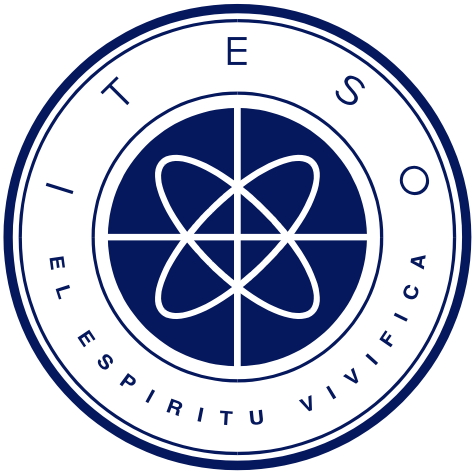
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Especialidad en Diseño de Circuitos Integrados

Reconocimiento de Validez Oficial de Estudios de nivel superior según Acuerdo Secretarial 15018, publicado en el *Diario Oficial de la Federación* el 29 de noviembre de 1976

DEPARTAMENTO DE ELECTRÓNICA, SISTEMAS E INFORMÁTICA



REFERENCE MODEL AND VERIFICATION ENVIRONMENT DESIGN FOR DDR SDRAM MEMORIES

Final Project Report
TO OBTAIN THE DIPLOMA OF

INTEGRATED CIRCUIT DESIGN SPECIALTY

PRESENTS:

Alejandro Güereña Morán

Final Project Director: Dr. Mariano Aguirre Hernández
Tlaquepaque, Jalisco. August 2011

Engineering Specialty (2011) ITESO

Integrated Circuit Design Specialty Tlaquepaque, Jal., México

TITLE: **Reference Model and Verification Environment Design for DDR SDRAM Memories.**

AUTHOR: Alejandro Güereña Morám

Bachelor of Electronics Technologies Engineering (ITESM, Guadalajara, Mexico)

FINAL PROJECT DIRECTOR: Dr. Mariano Aguirre Hernández

Bachelor of Communications and Electronic Engineering (IPN, ESIME, Mexico)

Master in Science, Electronic Specialty (INAOE, Mexico)

Doctorate in Science, Electronic Specialty (INAOE, Mexico)

NUMBER OF PAGES: vii, 54

First, I would like to acknowledge
the patience and support
of my entire family.

Second, I appreciate the
support and assistance
of my project director.

I would also like to thank
ITESO, IJJ and INTEL
for providing me the resources
for my technical preparation on the
Integrated Circuit Design Specialty.

Last, I appreciate the valuable lessons
taught by all my teachers and classmates
during my preparation on the Specialty.

Abstract

Reference models (RMs) and verification environments are used along the design flow of a digital system, in order to validate the correct and complete behavior of the system model.

The analysis and improvement of a reference model (RM), and the implementation of a verification environment for Double Data Rate Synchronous Dynamic Random Access Memories were carried out in this work. The development of these elements is intended to reduce the effort of validate every DDR SDRAM that is used on any application, because the same reference model and environment can be used, just changing some configuration parameters.

For this project, it is essential the understanding of the communication protocol for DDR SDRAM memories, which are recent, with a constant evolution. A RM analysis and improvement was done in order to implement the correct functionality of the current protocol.

Also, learning the methodologies, tools, techniques and languages for the verification of digital systems were important for the development of a reference model and a verification environment.

The design and implementation of a complete verification environment for DDR SDRAM are presented as a result of this work, using System Verilog as verification language and taking advantage of object oriented programming methodology. Mentor Graphics Questa Advanced Simulator was used as the main tool for this project.

Finally, simulations were executed for the RM in order to confirm the expected results, where the verification environment automatically generates the entire set of stimulus and analyzes all the results required for a correct and complete verification.

Content

Introduction

This section provides a description of the project, its justification and the work methodology applied during its realization.

Project Description

The final project for the Integrated Circuit Design Specialty consists on analyzing and improving a DDR SDRAM reference model, which is provided by some memories manufacturers, like Samsung Electronics, Micron Technology and Hynx.

Furthermore, a verification environment is implemented using System Verilog as a verification language and an object oriented programming methodology, in order to verify all the requirements established by the DDR SDRAM interface standard published by Joint Electron Devices Engineering Council (JEDEC). This verification environment is built up by drivers, monitors, checkers and coverpoints that validate the RM.

Project Justification

Pre-silicon verification process proves that an implementation is a correct functional representation of its specification. This increases the quality of an electronic product ensuring that it behaves according to the specifications in all scenarios.

Finding an error in pre-silicon verification stage, has a very low cost compared to errors found in later testing stages, like post-silicon, pre-product or post product stages.

Verification consumes 70% of the developing effort for a product, since it involves various stages like planning, development environments, simulation, debugging... [Janick-Bergeron-01]

By implementing a RM and a general verification environment, that can be used to validate any Register Transfer Level (RTL) model which implements the protocol for DDR

SDRAM interface, the effort and time spent as a result of the development of a RM and environments, are greatly reduced.

The development of this project gives the student an experience in developing projects that take place in the industry. Also, the student learns about a memory communication protocol which is recent and very used, and learns methodologies, tools, techniques and languages used for digital systems verification.

Work Methodology

For this project, the work methodology consisted of the steps to realize the project.

These steps are:

- Investigate information about DDR SDRAM and pre-silicon verification.
- Understand the DDR SDRAM protocol and its functionality.
- Review concepts of pre-silicon verification.
- Research on the implementation of RMs and verification environments using Verilog, System Verilog and object oriented programming.
- Analyze the RM for DDR SDRAM.
- Improve the RM for DDR SDRAM.
- Implement the verification environment for DDR SDRAM reference model.
- Implement test scenarios for the RM and verification environment.
- Perform testing and debugging of the RM.
- Document the work carried out.

1. Theoretical Framework

This section provides an introduction to the theory of operation of SDRAM and DDR SDRAM memories and the main features of the memory used in this project.

1.1. Synchronous Dynamic Random Access Memories (SDRAM)

Dynamic Random Access Memories (DRAM), is a type of RAM that stores each bit in a capacitor within an integrated circuit. Because capacitors have a leakage current, the stored information eventually fades unless the capacitor charge is refreshed several times each second.

The main advantage of this kind of memories is that only one transistor and a storage capacitor is required per bit as shown in Figure 1, allowing integrated circuits with high-DRAM density, compared to SRAM memories, which requires 6 transistors to store a bit.

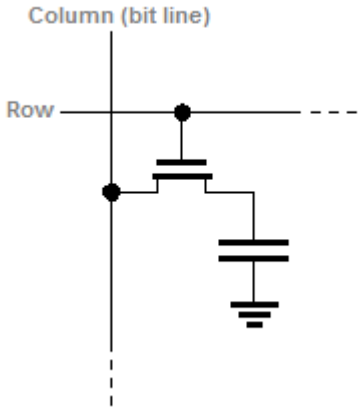


Figure 1. DRAM Memory basic array. [Floyd-01]

DRAM memories contain an asynchronous interface, which responds to changes in inputs or control signals. In order to synchronize the memory with the system clock in a computer, synchronous memories are used. SDRAM memories have their read and write cycles synchronized with the system clock. This allows having complex control logic to execute

memory instructions and to drive a state machine that pipelines incoming instructions, enabling higher speeds than DRAM memories.

A pipelined structure can receive a new instruction before completing the processing of the previous instruction. When a write command is issued to the memory, another instruction can be sent to the memory without having to wait for the data to be written to the memory. When a read command is issued to the memory, the data you want to read appears in the output port after a fixed number of clock pulses after the issued read instruction, allowing to send additional instructions to the memory during a fixed number of clock cycles (latency).

1.2. Double Data Rate Synchronous Dynamic Random Access Memories (DDR SDRAM)

Double data rate memories provide the same characteristics of SDRAM, with some improvements. DDR provides source synchronous data capture at a rate of twice the clock frequency. “This is accomplished by utilizing a 2n-prefetch architecture where the internal data bus is twice the width of the external data bus and data capture occurs twice per clock cycle, at the rise and fall edges of the clock signal. To obtain high speed signal integrity, DDR SDRAM utilizes a bidirectional data strobe signal (DQS) with a differential pair for the system clock and therefore will have both a true clock (CK) and complementary clock (CK#) signal, not present on SDRAM”. [Micron-01]

SDRAM or Single Data Rate SDRAM and DDR SDRAM, both have an identical addressing and command control interface; both have a four bank memory array; and both incorporate the same refresh requirements. The fundamental differences are found in the data interface.

“The SDR memory data interface is a fully synchronous design where the data is only captured on the positive clock edge. The internal bus is the same width as the external data bus and data latches into the internal memory array sequentially as it passes through the I/O buffers. SDR memory also supports a Data Mask (DM) signal that acts as a data mask during a write operation or an output enable for a read

The DDR memory data is a source synchronous design, where the data is captured twice per clock cycle with a bidirectional data strobe. Commands are entered on the positive edges of clock, and data occurs on both positive and negative edges of the clock. DDR architecture employs a 2n-prefetch architecture, where the internal data bus is twice the width of the external bus. This allows the internal memory cell to pass data to the I/O buffers in pairs.

With DDR, there is no output enable for read operations, but DDR does support a burst terminate command to quickly end a read instruction in process. During a write operation, the DM signal is available to allow the masking of invalid write data. The DDR command bus consists of a clock enable, chip select, row and column addresses, bank address, and a write enable as shown in Table 1". [Micron-01].

In DDR, positive edges of the clock for write instructions are different than those for read instructions. This is because of differences in latencies and because the array access occurs at the beginning of a read operation but at the end of a write operation. This represents a main feature in DDR, where a DQS signal is added to the interface.

In a purely synchronous system, data output and capture are referenced to a common, free running system clock. However, the maximum data rate for such a system is reached when the sum of output access time and flight time approaches the bit time (the reciprocal of the data rate). Although generating delayed clocks for early data launch and/or late data capture will allow for increased data rate, these techniques do not account for the fact that the data valid window (or data eye) moves relative to any fixed clock signal, due to changes in temperature, voltage, or loading. So, to allow for even higher data rates, data strobe signals were added to DDR devices. [Micron-01]

Symbol	Signal Name	Description
DQ	Data	Data Bus
DM	Data Mask	DM is an input mask signal for write data. Input data is masked when DM is sampled high along with that input data during a write access. DM is sampled on both edges of DQS.
DQS	Data Strobe	Output with read data, input with write data. Edge-aligned with read data, centered in write data. Used to capture write data.
CK, CK#	System Clock	CK and CK# are differential clock inputs. All address and control input signals are sampled on the positive edge of CK and negative edge of CK.
CKE	Clock Enable	CKE high activates, and CKE low deactivates internal clock signals, and device input buffers and output drivers. CKE is synchronous for all functions except for disabling outputs, which is achieved asynchronously.
CS#	Chip Select	CS# enables (registered low) and disables (registered high) the command decoder. CS# is considered part of the command code.
RAS#	Row Address Strobe	RAS# defines the command being entered.
CAS#	Column Address Strobe	CAS# defines the command being entered.
WE#	Write Enable	WE# defines the command being entered.
BA0-BA1	Bank Address	BA0 and BA1 define to which bank ACTIVE, READ, WRITE or PRECHARGE command is being applied.
A0-Ai	Address	Provide the row address for ACTIVE commands, the column address and AUTO PRECHARGE bit for READ/WRITE commands, to select one location out of the memory array in the respective bank. A10 is sampled during a PRECHARGE command to determine whether the PRECHARGE applies to one bank (A10 low) or all banks (A10 high). If only one bank is to be precharged, the bank is selected by BA0, BA1. The address inputs also provide the op-code during a MODE REGISTER SET command. BA0 and BA1 define which mode register is loaded during the MODE REGISTER SET command (MRS or EMRS).

TABLE 1. DDR SDRAM Inputs/Outputs. [Samsung-01]

The data strobes are non free running signals driven by the device, which is driving the data signals (the controller for write instructions, the DRAMs for read instructions). At the DRAM device level, for read instructions, the data strobe (DQS) signals are effectively

additional data outputs (DQ) with a predetermined pattern; for write instructions, the strobe signals are used as clocks to capture the corresponding input data.

For read instructions, the data strobe signals are edge-aligned with the data signals, meaning that all data and data strobes are clocked out of the device by the same internal clock signal, and all will transition at the outputs at nominally the same time as shown in Figure 2. The controller will internally delay the received strobe to the center of the received data eye.

For write instructions, the controller must provide the data strobes center-aligned relative to data. That is, strobe transitions occur nominally 90 degrees (relative to the clock frequency) out of phase with data transitions as shown in Figure 3. The DRAM device uses internally matched routing for the strobes and data such that the strobes can be used directly to capture input data. Note that the reason read and write instructions use a different alignment scheme is so that the delay circuitry can be centralized in one place (the controller) and does not have to be replicated in every DRAM device in the system. [Micron-01]

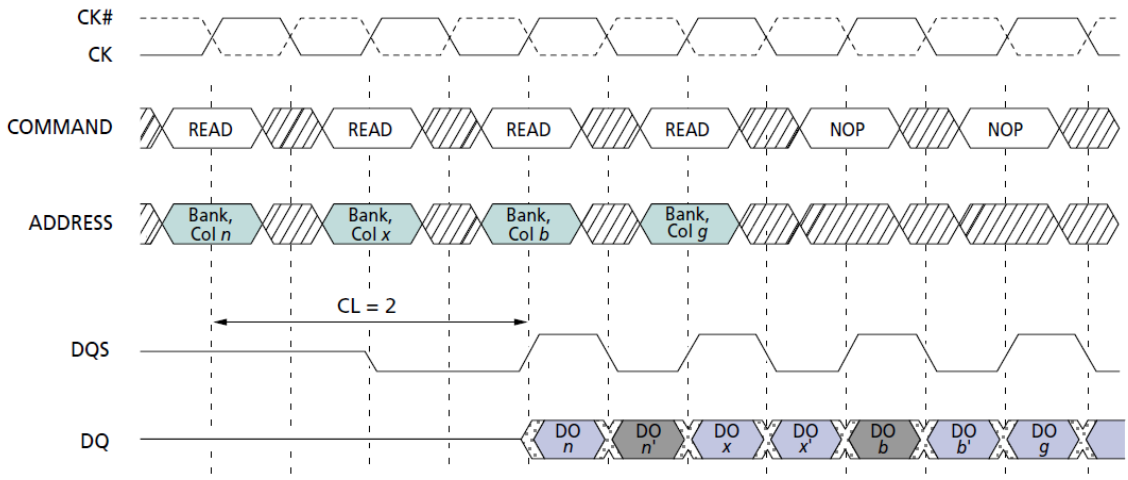


Figure 2. Minimum Data Time Slot for 2n-Prefetch Read. [Micron-01]

