Verification of a DFI-Based DDR4 PHY Bridge for DDR PHY Modules

A Project report submitted in partial fulfillment of the requirement for the Award of the Degree of

MASTER OF TECHNOLOGY

in VLSI DESIGN

Submitted By

Shitansh Kumar Rai

602362034

Under Supervision of

Dr. Ravi Kumar

Associate Professor

Under Mentorship of

Saikiran Aowsula

Design Verification Engineer

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

THAPAR INSTITUTE OF ENGINEERING & TECHNOLOGY

(A DEEMED TO BE UNIVERSITY), PATIALA, PUNJAB

June 2025

CERTIFICATE

Regd. Office:
Intel Technology India Private Limited
23-56P, Outer Ring Road,
Devarabeesanahalli, Varthur Hobli website: www.intel.in
Bellandur Post
Bangalore 560 103, India
CIN-U85110KA1997PTC021606

Tel: +91-80-2605 3000
Fax: +91-80-2605 6190

To Whomsoever It May Concern

WWID: 12250374

Employee Name: Shitansh Kumar Rai

Internship Dates: 24/06/2024 to 23/06/2025

The letter is to confirm the mentioned above has undergone internship at Intel Technology India Private Limited.

We wish you all the best for your future assignments.

Yours Sincerely

Gowre Saseedaran

Date:

Place: **Bangalore**

DECLARATION

I, **Shitansh Kumar Rai** hereby declare that the work presented in this thesis entitled **"Verification of a DFI-Based DDR4 PHY Bridge for DDR PHY Modules"** in partial fulfillment of the requirement for the award of degree of **Master of Technology (VLSI Design)** submitted at Electronics & Communication Department**,** Thapar Institute of Engineering & Technology (Deemedto be University), Patiala is an authentic record of work carried out under the supervision of **Dr. Ravi Kumar** (Associate Professor, Electronics & Communication Department, Thapar Institute of Engineering &Technology)

Date:

Shitansh Kumar Rai

602362034

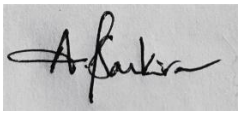
Ravi Kumar

Dr Ravi Kumar

Associate Professor

ECE Department

TIET Patiala



Saikiran Aowsaul

Design Verification Engineer

Intel India Pvt. Limited

ACKNOWLEDGEMENT

The task could not have been completed without acknowledging those who guided and supported me continuously to make efforts successful. Taking this opportunity, I expressmy deepest gratitude and respect to my supervisor, **Dr. Ravi Kumar,** Associate Professor, Department of Electronics and CommunicationEngineering, Thapar Institute of Engineering and Technology for his guidance and encouragement throughout this project

Shitansh Kumar Rai

602362034

ABSTRACT

In this paper we introduce a high-performance DDR4 SDRAM DDR PHY Training Unit design, this design is independent of the SOC and will only work with the PHY for its training and will provide a better handshake mechanism between PHY and DDR PHY Training Unit. Our DDR PHY Training Unit is able to perform DRAM initialization, refresh, and calibration. Its design is extensible, allowing for further development of other types of DDR4 memory controllers and adaptation for various DDR4 speed grades. The development process involved creating the DDR PHY Training Unit's logic blocks in RTL from the ground up. Standalone verification of each designed module was conducted, followed by the coverage plane was created to do the complete validation of the entire integrated product. To evaluate the performance of our DDR PHY Training Unit, we conducted extensive assessments using both EEMBC benchmarks and synthetic benchmarks in simulation. These evaluations provide a comprehensive comparison of their performance across various scenarios, offering valuable insights for further developments in the field.

TABLE OF CONTENTS

LIST OF FIGURESSS

LIST OF TABLES

CHAPTER 1: INTRODUCTION

The fabrication of Integrated Circuits (ICs) by integrating billions of transistors on a single silicon die is known as Very Large Scale Integration (VLSI). Digital designers generally describe the behavior and functionality of these ICs using hardware description languages (HDLs) such as Verilog and SystemVerilog [12]. VLSI technology first emerged during the 1970s, enabling the development of early microprocessors, communication ICs, and complex semiconductor devices. Before the advent of VLSI, most ICs were limited to performing fixed, relatively simple functions. The introduction of VLSI made it possible to integrate diverse components—including RAM, ROM, flash memory, EEPROM, multiple processor cores, and custom logic blocks—within a single chip, significantly increasing functionality and performance.

A typical VLSI design follows a systematic flow tailored for applications like System on Chip (SoC), Application-Specific Integrated Circuits (ASICs), and large-scale Field Programmable Gate Arrays (FPGAs). This flow consists of several discrete stages supported by Electronic Design Automation (EDA) tools, guiding the design from specification to final silicon. The process starts with defining functional specifications and proceeds through high-level design, RTL coding, functional verification, logic synthesis, placement and routing, gate-level simulation, fabrication, and finally, post-silicon validation [13]. Over the years, the entire IC implementation flow—from RTL to GDSII layout—has benefited significantly from Moore's Law[1], which has driven continuous scaling and performance improvements.

In addition, FPGA prototyping and emulation play a critical role in the verification process by enabling early integration of hardware and software components. These techniques help engineers test functionality under real-world conditions, reduce development cost, and mitigate project risk [14]. Throughout the design cycle—which spans pre-silicon design, prototyping, and post-silicon validation—designs often require retargeting and updates to adapt to changing requirements or constraints. This iterative process demands specialized tools, domain expertise, and careful exploration of design trade-offs, which do not always guarantee perfectly predictable outcomes.

To address these challenges and achieve functional verification closure, designers often rely on emulation and FPGA prototyping technologies. Emulation systems automatically map a design's RTL representation into an internal programmable gate array, supporting comprehensive verification of both hardware and software aspects of the system. This approach facilitates early hardware-software co-development and optimizes resource utilization by sharing design tools, processes, and engineering expertise across multiple development phases.

DDR4 and DDR PHY Training Unit Background?

This chapter furnishes the essential background information needed for the subsequent chapters. Initially, we delve into the specifics of DDR SDRAM memories in Section 1.1, with a focus on DDR4. We explore the novel features introduced in DDR4 that were absent in DDR3. After establishing a foundational understanding of DDR4, we shift our focus to DDR PHY DDR PHY Training Unit in Section 1.2. In this section, we delve into the critical attributes of DDR PHY Memory Controller. Finally, in Section 1.3 we discuss related works

1.1 DDR4 BACKGROUND:

Due to its favorable cost-per-bit, Double Data Rate Synchronous Dynamic Random-Access Memory (DDR SDRAM), has consistently been preferred for designing main memory subsystems. Notably, the cost-per-bit of DRAM has been diminishing as process technology scales, allowing significantly more DRAM cells to be integrated within the same die area. For this reason, DRAM has been extensively used in many computer systems. In contrast to the ongoing reduction in cost, the latency of DRAM has remained relatively constant. We commence by providing essential background details on DDR. These memories transfer data at both rising and falling edges, which is why they are termed DDR (Double Data Rate). There have been multiple generations of DDR, spanning from the 1st to the 5th. As DDR1 and DDR2 are no longer in common use, we will begin by outlining the features of DDR3. The DDR3 memory system is organized hierarchically, consisting of channels, ranks, and banks and its clock rate ranges from 400 MHz to 1033 MHz. However, the more recent DDR4 generation, introduced in 2013, presents a higher clock rate ranging from 800MHz to 1600MHz while exhibiting lower power consumption. Furthermore, DDR4 introduces an additional address level called bank group, where banks within the same bank group are interdependent. This implies that accessing bank B0 within bank group BG0, denoted as access A, incurs larger timing constraints on subsequent accesses to all banks of bank group BG0 compared to accesses to the banks in other bank groups. The latest generation, DDR5, unveiled in 2020, boasts significantly increased density compared to its predecessor, DDR4. DDR5 features up to twice the number of bank groups as DDR4 and introduces two distinct 32-bit channels compared to DDR4, which provides only one 64-bit channel. Additionally, the fastest DDR5 variant, with a speed grade of 5600 MT/s, exceeds the speed of the fastest DDR4, which operates with a speed grade of 3200 MT/s, by more than 1.5 times. In the rest of the chapter, we focus on DDR4 as our DDR Phy Training Unit designs targets this DDR generation.

To store data into memory, a user typically supplies both an address and the corresponding data, whereas reading data requires only providing the address. In contemporary virtual memory architectures, this user-supplied address is referred to as the logical address. Prior to accessing DRAM, this logical address undergoes translation into a physical address by the memory management system. The memory controller then interprets this physical address by dividing it

into fields corresponding to bank, bank group, row, and column addresses [22], [23]. These subdivisions collectively determine the precise memory location for read or write operations. As illustrated in Figure 2.2, decoding the row and column address fields enables the system to locate specific data.

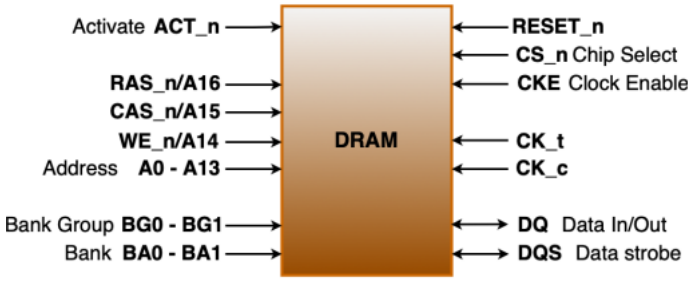


Figure 1.1: DRAM Top Level Diagram

After identifying the correct bank and bank group, the row component of the physical address activates a specific word line within the DRAM array. Activating this word line transfers data from the memory cells into sense amplifiers, while the bit line's width corresponds to the column size. Subsequently, the column address determines which part of the word stored in the sense amplifiers is accessed. DRAM devices are commonly categorized by column widths such as ×4, ×8, or ×16, which directly indicate the device bit width.

