CHAPTER 1

INTRODUCTION

1.1 Introduction to Memory

RAM (Random Access Memory) is a part of personal computer's main memory which can be accessed directly by CPU (central processing unit). Random access memory used for the purpose of read data from, and writes data into memory, Then CPU randomly accessing memory. Nature of RAM is volatile; it means that information stored is lost, when power goes off.

1.1.1 Type of RAM chips

- **SRAM (Static RAM)**
- **DRAM (Dynamic RAM)**

The SRAM memories are able to regain the stored information, when power is applied. Also SRAM memory needs constant power.

DRAM is used to stores binary information in the form of electric charges that applied to capacitors. Over a period of time, the stored information on the capacitors is discharged or it's going to lose. So, in order to retain their usage, capacitors must be recharged periodically.

1.1.2 Types of DRAM

There are 5 types of Dynamic random-access memory

- **ADRAM also refer as (Asynchronous DRAM):** In this type of memories, timing is controlled asynchronously. To control the timing operation, a memory controller circuit is used to generate the control signals.
- **SDRAM also refer as (Synchronous DRAM):** Here the access time is in synchronization with the CPU's clock directly. When CPU expects them to be ready, the memory chips remain ready for operation. SDRAM operates only on the pos-edge of the clock signal.
- **DDR SDRAM also refers as (Double Data Rate SDRAM):** This DDR SDRAM performs its data operation on both pos-edge and neg-edge of the clock cycles, so the data transfer rate is doubled.
- **RDRAM also refer as (Rambus DRAM):** This memory provides a very high data transfer rate. Here the signal timings are very fast.

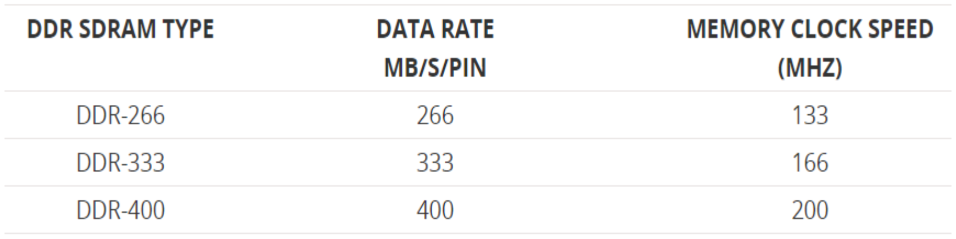
- **CDRAM also refer as (Cache DRAM):** Cache DRAM is a special type DRAM memory with utilizes on-chip cache memory (SRAM) that acts as a high-speed buffer for the main DRAM.

SDRAM is a type of DRAM memory; where speed of operation is faster than the previous conventional DRAM memories. After the DDR SDRAM introduced to the DDR markets, many further Next generation DDR are designed and implemented by different companies mainly Intel, Micron and JEDEC, are the developers, those are DDR1, DDR2, DDR3, DDR4 and DDR5.

1.1.3 Types of SDRAM

- **SDR SDRAM:** The basic version of Synchronous DRAM. Here data transfer happens only on pos-edge of the clock cycles.
- **DDR SDRAM:** Released in the year 2000, with internal clock speed rate is 133 to 200MHz and bus clock rate is 133 to 200 MHz, with data prefetch rate of 2n, data rate of 266 to 400MHz, transfer rate of 2.1 to 3.2 GB/s, voltage 2.5/2.6V and available sizes are 256mb, 512mb, 1GB. Also known as DDR1 SDRAM. That is transaction of data on pos-edge and neg-edge of the clock cycle.

Table 1.1 DDR SDRAM Data rates and clock speeds



DDR SDRAM TYPE	DATA RATE MB/S/PIN	MEMORY CLOCK SPEED (MHZ)
DDR-266	266	133
DDR-333	333	166
DDR-400	400	200

- **DDR2 SDRAM:** Released in the year 2003, with internal clock speed rate is 133 to 200MHz and bus clock rate is 266 to 400 MHz, with data prefetch rate of 4n, data rate of 533 to 800MHz, transfer rate of 4.2 to 6.4 GB/s, voltage 1.8V and available sizes are 512mb, 1GB.

- **DDR3 SDRAM:** Released in the year 2007, with internal clock speed rate is 133 to 200MHz and bus clock rate is 533 to 1000 MHz, with data prefetch rate of 8n, data rate of 800 to 2133

MHz, rate of transfer is min 8.5 to max 14.9 Gigabit/seconds, voltage 1.35/1.5 and available sizes are 512mb, 1GB, 4GB and 8GB. It has lower power consumption rate, when compared to before DDR.

- **DDR4 SDRAM:** Released in the year 2014, with internal clock speed rate is 133 to 200MHz and bus clock rate is 1066 to 1600 MHz, with data pre-fetch rate of 8n, data rate of 1600 to 3200 MHz, rate of transfer is min 17 to max 21.3 Gigabit/seconds, voltage 1.2 and available sizes are 2GB, 4GB, 8GB and 16GB. It provided lower operating voltage with faster data transfer rate.
- **DDR5 SDRAM:** DDR5 is currently under development. The DDR5 specifications are already announced in 2016 with expected first production in 2019. He main purpose is to reduce power consumption while doubling bandwidth and capacity.

1.2 Objective of the Project

The main objective of the work is to design a DDR SDRAM and test for higher coverage using SystemVerilog with random test cases. The DDR SDRAM memory size is of 512Mbits. If this memory model has to be tested using Verilog HDL testbench, if test bench may not produce 100% bug free functionality. At present SystemVerilog is used to test the complex designs, which not only find the bugs, and also clear it. In this context the proposed design is tested for all the test cases randomly to achieve 100% coverage using SystemVerilog. Coverage functionality testing can be accomplished by using cover bins, which checks all input pattern combinations and corner cases combinations to be covered randomly and send the input values to the memory banks and also receive the same values from the memory bank with read and write commands.

1.3 Motivation of the project

SDR (single data rate) SDRAM is the first-generation synchronous DRAMs. Which is slower than DDR1 (double data rate), because of only one word of data is transacted per clock cycles. DDR stands for the double data rate, which means the data reads or writes two words per clock cycles. With slight modification in the SDR interface timings and supply voltage, a major difference observed between SDR and DDR memory. For the purpose of storage data in personal computers, DRAMs are traditional used for storage data. Because of its speed of operation, highly efficient, and available at cheaper cost, while comparing with SDR SDRAM memories.

Currently industry requirement is more on SystemVerilog based Verification IP's. By using SystemVerilog language, it's easier to find the bugs and higher functional coverage.

1.4 Organization of the Project

The formal discourse has been organized into different chapters as listed below:

Chapter 1: In Introduction, Discussion about the history of RAM, and the different type of RAM, static and dynamic random-access memories. Basic about the SDRAM (synchronous dynamic random-access memory) and their types, SDR, and DDR.

Chapter 2: This includes the brief history of the literature survey on DDR SDRAM.

Chapter 3: This chapter deals with the DDR SDRAM design specification; those are general description, initialization, register definition, commands, state machines, operation of read and write.

Chapter 4: This chapter includes the brief explanation of software needed for the simulation operation of DDR SDRAM.

Chapter 5: This chapter contains the DDR SDRAM functional block diagram, controller architecture, FSM operation, and refreshes operation.

Chapter 6: This chapter includes the SystemVerilog verification environment, contains the SystemVerilog testbench.

Chapter 7: This chapter about the coverage's, like functional coverage, cover group, cover points, and cover bins.

Chapter 8: This chapter describe about the Results obtained from the simulator such as Modelsim and Questsim tool.

Chapter 9: This chapter includes the Conclusion and future scope of the proposed project work.

CHAPTER 2

LITERATURE SURVEY

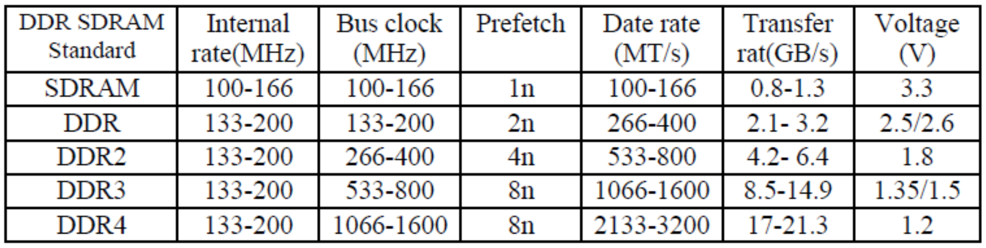
2.1 History of the existing systems

Jordi Sempere Agullo has deal with DDR-SDRAM controller. The FPGA implementation of high-speed interface is done for SDRAM which works at Double Data Rate. here a memory controller was designed which has high performance factor. High speed was necessary due to the reason that the controller was used for image processing where large amount of data was available and processed at faster rate. The core work is in memory interface data path architecture, to speed up the memory interface two clocks are used where one is memory nominal clock and the other is delayed clock. DQS, read data capture, write data timing and address and command timing has improved performance of data access at faster rate. The DQS read data capture architecture was implemented on FPGA.

Deepali Sharma et.al in their work explained about the important parameters to concentrate while designing a controller. The system designer should design a controller which is capable of providing proper commands and this command should take care of SDRAM initialization, read/write accesses and refresh of memory. To achieve this DDR SRAM controller was designed which consisted of a Finite State Machine for initialization of the SDRAM controller, signal generation by a signal module different commands like refresh, Active, Read, write, Refresh cycle for working. The data path module was also designed to know the data flow between SDRAM and system interface. DDR allow most optimal design techniques. DDR has remarkable performance gains.It is also understood that DDR SDRAM is a volatile memory device. All contents are lost when power goes off, due to this when memory is powered up, the device has to be initialized and configures for operating parameters.

Li Wang, Ju Wang worked on DDR SDRAM memory controller that will be based on Field programmable gate arrays with elements of huge limit and rapid, has a decent which requires a lot of information collection. Because of the distinction of the sign handling calculations, the time can't be effectively utilized amid perusing and composing as a part of conventional configuration of DDR SDRAM controller, diminishing the proficiency of information preparing. Another methodology of perusing and composing and afterward actualizes a DDR SDRAM controller. Due to the unusual and intensive control method of reasoning and timing essentials of DDR SDRAM, it requires outstandingly point by point work for right utilization.

Table 2.1: Comparison of DDR SDRAM Standards



DDR SDRAM Standard	Internal rate(MHz)	Bus clock (MHz)	Prefetch	Date rate (MT/s)	Transfer rat(GB/s)	Voltage (V)
SDRAM	100-166	100-166	1n	100-166	0.8-1.3	3.3
DDR	133-200	133-200	2n	266-400	2.1- 3.2	2.5/2.6
DDR2	133-200	266-400	4n	533-800	4.2- 6.4	1.8
DDR3	133-200	533-800	8n	1066-1600	8.5-14.9	1.35/1.5
DDR4	133-200	1066-1600	8n	2133-3200	17-21.3	1.2

Shruti Bhargava proposed system designer should design a controller which is capable of providing proper commands and this command should take care of SDRAM initialization, read/write accesses and refresh of memory. To achieve this DDR SRAM controller was designed which consisted of a Finite State Machine for initialization of the SDRAM controller, signal generation by signal module different commands like refresh, Active, Read, write and Refresh cycle for working. The data path module was also designed to know the data flow between SDRAM and system interface. DDR allow most optimal design techniques. DDR has remarkable performance gains. It is also understood that DDR SDRAM is a volatile memory device. All contents are lost when power goes off, due to this when memory is powered up, the device has to be initialized and configures for operating parameters. The implementation has been done in VHDL by using Modelsim 6.6d and Xilinx ISE 9.2i and then stated the conclusion that it works at 150.2 clock frequency.

Mayuri Sanwatsarkar and Jagdish Nagar proposed that system designer should design a controller which is capable of providing proper commands and this command should take care of SDRAM initialization, read/write accesses and refresh of memory. To achieve this DDR SRAM controller was designed which consisted of a Finite State Machine for initialization of the SDRAM controller, signal generation by signal module different commands like refresh, Active, Read, write and Refresh cycle for working. DDR SDRAM controller is designed and implemented in Verilog hardware description language and synthesis, simulation is carried out by using Modelsim and Xilinx ISE accordingly.

Santhoshima .K.G et.al. worked on an optimized DDR SDRAM controller implementation on FPGA. SDRAM memory was utilized as a part of numerous applications like advanced mobile phone, portable workstation in light of its decreased range, speed, configurable inertness and superior furthermore it gives two-fold information rate. The system designer should design a controller which is capable of providing proper commands and this command should take care of SDRAM initialization, read/write accesses and refresh of memory. To achieve this DDR SRAM controller was designed which consisted of a Finite State Machine for initialization of the SDRAM controller, signal generation by a signal module different commands like refresh, Active, Read, write, Refresh cycle for working. The data path module was also designed to know the data flow between SDRAM and system interface. DDR allow most optimal design techniques. DDR has remarkable performance gains. It is also understood that DDR SDRAM is a volatile memory device. The upgraded DDR SRAM controller has focused to more than 150MHz, with less power utilization furthermore plan has been centred around both perused and compose operation. Table 2.1 shows the Comparison of DDR SDRAM Standards.

Sushil Kumar Pachauri proposed controller is targeted at high bandwidth applications such as live video processing. It is designed to drive 256-bit DDR SDRAM memory. The system designer should design a controller which is capable of providing proper commands and this command should take care of SDRAM initialization, read/write accesses and refresh of memory. To achieve this DDR SRAM controller was designed which consisted of a Finite State Machine for initialization of the SDRAM controller, signal generation by a signal module different commands like refresh, Active, Read, write, Refresh cycle for working.

Avula Priyatham proposed DDR SDRAM (Double Data Rate Synchronous Dynamic Random-Access Memory) is upgrading the most alluring class of recollections are utilized as a part of PCs and some electronic gadgets because of its rapid, pipeline and burst get to include. Summon signals are utilized for memory no activity, read, write, precharge, auto precharge and load mode and so on tasks of SDRAM digitization has been given by memory controller. DDR basically utilizes complex circuit methods to satisfy fast. DDR SDRAM exchanges the information on both the pos-edge and neg-edge of the clock cycles. Here we can build up the useful planning check condition for the DDR SDRAM by utilizing System Verilog with declarations. This DDR SDRAM utilitarian conduct includes settled deferrals and stage connections amongst data sources and yields and builds up there are no bugs on tickers or postponed signals. System Verilog statements are evaluated on progressive events of an occasion or timing articulation. System Verilog Assertions and scopes are playing huge position in test seat improvement which helps accomplishing most extreme certainty on bug free plan. In this verification we are utilizing oops concepts like inheritance, randomization, coverage and assertions.

The outcome of the literature survey is that, Researchers working on designing next generation DDR and implemented DDR SDRAM memory controller using Verilog, system Verilog and UVM, but more researchers are attempted to find the functional coverage of DDR SDRAM memory controller, But Coverage can be achieved by using SystemVerilog verification environment with the help of cover bins.

CHAPTER 3

DDR SDRAM DESIGN SPECIFICATIONS

One of the newest commonly used synchronous SDRAM memories is the **Double Data Rate (DDR) SDRAM.** Addresses are supplied by a multiplexed address bus, controlled by RAS and CAS signals. Packetized reads from the memory, synchronized by processor clock signals take place in this memory, built as a multi-bank structure. The essential difference in the functioning of the DDR SDRAM memory comparing the SDRAM is a different operation control. The DDR SDRAM control method is illustrated in the figure below.

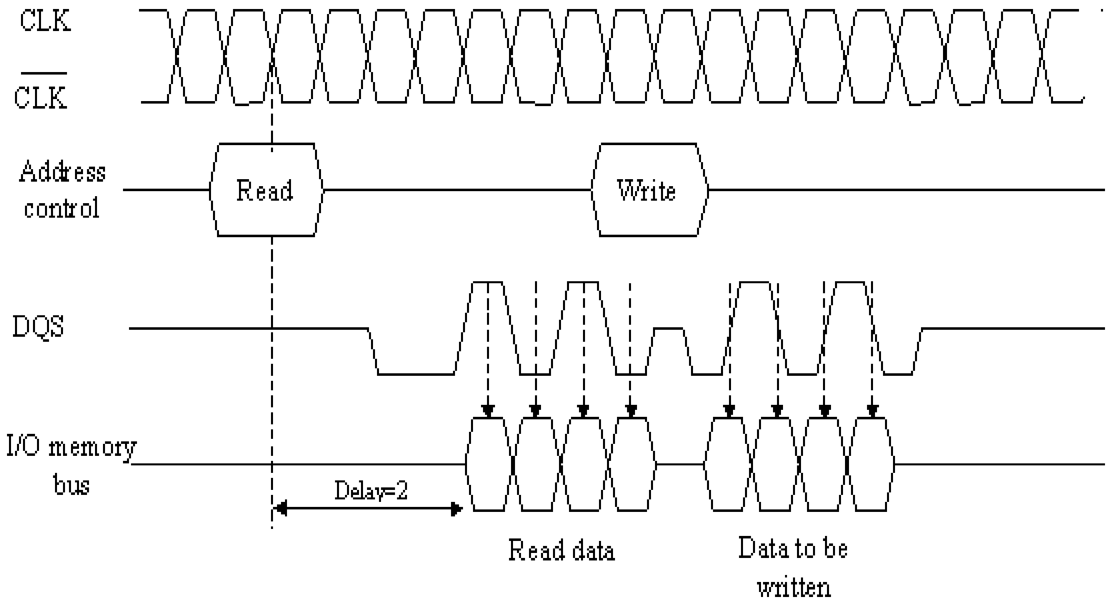


Figure 3.1: Simplified timing signals for the read and write in DDR memory

In the DDR SDRAM memory, two clock signals appear - CLK and complemented CLK, which are mutually shifted by half cycle. It is accompanied by the special DQS signal (from Data Query Strobe), which controls the type of operation: read or write. The two clock signals enable two times more frequent accesses to the memory, since access operations can be controlled by a coincidence of edges of both signals. In the packet mode (burst), this enables two times faster performance of memory. DQS is synchronous with clock signal but it contains three levels: zero, low and high. The zero level is set in the passive state of the memory. To perform a write, the computer memory controller (a chipset) generates the DQS signal - (4 high-low pulses), which denote data present on the input memory bus. The pulse centres denote data present on the bus. After readout, the DDR SDRAM memory control unit generates the DQS signal (4 pulses), which informs the processor that data are present on the memory output and on the memory bus. In this case, the edges of the signal (rising and falling down) determine the presence of data on the data bus. With the clock frequency 133 MHz, the transmission data rate for the DDR memory is equal to 2,1 GB/s.

3.1 FEATURES

1. DDR architecture, per each clock cycles, there are 2 data transaction.
2. With Data, Data strobe (DQS) is transmitted or received, and also, it's Bidirectional, and at the end of the receiver it's used for capturing data.
3. For data read operation DQS edge aligned with data, and for write operation, it's centre aligned with data.
4. It uses the clock inputs (those are differential) namely (CK and CK#).
5. Delay locked loops uses two signals namely DQ (data strobe) and DQS (data strobe signal) transitions with respect to (clock) CK transitions.
6. For each rising clock (CK) edges Commands are entered. And both data and data mask are referenced to edges of DQS.
7. It has performed operations in concurrent, that uses Four internal banks.
8. Data mask used for write data.
9. It's support burst length of 2, 4, or 8.
10. CAS Latency is two and 2.5.
11. AUTOPRECHARGE option per each burst access.
12. There are 2 refresh modes called Auto Refresh and Self-Refresh Modes.

3.2 General Description

DDR SDRAM is a quad bank DRAM (internally configured). The number of bits used in this device includes: 64 Mb,128 Mb.256 Mb,512 Mb ,1 Gb.

To realize the high speed of operations DDR uses the design called Double data rate. Its 2n prefetch design and includes interface designed to transact data on both pos-edge and neg-edge of the clock cycle input and output pins. Also, at the internal DRAM core, data transfer performs only one clock cycle. and 2 operations with respect to the corresponding N bit wide, and data transfer of one-half clock cycle at the Input and Output pins. A signal called bidirectional data strobe (DQS) that is transmitted externally with data and also used in data capturing at receiver end. For the read operation DQS strobe is transmitted by the DDR SDRAM and for the write operation DQS strobe can be transmitted by the memory controller. And also, for read operation its edge aligned with information, and for write DQS is centre aligned with information. And DDR uses the differential clocks for their operation (CK and CK#; the crossing of CK goes higher and CK goes lower are mentioned because of the pos-edge of clock (CK)). There are two commands signals namely address signals and control signals are operated at rising edge of the clock cycles.

DDR are burst oriented for Read and write accesses; those accesses start at a particular place or location and it continue for some extended programmed variety of locations in an exceedingly programmed sequence. When an Active command is registered, thereby begins the accessing of locations, which are followed by two command operations namely READ or WRITE command. When an particular address bits are registered with the ACTIVE command are correspondingly choose the particular bank and row to be accessed. Similarly address bits registered in coincident with the two operations READ or WRITES command for choosing the banks also beginning column location for burst access. The double data rat supports the programmable write and read burst lengths of two, four or eight locations.

For providing highly effective bandwidth, DDR designed in the form of pipelined and utilizing the multi bank architecture, that makes the operation write and read faster, and also for concurrent operations to perform, there by hiding the activation time and Row pre-charge. Where at the side of the power saving and power down modes an AUTO REFRESH mode is provided.

3.3 Pin Description

Table 3.1: Pin Description

SYMBOL	TYPE	DESCRIPTION
CK and CK#	Input	Clock: CK and CK# are differential clock inputs. All inputs except DQs and DMs are sampled in the pos-edge of the CK.
CKE	Input	When its high, activates the CK signal. And deactivates the CK signal, when it's low. Thereby initialization of either the power mode or the self-refresh mode operations.
Chip select (CS)	Input	When it low, CS bar enable the command decoder. And disable when it's high. When the command decoder is disabled at that that all new commands are ignores, but the previous operations still continues.
RAS, CAS, WE	Input	RAS, CAS and WE along with Chip select (CS) define the command being executed.
DM	Input	In write mode of operation, Dm has latency of 0. And operate as work mask by allowing input data to be written if it is low but blocks the write operation if it is high.
BA0, BA1	Input	Select which bank is to be active.
A0—A21	Input	Address inputs: when command activate command cycle, A0-A21 defines the row address when sampled at the pos-edge of the clock. During write or read command, A0-An define the column address (CA0-CAn) when sampled at the pos-edge of the clock cycles.
DQ	I/O	Data input or data output bus.
DQS	I/O	Active on both edges for data input and output. Centre aligned to input data. And edge aligned to the output data.
VDDQ and VSSQ	Supply	Isolated power supply and ground for the output buffers to provide improved noise immunity.
VDD, VSS	Supply	Power and ground supply for the input buffers and the core logic.
VREF	Input	SSTL reference voltage for the inputs.