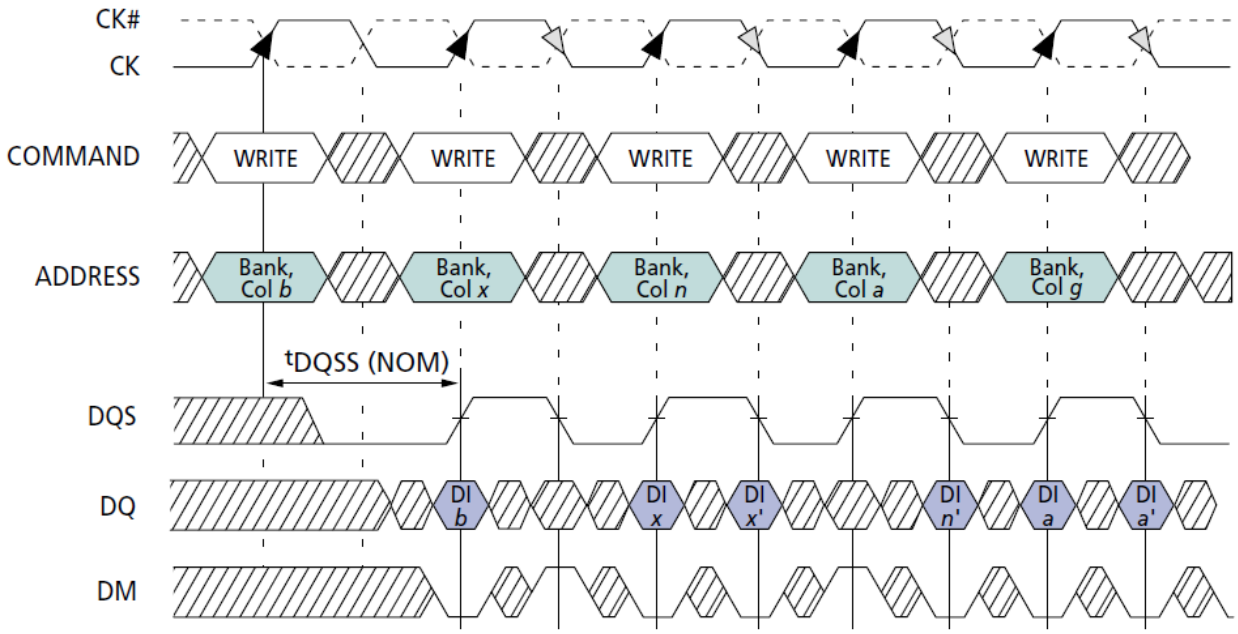


Figure 3. Minimum Data Time Slot for 2n-Prefetch Write. [Micron-01]

The data strobe timing pattern consists of a preamble, toggling, and postamble portion. Figure 4 shows the strobe pattern and alignment to data for read instructions, and Figure 5 shows the same for write instructions. The preamble portion provides a timing window for the receiving device to enable its data capture circuitry while a known/valid level is present on the strobe signal, thus avoiding false triggers of the capture circuit. Following the preamble, the strobes will toggle at the same frequency as the clock signal for the duration of the data burst. Each high transition and each low transition is associated with one data transfer. The low time following the last transition is known as the postamble. Most controllers have an internal clock running at twice the memory clock frequency, so generating a strobe shifted 90 degrees relative to data is fairly straightforward.

For read-to-read data bus transitions where the read instructions are from different physical banks of DRAMs, or read-to-write data bus transitions (i.e., transitions from one device driving the data bus to another device driving the data bus), there is a hand-off of strobe signals, and the full strobe pattern (including preambles and postambles) is needed from each source. The conservative approach is to space requests such that the postamble from the first source completes before the preamble from the second source begins. Consecutive read bursts from the same bank of DRAMs are achieved by extending the toggling portion of the data strobe pattern. (A postamble and preamble are not needed between consecutive read bursts from the same source.) Consecutive write bursts are also possible, even to different physical banks of DRAMs. The destination DRAMs for the second consecutive write burst has no way of knowing that a first write burst to another bank of DRAMs occurred. This means that the DRAMs must be capable of accepting different preamble timing, depending on whether there was prior write activity on the bus.

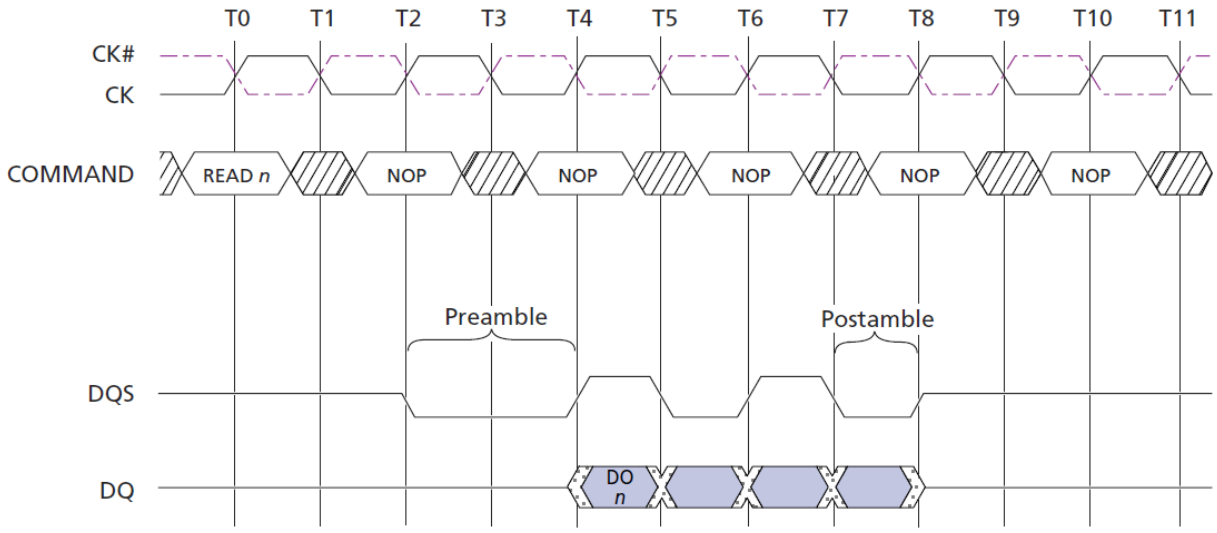


Figure 4. DQS Pattern for Read, Showing Preamble and Postamble. [Micron-01]

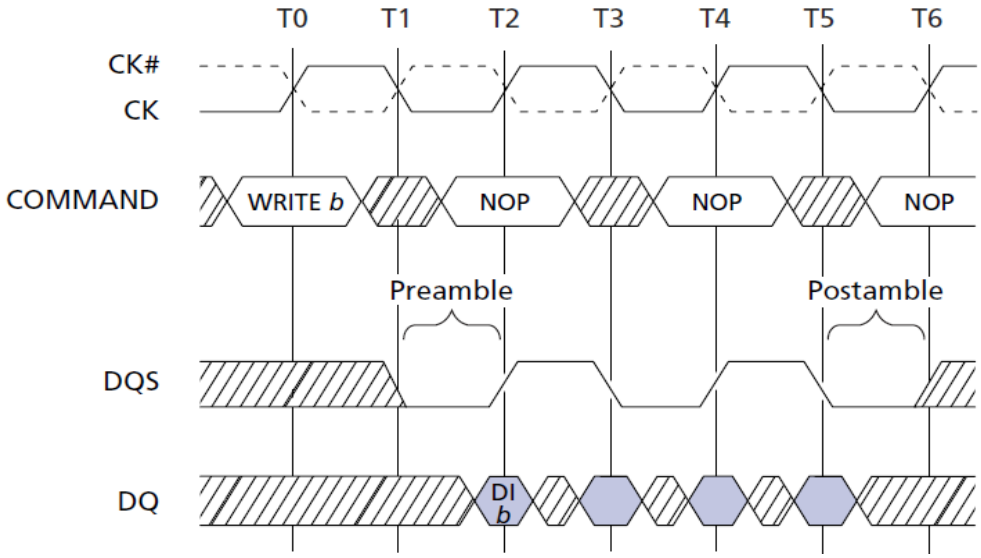


Figure 5. DQS Pattern for Write, Showing Preamble and Postamble. [Micron-01]

Different commands are applied to DDR SDRAM memories. Commands for configuration, read and write data, auto refreshing, precharge and others are issued to the memory by the inputs ports of it. The next table shows the configuration of the memory inputs to execute each command. [Samsung-01]

It is necessary to understand all this commands and their functions to analyze the reference model and to implement a verification environment, in order to stimulate and analyze the reference model and determine if the results are correct or incorrect. [Samsung-01], [Winbound-01], [Hynix-01].

COMMAND			CKEn-1	CKEn	$\overline{CS}$	$\overline{RAS}$	$\overline{CAS}$	$\overline{WE}$	$BA_{0,1}$	A_{10}/AP	A_{11}, $A_9 \sim A_0$	Note
Register	Extended MRS		H	X	L	L	L	L	OP CODE			1, 2
Register	Mode Register Set		H	X	L	L	L	L	OP CODE			1, 2
Refresh	Auto Refresh		H	H	L	L	L	H	X			3
	Self Refresh	Entry		L								3
		Exit	L	H	L	H	H	H	X			3
					H	X	X	X				3
Bank Active & Row Addr.			H	X	L	L	H	H	V	Row Address		
Read & Column Address	Auto Precharge Disable		H	X	L	H	L	H	V	L	Column Address	4
	Auto Precharge Enable									H		4
Write & Column Address	Auto Precharge Disable		H	X	L	H	L	L	V	L	Column Address	4
	Auto Precharge Enable									H		4, 6
Burst Stop			H	X	L	H	H	L	X			7
Precharge	Bank Selection		H	X	L	L	H	L	V	L	X	
	All Banks								X	H		5
Active Power Down		Entry	H	L	H	X	X	X	X			
					L	V	V	V				
		Exit	L	H	X	X	X	X				
Precharge Power Down Mode		Entry	H	L	H	X	X	X	X			
					L	H	H	H				
		Exit	L	H	H	X	X	X				
					L	V	V	V				
DM			H	X					X			8
No operation (NOP) : Not defined			H	X	H	X	X	X	X			9
					L	H	H	H				9

TABLE 2. DDR SDRAM Command Truth Table. [Samsung-01]

1.3. Verification Concepts

It is important to introduce the fundamental concepts of verification, in order to understand that the verification of a digital system is a process used to determine that an implementation behaves according to a given set of requirements (specs) for all possible scenarios.

When we think about digital system verification, the first think that come to mind, is the term "testbench". The term testbench usually refers to simulation code used to create a predetermined input sequence to a design and to observe the response. The verification challenge is to determine what input patterns to supply to the design and what is the expected output of a

properly working design when stimulated to those input patterns. However, verification is not a set of testbenches, verification is a process. [Janick-Bergeron-01]

There are two types of verification process for a digital system: Formal Verification and Functional Verification. Functional Verification probes the equivalence between two representations or models, the reference model and the functional model-RTL- using simulation tools. Formal Verification uses formal methods (mathematical processes, assertions checking…) to probe equivalence between two models. [Janick-Bergeron-01]

For this project, Functional Verification is used, since simulation tools are used.

Functional Verification process is based on two principles:

- Controllability: Ability of internal/external inputs to move the model from any initial state to another state: Stimulation.

- Observability: Ability of internal/external outputs to get the current state of the model for analysis: Responses.

Functional Verification uses these two principles to create simulation based verification. Stimuli are provided to exercise the HDL code. An environment is built to functionally verify the design by providing meaningful scenarios to observe and check that given certain inputs, the design performs the specification. [Janick-Bergeron-01]

A verification environment is typically composed of several types of elements:

- Testbench: A module, component or program description using a Hardware Description and Verification Language (HDVL) and object oriented programming techniques that contains all the verification instances, which is the top level implementation. The main objective is to enable the interaction with the device under verification (DUV). The testbench is defined by the strategies from the verification plan or test plan.

- Device Under Test (DUV): It is the RTL implementation of the system, full chip, block/cluster, or component that is going to be verified.

- Reference Model: It is a high level description of the system, full chip, block/cluster, or component to obtain the expected values that will be compared with the obtained values to declare a failed/passed test. A RM is defined by the spec interpretation and the verification plan requirements, which are the answer to the question, what must be checked? [Janick-Bergeron-01]

A RM is meant to be implemented using object oriented programming techniques, and a HDVL or other languages, with a high level of abstraction. If the RM's implementation is

difficult to implement or takes a lot of time, it is not recommended to implement one for a verification environment. The RM is also used to validate the elements of the verification environment. This is important when the RTL is not available yet.

- Interface: Provides the connectivity to/from the DUV and the RM by a hierarchical reference in the testbench to simplify the controllability, and the analysis during the simulation. An interface can be described as a structure or object that encapsulates the connectivity; it should be an instance in the testbench.

- Transaction: It is a single transfer of data or control between two systems. A transactor transforms transactions into interface level activity and vice versa by specific operations or methods. The most important transactors are:

* Drivers: A driver is an active element that transforms a transaction into interface level activity. It is an object that encapsulates the functionality using the minimum arguments and implemented as a method.

* Monitor: A monitor is a passive element that transforms the interface level activity into transactions to allow observability.

* Responder: A responder is a reactive element that transforms the interface level activity into transaction and response with other transaction. It can be considered like a driver with a monitor.

* Bus Functional Model (BFM): A BFM is a high level model that is used to emulate the circuit interfaces by transactions. In a BFM, stimulus generator, drivers, and responders are working together. BFM's are commonly used to emulate protocols into testbenches. In this project, BFMs will be used to stimulate the DDR SDRAM.

BFMs are common objects with methods to drive and respond to transactions with specific controls.

- Generation Elements: These elements are structures or objects that will provide data to create transactions. Data will be used by the transaction or interface layer to excite the DUV and Reference Model.

- Analysis Elements: These elements evaluate and measure the responses during the simulation using monitors. The following are analysis elements:

* Checker: A checker is an analysis element that determines whether the functionality of the DUV has been correct during the simulation using the information from the

transactors. They are implemented by process or tasks using a HDVL. Some checkers compare expected values and obtained results using RMs. Others checkers are based on monitors and others are based on assertions.

* Coverage: It is an analysis element that determines whether a functional area of the DUV has been exercised during the simulation using the information from the transactors. They are implemented by process or tasks using a HDVL. Coverpoints are used to obtain coverage, based on monitors of single conditions. [Janick-Bergeron-01]

- Tests: Tests objects are elements that will orchestrate the DUV's and the RM's activity using transactors, generation and analysis elements. A test is also used to construct different scenarios or testcases, where each scenario is used to validate different functional areas.

1.4. Verification Methodology

A verification methodology is a set of procedures designed to verify circuits' functionality by simulation. This methodology provides a structure for developing testbench architectures and utilities that can be used to make easier the verification tasks:

-Controllability: Manipulating the input circuit interface - stimulus.

- Observability: Capturing all the circuit activity - monitors.

- Analysis: Making a decision about the correctness (Does it works?) and completeness of the verification (Are we done?).

The main objective of the verification methodology is to obtain a real verification environment where the digital system will be used.

Current verification methodologies are based on layered architecture that allow:

- Abstraction: Grouping simple and repetitive tasks in a generic task.

- Reuse: Using generic tasks to obtain more complex tasks.

-Modularity: Making easier the integration of new elements.

-Interfaces: Making easier the interconnection of new elements.

-Automation: Reporting results instead of wave form analysis.

The methodology based on layers that is used in this project is shown in Figure 6.

