Project Report on

Design And Verification of High Speed AXI4-LITE To SDRAM Controller Interface

Submitted in Partial
Fulfillment for the award of

Post Graduate Diploma in VLSI Design

(PG-DVLSI)

from

C-DAC, ACTS (Pune)

PROJECT GUIDE BY

Mr. Suyash Jain

GROUP MEMBERS

Abrar Ul Haq Sanadi	PRN:240340133001
Chandrakant Mohadikar	PRN:240340133005
Hashim Khan	PRN:240340133008
Vinayak Hasure	PRN:240340133009
Siddhant Sukale	PRN:240340133020

ACKNOWLEDGEMENT

This project “Design and Verification of High Performance AXI4-LITE To SDRAM Controller Interface” was a great learning experience for us and we are submitting this work to Advanced Computing Training School (CDAC ACTS).

We are very glad to mention the name of Mr. Suyash Jain for his valuable guidance to work on this project. Their guidance and support helped me to overcome various obstacles and intricacies during the course of project work.

This project, focused on the design and Verification of an AXI4-Lite to SDRAM controller interface, would not have been possible without the support and contributions of several individuals and organizations. We would like to express our sincere gratitude to our academic advisors and mentors for their invaluable guidance and encouragement throughout the development process. Their expertise in digital design and verification has greatly influenced the quality and direction of this project.

CONTENTS

1. INTRODUCTION

1.1 Introduction

In modern system-on-chip (SoC) designs, the integration of memory subsystems like Synchronous Dynamic Random-Access Memory (SDRAM) requires adherence to standardized interfaces for efficient communication and interoperability. The Advanced eXtensible Interface (AXI4-LITE) protocol, widely adopted in ARM-based architectures and beyond, offers a robust framework for connecting memory controllers and other peripherals within complex SoC environments.

This project focuses on enhancing an existing SDRAM controller to support the AXI4-LITE protocol, thereby enabling seamless integration into AXI4-LITE based SoC designs. The transition involves inserting an intermediary logic module that acts as an adapter between the conventional SDRAM controller and the AXI4-LITE interface. This adaptation is essential for ensuring that the SDRAM controller can efficiently communicate using AXI4-LITE compliant signals and protocols, facilitating improved performance, scalability, and compatibility with diverse system architectures.

The objective of this project is to explore and implement these enhancements systematically, ensuring robust functionality thorough verification and testing. By developing a flexible and interoperable AXI4-LITE SDRAM controller, this project aims to contribute to advancements in SoC design, facilitating more efficient and scalable memory subsystem integration across a wide range of embedded and computing applications.

1.2 Abstract

This project aims to transform a conventional SDRAM controller into an AXI4-LITE (Advanced eXtensible Interface) compatible SDRAM controller, enhancing its versatility and integration capabilities within modern system-on-chip (SoC) designs. The adaptation process involves introducing an intermediate logic layer that bridges the gap between the existing SDRAM controller and the external AXI4-LITE interface. This intermediary module ensures seamless communication by converting the SDRAM controller's native operations and data transactions into AXI4-LITE compliant signals and protocols.

Key to the project's success is the parametrization of the AXI4-LITE interface module, which allows for customizable configurations of critical signal widths including ID, address, user signals, and data. This flexibility ensures compatibility with diverse implementation requirements across different SoC architectures, maintaining adherence to AXI4-LITE protocol standards while accommodating specific design constraints and optimization goals.

By integrating this adaptable AXI4-LITE SDRAM controller into SoC designs, the project aims to enhance system performance and flexibility. It facilitates efficient memory access and management, crucial for applications ranging from embedded systems to high-performance computing environments.

This project contributes to advancing the field of SoC design by providing a scalable and interoperable solution for integrating SDRAM memory subsystems into AXI4-LITE-based architectures, paving the way for enhanced system efficiency and performance optimization in diverse computing and embedded applications.

1.3 Aim of Project

The project aims to design an AXI4-Lite to SDRAM controller IP core to enhance system performance and flexibility, enabling seamless integration into AXI-based System-on-Chip (SoC) designs. This controller will facilitate efficient memory access and management, which is crucial for a variety of applications ranging from embedded systems to high-performance computing environments.

The architecture of the AXI4-Lite to SDRAM controller consists of several main components, including an AXI4-Lite interface that handles the protocol and decodes transactions, a command queue that buffers incoming read/write commands, and an SDRAM command scheduler that manages the generation of necessary commands such as ACTIVATE, READ, WRITE, and PRECHARGE. Additionally, a control finite state machine (FSM) will oversee the overall operation of the controller, ensuring smooth interaction between components. The implementation of this controller will not only improve system performance by enabling efficient access to SDRAM memory from AXI masters but also reduce design effort for SoC integrators by providing a pre-verified SDRAM controller IP. Ultimately, this project will enhance flexibility by supporting multiple SDRAM protocols and configurable parameters, thereby enabling high-performance applications that require fast and reliable memory access.

CHAPTER 2

THEORETICAL DESCRIPTION OF PROJECT

2.1 Introduction to AMBA

The Advanced microcontroller bus architecture (AMBA) family enables extensive testing of intellectual property (IPs) such as from ARM and other IP suppliers and system-on-chip (SoC) design through metric-driven protocol compliance verification. It delivers high- frequency operation using sophisticated bridges and supports tremendous performance and high-frequency. It is appropriate for high-bandwidth and low latency designs. The Advanced eXtensible Interface 4 (AXI4) bus family, defined as part of the ARM - AMBA standard's fourth version. The AMBA – AXI4-lite bus protocol is a subset of the AXI4 bus protocol with a simpler interface than the full-featured AXI-4 bus protocol. It only supports one ID thread per master, therefore it's best for an endpoint that only has to connect with one master at a time. There are five channels in the AXI4-Lite interface: read data, read address, write data, write address, and write response. Burst lengths up to 256 bits are supported by the AXI4 family.

2.2 Features of AXI4-lite

AXI4-Lite is a simplified version of the AXI4 (Advanced eXtensible Interface) protocol, designed for use in simpler, lower-performance peripheral devices. It is part of the ARM AMBA (Advanced Microcontroller Bus Architecture) family of protocols and provides a lightweight, memory-mapped interface that is easy to implement in hardware and software.

1. Simple Address/Data Bus Interface

Address Width: AXI4-Lite uses a fixed address width, typically 32 bits. This allows the master to address up to 4 GB of memory space.

Data Width: The data bus can be either 32 or 64 bits wide, supporting simple and efficient data transfers.

2. Unidirectional Channels

AXI4-Lite defines two independent, unidirectional channels:

Write Address and Data Channel: Handles the transmission of address, data, and control signals for write operations.

Read Address and Data Channel: Handles the transmission of address, data, and control signals for read operations.

3. Simple Handshaking Mechanism

AXI4-Lite uses a simple ready/valid handshake protocol to transfer data between master and slave.

AWVALID/ARVALID: Asserted by the master to indicate that the address/control information is valid.

WVALID: Asserted by the master to indicate that write data is valid.

AWREADY/ARREADY/WREADY: Asserted by the slave to indicate that it is ready to accept the address/control information or write data.

RVALID: Asserted by the slave to indicate that read data is valid and ready to be accepted by the master.

RREADY/BREADY: Asserted by the master to indicate that it is ready to accept read data or write response.

4. Single-Beat Transfers

Unlike full AXI4, which supports burst transfers (multiple data beats per address phase), AXI4-Lite only supports single-beat transfers. This simplifies the protocol and reduces the complexity of the hardware.

5. Memory-Mapped Addressing

AXI4-Lite is designed for memory-mapped peripherals, meaning every transaction targets a specific address within the memory map.

This feature makes AXI4-Lite ideal for simple control registers, where each register can be directly mapped to a specific address in memory.

6. Simple Control Signals

AWPROT/ARPROT: These optional control signals indicate the protection level of the transaction, such as whether the transaction is secure or not, and whether it is a data or instruction access.

7. Write and Read Responses

BRESP: The slave uses this signal to send a response back to the master after a write operation, indicating whether the write was successful (OKAY) or if an error occurred (SLVERR or DECERR).

RRESP: The slave uses this signal to send a response back to the master after a read operation, similarly indicating the success or failure of the read.

8. No Support for Out-of-Order Transactions

AXI4-Lite transactions are processed strictly in order, with no support for out-of- order execution. This further simplifies the design.

9. No Cache and QoS Signals

AXI4-Lite does not support the cache or Quality of Service (QoS) signals present in the full AXI4 protocol. These signals are unnecessary for the simpler peripherals that AXI4-Lite is designed for.

10. Lightweight Implementation

Due to the absence of burst transfers, out-of-order execution, and other complex features, AXI4-Lite can be implemented with minimal logic, making it suitable for low-resource environments such as FPGA designs or low-power microcontrollers.

2.3 Channel Handshake Mechanism of AXI4-lite

AXI4-Lite has two primary channels:

- Write Address/Write Data Channel: Used for write operations.
- Read Address/Read Data Channel: Used for read operations.

Each channel uses a VALID and READY signal pair to manage the handshake:

- **VALID**: Asserted by the source (master or slave) to indicate that the data or control information is available and valid.
- **READY**: Asserted by the destination (slave or master) to indicate that it is ready to accept the data or control information.

AXI4-lite Channels

AXI4-Lite has the following channels:

- Write Address Channel (AW)
- Write Data Channel (W)
- Write Response Channel (B)
- Read Address Channel (AR)
- Read Data Channel (R)

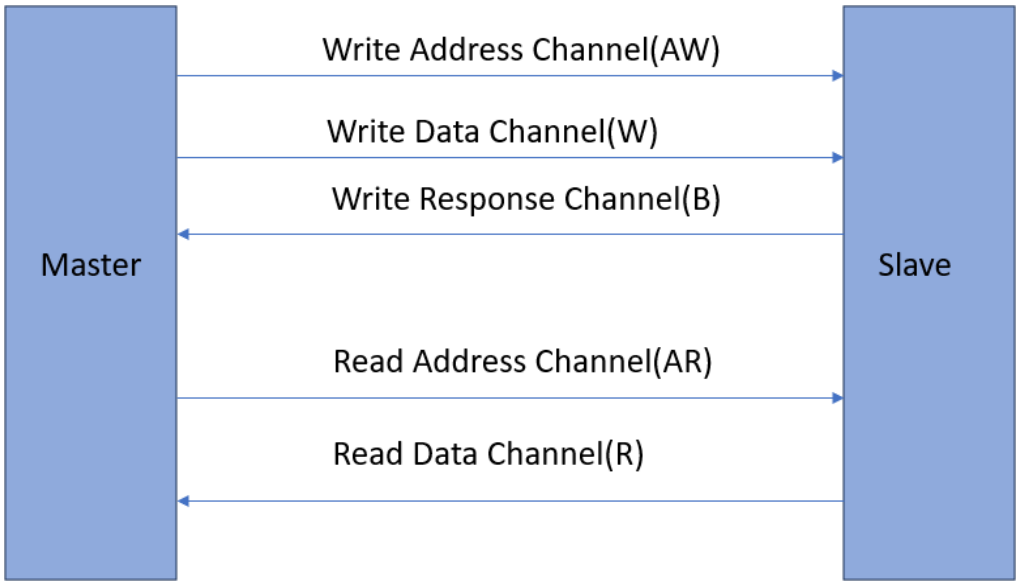

Fig. 2.1 AXI4-lite Channels

In AXI4-Lite, the **Write Address Channel** is responsible for carrying the address and control signals for write transactions. The master sends the address to the slave, which then acknowledges its readiness to receive the data.

