted high. Wen and wack both are internal signals for checking the working of write data on the memory array.

3. Write Response Channel

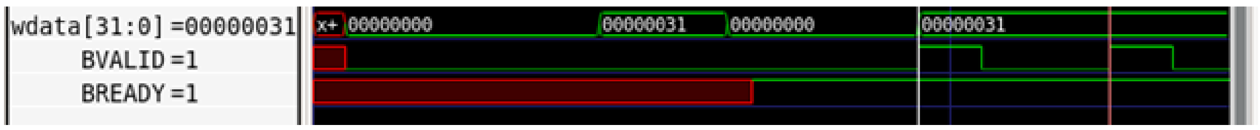


Figure 4.3: Write Response channel simulation

Above fig.4.3 show that the write response channel at the slave in that BREADY signal always asserted high from the master side. When slave latches some data in wdata then write transaction was successful complied and BVALID also asserted high at the white marker. When both BVALID and BREADY asserted high it means handshaking complied and all required data and address latch into a slave.

4. Master and slave connect back to back

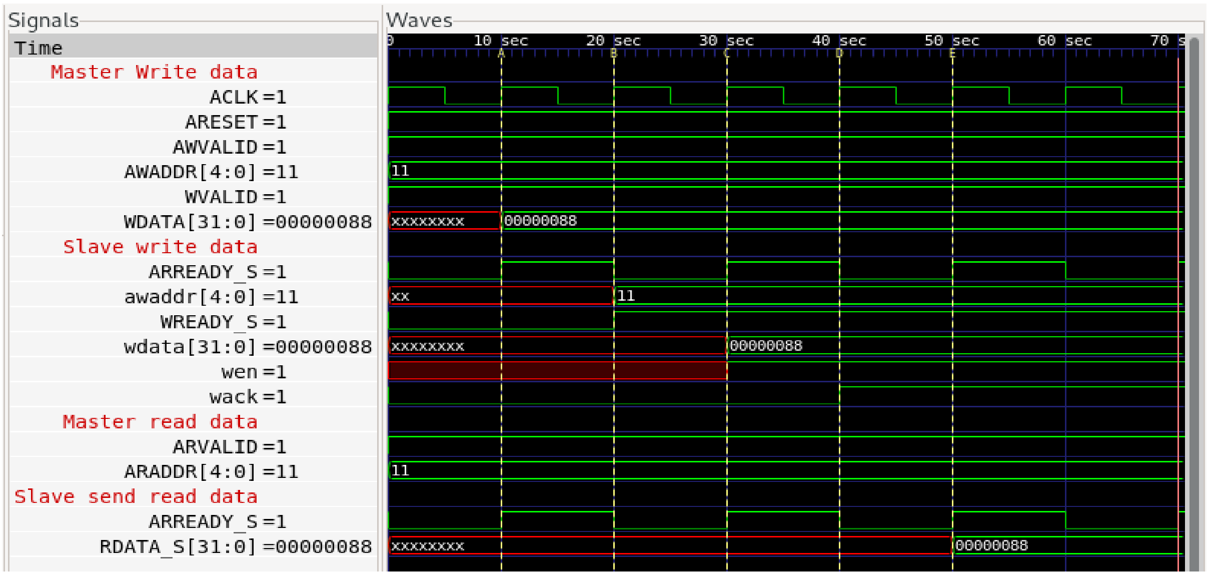
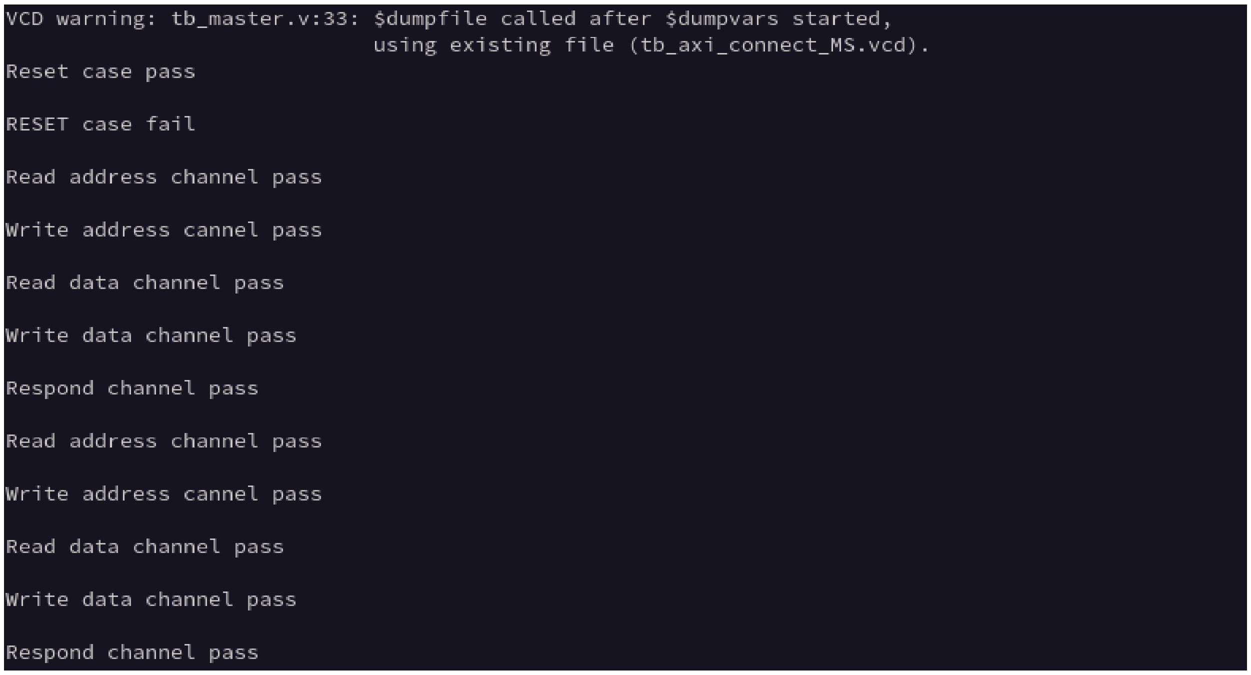