3.4 Initialization

DDR SDRAM Powered by battery and initialized in predefined manner. There is some operational procedure are specified other than that may result in un-defined operations. During the power up or power down, there is no sequencing of power is specified. There are at least of meeting the two-condition minimum. And exception for the CKE (clock enable) signals, until the reference voltage (VREF) is applied. CKE (clock enable) signal is an SSTL_2 input; after VDD is applied it will detect a Low voltage CMOS LOW level. While during the power up to Maintain a low voltage CMOS in low level on clock edge in order to guarantee both outputs are in high state namely (DQ) data strobe and (DQS) data strobe signals.

In making the clock stable to provide the power and reference voltages area unit stable, DDR requires a delay of 200ns which is prior to applying an executable command. No operation command is applied and makes the signal clock edge CKE to high, once it reached the 200ns delay. After the No operation command applied, a PRECHARGE command is executed. For external Mode REGISTER SET command performed. In order to change the delay locked loops (DLL) that is delay locked loops and the same command issued to the mode register for the purpose of Reset of DLL delay locked loops. There it requires minimum 200ns clock cycle and between the commands. Then a PRECHARGE all command is issued, there is a state called "all banks idle "" state, where the devices are placed. Within the idle states there are two AUTO REFRESH operations are executed. Nd upon completion of that mode register set command issued to the mode register with the deactivation of the reset delay locked loops bit DLL signal.

3.5 Register Definition

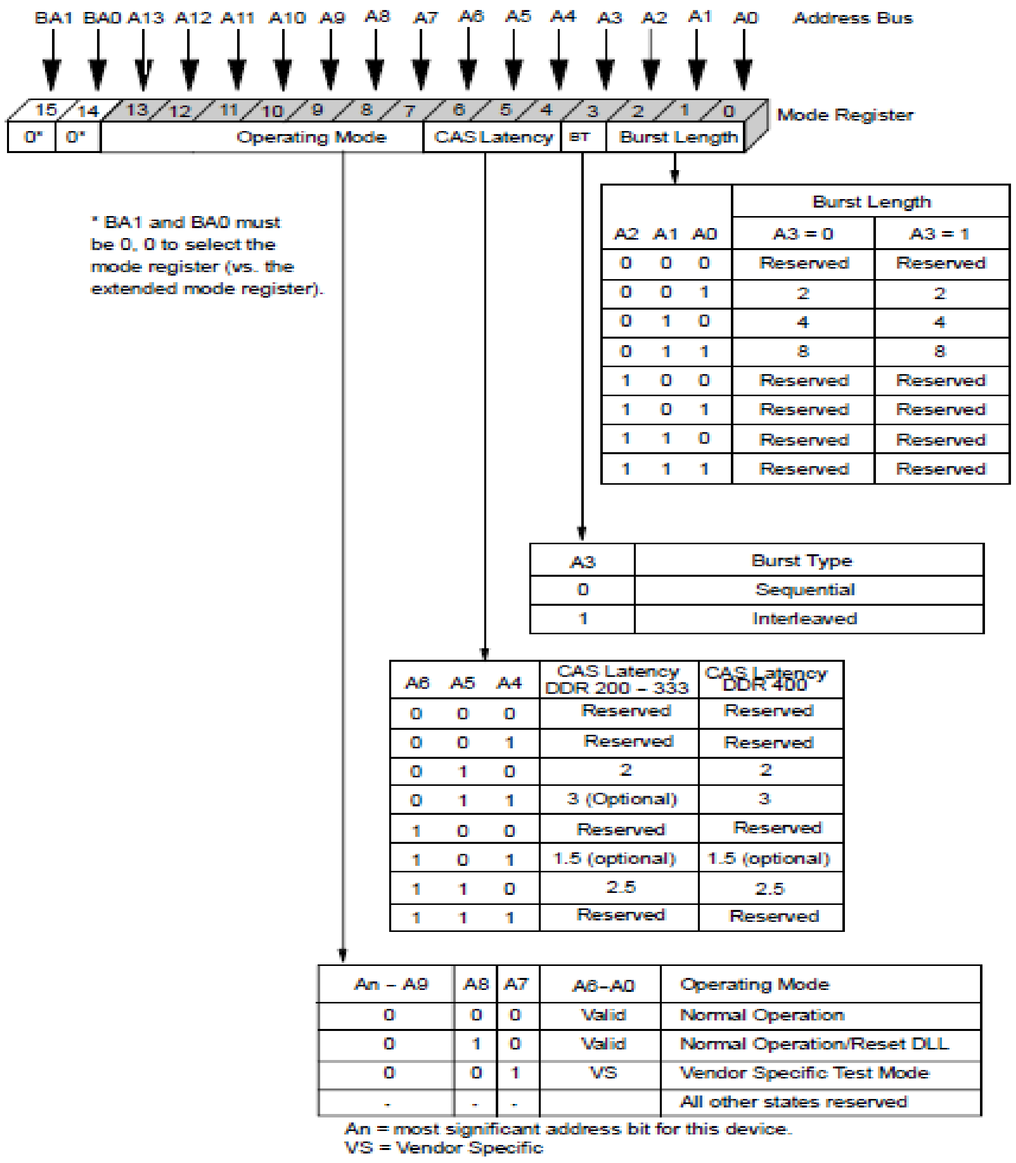
- **Mode Register**

In order to define the specific modes of the DDR SDRAM operations, it may contain the Burst length, burst type selection, CAS latency selection and mode of operation. Programmed the Mode Register through the MODE REGISTER SET commands that with (with BA0 = 0 and BA1 = 0) and it will maintain or retained stored information or data till it's been programmed again or whenever device loose power. Some exception cases are there for bit 8, Which is self-clearing bit. As from the mode register set figure, Burst length address from A0-A2, the type of burst from the bit A3, whether it's sequential or interleaved, and the CAS latency from the address locations A4-A6,and operating modes started from address location A7-A13 in that particularly address location bit A12 on 256Mb/512Mb and A13 bit on 1Gb see figure.

When banks are in idle states, No burst are in progress, at that time mode register has to load. Before initiating any operations, controller wait for specific interval of time, ignoring or breaking either any of these rules may result in unspecified operations.

- **Burst Length**

Accessing to the write or read operation of the DDR SDRAM is burst oriented, and that will be being programmable. For given WRITE or READ accessing commands, determine maximum number of column locations uses burst length. There are two types sequential and interleaved, uses available Burst lengths of two, 4, or 8 locations.


BA1 BA0 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0
Address Bus
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Mode Register
0* 0*
Operating Mode
CAS Latency
BT
Burst Length
* BA1 and BA0 must be 0, 0 to select the mode register (vs. the extended mode register).

A2	A1	A0	Burst Length A3 = 0	Burst Length A3 = 1
0	0	0	Reserved	Reserved
0	0	1	2	2
0	1	0	4	4
0	1	1	8	8
1	0	0	Reserved	Reserved
1	0	1	Reserved	Reserved
1	1	0	Reserved	Reserved
1	1	1	Reserved	Reserved

A3	Burst Type
0	Sequential
1	Interleaved

A6	A5	A4	CAS Latency DDR 200 – 333	CAS Latency DDR 400
0	0	0	Reserved	Reserved
0	0	1	Reserved	Reserved
0	1	0	2	2
0	1	1	3 (Optional)	3
1	0	0	Reserved	Reserved
1	0	1	1.5 (optional)	1.5 (optional)
1	1	0	2.5	2.5
1	1	1	Reserved	Reserved

An – A9	A8	A7	A6–A0	Operating Mode
0	0	0	Valid	Normal Operation
0	1	0	Valid	Normal Operation/Reset DLL
0	0	1	VS	Vendor Specific Test Mode
-	-	-		All other states reserved

An = most significant address bit for this device.
VS = Vendor Specific

Figure 3.2: Mode Register Definition

- **Burst Definition**

The states which are reserved are not able to use. May results in unknowing operation. When the commands WRITE or READ are applied, and then columns of block are equal to burst length is selected for the operations. Within this block only all burst operation access is taking place, meaning that if a boundary or end lune is reached, within the block burst will wrap. Selection of Block is very unique. By A1-Ai, when burst length make to two. By A2-Ai, when burst length make to four. By A3-Ai, when the burst length is adjusted or make to eight. For the given configuration Ai is the most significant column, within block choose the starting location, uses remained addressed bits that is least significant address bits. For both read and write, programmed burst length is applied.

Table 3.2: Burst Definition chart

Burst length	**Starting column address**		**Sequential**	**interleaved**
2	**A0**			
	0		**0-1**	**0-1**
	1		**1-0**	**1-0**
4	**A1**	**A0**		
	0	**0**	**0-1-2-3**	**0-1-2-3**
	0	**1**	**1-2-3-0**	**1-0-3-2**
	1	**0**	**2-3-0-1**	**2-3-0-1**
	1	**1**	**3-0-1-2**	**3-2-1-0**

- **Burst Type:**

Given Burst accessing can be programmed to wither the interleaved or sequential types. And it's also referred to the type of burst and it's selected through the A3 bit. Starting column address and burst type and Burst length all are used to determine the ordering of the accesses.

- **Read Latency**

Delay in the clock cycles are called as READ latency, it occurs in between registration of the first output data and registration of the READ command. The latency or delay set two and 2.5 times the clock and latencies of 1.5 or 3, are optional latencies. Here the optional latencies are 2 or 2.5, again it depends on some vendors to support one or both latencies. The output data will be available with the clock edges corresponds to the n+m, that is in case of READ command, the latencies is of m clocks and the commands assigned at n clock edges. For unknown operation, the states which are reserved can't be used, and may result in incompatibility versions.

- **Operating Mode**

By issuing the command to the mode register set, with bits A7 to A13 each set to 0.and bits correspond to A0-A6 are set to some desired values.

The initialization of the reset delay locked loops DLL, by applying the command to mode register set with A7 bits and the bits A9-A13 each are set to 0. Then the Bits A8 is set to 1, and the bits from A0-A6 are set to the desired values. The common reset delay locked loops DLL is getting by mode register set command, to select the normal operating modes (that's referred with A8 is equal to 0). Some of the bits are for test modes of operation and reserved for future use, they are other combination if A7-A13 bits. Those reserved for the future use and test modes can't be used, may result in incompatibility versions,

3.6 Terminology Definitions

DDR200: Here the frequency of clock is 100 Mega Hz, and transfer of data rate is 200Mega Hz.
DDR266: Here the frequency of clock is 133 Mega Hz, and transfer of data rate is 266MegaHz
DDR333: Here the frequency of clock is 167 Mega Hz, and transfer of data rate is 333Mega Hz
DDR400: Here the frequency of clock is 200 Mega Hz, and transfer of data rate is 400Mega Hz

- **Extended Mode Register**

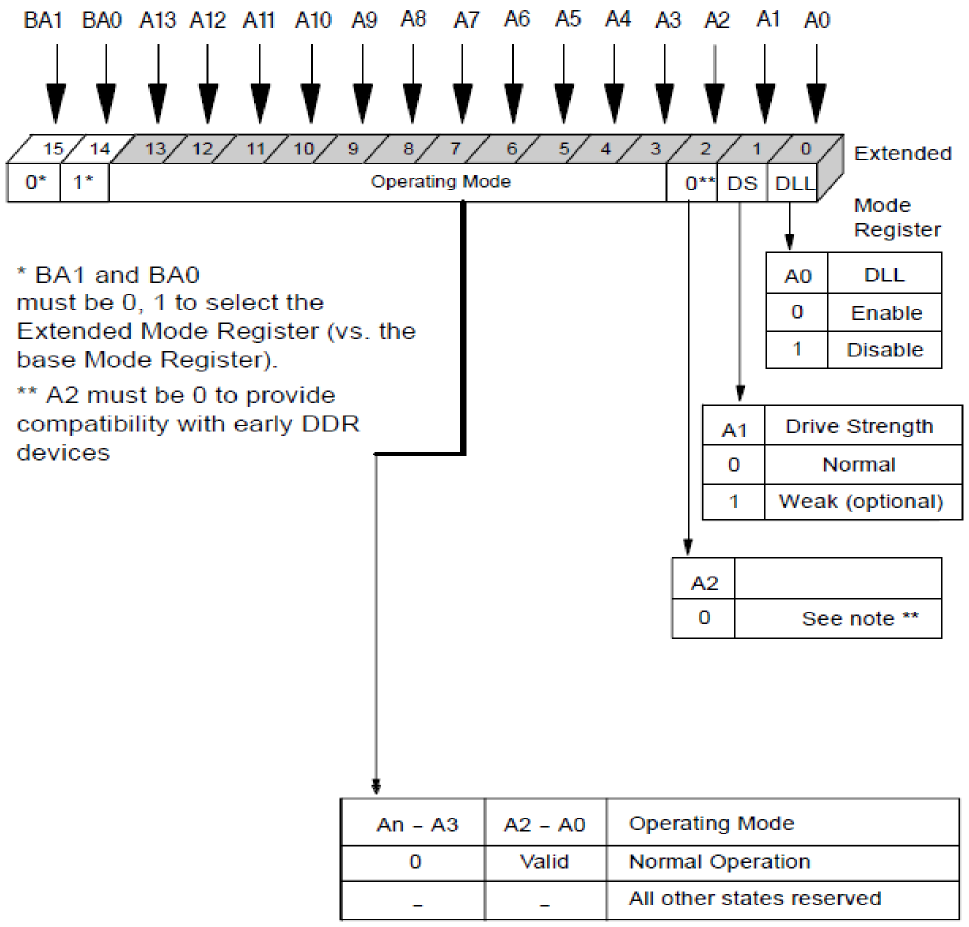
This mode register controls the functions beyond those controlled by the register mode. It has included some additional functions for their operation like enable or disable for DLL, there is an optional output drive strength selection, and bits are used to control these functions as shown in figure. And it's programmed via the command mode register set (with BA0=1 and BA1=0) that retain the information stored till its losses power or programmed again. While banks in IDLE state, at the time external mode register have to be loaded, and controller waits for interval of known time before starting of the next series operation, ignoring any of these requirements results in unknown.

- **DLL Enable/Disable**

For normal operations DLL delay locked loops is enabled. Throughout the format of power up, DLL alters, and when its return to normal operation after DLL is been disabled, for the two main purposes are debug and evaluation and for self-refresh modes at that time DLL is automatically enabled. Before any commands can be issued, 200 clock cycles must occur and at that time DLL reset also follows.

- **Output Drive Strength**

For all outputs the normal drive strength is mentioned as SSTL_2 class II. The normal drive strength for all outputs is specified to be SSTL_2, Class II. Some of the vendors support the weak driver strength option, that's specific to some point to point environments.


BA1
BA0
A13
A12
A11
A10
A9
A8
A7
A6
A5
A4
A3
A2
A1
A0
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
0*
1*
Operating Mode
0**
DS
DLL
Extended
Mode
Register
* BA1 and BA0 must be 0, 1 to select the Extended Mode Register (vs. the base Mode Register).
** A2 must be 0 to provide compatibility with early DDR devices

A0	DLL
0	Enable
1	Disable

A1	Drive Strength
0	Normal
1	Weak (optional)

A2	
0	See note **

An – A3	A2 – A0	Operating Mode
0	Valid	Normal Operation
–	–	All other states reserved

Figure 3.3: block diagram of Extended Mode Register Definition.

3.7 Commands

Table 3.3: Truth table of DDR SDRAM Commands

NAME (Function)	CS	RAS	CAS	WE	ADDR
DESELECT (NOP)	H	X	X	X	X
NO OPERATION (NOP)	L	H	H	H	X
ACTIVE (Select bank and activate row)	L	L	H	H	Bank/Row
READ (Select bank and column, and start READ burst)	L	H	L	H	Bank/Col
WRITE (Select bank and column, and start WRITE burst)	L	H	L	L	Bank/Col
BURST TERMINATE	L	H	H	L	X
PRECHARGE (Deactivate row in bank or banks)	L	L	H	L	Code
AUTO refresh or Self-Refresh (Enter self-refresh mode)	L	L	L	H	X
MODE REGISTER SET	L	L	L	L	Op--Code

Table3.4: Truth Table of DM Operation

NAME (Function)	DM	DQs
Write Enable	L	Valid
Write Inhibit	H	X

Table 3.5: Interface commands

COMMAND	VALUE	DESCRIPTION
NOP	000b	No operation
READ	001b	Sdram read with auto precharge
WRITE	010b	Sdram write with auto precharge
REFRESH	011b	Sdram auto refresh operation
PRECHARGE	100b	Sdram precharge all banks
LOAD_MODE	101b	Sdram load mode registers
LOAD REG1	110b	Load controlled configuration register
LOAD REG2	111b	Load controlled refresh period register

3.8 Simplified FSM State Diagram

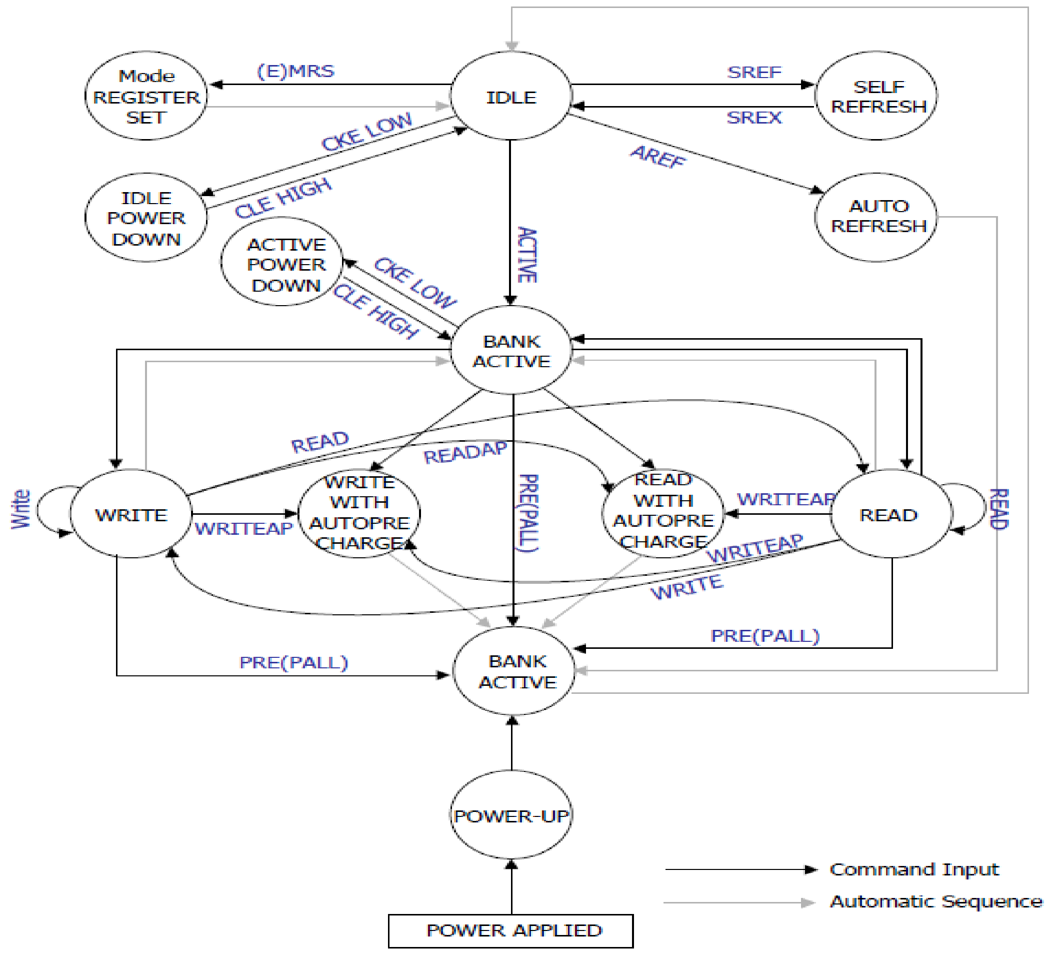


Figure 3.4: Simplified FSM state Diagram

- **Deselect**

When chip select signal goes high, the command signal DESELECT performed and it avoids new commands to be executed during this command executed. DDR deselects this command effectively. The operations are already progressing or running is not affected by this command.

- **No Operation (NOP)**

When chip select is Low, then the command NO OPERATION command is issued. This avoids the unwanted commands form register through the wait states or in idle states. The operations are already progressing or running is not affected by this command.

- **Mode Register Set**

For mode register set, registers are load via the A0-A13 bits. The description for the same can be mentioned with tables. Command applied, only when all banks in the sates of idle. Also at the same time No burst are operated. And any next command or subsequent commands in the list are not executes or not issued till the tMRD is meeting.

- **Active**

For the subsequent access, the command called ACTIVE command is issued to open or activate a particular bank. The values on the input bits BA0 and BA1 Select banks, and address inputs A0-A13 selects rows, Until the precharge command applied to particular specified bank, either read or write with command auto precharge, till then row remains active for accessing. Before open different row in the same bank, PRECHARGE command must be applied.

- **Read**

For the command read is issued to start a burst scan access make an active row. Input bits BA0 and BA1 have values that will select the bank and correspondingly address on input bits A0-Ai as shown in the table that selects the location of the initiating column. The input bit A10 is used to determine whether or not auto precharge is issued. Then there are two types of chosen, if in-case precharge is selected at the read burst end, the row being precharged. And if not selected then row will remain open for the next access.

- **Write**

To initiate or start the write burst access to an active row, we used command called WRITE command. For selection of the bank, the values BA0 and BA1 input bits are used. And same address from input values of A0-Ai, as shown in table that will select the selects location of the starting column. The input bit A10 is used to determine whether or not auto precharge is issued. Then there are two types of chosen, if incase precharge is selected at write burst end, then row being precharged. if not, then row will remain open for the next access. And data that come from the DQs signal is writing to the memory array. When at low DM signal, then data corresponding to it is written to the memory or if high DM then corresponding signal data is ignored, also write operation will not be executed with respect to the byte/column location.

- **Burst Terminate**

Command is used to truncate read bursts while doing that make sure that autoprecharge is disabled. The one with most rece READ command is prior to the command BURST TERMINATE will be truncated.