The **Write Data Channel** is used to transmit the actual data to be written to the specified address. The master sends the data, and the slave confirms its readiness to accept it.

The **Write Response Channel** provides feedback from the slave after a write operation. It informs the master whether the write was successful or if an error occurred.

The **Read Address Channel** carries the address and control signals for read transactions. The master sends the read address, and the slave acknowledges when it is ready to provide the requested data.

The **Read Data Channel** returns the requested data from the slave to the master. The slave sends the data, and the master indicates its readiness to receive it.

CHAPTER 3

3.1 Introduction to SDR SDRAM

This section will give a more specific review of the timing requirements and relevant commands involved in the operation of a DRAM. In this case, the focus will be specifically the SDR SDRAM. Table 3.1 shows the different signals in a SDR SDRAM chip.

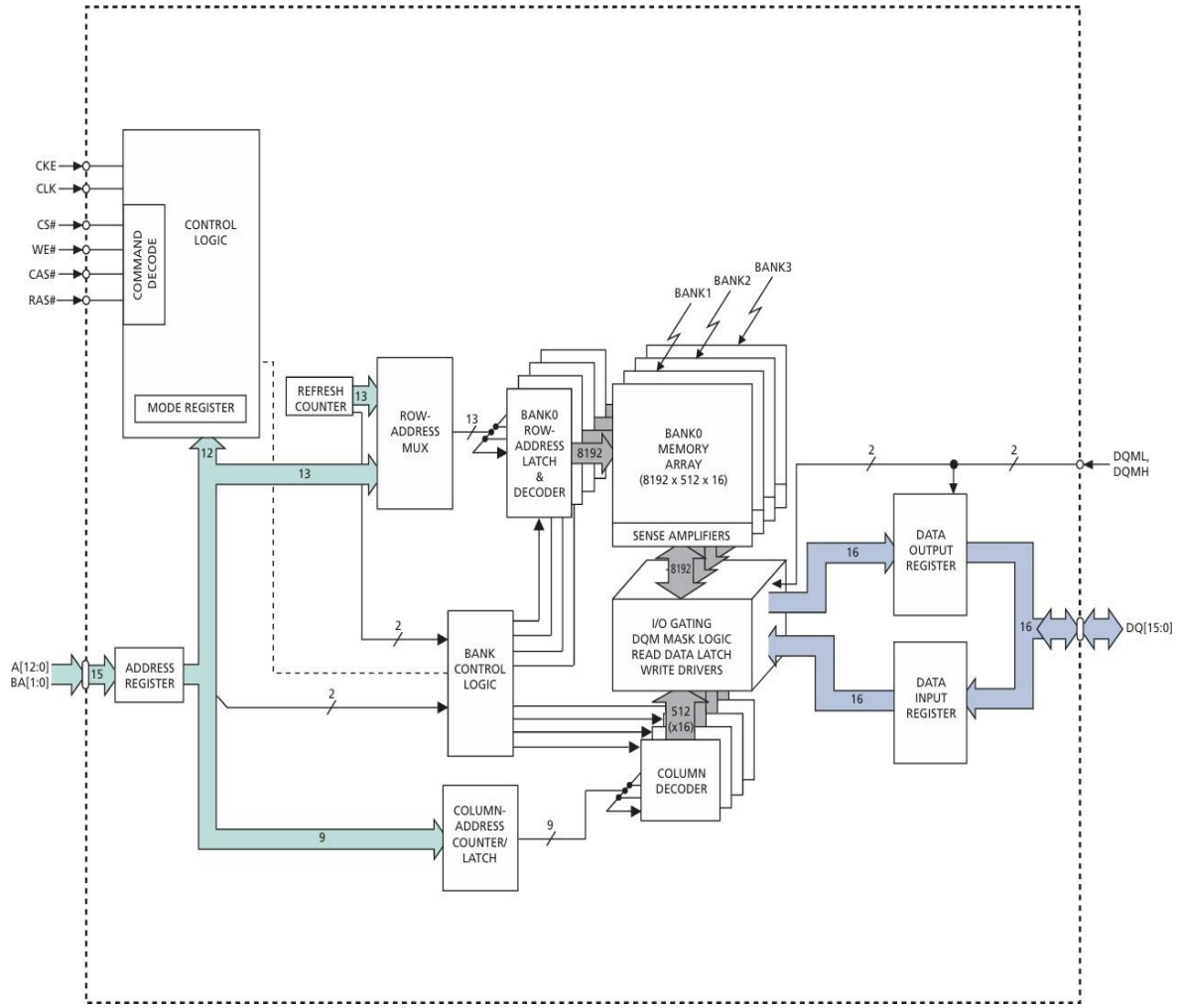

Fig. 3.1 16 Meg x 16 Functional Block Diagram

Pin and Ball Descriptions

Name	Width	Type	Description
CLK	1	Input	Reference clock. All SDRAM input signals are sampled at the positive edge of the clock.
CKE	1	Input	Clock enable, active high. When low, disables the CLK signal. It is used to control the low power operation modes of the SDRAM.
CS	1	Input	Chip select, active low. Enables or disables the command decoder.
WE	1	Input	Write enable. Command bit in the SDRAM, in a DRAM it was used to indicate the type of operation (read or write).
CAS	1	Input	Column Address Strobe. Command bit in the SDRAM, in a DRAM it was used to indicate the address was a valid column address.
RAS	1	Input	Row column strobe. Command bit in the SDRAM, in DRAM it was used to indicate the address was a valid row address.
A[12:0]	>11	Input	Address inputs: A[12:0] are sampled during the ACTIVE command (row address A[12:0]) and READ or WRITE command, column address A[8:0] for x16; with A10 defining auto precharge to select one location out of the memory array in the re-spective bank. A10 is sampled during a PRECHARGE command to determine if all banks are to be precharged (A10 HIGH) or bank selected by BA[1:0] (LOW). The address inputs also provide the op-code during a LOAD MODE REGISTER command.
BA	1,2,3	Input	Bank selection bits. Width depends on the number of banks in the SDRAM. SDRAMs can have 2, 4 or 8 banks.
DQ	4,8,16,32	Input	Bidirectional data bus, it can have 4, 8, 16 or 32 bits.
DQM	1,2,4	Input	Data mask. There is one mask bit for each byte in the data bus.

Table 3.1 SDR SDRAM interface signals

SDRAM modules are provided with different configurations of capacity, element size and maximum frequency. Internally the modules are configured as quad-bank DRAM with a synchronous inter face: the accesses are synchronous to the reference clock and the data is registered at the positive edge of the clock. Accesses to the SDRAM are burst oriented: each request can be made up of multiple transfers. Read and write operations start at a given address and continue for a programmed number of locations, which can be of 1, 2, 4 and, 8 locations or without limit. In case of unlimited burst, referred to as a full-page burst, an additional command must be sent to terminate the operation.

The commands of the SDRAM will be described in more detail on the following lines, along with the main timing requirements. Additionally, some examples of the basic operations will be provided.

3.2 Commands

To control the operation of the SDRAM a set of commands are sent through the CS, RAS, CAS, and WE pin, then the mask (DQM) bank (BA) and address (A) bits provide additional configuration for some commands. Table 3.2 summarizes the commands used to operate with an SDRAM. Command names can differ from one vendor to another, but their functionality is the same. The 11th bit of the address (A[10]) is used to control the precharge of the SDRAM. Since it has a special functionality, it is given a dedicated column separated from the rest of the address. On the following lines, each command will be described with a bit more detail to have a better understanding of their functionality and how it relates to the internal operation of the SDRAM.

Command	**CS**	**RAS**	**CAS**	**WE**	**DQM**	**BA**	**A**	**A[10]**
No Operation	L	H	H	H	x	x	x	x
Active	L	L	H	H	x	Bank	Row	x
Read	L	H	L	H	Mask	Bank	Col	Precharge
Write	L	H	L	L	Mask	Bank	Col	Precharge
Burst Terminate	L	H	H	L	x	x	x	x
Precharge	L	L	H	L	x	Bank	x	All banks
Auto refresh	L	L	L	H	x	x	x	x
Load Mode Register	L	L	L	L	x	Opcode	Opcode	x

Table.3.2 Truth Table – Commands and DQM Operation

- **No operation**

The No Operation is an auxiliary command that is used when no other command should be sent to the SDRAM, either because it is in idle or waiting for the completion of another command. It is equivalent to disabling the chip select.

- **Precharge**

The Precharge command deactivates an active row, which is done by resetting the sense amplifiers and precharging the bit-lines to the reference voltage so they are ready for another row access. There is a minimum time, tRP, between a Precharge command and the next Active command.

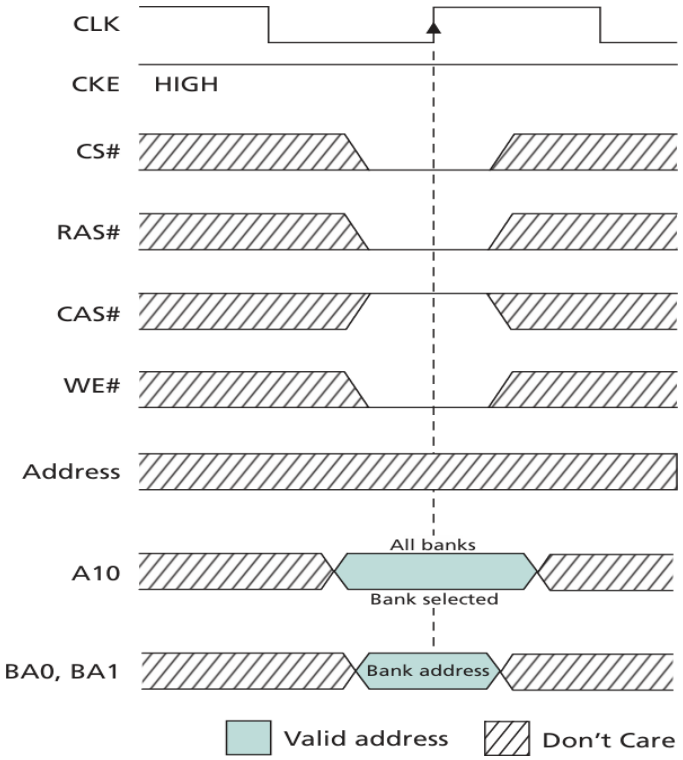

Fig. 3.2 Precharge Command

- **Auto Refresh**

The Auto Refresh command must be sent during the normal operation of the SDRAM to refresh its contents. The SDRAM automatically generates the address that needs to be refreshed with an internal counter, the refresh counter is updated each time the Auto Refresh command is sent. All the rows in the SDRAM must be refreshed within the refresh period, tREF. Commands can be distributed along the period or sent all of them in a burst. After a Precharge command a minimum time, tRP must be respected before sending the Auto Refresh command. There is also a minimum time, tRFC, between two consecutive Auto Refresh commands.