Figure 4.4: Master and slave connect back to back

Above simulation done for checking master and slave back to back connection. In the above fig. 4.4 Position A show master insert AWADDR = 0x11, and WDATA = 0x88 with respective valid signals. Marker B signifies slave get address form AWADDR and latch into awaddr. Marker C signifies WDATA also latch into wdata. Before both information latch, respective ready signal asserted high so handshaking was ensured before feathering any process. At master side also asserted ARADDR and ARVALID with 0x11 and 0x1 respective. It shows that I want to write and read at the same location or address. For a replay of a read, channel slave put related data by given address on RDATA_S which is indicating by marker E.

5. Self-checking Test-bench results



```
VCD warning: tb_master.v:33: $dumpfile called after $dumpvars started,
                             using existing file (tb_axi_connect_MS.vcd).
Reset case pass

RESET case fail

Read address channel pass

Write address cannel pass

Read data channel pass

Write data channel pass

Respond channel pass

Read address channel pass

Write address cannel pass

Read data channel pass

Write data channel pass

Respond channel pass
```

Figure 4.5: Self-checking test-bench results

Chapter 5: Conclusion

In this report, we started out with a conversion about AXI Protocol and AMBA protocol significance, along with its versions variant. In the present digital era, we need faster memories and enormous speed for communicating with different devices. For computing, those requirements need heterogeneous peripherals on SoCs along with communication protocol. To full fill these requirements with AMBA-AXI.

The work focused on design an AXI-Lite 32-bit master and slave using Verilog. And verified design by applying test cases. For Low memory peripheral requirement full fill by using memory array at slave side. Here, we conclude with the synthesis of AXI-LITE master and slave Memory-mapped peripheral along with simulation waveform. At the end, we performing effective data transactions in back-to-back master-slave connection.

References

[1] A. Mittal, “AXI Vs AHB. Difference Between AXI and AHB with design methodologies,” Vlsiiip.com. [Online]. Available: http://www.vlsiiip.com/amba/axi_vs_ahb.html. [Accessed: 12-May-2021].

[2] “Documentation – Arm Developer,” Arm.com. [Online]. Available: https://developer.arm.com/documentation/e/AMBA-AXI3-and-AXI4-Protocol-Specification. [Accessed: 12-May-2021].

[3] https://anysilicon.com/understanding-axi-protocol-quick-introduction/#:~:text=The%20AXI%20is%20a%20point,of%20all%20AMBA%20interface%20interconnect. [Accessed: 12-May-2021]

[4] Jakub Szefer, “AXI4-Lite Interface Development”, EENG 428 / ENAS 968 Cloud FPGA.

[5] Embedded Systems in All Programmable SOC, Electrical And Computer Engineering Department, Oakland University, 2015.

