32-Bit Memory System design: Design of Memory Controller for Micron SDR SDRAM

By

CHIN CHUN LEK

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION TECHNOLOGY (HONS)

COMPUTER ENGINEERING

Faculty of Information and Communication Technology

(Perak Campus)

JAN 2015

DECLARATION OF ORIGINALITY

I declare that this report entitled “**Design of Memory Controller for Micron SDR SDRAM**” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature : ___________________________

Name : Chin Chun Lek

Date : 6 APRIL 2015

ACKNOWLEDGEMENTS

First and foremost I would like to take this opportunity to express my gratitude to my final year project supervisor, Mr. Mok Kai Ming, for his guidance and wisdom during the entire course of this project. I would also like to thank my friends, Chang Boon Chiao, Goh Dih Jian and Arthur for providing me supports. Lastly, I would like to thank my parents for the moral, emotional and financial support they have been providing.

By Chin Chun Lek

ABSTRACTS

This project focuses on the design of SDRAM Controller that is compatible with Micron SDR SDRAM MT48LC4M32B2 (1 Meg x 32 x 4 banks). After reviewing the previous work, the SDRAM controller is working but there are some differences with the conventional design that makes it to become complicated. This topic will be further discussed in the Literature Review and Design Methodology.

Currently, the interface of SDRAM controller connects to the host is not fully determined. The bus interface within the controllers is required to redesign in order to enable the caches to access the main memory. Therefore, this project is aiming to provide verification to the integration between the SDRAM controller and the cache controller.

Table of Contents

List of Tables

LIST OF TABLES

Table Number **Title** **Page**

List of Figures

LIST OF FIGURES

Figure Number **Title** **Page**

List of Abbreviations

ASIC	Application-Specific Integrated Circuit
CPU	Central Processing Unit
DRAM	Dynamic Random-Access Memory
FSM	Finite State Machine
HDL	Hardware Description Language
ISA	Instruction Set Architecture
Inc	Incorporated
I/O	Input /Output
MIPS	Microprocessor without Interlocked Pipeline Stages
MMU	Memory Management Unit
PSP	Sony Playstation Portable
RISC	Reduced Instruction Set Computer
RTL	Register Transfer Level
SoC	System-on-Chip
SDRAM	Synchronous Dynamic Random-Access Memory
SRAM	Static Random-Access Memory
TLB	Translation Lookaside Buffer
VCS	Verilog Compiled code Simulator
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

Chapter 1: Introduction

1.1: Background

With the widening gap between processor and memory speeds, system performance has become gradually more reliant upon the efficient use of memory hierarchy [1]. Many computations executed on current machine are often than not limited by the response of the memory system rather than the speed of the processor [2]. The introduction of high speed cache into the memory hierarchy is to bridge this speed gap. However, this introduction is not perfectly without flaw. By organizing memory system into hierarchy, it also indicate more complex analysis have to be done on the performance of the memory system. Nevertheless, since the benefit brought forward by implementing hierarchical ordering in memory design outshone its flaws [3-4], it is unavoidable to use this method in our memory system which is recently compiled and interfaced using Verilog [5]. Therefore, our project will be focused on the design and the implementation of a 32-Bit Memory System in particular the integration of caches, cache controllers, Translation Lookaside Buffer (TLB), Memory Management Unit (MMU), SDRAM and SDRAM controllers, and the verification for the memory system integrated to a Reduced Instruction Set Computers 32-bit (RISC32) processor. RISC32 is a 32-bit processor which is compatible to the MIPS ISA compatible. It runs a subset of MIPS instructions set, which uses small and highly-optimized set of instructions.

1.1.2 MIPS – a RISC processor

MIPS (Microprocessor without Interlocked Pipelined Stage) is a RISC (Reduced Instruction Set Computers) processor which use hardware implementation to directly execute instructions, without microprogrammed control. MIPS is widely used in digital consumer, home networking, personal entertainment, communications and business applications, such as Sony Playstation Portable (PSP), Smart Tab 1 (Karbonn Mobiles) and *Linksys wireless router which primarily used in MIPS implementations. MIPS can be develop using Verilog* – a hardware description language (HDL).