Features	DDR SDRAM	DDR2 SDRAM	DDR3 SDRAM	DDR4 SDRAM
Clock Frequency	100 / 133 / 166 / 200 MHz	200 / 333 / 400 MHz	400 / 533 / 667 / 800 MHz	800 / 1600 / 1866 / 2133 MHz
Clock Input	Differential clock	Differential clock	Differential clock	Differential clock
I/O Width	x4 / x8 / x16 / x32	x4 / x8 / x16	x4 / x8 / x16	x4 / x8 / x16
Data Strobe	Single data strobe	Differential data strobe	Differential data strobe	Differential data strobe
Prefetch	2 bit	4 bit	8 bit	8 bit
Supply Voltage	2.5 V	1.8 V	1.5 V	1.2 V
On Die Termination (ODT)	Unsupported	Supported	Supported	Supported
Burst Length	2, 4, 8	4, 8	8, 4 (Burst chop)	8, 4 (Burst chop)
CAS Latency (CL)	2, 2.5, 3 clock	3, 4, 5 clock	5, 6, 7, 8, 9, 10 clock	5, 6, 7, 8, 9, 10 clock

Table 1.1: Comparing DDR family

Due to the complexity and the need for interoperability across different designs and vendors, DRAM technologies are governed by industry standards developed by the JEDEC Solid State Technology Association [7]. For instance, the JEDEC JESD209-4B specification defines the

LPDDR4 standard, detailing key aspects including functionality, electrical and timing characteristics, package configurations, and signal ball assignments to ensure consistency and compatibility within memory systems.

DDR4 memory utilizes an 8-bit prefetch architecture combined with parallel bank access to achieve higher data transfer rates. The architecture is organized into 16 banks, which are further grouped into 4 bank groups, each containing 4 banks. Within this structure, the row and column addresses determine the exact bank where the requested data resides. Each bank itself is an independent memory array, and a "page" in DDR4 is defined by a unique combination of bank group, bank number, and row address. The page size corresponds to the number of bits per row; for example, if the column address is 10 bits wide, each row consists of 1024 bit lines. At the top of the logical hierarchy is the "rank," which serves to expand memory capacity [19]. For instance, a single 16 GB memory die equipped with one chip select signal (CS_n) is classified as Single-Rank. If the same total memory is organized into two 8 GB units, it becomes Dual-Rank, and when split into four 4 GB units, it is referred to as Quad-Rank—this practice is also known as depth cascading.

The DDR4 command bus uses six primary control signals to coordinate memory operations. Memory access typically starts with an activate (ACT) command, which is then followed by either a read (RD) or write (WR) command [22]. Both read and write operations can occur as single transactions or as bursts. In single write mode, each data word requires a separate address [30]. Conversely, in burst mode, the controller issues a single starting address, and the memory continues reading or writing sequentially for a fixed burst length of eight or a chopped burst length of four [31]. All these operations are synchronized to the rising edge of the clock.

The data access process itself consists of two main steps: first, the ACT command opens a specific row within a bank, making it available for subsequent read or write operations. This step is known as the Row Address Strobe (RAS). Next, during the Column Address Strobe (CAS) step, the registered column address bits are matched to complete the access operation, coinciding with the RD or WR command. As illustrated in Figure 2.4, the DDR4 command state diagram includes various other commands and state transitions such as precharge, preselect, no operation (NOP), refresh, and mode register set (MRS). The precharge (PRE) command is responsible for closing an active row, while auto-precharge features—implemented through write with auto precharge (WRA) and read with auto precharge (RDA)—allow rows to be deactivated automatically once the data transfer completes, thereby simplifying memory management. Table 2.3 provides a detailed truth table describing the conditions and effects of each DDR4 command.

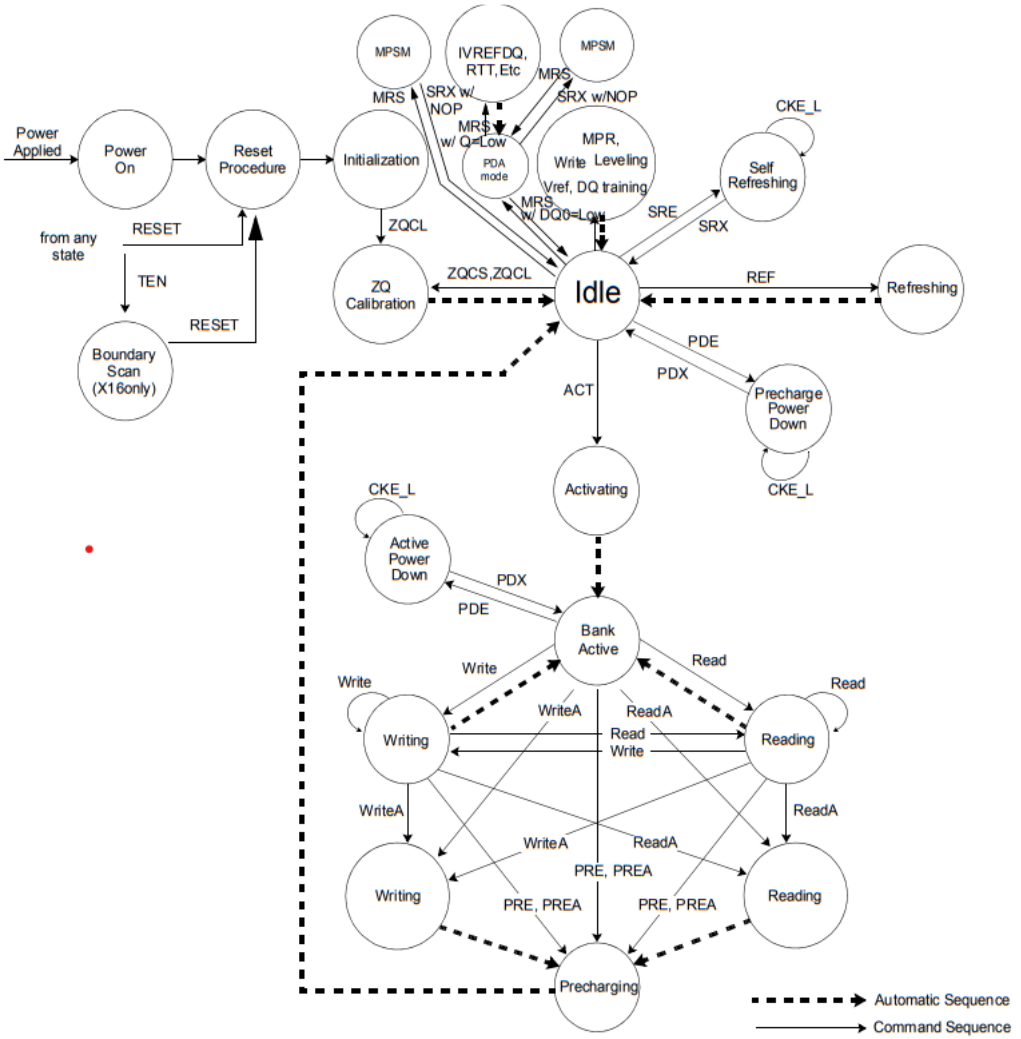


Figure 1.2: State Diagram of DDR4 (Source: [11], p. 35)

Function	Code	CS_n	ACT_n	RAS_n	CAS_n	WE_n	AP
Refresh	REF	L	H	L	L	H	H or L
Write	WR	L	H	L	L	L	L
Read	RD	L	H	L	L	H	L
Bank Activate	ACT	L	L	row address			
Single Bank Precharge	PRE	L	H	L	H	L	L
Write with Auto-Precharge	WRA	L	H	L	L	L	H
Read with Auto-Precharge	RDA	L	H	L	L	H	H

Table 1.2: DDR Command table (Source: [11], p. 35)

1.2 DDR4 STRUCTURE:

In general, a DRAM device is like a three-dimensional array, with its levels named bank, row, and column. A set of DRAM devices is called a rank. The DRAM chip always consists of 16 or 8 banks which can be accessed at the same time while sharing the same command and data bus. Inside each bank, there is a two-dimensional array of memory cells each storing 1 bit. In addition, there is a row buffer within each bank that temporarily stores the content of the most

recently accessed row of that bank. Once the data is placed into the row buffer, subsequent read and write commands to that bank can be performed quickly, and this reduces the latency associated with accessing different columns sequentially. Also, utilizing a row buffer reduces energy consumption as activating a row consumes more energy compared to column access. Additionally, DDR4 leverages parallelism in bank groups to reduce the latency of a request and to significantly increase the overall performance of the system. Each bank group is assigned to one DDR4 memory chip, and within each memory chip, there are four banks. There are three models of DDR4 chips namely x4, x8, and x16. The number represents the output data width of the chip, 4, 8, and 16 bits, respectively. Models x4, and x8 have 4 bank groups and 16 banks in total while model x16 has 2 bank groups and 8 banks in total. In Figure 1.1, the bank group structure of DDR4 is depicted. Only one bank will be activated during any read and write operations. After the data is amplified through sense amplifiers, it will be connected to Global IO Gating using Local IO Gating. The width of the Data I/O can be 4, 8, or 16 bits.

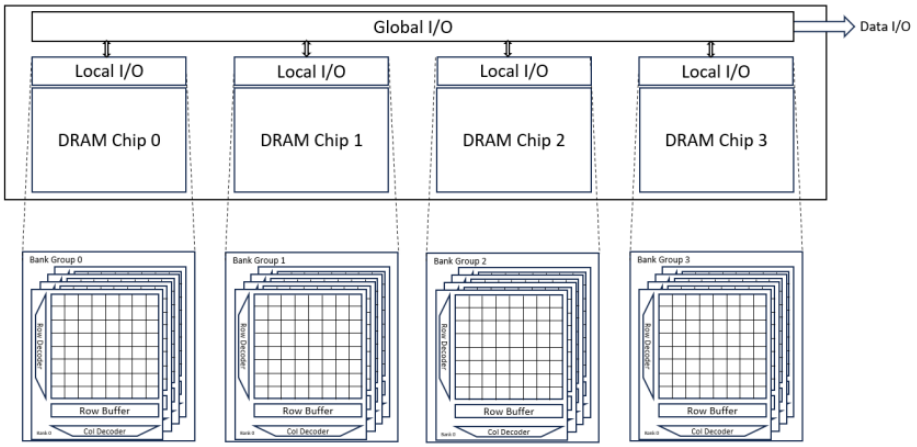


Figure 1.3: Bank group organization in x4/x8 model. (Source: [4], p. 14)

1.3 DDR4 ACCESS PROTOCOL:

There are various pins on the DDR4 memory chip. Pins CAS, RAS, and ACT are used to encode different types of commands. Moreover, pins RAS and CAS are used as address bit 16 and 15 respectively. Additionally, there are separate address pins (A) covering bits 0th to 14th. These pins are used to determine the row and column address. Furthermore, bank groups and banks are assigned through dedicated pins called BG, and BA, respectively. The DRAM operates in two separate clock domains. The main differential clock pin (clk) serves as the reference for command transfer between the DDR PHY Training Unitand the memory module. In contrast, the DQS pin acts as a separate clock used as a reference for transmitting data to/from the memory module, with the actual data passing through the DQ pins. on DDR memories data is transferred both on the positive and negative edge of DQS, and that is where the double data rate word comes from. Although both clk and DQS operate at the same frequency, they are not synchronous, and over time, they may drift away from each other

The temporary data stored in the row buffer is sent back to the respective memory cells within the activated row. This phase is essential for preserving data integrity. In addition to PRE, there is a particular command which applies PRE to all banks within the memory, called PREA.