- **Load mode register**

The Load mode register command is used to configure different parameters of the related to the operation of the SDRAM, such as the burst length or the CAS latency. The Load mode register can only be sent when all banks are precharged. After sending it some time, tMRD, must be waited for the SDRAM to be configured and no other commands can be sent.

- **Active**

The Active command is used to open (activate) a row in a particular bank so it can be further accessed with a Read or Write command. When using the Active command, the bank bits select the bank while the address bits determine the row. Internally this operation moves the data from the memory cells of the selected row and bank in the SDRAM array to the sense amplifiers, then restores the values of the memory cells so data is not lost. The whole row of data is moved into the sense amplifier. This means that multiple accesses to read or write data can be done on different columns of the same row without having to send the Active command each time. There is a minimum time, tRCD, before Read or Write commands can be sent after having sent an Active command. An Active command to another row of the same bank can only be sent after precharging the bank. The sense amplifiers of the bank have to be reset before reading new data. On the other hand, it is possible to send an Active command to a different bank, since each bank has its own set of sense amplifiers. There is a minimum time, tRRD, between consecutive Active commands to different banks.

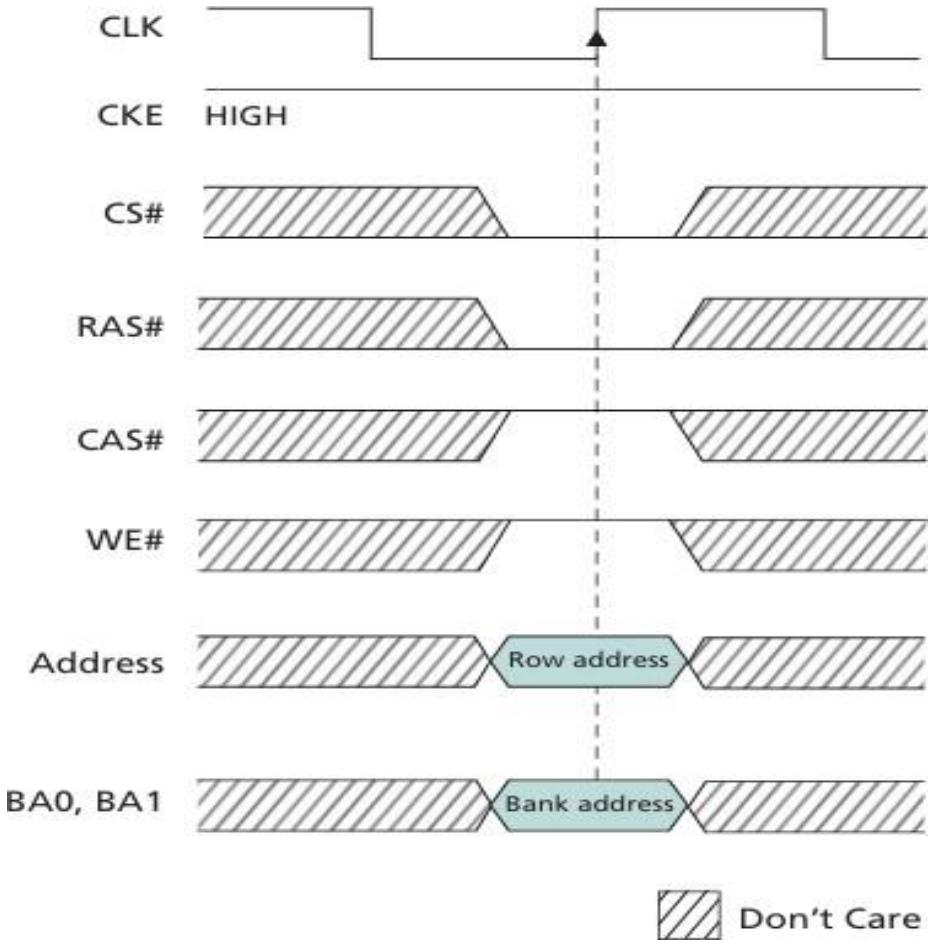

Fig.3.3 Active Command

- **Read**

The Read command is used to start a read access in a previously opened row. The bank bits select the bank while the address bits select the column to be read. Additionally, the 11th bit of the address determines if an auto precharge will be performed by the SDRAM after data has been read. Internally, this command moves the data from the sense amplifier of the given bank to the output data bus. Once the command is sent, the read data is available on the data bus after tCAS. There is a minimum time, tCCD, between two consecutive Read or Write commands.

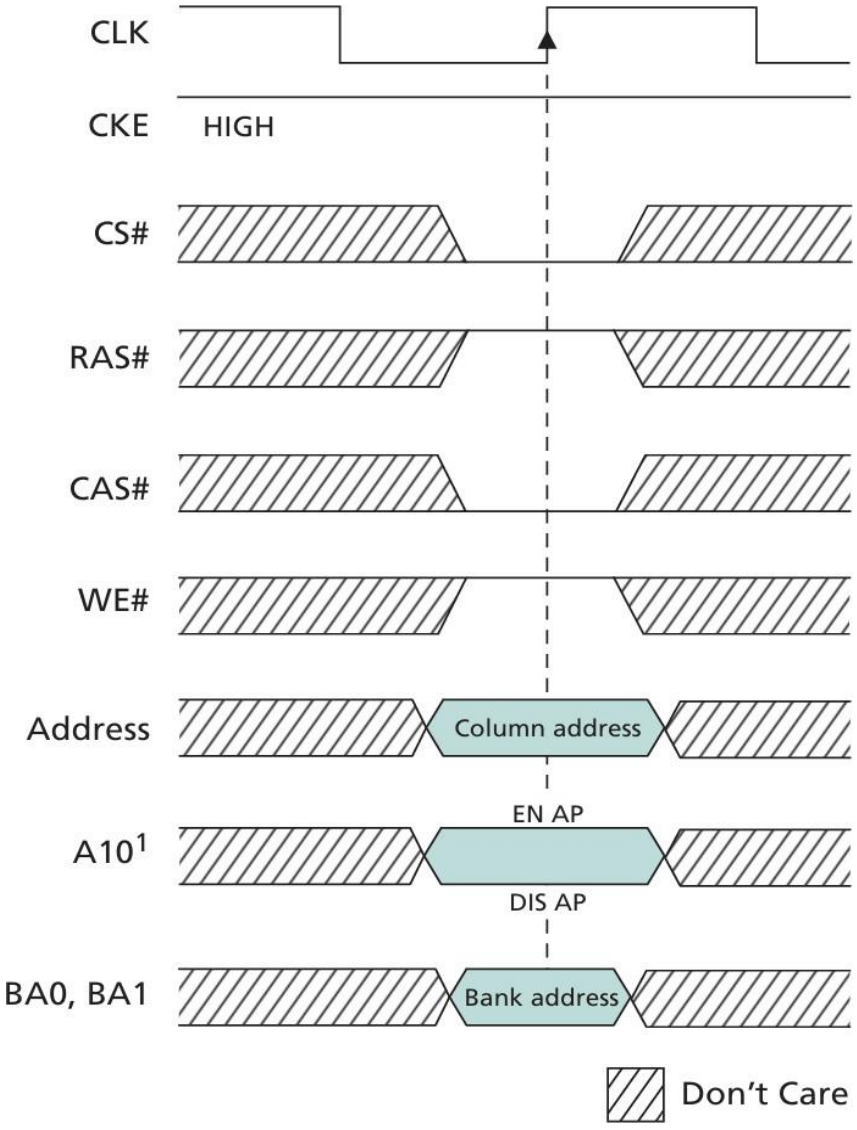

Fig 3.4 Read Command

- **Write**

The Write command starts a write access to a previously opened row. As with the Read command, the bank bits select the bank and the address bits the column with the 11th bit enabling auto precharge. Internally the Write command copies the data from the data bus into the sense amplifiers of the given bank. There is a minimum time, tWR, between a Write and a Precharge command to ensure the sense amplifiers are not precharged until data is written into memory.

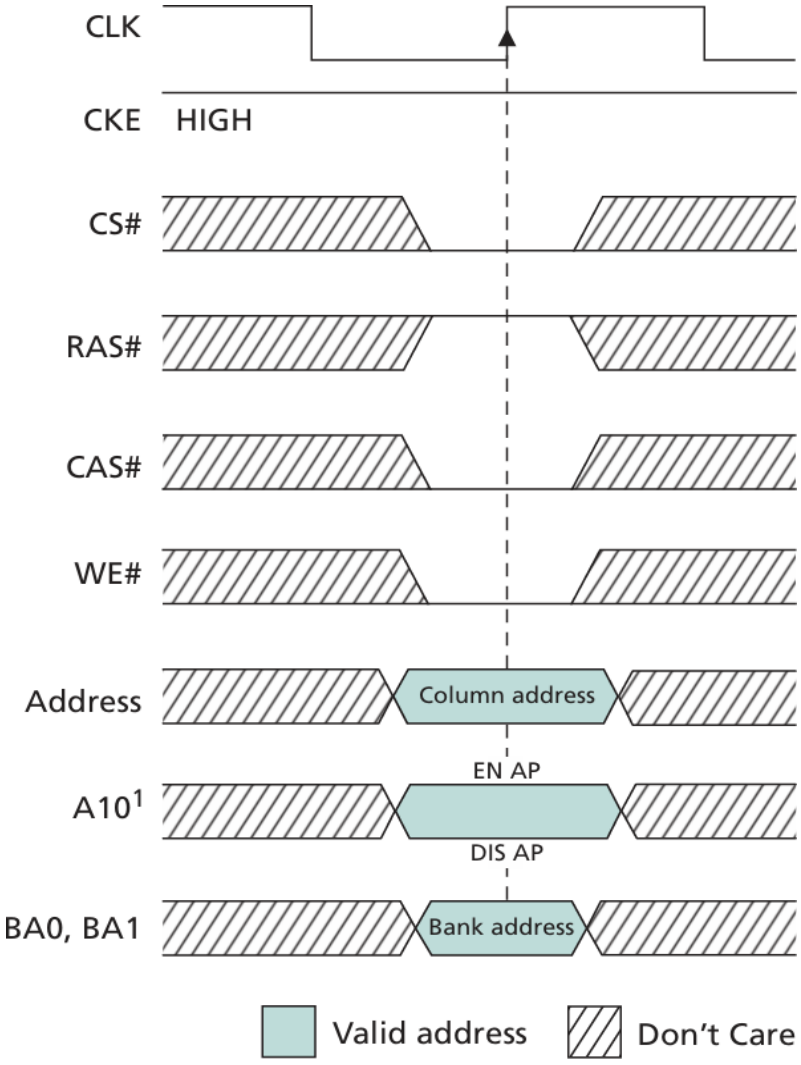

Fig. 3.5 Write Command

- **Burst terminate**

In an SDRAM, reads and writes are burst oriented, which means each Read or Write command performs several accesses to memory. The Burst terminate command is used to truncate the burst started with the last Read or Write command.