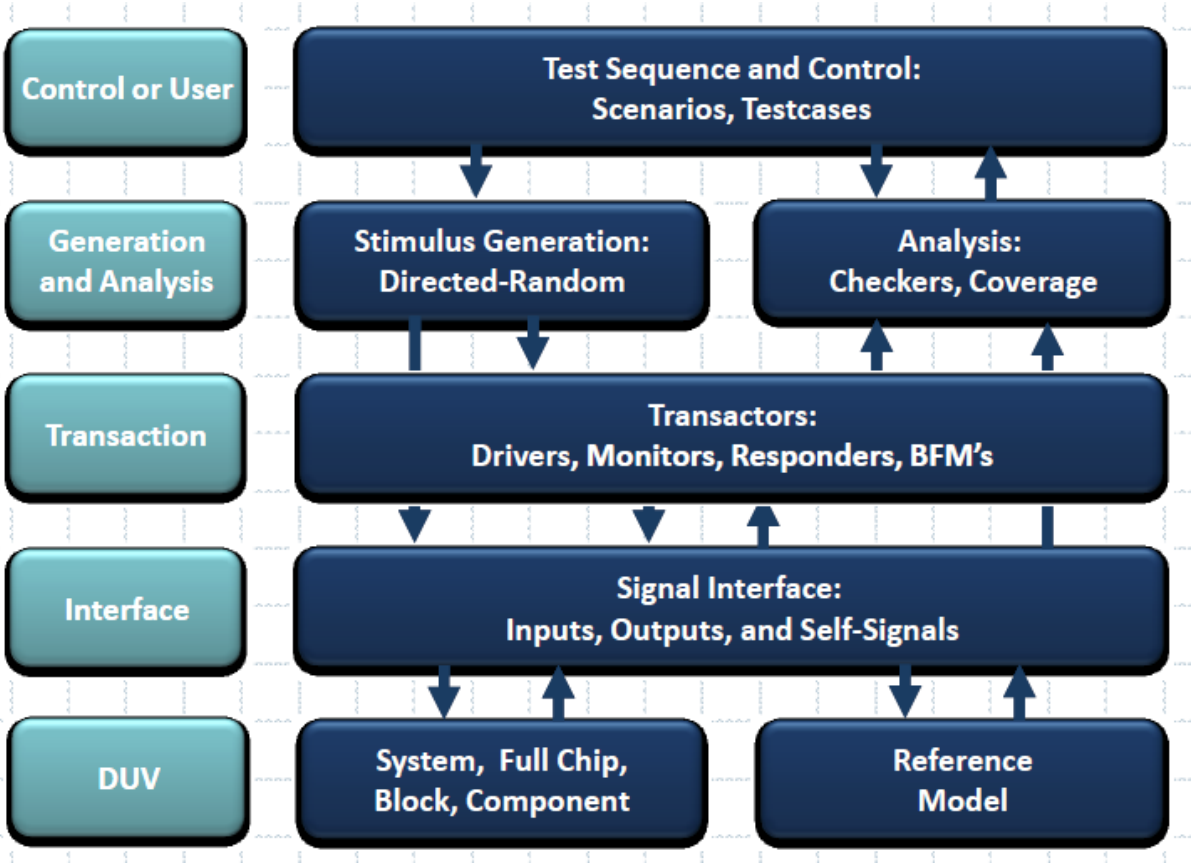


Figure 6. Verification Methodology Based on Layers. [Mark-Glasser-01]

This verification methodology integrates all the verification elements defined in the past section. Each of these layers contains specific verification elements that communicate with elements of others layers, to validate the digital system, where new elements can be easily added and allowing modularity. The Interface layer interconnects the transactors with any amount of DUVs and RMs in an easily way. Transaction layer transform the data generated in the generation layer to bits, so it can be transferred to DUV layer via Interface layer. Generation and analysis layer are structures and objects that provide data to transactors or receive data from transactors. These structures and transactors can be grouped to form generic tasks for abstraction and can be reused for any verification project where the same structures or transactors are used.

When the implementation of each layer is done, and the communication between layers is realized, then a verification environment is obtained, where generation and analysis tasks are controlled by the User layer. Using an object oriented programming language like System

Verilog permits to encapsulate each layer like an object and communicate data between the layers automatically. Also, System Verilog permits to report results which can be easy and rapidly analyzed by a verification engineer.

2. Analysis / Improvement of the Reference Model

DDR SDRAM memories manufacturers provide reference models of their memories on languages like VHDL, Verilog, System Verilog and others, so their customers can use them to extract their functionality for a specific project. It is important to verify these reference models, to check if they are correct and complete, according to the protocol standard. After being verified, these reference models are used with a RTL memory controller model, to assist in its verification.

The reference model that is used in this project is provided by Micron Technology Inc, for DDR SDRAM memories of 4, 8 and 16 data bits. [Micron-02] The DDR SDRAM reference model features are the next ones:

- Two data transfers per clock cycle.
- Bidirectional data strobe (DQS).
- Four banks operation.
- Differential clock inputs.
- Delay Locked Loop (DLL).
- Column Address Strobe (CAS) Latency (1.5, 2, 2.5 3).
- Burst Length (2, 4, 8).
- Burst Type (Sequential and Interleave).
- All inputs except data and DM are sampled at the positive going edge of the system clock.
- Data I/O transactions on both edges of data strobe.
- Edge aligned data output, center aligned data input.
- Lower Data Mask (LDM), Upper Data Mask (UDM) and Data Mask (DM) signals for write masking only.
- Auto Refresh.
- Timing parameters can be configured.

- Address parameters can be configured.
- No Power Down.
- No Self Refresh.

2.1. Reference Model Structure

The reference model structure is mainly divided in three blocks: processes, tasks and width, setup and hold time's checkers. The processes are initial or always type, which wait for an event to occur. Tasks are executed inside the process. Checkers for width, setup and hold times use special Verilog timing system tasks. The operation of each block will be explained in the next section.

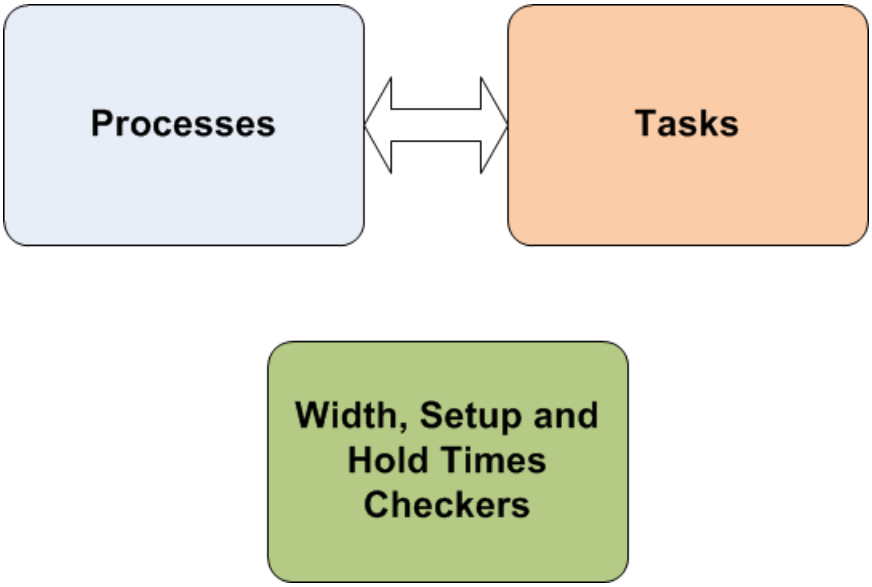


Figure 7. Reference Model Structure.

There are 7 main processes in the reference model. Each process has a special function inside the reference model. Some of these processes activate tasks and uses data that are processed inside the tasks. The processes are shown in Figure 8.

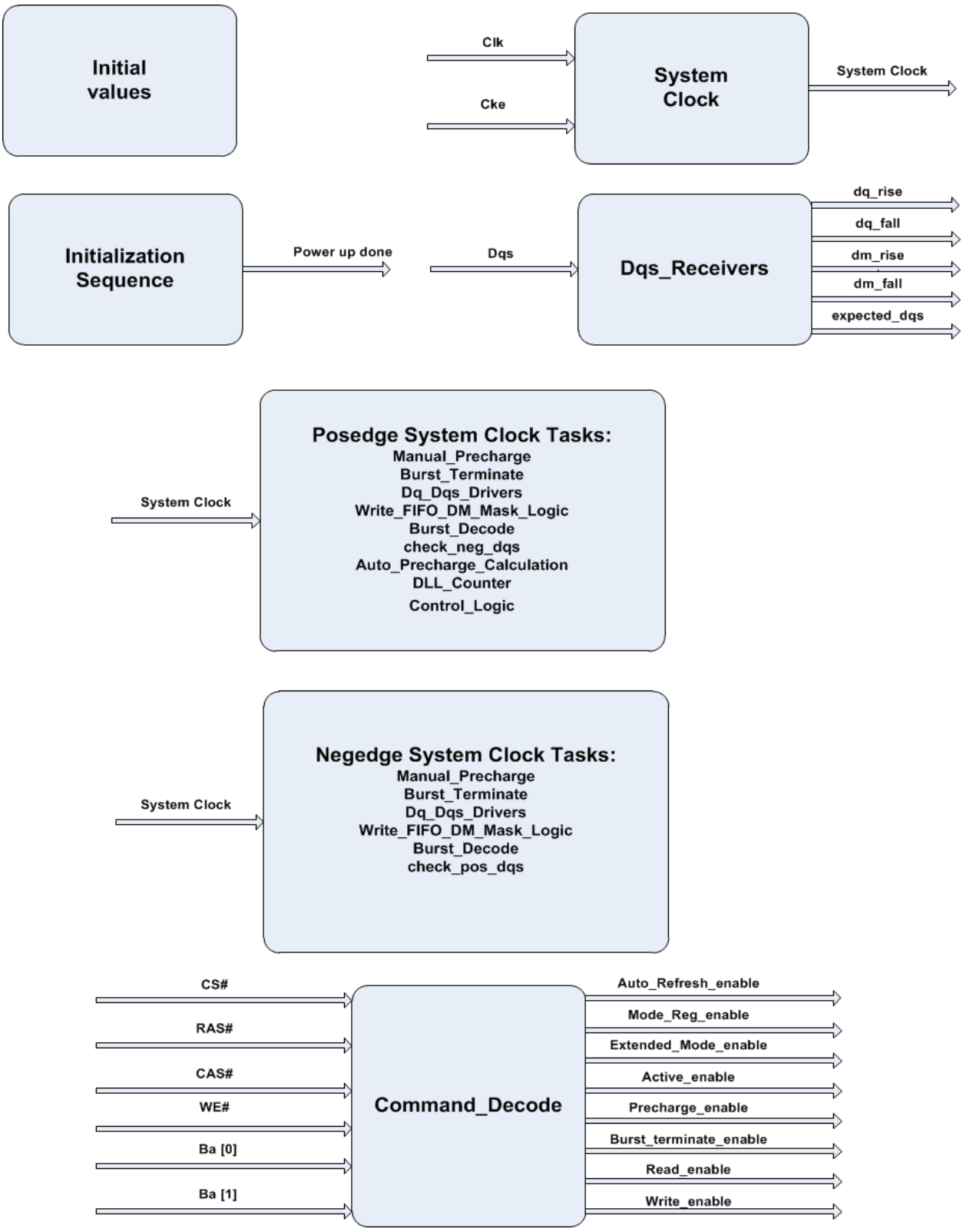


Figure 8. Reference Model Processes.

There are 12 tasks in the reference model. Each task has a special function inside the reference model. All the tasks are called by one or more processes and use data that are processed inside the processes or others tasks. Some of these tasks have checkers inside them, to analyze data that is easier to analyze in the reference model than outside of it. The tasks are shown in Figure 9.

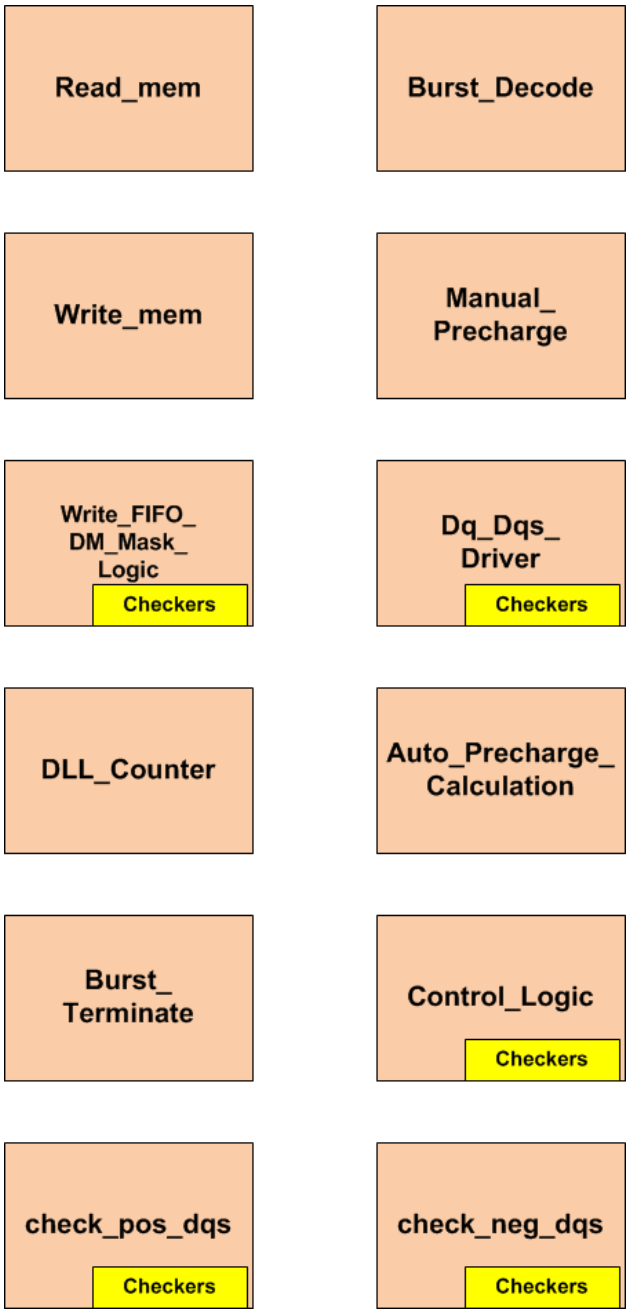


Figure 9. Reference Model Tasks.

2.2. Operation of Each Block/Function of the Reference Model

In this section, the operation of each block shown in the previous section will be described.

2.2.1 Processes

Inside the reference model, there are 7 processes which each one realizes specifics functions. The processes are:

1. - Initial Values: This initial process sets the initial values to the data types that require an initial value. All data are set to a 0 value, except Activate data types; because it is assumed that the memory is initially activated. Also, DQS and DQ signals have a "z" value, because that is the default value when data are not being read or written.

2. - Initialization Sequence: This initial process determines if the correct power up and initialization sequence of the memory have occurred. Read and write data are not possible if the memory is not initialized. The correct sequence for memory power up and initializations is [Micron-02], [Winbound-01]:

- Maintain CKE at a low state (all other inputs may be undefined.)
- Start clock and maintain stable condition for a minimum of 200us.
- The minimum of 200us after stable clock (CK, CK#), apply NOP command & take CKE high.
- Issue precharge commands for all banks of the device.
- Issue EMRS to enable DLL.(To issue "DLL Enable" command, provide "Low" to A0, "High" to BA0 and "Low" to BA1)
- Issue a mode register set command for "DLL reset" (To issue DLL reset command, provide "High" to A8 and "Low" to BA0). The additional 200 cycles of clock input is required to lock the DLL.
- Issue precharge commands for all banks of the device.
- Issue 2 or more auto-refresh commands.
- Issue a mode register set command with low to A8 to initialize device operation.

3. - System Clock: System clock flip flop logic. If Cke is enabled, then the System Clock is on. Since all commands depend if the previous cycle was Cke enabled, flip flop logic is implemented to generate the System clock for the reference model.

4. - Dqs Receivers: Receive the DQ and DQS data signals, when both edges of signal DQS occur. DQ, DM and DQS signals are stored with this process when a write is executed. The DQS signal stored is checked by a task, to determine if it was sent correctly.