In the subsequent phase, referred to as the activation phase, when the ACT command is given along with the row address, the designated row is activated, and its data is moved to the row buffer. During this phase, the entire row is transferred to the row buffer because subsequent read or write operations might involve different column addresses within the same row. Requests that target the currently activated row within a bank are termed “open requests”, while those targeting rows different from the currently activated row are referred to as “close requests.”

In the final phase, known as the I/O gate phase, four different commands can be issued. RD and WR commands, accompanied by a column address, are used for read and write operations, respectively. Additionally, RDA and WRA commands are available, which are equivalent to RD and WR but also include an auto-precharge function. After executing these commands, there is no requirement to initiate precharge at the start of the new request.

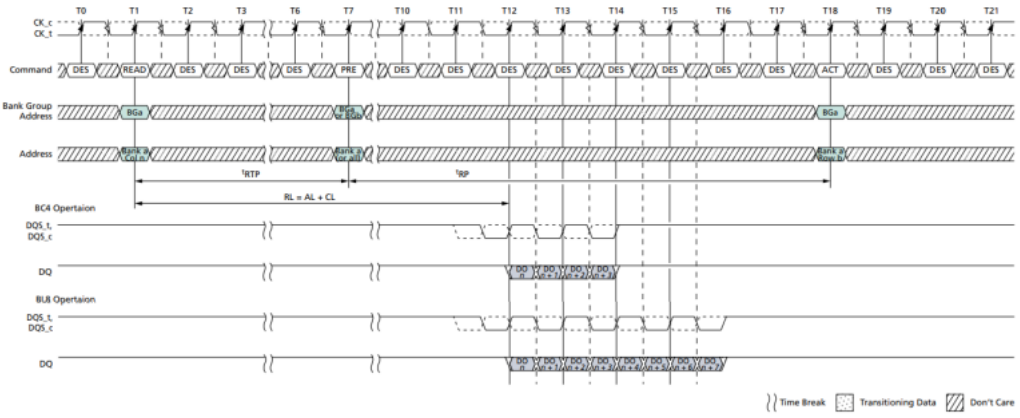


Figure 1.4: The example shows the relative timing constraint between a Read CAS, followed by PRE, and ACT targeting bank a in bank group a. (Source: [6], p. 21)

Additionally, DDR4 supports other types of commands, with one crucial command being the REF (refresh) command. This command is employed to refresh all the stored bits within all cells in the memory banks. Because DRAM essentially stores data as charges in capacitors, and these capacitors tend to lose their charge gradually over time, the refresh command plays a crucial role in preserving data integrity and ensuring the continued reliability of stored information. Furthermore, to account for voltage and temperature variations across the board, the drivers of pins in DRAM need to be calibrated periodically, as ZQCS and ZQCL are the designated commands for performing these calibration tasks.

1.4 JEDEC FAMILY:

JEDEC has introduced three groups of DDR memories that cater to different needs including data rate, power consumption, and application use cases. DDRx series are used for general computing and mostly are employed in computer systems. GDDRx is designed to provide high memory bandwidth for graphical processing units. Lastly, LPDDRx was introduced to provide a high data rate while focusing on power consumption. These series are suitable for mobile computing systems. Our DDR PHY Training Unitis specifically designed for DDR4 memories

1.5 PHY:

The PHY serves as the fundamental physical interface to an external DDR4 DRAM device, incorporating both hard and soft blocks crucial for ensuring the reliable operation of the physical interface. Hard blocks are designed to serialize read data from the DRAM device and de-serialize write data for transmission to the DRAM device. Additionally, the hard block includes a PLL primarily responsible for generating the clock for the DRAM device. On the other hand, the soft core handles the initialization steps necessary to transition the DRAM device into normal mode after being powered on. Furthermore, the soft block provides the calibration logic to set all delays in both hard and soft blocks to interface properly with the DRAM device.

The PHY talks to the other building blocks of the MIG through an interface called the

Native Interface. This interface neither supports reordering nor incorporates any buffers. On one side, the PHY receives/sends the data in a burst of 8 from/to the DRAM device in the DRAM's clock domain (serialization, and de-serialization respectively), and on the other side, it transmits the data in a single clock cycle in the memory controller's clock domain. For instance, for a device with data width of 8 bits, PHY receives 64-bit data from upstream (the User Interface) in a single clock at a rate of 300 MHz, while a DDR4 device receives the same data in a burst of 8, in four clock cycles at a rate of 1200 MHz.

Considering that the maximum frequency achievable on implemented logic blocks on FPGAs is approximately 300 MHz, the DDR PHY Training Unitlogic in MIG operates with a DRAM to a system clock ratio of 4:1. For instance, if the DDR4 memory speed grade is 2400 MT/s, given that DDR4 is a double data rate memory, the command frequency effectively becomes half of the speed grade, which amounts to 1200 MHz. This indicates that commands should ideally be issued at a frequency of 1200 MHz, a frequency that is considerably high for logic implementations on FPGAs. Given the 4:1 ratio, the DDR PHY Training Unitwould operate at 300 MHz. Consequently, MIG issues a pack of four commands in a single clock cycle to the PHY, resulting in the DRAM device receiving a single command with a frequency of 1200 MHz

1.6 DDR PHY TRAINING UNIT:

Building upon the foundational knowledge of DDR4 background outlined, we now delve into the details of DDR PHY Training Unit design. Specifically, in the following sections, we introduce a widely employed architectural framework. A DRAM DDR PHY Training Unit serves as the bridge between the requestors and the DRAM memory module. Its primary responsibility is to manage access to the DRAM.

The DDR PHY Training Units designed with the aim of efficiently handling read, and write requests received from User Interface block. It accomplishes this while ensuring low latency, adhering to all JEDEC protocol and timing requirements, and utilizing minimal FPGA resource

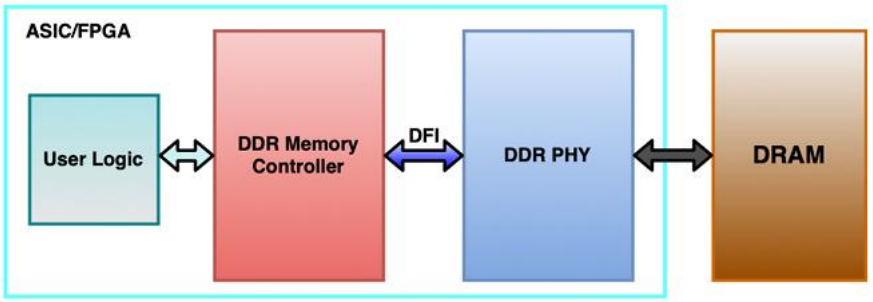


Figure 1.5: DDR Memory Sub-system

1.7 AMBA APB PROTOCOL:

Advanced Peripheral Bus (APB) is a low-power, low-bandwidth interface within the Advanced Microcontroller Bus Architecture (AMBA), developed by ARM. It is primarily designed for connecting simple peripheral devices in System-on-Chip (SoC) designs. Unlike the high-performance Advanced eXtensible Interface (AXI) and Advanced High-performance Bus (AHB), which are optimized for high-speed data transfers, APB is optimized for simplicity and power efficiency, making it ideal for peripheral devices that do not require high-speed data movement.

APB is commonly used for interfacing with peripherals such as General-Purpose Input/Output (GPIO), Universal Asynchronous Receiver/Transmitter (UART), timers, interrupt controllers, and watchdog timers. These peripherals often operate at lower data rates and do not require complex transaction mechanisms like burst transfers or pipelining.

- **APB Timing Diagrams**

The timing diagram below illustrates an APB write follow by a read.

An APB write transaction starts when the requestor drives apb_psel and apb_pwrite high. It also drives write address and write data at the same time.

One cycle later, the requestor drives the apb_penable signal high.

Sometime later, the target agent accepts the write transaction by asserting apb_pready high for one cycle.

The requestor de-asserts apb_psel and apb_penable to complete the write transaction when apb_pready is high.

An APB read transaction starts when the requestor drives apb_psel high and apb_pwrite low.It also drives read address at the same time.

One cycle later, the requestor drives the apb_penable signal high.

Sometime later, the target agent return read data by asserting apb_pready high for one cycle.

The requestor de-asserts apb_psel and apb_penable to complete the read request when apb_pready is high

CHAPTER 2: UVM FLOW

2.1 SystemVerilog for Verification

In the earlier stages of integrated circuit (IC) design, the process was relatively simple, enabling designers to perform verification using custom testbenches created solely in Verilog or in combination with C/C++. However, as IC designs have evolved, their complexity has increased significantly. Today, verification has become a major part of the design lifecycle, prompting the emergence of advanced verification tools and structured methodologies [47]. Several hardware verification languages such as Vera, Specman e, and SystemVerilog have gained popularity in this domain. Among them, SystemVerilog stands out due to its object-oriented programming (OOP) capabilities, which enhance verification efficiency. It facilitates the creation of layered testbenches using constrained random stimuli to thoroughly test design behavior [10]. A typical object-oriented testbench architecture, as depicted in Figure 2.8, includes components such as the program block, clocking block, interface, packet, driver, environment, monitor, scoreboard, and coverage. These elements are organized within a top-level module that connects to the Design Under Test (DUT) and associated interfaces. Table 2.4 provides a detailed overview of the roles and functions of each component within the OOP-based testbench structure.