3.3 Operation

- **Initialization**

Before normal operation, the SDRAM must follow a set of initialization steps. Once power is applied and the clock is stable, SDRAM requires a delay of 100us. During this period, only the **No Operation** command can be sent. Alternatively, the chip select can be disabled. After the initial delay, the **Precharge** command must be sent to all banks to ensure there is not any open row. Then at least two refresh cycles must be performed by sending the **Auto Refresh** command. After that the SDRAM is ready. Since the state of the mode register will be undetermined after power up, the last step previous to use the SDRAM is to load the mode register with the right configuration using the Load Mode Register command.

- **Write operation**

Figure 3.5 shows an example of a basic write request to SDRAM memory. Initially, the bank is precharged. Since the 11th bit of the address (A[10]) is low, only the selected bank will be precharged. Then the Active command is sent to open the selected row in the selected bank. after waiting for the required time, the Write command is sent to write the data into the specified column of the previously opened row in the selected bank. Since the 11^{th} bit of the address is not enabled, the SDRAM will not do an automatic precharge after writing

- **Read operation**

Read operation is similar as the write operation. In this case, as shown in figure 3.4, after the Read command, the data will be ready after the number of cycles defined by the CAS latency. In case of a burst operation, subsequent data will be updated on next cycles.

Parameter	Description
tRCD	Row to Column Delay, it is the time it takes for the Active command to move the data from the SDRAM cell array to the sense amplifier.
tRRD	Row activation to Row activation delay, it is the minimum time between two Active commands.
tRAS	Row Access Strobe latency, is the time it takes for the Active command to discharge and restore the data from the row of SDRAM cells. After this time the sense amplifiers have completed the restoration of data to the SDRAM cells and the sense amplifiers can be precharged to access another row.
tCAS	Column Access Strobe latency, it is the time it takes to the SDRAM to put the data on the data bus after receiving the Read command.
tCCD	Column to Column Delay, it corresponds to the minimum burst duration and it is determined by the internal prefetch length of the SDRAM.
tRP	Time needed before the bit lines and sense amplifiers are properly precharged after a Precharge command.
tWR	Write recovery time, is the time needed for the write data to be written into the internal SDRAM arrays. It the minimum time between a Write and a Precharge command.
tRFC	Required time between two consecutive Auto Refresh commands.
tMRD	Required time between a Load Mode Register and the next Active or Auto Refresh command.
tREF	Refresh period within all the rows of the SDRAM should be re freshed.

Table 3.3 Summary of SDRAM timing parameters

CHAPTER 4

Design Methodology

4.1 Methodology

The purpose of this project is to design a memory controller that acts as an interface between the SoC and the external memory. For one side, the controller has to communicate with the main processor using the AXI protocol, previously described in section 2.2. On the other side, the controller has to send the required request to the external SDR SDRAM. The design has been designed in Verilog, then synthetized, and finally verification has been done. This chapter describes the different steps and methodologies used involved in the design of the AXI4-lite-SDRAM controller.

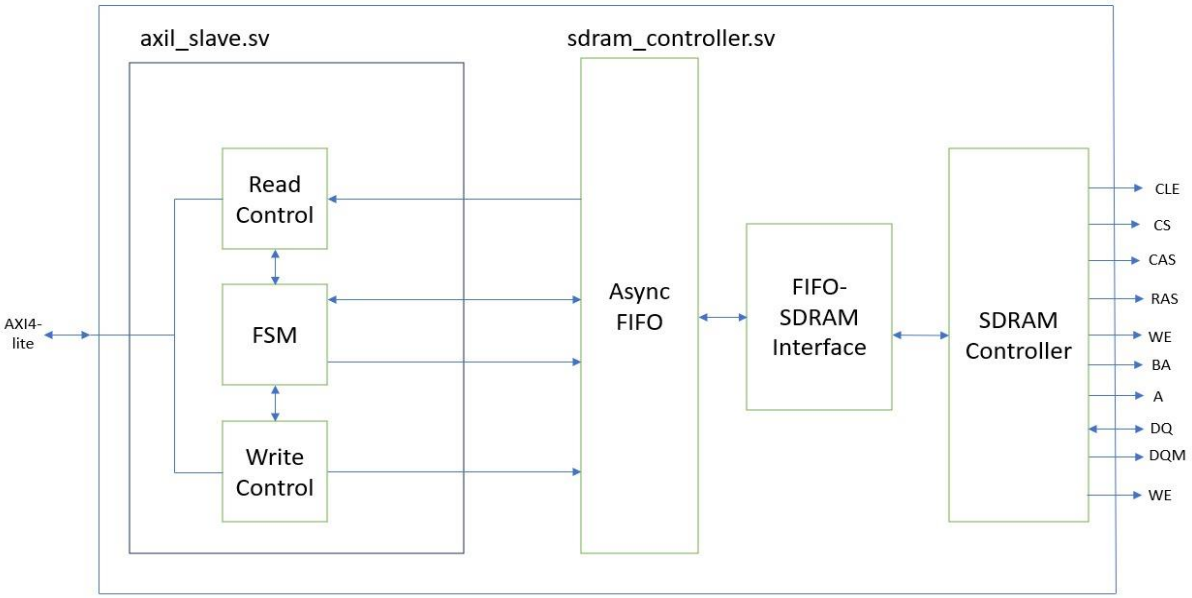

Fig.4.1 Organization of the Design Module

The purpose of the controller is to provide a simplified way to the accesses the memory.

The proposed design is composed of two cascaded blocks as shown in figure 3.1: AXI4-lite interface: This module is used to adapt the previous SDRAM interface to the AXI4-lite specification, turning the design into an AXI4-lite slave that can communicate with the main processor through the AXI4-lite bus.

SDRAM controller: This module interacts with the external SDRAM memory and provides a simple interface that makes the read and write operations transparent to the user. It contains the logic required to manage the accesses to the SDRAM, generate the appropriate commands and control the refresh sequence. Additionally, it contains an asynchronous FIFO which is used to synchronize the data between different clock domains, from the main clock to the SDRAM clock and the other way around.

Both blocks are wrapped together to provide the final AXI-SDRAM interface. On the follow

ing lines the actual implementation of the different modules in the design will be described.

4.2 AXI4-lite Interface

To implement the final AXI4-Lite SDRAM controller, an extra layer of logic is needed to convert the previous SDRAM controller into an AXI4-Lite slave-compatible device. This module is placed between the SDRAM controller and the external AXI4-Lite interface, which means it has to properly match with both interfaces.

Read and Write signals in AXI4-lite:

- **Clock and Reset Signals:**

  s_axil_clk: The clock signal for the AXI4-Lite interface.

  s_axil_resetn: The active-low reset signal for the AXI4-Lite interface.

- **Write Address Channel Signals:**

  s_axil_awaddr: The address where data will be written.

  s_axil_awvalid: Indicates that the write address and control signals are valid.

  s_axil_awready: Indicates that the slave is ready to accept the write address.

- **Write Data Channel Signals**

  s_axil_wdata: The data to be written to the specified address.

  s_axil_wvalid: Indicates that the write data is valid.

  s_axil_wstrb: Byte enable signals indicating which bytes of s_axil_wdata are valid.

  s_axil_wready: Indicates that the slave is ready to accept the write data.

- **Write Response Signals**

  s_axil_bready: Indicates that the master is ready to accept the write response.

  s_axil_bresp: The response from the slave indicating the status of the write transaction.

  s_axil_bvalid: Indicates that the write response is valid.

- **Read Address Channel Signals:**

  s_axil_araddr: The address from which data will be read.

  s_axil_arvalid: Indicates that the read address and control signals are valid.

  s_axil_arready: Indicates that the slave is ready to accept the read address.

- **Read Data Channel Signals**

  s_axil_rdata: The data read from the specified address.

  s_axil_rresp: The response from the slave indicating the status of the read transaction.

  s_axil_rvalid: Indicates that the read data is valid.

  s_axil_rready: Indicates that the master is ready to accept the read data.

FSM for Read Channel:

The provided FSM for the AXI4-Lite read channel operates in three states: IDLE, DATA, and READ RESPONSE. In the IDLE state, the FSM waits for a valid read address (s_axil_arvalid) and, if accepted, stores it and enables the address FIFO input if it's not full. It then transitions to RD_DATA, where it checks if the master is ready for data and if data is available in the FIFO. If both conditions are met, the data is loaded and marked valid (reg_rvalid), and the FSM moves to READ RESPONSE. In this final state, the data path is disabled, reg_rvalid is cleared, and the FSM returns to IDLE to await the next transaction.

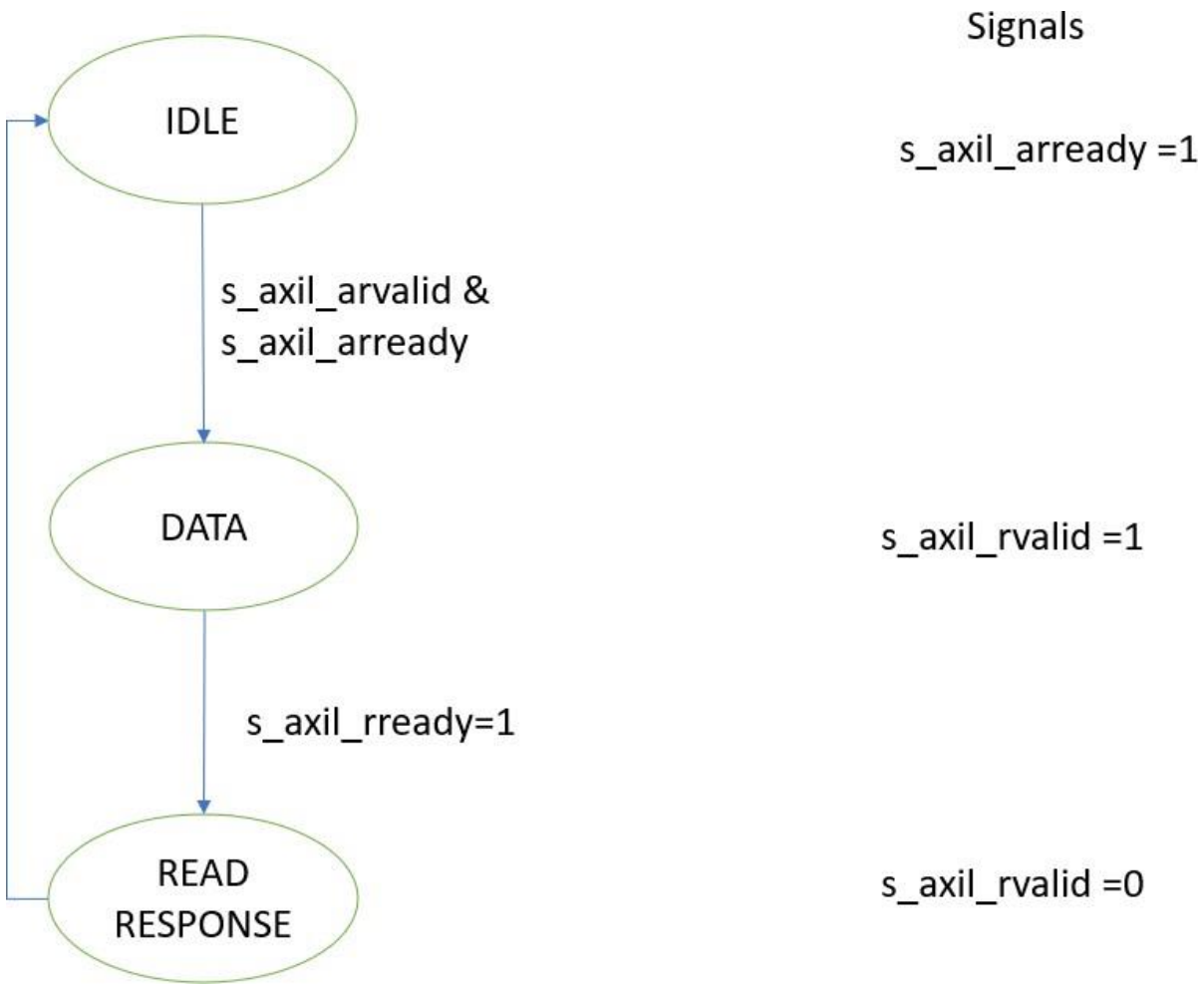

Fig 4.2 FSM for Read Channel

FSM for Write Channel:

The provided FSM for the AXI4-Lite write channel operates through four states: IDLE, DATA, RESPONSE, and DONE. In the IDLE state, the FSM waits for a valid write address (s_axil_awvalid) and, upon a handshake, captures the address and moves to the DATA state if the address FIFO is not full. In DATA, the FSM waits for valid write data (s_axil_wvalid), and if the data FIFO is not full, it captures the data, deasserts wready, and either moves to the RESPONSE state or directly to DONE if the response is already acknowledged by the master (s_axil_bready). In the RESPONSE state, the FSM asserts bvalid to signal a successful write, then transitions to DONE. In the DONE state, bvalid is deasserted, and the FSM returns to IDLE to begin a new transaction.