5. - Posedge System Clock Tasks: This always process activates tasks that drive data, executes the command sent to the memory and runs the DLL counter task. Also, Precharge and Activate task, called Auto Precharge Calculation is also executed in by this process, since all commands are executed in the positive edge of the clock.

6. - Negedge System Clock Tasks: This always process activates the tasks that depends of the negative edge of the system clock. These tasks are the ones that drives data, data mask and data strobe signals since DDR SDRAM data signals depends on both clock edges.

7.- Command Decode: This always process decodes the DDR SDRAM command inputs and bank inputs, in order to detect which command is being sent to the reference model. The decoding is based on the lists of commands shown in Table 2. The command that is detected is activated and executed by the control logic task.

2.2.2 Tasks

Inside the reference model, there are 12 tasks which each one realize specifics functions. The tasks are:

1. - Read memory: Data is read from memory array. Bank, row and column address are provided as inputs (Column address represent the LSB and Bank address present the MSB of the address input). Data stored in the input address is returned as output.

2. - Write memory: Data is stored in memory array. Bank, row and column address are provided as inputs (Column address represent the LSB and Bank address present the MSB of the address input). Data to store is also provided as input, which is stored in the input address.

3. - Burst Decode: Determines if the burst type is sequential or interleave. Also, burst length logic is done in this task.

4. - Manual Precharge: Logic to interrupt a Read with a Precharge command.

5. - Write FIFO and DM Mask Logic: This task performs all the logic to write data to memory, when a write command is issued. It determines the address where the data is going to be written. Performs the DM mask logic, where data that is masked with DM signal is not written to memory and manages pipeline for write bursts. Also, check if the times between write commands and between write and read commands are satisfied.

6. - Dq and Dqs Driver: This task performs all the logic to read data from memory, when a read command is issued. It determines the address where the data is going to be read, generates the DQS signal and manages pipeline for read bursts. Also, checks if an invalid address is sent to the memory for data reading.

7. - DLL Counter: This task counts the 200 cycles of clock input to lock the DLL. The count starts when a DLL reset is issued.

8. - Auto Precharge Calculation: Logic and implementation of read/write with auto precharge. For Read commands with auto precharge, the memory starts internal precharge burst length/2 clock later from a read with auto precharge command when tRAS is satisfied. For Write commands with auto precharge, the memory starts internal precharge burst length/2 clock plus write latency duration plus tWR later from a read with auto precharge command when tRAS is satisfied.

9. - Burst Terminate: Logic to stop a read burst depending on the CAS latency value.

10. - Control Logic: This task is in charge of determining, which actions are going to be implemented, depending of the command sent to the memory.

If an Auto Refresh command is detected, it checks that all banks are precharged. If not, a checker will determine an error. Also, it checks that times between other commands and auto refresh command are satisfied.

If a Load Extended Mode Register command is detected, it checks that all banks are precharged. If not, a checker will determine an error. Also, it determines the DDL enable logic and checks that times between other commands and load extended mode register command are satisfied.

If a Load Mode Register command is detected, it checks that all banks are precharged. If not, a checker will determine an error. Also, it determines the DDL logic, burst type, CAS

latency, burst length and checks that times between other commands and load mode register command are satisfied.

If Activate command is detected, it checks that the bank that is going to be activated is precharged. If not, a checker will determine an error. If the bank is previously precharged, it will activate it and will activate the row indicated in the DDR SDRAM address input. Also, it checks that the times between other commands and activate command are satisfied.

If a Precharge command is detected, it checks that the bank that is going to be precharged is activated. If not, then a NOP will occur. If the bank is previously activated, it will precharge it. If precharge all banks feature is enabled, then it will precharge all banks. Also, it checks that the times between other commands and activate command are satisfied.

If a Burst Terminate command is detected, it implements the logic to terminate a read burst. Also, it checks if a write to memory or a read with auto precharge are being executed. If one of these is detected, a checker will determine an error, since it is illegal to terminate a write or a read with auto precharge.

If a Read command is detected, it checks that the bank that is going to be read is activated. If not, a checker will determine an error. It implements the logic to read data with others tasks and to determine if auto precharge is needed. Also, it checks that times between other commands and activate command are satisfied. A read command can interrupt a write, but it is illegal to interrupt a read with auto precharge.

If a Write command is detected, it checks that the bank that is going to be written is activated. If not, a checker will determine an error. It implements the logic to write data with others tasks and to determine if auto precharge is needed. Also, it checks that times between other commands and activate command are satisfied. It is illegal for a write command to interrupt a read with auto precharge, a read without burst termination and a wirte with auto precharge.

11. - Check Posedge Dqs: Task that checks if the DQS signal is sent correctly during a write command. Checks only the positive edges of Dqs signal.

12. - Check Negedge Dqs: Task that checks if the DQS signal is sent correctly during a write command. Checks only the negative edges of Dqs signal.

2.2.3 Width, Setup and Hold Times Checkers

Verilog timing system tasks, $widht and $setuphold, are used to check if width, setup and hold AC timing parameters are satisfied.

DQS-in high level width, DQS- in low level width, Setup and Hold times for each DDR SDRAM input to Clk are checked.

2.3. Reference Model Improvement

The DDR SDRAM standard provided by JEDEC and different datasheets were used for the understanding of the DDR SDRAM protocol. The reference model provided my Micron Technology was analyzed for this project, and based in various DDR SDRAM memories manufacturers, new features were added to the reference model.

The reference model improvements are:

- Start clock and maintain stable condition for a minimum of 200us feature for memory power up on initial sequence process.
- Issue Extended Mode Register Command to enable DLL for memory initialization on initial sequence process.
- Issue a Mode Register Set Command with low to A8 for memory initialization on initial sequence process.
- CAS Latency of 1.5 clock cycles for Samsung memories on control logic task.
- DLL Enable and Reset Logic on control logic.
- Micron and Samsung memories Burst Length and CAS Latency decoding on control logic task.
- Timing violation checkers were added.
- Macros for timing violation checkers.
- DQS logic for write burst on check_dqs tasks.
- Verification Messages.

3. Verification Environment

This section describes the verification environment structure implemented for this project.

3.1. Verification Environment Structure

The verification environment structure is presented in Figure 10:

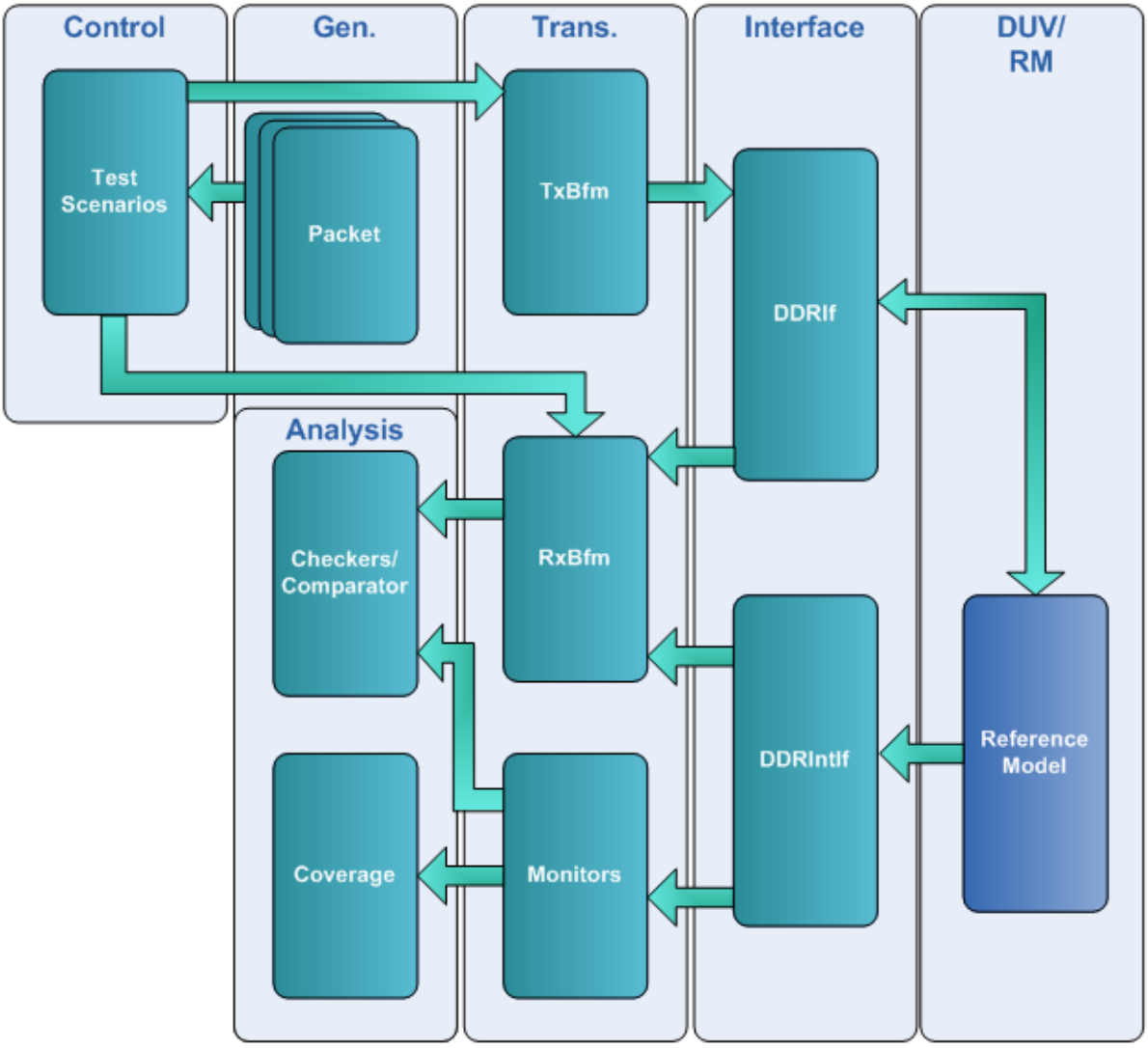


Figure 10. Verification Environment Structure.

The verification environment structure is formed by all the layers presented in the verification methodology (Section 1.4). Each layer has elements that perform the specific functions of each layer. DUV/RM layer contains the module to verify. Interface layer communicates the RM with the transaction layer. Transaction layer contains transactors that transmit (TxBfm) bits to the RM and transactors that receive (RxBfm) and monitor (Monitors) bits from the RM. Generation layer contains structures (Packet) that provides data to transactors via control layer. Analysis layer receives data from transactors or monitors to check the correct functionality and to obtain coverage. Control layer contain test scenarios that will be configured with user defined parameters. These test scenarios will control all the transactors to verify the RM.

When the verification environment is ready, then it is used by a testbench, which is the top level implementation that interacts with the verification environment.

The verification environment and testbench for this project are implemented using System Verilog and object oriented programming techniques. The testbench is a module that will contain all the element instances of the verification elements that conforms the verification environment's layers.

The HDVL verification environment structure is presented in the Figure 11.

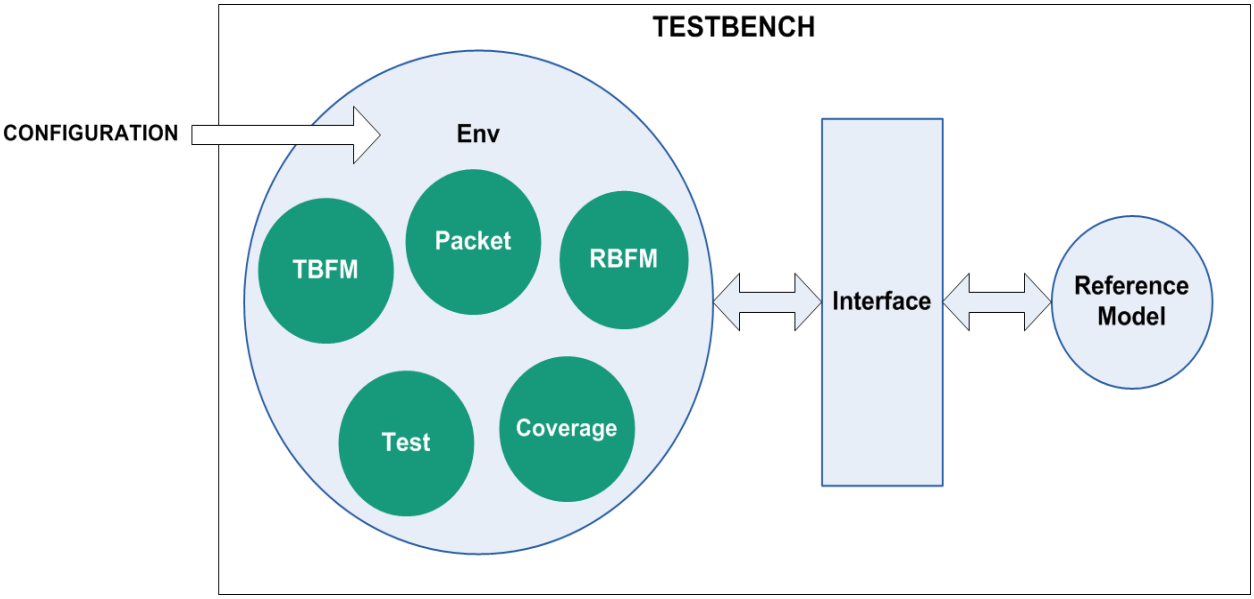


Figure 11. HDVL Verification Environment Structure.

Objects, interfaces or modules, which are the elements of the verification environment, are instanced in the testbench or in the environment class. The environment object (Env) shown in Figure 11, is a class that contains various classes inside, instanced like objects. Inside this class, the communication between these objects occurs using object references. These objects inside Env class, contain virtual interfaces that connects the objects with the interfaces. The interfaces are in charge to transform transactions of the Env instances into data bits, which are sent to the reference model. Also, the same interfaces receive data bits from the reference model, and send this information to Env objects for analysis.

There are some configuration features, which configure the environment according to the test scenario that will be executed. With this configuration, the verifier can choose which test scenario will be executed and which test content will be generated to stimulate de reference model. [See Section 4.4]

3.2. Operation of Each Block/Function of the Verification Environment

In this section, the operation of each block shown in the verification structure (Figure 10), environment structure (Figure 11) and the testbench will be described.

- Reference Model: The reference model is the element to be verified in this project, since DUV is not presented in this project. The reference model is implemented as a module, and is instanced in the testbench.

- Interface: The connectivity with the reference model module is implemented using interfaces. Interfaces allow virtual mode for use with classes, allowing the connection between objects and the reference model or DUV. DDRIf Interface allows the connection with the reference model inputs/outputs, and DDRIntIf allows the connection with internal signals of the reference model.

- TxBfm: This transaction layer block contains all the functions to stimulate the reference model. TBFM class encapsulates this block inside the verification environment.

- RxBfm: This transaction layer block contains elements to receive and store data from the test scenarios and from the reference model. The received data is stored with the objective to

be checked or compared. RBFM class encapsulates this block inside the verification environment.

- Monitor: This transactor monitors the responses sent by the reference model with the objective to be analyzed. RBFM class encapsulates this block inside the verification environment.

- Packet. This generation layer block contains structures that provide data to be used by the transactors to stimulate the reference model. Random and directed data can be generated by this block. Packet class encapsulates this block inside the verification environment.

- Checkers/Comparators: Analysis elements that check the functionality of the reference model during simulation using information from Monitor and RxBFM block. RBFM class encapsulates this block inside the verification environment.