Component	Description
Program Block	Aprogram block separates the testcase and testbench entity, so multiple numbers of testcases can use the same testbench. It serves as a border between the RTL code and testbench.
Clocking Block	A clocking block allows separating the functional behavior of the design from its clocking behavior. It assembles signals that are synchronous to a particular clock and makes their time explicit.
Interface	In the purest form, an interface is simply a bundle of wire. It is just like another module which can be instantiated and can also instantiate other interfaces.
Driver	The driver acts as the active element within the design's verification environment. It is responsible for implementing the logic that interacts directly with the Design Under Test (DUT). Specifically, the driver receives transaction data from the sequencer and transmits it to the DUT. This process is typically continuous, with the driver actively sending data until the sequencer halts its operation or stops generating new transactions.
Monitor	The monitor is a passive component in the verification environment that observes, captures, and collects transaction data transmitted from both the Design Under Test (DUT) and the reference model. Its primary role is to non-intrusively monitor these communications and forward the gathered transactions to the coverage checker for further analysis and validation.
Environment	The environment acts as an integrated framework that organizes and links essential verification components including the agent, monitor, scoreboard, and coverage checker. It facilitates effective interaction and coordination between these elements, enabling a structured and efficient verification process.
Scoreboard	The scoreboard is responsible for verifying the functional correctness of the design by comparing the output generated by the

	Design Under Test (DUT) against the expected output from the reference model. This comparison helps identify any mismatches or functional discrepancies in the DUT's behavior.

Table 2.1: OOP Testbench Components

2.2 UVM CLASS HIERARCHY

The Universal Verification Methodology is a framework or collection of system Verilog classes which uses OOPS concept from which fully functional testbenches are derived. This ensures that testbench is uniform across different verification teams, cross compatibility, and ability to reuse or modify one testbench for different Ips [3].

There are many advantages of using UVM methodology which of them are listed below:

Modularity and Reusability: Modular components (Driver, Sequencer, Agents, env etc.) can be used across unit level to multi-unit or chip level verification as well as across projects.

- Separating Tests from Testbenches: tests in terms of stimulus/sequencers are kept separate from the actual testbench hierarchy and hence there can be reuse of stimulus across different units or across projects.
- Simulator Independent: The base class library and the methodology is supported by all simulators and hence there is no dependence on any specific simulator.
- Stimulus Generation: Several ways in which sequences can be developed which includes randomization, layered sequences etc. which provides a good control and rich stimulus generation capability.

It mainly consists of 3 base classes from which other classes are derived.

- uvm_object : All uvm_transaction and uvm_component are derived from the uvm_object o uvm_transaction : Used in stimulus generation and analysis
- uvm_component : Components are quasi-static objects that exists throughout the simulation uvm_driver, uvm_sequncer, uvm_monitor etc. are derived from it.

2.1.1 UVM FACTORY

The main advantage of UVM is UVM Factory. The UVM Factory is the place where all classes are created, registered, and override/swapped out if required. The purpose of the UVM factory is to enable an object of one type to be substituted with an object of a derived type without changing the testbench structure or even the testbench code. The mechanism used is referred to as an override, by either instance or type [4]. This functionality is very handy for changing sequence behavior or replacing one version of a component by another. Any two components to be swapped must be polymorphically compatible.

2.1.2 UVM PHASING

To have a consistent testbench execution flow, the UVM uses phases to order the major steps that take place during simulation. The uvm_component base class contains virtual methods which are called by

each of the different phase methods, and these are populated by the testbench component creator according to which phases the component participates in. Using the defined phases allows verification components to be developed in isolation, but still be during phasing they get connected. There are three groups of phases, which are executed in the following order [5]:

- Build phases: Where the testbench is configured and constructed. All the build phase methods are functions and therefore execute in zero simulation time.
- Run-time phases: Where time is consumed in running the testcase on the testbench. The run phase is implemented as a task, and all uvm_component run_phase() tasks are executed in parallel.
- Clean up phases: Where the results of the testcase are collected and reported. Used to extract information from scoreboards and functional coverage monitors.

2.2 **UVM TESTBENCH ENVIRONMENT**

2.2.1 UVM TEST

The test is the topmost class. the test is responsible for, o

configuring the testbench.

- Initiate the testbench components construction process by building the next level down in the hierarchy ex: env.
- Initiate the stimulus by starting the sequence.

2.2.2 UVM ENVIRONMENT

Env or environment: The environment is a container component for grouping higher level components like agent's and scoreboard.

2.2.3 UVM AGENT

UVM agent groups the uvm_components specific to an interface or protocol. example: groups the components associated with BFM (Bus Functional Model).

2.2.4 UVM SEQUENCE ITEM

The sequence-item defines the pin level activity generated by agent (to drive to DUT through the driver) or the activity has to be observed by agent (Placeholder for the activity monitored by the monitor on DUT signals).

2.2.5 UVM DRIVER

Responsible for driving the packet level data inside sequence_item into pin level (to DUT).

2.2.6 UVM SEQUENCE

Defines the sequence in which the data items need to be generated and sent/received to/from the driver.

2.2.7 UVM SEQUENCER

Responsible for routing the data packet's(sequence_item) generated in sequence to the driver or vice versa.

2.2.8 UVM MONITOR

Observes pin level activity on interface signals and converts into packet level which is sent to components such as scoreboards.

2.2.9 UVM SCOREBOARD

Receives data items from monitor's and compares with expected values. Expected values can be either golden reference values or generated from the reference model.

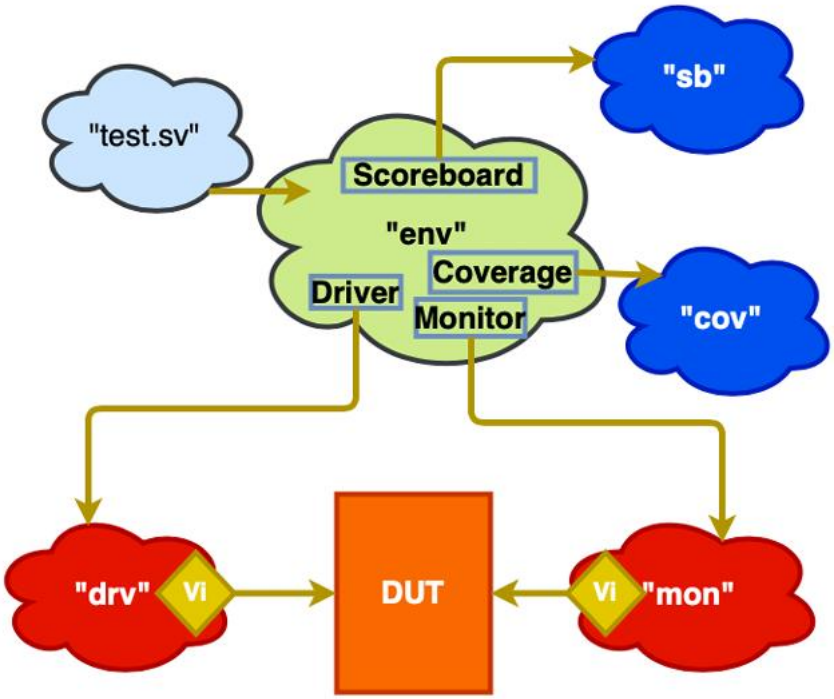


Figure 2.1: OOP Testbench Architecture

CHAPTER 3: LITERATURE REVIEW

1. M. Hassan et al. (2018)

The paper "MCXplore: Automating the Validation Process of DRAM Memory Controller Designs" by M. Hassan and H. Patel presents a systematic approach to automating the validation process of DRAM memory controller (MC) designs. This work is significant in addressing the challenges associated with ensuring correctness and efficiency in DRAM MC design by leveraging model checking and benchmark testing techniques. Below is a literature survey highlighting the key aspects of the paper, its contributions, and related works in the domain of memory controller validation.

2. HimaBindhu et al. (2018)

In their paper, "Implementation of High Speed DDR3 SDRAM Memory Controller by Using XILINX Software," HimaBindhu and colleagues present a detailed design and implementation of a DDR3 SDRAM memory controller utilizing Xilinx software tools. The authors emphasize the importance of high-speed memory access in modern applications and discuss their approach to optimizing controller performance through efficient state machine design and timing analysis. This work lays a foundation for understanding the intricacies of memory controller architecture and provides practical insights into implementation strategies using FPGA technologies.

3. Ko et al. (2021)

Ko and co-authors explore a novel controller PHY in "A controller PHY for managed DRAM solution with damping-resistor-aided pulse-based feed-forward equalizer." This study addresses challenges related to signal integrity in high-speed DRAM interfaces. The authors propose a damping-resistor-aided feed-forward equalizer to enhance data transmission reliability. Their findings demonstrate significant improvements in performance metrics, highlighting the critical interplay between physical layer design and memory controller efficiency. This work is particularly relevant for understanding the hardware-level challenges faced in high-speed memory systems.

4. Germchi (2024)

In his MS thesis, "A High Performance DDR4 Memory Controller on FPGA," Danesh Germchi presents a comprehensive study on the design and implementation of a DDR4 memory controller. The thesis details the controller architecture, focusing on achieving high throughput and low latency while ensuring power efficiency. Germchi’s work also incorporates advanced error correction techniques and emphasizes the adaptability of the design for various applications. This study is pivotal for researchers seeking to develop next-generation memory controllers that leverage FPGA technology.

5. **Lee et al. (2024)**

The paper "NPC: A Non-conflicting Processing-in-memory Controller in DDR Memory Systems" by Lee and colleagues introduces a novel processing-in-memory (PIM) controller designed to minimize data conflicts within DDR memory systems. The authors present an innovative architecture that allows for parallel data processing, significantly improving memory access efficiency. Their findings contribute to the ongoing discourse on integrating processing capabilities within memory systems, addressing the limitations of traditional architectures in handling data-intensive workloads.

6. **Rodríguez-Castellón (2024)**

In "Evolution and challenges of DDR: A policy review through the prism of Colombia's DDR experience," Rodríguez-Castellón examines the broader implications of DDR technology development in a socio-political context. While the focus is not exclusively on technical advancements, this work highlights the challenges faced in policy and implementation that can impact the deployment of advanced memory technologies. It provides a unique perspective on the intersection of technology and policy, relevant for understanding the global landscape of DDR advancements.

7. **Schmitz (2015)**

Tamara Schmitz, in her white paper "The rise of serial memory and the future of DDR," discusses the transition from parallel to serial memory architectures. This paper outlines the advantages of serial memory in terms of scalability and performance, while also addressing challenges such as increased latency and complexity in design. Schmitz's insights are crucial for researchers exploring the future trajectory of memory technologies, particularly in the context of evolving application requirements.

8. **Hajkazemi et al. (2015)**

In "Wide I/O or LPDDR? Exploration and analysis of performance, power and temperature trade-offs of emerging DRAM technologies in embedded MPSoCs," Hajkazemi and colleagues conduct a comparative analysis of Wide I/O and LPDDR technologies. Their findings illustrate the performance and power consumption implications of different memory architectures in multi-processor systems on chip (MPSoCs). This work is significant for designers considering the trade-offs in memory selection for embedded applications.