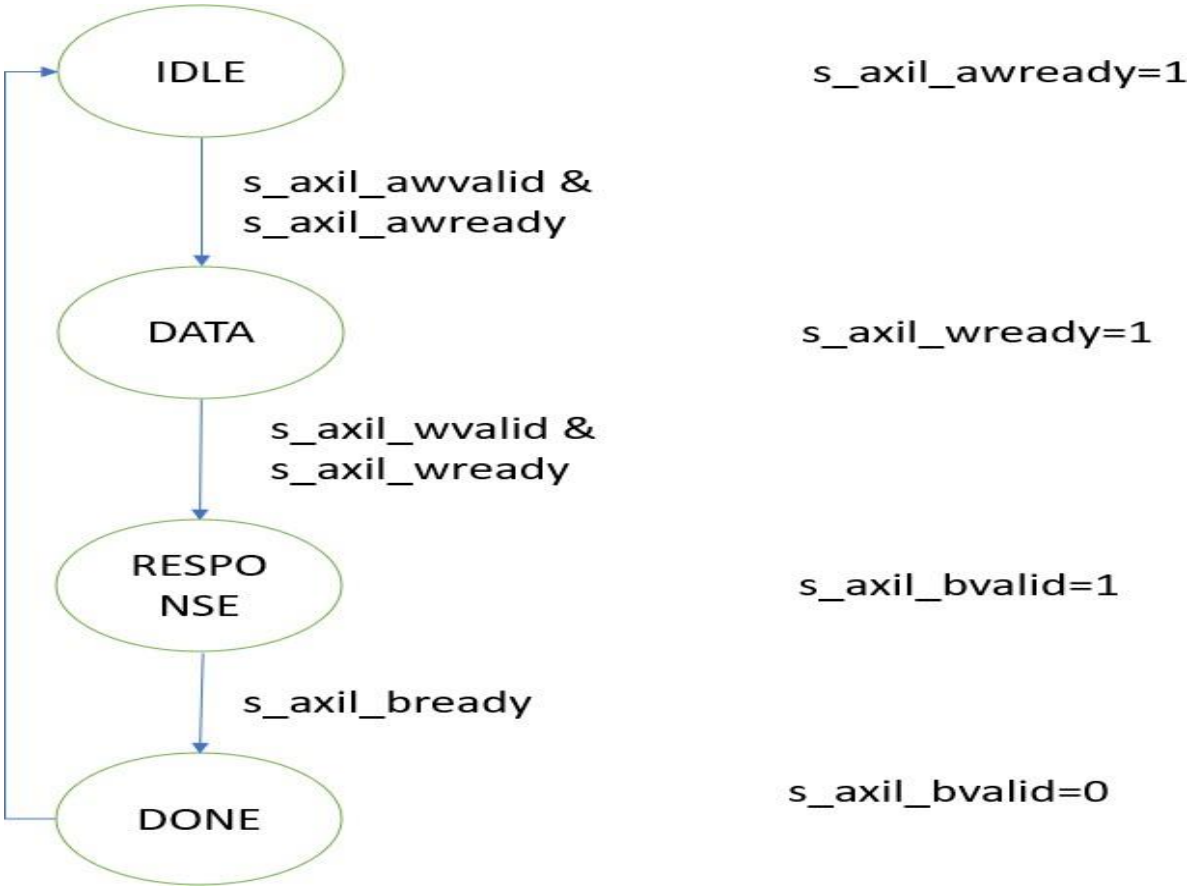

Fig 4.3 FSM for Write Channel

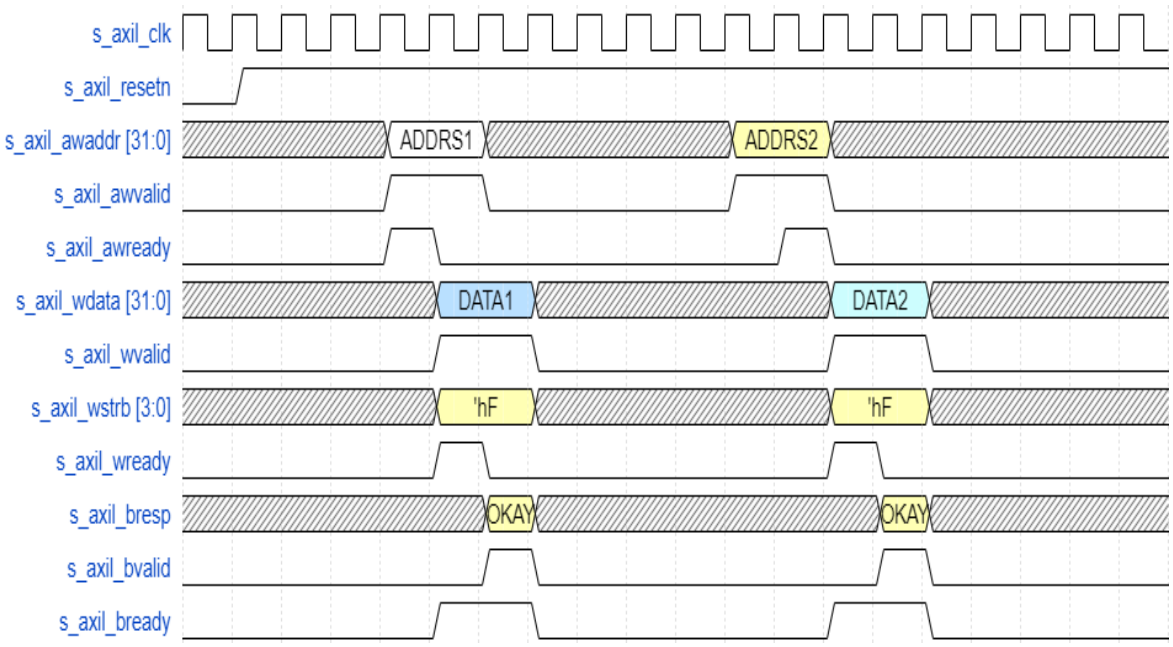

Fig. 4.4 AXI4-lite Write Channel Waveform

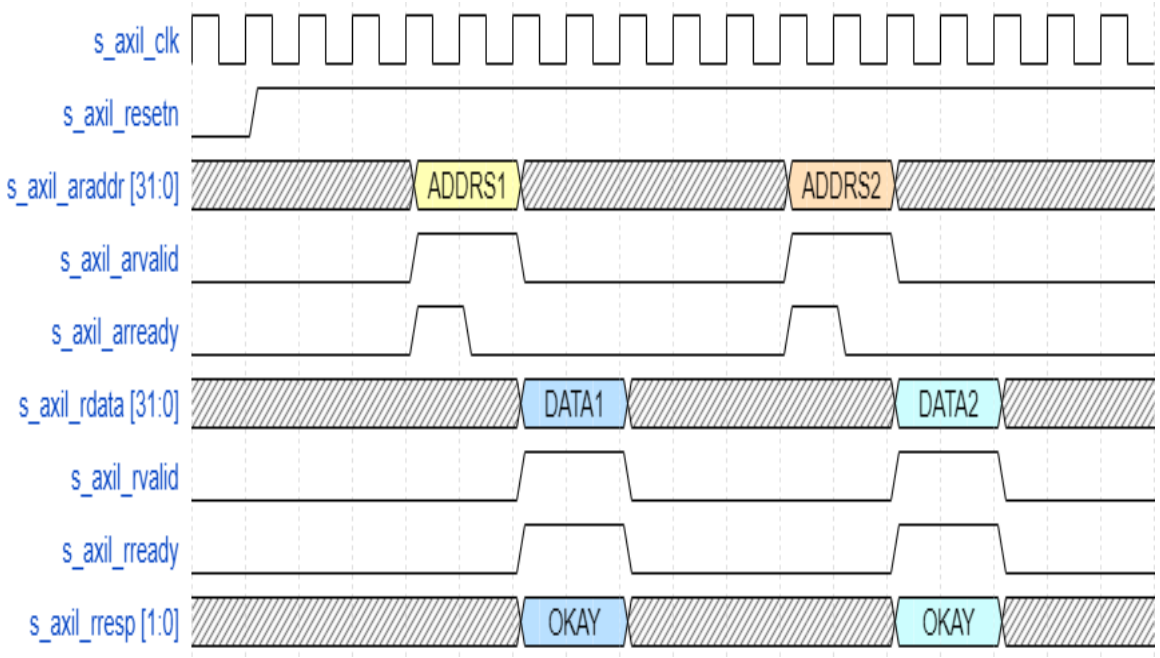

Fig. 4.5 AXI4-lite Read Channel Waveform

4.3 SDRAM Controller

As previously seen in section 3.1, SDRAM has a standard interface composed of several control signals (CAS, RAS, WE and CS) and an address, bank, data and mask buses. The SDRAM controller is built on top of that and provides a simpler interface which consist of an address, both input and output data buses, a RW signal and a set of valid and ready signals to control the transactions, in a similar way as the handshake in the AXI4-lite bus works. As seen in figure 4.1 the SDRAM controller module is composed of various blocks: the Refresh controller, the asynchronous FIFO, the address generator, the data generator and a Finite state machine that controls the whole operation.

4.3.1 FSM for SDRAM Controller

The operation of the SDRAM controller is controlled by a FSM, which is shown in figure 4.1. In total there are10 states:

Init Delay: In this state the controller waits for the initial delay of 100us required to Initialize the SDRAM. This is done by means of an internal counter, once the counter limit is reached it jumps to the Precharge state. While in this state the controller keeps sending the No Operation command. This is the default state after a reset.