- **Precharge**

In order to deactivate or terminate the opened row in any particular bank or all the banks we used a command called PRECHARGE command. Once applying the precharge command, then banks will be free or available for the access of next rows at a specified time called (tRP). The value of bit A10 input is used to determine whether one bank needs to be precharged or all the banks or not. For some operations only one bank needs to be precharged, then it selects the bank corresponds to the BA0 and BA1 inputs. If not, then both BA0 and BA1 inputs become don't cares. When during bank precharging, precharge state must be in state called idle state, and activates to any of the write or read commands based on the priority issued to the bank.
If there is no open row in any bank then precharge command become NOP (no operation command) or even if any previously opened row already in precharging.

- **Auto Precharge**

In order to perform individual bank precharge function, used a command called AUTO PRECHARGE command and also it doesn't require and external command to execute the operation.this can be done by using the bit A10 make it to enable state, with in coordination of specific commands that is Read or Write commands. upon finishing of the some write burst and read burst operations, precahrge the bank or row can automatically do with addressing to the Read and write commands. And for every individual read and write operation, it may be enabled or disabled. Until the precharge time called tRP is going to completed, user can't give any other commands to the same banks.

- **Refresh Requirement**

DDR must require the refreshing of all the rows it has in any rolling interval of 64ms. There are two ways of generating the refresh operations, those are related to the external Auto refresh command and the other one is related to the internal self-refresh mode of operation. Average refresh interval called tREFI, its nothing but driving the number of device rows interval of 64ms. For an example, a 256Mb DDR SDRAM contains or includes rows of 8192, that will result in t7.8µs of tREFI. To avoid tthe excessive interrupts to controller memory, the DDR controller has to maintain the average refresh time of 7.8us and to operate some multiple internal bursts refresh.

- **Auto Refresh**

AUTOREFRESH is applied whenever or at each times of the refresh is required. Internal refresh controller generates the refresh addressing to perform. During Auto refresh, address bits becomes Don't care. At an average number of intervals period maximum tREFI, DDR requires the auto refresh command. In order to improve scheduling, task switching and the efficiency of the operations, some flexible interval refresh time is required. For any given DDR, 8 max auto refresh commands applied, and maximum gap of interval between next introduced auto-refresh command, and also any auto refresh command is given by 8* tREFI.

- **Self-Refresh**

In order to retain or restore data in DDR SDRAM memory, while rest of the system status of powered off. Without help of external clock clocking signal, DDR retains the data. It can be initialized with the clock edge signal CKE is made Disabled. When it enters into the self-refresh mode, DLL delay; locked loops are automatically disabled. And its automatically enabled, when upon existing self-refresh operation. During self-refresh operation, the input signal except the Clock edges are considered as don't cares. There is a procedure for existing self-refresh, that will require a command sequence.

Considered start with, edges clock must be maintained in constant before the Clock edge signal goes back to high state. Once it will enter into the HIGH, DDR issues a No operation command to the tXSNR. There is a formula or way to meet those requirements, that is requirements between the Refresh and the DLL is to issue the no operation command to 200 clock cycles, before issued to any other commands in the list. Once the completion of the signal from the self-refresh operation, an extra command called refresh commands are recommended.

3.9 Operation of Read and Write commands:

- **Bank/Row Activation:**

Within the DDR SDAM, any read or write commands can be applied to the bank before that we need to check that the row in that bank must be opened for the operation to perform. This be achieved via the command called ACTIVE command as shown in figure. The command active can select both row and bank to be activated. Once after applied active command then opening a row, the READ and WRITE can also is applied to same row. Once previous or older active row closed, and then next active command is applied the different rows in the same banks.

- **Reads**

READ command can be used to initiate the READ burst. As shown in the figure.
With the addition to that starting bank address and column address are specified. Auto precharge state may be high or low for specific burst access. Then consider first with enabling the AUTO PRECHARGE, at the burst completion, the row will start precharge. When during READ burst, starting from column address, valid dataout elements are available and are followed by the CAS latency. The next subsequent dataout element is valid at further neg or pos edge clocks. From the CAS Latency settings of (CL=2, CL=2.5 and CL=3) general timings are shown in figure.

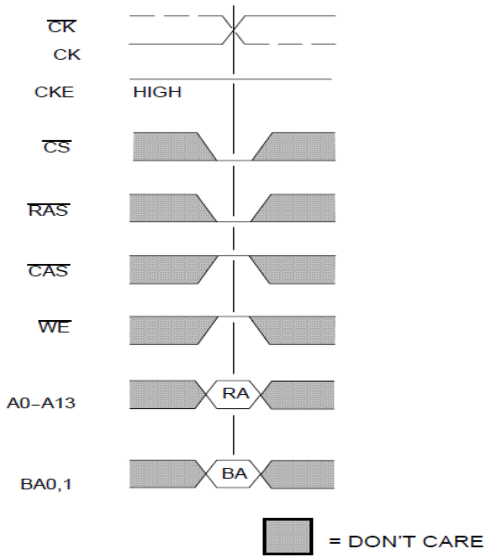


Figure 3.5: Activating a specific Row in a Specific Bank

DDR drives the DQS signal with output data. Read preamble is initial lower level or state on signal DQS. And low state is coincident with the data out element of the last element called read postamble. Once burst completes, let assume non commands other than this are initiated, once the DQS will go reach the High impedance state. Read burst may have a data that may be truncated or concatenated with data. New burst have first data element that follows either completed last burst element or longer bust of a last desired element is being truncated.

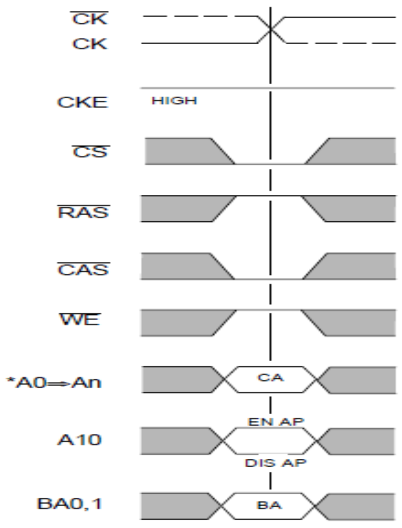


Figure 3.6: Read Command

- **Writes**

The command called WRITE is used to initiate the WRITE bursts as shown in the figure.

With write commands, the column address and bank address are selected. And either enabled or disable for that type of access. Then consider first with enabling the AUTO PRECHARGE, at the burst completion, the row will start pre-charge. When during burst WRITE, on first pos-edge of the DQS (data strobe signal) signal, first data-in element will be registered following with write command. And registered the next data elements for the next DQS signals, WRITE preamble is the one with low state on data strobe signal DQS between write command and first pos-edge. Write postamble is the same Low state on data strobe DQS signal that follows the last data in element. The time difference between write commands and first pos-edge of the DQS (tDQSS) is provided with a relatively wide ranges from 75% to the 125% of one cycle clock.

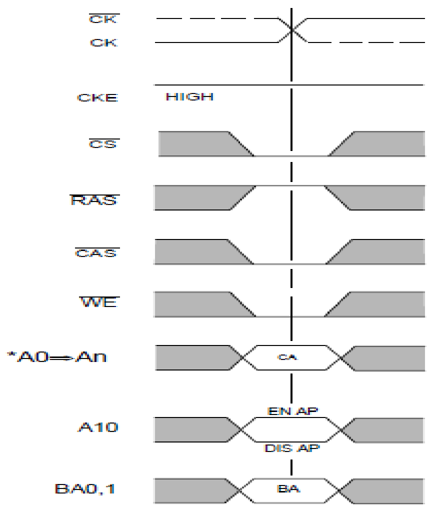


Figure 3.8: Write command

- **Precharge**

In order to deactivate open row in any particular bank, PRECHARGE command is used. Then those banks are accessible to the next in series row access with time (tRP). The bit corresponds to A10 determines the pre-charged that is whether one bank or all, to be pre-charged. When considering only one bank to be pre-charged, Input bit BA0 and input bit BA1 selects the bank for the operations. When considered the case, all banks be pre charged, then Inputs BA0 and BA1 becomes don't cares. Once bank in pre-charged state, nothing but idle state. Activated again only when commands like write or read command applied.

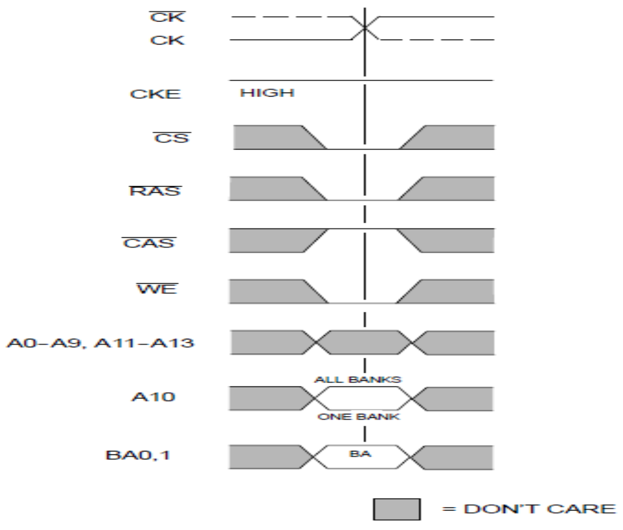


Figure 3.9: Precharge Command**.**

CHAPTER 4
SOFTWARE REQUIREMENTS

Application specific integrated chip (ASIC) Engineers are using different simulators to carried out their simulation activities. Today most of ASIC engineers are using Questasim, Modelsim and VCS. Those are more popular and best simulators, presently using by the well-known industries. Modelsim and Questasim simulators are from mentor graphics. some difference is in

- Language Support
- Simulation
- Design entry, Debug and Analysis
- Advanced Verification Methods
- Verification Management and Coverage

ModelSim is a function simulator from Mentor graphics for ASIC /FPGA designs. It supports both Verilog/SystemVerilog and VHDL languages, but have limited support for advanced System Verilog language (and specifically OVM/UVM/ etc. It can be used for Altera (now Intel) FPGA design simulations. Otherwise there is nothing like ModelSim-Altera.

Mentor graphics also has another simulator called **QuestaSim** which is more advanced and is latest compared to ModelSim and is more common than ModelSim in industries for ASIC and SOC designs.

VCS is similarly a functional simulator from Synopsys that also supports Verilog/SystemVerilog and VHDL based ASIC/FPGA/SOC design simulations. It is also as advanced and similar to QuestaSim.

NC-verilog is a subset of a similar functional simulation tool from Cadence called Incisive simulator. It is used for Verilog design simulations. There is also NC-VHDl part of same Incisive simulator from Cadence for VHDL design simulations.

4.1 ModelSim Advance Simulation and Debugging

Design the DDR SDRAM memory controller using Verilog HDL, and simulation results are taken from Modelsim 6.6d tool.

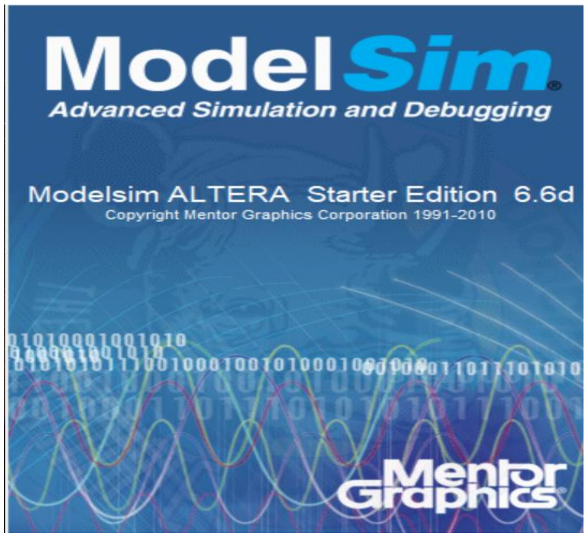

ModelSim
Advanced Simulation and Debugging
Modelsim ALTERA Starter Edition 6.6d
Copyright Mentor Graphics Corporation 1991-2010
Mentor Graphics

Figure 4.1: Modelsim ALTERA starter Edition 6.6d

4.2 QuestaSim Advanced Verification

DDR SDRAM memory controller verification using SystemVerilog HVL, and simulation results are taken from QuestaSim 10.0b.

Figure 4.2: QuestSim simulator

Some of the key differences of Questa over ModelSim:

- Compile flow optimizations
- Simulation Performance optimizations (2-50X)
- Post processing analysis (i.e. run a simulation in batch while viewing the results of a previous simulation)
- Multiple Wave Windows
- 64-bit mode support (ModelSim is only released to run in 32-bit mode)
- Links to analog/mixed-signal simulation
- Job control and Integration with Simulation farms
- Access to Advanced SystemVerilog Testbench features (assertions, constraints, and functional coverage)

CHAPTER 5
IMPLEMENTATION OF DDR SDRAM

5.1 DDR SDRAM functional Block Diagram

Double Data Rate (DDR) synchronous Dynamic random-access memory (SDRAM) used to transfer data at a rate of information twice per clock cycles that data transact on both pos-edge and neg-edge of the clock cycles. And DDR SDRAM supports user to alter the data widths, and burst exchange rates, and other important one is CAS dormancy of the design. Since the DDR SDRAM (synchronous dynamic random-access memory) Controller deals with initiating/pre-charging the banks, user just needs to issue straight forward read /write commands.

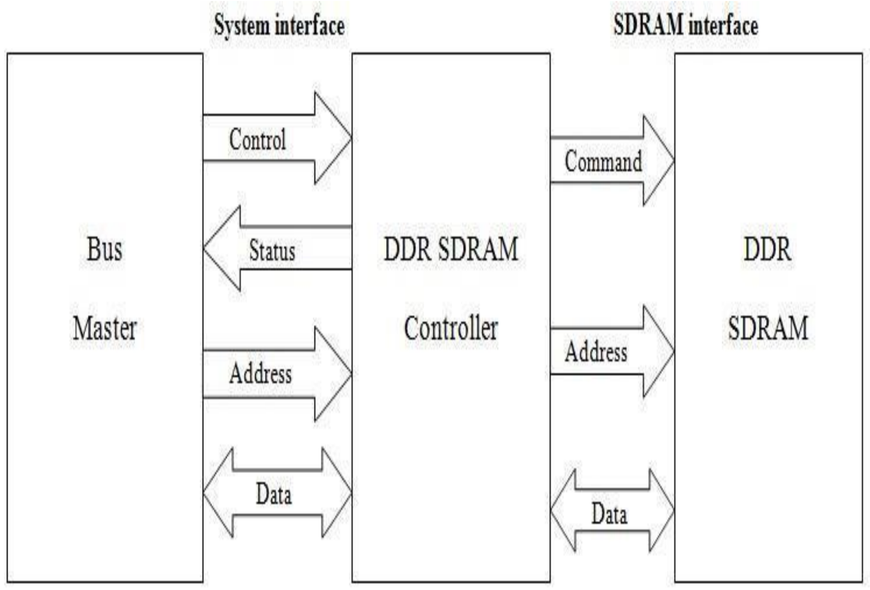


Figure 5.1: DDR SDRAM memory device is connected with Bus master or device through DDR SDRAM memory controller

As from figure 5.1, DDR SDRAM memory is connected to devices (Bus master in the figure) is via the DDR SDRAM memory controller. When developing dynamic rand access memories-based architectures, it’s pretty common, that designers uses the synchronous protocol. Therefore, DDR SDRAM becomes synchronous DDR SDRAM. This makes the user to happily uses the some more features like speed, pipeline features and their Burst access. Some application has some type of requirement, Hence DDR SDRAM controller can build according to the particular applications. There are very few tests to perform by the controller, mainly initialize the memory and to provide the commands. The main concern is with design of the DDR SDRAM memory controller as per technology requirements. Here Double rate of information transfer provides the higher throughput. The best way of spot the difference is, when a design in working under 160MHz of clock frequency in an normal

SDRAM based designs. For DDR based designs then the frequency can be improved to 320MHz that to same time when the data rate is doubled.

5.1.1 Write Operation in DDR SDRAM

The flow chart shown explains the operation of write by DDR SDRAM .When the write command is initialized, first the data is read and stored in the temporary register when a clock occurs, the data from the input is sent to the memory in 2 half clock cycles which gets stored in the memory, the data in not directly written to the memory but it is firs stored in the register and then transferred to the memory.

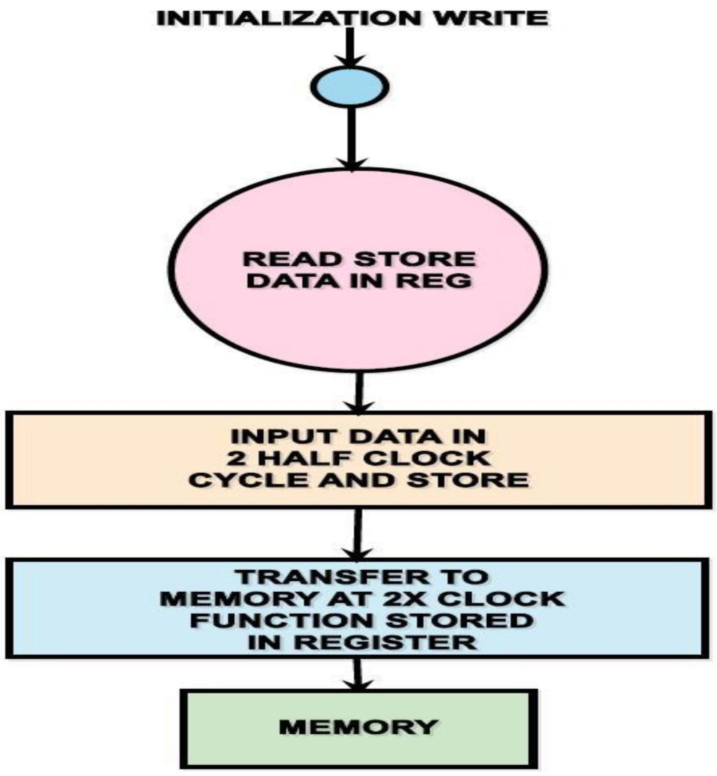

INITIALIZATION WRITE
READ STORE DATA IN REG
INPUT DATA IN 2 HALF CLOCK CYCLE AND STORE
TRANSFER TO MEMORY AT 2X CLOCK FUNCTION STORED IN REGISTER
MEMORY

Figure 5.3: Flow chart for write operation in DDR SDRAM

5.1.2 Read Operation in DDR SDRAM

The below flow chart can read the important operations to perform for the reading of the data in DDR SDRAM. When an operation is set to be read by the DDR, then data read from memory at a speed of clock is twice as much of clock cycles. And data read on both the pos-edge and neg-edge of the clock cycles. That means data read from the memory is at higher speed compared to SDR based designs.

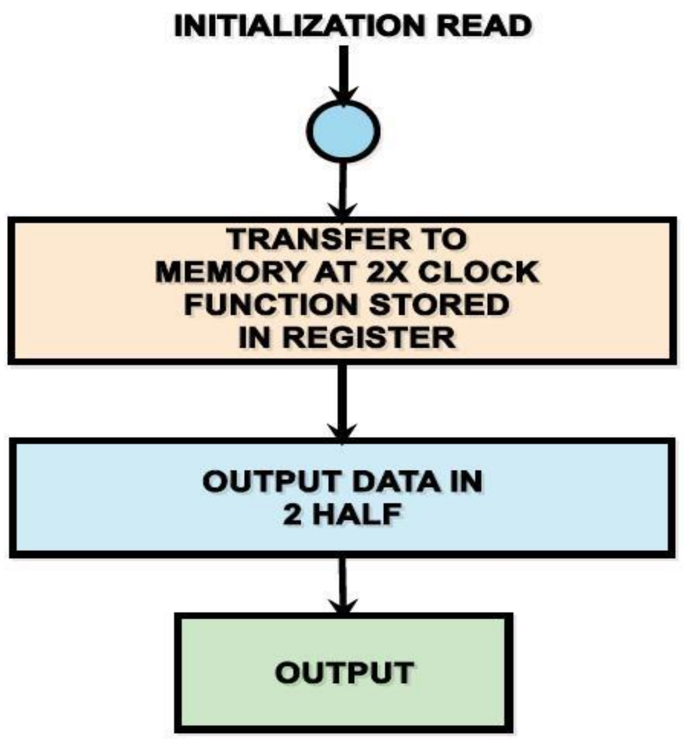


Figure 5.2: Flow chart for read operation in DDR SDRAM

5.2 DDR SDRAM Controller Architecture

Controller architecture explains the design with the help of different blocks to meet the requirements, such that there are three modules namely with the DDR SDRAM memory controllers are Control Interface, Data Path and Command Modules.

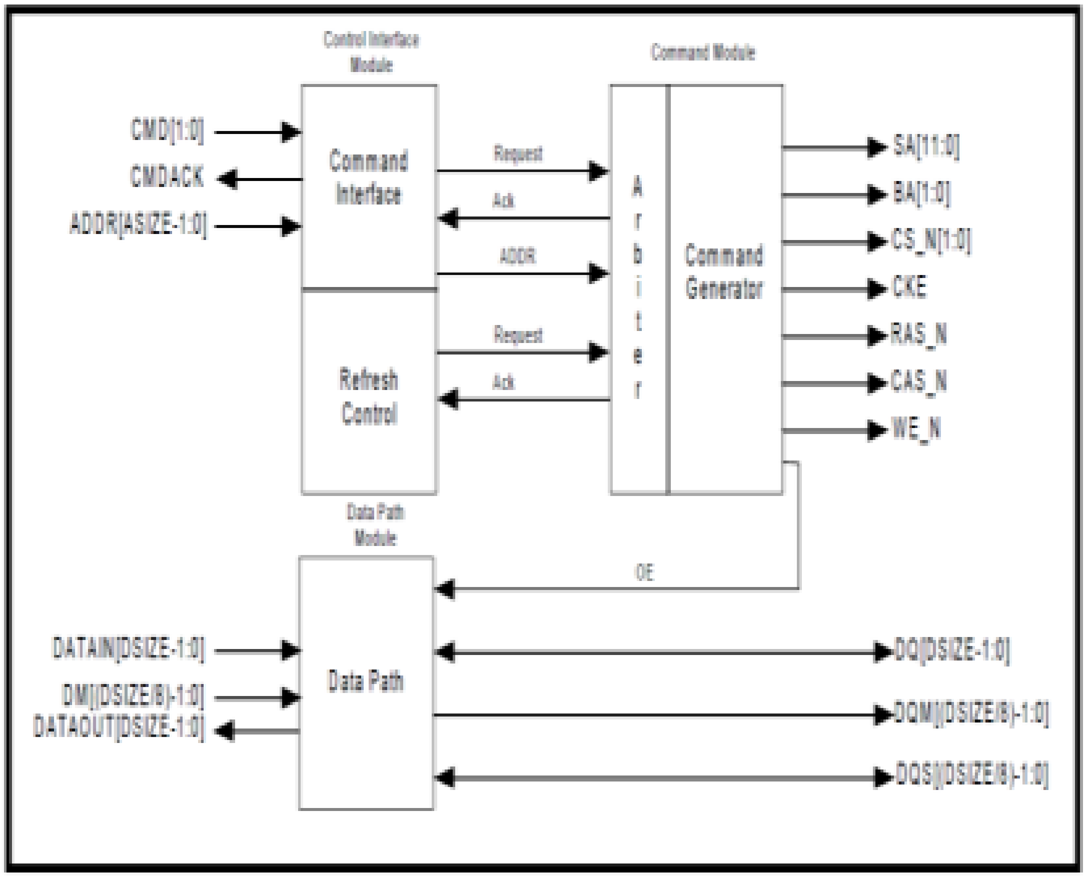


Figure 5.4: Proposed diagram of DDR SDRAM memory controller

5.2.1 DDR SDRAM Control Interface Module:

Control Interface is the primary module and contains finite sate machine, a programmable 16-bit counter, which is used to perform the auto refresh operation. The controller interface module accepts the commands from the processor and it will decode them and send the decoded WRITEA, NOP, READA, PRECHARGE, REFRESH, and LOAD_MODE signal commands are passed to the module command. There are two registers used, load register 1 and load register 2. The DDR SDRAM uses a different means separate clock signals: one is CLK and other is CLK# (the crossed of CLK going high and CLK# going low is just taken as the positive edge of the clock). READ command and WRITE are the basic commands are used to access the SDRAM memory. The DDR SDRAM will support the burst length of 2, 4, or 8 locations for programmable READ/WRITE operations. When access to SDRAM started at any location and it will continue until it reaches the burst length. The READ and WRITE signal operation started by sending the ACTIVATE commands.