9. **Hollis et al. (2019)**

The article "Recent Evolution in the DRAM Interface: Mile-Markers Along Memory Lane" by Hollis and co-authors provides a comprehensive overview of the historical development and technical evolution of DRAM interfaces. The authors trace key milestones and innovations that have shaped the current state of memory technology, offering insights into future trends and challenges. This paper serves as a valuable resource for understanding the broader context of memory controller development

10. JEDEC website (jedec.org)

Serves as a central hub for accessing detailed specifications, white papers, and other documentation relevant to memory technologies. The website's comprehensive collection of standards documents supports the understanding of various DDR4 features and design principles, allowing industry professionals to stay updated on technological advancements.

11. EDEC's DDR4 SDRAM Standard (JESD79-4A)

Provides a formal definition of the DDR4 SDRAM architecture, including its electrical, functional, and operational characteristics. Published in 2013, this document outlines enhancements over previous DDR standards, such as improved performance, reduced power consumption, and higher data transfer rates. These improvements address the growing demand for energy efficiency and speed in modern computing applications.

12. Binkert et al. (2011)

Present gen5, a modular and highly configurable simulator for computer architecture research. Gem5 supports multiple ISAs and offers detailed modeling for CPUs, GPUs, and memory systems. It combines features from the m5 and GEMS simulators, providing researchers with a unified platform for full-system and application-level simulations. This paper emphasizes gem5's adaptability for a wide range of use cases, making it invaluable for exploring architectural innovations and evaluating performance.

13. Ecco and Ernst (2015)

Propose methods to enhance DRAM timing analysis for real-time systems, focusing on read/write bundling to reduce latency variability. Their approach refines timing bounds in real-time DRAM controllers by exploiting predictable access patterns. This work contributes to better worst-case execution time (WCET) guarantees, enabling more efficient resource utilization in systems with stringent timing constraints.

14. Hassan et al. (2015)

Develop a scheduling framework for DRAM memory in mixed-criticality multi-core systems. The framework minimizes interference between time-critical and non-critical tasks by using a predictable memory access scheduling mechanism. Their methodology improves temporal isolation, ensuring real-time guarantees for high-priority tasks while optimizing performance for lower-priority applications.

15. Hassan and Pellizzoni et al.(2018)

DRAM interference in Commercial Off-The-Shelf (COTS) heterogeneous multi-processor systems-on-chip (MPSoCs). They present techniques to analyze and bound memory interference, ensuring real-time

performance for critical tasks. This work highlights the challenges of mixed-criticality systems in shared memory environments and offers practical solutions for deploying COTS hardware in safety-critical domains.

16. **Hassan and Pellizzoni (2020)**

Extend their earlier work by providing a detailed analysis of memory contention in heterogeneous COTS MPSoCs. They propose a formal model to quantify interference and evaluate its impact on task execution. Their findings inform the design of memory access protocols that enhance predictability and support real-time operations in mixed-criticality environments.

17. **Kim et al. (2014)**

Propose techniques to bound memory interference delays in multi-core systems using COTS components. Their approach includes hardware and software mechanisms to mitigate contention, improving temporal predictability. This work is foundational for designing real-time systems that rely on shared memory, providing a framework for ensuring timing constraints in multi-core architectures.

18. **Intel Platform and Component Validation" (2015)**

Intel's white paper titled "Intel Platform and Component Validation" (2015) offers a detailed overview of the processes, methodologies, and frameworks involved in ensuring the reliability and performance of Intel's platforms and components. Validation is a critical stage in the development lifecycle, focusing on confirming that products meet design specifications and can operate effectively under expected real-world conditions.

19. **H.-M. Koo et al. (2006)**

This paper explores the use of SAT-based bounded model checking (BMC) as a method for generating test cases to validate pipelined processors. The approach leverages formal methods to verify the correctness of processor designs, addressing critical challenges such as state-space explosion and coverage limitations in traditional testing techniques. The authors propose using BMC to automatically generate inputs that exercise corner cases, ensuring robust validation of complex pipeline structures. This work is particularly relevant for improving the efficiency and reliability of design verification in modern microprocessors, where pipelining introduces intricate timing and data hazards.

20. ***R. Ausavarungnirun et al. (2012)***

This paper introduces Staged Memory Scheduling (SMS), a novel approach designed to improve performance and scalability in heterogeneous computing systems. The proposed SMS technique divides the memory scheduling process into distinct stages, each targeting specific aspects of scheduling (e.g.,

prioritization and arbitration). By decoupling these functions, SMS mitigates contention and ensures fairness across diverse workloads while maintaining high throughput. The authors demonstrate the utility of SMS in addressing the growing complexity of heterogeneous systems, where various components (e.g., CPUs, GPUs) demand optimized memory access.

CHAPTER4 : ARCHITECTURE OVERVIEW

In this chapter We are describing the design architecture of the DFI DDR4 Memory PHY Bridge

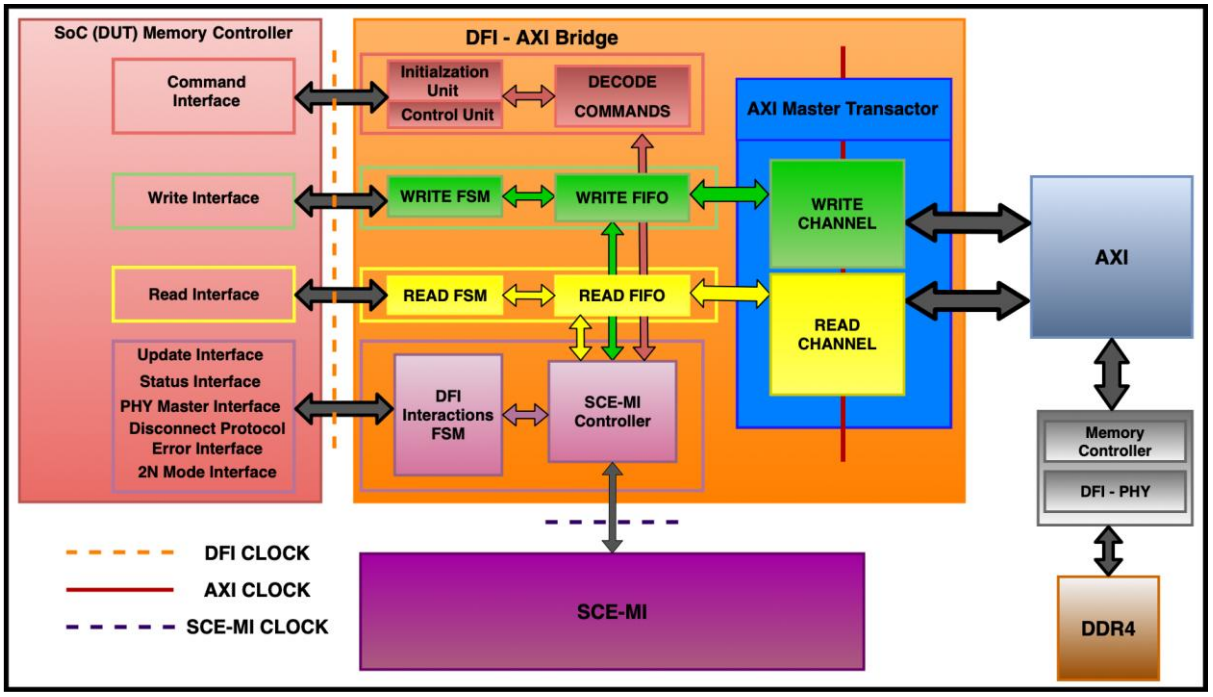


Figure 4.1: System-level Architecture of Memory Sub-system

4.1 DFI INTERFACE

The DDR PHY Interface (DFI) is a standardized protocol that establishes the communication interface between a DDR memory controller (MC) and the physical layer (PHY) for various DDR memory types, including DDR1, DDR2, DDR3, and their low-power counterparts such as LPDDR1 and LPDDR2. It outlines the required signals, their interrelationships, and the timing behavior necessary for effective transmission of both control and data signals to and from DRAM devices via the PHY.

Importantly, the DFI specification is limited to defining the interface between the controller and the PHY. It does not attempt to cover the complete internal functionalities of either component. Furthermore, it imposes no restrictions on how the MC and PHY interact with other subsystems such as Design for Test (DFT), calibration features, or additional proprietary signals that might be implemented in a specific design.

The DFI standard does not define exact timing constraints for the signals between the MC and PHY. The primary requirement is the presence of a DFI clock, with all DFI-defined signals driven by registers synchronized to the rising edge of this clock. However, there are no limitations on how these signals are received, nor is there any specification for the origin of the DFI clock itself. As a

result, the compatibility between the MC and PHY at different frequencies is implementation-specific and must be ensured by the designer based on their respective capabilities.

The DDR PHY Interface (DFI) is a standardized communication protocol that defines the interface between the DDR memory controller (MC) and the physical interface (PHY). It specifies a set of control and data signals, along with their timing requirements, that facilitate seamless data transfers to and from DRAM devices. DFI is widely adopted in electronic systems such as smartphones, computers, and networking equipment, where high-speed and low-power memory interfaces are essential.

A major advantage of DFI is its support for IP interoperability. Since the memory controller and PHY are often designed by separate vendors, DFI ensures that these independently developed components can work together reliably, promoting modularity and IP reuse in SoC designs.

- Notable Features of the DFI Protocol
  - Low Power Mode Support: The protocol includes mechanisms for energy-efficient operation. When the PHY anticipates a prolonged idle period, it can transition into a controller-initiated low-power state, helping to reduce system-wide power consumption.
  - Improved Signal Alignment: High-speed operation in DDR systems demands precise timing. DFI supports write and read training routines that enhance the precision of signal placement, thereby improving timing margins and data integrity during high-speed transfers.
  - Clock Frequency Ratio Flexibility: DFI allows dynamic changes in the operating frequency ratio between the MC and PHY without requiring a system reset. Supported frequency ratios include 1:1, 1:2, and 1:4, enabling performance scaling and system flexibility.
  - Reduced Power and Noise via Data Bus Inversion (DBI): The implementation of Data Bus Inversion helps lower switching activity on the data lines, which in turn minimizes power consumption and reduces signal noise, improving overall signal quality and power efficiency.