- Coverage: Analysis block that observe and analyze the reference model to determine if the functional areas have been exercised (completeness). This block generates a document that shows how many times the functional areas have been exercised. Coverage class encapsulates this block inside the verification environment.

- Test Scenarios: Control layer block used to construct different test sequences. It uses the utilities from the transaction, generation and analysis layers, like packet, TxBfm and RxBfm tasks or structures to verify to reference model. Test class encapsulates this block inside the verification environment.

- Environment: The environment class, Env, encapsulates Test, Packet, TBFM, RBFM and Coverage classes, where each class is instanced as an object. This class establishes the communications between these objects and executed their tasks.

- Testbench: Top level module where the reference model, interfaces and environment classes are instanced. It generates the clock signal for the reference model, connects the environment with the interfaces and reference module instance and executes the verification environment tasks.

4. Verification Plan

Functional Verification is the flow that allows determining the correct behavior of an implementation for all possible scenarios.

The process starts with the specification document (Spec), continues with the definition, implementation, and integration of verification elements, and finishes when the verification goals are satisfied. In order to achieve this, it is necessary to define a methodology that follows this process.

The first step in the methodology is receiving the general definition of the digital circuit. It mainly contains the functional description that allows designing and verifying the circuit. In this project, DDR SDRAM datasheets contains the definition and specifications of the memory.

Then, a Verification Plan is defined. A Verification Plan is just a document that defines what and how will be verified: elements obtained by interpretation of the Spec.

This Verification Plan must define the Verification Strategies, the Functional Areas and the Verification Environment elements. Also, the verification plan should provide a set of requirements for environment design, reference model and test content. [Janick-Bergeron-01]

4.1. Verification Strategies

A verification strategy is necessary to verify a digital system with detail, for the maximum error detection. This strategy is composed by the functional verification approach, hardware verification hierarchy, synchronization and comparison strategy with the reference model. The verification strategy defined for this project is:

- *Functional Verification Approach:*

The approach for this project is defined as "Gray Box". [Janick-Bergeron-01]

DDR SDRAM reference model will be verified like a "gray box", where input and output signals are verified, and also a few of the internal signals, interconnections or variables that conforms the reference model, but not all. The reference model models the behavior of the input and output signals of the memory, depending the command sent to the memory.

- *Synchronization and Comparison Strategy with the RM:*

Since this project verifies only the reference model, and a DUV is not presented, there isn't comparison between this two. It is expected to adjust time delays for the stimulation and

responses of DUV, with the outcome signals of the RM, to compare in real time simulation, the observed data with the data from the RM.

- *Hardware Verification Hierarchy:*

Block/Cluster:

* DDR_SDRAM Reference Model without controller.

System: DDR_SDRAM

4.2. Functional Areas

A functional area is just a functional description of features obtained from the spec by interpretation. It is a short description in terms of what must be verified: the correctness and the completeness. The functional areas for this project are obtained from the DDR SDRAM protocol. The functional areas are:

- Write data to memory.
    * Burst Write data to memory.
    * Write data with Auto Precharge.
- Read data from memory.
    * Read Burst data from memory.
    * Read data with Auto Precharge.
- Write and Read data alternately.
- Memory Power Up and Initialization.
- Memory Bank Precharge.
- Memory Bank Activate.
- Memory Auto Refresh.
- Mode Register Load and Configuration.
- Extended Mode Register Load and Configuration.
- Burst Terminate.

4.3. Verification Environment

The verification environment has the objective to define the elements to control and observe the functional areas defined for the DDR SDRAM.

For this project, the verification environment will consist in the following elements:

- Reference Model
- Interfaces
- Stimulation
- Monitors
- Checkers
- Coverage

4.3.1 Reference Model

The reference model is the element to be verified in this project. The reference model is implemented as a module, where an instance of it is written in the testbench, in order to stimulate its inputs and observe the results, where checkers, monitors and coverage will determine if the reference model is correct and complete.

4.3.2 Interfaces

The connectivity of a module is done by a group of input/output ports or interfaces. The interfaces have the function to extract all the connectivity of the DUV and the reference model and connect them with the verification environment. Two interfaces are defined, one that extracts the connectivity of DDR SDRAM reference model inputs/outputs, and other that extracts the connectivity of internal signals of the reference model.

The interfaces modports, which define the direction for each signal and restrict their access, are:

- TX modport: Interface modport to stimulate the DDR SDRAM Reference Model.

-MONITOR modport: Interface mordport to monitor DDR SDRAM Reference Model input/output signals and internal signals of the reference model.

4.3.3 Stimulation

The stimulation consists in implementing tasks or functions that stimulate the DDR SDRAM reference model. The stimulations defined for the reference model, according to the functional areas are:

- Power Up and Initialization: Active element that stimulate the memory for the correct power up and initialization of the memory.

- Load Mode Register/ Extended Mode Register: Active element that access and configures the mode register or extended mode register.

- Memory Refresh: Active element that is responsible for refreshing the memory. Inputs are configured so the memory detects an Auto Refresh command.

- Burst Terminate: Active element that is responsible for terminate a burst read or a burst write. Inputs are configured so the memory detects a Burst Stop command.

- Precharge: Active element that is responsible for precharge a specific bank or all banks of the memory. Inputs are configured so the memory detects a Precharge command.

- Activate: Active element that is responsible for activate a specific bank and a specific row of the memory. Inputs are configured so the memory detects a Precharge command.

- Write Data: Active element that is responsible for writing data to the memory, generating the DQS signal depending on the burst length value. DM bits pattern and Bank and column address are defined in this task. Also, the option for write with auto precharge is supported by this task.

- Read Data: Active element that is responsible for reading data from the memory. Bank and column address are defined in this task. Also, the option for read with auto precharge is supported by this task.

- Nop Command: Active element that is executed after others memory commands. It is used to generating the proper delays between each memory command, avoiding timing violations between DDR SDRAM commands.

- Deselect Command: Active element that is responsible for maintaining the memory in a idle state.

4.3.4 Monitors

Monitors are verification elements that observe signals to active events. These events are used by checkers and for coverage elements to analyze the results of the reference model. The monitors defined for the reference model, according to the functional areas are:

- Monitor for memory Commands: Memory command inputs are observed by this monitor, to detect which command was sent to the memory.

- Monitor for memory Power Up and Initialization: A signal is observed by this monitor, to determine if the power up and initialization of the memory is done.

- Monitor for memory Read: A signal is observed by this monitor, to determine if a read command was sent to the memory. If a read command is detected by this monitor, signals for burst type, latency and burst length are observed for analysis.

- Monitor for memory Write: A signal is observed by this monitor, to determine if a write command was sent to the memory. If a write command is detected by this monitor, signals for burst type, and burst length are observed for analysis.

- Monitor for memory Active: A signal is observed by this monitor, to determine if a activate command was sent to the memory. If a activate command is detected by this monitor, signals for banks and rows address are observed for analysis.

- Monitor for memory Precharge: A signal is observed by this monitor, to determine if a precharge command was sent to the memory. If a precharge command is detected by this monitor, signals for which bank is precharged are observed for analysis.

- Monitor for Burst Terminate: A signal is observed by this monitor, to determine if a burst terminate command was sent to the memory. If a burst terminate command is detected by this monitor, a signal to determine the latency is observed for analysis.

- Monitor for memory DQS signals: DQS signals are observed for analysis when a read command is detected. Signals for burst length and latency are observed, for a proper DQS monitoring.

- Monitor for memory DQ signals: DQ signals are observed for analysis when a read command is detected. Signals for burst length and latency are observed, for a proper DQ monitoring.

4.3.5 Checkers

For evaluating the results during the simulation, checkers are implemented, which have the objective of analyzing the DDR SDRAM functions, using the monitors defined previously. Checkers determine if the results observed are correct or incorrect. The checkers defined for the reference model, according to the functional areas are:

- Checker for memory Power Up and Initialization: Analysis element that verifies the correct initialization of the DDR SDRAM memory.

- Checker Burst Terminate: Analysis element that verifies the correct burst termination of a memory read.

- Checker for Write and Read memory data: Analysis element that verifies that data was written correctly to the memory. This is verified by reading the same memory address, after being written and obtained the same data that was written.

- Checker for memory DQS signals: Analysis element that verifies that the DQSs signals are generated correctly by the memory reference model.

4.3.6 Coverage

The coverage defined for this project permits to know how many times the functional areas have been exercised. Coverpoints are defined to know which functional areas must be exercised, in order to obtain a complete verification:

- Write data to memory.
- Read data from memory.
- Write and read data with burst data length of 2, 4 and 8.
- Write and read data with burst sequential type and interleave type.
- Read data with all different latencies.

- Active all banks.
- Burst terminate with all different latencies.
- Precharge all banks.
- Power Up and Initialized memory.
- Load Mode Register.
- Load Extended Mode Register.
- Write with auto precharge.
- Read with auto precharge.

4.4. Test Content

4.4.1 Test Scenarios

Different test scenarios are defined for this project, where each scenario has the objective to directly verify one or some functional areas in specific. These scenarios are the typical test scenarios applied to memories. [Carlos-Flores-01]

The test scenarios are:

- Power Up and Initialization of DDR SDRAM: It performs the initialization of the memory, using the correct initialization sequence.

- Writing and reading data in all memory locations: Write commands are sent to the memory without delays between them and then, read commands are executed to verify that the written data is the same as the read data.

- Writing and reading data alternately in all memory locations: A write command is sent to the memory and then, a read command is sent to the same memory location, to verify that the data was correctly written and read between write and read commands. This test scenario is important because the Dqs signal behavior is different from the previous described test scenario.

- Random Command generation: Write and read commands are sent to the memory with or without precharge, with random burst lengths, random burst types, random latencies, to random memory locations.

4.4.2 Random Generation Elements

Random elements are defined within the class of packet generation. A packet object consists of random data types, which are used as variables in the stimulation.

This packet contains all the memories parameters that can be configure and sent to the memory.

Random banks, rows and columns address, data, data mask bits, latency, burst types, burst lengths and commands can be randomized and sent to the memory as random stimulus.

4.4.3 Directed Generation Elements

Constraints represent elements designed to realize directed tests. The conditions defined by these constraints, permit to perform directed tests according to what specifically we want to verify.

Within the verification of DDR SDRAM, constraints are needed because commands cannot be sent randomly, because there are several conditions that can eventually lead to violations within the memory function.

5. Verification Results

This section presents the results obtained by the verification of the DDR SDRAM reference model, using the verification elements defined for this project. Waveforms and text messages will be shown for results. Text messages are displayed by the verification environment and by improvements to the reference model, which are easy to analyze than waveforms, for verification purposes. Each message displays the process or tasks that is executed, the class that call the task and useful information for analysis.

5.1. Stimulation and Analysis

Stimulation and analysis results will be presented by test scenarios.

- The first test scenario performs the improved initialization of the memory, using the correct initialization sequence. The results of this test are shown in Figure 12 and 13. Each arrow in Figure 12, represent each of the sequence steps for DDR SDRAM power up and initialization. A checker is implemented to check that the power up and initialization are executed correctly. If the power up and initialization sequence is not executed correctly, an error message will be displayed if a write or read command is issued. Figure 13 shows the coverage report for the coverpoint Power Up and Initialized memory, which is covered since it was executed.

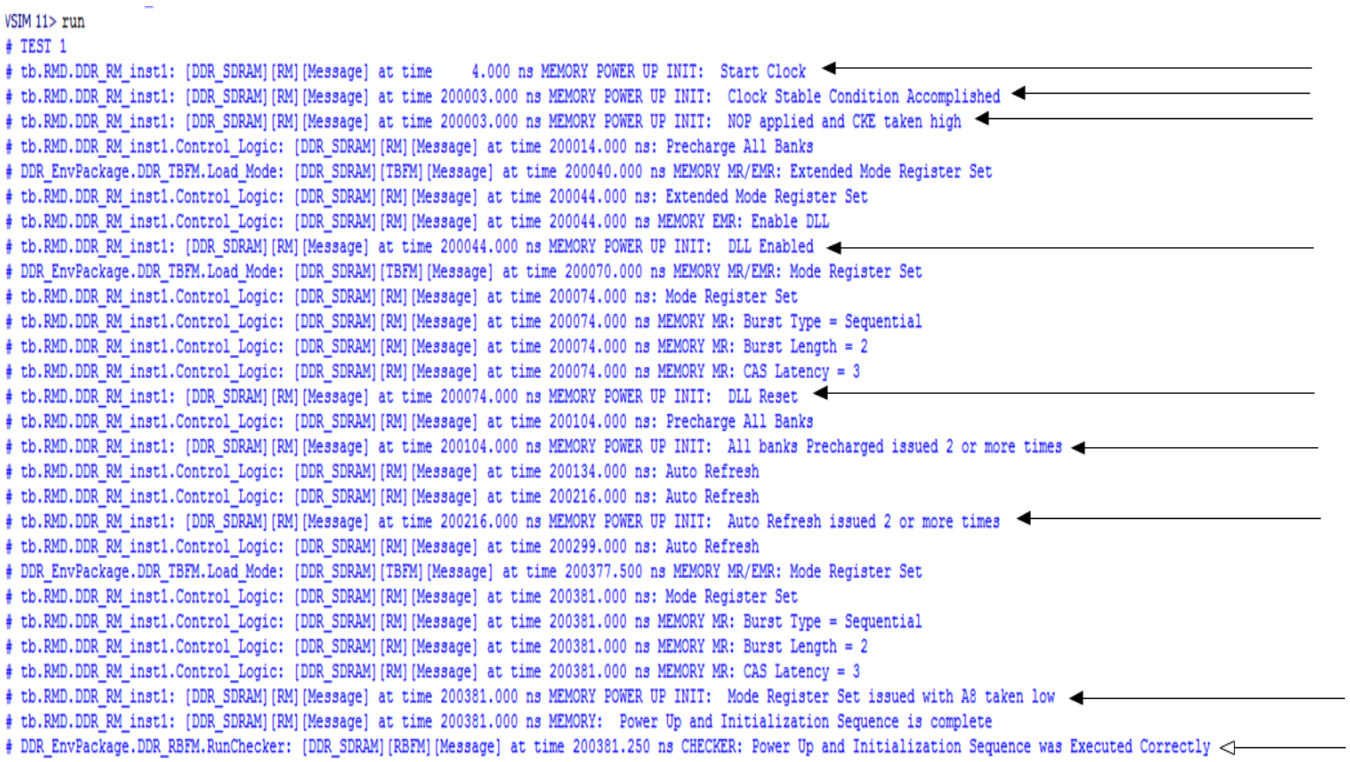


Figure 12. Test for Power Up and Initialization Sequence.

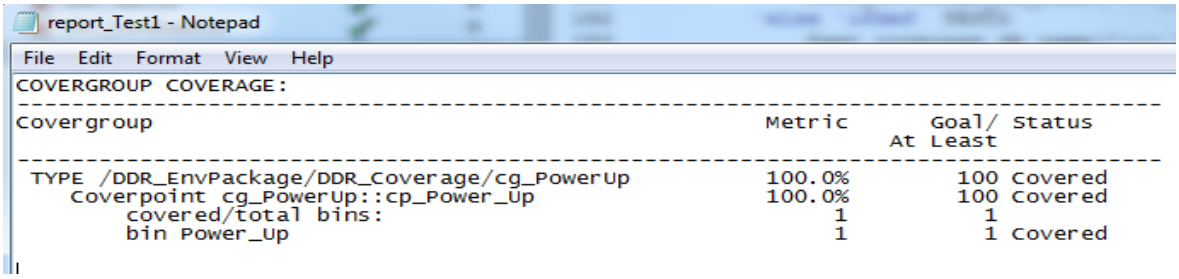


Figure 13. Coverage for Power Up and Initialization Sequence.