The controller Interface consists of a finite state machine shown in figure 2. The initial state is considered as ideal state and the next state of the controller can be in either PRECHARGE, REFRESH, LOAD_MODE, or ACTIVATE state those all states depending on the request from the processor.

5.2.1.1 Finite State Machine operation

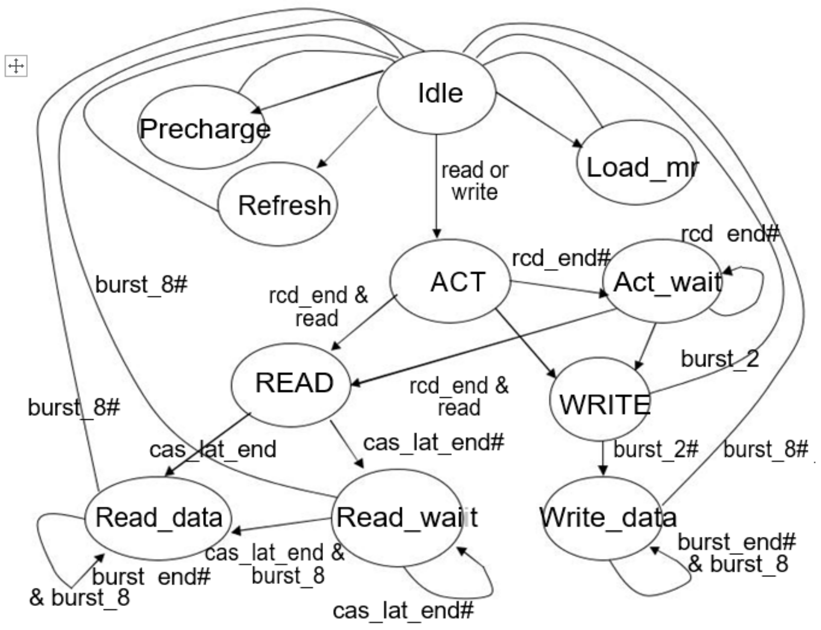


Figure 5.5: Finite state machine diagram

Those lines without indications in the figure 2 indicate an automatic sequence. The READ and WRITE command operation are followed by command ACT. The ACT command signal is used for particular bank to open a row, and where PRECHARGE signal command function is to close the bank which is open and if the open bank is other than that we want to access. When a WRITE command is passed and executed, the controller will first go to the ACT command state and then tit will enter to the WRITE state. The WRITE command operation will be executed till the burst length is completed or its Up to the burst terminate is inserted. When the READ operation is executed, the initial address is registered and during the READ operation the data will comes or arrival time is 1-3 clock cycles later on the data bus and this delay is because of the time that is actually needed to read the internal DRAM memory. This delay time is CAS latency.

Once on completion of the burst READ and WRITE operations the controller will go settle back to the initial IDLE state.

5.2.2 DDR SDRAM Data Path Module

The function of the module data path is to store the data, when write operation performed, and calculate the value for read data. Its main operation is to data generation and sampling of data to DDR. When the READ operation is executed, the data is sampled from the module data path. Data will synchronize the clock signal and transacted as one-word per clock cycle. When the WRITE operation is executed, the user interface gets the data at one word per clock cycle. The data will be resynchronized by the module data path and it will transfer at a rate called DDR double data rate.

Between memory and processor, data path module is interfaced. Between memory and processor, two operations are performed those are latching of data and dis-patching of data. Module data path has tristate buffer which is controlled by OE signal, that comes from the module command. Whatever the data to be written is received on the DATAIN pin during the WRITE operation and during the READA operation data is provided on the DATAOUT pin.

5.2.3 DDR SDRAM Command Module:

The command module mainly consists of an arbiter, multiplexer and three shift registers. An arbiter is used while in case if multiple requests come from different-different devices. If both high and low priority device need to transfer data at a time, then device with high priority gets first priority. The low priority device not able to transfer data until the high priority device has transferred all its data. In my work high priority is REFRESH control operation. So, if the REFRESH operation is executing on and thereby command from the host comes, then the controller will first continue the refresh operation to be done and it hold command requests by not assert this command to the command acknowledge (CMDACK). Once refresh operation is finished the command module will start performing the requested commands from the host. Module inside command module is the shift registers and is used to maintain the timing between the command signals assigned to the SDRAM memory controller. Other module it contains is multiplexer which is used to multiplexing the address. Where address of rows are multiplexed into Synchronous DRAM during the signal ACTIVATE (RAS) command. Other column portion is also multiplex the address to SDRAM address outputs happen during a signal READ (WRITE) command.

- **Basic Commands of DDR SDRAM Memory Controller**:

DDR SDRAM memory controller is to provide connection between memory & device connected to it, so there are number of commands can be accessed to, mainly pre-charge, read, write, refresh and so on, as shown in table.

Table 5.1: DDR SDRAM Control Interface Commands**.**

Command	Sign	CS_N	WE_N	RAS_N	CAS_N
No-operation	NOP	L	H	H	H
Load mode register	LMR	L	L	L	L
PRE-CHARGE	PRE	L	L	H	L
Read	READ	L	H	L	H
Write	WRITE	L	H	L	L
Active	ACT	L	L	H	H
Auto-refresh	REF	L	L	L	H

5.4 Refresh in DDRSDRAM

In order to store or hold the information a periodic refresh operation is needed in DERAM because of capacitor usage in circuitry needs refresh regularly on time basis. The command called auto refresh is initialized, at the time all banks will be out of states. Signal called sdr_REF_ACK will identify the acknowledgment of the signal called sdr_REF_REQ. and through the full refresh cycle, this signal will be dynamic. Untill the signal called sdr_REF_ACK goes dynamic, we have to kept the signal sdr_REF_REQ. under this condition, there is no chance for the read/write cycle to access.

With the help of state machine diagram, when state machine enters into the NOP (no operation) if there is no commands are given to the design. And then it will initialize for the next instruction and waiting for the next instruction provided. Once it gets commands signal, then it will enter into the lode mode reg. it may be load register 1 or may be load register 2. Depending on the data whether it’s from the device or memory. Also depends on write operation and read operation, it transfers into respective states and generates some of the appropriate signal to operate the given particular tasks. Once it will finish the task assigned to it, then it will generate the acknowledgement signal. Then the device is either in two states, one is refreshed or other is pre-charge state and then return back to load registers state, here afterwards waits for next operations to be assigned as an input.

CHAPTER 6

VERIFICATION ENVIORNMENT ARCITECTURE

Verification is the most significant side of the VLSI configuration stream. Around 70% of the time is distracted in the verification procedure. In this way, it is the most time distracting process. Due to the enhance in number of transistors in the incorporated circuit (IC), lessening highlight estimate and enhanced outline apparatuses. This made of occurrence of bugs in the design. So the requirement for the confirmation of the IC ended up fundamental. System Verilog is a standard confirmation dialect used to check the RTL (Register Transfer Level) outline. It comprises of base class library coded in System Verilog. The check architect can make distinctive confirmation parcels by broadening these classes. Also, system Verilog gives numerous other helpful check highlights, for example, utilization of macros for actualizing complex capacity, industrial facility for protest creation.

For a designer to read the hardware specification for the blocks. They are converted to human language prospective. And they will create a logic for the block, that's in format of computer readable or machine language. Most common way of writing the register transfer level codes using Verilog or VHDL languages. In order to do all this things user or designers have to know the three main common things those are transformation functions and complete understanding format of both the input and outputs.

There are multiple of reasons "why SystemVerilog is chosen over Verilog for verification" but more often it refers to reusability features, minimum debug effort ease of coding and better productivity results.

Some of the advantages of SystemVerilog over Verilog in terms of verification are:

Verilog doesn't provide race free testbenches. That's why the tweak is, if you write the DUT in positive edge of clock then you have to drive the TB at the negative edge of clock. This will remove race condition but not completely, more often Verilog TB suffers time zero race condition. While SystemVerilog has provide race free testbenches with the help of program block which schedules the occurrence of TB at the reactive region. But if someone judiciously use clocking block that will also be sufficient to provide the elimination from race condition. The task defined in Verilog are static task with which you are not able to provide the same value of the arguments defined each time, that means value gets overridden each time. But SystemVerilog tasks and function are by default automatic and provide the flexibility to reuse the same value of argument through tasks across module and classes.

The randomization of stimulus in Verilog is very limited but SystemVerilog provides huge scope of randomization using **CDV** (Coverage Driven Verification) and **CRV (**Constrained Random Verification) and provides useful tool like functional coverage to hit more uncovered areas and to determine whether testing has been completed or not. SystemVerilog is provided with the powerful features of Assertion which increases the observability of a block to check whether any bug is there by providing intermediate check points which is completely absent in Verilog. Moreover, in SystemVerilog assertions there are constructs like assert and cover which are used to assert and cover a property which reduces debug time and verification spin cycle. SystemVerilog assertions has assume property which is useful in formal verification and finding bugs which are absent with Verilog. SystemVerilog has three different fork join construct like (fork and join_all, fork and join_none, fork and join_any) which provides flexibile coding and helps to use a task inside a function while Verilog has to dealt with only fork join and has limitation of function calling only function but not task.

Since SystemVerilog is class/dynamic based testbench it supports powerful OOP features like inheritance, polymorphism and encapsulation which provides reusability and minimizes coding effort while Verilog which is a static/module based testbench language doesn't support any of them. SystemVerilog is compatible enough with many vendor specific tools and is widely used while Verilog as a testbench language is still not providing the flexibility.

Verilog is a HDL (Hardware Description Language) while SystemVerilog (SV) is both HDL and HVL (Hardware Verification Language), so combined termed as HDVL. Verilog has mainly 2 data types Reg and Wire which are 4 valued logic 0,1,x,z while SV is enriched with wide variety of data type like int, shortint, longint, logic, bit, real, realtime, reg, chandle, user defined data type, etc. which are both combination of 4 and 2 valued logic. Memory and arrays declaration on Verilog are static in nature while in case of SV it's dynamic in nature means declaration can be changed during compile time. Verilog has a single always block for implementation of combinational and sequential logic while SV uses always_ff, always_comb, always_latch construct for use of different logic. FSM implementation in SV is much easier in SV with the use of enumeration data type which has a number of methods like number, first, last, next, previous which helps in debugging purpose unlike using parameter in Verilog which is hardcoded. SystemVerilog uses interface construct which has used for bunching of all the signals along with clocking block which is used for synchronization unlike Verilog in which instantiation with the DUT becomes tedious because of large number of signals. Verilog uses module level testbench while SV uses Class based testbench which is dynamic in nature. SystemVerilog has got OOPS (object-oriented program) concepts, which is very very essential for bug hunting, without it you can't have bug free HW. It also provides User defined data types like Class, enumeration, struct & assertions, randomization which makes to create test cases easily. It provides lot of reusable blocks which help in code reusability.

Those advantage of SystemVerilog makes us to fast development of our RTL code. And thereby maintaining the code, also minimize the existence of possible situations where RTL code simulates completely different compared to the synthesized netlist. In SystemVerilog you can design at an abstraction level of very high. Because of that reasons portability and code reliability are increased. There are some Advanced features includes interfaces, concise port naming, explicit hardware constructs, and special data types, So on.

6.1 SystemVerilog Basic Testbench Functionality

In order to check for the correctness of the design under test (DUT), need to write testbench. There some simple steps to follow

a. Generating the stimulus.
b. Apply or pass the stimulus to the Design under test.
c. Capture or mark the response.
d. Checking for the correctness.
e. Measuring the progress with respect to the overall goals of verification.

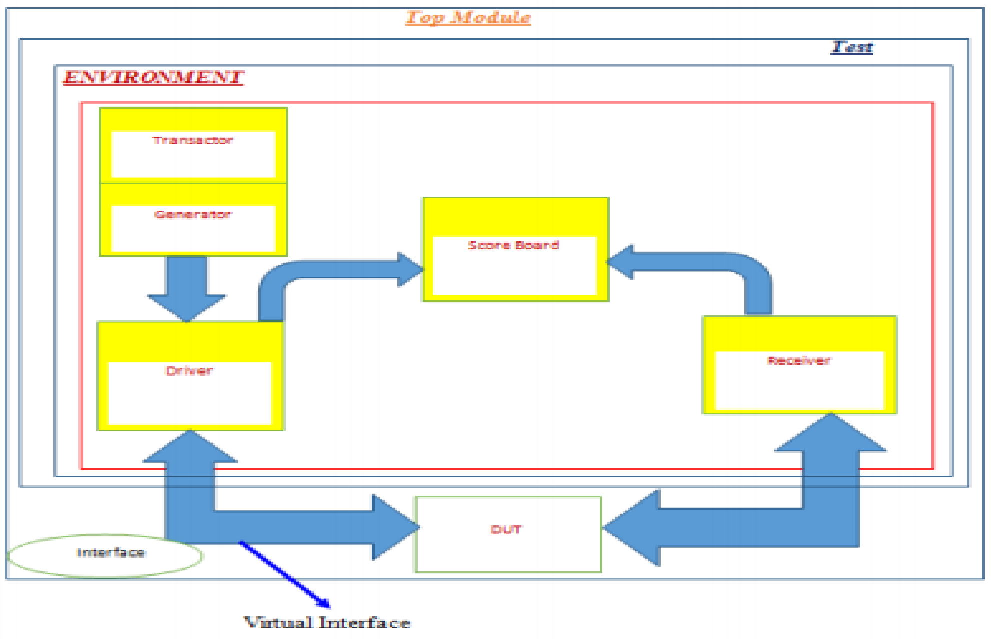


Figure 6.1: DDR SDRAM memory controller verification environment

6.1.1 Transactor Class:

In this class the inputs are declared with RAND and RANDC key words and the type of inputs declaration must be same as in the interface. Declare the outputs using the key word bit. Take one static variable of int type to store the no of transaction_ids. Display the input variables, output variables and transaction _id in display Method which is void type function. We can write the compare logic/check logic to verify that the design is working according to the specified logic are not in compare function. we must pass the arguments transaction class handle as input, output string message for displaying the success or failure (it is optional one) to compare function. finally call the display function in post_randomize() method (in-built method).

6.1.2 Generator Class

Create the 2 handles for transaction class. one for randomizing the input data and another one used for data transferring to driver. Define a mailbox which is connecting the generator and driver at the time of object creation of generator class. Define the function new () constructor and pass mailbox as argument to provide connection between generator and driver. Provide connection between generator and driver. Create the object for transaction class. Define the function generate data () here in this function input data is going to be randomized. Here we can check the data is randomizing are not using randomize () method which is in-built one. Define virtual task start () here we going to put the randomized vales into driver through mailbox.

6.1.3 Driver Class:

First of all, declare the handle for virtual interface and 2 mailboxes. One is used for getting the inputs from generator, another one is used for driving the inputs to scoreboard. Define the handle for transaction class. Declare one event which is used to control the functioning of operation in correct way. Define a function new () constructor and pass above declared virtual interface, mailboxes, event as arguments and provide end to end connections. Write virtual task drive () to driving the values to virtual interface (it internally connects to DUT in top module) by receiving the data from generator via mailbox. Give some delay to complete the given task (driving the inputs to virtual interface) then trigger the event handler.

Define the virtual task start () it is going to put the randomized values to scoreboard via mailbox.

6.1.4 Receiver Class:

The receiver is also known as monitor, used to declare the mailbox and handle for virtual interface. Those are used to transfer the received data from receiver to scoreboard. Define a function new () constructor and passes above declared virtual interface, mailboxes, event as arguments and provide end to end connections. Write the virtual task receive () it will check the event handler is triggered are not. If it is triggered data is derived successfully and is not it will display the event driven is not triggered. And we are receiving data values from virtual interface. Define virtual task start () and put received data to scoreboard via mailbox.

6.1.5 Score Board Class:

This class defines 2 mailboxes one is generator to driver mailbox, another one is receiver to scoreboard. Declare the 3 handles for transaction class, for data transfer from driver, receiver and other for coverage purpose. Declare one event handler for verifying coverage using int variables. Define the cover group for coverage purpose (coverpoints, bins) and function new () constructor in this provide connections to mailboxes and create object for cover group. Declare virtual task start(); in this we are going to call virtual task check this virtual task check(); will call compare function which is defined in transaction class. If compare function is not successful it will display the values stored in variables. If compare function is successful then it will trigger the coverage by calling in-built functions like sample (). The data verified is greater than or equal to no_of_transactions trigger the event for generating the report write the function report which will not return anything. I.e. (function void report ()..).

6.1.6 Environment Class:

Using compiler directive (`include) copy the files (transact or, generator, driver, receiver, and scoreboard). Declare virtual interface handles and create the objects for mailboxes which are generator to driver, driver to scoreboard and receiver to scoreboard. Declare HANDLES for each and every class. Write the function new () constructor inside the new function we are going to provide the end—to –end connections between blocks (generators, driver, receiver and scoreboard) by passing the mailboxes. Write the virtual task build () in this we are going to crate the objects for all classes. Write the virtual task start () it will call the start () task of all classes. Write the virtual task stop () it will stop the execution after the event is triggered. We will write the task run () and call all the virtual tasks (build (), start (), stop () … and scoreboard report () function…).

6.1.7 Test Program:

The test program is executing in reactive region and only one time it will execute.it will not allow always block. Pass the interface mod ports (driver, receiver). [ex: program test_case_name (interface_name. modport_name. handler name) Import the whole package into test program.Create the handle to the environment class. Inside the initial block we are going to create the object for environment class and we can modify the no_of _transactions and we invoke the build (), and run () methods of environment class.

6.1.8 Multiple test case:

The test program is executing in reactive region and only one time it will execute and it will not allow always block. Pass the interface modports (driver, receiver). [ex: program test_case_name (interface_name. modport_name. handler name) Write the class inside the program inside the class we can write the constraints and we can declare the input variables as RANDC Declare the handles for environment class and the class which is written newly in program. Inside the initial block we are going to create the object for environment class and object for newly written class and assign the this newly class handler to generator and we can modify the no_of _transactions and we invoke the build (), and run () methods of environment class.

6.1.9 Interface:

Interface is the method of connecting testbench to the DUT. For example, there is need of physical wires to connect 2 hardware unit blocks. In order to add new connection easily by taking the use of interface block, and the list of ports are very compact.

By using modport, Interface block always carries the directional information and clocking blocks. Declare all inputs and outputs with logic. We can write clocking blocks for driver and receiver (in the case of sequential cuts). We will write the modports for driver and receiver (mod port is providing the particular connections in one direction…). An interface block transfers the communications occurs between DUT and Testbench that will contain connectivity, called bundle of wires. Connecting the one or more Bundles may be used for different tests and Timing (Clocking blocks), devices Directional information (modports), (routines, assertions, initial/always blocks) are the functionalities.

6.1.10 Top Module

Top module is used to declare the global signals, clock and reset. Define handle to interface and Instantiate test class. Instantiate the DUT and provide the connections by using name-based connections via interface handler and generate the clock signal.

6.1.11 System Verilog Assertions:

In order to detect the design expected behavior, the code will be written in HDL or Verilog by using a tool or mechanism called assertions.

When we are checking for a property in simulation if it’s not behaved like what we expect ot behave it, then assertion said to failed. And Assertions are used to capturing the designer’s interpretation of the specification. It normally Describes the property of the design Assertion but that’s not help in designing any module or block or entity. But it always checks for the design behavior. When considered Between modules, assigning the Testbench and design under test to check for communications between the stimulus constraints and perhaps the modules. Assertions also used inside individual modules, in order to verify the design. The affirmation-based confirmation is an extreme administration to plan and check engineers who are vowed for the RTL outline of computerized pieces and systems. Declaration based check engineers catch the confirmation data amid the plan time and it additionally empowers inner state, information way, and blunder precondition scope examination. Attestations are primarily used to approve the conduct of an outline. The statements are check inserted in, or bound to an outline unit amid the reenactment. Notices or blunders are producing on disappointment of particular condition or grouping of occasions. The attestations are composed in HDL, yet HDL declarations can be long and entangled. This catastrophe the reason for statements, which is guarantee the accuracy of the outline. Extensive, complex HDL declarations can be difficult to make and subject to bugs themselves. System Verilog has highlights to determine declarations of a system and affirmations are indicates a conduct of the system. System Verilog Assertions can be grouped into two kinds.

6.1.11.1 Immediate Assertions:

At a given point and in time, the Immediate Assertions are used to determine or check the state of a condition. When the initialization is operated in the procedural code, So the immediate assertions is a request assigned. On sometimes the off chance that the articulation make access to X, Z or 0, and the point it's translated is considered being false and the affirmation is said to be fizzled. Something else, the articulation is translated as being valid and the affirmation is said to be passed. Prompt declarations are for all intents and purposes proportionate of VHDL attest proclamation. Ordinarily we favored $fatal, $error, $warning, which can be smothered in an instrument particular way, $info tells the declaration disappointment conveys no particular seriousness to print a few messages for pass or fizzled code, one can utilize $display, or any normal Verilog code, such as setting off an occasion, or increasing a counter, or calling capacity in pass/come up short piece code.