- Interface Group Architecture

  The DFI protocol is structured into several interface groups, each comprising a specific set of signals and parameters responsible for distinct functions in the MC-PHY communication path. These groups streamline the design and implementation process by clearly defining responsibilities and timing constraints for each functional block. A visual representation of these groups and their interconnections is shown in Figure 4.2.

- Command Interface

The Command Interface is responsible for conveying address and command information from the memory controller to the DRAM via the PHY. It ensures that these signals maintain accurate timing alignment to meet DRAM protocol requirements.

This interface is primarily composed of the Command Address (CA) bus, which includes:

- Command and control signals such as dfi_act_n, dfi_ras_n, dfi_cas_n, and dfi_we_n
- Address-related signals like dfi_address, dfi_bank, and dfi_bg

These signals encode read and write transactions, along with row, bank, and chip-select information required by DRAM devices. The definitions of these signals are provided in Table 3.1, while their corresponding timing constraints are outlined in Table 3.2. The CA bus serves as the key pathway through which the memory controller issues access commands to the DRAM.

- The DFI is subdivided into the following interface groups:
  - Control Interface
  - Write Data Interface
  - Read Data Interface
  - Update Interface
  - Status Interface

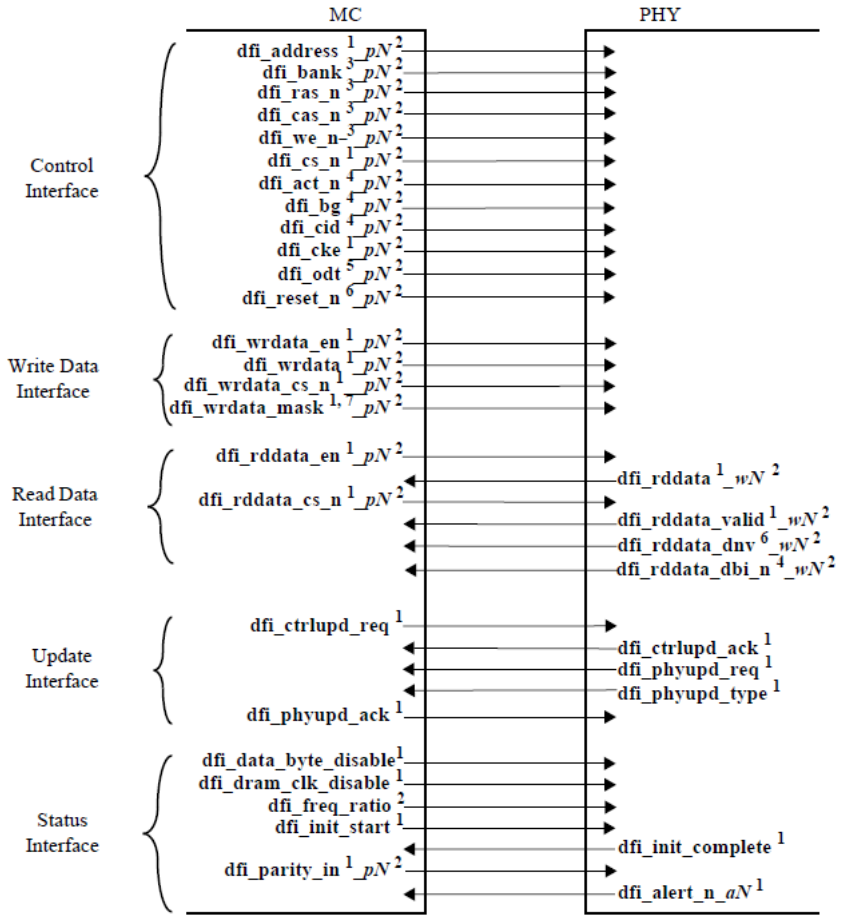


Figure 4.2: Block diagram of DDR PHY Interface signals (Source: [14], p. 17)

4.2 INTERFACE SIGNAL GROUP

The control interface within the DFI specification serves as a direct representation of the DRAM control mechanism. It encompasses signals related to the memory address, bank selection, chip select, row strobe (RAS), column strobe (CAS), write enable (WE), clock enable (CKE), and On-Die Termination (ODT), depending on the specific memory technology in use. In conjunction with this, the write data and read data interfaces are responsible for transmitting valid write data to the DRAM and receiving valid read data from it, ensuring proper data flow across the DFI. The update interface plays a crucial role in allowing either the PHY or the memory controller to interrupt or temporarily halt activity on the DFI, providing flexibility for calibration or reconfiguration tasks. The status interface supports system initialization processes and enables or disables valid clocks to the DRAM, aiding in the management of clock control and feature monitoring. Additionally, the training interface facilitates timing calibration procedures such as data eye training, gate training, and write leveling, which are essential for maintaining signal integrity at high speeds. Lastly, the low power control interface is designed to manage and transition the PHY into various low power states, thereby supporting energy-efficient operation of the memory subsystem.

- **Write Data Interface**

The Write Data Interface is responsible for transferring valid write data from the memory controller to the PHY and subsequently to the DRAM. It includes a defined set of signals and enforces specific timing relationships to ensure data integrity. This interface coordinates closely with the command signals to align the data beats with the correct write transactions.

- **Read Data Interface**

The Read Data Interface handles the return path for read data from the DRAM through the PHY back to the memory controller. Like the write interface, it specifies signal definitions and timing constraints that guarantee data stability and synchronization with the read command. This interface plays a crucial role in maintaining data accuracy during high-speed memory operations.

- **Update Interface**

The Update Interface enables either the PHY or the memory controller to temporarily interrupt or stall operations on the DFI. This feature is useful during calibration, training, or dynamic configuration changes, allowing for safe and controlled transitions without data corruption.

- **Status Interface**

The Status Interface supports system initialization and feature monitoring, and it also helps control the validity of clocks supplied to the DRAM. Through this interface, the system can monitor readiness, detect faults, and manage configuration-dependent behaviors during startup or mode transitions.

- **Training Interface**

  The Training Interface is designed for executing critical timing calibration procedures such as Data eye training, Gate training, Write leveling. These operations are essential for aligning data strobe and data signals accurately, particularly in high-speed DDR environments where timing margins are minimal.

- **Low Power Control Interface**

  The Low Power Control Interface manages the transitions into and out of various low-power states supported by the PHY. This interface allows the system to reduce power consumption during idle periods by coordinating with the PHY to enter power-efficient modes without compromising data retention or system stability

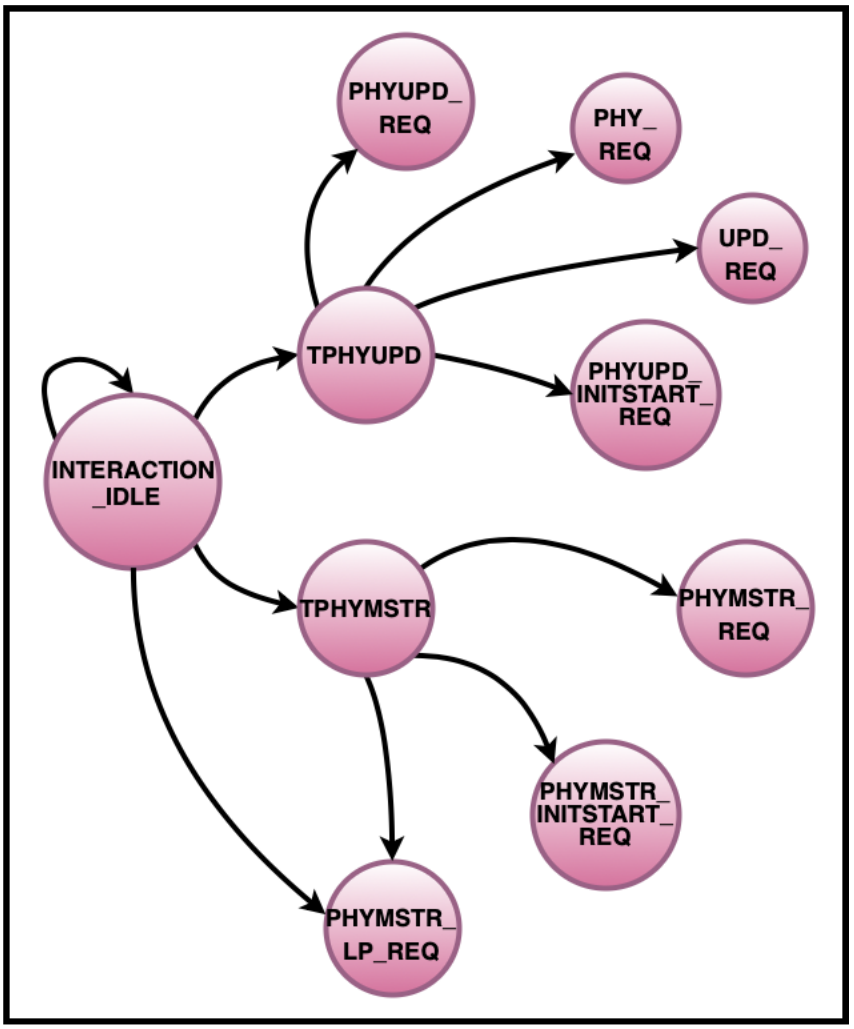


Figure 4.3: DFI Interactions State Machine

dfi_phyupd_req_r	dfi_phymstr_req_r	dfi_ctrlupd_req_r	dfi_init_start_r	dfi_lp_ctrl_req_r	dfi_lp_data_req_r	INTERACTIO	INTERACTION NEXT STATE

						N STATE	
1	0	0	0	x	x	TPHYUPD	PHYUPD_REQ
1	1	0	0	x	x	TPHYUPD	PHY_REQ
1	0	1	0	x	x	TPHYUPD	UPD_REQ
1	0	0	1	x	x	TPHYUPD	PHYUPD_INITSTART_REQ
0	1	0	0	0	0	TPHYMSTR	PHYMSTR_REQ
0	1	0	1	0	0	TPHYMSTR	PHYMSTR_INITSTART_REQ
0	1	0	0	1	1	TPHYMSTR	PHYMSTR_LP_REQ

Table 4.1: Truth table of DFI Interaction Signals

4.3 SCE-MI

The Standard Co-Emulation Modeling Interface (SCE-MI) is a widely accepted protocol that establishes a multi-channel communication link between a hardware design under emulation and software-based bus functional models (BFMs) or testbenches. This interface is particularly beneficial in bridging the gap between high-speed emulation and traditional simulation environments. It effectively replicates the behavior of original simulation interfaces to enable seamless interaction with the Design Under Test (DUT) during emulation.