Precharge: This state sends the Precharge command to the SDRAM, after that it goes into the Wait state to wait for the SDRAM to do complete the operation. From there it will resume to the Init Precharge, Refresh or Activate state depending on the operation being performed by the controller.

InitRefresh: During the initialization process the SDRAM needs to be refreshed 2 times. In this state the refresh command is sent to the SDRAM and then goes into wait state. After waiting the state will be resumed back to InitRefresh. An internal counter keeps track of the number of refreshes.

On the second refresh, it will jump again to Wait state, but this time it will resume in the Load Mode Register state.

Load Mode Register: This state configures the mode register of the SDRAM. After that if goes to Wait state to wait for the required delay. After the delay it will resume to Idle state, and the SDRAM will be ready to take requests.

Wait: It's an auxiliary state used to wait for the delays required by the different commands of the SDRAM. It has an internal counter whose limit is updated with the right value each time the controller jumps into this state, at the same time it is also specified the state to which the controller should jump after the waiting period. Then when the internal counter reaches the limit the controller resumes into the programmed state.

Idle: This is the state where the controller will remain after initialization, it remains in this state until a refresh request or a user request are received. Once a request is received it jumps to the Precharge state.

Refresh: In this state the Refresh command is sent to the SDRAM. After that it goes to Wait state and from there it will resume to Idle state.

Activate: The controller reaches this state when a user request is being processed. In this state the Activate command is sent to the SDRAM. After that jumps to the Wait state and resumes to Write or Read state depending on the type of request being performed.

Write: In this state the controller sends the Write command to the SDRAM. In a burst transfer an internal counter keeps track of the number of bursts. The controller remains in this state until the last burst of data is sent, then goes to Idle state.

Read: This state sends the Read command to the SDRAM. Then it goes into Wait state to wait for the CAS latency, then it will come back to the Read state. Similarly, as with the Write state, an internal counter keeps track of the bursts. When the last burst is received, it goes into Idle state.

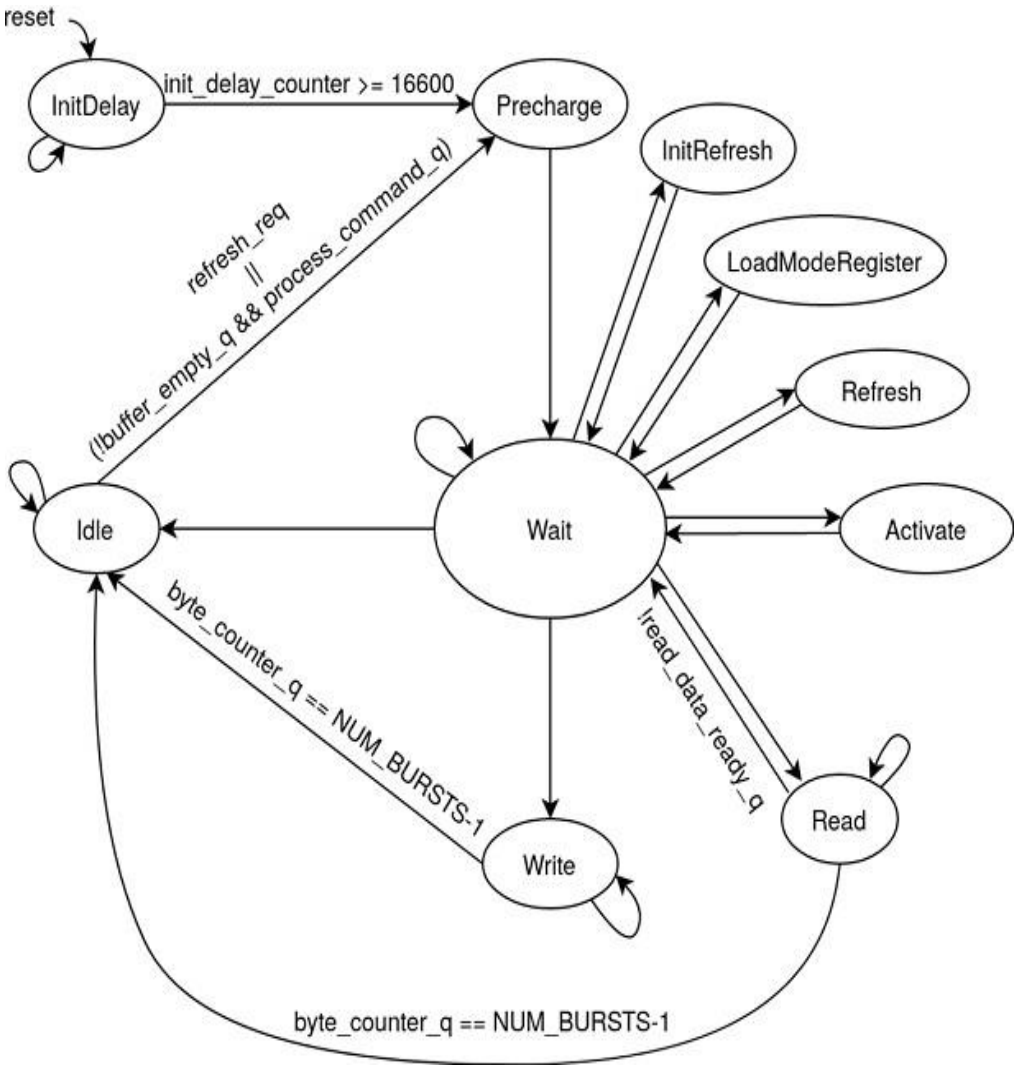

Fig 4.6 FSM FOR SDRAM CONTROLLER

CHAPTER 5

VERIFICATION

This report provides an overview of the verification plan designed for an AXI4 to SDRAM interface, utilizing a UVM- based environment. It outlines the key components, strategies, and goals of the verification effort. The following components are present in the proposed verification environment.

5.1 Verification Environment for AXI Lite SDRAM Controller Interface

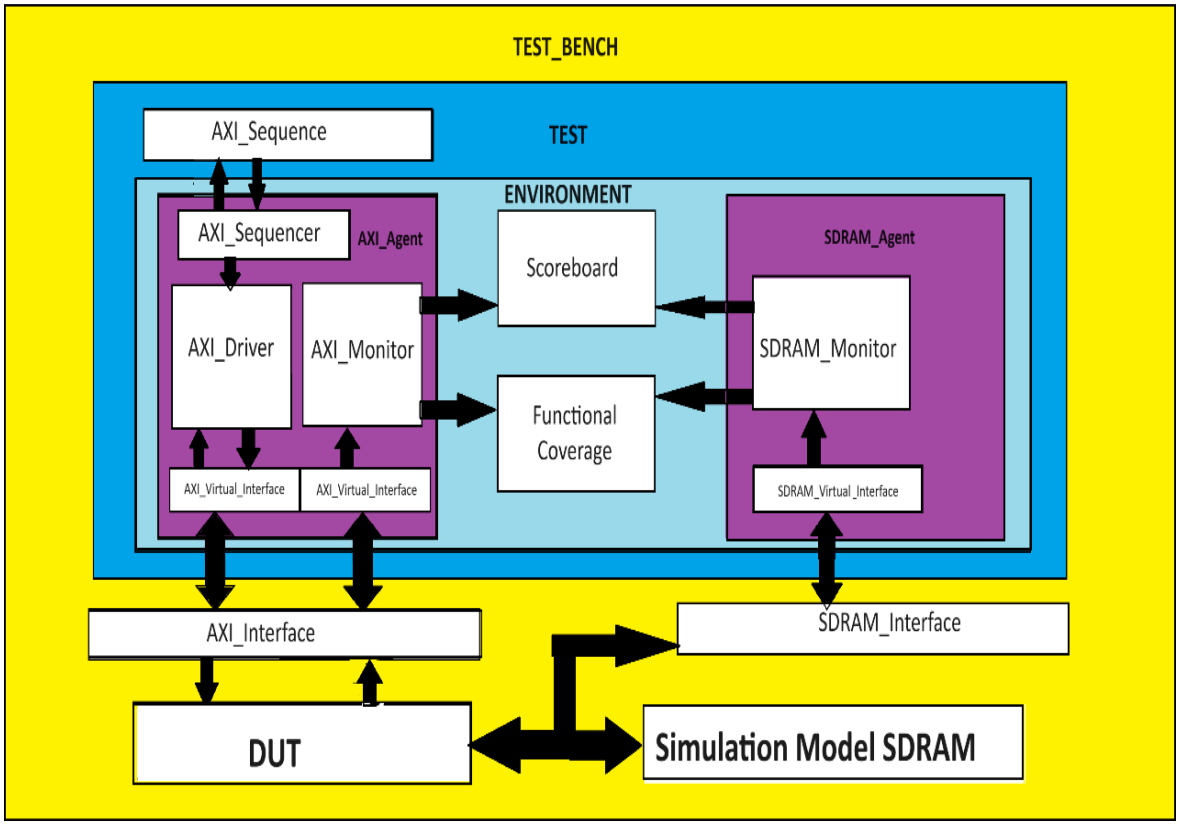

Fig. 5.1 Testbench Environment

5.2 UVM Testbench Topology

```
# ----------------------------------------------------------------------
  Name                      Type                    Size          Value
# ----------------------------------------------------------------------
# uvm_test_top              AXI_Burst_Write_Test    -             @478
# env                       AXI2SDRAM_Environment   -             @485
# axi_agt                   AXI_Agent               -             @496
# axi_drv                   AXI_Driver              -             @666
# rsp_port                  uvm_analysis_port       -             @681
# seq_item_port             uvm_seq_item_pull_port  -             @673
# axi_mon                   AXI_Monitor             -             @689
# item_collected_port       uvm_analysis_port       -             @696
# axi_seq                   AXI_Sequencer           -             @557
# rsp_export                uvm_analysis_export     -             @564
# seq_item_export           uvm_seq_item_pull_imp   -             @658
# arbitration_queue         array                   0             -
# lock_queue                array                   0             -
# num_last_reqs             integral                32            'd1
# num_last_rsps             integral                32            'd1
# axi_sd_cvg                AXI_SDRAM_Coverage      -             @533
# analysis_imp_c            uvm_analysis_imp_port_c -             @540
# analysis_imp_d            uvm_analysis_imp_port_d -             @548
# axi_sd_scb                AXI_SDRAM_Scoreboard    -             @510
# analysis_imp_a            uvm_analysis_imp_port_a -             @517
# analysis_imp_b            uvm_analysis_imp_port_b -             @525
# sdram_agt                 SDRAM_Agent             -             @503
# sdm_mon                   SDRAM_Monitor           -             @716
# item_collected_port       uvm_analysis_port       -             @723
# ----------------------------------------------------------------------
```

5.3 Verification Components of AXIL

- AXIL_Interface
- AXIL_Packet
- AXIL_Sequence
- AXIL_Base_Sequence
- AXIL_Sequencer
- AXIL_Driver
- AXIL_Agent
- AXIL_Monitor

- ### AXIL_Interface

  Verification of AXI interfaces is crucial due to the complexity of modern digital designs and the need for reliable data transfer. The verification process ensures that the AXI interface adheres to its specifications and functions correctly under various operating conditions.

- ### AXIL_Packet

  Purpose: To encapsulate input signals for the DUT, ensuring organized and reusable test cases.