- In the second test scenario, write commands are sent to the memory without delays between them and then, read commands are executed to verify that the written data is the same as the read data. In this scenario, the power up and initialization of the memory is also executed, since it is necessary for the operation of the memory.

Figure 14 shows the test sequence when write commands are sent to the memory. First the Mode Register is configured with sequential burst type, burst length of 4 and CAS latency of 3 clock cycles, as a result of the decoding improvement for the reference model. The user can choose the mode register configuration values to be random or directed. Then, activation of a bank and a row it is necessary for writing. Data is written with a burst length of 4, like is shown in Figure 15. The waveform of Figure 15 shows the data being sent to the memory, where Dqs signal is sent correctly for a write burst. When DM signal is high, the data sent to the memory is masked and is not written to memory. Figure 14 shows that data sent to memory is displayed as "xxxx", meaning that is ignored by the memory because is masked by the DM signal.

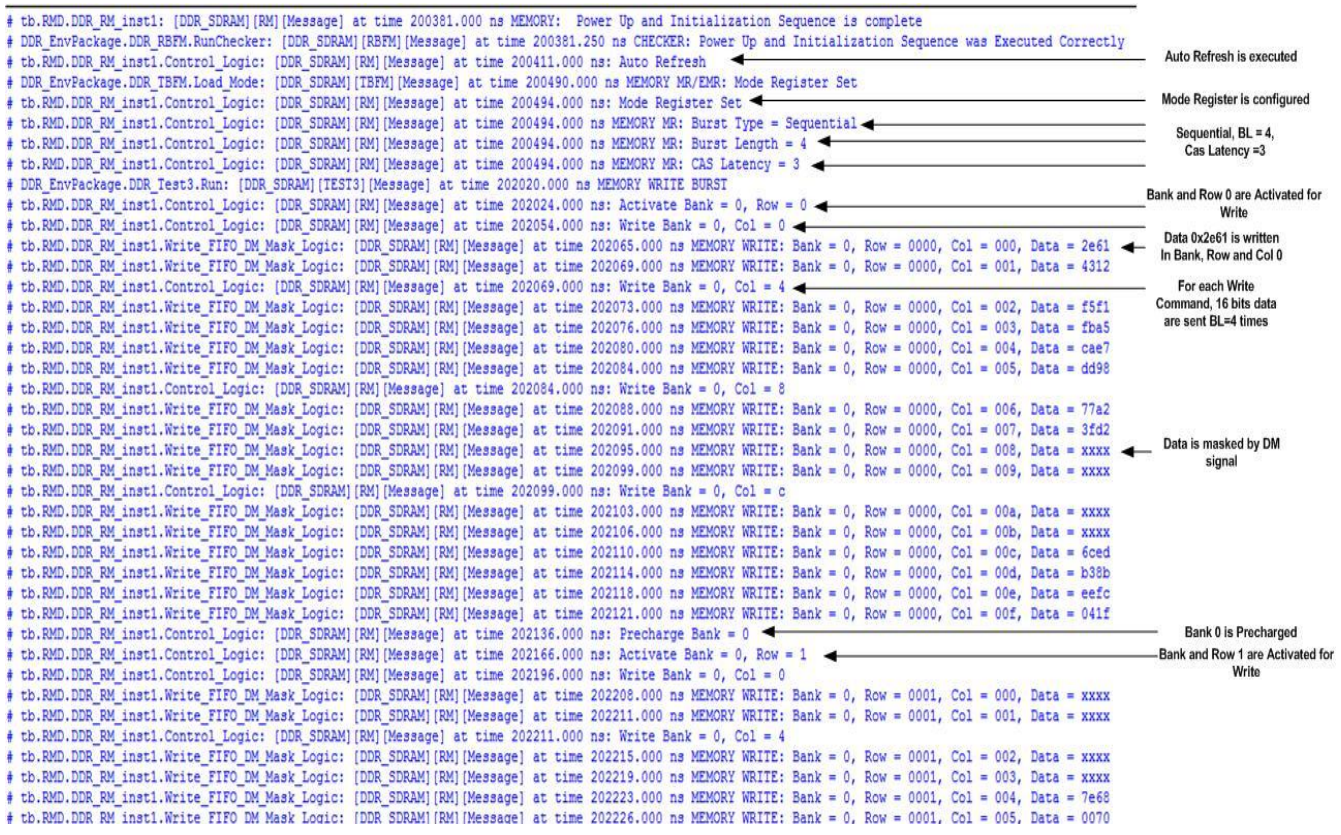


Figure 14. Test for Write Burst.

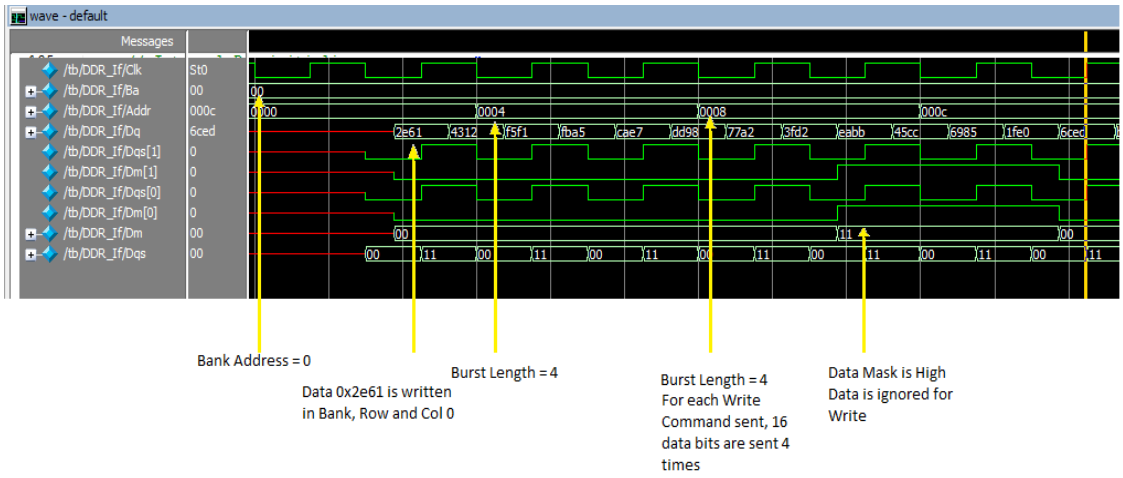


Figure 15. Write Burst Waveform

.

Figure 16 shows the data bank and row being activated, before a write burst to memory. When the write burst is done, the bank is precharged, so another memory location can be activated to write on it.

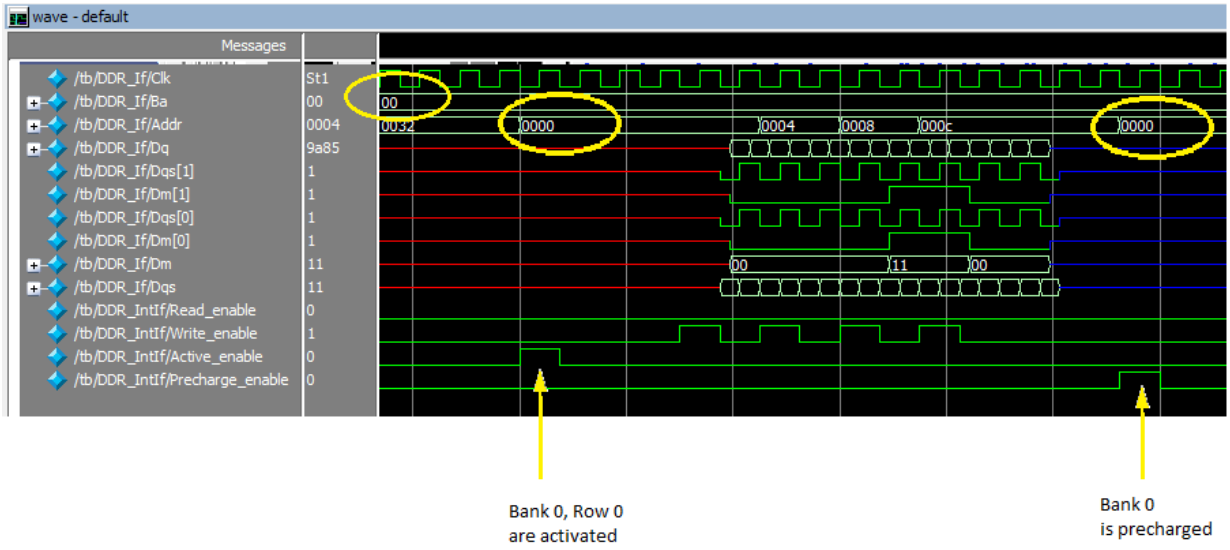


Figure 16. Bank Activate and Precharge.

Figure 17 shows the test sequence when read commands are sent to the memory. First, the activation of a bank and a row is done, because it is necessary for reading. Data is read with a burst length of 4, like is shown in Figure 18. The waveform of Figure 18 shows the data being

read from the memory with a CAS latency of 3 cycles, where Dqs signal is received correctly for a read burst. A monitor and a checker implemented for Dqs signal are the ones that monitor and check that Dqs signal was received correctly. Monitors and checkers for read data from memory are implemented, to compare the data read from the data written previously in the memory.

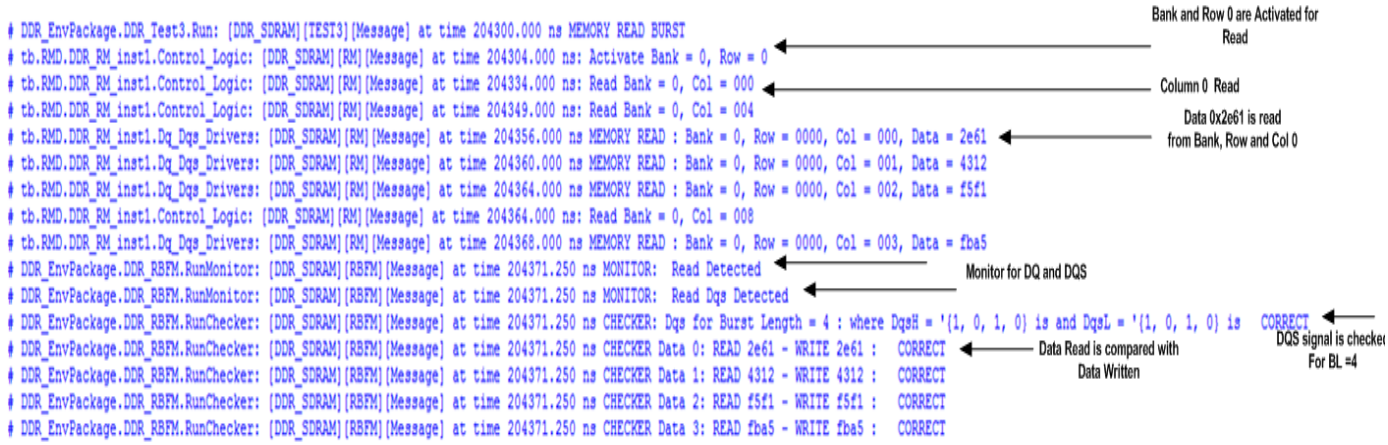


Figure 17. Test for Read Burst.

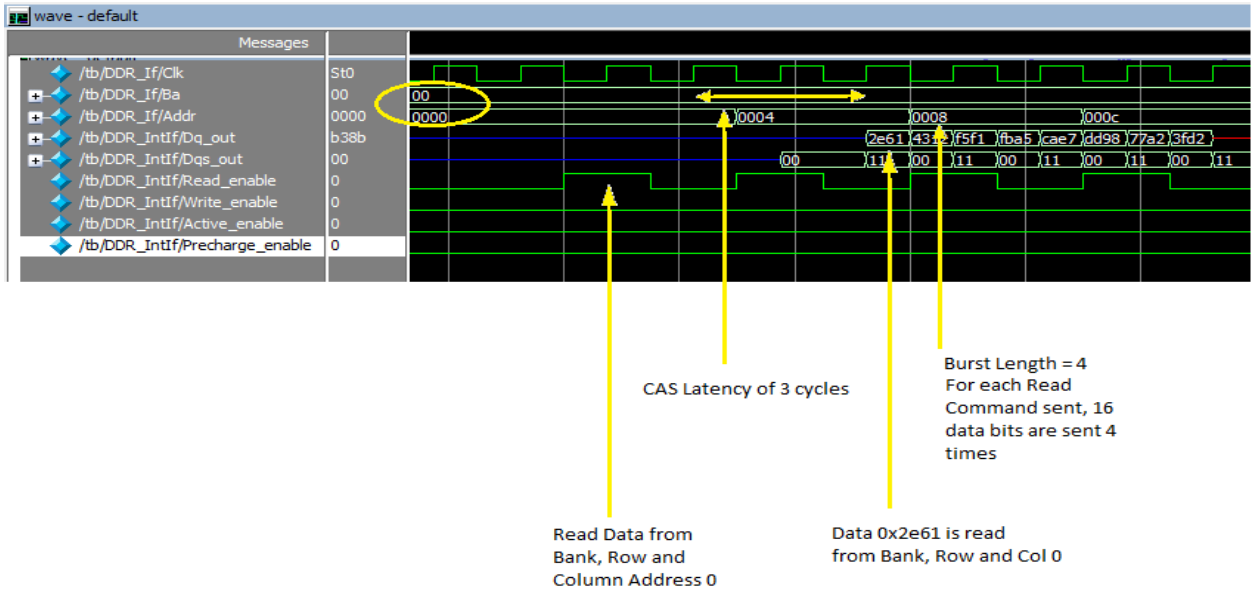


Figure 18. Read Burst Waveform.

- In the third test scenario, write and read commands are sent alternately with the proper timing delays between them. If these timing delays are violated, error messages will be displayed by the checkers implemented for the reference model. Read commands are executed to verify that the written data is the same as the read data. In this scenario, the power up and initialization of the memory is also executed, since it is necessary for the operation of the memory.

Figure 19 shows the test sequence when write and read command are sent to the memory alternately. First the Mode Register is configured with interleaved burst type, burst length of 8 and CAS latency of 2 clock cycles. Then, activation of a bank and a row is necessary for writing and reading. Data is written with a burst length of 8, like is shown in Figure 20. The waveform of Figure 20 shows the data being sent to the memory, where Dqs signal is sent correctly for a single write command, with the correct preamble, toggle and postamble bits. After writing the data to memory, data is read from the same memory location. Data is read with a burst length of 8, like is shown in Figure 21.

The waveform of Figure 21 shows the data being read from the memory with a CAS latency of 2 cycles, where Dqs signal is received correctly for a single read command, with the correct preamble, toggle and postamble bits. A monitor and a checker implemented for Dqs signal are the ones that monitor and check that Dqs signal was received correctly. Monitors and checkers for read data from memory are implemented, to compare the data read from the data written previously in the memory.