In the context of the DFI DDR4 Memory PHY Bridge verification environment, the SCE-MI block performs several critical functions that support the accurate configuration and execution of memory transactions. It acts as a communication mediator that delivers protocol-specific transaction data from the software domain to the emulated DUT.

The primary responsibilities of the SCE-MI block include:

First, it sends all required timing and transaction parameters to both the memory controller and the DFI DDR4 Memory PHY Bridge module. For write operations, it transfers address channel data to the write FIFO using the MessageIn port. The format of the transmitted data includes the following fields:

{AWDUMMY3, AWUSER, AWREGION, AWQOS, AWLEN, AWDUMMY1, AWSIZE, AWBURST, AWLOCK, AWCACHE, AWDUMMY2, AWPROT, AWID}.

Additionally, for write data transactions, a similar structure is used to send data-related channel parameters to the write FIFO, again via MessageIn.

In read transactions, the SCE-MI block transmits the read address channel parameters to the read FIFO, usinga structured format that includes:{ARDUMMY3, ARUSER, ARREGION, ARQOS, ARLEN, ARDUMMY1, ARSIZE, ARBURST, ARLOCK, ARCACHE, ARDUMMY2, ARPROT, ARID}.

Beyond just transferring data, the SCE-MI block also maintains interaction with the SCE-MI controller and interfaces with command decoding logic. This interaction helps interpret incoming message structures and ensures the correct execution of emulation tasks in line with the expected DDR transaction protocol.

CHAPTER 5: STATEMENT OF PROBLEM

5.1 PROBLEM STATEMENT:

IP Verification is done post RTL for an IP is designed so that when IP is ready for release to soc team. Generally, many IPs are there on SOC therefore it is important to verify an individual IP so that if any fault is found, then it is debugged and fixed at an early stage.

Create one single script to make the verification framework automated to verify the IP here regression or single test both can be launched as needed so that if any changes in attributes are encountered as per the customer needs, then it can be done at one place instead of going through so many compilation and simulation scripts. If any test fails then check the failure if it is at testbench level fix it, or if it is at RTL Level then inform the concerned team and then again verify it.

5.2 MOTIVATION:

Current DDR PHY Training Unit lack modularity for easy extension and adaptation to future DDR memory standards, limiting their applicability in evolving memory systems is necessary as creating new DDR PHY Training Unit for each use case of memory can be costly.

With the Increase in complexity of IP, Verification plays an important role. To Verify the IP understanding of UVM is needed. Hence it is always motivating to learn a new methodology and to have hands on it by creating any sequential or non-sequential design. Learning new simulator such as Synopsys Verdi waveform viewer and learning to debug the failure using the logs.

Perl is a scripting language used for Automation of Verification flow. It was quite motivating to learn it for the IP

CHAPTER 6: RESULT

6.1 Weekly Triaging and Report Generation:

A rigorous Weekly triaging process was conducted to identify and prioritize issues found during regression testing. Alongside, automated reports were generated for both regression results and coverage analysis, ensuring transparency and effective communication with the team. These reports provided critical insights into the test results and design behaviour, allowing the team to act promptly on any identified issues.

Generated an easy-to-read report summarizing results, highlighting priority issues, and providing actionable insights. Delivered reports in formats like Excel.

6.2 Automation for report generation and running regression:

Objective: Streamline the process of analyzing and reporting regression results.

Developed a script to parse regression logs and extract critical information (pass/fail status, test failures, coverage gaps, etc.). Automated root-cause analysis by correlating failure points with design features or recent changes. Integrated features to categorize failures (e.

g., setup issues, design bugs, testbench issues).

Designed a script to schedule and execute tests across various configurations automatically. Included features to monitor regression progress in real-time and handle resource allocation dynamically. Added retry mechanisms (recall) for tests that fail due to transient issues or infrastructure limitations.

Automated the collection and aggregation of test results from different simulation environments.

6.3 Re-running the Failed Test Cases with Fixes and Updating the Mainline:

After identifying bugs or failures, fixes were applied, and the affected tests were re-executed. The re-run tests confirmed the resolution of the issues, and the mainline was updated accordingly, ensuring that the latest changes were integrated without introducing regressions. This iterative process helped maintain the stability of the verification flow and aligned the design with expected functionality.

6.4 Regression Improvement:

Regression pass status also saw a remarkable improvement, with the pass rate reaching to 100% for DDR5 and about 98.5 % for LPDDR4, demonstrating the stability of the designand the effectiveness of the fixes applied

Testlist	Total tests	Passing	Failing	No status	Pass percentage
DDR4_LEVEL1	683	683	0	0	100.0%
DDR4_LEVEL2	1466	1466	0	0	100%
DDR4_LEVEL3	90	90	0	0	100%
LPDDR4_LEVEL1	767	760	7	0	99.08%
LPDDR4_LEVEL2	1000	992	8	0	99.20%
LPDDR4_LEVEL3	84	84	0	0	100.00%

Table 6.1 Regression Result

6.5 Coverage Analysis

Coverage serves as a key metric in evaluating how thoroughly the stimulus exercises the DFI-DDR4 Memory PHY Bridge module. To generate and analyze this coverage data, the Synopsys Verdi is utilized as the primary functional verification tool. The code coverage of the i-dfi_bridge module offers valuable insight into the accuracy and completeness of the RTL logic implementation. To ensure functional correctness, the testbench incorporates a series of assertions that validate the expected behavior of the design.

An overview of both the anticipated and actual coverage results obtained during simulation is presented in Table 6.2. Additionally, Figure 6.1 provides a visual representation of the IMC environment's coverage report, illustrating the extent to which various parts of the design were exercised.

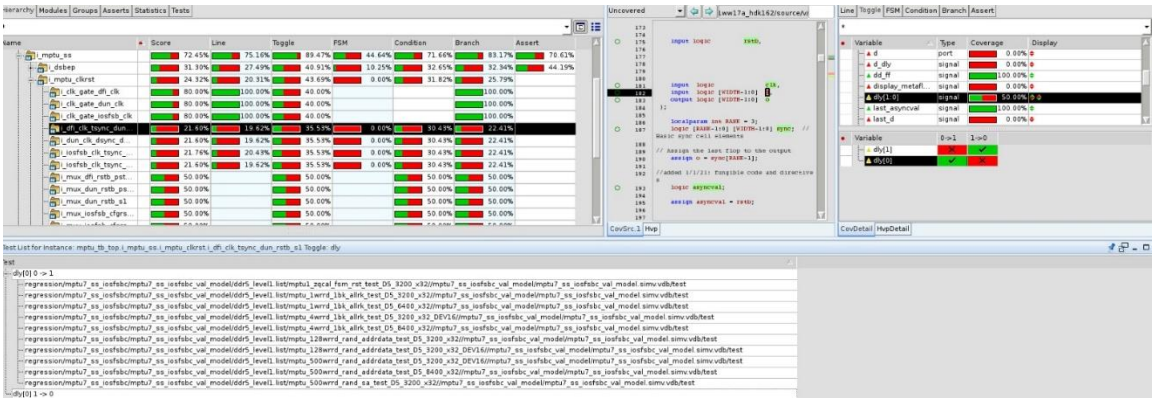


Fig 6.1 Coverage Report

	TECH	Type	WW11 without exclusions	WW11 with exclusions
Functional Coverage : Commands & features(BL, Datawidth, Device density, Frequency, Bank org, Ranks, Timing sets) Coverage	LPDDR4	LPDDR4 Commands	100	100
		LPDDR4 Features	90.54	90.43
		LPDDR4 Features Cross coverage	23.31	24.51
	DDR4	DDR4 Commands	100	87.50
		DDR4 Features	88.73	89.71
		DDR4 Features Cross coverage	16.53	17.71
Code Coverage: At i_mptu_ss hierarchy	LPDDR4+DDR4 (BOTH)	LINE	80.07	86.17
		TOGGLE	80.75	82.91
		FSM	48.91	67.00
		CONDITION	76.65	78.51
		BRANCH	75.60	80.80
		ASSERT	97.71	97.71
		SCORE	76.61	82.18

Table 6.2 Coverage Result

6.6 Testcase and Simulation Result

The test environment supports three distinct frequency ratio configurations, as outlined in Table 5.1. Additionally details the different types of write requests that the bridge module can process, along with the corresponding number of data beats expected for each type.

The bridge is designed to handle three categories of write transactions: single writes, bursts of 4 (BC4), and bursts of 8 (BL8). In this particular test scenario, the configuration uses frequency mode 0 and initiates a burst write of length 8 (BL8).

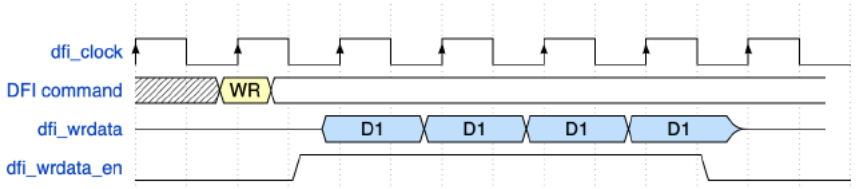


Figure 6.2: Write Interface Signal Transaction with Burst of 8 (Source: [15], p. 6)

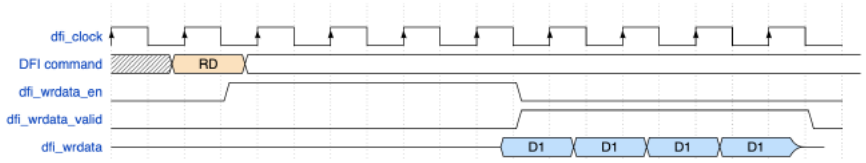


Figure 6.3: Read Interface Signal Transaction with Burst of 8(Source: [15], p. 7)

The expected outcomes from the simulation based on this configuration are illustrated in Figures 6.2, 6.3, 6.4, and 6.5, which provide visual confirmation of the correct functionality and data handling behavior of the design under test.

A write operation—either a single write or a burst write—is initiated when specific control signals are asserted. A valid write command is identified when dfi_cs_n_p0_r, dfi_cas_n_p0_r, and dfi_we_n_p0_r are all driven low, while dfi_act_n_p0_r and dfi_ras_n_p0_r remain high. The value of the dfi_we_n_p0_r signal is also used to determine the size of the burst issued by the memory controller (MC).

Depending on the burst type, the bridge module receives a corresponding number of data beats:

- For a **burst of 1**, one data beat is transferred.
- For a **burst of 4**, two data beats are received.
- For a **burst of 8**, four data beats are expected.