  The packet class plays a crucial role in generating, representing, and manipulating AXI4 transactions, with randomized variables providing variability and non-randomized variables ensuring adherence to the AXIL protocol specifications.

- ### AXIL_Sequence

  Purpose: To coordinate the generation of transactions in a structured manner, forming test sequences.

Created Sequences

I. Reset _Sequence

The reset sequence ensures that the DUT is properly initialized before the main verification sequences are executed, establishing a known and predictable starting point for theverification process.

II. **Single_Write_Sequence**

Write known data to a specific SDRAM address. Read back the data from the same address. verify the data. The excepted outcome will be Write and read data should match.

III. **Single_Read_after_Write_Sequence**

Verify that data written to a specific address in SDRAM can be accurately read back to ensure correct read/write operations. Ensure that the AXIL2SDRAM interface correctly handles a single read operation after a write operation.

IV. **Single_Burst_Write_Sequence**

Write a burst of data to consecutive SDRAM addresses. Read back the data from the same addresses. Verify the data. The excepted outcome will be Burst data written and read should match.

V. **Burst_Read_after_Write_Sequence**

Verify that a burst read operation correctly retrieves a range of data that was previously written to SDRAM in a burst write operation. Ensure that the AXIL2SDRAM interface correctly handles a burst read operation after performing a burst write operation.

- ## AXIL_Base_Sequence

The AXI_base_Sequence class provides a foundational structure for creating AXI transaction sequences in a UVM testbench. It includes essential methods for managing the sequence lifecycle, such as raising and dropping objections during execution.

- ## AXIL_Sequencer

The sequencer controls the flow of request and response sequence items between sequences and the driver. Sequencer is connected to a single driver so that the read and write operation are independent of each other and can happen in parallel. Sequencer and driver use TLM Interface to communicate transactions.

- **AXIL_Driver**

  The driver receives transactions from sequences. These transactions represent various AXIL operations such as reads, writes, or other control operations.

  This translation involves mapping the transaction attributes (e.g., addresses, data) to the corresponding AXIL signals (e.g., AWADDR, WDATA) and controlling the timing and sequencing of these signals according to the AXIL protocol specifications. The driver ensures that signals are driven onto the interface with the correct timing and sequencing to emulate the behavior of an AXIL initiating transactions with the DUT**.**

- **AXIL_Monitor**

  The AXI Monitor class is a comprehensive implementation designed to observe and capture AXIL transactions in a UVM testbench. It effectively translates low-level signal activity into higher-level transaction packets, which can be further analyzed or checked against expected behavior. This monitor serves as a critical component in verifying the correctness of AXIL-based designs by ensuring that all relevant transactions are captured and reported correctly.

- **AXIL_Agent**

  AXIL_Agent comprises of the AXIL_Sequencer, AXIL_Driver and the AXIL_Monitor of an interface. Multiple agents could be used to drive multiple interfaces, and they are all connected to the test-bench through the environment component. This structured approach enhances the overall quality and reliability of the verification process in complex digital systems.

5.4 Verification Components of SDRAM

- SDRAM_Interface
- SDRAM_Monitor
- SDRAM_Agent

- **SDRAM_Interface**

  SDRAM interfaces allow processors and other devices to communicate with SDRAM memory chips. The interfaces handle the necessary SDRAM commands, timing and signaling to read and write data to the SDRAM.

- **SDRAM_Monitor**

  The SDRAM monitor continuously samples the SDRAM interface signals, such as command, address, and data signals. It decodes the SDRAM commands (e.g. ACTIVATE, READ, WRITE, PRECHARGE, REFRESH) from the sampled signals. The monitor captures the SDRAM bank, row, and column addresses along with the read/write data.

  The transaction object is sent to the scoreboard's analysis port for comparison with the expected results from the AXI side. The verification environment can capture the actual SDRAM transactions and compare them against the expected results from the AXI side. This helps in identifying any mismatches or issues in the AXI to SDRAM interface logic.

- **SDRAM_Agent**

  The SDRAM agent receives AXI transactions from the AXI agent's sequencer. The SDRAM monitor captures the SDRAM transactions and sends the data to the scoreboard.

  The scoreboard compares the expected results from the AXI side with the actual SDRAM transactions to verify the correctness of the AXI to SDRAM interface**.**

5.5 Verification Components of Testbench Top

- AXIL_SDRAM_Scoreboard
- AXIL_SDRAM_Coverage
- AXIL_SDRAM_Environment
- AXIL_SDRAM_Test
- AXIL_SDRAM_Pkg
- Tb_top

- **AXIL_SDRAM_Scoreboard**

The scoreboard should have two queues - one to store expected results from AXIL transactions and one for actual results captured from the SDRAM interface.

The scoreboard needs to receive transactions from two places:-

- From the AXIL monitor, which captures AXIL read/write transactions.
- From the SDRAM monitor, which observes the SDRAM interface.

By implementing the scoreboard in this way, it will continuously compare the expected AXIL transactions against the actual SDRAM transactions, reporting any mismatches. The scoreboard receives transactions from the AXIL and SDRAM monitors through the analysis ports.

- **AXIL_SDRAM_Coverage**

The coverage collector should have cover groups to sample the important aspects of AXIL and SDRAM transactions.

For AXI transactions, the coverage should include:

- AXIL burst types
- AXIL burst lengths
- AXIL data widths
- Combinations of read/write bursts

The coverage should also include combinations of AXIL and SDRAM transactions such as:

- AXIL write followed by SDRAM activate, write
- AXIL read followed by SDRAM activate, read

- **AXI_SDRAM_Envirnoment**

The environment should encapsulate the AXIL agent, SDRAM agent, scoreboard, and coverage collector. The environment connects the analysis ports of the AXIL and SDRAM monitors to the scoreboard and coverage collector. The AXIL agent's sequencer will drive AXIL transactions that will be converted to SDRAM commands by the SDRAM agent. The scoreboard and coverage collector will monitor the transactions and check for correctness and coverage.

- **AXIL_SDRAM_Test**

AXIL_SDRAM_Test is a top-level component that encapsulates all the test-specific information. The functions of the test component are- instantiating the environment, configuring it and invoking. In test benches where there are multiple focus tests, a base test is first written extending from the uvm_test class that instantiates the top-level environment and all the other focus tests are extended from this base test for more specific testing.

- **Tb_top**

The Tb_top instantiates the DUT and the AXIL_SDRAM_Test and configures their connections. This is important as most of the times, the AXIL_SDRAM_Test is passed at run-time and needs to be dynamically instantiated. Since all the tests are registered with UVM Factory, the test-bench can instantiate any test that is already registered.