Another detail to observe about this is, is the interleaved burst type, where columns are accessed in a different way that the sequential mode.

```
# tb.RMD.DDR_RM_inst1.Control_Logic: [DDR_SDRAM][RM][Message] at time 200411.000 ns: Auto Refresh
# DDR_EnvPackage.DDR_TBFM.Load_Mode: [DDR_SDRAM][TBFM][Message] at time 200490.000 ns MEMORY MR/EMR: Mode Register Set
# tb.RMD.DDR_RM_inst1.Control_Logic: [DDR_SDRAM][RM][Message] at time 200494.000 ns: Mode Register Set
# tb.RMD.DDR_RM_inst1.Control_Logic: [DDR_SDRAM][RM][Message] at time 200494.000 ns MEMORY MR: Burst Type = Interleave
# tb.RMD.DDR_RM_inst1.Control_Logic: [DDR_SDRAM][RM][Message] at time 200494.000 ns MEMORY MR: Burst Length = 8
# tb.RMD.DDR_RM_inst1.Control_Logic: [DDR_SDRAM][RM][Message] at time 200494.000 ns MEMORY MR: CAS Latency = 2
# DDR_EnvPackage.DDR_Test4.Run: [DDR_SDRAM][TEST4][Message] at time 202020.000 ns MEMORY WRITE to READ : Uninterrupting
# tb.RMD.DDR_RM_inst1.Control_Logic: [DDR_SDRAM][RM][Message] at time 202024.000 ns: Activate Bank = 0, Row = 0
# tb.RMD.DDR_RM_inst1.Control_Logic: [DDR_SDRAM][RM][Message] at time 202054.000 ns: Write Bank = 0, Col = 1
# tb.RMD.DDR_RM_inst1.Write_FIFO_DM_Mask_Logic: [DDR_SDRAM][RM][Message] at time 202065.000 ns MEMORY WRITE: Bank = 0, Row = 0000, Col = 001, Data = 2e61
# tb.RMD.DDR_RM_inst1.Write_FIFO_DM_Mask_Logic: [DDR_SDRAM][RM][Message] at time 202069.000 ns MEMORY WRITE: Bank = 0, Row = 0000, Col = 000, Data = 4312
# tb.RMD.DDR_RM_inst1.Write_FIFO_DM_Mask_Logic: [DDR_SDRAM][RM][Message] at time 202073.000 ns MEMORY WRITE: Bank = 0, Row = 0000, Col = 003, Data = f5f1
# tb.RMD.DDR_RM_inst1.Write_FIFO_DM_Mask_Logic: [DDR_SDRAM][RM][Message] at time 202076.000 ns MEMORY WRITE: Bank = 0, Row = 0000, Col = 002, Data = fba5
# tb.RMD.DDR_RM_inst1.Write_FIFO_DM_Mask_Logic: [DDR_SDRAM][RM][Message] at time 202080.000 ns MEMORY WRITE: Bank = 0, Row = 0000, Col = 005, Data = 6363
# tb.RMD.DDR_RM_inst1.Write_FIFO_DM_Mask_Logic: [DDR_SDRAM][RM][Message] at time 202084.000 ns MEMORY WRITE: Bank = 0, Row = 0000, Col = 004, Data = 30d3
# tb.RMD.DDR_RM_inst1.Write_FIFO_DM_Mask_Logic: [DDR_SDRAM][RM][Message] at time 202088.000 ns MEMORY WRITE: Bank = 0, Row = 0000, Col = 007, Data = 4831
# tb.RMD.DDR_RM_inst1.Write_FIFO_DM_Mask_Logic: [DDR_SDRAM][RM][Message] at time 202091.000 ns MEMORY WRITE: Bank = 0, Row = 0000, Col = 006, Data = ad92
# tb.RMD.DDR_RM_inst1.Control_Logic: [DDR_SDRAM][RM][Message] at time 202099.000 ns: Read Bank = 0, Col = 001
# tb.RMD.DDR_RM_inst1.Dq_Dqs_Drivers: [DDR_SDRAM][RM][Message] at time 202114.000 ns MEMORY READ : Bank = 0, Row = 0000, Col = 001, Data = 2e61
# tb.RMD.DDR_RM_inst1.Dq_Dqs_Drivers: [DDR_SDRAM][RM][Message] at time 202118.000 ns MEMORY READ : Bank = 0, Row = 0000, Col = 000, Data = 4312
# tb.RMD.DDR_RM_inst1.Dq_Dqs_Drivers: [DDR_SDRAM][RM][Message] at time 202121.000 ns MEMORY READ : Bank = 0, Row = 0000, Col = 003, Data = f5f1
# tb.RMD.DDR_RM_inst1.Dq_Dqs_Drivers: [DDR_SDRAM][RM][Message] at time 202125.000 ns MEMORY READ : Bank = 0, Row = 0000, Col = 002, Data = fba5
# tb.RMD.DDR_RM_inst1.Dq_Dqs_Drivers: [DDR_SDRAM][RM][Message] at time 202129.000 ns MEMORY READ : Bank = 0, Row = 0000, Col = 005, Data = 6363
# tb.RMD.DDR_RM_inst1.Dq_Dqs_Drivers: [DDR_SDRAM][RM][Message] at time 202133.000 ns MEMORY READ : Bank = 0, Row = 0000, Col = 004, Data = 30d3
# tb.RMD.DDR_RM_inst1.Dq_Dqs_Drivers: [DDR_SDRAM][RM][Message] at time 202136.000 ns MEMORY READ : Bank = 0, Row = 0000, Col = 007, Data = 4831
# tb.RMD.DDR_RM_inst1.Dq_Dqs_Drivers: [DDR_SDRAM][RM][Message] at time 202140.000 ns MEMORY READ : Bank = 0, Row = 0000, Col = 006, Data = ad92
# DDR_EnvPackage.DDR_RBFM.RunMonitor: [DDR_SDRAM][RBFM][Message] at time 202143.750 ns MONITOR:  Read Detected
# DDR_EnvPackage.DDR_RBFM.RunMonitor: [DDR_SDRAM][RBFM][Message] at time 202143.750 ns MONITOR:  Read Dqs Detected
# DDR_EnvPackage.DDR_RBFM.RunChecker: [DDR_SDRAM][RBFM][Message] at time 202143.750 ns CHECKER: Dqs for Burst Length = 8 : where Dqs  = '{0, 0, 1, 0, 1, 0, 1, 0, 1, 0} is  CORRECT
# DDR_EnvPackage.DDR_RBFM.RunChecker: [DDR_SDRAM][RBFM][Message] at time 202143.750 ns CHECKER Data 0: READ 2e61 - WRITE 2e61 :  CORRECT
# DDR_EnvPackage.DDR_RBFM.RunChecker: [DDR_SDRAM][RBFM][Message] at time 202143.750 ns CHECKER Data 1: READ 4312 - WRITE 4312 :  CORRECT
# DDR_EnvPackage.DDR_RBFM.RunChecker: [DDR_SDRAM][RBFM][Message] at time 202143.750 ns CHECKER Data 2: READ f5f1 - WRITE f5f1 :  CORRECT
# DDR_EnvPackage.DDR_RBFM.RunChecker: [DDR_SDRAM][RBFM][Message] at time 202143.750 ns CHECKER Data 3: READ fba5 - WRITE fba5 :  CORRECT
# DDR_EnvPackage.DDR_RBFM.RunChecker: [DDR_SDRAM][RBFM][Message] at time 202143.750 ns CHECKER Data 4: READ 6363 - WRITE 6363 :  CORRECT
# DDR_EnvPackage.DDR_RBFM.RunChecker: [DDR_SDRAM][RBFM][Message] at time 202143.750 ns CHECKER Data 5: READ 30d3 - WRITE 30d3 :  CORRECT
# DDR_EnvPackage.DDR_RBFM.RunChecker: [DDR_SDRAM][RBFM][Message] at time 202143.750 ns CHECKER Data 6: READ 4831 - WRITE 4831 :  CORRECT
# DDR_EnvPackage.DDR_RBFM.RunChecker: [DDR_SDRAM][RBFM][Message] at time 202143.750 ns CHECKER Data 7: READ ad92 - WRITE ad92 :  CORRECT
# tb.RMD.DDR_RM_inst1.Control_Logic: [DDR_SDRAM][RM][Message] at time 202144.000 ns: Write Bank = 0, Col = 9
```

Auto Refresh is executed

Mode Register is configured

Interleave, BL = 8, Cas Latency =2

Bank and Row 0 are Activated for Write

Column 1 Write

Interleaved Burst Type

Column 1 Read

Monitor for DQ and DQS

Data Read is compared with Data Written

CORRECT DQS signal is checked For BL =4

Figure 19. Test for Write and Read Alternately.

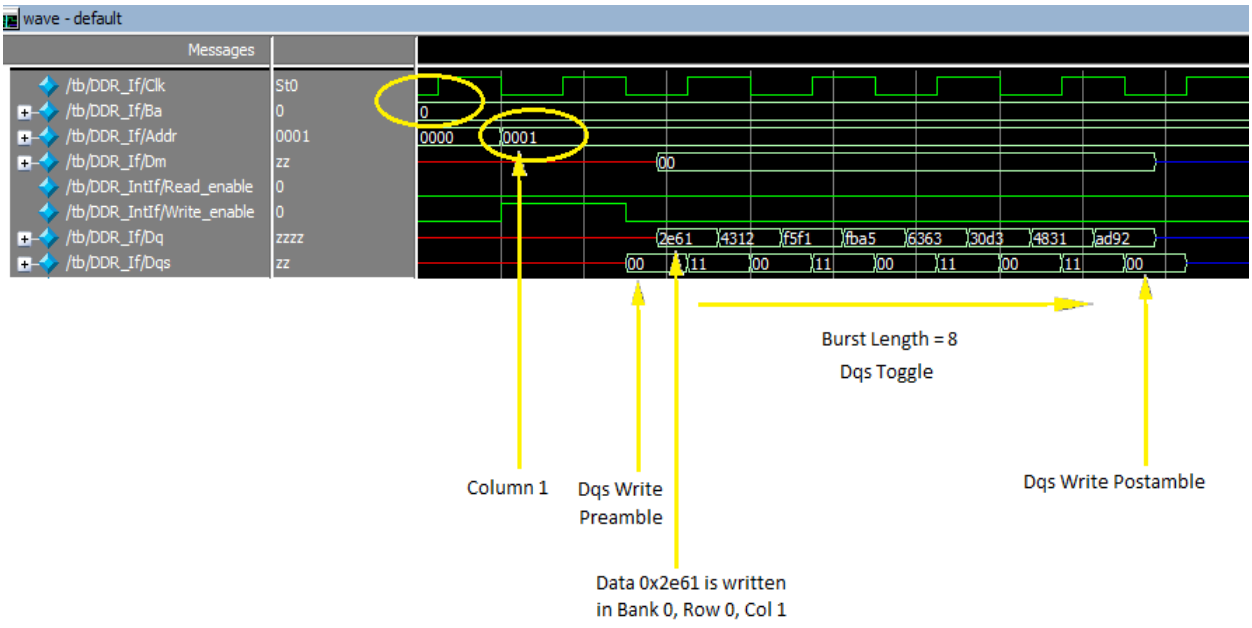


Figure 20. Write Waveform.

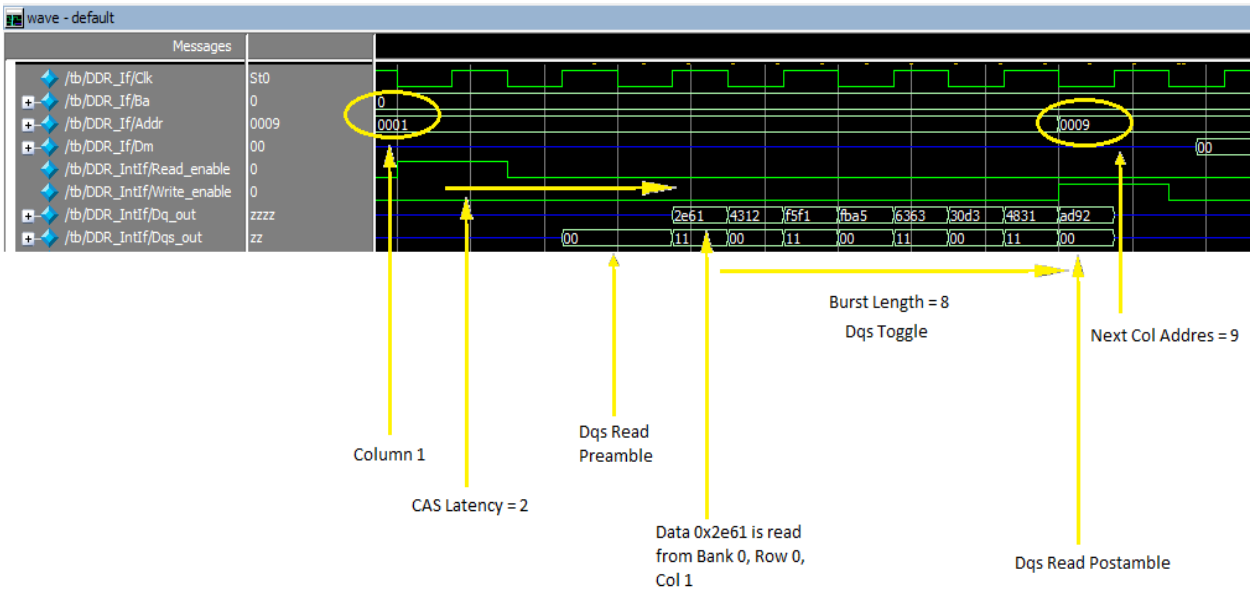


Figure 21. Read Waveform.

- In the fourth test scenario, write and read commands are sent to the memory with or without precharge, with random burst lengths, random burst types, random latencies, to random memory locations. Also, burst terminates are applied randomly to read commands. In this scenario, the power up and initialization of the memory is also executed, since it is necessary for the operation of the memory.

For this test, a burst terminate example is shown in Figure 22. A read command with burst length of 8 and CAS latency 2 is stopped by a burst terminate command, that terminates the read command, 2 cycles after it was issued.

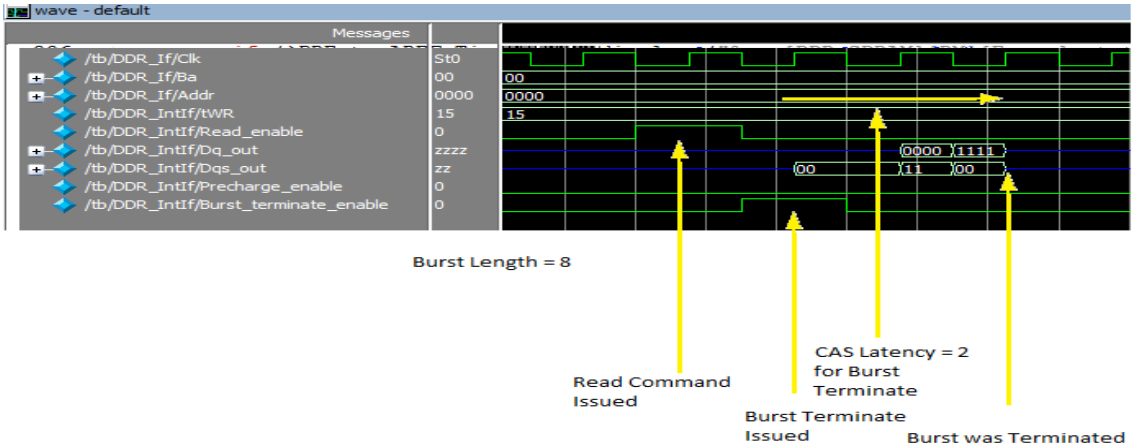


Figure 22. Burst Terminate Waveform.