[6] Cristian Sisterna, “Introduction to AXI–Custom IP” , Universidad Nacional de San Juan Argentina

Appendix

Verilog Code for Master:

```
module axi_master#(
            parameter M_AXI_LITE_SIZE=5,
            parameter M_AXI_BUS_SIZE=32
)(
           input ACLK, ARESET,
           input WREADY,
           output reg WVALID,
           output reg [M_AXI_BUS_SIZE-1:0]WDATA,
           output reg AWVALID,
           input AWREADY,
           output reg [M_AXI_LITE_SIZE-1:0]AWADDR,
           input RVALID,
           input [M_AXI_BUS_SIZE-1:0]RDATA,
           output reg RREADY,
           input ARREADY,
           output reg ARVALID,
           output reg [M_AXI_LITE_SIZE-1:0] ARADDR,
           input BVALID,
           output reg BREADY
);


//read address channel


//lowercase is input as a given by master
reg [M_AXI_LITE_SIZE-1:0]araddr;
reg [M_AXI_LITE_SIZE-1:0]awaddr;
reg [M_AXI_BUS_SIZE-1:0]wdata;
reg [M_AXI_BUS_SIZE-1:0]ram_read[M_AXI_BUS_SIZE-1:0];
reg rawk;
reg ren;
reg [M_AXI_BUS_SIZE-1:0]rdata;
reg temp_RVALID;
//assign temp_RVALID = RVALID;

initial begin
```

```
araddr = 'h11;
awaddr = 'h11;
wdata = 'h88;
end


always @(posedge ACLK) begin
        if (!ARESET) ARADDR <=0;
        else if (ARREADY && araddr) ARADDR <= araddr;
        else araddr <= araddr;
end

always @(posedge ACLK) begin
        if(!ARESET) ARVALID <=0;
        else if (araddr!==0 && araddr!=='hx) ARVALID <= 1;
end

always @(posedge ACLK) begin
        if(!ARESET) RREADY <= 0;
//      else if (ARREADY && araddr && RVALID) RREADY <= 1;
        else RREADY <= 1;
end
//reg ren;
always @(posedge ACLK) begin
        if(!ARESET) rdata <= 0;
        else if (ARVALID && ARREADY && RVALID) begin rdata <= RDATA; ren <=1; end
//rdata opration move in ram lowercase
        else begin
        //      RDATA <= RDATA;
                ren <= 0;
        end
end

//reg rawk;
always @(posedge ACLK) begin
        if(ren) begin
                ram_read[araddr] <= rdata;
                rawk <= 1;
        end
        else begin
                ram_read[araddr] <= ram_read[araddr];
                rawk <= 0;
        end
end
```

```
// write channel
//
always @(posedge ACLK) begin
	if(!ARESET) AWVALID <= 0;
	else if (awaddr!==0 && awaddr!=='hxx) AWVALID <= 1;
end

always @(posedge ACLK) begin
	if(!ARESET) WVALID <= 0;
	else if (wdata!==0 && wdata!=='hxx) WVALID <= 1;
end

always @(posedge ACLK) begin
	if(!ARESET) WDATA <= 0;
	else if ((wdata!==0 || wdata!=='hxx) && WVALID) WDATA <= wdata;
	else wdata <= wdata;
end

always @(posedge ACLK) begin
	if(!ARESET) AWADDR<=0;
	else if(awaddr!==0 && awaddr!=='hxx) AWADDR<=awaddr;
	else awaddr<=awaddr;
end

//response channel
always @(posedge ACLK) begin
	if(!ARESET) BREADY<=0;
	else BREADY<=1;
end

/*
//for ws trob affact on wdata block only
always @(posedge ACLK) begin
	if(!ARESET) begin
		WDATA = 0;
		WSTRB = 0;
	end
	else if
end

*/
endmodule
```