CHAPTER 6

Simulation Results

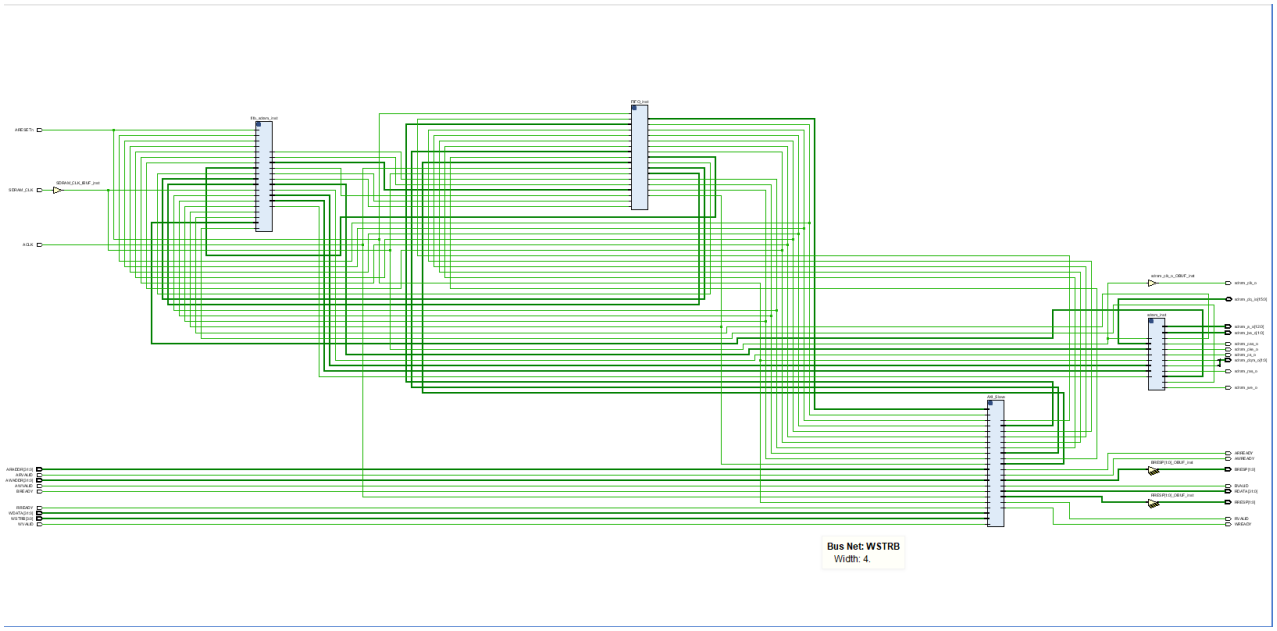

Fig 6.1 Elaborated Diagram

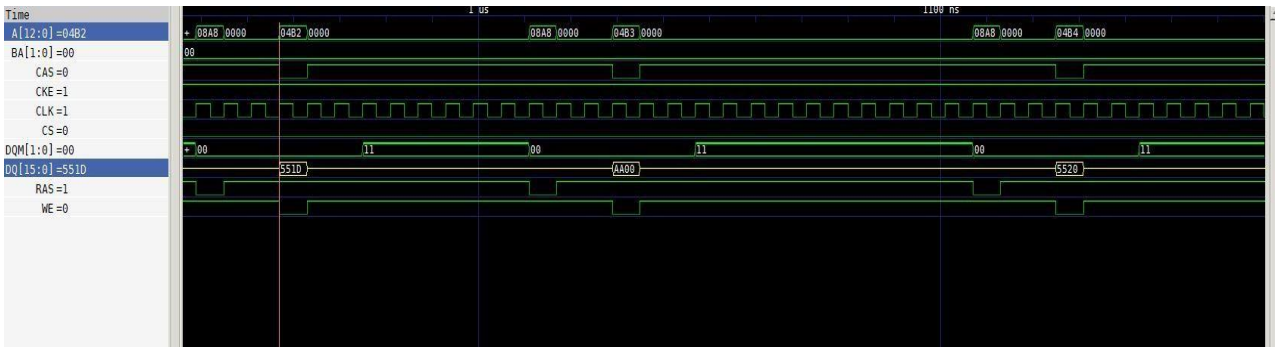

Fig 6.2 Write operation

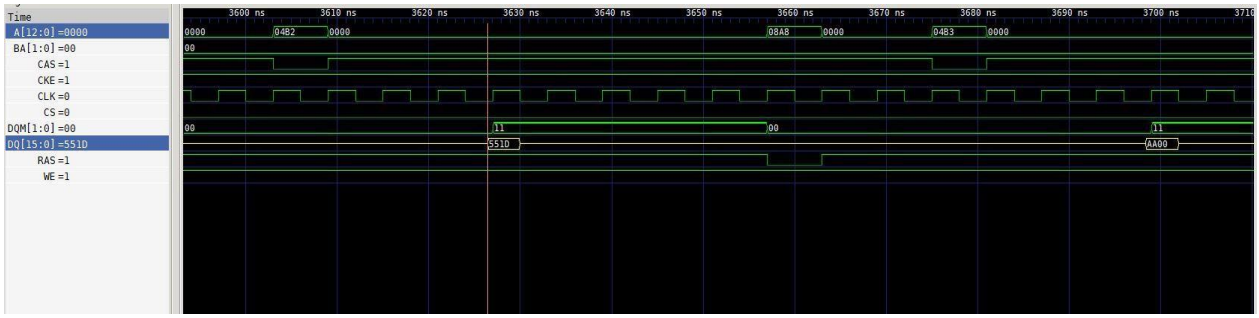

Fig 6.3 Read operation

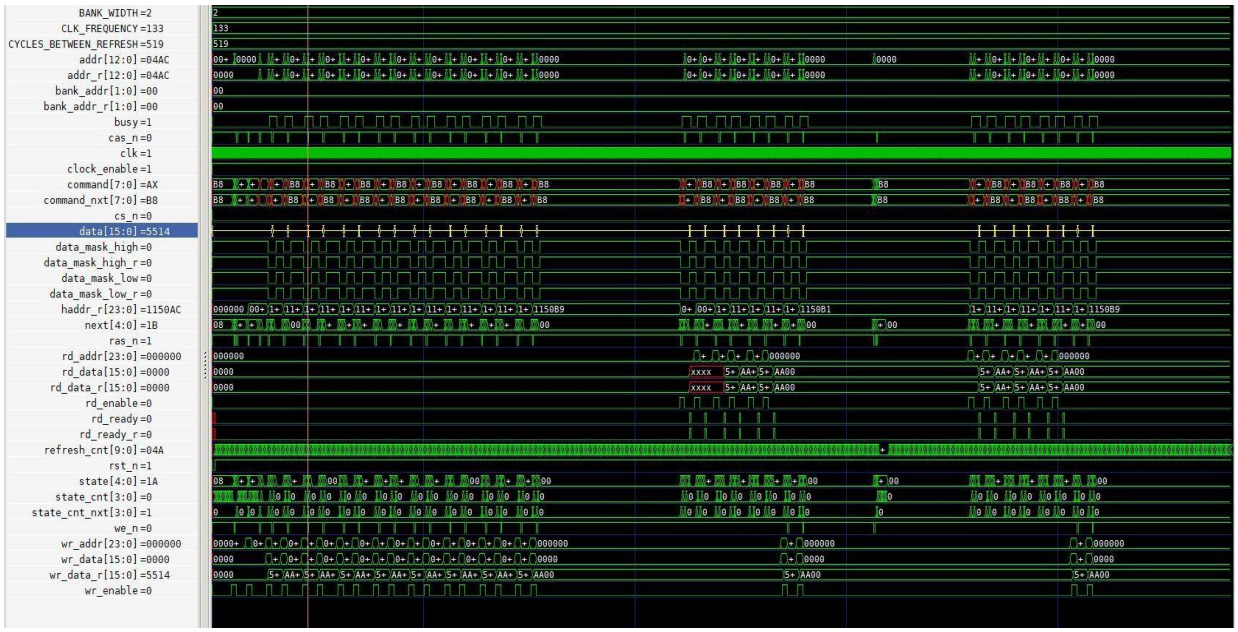

Fig 6.4 SDRAM_controller_read_write

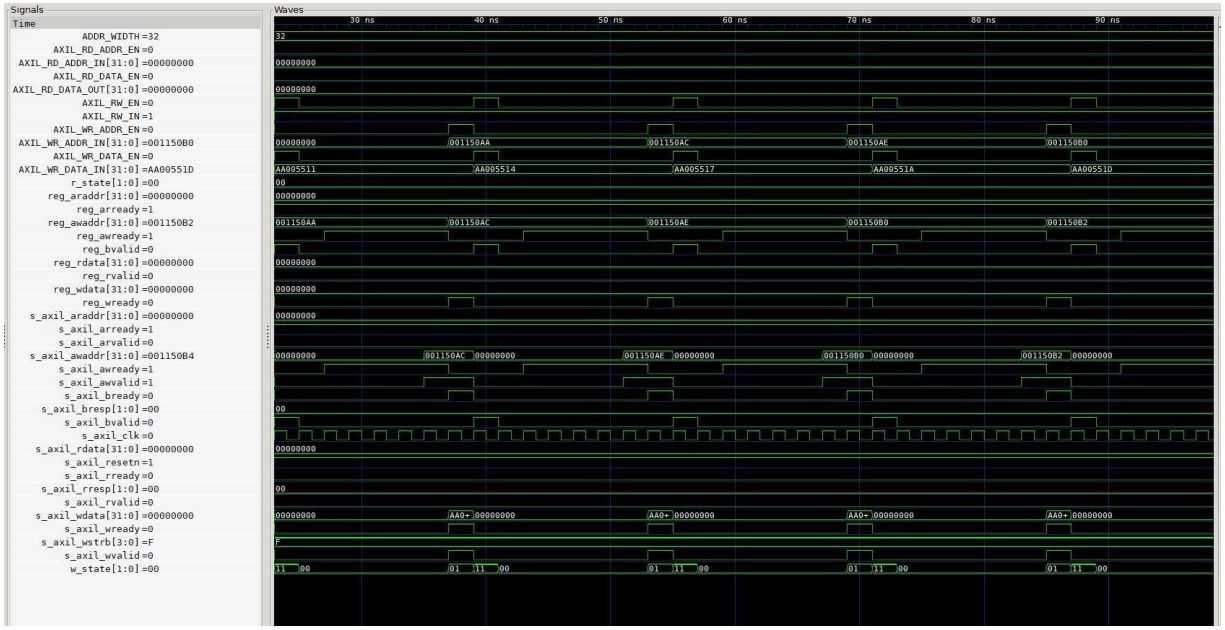

Fig 6.5 AXI4_Lite_transaction

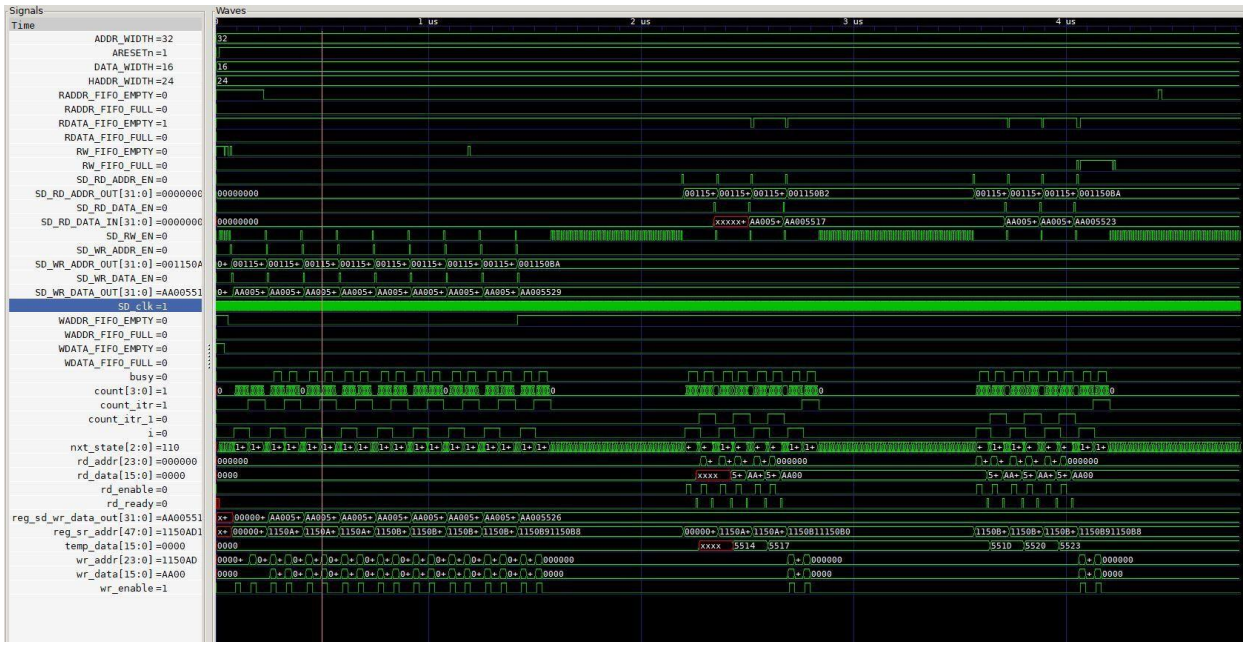

Fig 6.5 fifo_sdram_controller_interface

CONCLUSION

The main goal of this project, the design of an AXI-SDRAM interface IP has been successfully completed. A significant effort was put into the verification, since the first priority was to obtain a functional design that can actually be integrated into an ASIC as the final purpose is to use this IP in a future version of the DRAC SoC.

Once the standalone AXI-SDRAM controller has been successfully verified, it has been integrated with the pre DRAC SoC to verify the whole system. After some iterations to f ix design errors the design successfully passed all the tests. The designed AXI-SDRAM interface was meant to replace the current interface to external memory to improve the performance of the system. This has been achieved and a significant increase on the performance of the access to memory has been obtained. But it has to be taken into account that the current tests performed are meant to verify the proper functionality of the design and are not meant to be used as performance metrics.

An extra step of power analysis has been done. At this point, there are no defined power requirements but this is usually one of the major concerns. From the results obtained, there is little difference between the power consumption while it is operating and when it is in idle. The main power contribution comes from the clock. As a further step, low power techniques could be applied to minimize the power, being clock gating a good possible option to reduce the power.

FUTURE SCOPE

- Implementation of pipeline in the SDRAM controller to increase the bandwidth.

- Making the AXI4-lite-SDRAM controller configurable. The current version the controller is parametrizable, which means the parameters can be set during instantiation. In future versions it is expected to make it configurable, so the internal timing parameters are configured with internal programmable registers.

- Revision and improvement of the verification setup. Expected additional features will require more testing. Then future work also includes revising the current verification setup to test the new features.

- FPGA implementation of the whole pre DRAC SoC with the AXI4-lite-SDRAM controller. This requires additional work since the design doesn't fit in the FPGA used in this 64project, then another FPGA will have to be used. Higher end FPGA boards don't include SDR SDRAM, then an additional PCB with the required memory has to be previously designed.

- Analysis of low power techniques such as clock gating to reduce power consumption of the design. As it has been mentioned, power consumption could be probably reduced, but it requires further analysis and testing to ensure this is done without affecting the functionality of the design.

REFERENCES

https://www.arm.com/architecture/system-architectures/amba

https://documentation-service.arm.com/static/5f915920f86e16515cdc3342?token=

https://developer.arm.com/documentation/ihi0022/latest/

MT48LCxxM4/8/16A2 Datasheet by Micron Technology Inc. | Digi-Key Electronics (digikey.com)

https://upcommons.upc.edu/handle/2117/192250