6.1.11.2 Concurrent Assertions:

Here the conditions can be checked overtime, thereby Concurrent Assertions are used. Simultaneous affirmations depict the conduct that reaches after some time. These are assessed just at the event of a clock ticks. The estimations of factors utilized as a part of this assessment are tested esteems. Thusly result will acquired from the examination and paying little mind to the test system's inward instrument of connect with occasions and assessing occasions. This model of choking likewise compares to the union model of equipment translation from an RTL depiction. An articulation utilized as a part of a statement is constantly settled to clock. The examined esteems make utilized in order to access the esteem difference in articulations that are required to decide a match of a grouping. The watchword property recognizes a simultaneous statement from a prompt declaration.

CHAPTER 7

SYSTEMVERILOG COVERAGE

Coverage is usually used to measure tested and untested part of the design. There are 2 types of coverages. Functional coverage and code coverage.

Code coverage as the name implies measure the coverage across the code. This gives an indication of how well the code is tested by your stimulus. The information is normally collected by simulation tools and the user can analysis and improves their stimulus. Code coverage is normally classified as following types:

1. Line/Statement coverage - Number of lines/statements executed.
2. Expression coverage - Various input combinations in a logical expression executed.
3. Branch/Condition coverage - Branches or conditions executed.
4. Toggle coverage - Signals/nets toggled.
5. FSM Coverage - State transitions in state machine logic.

7.1 Functional coverage is a way in which a user writes certain instrumentation logic to monitor how well the stimulus is covering various functionality. Typically, in SystemVerilog language, a user can use the coverage constructs to write cover points and bins and during simulation, the tools will monitor for coverage of these events.

Functional coverage is more important in a random testing (constrained random verification) to measure the quality of stimulus and completeness. Functionality and interesting design points are monitored using functional cover points and coverage tells how well those are covered.

- **The test points**

It is sometimes called the function point, all of the test points set is called verification space, test point is the minimum segment of the RTL function and design spec, test point do an image of the parable: the test point is the atomic, RTL function and design spec is a molecular, verification space is higher biological.

- **Features of test points**

When we analyze the test points, we must pay attention to these suggestions:

- Certainty: Test points must be analyzed from RTL function and design specifications;
- Uniformity: Each test point in a medium size, don't appear one big test point include multiple small test points;
- Completeness: The test points description should include input data, process, output data;
- Readability: Using professional language descripts test point and don't create ambiguity.

Test point doesn't focus on the accidental error of RTL and non-reasonable doubts.

- **Methods to make test points:**

The test points analyze methods generally have the following kinds:

- Equivalence partitioning: According to class, for example: interface, function, architecture and performance;
- Boundary value: Focus on the test point boundary value, such as the lower bound, upper bound, and random boundary;
- Flow chart: Focus on the realization of the function of RTL process;
- Error backstopping: According to the experience that may cause chip mistakes;
- Scene analysis According to the application of the chip scene analysis test points.

Use function coverage

Once upon a time, there's no function coverage verification method, then it is very difficult to measure verification. On the one hand verification engineers based on the RTL code coverage and RTL state machines coverage, on the other hand the verification engineers based on success rate of the test cases, the two aspects are difficult to affirm all test points have been covered, everyone don't know when to say verification should been done, so there is one sentence from old engineer: validation is everlasting. The relationships among spec, design and the verification just like the three circles above, function coverage can let these three circles into one circle.

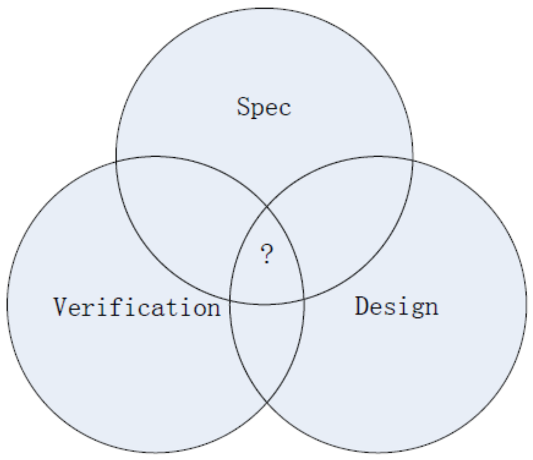


Figure 7.1: Relations among Specification, Design, Verification

Who write function Coverage?

- Verification engineer analyzes/writes/uses the function coverage module.
- Design engineer check the function coverage module.
- Project manager track the function coverage.

SystemVerilog supports function coverage grammar in the language itself, verification engineer would be feeling easier to study them, Questa and VCS have visual analysis GUI and function coverage can be back annotated to documents such as doc and xml, convenient design engineer checking and project manager tracking function coverage over the whole chip.

- **When verification engineer writes function coverage:**

Writing function coverage throughout the whole verification work, it is a very important link, the starting point is preliminary verification environment completed, which provide design engineer to test some easy functions. The primary environment is called sanity testbench and verification engineer also provide design engineer a test case which is called sanity test case, the sanity test case should guarantee the RTL registers, memory and the various input/output interface can be triggered, sanity test case can detect the most basic function of RTL. Design engineer use these components to do preliminary test. When the above things being done, the main energies of the verification engineer should be thrown into the work of analyzing and writing functional coverage.

- **How to write function coverage**

Base on the function coverage of the SystemVerilog, should need to understand the 3 basic concepts.

7.2 Covergroup and Coverpoint:

- **Covergroup**:

SystemVerilog covergroup is a user defined type that compact the specification of a coverage model. They can be defined once and instantiated multiple times. Covergroup can be used in a module, interface or class, program, package and usually encapsulates following information: Covergroup corresponds to the test point, the relationship is one-to-one usually, that is one covergroup corresponds to one test point, covergroup be composed by coverpoints.

1. Group of coverage points.
2. coverage points in between cross coverage.
3. An event that defines when the covergroup is sampled.
4. Other options to configure coverage object.

- **Coverpoint:**

Evaluation of the coverpoint expression happens when the covergroup is sampled. The SystemVerilog coverage point can be optionally labeled with a colon: The example shown below randomizes the two variables mode and cfg multiple times and is assigned a value on every neg-edge of the clock cycle. The covergroup specifies the samples at every occurrence of a rising edge of the clock. So two variables are randomized 5 times at the falling edges of clock and sampled at rising edges of clock. Coverpoint be concerned by test points, the covergourp and coverpoint can be one-to-one or one-to-many relationship, such as a test point: when A send ten back-to-back packets to B continuous, then B will send back pressure signal to A and B, in this example, the number of packet is one coverpoint, the range between packets is one coverpoint and the back pressure signal is one coverpoint, finally these coverpoint should be crossed to ensure cross validation of space can be overwritten to.

7.3 Coverage Bins

SystemVerilog automatically creates a bin for coverage point, and define explicitly the bins construct to name each bin. The ways creating or generating bins are.

- Generic coverage group and Implicit bins creation.
- Explicitly bins creation and Array of bins creation.
- Default bins creation/ and Wildcard bin creation.
- Ignore bin creation, illegal bin creation, and Transition bins creation.

7.4 Advantages of SystemVerilog

S**ystemVerilog** was originally intended as an extension to Verilog 2005 and became IEEE standard 1800. It was published as a separated documented and consists of hundreds of enhancements and extensions to verilog. In 2009 it officially became a super set of Verilog and was again updated in 2012 as **IEEE 1800-2012** standard.

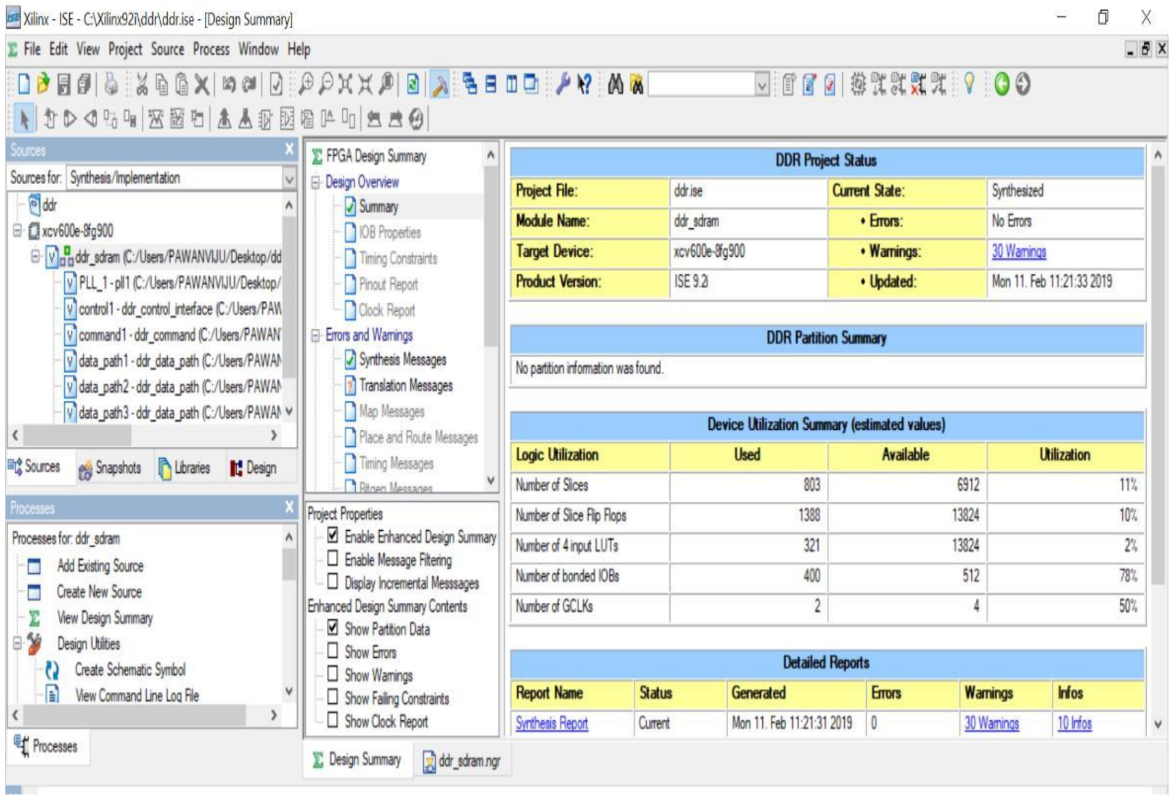
There were 5 major areas where enhancements were added in SystemVerilog

- **SVD-System Verilog for Design:** This includes several enhancements to design constructs.
- **SVTB-SystemVerilog for Testbenches:** This was the biggest set of enhancement in SystemVerilog for support all the Testbench modelling and needs for newer verification methodologies. This includes at a high level - Object Oriented Programming support with Classes, a constraint solver with several capabilities for creating constrained random stimulus, concurrent processes, semaphores, mailboxes and many more.
- **SVA-System Verilog Assertions: This includes several** Features for temporal and concurrent assertions like properties and sequences.
- **SVDPI-SV Direct Programming Interface: This includes features for** better C/C++ integration.
- **SVAPI-SV Application Programming Interface:** This includes features for better integration of APIs for Coverage and assertions.

CHAPTER 8

RESULT AND DISCUSSIONS

High speed, pipeline and burst access features makes DDR SDRAM controller has best choice in system design. Use of DDR SDRAM has two advantages that it reduces the system cost and increases the data transfer throughput. In this proposed design DDR SDRAM controller is designed and verified using Xilinx simulator and coverage is carried using Questsim SystemVerilog. DDR SDRAM controller has been implemented using Verilog HDL.



DDR Project Status

Project File:	ddr.ise	Current State:	Synthesized
Module Name:	ddr_sdram	• Errors:	No Errors
Target Device:	xcv600e-8fg900	• Warnings:	30 Warnings
Product Version:	ISE 9.2i	• Updated:	Mon 11. Feb 11:21:33 2019

DDR Partition Summary

No partition information was found.

Device Utilization Summary (estimated values)

Logic Utilization	Used	Available	Utilization
Number of Slices	803	6912	11%
Number of Slice Flip Flops	1388	13824	10%
Number of 4 input LUTs	321	13824	2%
Number of bonded IOBs	400	512	78%
Number of GCLKs	2	4	50%

Detailed Reports

Report Name	Status	Generated	Errors	Warnings	Infos
Synthesis Report	Current	Mon 11. Feb 11:21:31 2019	0	30 Warnings	10 Infos

Figure 8.1: Design summary for the designed DDR SDRAM controller

The design was coded in Verilog HDL and was synthesized on XILINX ISE tool to obtain the synthesis results. Figure 8.1 shows the design summary. The design occupies very less numbers of slices (10% of total available slices) and LUTs (2% of total available LUT’s) and hence it was possible to implement on XILINX Spartan III FPGA.

Top level DDR SDRAM Schematic:

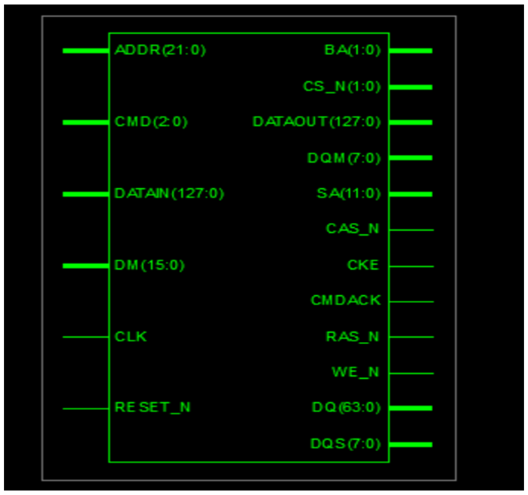


Figure 8.2: Schematic of DDR SDRAM controller

Figure 8.2: top-level schematic of DDR SDRAM generated from Xilinx tool.

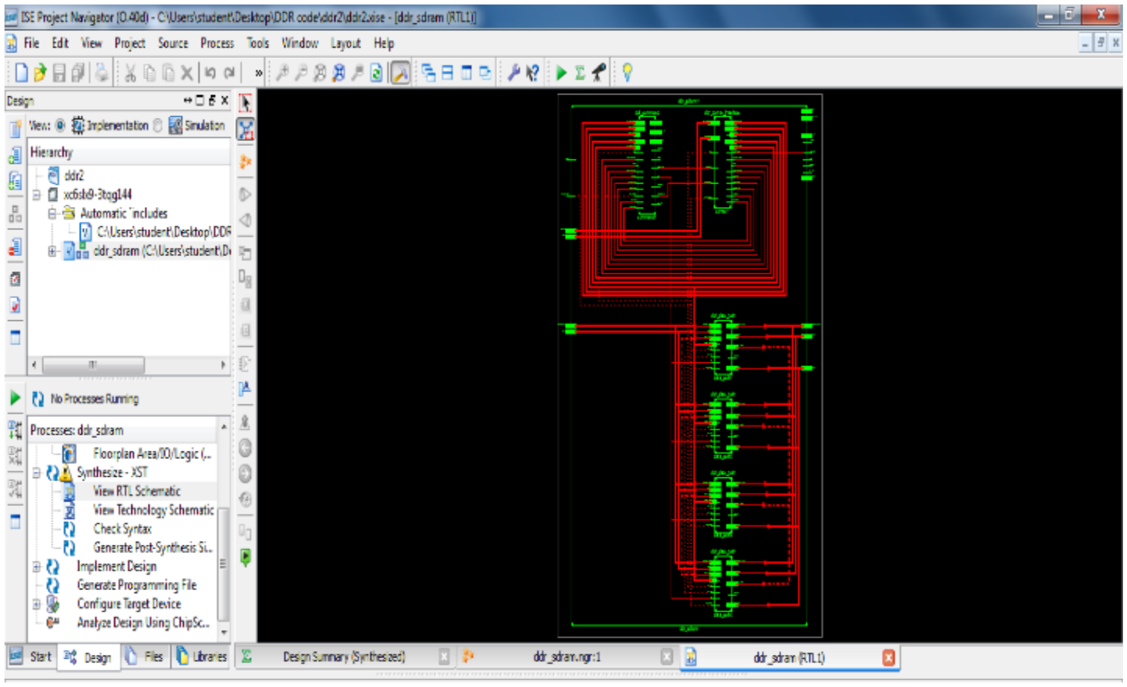
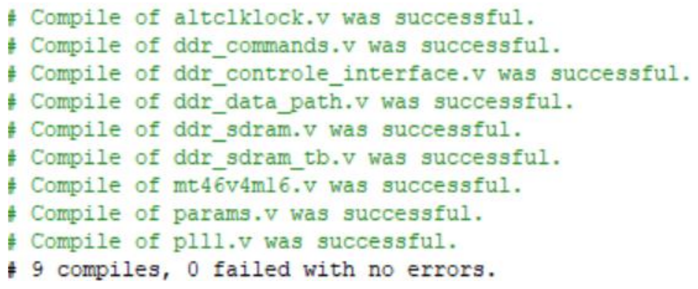

ISE Project Navigator (O.40d) - C:\Users\student\Desktop\DDR code\ddr2\ddr2.xise - [ddr_sdram (RTL1)]
File
Edit
View
Project
Source
Process
Tools
Window
Layout
Help
Design
View: Implementation Simulation
Hierarchy
ddr2
xc6slx9-3tqg144
Automatic `includes
C:\Users\student\Desktop\DDR
ddr_sdram (C:\Users\student\D
No Processes Running
Processes: ddr_sdram
Floorplan Area/IO/Logic (...
Synthesize - XST
View RTL Schematic
View Technology Schematic
Check Syntax
Generate Post-Synthesis Si...
Implement Design
Generate Programming File
Configure Target Device
Analyze Design Using ChipSc...
Start
Design
Files
Libraries
Design Summary (Synthesized)
ddr_sdram.ngr:1
ddr_sdram (RTL1)

Figure 8.3: RTL Internal architecture of SDRAM Controller

Figure 8.3 shows the internal architecture of DDR SDRAM memory controller is consisting of control interface module, command module and four data path modules. Each data path module has 32-bit data bus width, so a total of 128-bit data bus width used for this DDR SDRAM Controller.

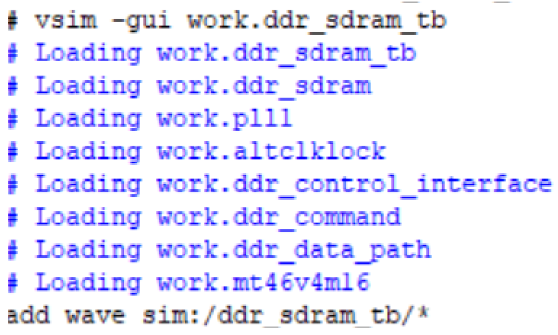
8.1 DDR SDRAM Design output in Verilog HDL

To test the design of DDR SDRAM controller, Verilog code is compiling and simulated. A sample completion of Verilog code and its simulation process using Modelsim is shown in figure 8.4 and 8.5.



```
# Compile of altclklock.v was successful.
# Compile of ddr_commands.v was successful.
# Compile of ddr_controle_interface.v was successful.
# Compile of ddr_data_path.v was successful.
# Compile of ddr_sdram.v was successful.
# Compile of ddr_sdram_tb.v was successful.
# Compile of mt46v4ml6.v was successful.
# Compile of params.v was successful.
# Compile of plll.v was successful.
# 9 compiles, 0 failed with no errors.
```

Figure 8.4: Compilation of Verilog codes

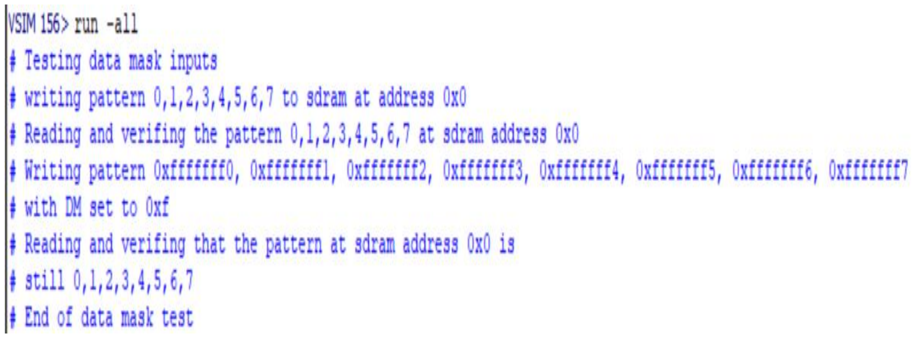


```
# vsim -gui work.ddr_sdram_tb
# Loading work.ddr_sdram_tb
# Loading work.ddr_sdram
# Loading work.plll
# Loading work.altclklock
# Loading work.ddr_control_interface
# Loading work.ddr_command
# Loading work.ddr_data_path
# Loading work.mt46v4ml6
add wave sim:/ddr_sdram_tb/*
```

Figure 8.5: Simulation of Verilog codes

8.1.1 Data mask operation

To have data integrity, data is encrypted. Its required to test whether encrypted data is written and read properly. The size of address line is 22-bits and data size is 128 bits. Figure 8.6 shows initialization of data mask inputs.