Next, it is shown the coverage report for the all the coverpoints implemented for this project. Since this test scenario executes all the commands with random values, it is important to verify the completeness of the verification. With the coverage report generated by the simulation tool, feedback is received, so we know which functional areas were exercised and which not. The coverage report generated for 1000us of simulation, which got at least 50% of coverage in each coverpoint with a seed value of 3, is presented:

```
COVERGROUP COVERAGE:
----------------------------------------------------------------------------------
Covergroup                                                 Metric      Goal/ Status
                                                                     At Least
----------------------------------------------------------------------------------
 TYPE /DDR_EnvPackage/DDR_Coverage/cg_Command              91.6%        100 Uncovered
    Coverpoint cg_Command::cp_Command                      91.6%        100 Uncovered
        covered/total bins:                                   11         12
        bin Write                                             61          1 Covered
        bin Read                                              56          1 Covered
        bin Auto_Refresh                                       8          1 Covered
        bin Extended_Mode_Reg                                  1          1 Covered
        bin Mode_Reg                                          22          1 Covered
        bin Active                                            47          1 Covered
        bin Burst_terminate                                    2          1 Covered
        bin Precharge                                         33          1 Covered
        bin Nop                                            13102          1 Covered
        bin Deselect                                           0          1 ZERO
        bin Read_Precharge                                     5          1 Covered
        bin Write_Precharge                                   11          1 Covered
 TYPE /DDR_EnvPackage/DDR_Coverage/cg_PowerUp             100.0%        100 Covered
    Coverpoint cg_PowerUp::cp_Power_Up                     100.0%        100 Covered
        covered/total bins:                                    1          1
        bin Power_Up                                           1          1 Covered
 TYPE /DDR_EnvPackage/DDR_Coverage/cg_Write               100.0%        100 Covered
    Coverpoint cg_Write::cp_Write                          100.0%        100 Covered
        covered/total bins:                                    1          1
        bin Write                                             61          1 Covered
    Coverpoint cg_Write::cp_Write_Burst_Type               100.0%        100 Covered
        covered/total bins:                                    2          2
        bin Sequential                                        33          1 Covered
        bin Interleaved                                       28          1 Covered
    Coverpoint cg_Write::cp_Write_Burst_Length             100.0%        100 Covered
        covered/total bins:                                    3          3
        bin Burst_Length_2                                     7          1 Covered
        bin Burst_Length_4                                    26          1 Covered
        bin Burst_Length_8                                    28          1 Covered
        illegal_bin Burst_Length_Illegal                       0            ZERO
 TYPE /DDR_EnvPackage/DDR_Coverage/cg_Read                100.0%        100 Covered
    Coverpoint cg_Read::cp_Read                            100.0%        100 Covered
        covered/total bins:                                    1          1
        bin Read                                              56          1 Covered
    Coverpoint cg_Read::cp_Read_Burst_Type                 100.0%        100 Covered
        covered/total bins:                                    2          2
        bin Sequential                                        36          1 Covered
        bin Interleaved                                       20          1 Covered
    Coverpoint cg_Read::cp_Read_Burst_Length               100.0%        100 Covered
        covered/total bins:                                    3          3
        bin Burst_Length_2                                    12          1 Covered
        bin Burst_Length_4                                    20          1 Covered
        bin Burst_Length_8                                    24          1 Covered
        illegal_bin Burst_Length_Illegal                       0            ZERO
    Coverpoint cg_Read::cp_Read_CAS_Latency                100.0%        100 Covered
        covered/total bins:                                    3          3
        bin CAS_Latency_2                                     32          1 Covered
        bin CAS_Latency_25                                     5          1 Covered
```

```
       bin CAS_Latency_3                                    19          1 Covered
       illegal_bin CAS_Latency_Illegal                       0            ZERO
TYPE /DDR_EnvPackage/DDR_Coverage/cg_Active               100.0%        100 Covered
   Coverpoint cg_Active::cp_Active                        100.0%        100 Covered
       covered/total bins:                                    5          5
       bin Active                                            47          1 Covered
       bin Active_B0                                          9          1 Covered
       bin Active_B1                                         16          1 Covered
       bin Active_B2                                         12          1 Covered
       bin Active_B3                                         10          1 Covered
TYPE /DDR_EnvPackage/DDR_Coverage/cg_Active_Row            53.1%        100 Uncovered
   Coverpoint cg_Active_Row::cp_Active_Row                 53.1%        100 Uncovered
       covered/total bins:                                   34         64
       bin auto['b0000000000000:'b0000001111111]              1          1 Covered
       bin auto['b0000010000000:'b0000011111111]              1          1 Covered
       bin auto['b0000100000000:'b0000101111111]              0          1 ZERO
       bin auto['b0000110000000:'b0000111111111]              0          1 ZERO
       bin auto['b0001000000000:'b0001001111111]              1          1 Covered
       bin auto['b0001010000000:'b0001011111111]              1          1 Covered
       bin auto['b0001100000000:'b0001101111111]              0          1 ZERO
       bin auto['b0001110000000:'b0001111111111]              0          1 ZERO
       bin auto['b0010000000000:'b0010001111111]              0          1 ZERO
       bin auto['b0010010000000:'b0010011111111]              1          1 Covered
       bin auto['b0010100000000:'b0010101111111]              1          1 Covered
       bin auto['b0010110000000:'b0010111111111]              0          1 ZERO
       bin auto['b0011000000000:'b0011001111111]              1          1 Covered
       bin auto['b0011010000000:'b0011011111111]              3          1 Covered
       bin auto['b0011100000000:'b0011101111111]              0          1 ZERO
       bin auto['b0011110000000:'b0011111111111]              1          1 Covered
       bin auto['b0100000000000:'b0100001111111]              0          1 ZERO
       bin auto['b0100010000000:'b0100011111111]              0          1 ZERO
       bin auto['b0100100000000:'b0100101111111]              2          1 Covered
       bin auto['b0100110000000:'b0100111111111]              2          1 Covered
       bin auto['b0101000000000:'b0101001111111]              1          1 Covered
       bin auto['b0101010000000:'b0101011111111]              0          1 ZERO
       bin auto['b0101100000000:'b0101101111111]              0          1 ZERO
       bin auto['b0101110000000:'b0101111111111]              0          1 ZERO
       bin auto['b0110000000000:'b0110001111111]              2          1 Covered
       bin auto['b0110010000000:'b0110011111111]              0          1 ZERO
       bin auto['b0110100000000:'b0110101111111]              0          1 ZERO
       bin auto['b0110110000000:'b0110111111111]              0          1 ZERO
       bin auto['b0111000000000:'b0111001111111]              1          1 Covered
       bin auto['b0111010000000:'b0111011111111]              0          1 ZERO
       bin auto['b0111100000000:'b0111101111111]              0          1 ZERO
       bin auto['b0111110000000:'b0111111111111]              2          1 Covered
       bin auto['b1000000000000:'b1000001111111]              2          1 Covered
       bin auto['b1000010000000:'b1000011111111]              1          1 Covered
       bin auto['b1000100000000:'b1000101111111]              0          1 ZERO
       bin auto['b1000110000000:'b1000111111111]              0          1 ZERO
       bin auto['b1001000000000:'b1001001111111]              1          1 Covered
       bin auto['b1001010000000:'b1001011111111]              1          1 Covered
       bin auto['b1001100000000:'b1001101111111]              0          1 ZERO
       bin auto['b1001110000000:'b1001111111111]              0          1 ZERO
       bin auto['b1010000000000:'b1010001111111]              1          1 Covered
       bin auto['b1010010000000:'b1010011111111]              1          1 Covered
       bin auto['b1010100000000:'b1010101111111]              1          1 Covered
       bin auto['b1010110000000:'b1010111111111]              2          1 Covered
       bin auto['b1011000000000:'b1011001111111]              0          1 ZERO
       bin auto['b1011010000000:'b1011011111111]              2          1 Covered
       bin auto['b1011100000000:'b1011101111111]              1          1 Covered
       bin auto['b1011110000000:'b1011111111111]              1          1 Covered
       bin auto['b1100000000000:'b1100001111111]              0          1 ZERO
       bin auto['b1100010000000:'b1100011111111]              1          1 Covered
       bin auto['b1100100000000:'b1100101111111]              3          1 Covered
       bin auto['b1100110000000:'b1100111111111]              0          1 ZERO
       bin auto['b1101000000000:'b1101001111111]              0          1 ZERO
       bin auto['b1101010000000:'b1101011111111]              1          1 Covered
       bin auto['b1101100000000:'b1101101111111]              1          1 Covered
       bin auto['b1101110000000:'b1101111111111]              0          1 ZERO
       bin auto['b1110000000000:'b1110001111111]              0          1 ZERO
```

```
        bin auto['b1110010000000:'b1110011111111]           0          1 ZERO
        bin auto['b1110100000000:'b1110101111111]           2          1 Covered
        bin auto['b1110110000000:'b1110111111111]           1          1 Covered
        bin auto['b1111000000000:'b1111001111111]           0          1 ZERO
        bin auto['b1111010000000:'b1111011111111]           1          1 Covered
        bin auto['b1111100000000:'b1111101111111]           0          1 ZERO
        bin auto['b1111110000000:'b1111111111111]           2          1 Covered
 TYPE /DDR_EnvPackage/DDR_Coverage/cg_Burst_terminate
                                                         50.0%        100 Uncovered
    Coverpoint cg_Burst_terminate::cp_Burst_terminate
                                                         50.0%        100 Uncovered
        covered/total bins:                                 2          4
        bin Burst_terminate                                 2          1 Covered
        bin CAS_Latency_2                                   2          1 Covered
        bin CAS_Latency_25                                  0          1 ZERO
        bin CAS_Latency_3                                   0          1 ZERO
        illegal_bin CAS_Latency_Illegal                     0            ZERO
 TYPE /DDR_EnvPackage/DDR_Coverage/cg_Precharge        100.0%        100 Covered
    Coverpoint cg_Precharge::cp_Precharge              100.0%        100 Covered
        covered/total bins:                                 6          6
        bin Precharge                                      33          1 Covered
        bin Precharge_All                                   2          1 Covered
        bin Precharge_B0                                    4          1 Covered
        bin Precharge_B1                                   11          1 Covered
        bin Precharge_B2                                    8          1 Covered
        bin Precharge_B3                                    8          1 Covered
TOTAL COVERGROUP COVERAGE: 86.8%  COVERGROUP TYPES: 8
```

5.2. Reference Model Bugs

During the verification of the reference model, the following bugs/errors were discovered by the verification process:

- Clock stabilization was not presented for memory power up sequence.
- DLL Reset and Enable conditions were wrong.
- Only one Precharge to any bank command was required for memory initialization sequence. It is required to issue minimum two Precharge commands to all banks.
- Mode Register Set Command with low to A8 was not required for memory initialization.
- Write data to memory was possible with DLL disabled. It must be enabled to write data.
- Burst Interleave type was done incorrectly.
- DQS write checkers were bad implemented for write bursts.
- Burst Length and CAS latency decoding was incorrect.

All these bugs were found and fixed before the improvement carried out on the RM.

Conclusions

This project represents a great experience in the generation of a complete verification environment, including the reference model of a digital system. All the verification flow and process was implemented during the project.

The project was divided in two main tasks, the analysis and improvement of a reference model, which requires the understanding of DDR SDRAM protocol, and the design-implementation of a verification environment, which requires the understanding of verification concepts and the use of simulation tools. Verification concepts and the use of simulation tools were learnt during the Integrated Circuit Design Specialty and DDR SDRAM's concepts were studied during the realization of the project.

Paradigms and challenges were also affronted during the project, answering questions like what am I really verifying?, when I really finish the verification?, who verifies my verification environment?, which interpretation is the best? and others. These questions were answered by the experience obtained by me during the specialty and during the design and implementation of the project. Also, the experience of my advisors and other verification engineers were keys for resolving these paradigms and challenges.

It was helpful to use a verification language like SystemVerilog, because it provided object oriented programming, which made easier and more understandable the creation of a verification environment, following the verification methodology based in layers using classes and objects.

The project results were satisfactory, since I understood the DDR SDRAM protocol, analyzed a reference model and implemented a verification environment that is capable to find errors in an implementation that was provided by memories manufacturers. This means that verification is always necessary because human errors are always present.

Finally, this project gave me the opportunity to be involved in the industry of integrated circuit design, specifically in the pre-silicon validation area, when all the knowledge and practice acquired from this project, is applied for validating new technology implemented by companies at Guadalajara, Mexico.

Appendix

A. SYSTEM VERILOG CODES

- SystemVerilog codes for reference model and verification environment are available in the Project CD:

- Folder "Codes":
  - DDR_Coverage.sv — Coverage Class
  - DDR_Environment.sv — Environment Class
  - DDR_EnvPackage.sv — Package for Testbench
  - DDR_Interface.sv — Interfaces Definitions
  - DDR_Packet.sv — Packet Generation Class
  - DDR_ParametersPkg.sv — DDR SDRAM Parameters File
  - DDR_RBFM.sv — Rx Transactors Class
  - DDR_RM.sv — DDR SDRAM Reference Model
  - DDR_TBFM.sv — Tx Transactors Class
  - DDR_Test1.sv — Test for Initialization of DDR SDRAM
  - DDR_Test2.sv — Test for Directed Tests
  - DDR_Test3.sv — Test for Writing and Reading all memory locations
  - DDR_Test4.sv — Test for Writing and Reading alternately
  - DDR_Test5.sv — Test for Random Commands
  - tb.sv — Testbench module

References

[Accellera-01] Accellera Organization, "System Verilog 3.1a Language Reference Manual", Accellera's Extensions to Verilog, CA, 2004.

[Carlos-Flores-01] Carlos Flores, "Diseño de módulos controladores de Memoria", *Proyecto Final EDCI*, ITESO, 2008.

[Floyd-01] Thomas L. Floyd, "Digital Fundamentals", Chapter 10-Memory and Storage, Pearson Prentice Hall, 9th Edition, USA, 2006.

[Hynix-01] Hynix, "DDR SDRAM Device Operation", Rev 1.1

[Jacob-Wang-01] B. Jacob, D. Wang, "DRAM: Architectures, Interfaces, and Systems. A tutorial", Electrical and Computer Engineering Dept. University of Maryland at College Park. http://www.ece.umd.edu/~blj/DRAM/

[Janick-Bergeron-01] Janick Bergeron, "Writing Testbenches: Functional Verification of HDL Models", Kluwer Academic Publishers, 2nd Edition, Massachusetts, USA, 2003.

[Jedec-01] JEDEC "Double Data Rate (DDR) SDRAM standard", http://www.jedec.org/standards-documents/docs/jesd-79f

[Lattice-01] Lattice Semiconductor Corporation, "DDR SDRAM Controller" Reference Design RD1020, April 2004.

[Mark-Glasser-01] Mark Glasser, "Open Verification Methodology Cookbook", Chapter 1, Springer, OR, USA, 2009.

[Micron-01] Micron Technology, Inc., "General DDR SDRAM Functionality", Micron Technical Note TN-46-05, 2001.

[Micron-02] Micron Technology, Inc., Micron Sim Models 256Mb DDR SDRAM, Jul 2011. Available at: http://www.micron.com/parts/dram/ddr-sdram/mt46v32m8tg-6t-it?pc=%7BAA753075-6558-4704-8F8E-11D9047E7207%7D

[Samsung-01] Samsung Electronics , "DDR SDRAM Specification Version 0.3", 2000.

[Samsung-02] Samsung Electronics, "DDR SDRAM Application Notes 2; Basic DDR SDRAM operations".

[Winbound-01] Winbound, "2M X 4Banks x 16 Bits DDR SDRAM" Winbound W9412G6JH Technical Note, April 2010.

Index