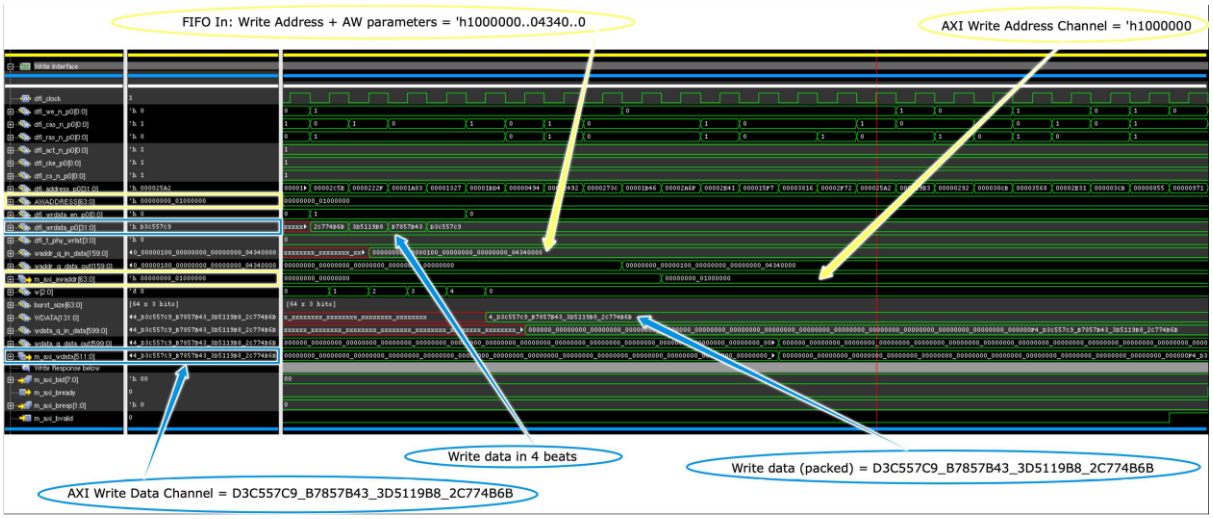


Figure 6.4: Write Data Interface Simulation Waveforms

A read command initiates a data transfer from the memory controller (MC) to the AXI interface. This command is triggered when the signals dfi_cs_n_p0_r and dfi_cas_n_p0_r are asserted low, while dfi_act_n_p0_r, dfi_ras_n_p0_r, and dfi_we_n_p0_r remain high, indicating a valid read operation.

The read address and associated AXI read address channel information are first written to the raddr_q_data_in register under the DFI clock domain. These values are then read from raddr_q_data_out in the AXI clock domain and forwarded to the AXI signal m_axi_araddr.

Once the AXI read transaction is completed, the read data is returned via the m_axi_ardata signal. This data is written to all_r_data_q_data_in on the AXI clock and subsequently read from all_r_data_q_data_out on the DFI clock. Finally, the data is passed to the dfi_rddata signal, completing the read path.

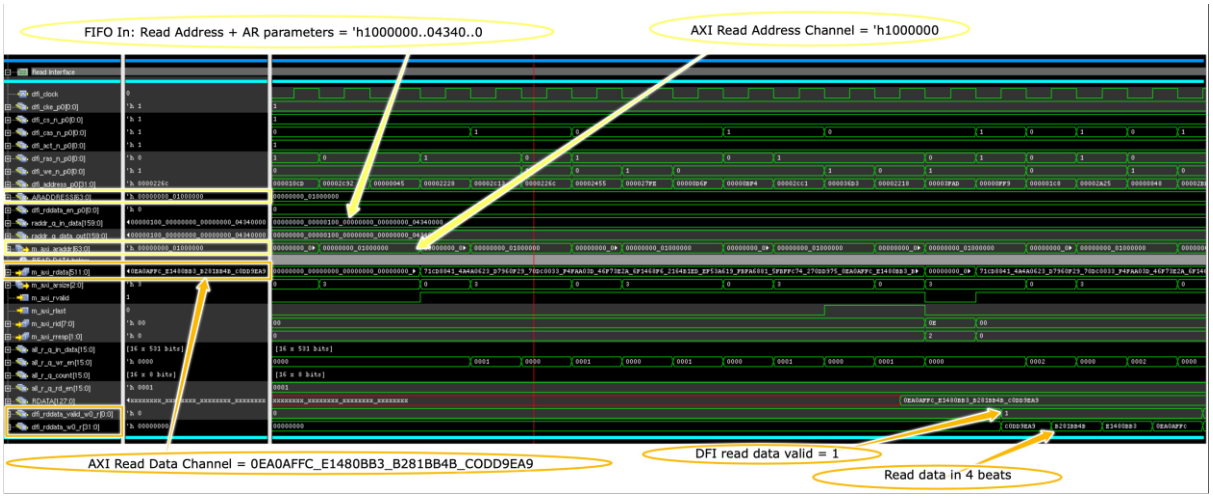


Figure 6.5: Read Data Interface Simulation Waveforms

CHAPTER 7: FUTURE SCOPE

This section outlines potential avenues for extending the current thesis work, with the aim of improving the verification framework of the DFI DDR4 Memory PHY Bridge. These suggestions can enhance overall functionality, increase test coverage, and support further scalability of the validation environment.

- Incorporating an active SCE-MI (standard co-emulation modeling interface) can facilitate bi-directional communication between the DUT and host software, allowing for dynamic transaction-level emulation and better debugging capabilities.
- The current verification environment can be significantly improved by developing additional constrained random testcases. This help uncover corner cases and improve confidence in the robustness of the design.
- As Enhancement in architecture of DDR PHY will be there simultaneously enhancement is scripts will also be needed.
- DFI interfacing protocol keeps on changing along with changes DDR hence scripts for DDR and DFI need to be created.
- To make the framework compatible with regression runs that include both Hip and Sip.

CHAPTER 8: CONCLUSION

This thesis presents a comprehensive methodology for the pre-silicon validation of DDR PHY training sequences using a DFI-PHY bridge. By combining RTL and SystemC models, the proposed framework enables early-stage, high-fidelity validation of memory interface behavior. This integrated approach improves the accuracy and efficiency of verifying complex DDR initialization and training processes, ultimately contributing to enhanced system reliability and performance. Beyond addressing existing challenges in validation workflows, this methodology also lays the groundwork for future advancements in memory subsystem verification and SoC-level design validation.

The Report also presents the Implementation of an automated framework for regression testing. The Framework aims to streamline the regression testing process for the DDR PHY by automating the flow involve in the verification of the PHY IP.

REFERENCES:

[1] M. Hassan and H. Patel, "MCXplore: Automating the Validation Process of DRAM Memory Controller Designs," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 5, pp. 1050-1063, May 2018.

[2] HimaBindhu, J., et al. "Implementation of High Speed DDR3 SDRAM Memory Controller by Using XILINX Software." International Conference on Communications and Cyber Physical Engineering 2018. Singapore: Springer Nature Singapore, 2024.

[3] Ko, Hyeongjun, et al. "A controller PHY for managed DRAM solution with damping-resistor-aided pulse-based feed-forward equalizer." IEEE Journal of Solid-State Circuits 56.8 (2021): 2563-2573

[4] Germchi, Danesh. A High Performance DDR4 Memory Controller on FPGA. MS thesis. University of Waterloo, 2024

[5] Lee, Seungyong, et al. "NPC: A Non-conflicting Processing-in-memory Controller in DDR Memory Systems." IEEE Transactions on Computers (2024).

[6] Rodríguez-Castellón, José Miguel. "Evolution and challenges of DDR: A policy review through the prism of Colombia's DDR experience." Heliyon 10.13 (2024).

[7] Schmitz, Tamara. "The rise of serial memory and the future of DDR." White paper: UltraScale Devices (2015): 1-9.

[8] M. H. Hajkazemi, M. K. Tavana and H. Homayoun, "Wide I/O or LPDDR? Exploration and analysis of performance, power and temperature trade-offs of emerging DRAM technologies in embedded MPSoCs," 2015 33rd IEEE International Conference on Computer Design (ICCD), New York, NY, USA, 2015, pp. 62-69

[9] T. M. Hollis et al., "Recent Evolution in the DRAM Interface: Mile-Markers Along Memory Lane," in IEEE Solid-State Circuits Magazine, vol. 11, no. 2, pp. 14-30, Spring 2019

[10] JEDEC Solid State Technology Association. [Online]. Available: https://www.jedec.org

[11] JEDEC Solid State Technology Association. (2013) DDR4 SDRAM Standard. [Online]. Available: https://www.jedec.org/standards-documents/docs/jesd79-4a

[12] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. SIGARCH Comput. Archit. News, 39(2):1–7, aug 2011.

[13] Leonardo Ecco and Rolf Ernst. Improved dram timing bounds for real-time dram controllers with read/write bundling. In 2015 IEEE Real-Time Systems Symposium, pages 53–64, 2015.

[14] Mohamed Hassan, Hiren Patel, and Rodolfo Pellizzoni. A framework for scheduling dram memory accesses for multi-core mixed-time critical systems. In 21st IEEE RealTime and Embedded Technology and Applications Symposium, pages 307–316, 2015.

[15] Mohamed Hassan and Rodolfo Pellizzoni. Bounding dram interference in cots heterogeneous mpsocs for mixed criticality systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 37(11):2323–2336, 2018.

[16] Mohamed Hassan and Rodolfo Pellizzoni. Analysis of Memory-Contention in Heterogeneous COTS MPSoCs. In Marcus V¨olp, editor, 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020), volume 165 of Leibniz International Proceedings in Informatics (LIPIcs), pages 23:1–23:24, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum f¨ur Informatik.

[17] Hyoseung Kim, Dionisio de Niz, Bj¨orn Andersson, Mark Klein, Onur Mutlu, and Ragunathan Rajkumar. Bounding memory interference delay in cots-based multi-core systems. In 2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS), pages 145–154, 2014.

[18] "Intel platform and component validation , a white paper," http://download.intel.com/design/chipsets/labtour/PVPT WhitePaper.pdf, Intel, 2015-08-31.

[19] H.-M. Koo et al., "Test generation using sat-based bounded model checking for validation of pipelined processors," in ACM Great Lakes symposium on VLSI, 2006.

[20] R. Ausavarungnirun et al., "Staged memory scheduling: achieving high performance and scalability in heterogeneous systems," ACM SIGARCH Computer Architecture News, 2012.