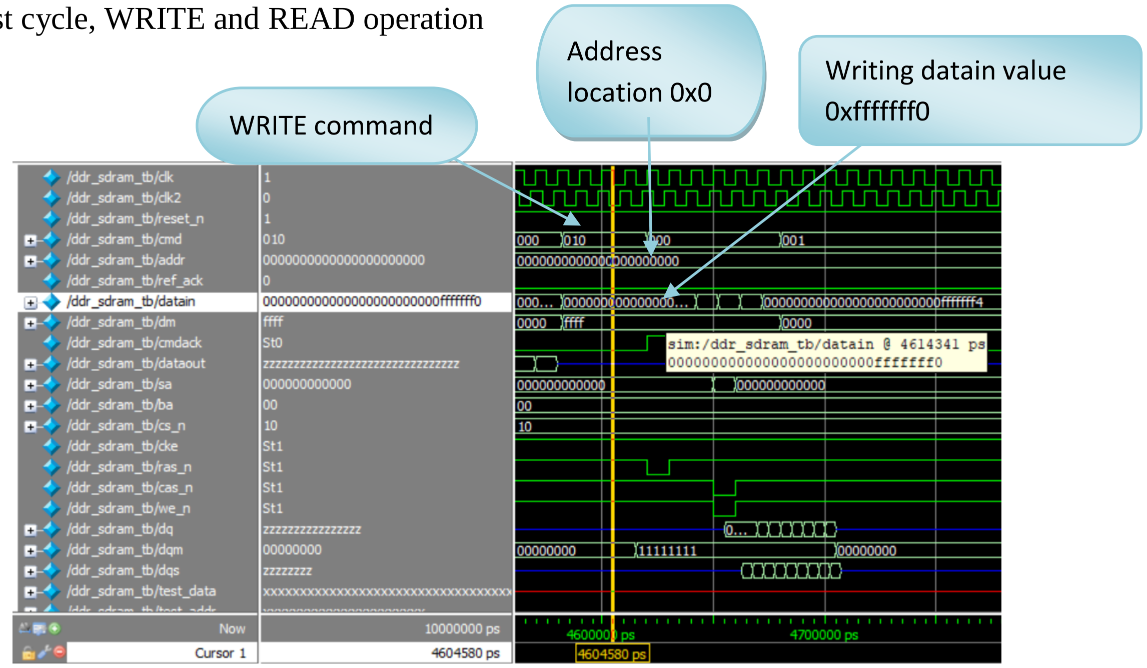


```
VSIM 156> run -all
# Testing data mask inputs
# writing pattern 0,1,2,3,4,5,6,7 to sdram at address 0x0
# Reading and verifing the pattern 0,1,2,3,4,5,6,7 at sdram address 0x0
# Writing pattern 0xfffffff0, 0xfffffff1, 0xfffffff2, 0xfffffff3, 0xfffffff4, 0xfffffff5, 0xfffffff6, 0xfffffff7
# with DM set to 0xf
# Reading and verifing that the pattern at sdram address 0x0 is
# still 0,1,2,3,4,5,6,7
# End of data mask test
```

Figure 8.6: Initialization of data mask inputs

- **Data Mask WRITE and READ operations**

1. First cycle, WRITE and READ operation

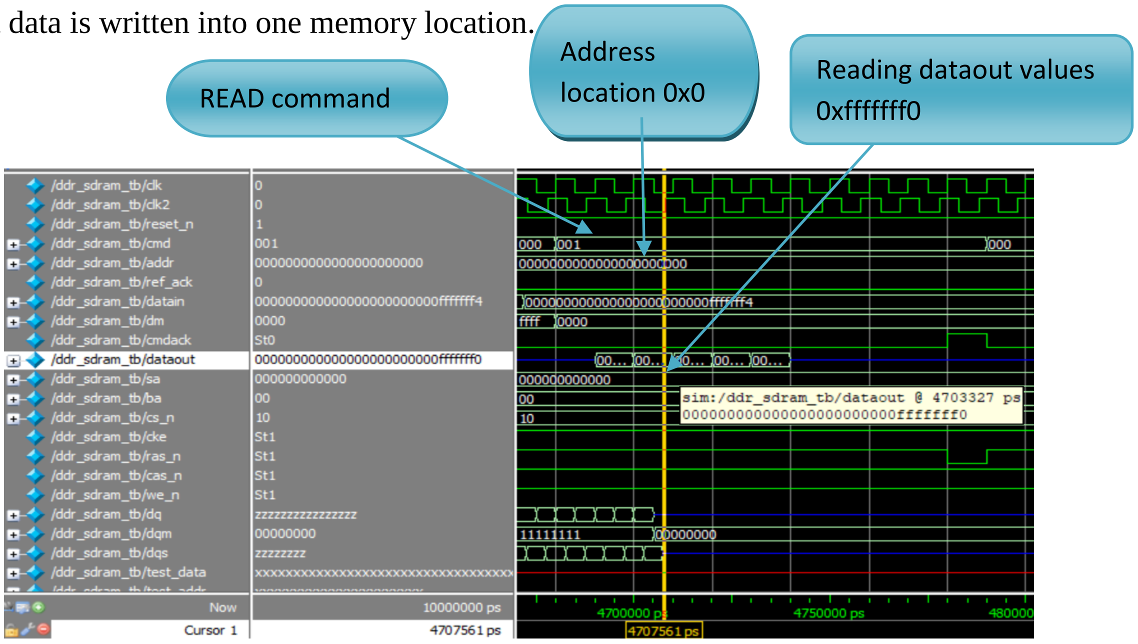


```
/ddr_sdram_tb/clk         1
/ddr_sdram_tb/clk2        0
/ddr_sdram_tb/reset_n     1
/ddr_sdram_tb/cmd         010
/ddr_sdram_tb/addr        0000000000000000000000
/ddr_sdram_tb/ref_ack     0
/ddr_sdram_tb/datain      000000000000000000000000fffffff0
/ddr_sdram_tb/dm          ffff
/ddr_sdram_tb/cmdack      St0
/ddr_sdram_tb/dataout     zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
/ddr_sdram_tb/sa          000000000000
/ddr_sdram_tb/ba          00
/ddr_sdram_tb/cs_n        10
/ddr_sdram_tb/cke         St1
/ddr_sdram_tb/ras_n       St1
/ddr_sdram_tb/cas_n       St1
/ddr_sdram_tb/we_n        St1
/ddr_sdram_tb/dq          zzzzzzzzzzzzzzzz
/ddr_sdram_tb/dqm         00000000
/ddr_sdram_tb/dqs         zzzzzzzz
/ddr_sdram_tb/test_data   xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Now                       10000000 ps
Cursor 1                  4604580 ps

sim:/ddr_sdram_tb/datain @ 4614341 ps
000000000000000000000000fffffff0
```

Figure 8.7: Write 0x fffffff0 to address location 0x0

Figure 8.7 Shows, examples of writing operation to DDR memory bank 0 (first bank). Initially the reset signal is made high and command 010 is selected for writing data into memory bank. During the negative edge of clock, the input data is written into memory location 0x0000000000000000000000 (address zero) and data written is 0xfffffff0. Thus successfully 128-bit data is written into one memory location.



```
/ddr_sdram_tb/clk         0
/ddr_sdram_tb/clk2        0
/ddr_sdram_tb/reset_n     1
/ddr_sdram_tb/cmd         001
/ddr_sdram_tb/addr        0000000000000000000000
/ddr_sdram_tb/ref_ack     0
/ddr_sdram_tb/datain      000000000000000000000000fffffff4
/ddr_sdram_tb/dm          0000
/ddr_sdram_tb/cmdack      St0
/ddr_sdram_tb/dataout     000000000000000000000000fffffff0
/ddr_sdram_tb/sa          000000000000
/ddr_sdram_tb/ba          00
/ddr_sdram_tb/cs_n        10
/ddr_sdram_tb/cke         St1
/ddr_sdram_tb/ras_n       St1
/ddr_sdram_tb/cas_n       St1
/ddr_sdram_tb/we_n        St1
/ddr_sdram_tb/dq          zzzzzzzzzzzzzzzz
/ddr_sdram_tb/dqm         00000000
/ddr_sdram_tb/dqs         zzzzzzzz
/ddr_sdram_tb/test_data   xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Now                       10000000 ps
Cursor 1                  4707561 ps

sim:/ddr_sdram_tb/dataout @ 4703327 ps
000000000000000000000000fffffff0
```

Figure 8.8: Read 0xfffffff0 from the address location 0x0

Figure 8.8 Shows, examples of reading operation from DDR memory bank 0(first bank). Initially the reset signal is made high and command 001 is selected for reading data from memory bank. During the positive edge of clock, the output data is read from memory location 0x0000000000000000000000 (address zero) and data read is 0xfffffff0. Thus successfully 128-bit data is read form one memory location.

2. Second cycle, WRITE and READ operation

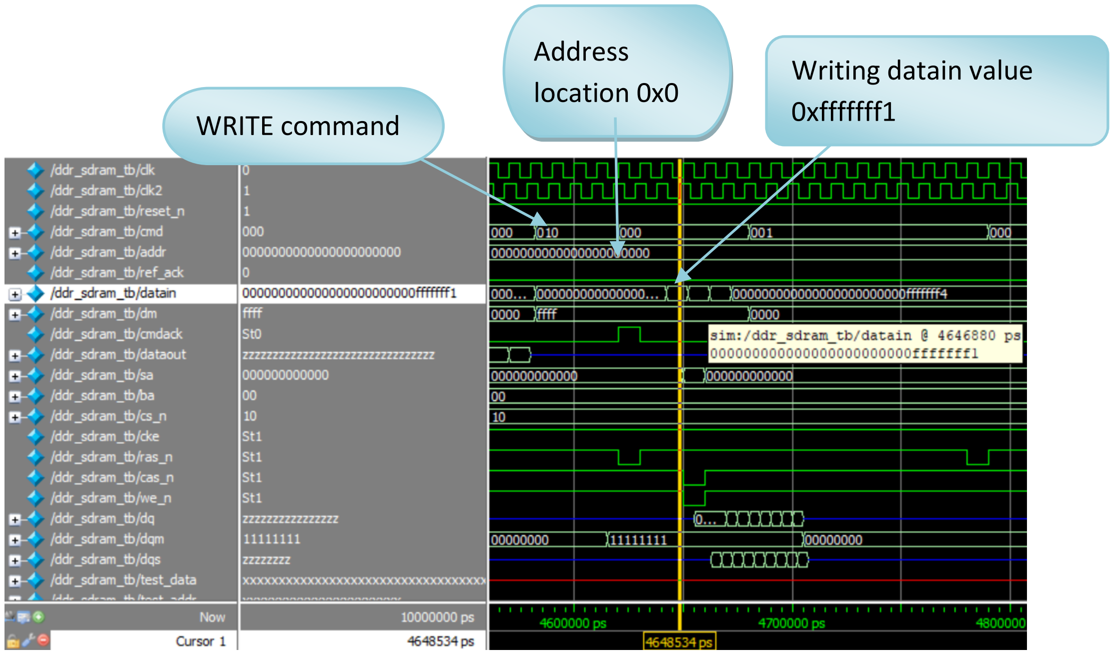
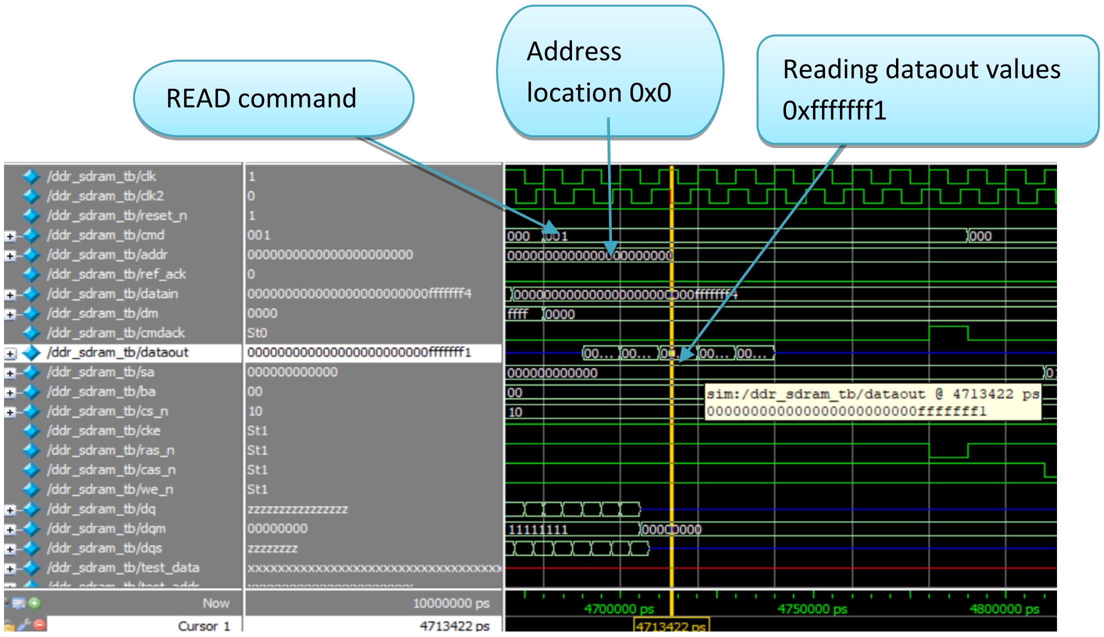


Figure 8.9: Write 0x fffffff1 to address location 0x0

Figure 8.9 Shows, examples of writing operation to DDR memory bank 0(first bank). Initially the reset signal is made high and command 010 is selected for writing data into memory bank. During the negative edge of clock, the input data is written into memory location 0x0000000000000000000000 (address zero) and data written is 0xfffffff1. Thus, successfully 128-bit data is written into one memory location.



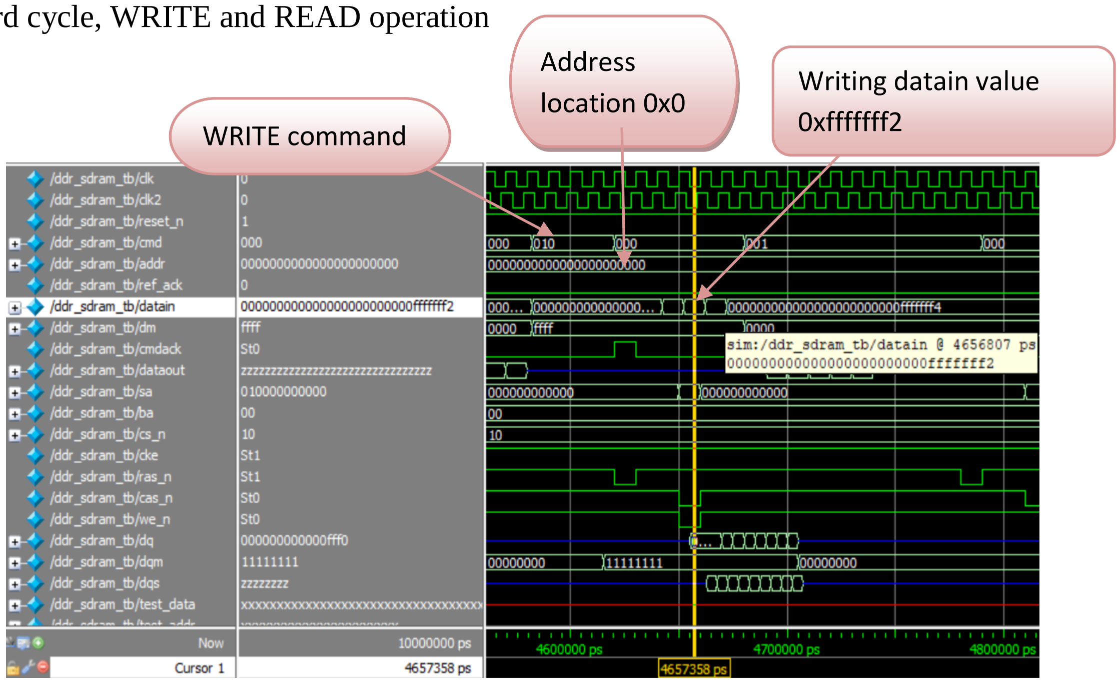
```
/ddr_sdram_tb/clk        1
/ddr_sdram_tb/clk2       0
/ddr_sdram_tb/reset_n    1
/ddr_sdram_tb/cmd        001
/ddr_sdram_tb/addr       0000000000000000000000
/ddr_sdram_tb/ref_ack    0
/ddr_sdram_tb/datain     000000000000000000000000fffffff4
/ddr_sdram_tb/dm         0000
/ddr_sdram_tb/cmdack     St0
/ddr_sdram_tb/dataout    000000000000000000000000fffffff1
/ddr_sdram_tb/sa         000000000000
/ddr_sdram_tb/ba         00
/ddr_sdram_tb/cs_n       10
/ddr_sdram_tb/cke        St1
/ddr_sdram_tb/ras_n      St1
/ddr_sdram_tb/cas_n      St1
/ddr_sdram_tb/we_n       St1
/ddr_sdram_tb/dq         zzzzzzzzzzzzzzzz
/ddr_sdram_tb/dqm        00000000
/ddr_sdram_tb/dqs        zzzzzzzz
/ddr_sdram_tb/test_data  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                  Now    10000000 ps
             Cursor 1    4713422 ps

sim:/ddr_sdram_tb/dataout @ 4713422 ps
000000000000000000000000fffffff1
```

Figure 8.10: Read 0xfffffff1 from the address location 0x0

Figure 8.10 Shows, examples of reading operation from DDR memory bank 0(first bank). Initially the reset signal is made high and command 001 is selected for reading data from memory bank. During the positive edge of clock, the output data is read from memory location 0x0000000000000000000000 (address zero) and data read is 0xfffffff1. Thus, successfully 128-bit data is read form one memory location.

3. Third cycle, WRITE and READ operation

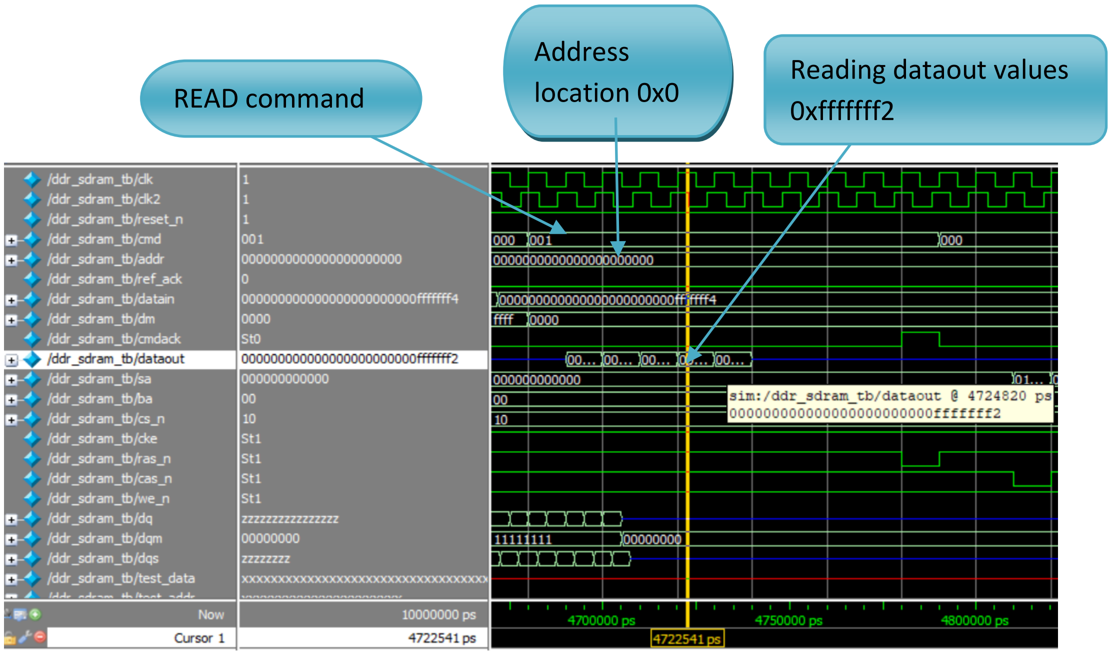


```
/ddr_sdram_tb/clk        0
/ddr_sdram_tb/clk2       0
/ddr_sdram_tb/reset_n    1
/ddr_sdram_tb/cmd        000
/ddr_sdram_tb/addr       0000000000000000000000
/ddr_sdram_tb/ref_ack    0
/ddr_sdram_tb/datain     000000000000000000000000fffffff2
/ddr_sdram_tb/dm         ffff
/ddr_sdram_tb/cmdack     St0
/ddr_sdram_tb/dataout    zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
/ddr_sdram_tb/sa         010000000000
/ddr_sdram_tb/ba         00
/ddr_sdram_tb/cs_n       10
/ddr_sdram_tb/cke        St1
/ddr_sdram_tb/ras_n      St1
/ddr_sdram_tb/cas_n      St0
/ddr_sdram_tb/we_n       St0
/ddr_sdram_tb/dq         000000000000fff0
/ddr_sdram_tb/dqm        11111111
/ddr_sdram_tb/dqs        zzzzzzzz
/ddr_sdram_tb/test_data  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                  Now    10000000 ps
             Cursor 1    4657358 ps

sim:/ddr_sdram_tb/datain @ 4656807 ps
000000000000000000000000fffffff2
```

Figure 8.11: Write 0x fffffff2 to address location 0x0

Figure 8.11 Shows, examples of writing operation to DDR memory bank 0(first bank). Initially the reset signal is made high and command 010 is selected for writing data into memory bank. During the pos-edge of clock, the input data written into memory location 0x0000000000000000000000 (address zero) and data written is 0xfffffff2. Thus successfully 128-bit data is written into one memory location. For all these write operation, data mask set to 0xffff.



```
/ddr_sdram_tb/clk        1
/ddr_sdram_tb/clk2       1
/ddr_sdram_tb/reset_n    1
/ddr_sdram_tb/cmd        001
/ddr_sdram_tb/addr       0000000000000000000000
/ddr_sdram_tb/ref_ack    0
/ddr_sdram_tb/datain     000000000000000000000000fffffff4
/ddr_sdram_tb/dm         0000
/ddr_sdram_tb/cmdack     St0
/ddr_sdram_tb/dataout    000000000000000000000000fffffff2
/ddr_sdram_tb/sa         000000000000
/ddr_sdram_tb/ba         00
/ddr_sdram_tb/cs_n       10
/ddr_sdram_tb/cke        St1
/ddr_sdram_tb/ras_n      St1
/ddr_sdram_tb/cas_n      St1
/ddr_sdram_tb/we_n       St1
/ddr_sdram_tb/dq         zzzzzzzzzzzzzzzz
/ddr_sdram_tb/dqm        00000000
/ddr_sdram_tb/dqs        zzzzzzzz
/ddr_sdram_tb/test_data  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                   Now   10000000 ps
              Cursor 1   4722541 ps

sim:/ddr_sdram_tb/dataout @ 4724820 ps
000000000000000000000000fffffff2
```

Figure 8.12: Read 0xfffffff2 from the address location 0x0

Figure 8.12 Shows, examples of reading operation from DDR memory bank 0 (first bank). Initially the reset signal is made high and command 001 is selected for reading data from memory bank. During the positive edge of clock, the output data is read from memory location 0x0000000000000000000000 (address zero) and data read is 0xfffffff2. Thus, successfully 128bit data is read form one memory location. For all these read operation, data mask set to 0x0000.

8.1.2 DDR SDRAM Write operation

DDR write operation starts with making the reset signal high, (CKE) clock enable signal is always high whenever command signal selected, except SELF_REFRESH operation. For write operation, select the bank and column to write. Make the CS (chip select), CAS (column access strobe), WE (write enable) signals to high and RAS (row access strobe) to low. and select write command 010 to perform write operation. select the any one bank to perform write operations, then it starts executing the operation by writing the input data values to the particular address locations in a memory bank.