Verilog Code for Slave

```
module axi_slave#(
	parameter S_AXI_LITE_SIZE = 5,
	parameter S_AXI_BUS_SIZE = 32)
(input ACLK,ARESET,
	//write address lines
	input [S_AXI_LITE_SIZE-1:0]AWADDR,
	input AWVALID,
	output AWREADY_S,

	//wirte data
	input [S_AXI_BUS_SIZE-1:0]WDATA,
	input WVALID,

	//input [3:0]WSTRB,
	output WREADY_S,

	//write responce
	input BREADY,
	output BVALID_S,
//	input [1:0]BRESP_temp;
	//output [3:0]BRESP,

	//Read address by slave
	input [S_AXI_LITE_SIZE-1:0]ARADDR,
	input ARVALID,
	//input [2:0]ARPROT,
	output ARREADY_S,

	//read Data
	input RREADY,
	output RVALID_S,
	output [S_AXI_BUS_SIZE-1:0]RDATA_S
);


// data_buses for address--------------------------------------
reg [S_AXI_LITE_SIZE - 1:0] araddr; //only support 32bit at max
reg [S_AXI_LITE_SIZE - 1:0] awaddr; //only support 32bit
reg [S_AXI_BUS_SIZE - 1:0] wdata;
reg [S_AXI_BUS_SIZE - 1:0] rdata;
```

```
//--------------------------------------------------------------------------
// internal signal which help to define different output as a function of input
 //ready signal --------------------------------------
reg awready;
reg wready;
reg arready;
//valid signals----------------------------------------
reg bvalid;
reg rvalid;

//other signal--------------------------------------------------------------
//reg awlatch;
reg wen; //use as a not of reset or write enable
reg wack; // write address on clk
reg rv;   // for read valid on when data read tx.


//RAM for data store(write) oprations only-----------------------------------
reg [S_AXI_BUS_SIZE-1:0]ram[S_AXI_BUS_SIZE-1:0];
integer i;  //for ram address access
//wire [4:0]ram_addr = awaddr[4:0]; //assign address send by master to slave in ram address

//-----------------------------------------------------------------------------
//assign or maped all internal signal with output signal
assign AWREADY_S = awready;
assign WREADY_S  = wready;
assign BVALID_S = bvalid;
assign ARREADY_S = arready;
assign RVALID_S = rv;
assign RDATA_S = rdata;

//------------------------------------------------------------------------------
//by defult its value=1 due to no any error occuer at slave it depend on prot signal.
/*initial begin
        awready = 1'b1;
        wready = 1'b1;
end*/

//-----------------------------------------------------------------------------
//write response conditions

/*signal only when it drives a valid write response. When asserted, BVALID must
remain asserted until the rising clock edge after the master asserts BREADY.*/
always @(posedge ACLK) begin
```

```
        if(!ARESET) bvalid<=0;
        else if (~bvalid && WVALID && WREADY_S && AWVALID && AWREADY_S
&& BREADY) bvalid <= 1'b1;
//       else if (BREADY && ~bvalid) rvalid <= 1'b1;
        else bvalid <= 1'b0;
end

//-------------------------------------------------------------------------------
//latch by write address when awready is 1 .write address gives the address
//of the first transfer in a write
always @(posedge ACLK) begin
        if (!ARESET) awaddr <= 0;
        else if (awready) awaddr <= AWADDR;
        else awaddr <= awaddr;
end

//-------------------------------------------------------------------------------
//awready =1 when AWADDR and WVALID are on.
always @(posedge ACLK) begin
        if (!ARESET) awready <= 0;
//      else if (wready) awready <= 1'b0;
//      else if (AWVALID && WVALID && ~awready) awready <= 1'b1;
        else if (AWVALID && WVALID) awready <= 1'b1;
        else awready <= awready;
end

//-------------------------------------------------------------------------------
//wready=1 when WVALID and awready in on
always @(posedge ACLK) begin
        if(!ARESET || WDATA=='hxx) wready <= 0;
        else if(awready && WVALID) wready <= 1'b1;
        else wready <= 1'b0;
end

//-----------------------------------------------------------------------------------
// take data when wready=1
always @(posedge ACLK) begin
        if (!ARESET || WDATA=='hx) begin
                wdata <= 0;
                wen <= 0;
        end
        else if(!awready && !wready) wen<=1'b0;
        //else if (wen) wen <= 1'b0;
        else if (awready && wready && WDATA!=='hx) begin
                wdata <= WDATA;
```

```
            wen <= 1'b1;
        end
        else begin
            wdata <= wdata;
            wen <= wen;
        end
end

//-------------------------------------------------------------------------------------

//wack used as a mutex locking. dont allow to read and write togather used to ack for wdata.
always @(posedge ACLK) begin
        if(!ARESET) begin
                for(i=0;i<32;i=i+1) ram[i]<=31'b0;
        end
        else if (wen) begin
            ram[awaddr] <= wdata;
            wack <= 1'b1;
        end
        else begin
            ram[awaddr] <= ram[awaddr];
            wack <= 1'b0;
        end
end

//--------------------------------------------------------------------------------------
//arready=1 when ARVALID is 1, indicating a valid address to read.

always @(posedge ACLK) begin
        if(!ARESET) arready <= 0;
        else if (ARVALID && ~arready) arready <= 1'b1;
        else arready <= 1'b0;
end

//--------------------------------------------------------------------------------------
//rvalid=1 when RREADY=1 and arready=1.
always @(posedge ACLK) begin
        if(!ARESET) rvalid <= 0;
        else if (arready && ARVALID && RREADY) rvalid <= 1'b1;
        else rvalid <= 1'b0;
end

//------------------------------------------------------------------------------------
//latch ARADDR to araddr reg
always @(posedge ACLK) begin
```

```
        if (!ARESET) araddr <= 0;
        else if (arready) araddr <= ARADDR;
        else araddr <= arad
end

//------------------------------------------------------------------------------------------
//reading from ram giving address and send datain rdata bus
always @(posedge ACLK) begin
        if (!ARESET) begin
                rv <= 0;
                rdata <= 0;
        end
        else if (rvalid) begin
                rdata <= ram[araddr];
                rv <= 1'b1; //axi_rvalid
        end
        else begin
                rdata <= rdata;
                rv <= 1'b0;
        end
end
endmodule
```