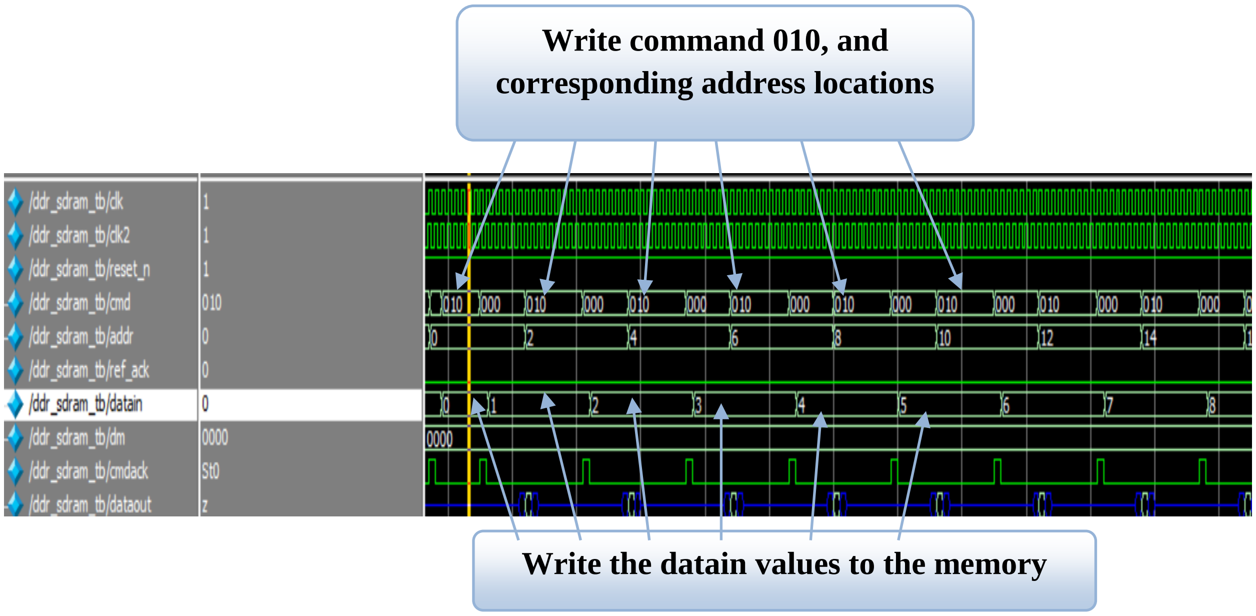


Figure 8.13: Writing 128-bit data to the first memory bank of DDR SDRAM.

From figure 8.13, it shows that input data (0),(1),(2),(3),(4),(5),(6),(7) (unsigned 128 bit data) are written to the corresponding address location (0),(2),(4),(6),(8),(10),(12),(14) (unsigned 22 bit address) respectively. Each address location can store one input data of (128 bit). For example, from in figure 8.13 input data of 0 (128 bit unsigned) is written to address location 0 (22-bit unsigned address). Similarly, different data input is chosen and are written too different address locations successfully.

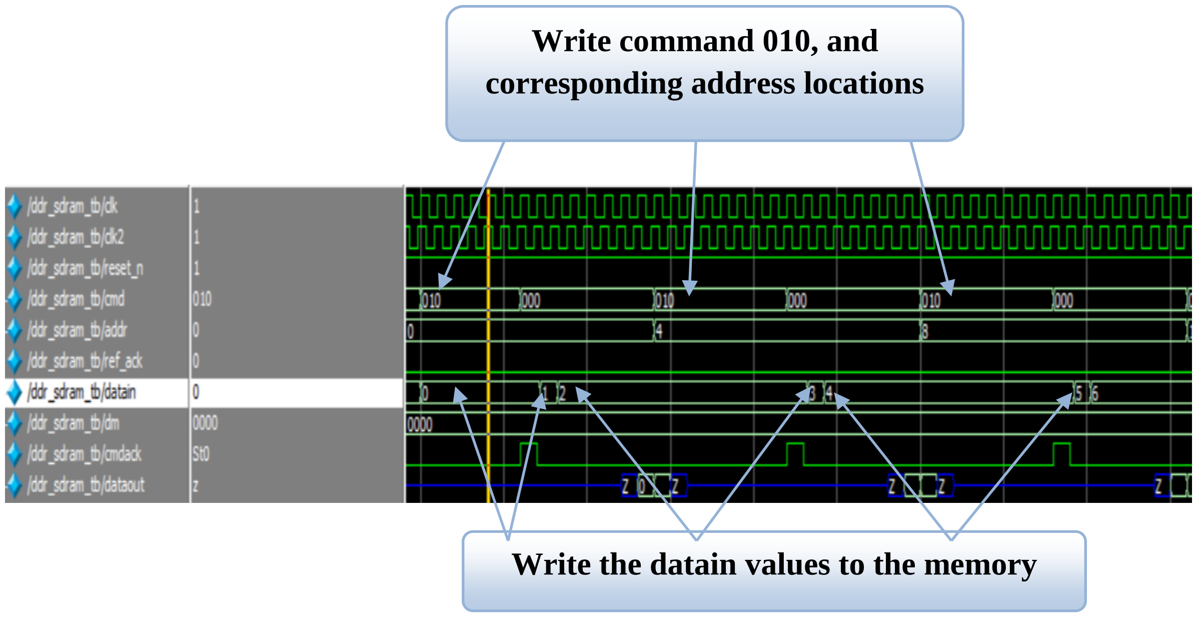


Figure 8.14: Writing two data (each of 64 bits) to first memory bank of DDR SDRAM

From figure 8.14 shows that input data value is (0 and 1), (2 and 3), (4 and 5), (each 64-bit unsigned data) are write to the corresponding address location (0), (4), (8) (each 22-bit unsigned address) respectively. Here per each address location two input data (each of 64-bit unsigned data, so total of 128 bits data) is writing to the first memory bank. For example, from the figure input data of (0 and 1) (each of 64-bit unsigned data, so total 128-bit data) are write to address location 0 (22-bit unsigned address). Input data of (2 and 3) (each of 64-bit unsigned data, so total 128-bit data) are write to address location of 4 (22-bit unsigned address). And input data of (4 and 5) (each of 64-bit unsigned data, so total of 128 bits data) are write to address location 8 (22-bit unsigned address). Similarly, different data input is chosen are written too different address locations successfully.

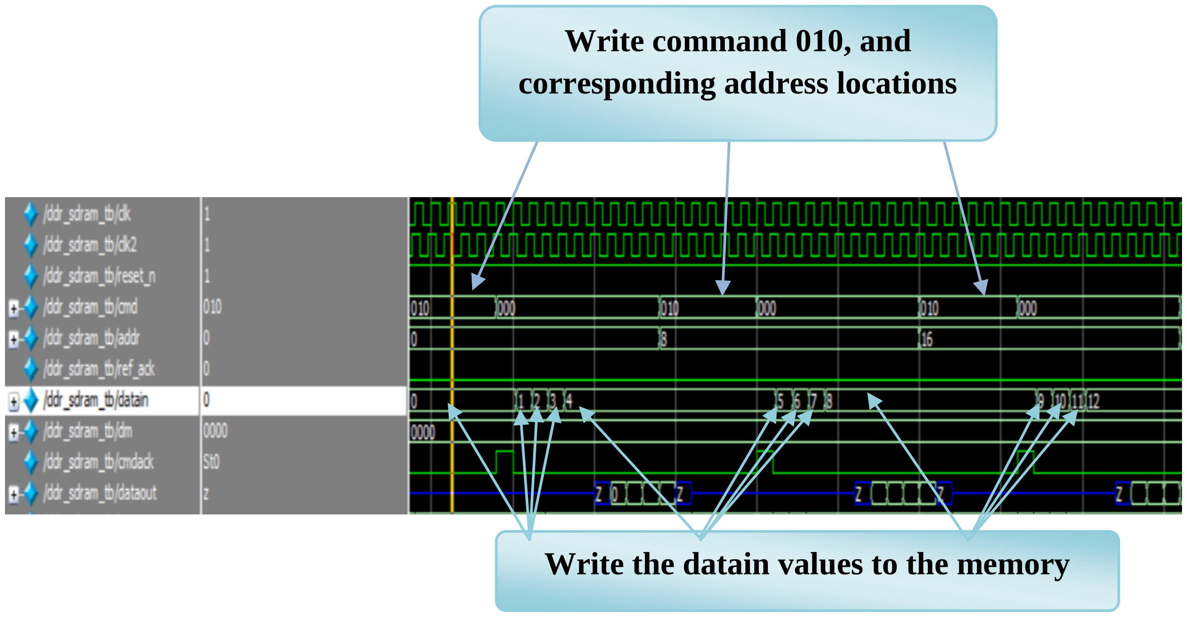


Figure 8.15: Writing four data (each of 32 bit) to first memory bank of DDR SDRAM

From figure 8.15 shows that input data value is (0, 1, 2 and 3), (4, 5, 6 and 7), (8, 9, 10 and 11), (each 32-bit unsigned data) are write to the corresponding address location (0), (8), (16) (each 22-bit unsigned address) respectively. Here per each address location four input data (each of 32-bit unsigned data, so total of 128 bits data) is writing to the first memory bank. For example, from the figure input data of (0, 1, 2 and 3) (each of 32-bit unsigned data, so total 128-bit data) are write to address location 0 (22-bit unsigned address). Input data of (4, 5, 6 and 7) (each of 32-bit unsigned data, so total 128-bit data) are write to address location of 8 (22-bit unsigned address). And input data of (8, 9, 10 and 11) (each of 32-bit unsigned data, so total of 128 bits data) are write to address location 16 (22-bit unsigned address). Similarly, different data input is chosen are written too different address locations successfully.

8.1.3 DDR SDRAM Read operation

DDR read operation starts with making the reset signal high, (CKE) clock enable signal is always high whenever passing command signal, except SELF_REFRESH operation. For read operation, select the bank and column to read, Make the CS (chip select), CAS (column access strobe), signals to high, and RAS (row access strobe) and WE (write enable) to low, And issue command 001 to command (cmd) signal, select the particular bank to perform read operations, then it starts executing the operation by reading particular data stored in particular address locations in a memory bank.

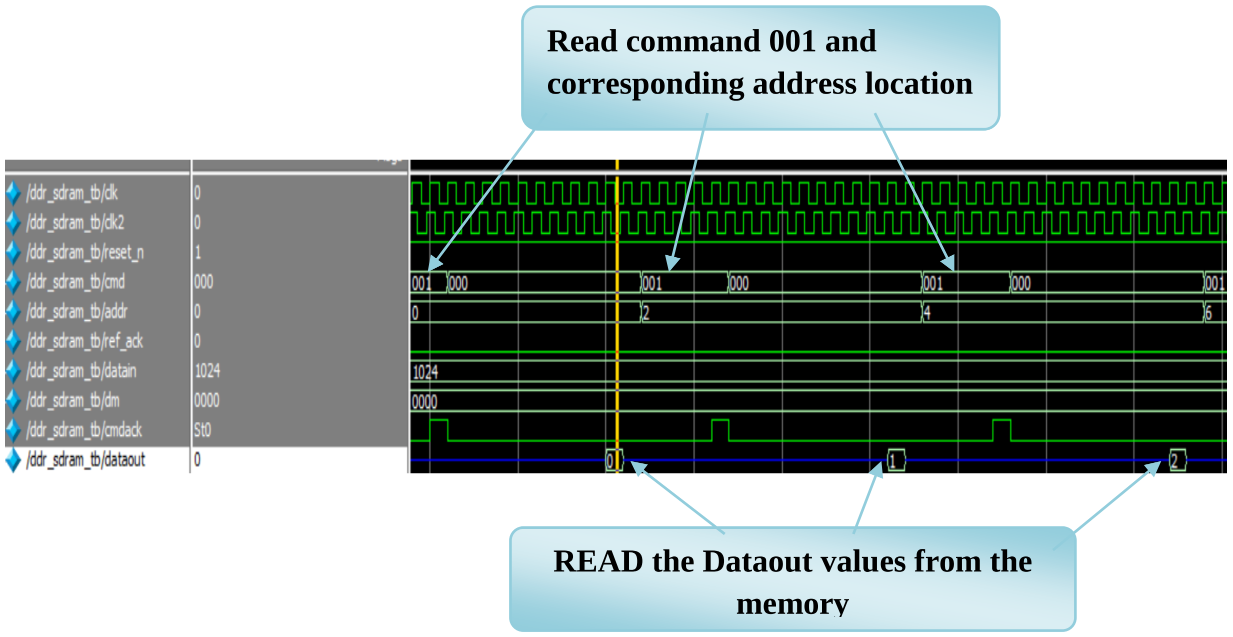


Figure 8.16: Reading one data (128 bit) from the first memory bank of DDR SDRAM

Figure 8.16 shows, at address location (0), (2), (4), (unsigned 22-bit address), data out reads the corresponding values (0), (1), (2) (each of 128-bit unsigned data) respectively. From each address location, can read one input data of size 128-bit unsigned data. For example, from in figure 8.15, output data reads 0 (128-bit unsigned data) from an address location 0 (22-bit unsigned address). Output data of 1 (128-bit unsigned data) is read from address location 2 (22-bit unsigned address). And output data of 2 (128-bit unsigned data) is read from address location 4 (22-bit unsigned address). Similarly, different data input is chosen are written too different address locations successfully.

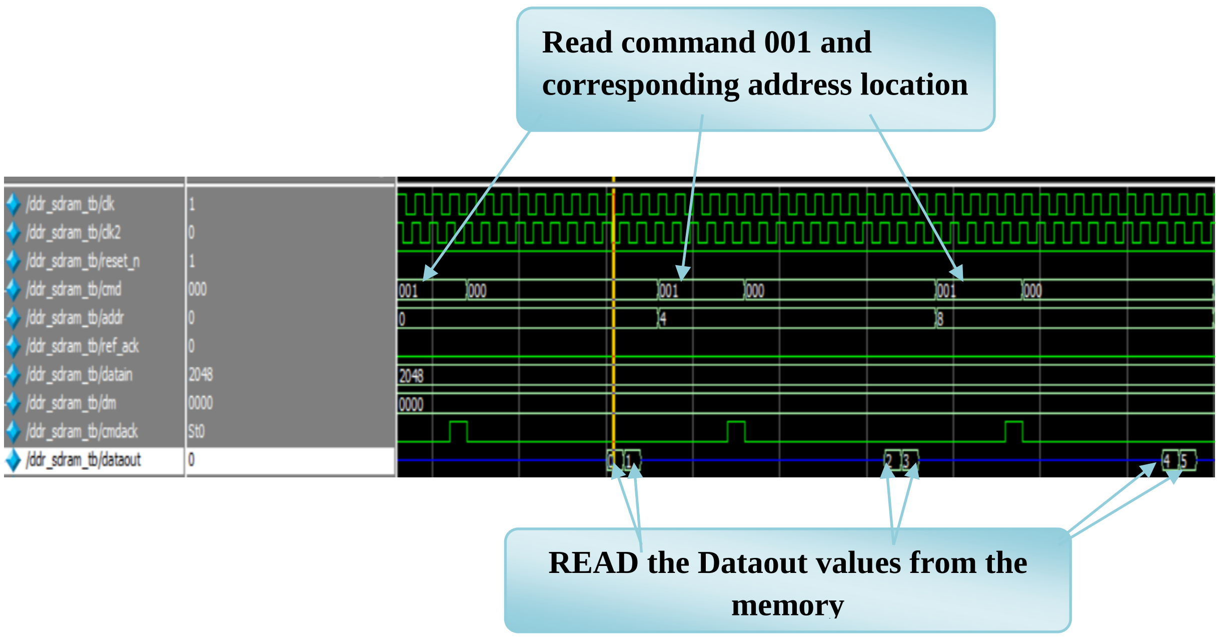


Figure 8.17: Reading two data (each of 64 bit) from the first memory bank of DDR SDRAM

Figure 8.17 shows, at address location (0), (4), (8), (unsigned 22-bit address), data out reads the corresponding values (0 and 1), (2 and 3), (4 and 5) (each of 64-bit, so total 128-bit unsigned data) respectively. From each address location, can read two input data of each of 64-bit unsigned data, so total of 128-bit data. For example, from in figure 8.16, output data reads (0 and 1) (each of 64-bit, so total 128-bit unsigned data) from an address location 0 (22-bit unsigned address). Output data of (2 and 3) (each of 64-bit, so total 128-bit unsigned data) is read from address location 4 (22-bit unsigned address). And output data of (4 and 5) (each of 64-bit, so total 128-bit unsigned data) is read from address location 8 (22-bit unsigned address). Similarly, different data input is chosen are written too different address locations successfully.

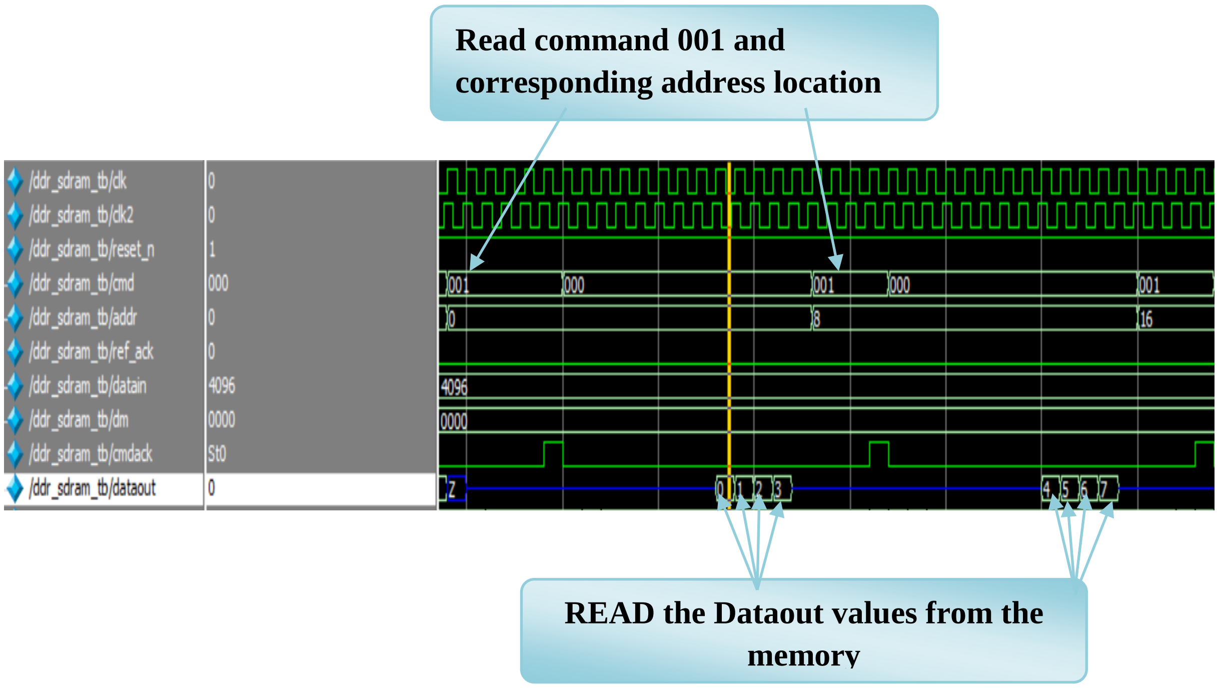


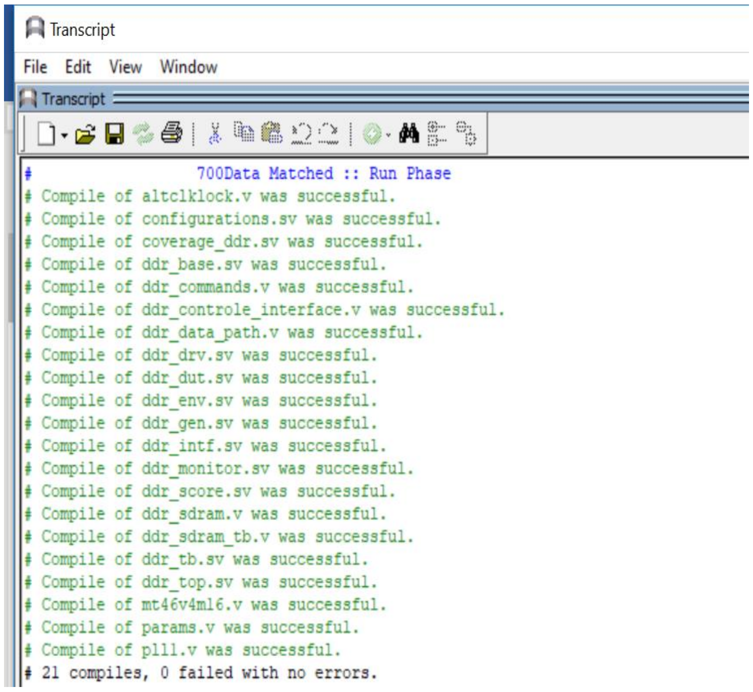
Figure 8.18: Reading four data (each of 32 bit) from the first memory bank of DDR SDRAM

Figure 8.18 shows, at address location (0), (8), (unsigned 22-bit address), data out reads the corresponding values (0, 1, 2 and 3), (4, 5, 6 and 7) (each of 32-bit unsigned data, so total of 128 bit) respectively. From each address location, can read four input data of each of 32-bit unsigned data, so total of 128-bit data. For example, from in figure 8.17, output data reads (0, 1, 2 and 3) (each of 32-bit, so total 128-bit unsigned data) from an address location 0 (22-bit unsigned address). Output data of (4, 5, 6 and 7) (each of 32-bit, so total 128-bit unsigned data) is read from address location 8 (22-bit unsigned address). Similarly, different data input is chosen are written too different address locations successfully.

8.2 DDR SDRAM Verification output in SystemVerilog HVL

The randomization of stimulus in Verilog is very limited but SystemVerilog provides huge scope of randomization using CDV (Coverage Driven Verification) and CRV (Constrained Random Verification) and provides useful tool like functional coverage to hit more uncovered areas and to determine whether testing has been completed or not. SystemVerilog is provided with the powerful features of Assertion which increases the observability of a block to check whether any bug is there by providing intermediate check points which is completely absent in Verilog. Moreover, in SystemVerilog assertions there are constructs like assert and cover which are used to assert and cover a property which reduces debug time and verification spin cycle.

SystemVerilog has got OOPS (object-oriented programming) concepts, which are very essential for finding bug, and provides bug free hardware. It also provides user defined data types like class, enumeration, structure, assertions and randomization which makes to create test cases easily. It provides lot of reusable blocks which help in code reusability. The advantage of SystemVerilog makes us to fast development of our RTL code



```
#                  700Data Matched :: Run Phase
# Compile of altclklock.v was successful.
# Compile of configurations.sv was successful.
# Compile of coverage_ddr.sv was successful.
# Compile of ddr_base.sv was successful.
# Compile of ddr_commands.v was successful.
# Compile of ddr_controle_interface.v was successful.
# Compile of ddr_data_path.v was successful.
# Compile of ddr_drv.sv was successful.
# Compile of ddr_dut.sv was successful.
# Compile of ddr_env.sv was successful.
# Compile of ddr_gen.sv was successful.
# Compile of ddr_intf.sv was successful.
# Compile of ddr_monitor.sv was successful.
# Compile of ddr_score.sv was successful.
# Compile of ddr_sdram.v was successful.
# Compile of ddr_sdram_tb.v was successful.
# Compile of ddr_tb.sv was successful.
# Compile of ddr_top.sv was successful.
# Compile of mt46v4m16.v was successful.
# Compile of params.v was successful.
# Compile of pll1.v was successful.
# 21 compiles, 0 failed with no errors.
```

Figure 8.19: Compilation of SystemVerilog code for verification

Figure 8.19 shows the compilation of both design and verification codes, and check for the simulation results, the main purpose of verification is to ensure that all input patterns are covered or not, that can be assured by using cover bins that will select the input combination pattern.
In order to check for the particular input pattern is working or not, we use cover bins. For example, the data bits are of 128 bits, checking for all the patterns of 2^{128} combination is difficult in Verilog, Hence SystemVerilog can be used as verification language. To check with help of using advance features such as OOPs, randomization, assertions, coverage, cover bins. To verify the coverage property.

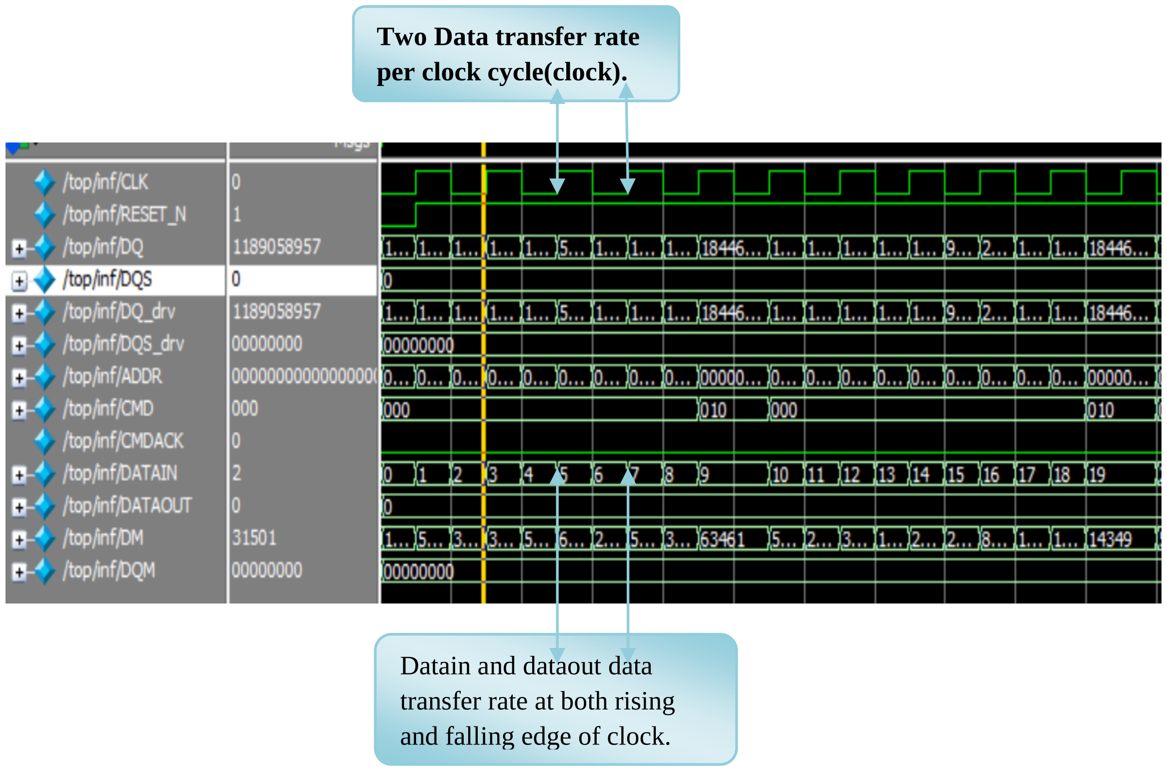
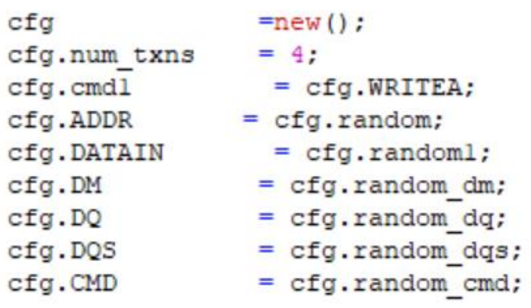

Two Data transfer rate per clock cycle(clock).
/top/inf/CLK 0
/top/inf/RESET_N 1
/top/inf/DQ 1189058957
/top/inf/DQS 0
/top/inf/DQ_drv 1189058957
/top/inf/DQS_drv 00000000
/top/inf/ADDR 0000000000000000
/top/inf/CMD 000
/top/inf/CMDACK 0
/top/inf/DATAIN 2
/top/inf/DATAOUT 0
/top/inf/DM 31501
/top/inf/DQM 00000000
Datain and dataout data transfer rate at both rising and falling edge of clock.

Figure 8.20: DDR SDRAM memory controller verification data transfer operation.

Figure 8.20 shows DDR SDRAM operate at double data rate, means transfer data at both pos-edge and neg-edge of clock cycles.
Here the DDR SDRAM memory controller was designed and tested each operation of READ, WRITE, auto-precharge, precharge, no-operation, lode_register, load_register1 and load_register2. Operation initialize with clock frequency set to 100MHz. and reset signal is made high. after providing some nanoseconds of delay the memory controller gets initialized.

then required data is stored to memory by writing data to particular memory locations. And read the same data values from memory locations.

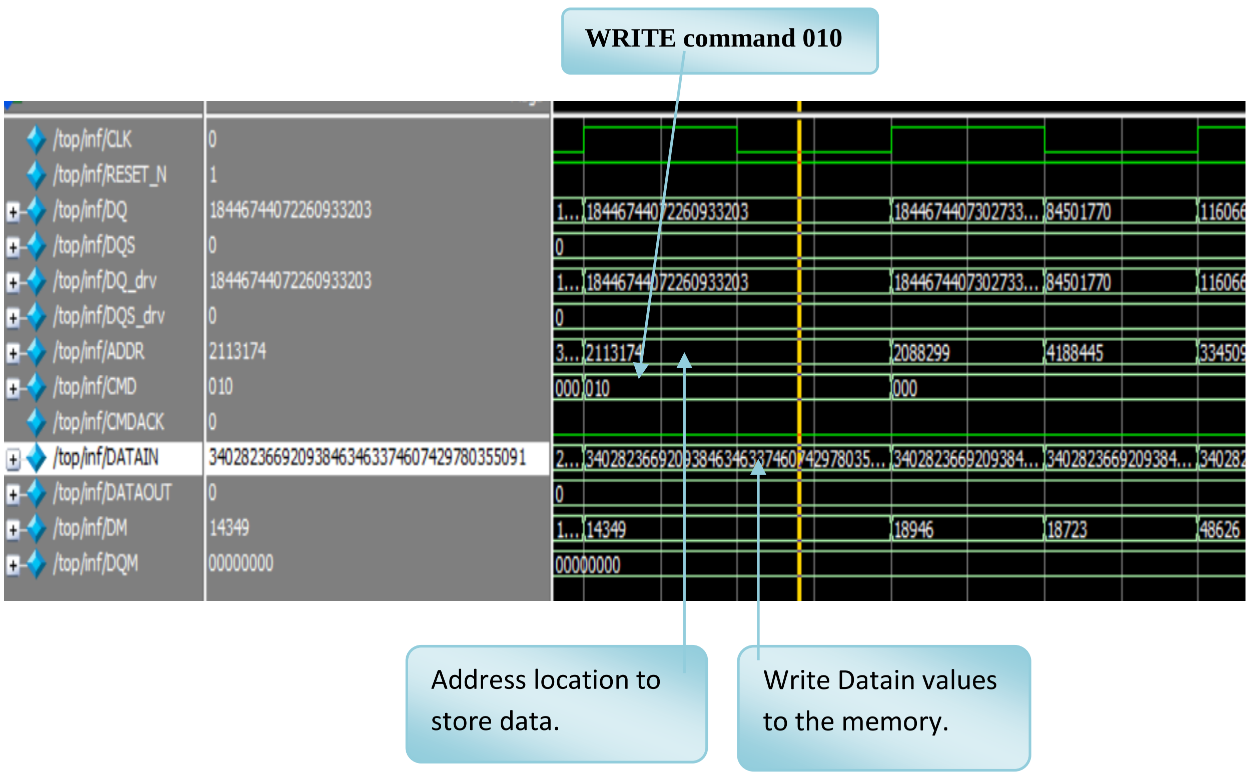
8.2.1 DDR SDRAM WRITE operation



```
cfg                =new();
cfg.num_txns       = 4;
cfg.cmd1             = cfg.WRITEA;
cfg.ADDR          = cfg.random;
cfg.DATAIN           = cfg.random1;
cfg.DM             = cfg.random_dm;
cfg.DQ             = cfg.random_dq;
cfg.DQS            = cfg.random_dqs;
cfg.CMD            = cfg.random_cmd;
```

Figure 8.21: Snapshot of SystemVerilog commands for READ operation.

Figure 8.21 shows WRITE operation, verification test bench for the DDR memory controller, where all address, datain, data mask and data strobe signals are taking the random input values.



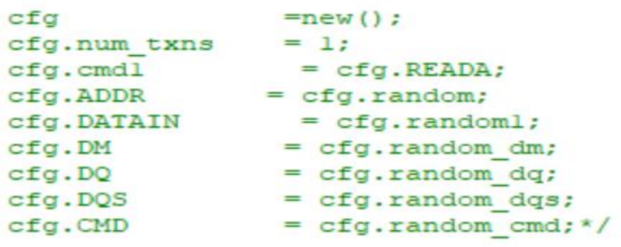
```
/top/inf/CLK        0
/top/inf/RESET_N    1
/top/inf/DQ         18446744072260933203     1...|18446744072260933203    |184467440730273...|84501770   |116066
/top/inf/DQS        0                        0
/top/inf/DQ_drv     18446744072260933203     1...|18446744072260933203    |184467440730273...|84501770   |116066
/top/inf/DQS_drv    0                        0
/top/inf/ADDR       2113174                  3...|2113174                 |2088299           |4188445    |334509
/top/inf/CMD        010                      000|010                      |000
/top/inf/CMDACK     0
/top/inf/DATAIN     340282366920938463463374607429780355091   2...|340282366920938463463374607429780355091742978035...|3402823669209384...|3402823669209384...|340282
/top/inf/DATAOUT    0                        0
/top/inf/DM         14349                    1...|14349                   |18946             |18723      |48626
/top/inf/DQM        00000000                 00000000
```

WRITE command 010
Address location to store data.
Write Datain values to the memory.

Figure 8.22: DDR SDRAM Memory controller writes operation.

Figure 8.22 shows the DDR memory controller verification write operation, starts with making the reset signal high, For write operation, select the bank and column to write, Make the CS (chip select), CAS (column access strobe), WE (write enable) signals to high, and RAS (row access strobe) to low, And issue write command 010 to command (cmd) signal, select the any one memory bank to perform write operation. Here address signal takes the random number 2113174(22-bit unsigned value) and datain signal takes the random number 340282366920938463346337460742978035509l(128-bit unsigned value) as shown in figures.

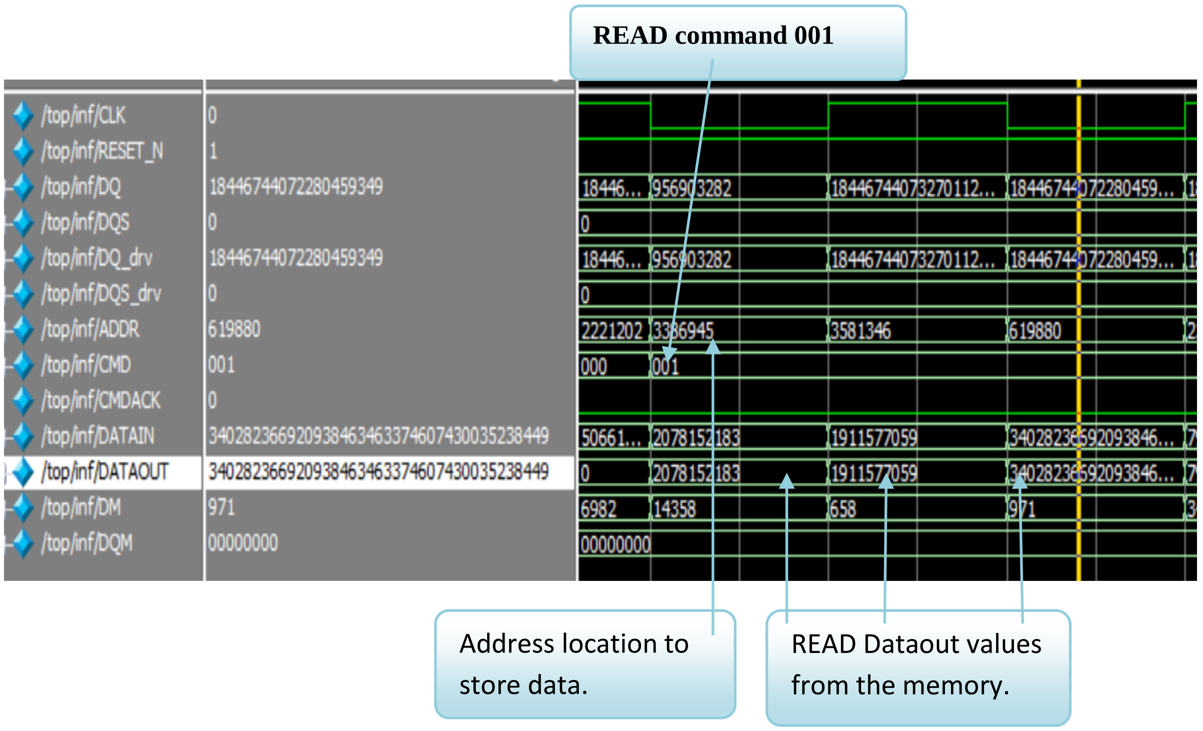
8.2.2 DDR SDRAM READ operation



```
cfg                 =new();
cfg.num_txns        = 1;
cfg.cmd1              = cfg.READA;
cfg.ADDR           = cfg.random;
cfg.DATAIN            = cfg.random1;
cfg.DM              = cfg.random_dm;
cfg.DQ              = cfg.random_dq;
cfg.DQS             = cfg.random_dqs;
cfg.CMD             = cfg.random_cmd;*/
```

Figure 8.23: Snapshot of SystemVerilog commands for READ operation.

Figure 8.23 shows READ operation, verification test bench for the DDR memory controller, where all address, datain, data mask and data strobe signals are taking the random input values.



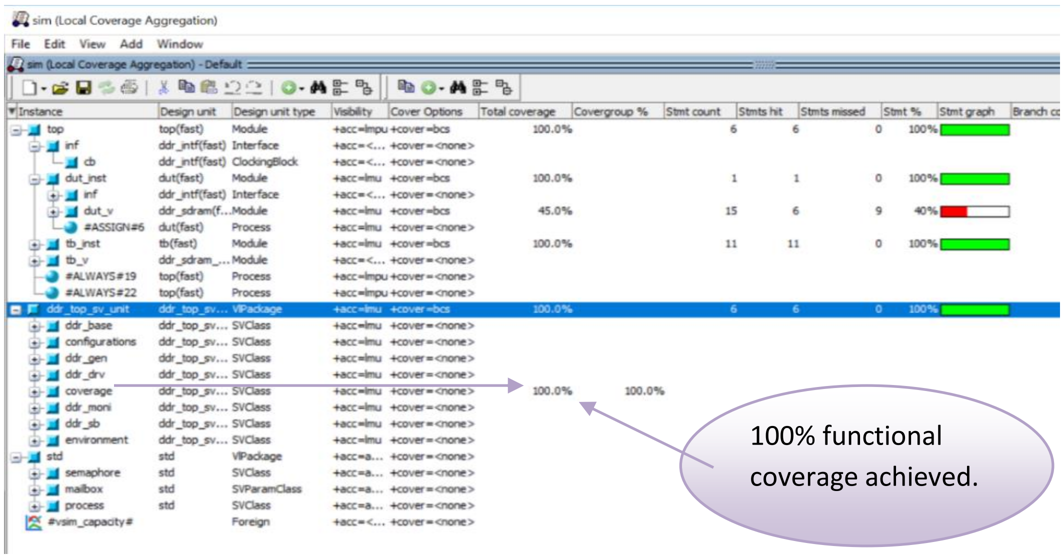
```
/top/inf/CLK       0
/top/inf/RESET_N   1
/top/inf/DQ        18446744072280459349                     18446...  956903282   18446744073270112...  18446744072280459...
/top/inf/DQS       0                                        0
/top/inf/DQ_drv    18446744072280459349                     18446...  956903282   18446744073270112...  18446744072280459...
/top/inf/DQS_drv   0                                        0
/top/inf/ADDR      619880                                   2221202   3386945     3581346               619880
/top/inf/CMD       001                                      000       001
/top/inf/CMDACK    0
/top/inf/DATAIN    340282366920938463463374607430035238449  50661...  2078152183  1911577059            340282366920093846...
/top/inf/DATAOUT   340282366920938463463374607430035238449  0         2078152183  1911577059            340282366920093846...
/top/inf/DM        971                                      6982      14358       658                   971
/top/inf/DQM       00000000                                 00000000
```

READ command 001
Address location to store data.
READ Dataout values from the memory.

Figure 8.24: DDR SDRAM Memory controller READ operation

Figure 8.24 shows the DDR memory controller verification read operation. initially reset signal is made high, select the bank and column to read, Make the CS (chip select), CAS (column access strobe), signals to high, and RAS (row access strobe) and WE (write enable) to low, And issue command 001 to command (cmd) signal, select the particular bank to perform read operations, Here address signal takes the random number 619880 (22 bit unsigned value) and datain signal takes random number 340282366920938463463374607300035238449 (128 bit unsigned value) and the same datain values is readout through dataout pin as shown in figure.

8.2.3 Functional Coverage report:


sim (Local Coverage Aggregation)
File Edit View Add Window
sim (Local Coverage Aggregation) - Default
100% functional coverage achieved.

Instance	Design unit	Design unit type	Visibility	Cover Options	Total coverage	Covergroup %	Stmt count	Stmts hit	Stmts missed	Stmt %	Stmt graph	Branch co
top	top(fast)	Module	+acc=lmpu	+cover=bcs	100.0%		6	6	0	100%	🟩	
inf	ddr_intf(fast)	Interface	+acc=<...	+cover=<none>								
cb	ddr_intf(fast)	ClockingBlock	+acc=<...	+cover=<none>								
dut_inst	dut(fast)	Module	+acc=lmu	+cover=bcs	100.0%		1	1	0	100%	🟩	
inf	ddr_intf(fast)	Interface	+acc=<...	+cover=<none>								
dut_v	ddr_sdram(f...	Module	+acc=lmu	+cover=bcs	45.0%		15	6	9	40%	🟥	
#ASSIGN#6	dut(fast)	Process	+acc=lmu	+cover=<none>								
tb_inst	tb(fast)	Module	+acc=lmu	+cover=bcs	100.0%		11	11	0	100%	🟩	
tb_v	ddr_sdram_...	Module	+acc=<...	+cover=<none>								
#ALWAYS#19	top(fast)	Process	+acc=lmpu	+cover=<none>								
#ALWAYS#22	top(fast)	Process	+acc=lmpu	+cover=<none>								
ddr_top_sv_unit	ddr_top_sv...	VlPackage	+acc=lmu	+cover=bcs	100.0%		6	6	0	100%	🟩	
ddr_base	ddr_top_sv...	SVClass	+acc=lmu	+cover=<none>								
configurations	ddr_top_sv...	SVClass	+acc=lmu	+cover=<none>								
ddr_gen	ddr_top_sv...	SVClass	+acc=lmu	+cover=<none>								
ddr_drv	ddr_top_sv...	SVClass	+acc=lmu	+cover=<none>								
coverage	ddr_top_sv...	SVClass	+acc=lmu	+cover=<none>	100.0%	100.0%						
ddr_moni	ddr_top_sv...	SVClass	+acc=lmu	+cover=<none>								
ddr_sb	ddr_top_sv...	SVClass	+acc=lmu	+cover=<none>								
environment	ddr_top_sv...	SVClass	+acc=lmu	+cover=<none>								
std	std	VlPackage	+acc=a...	+cover=<none>								
semaphore	std	SVClass	+acc=a...	+cover=<none>								
mailbox	std	SVParamClass	+acc=a...	+cover=<none>								
process	std	SVClass	+acc=a...	+cover=<none>								
#vsim_capacity#		Foreign	+acc=<...	+cover=<none>								

Figure 8.25: Functional coverage report of DDR SDRAM.

Figure 8.25 shows DDR SDRAM functional coverage report, generated from coverage program using cover bins to cover all combination input patterns. 100% functional coverage is achieved.

CHAPTER 9

CONCLUSION AND FUTURE SCOPE

The DDR SDRAM of 512 Mbits is designed using Verilog HDL. The Design is verified using Modelsim test bench and several test cases are written to verify the functionality of the design. Coverage functionality testing can be accomplished by using cover bins, which checks all input pattern combinations and corner cases combinations to be covered randomly and send the input values to the memory banks and also receive the same values from the memory bank with read and write commands. The proposed design is tested for all the test cases randomly to achieve 100% coverage using SystemVerilog.

Future scopes are, the design for DDR, the design can take for DDR2, can able to implement the environment for verification of DDR2, DDR3 and DDR4. The main difference between DDR and other memory controllers is the memory size are changes and blocks are changes, where memory size changes makes verification design also changes and this configuration only for DDR, further want to configure for DDR2, DDR3 and DDR4.