1.2: Motivation

The motivations to initiate the project are due to the following limitation:

- Microchip design companies develop microprocessors cores as Intellectual Property or IP for commercial purposes. The microprocessor IP includes information on the entire design process for the front-end (modeling and verification) and back-end (physical design) integrated circuit (IC) design. These are trade secrets of a company and certainly not made available in the market at an affordable price for research purposes.
- The microprocessor cores that are freely available from source such as the miniMIPS (www.opencores.org), the PH processor (Leicester University), uCore (www.opencores.org), Yellow Star (Manchester University), etc are incomplete in documentation and therefore do not provide good support for reuse. It is difficult to modify and extent the design for a specific applications under research. Apart from that, the cores are not well modeled and developed.
- The verification specification for a freely available RISC microprocessor core that is available on the Internet is not well developed and complete. Therefore, without a good verification specification, the verification process will be slow and hence, will slow down the overall design process.
- Since the freely available microprocessor cores and the verification are not well developed, this has affected the physical design phase. The physical design of the microprocessor cores is not well developed and complete.

The RISC32 project will look into the above problems, to create a 32-bit RISC core-based development environment to assist research work in the area of application specific hardware modeling. The RISC32 processor is a MIP-compatible ISA processor. In the RISC32 project, it is divided into several units based on the MIPS architecture. Up to date, a basic central processing unit (CPU) has been modeled at Register Transfer Level (RTL) using Verilog HDL (VHDL) and verified using a bus functional model. During the verification process, a high -level memory system unit model was developed and temporarily used. So currently, an RTL memory system unit model is not available.

1.2.1: Problem Statement

At present, a basic central 32-bit memory system that has been modeled at RTL using VHDL is the SDRAM controller design that compatible with Micron SDR SDRAM MT48LC4M32B2. However, the protocol controller block of SDRAM controller design is rather complicated and need to be resolved. Another problem has been encountered is the SDRAM controller can currently support a single cache, but typically RISC32 processor design has separated caches. Those caches are i-cache, d-cache, i-TLB, d-TLB, which will need to access to the SDRAM. This implies the limitation of the SDRAM controller interfaces and its redesigning is needed. Hence, the design of memory arbiter is also required to allow the shared bus for multiple caches.

Chapter 2: Literature Review

2.1: Memory Hierarchy

Computer memory is implemented with hierarchy (memory hierarchy) to take the advantage of principle of locality. There are three primary technologies used in building memory hierarchies. Main memory is implemented from DRAM, levels closer to processor (cache) use SRAM. The third technology is magnetic disk which is used to implement largest and slowest in the hierarchy. The price per bit and access time of these technologies vary widely. Therefore, we can take advantage by implementing memory hierarchy. Figure below shows the faster memory is close to the processor, while the slower memory is below it. This helps to present the user with more memory as is available in cheapest technology while it also provides the speed from the fast memory.

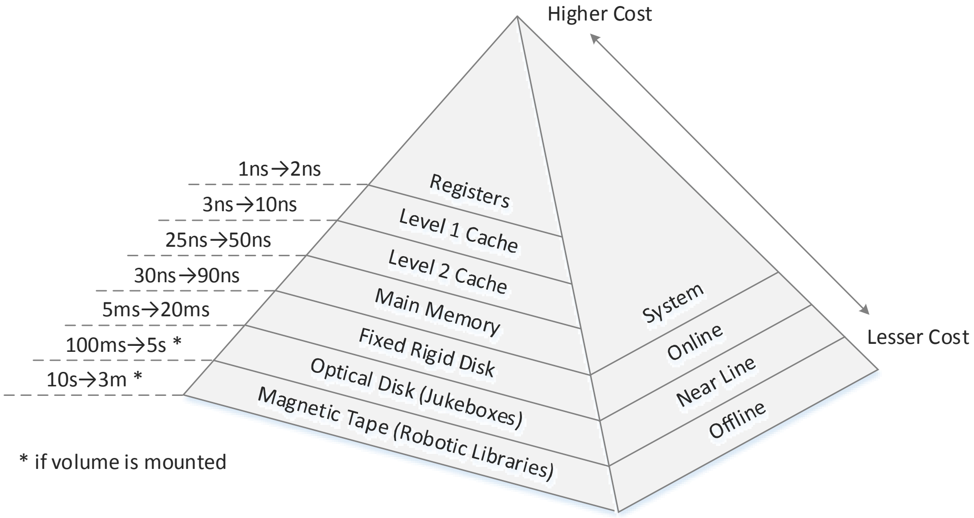


Figure 2.1: The Memory Hierarchy (Adapted from [4])

2.2: Processor and Main Memory Interfacing

The processor is connected to the main memory by a bus system [4] and the bandwidth of the bus system has a significant impact on miss penalty. This is due to the clock rate for the bus is always slower than the processor as much as a factor of 10. Therefore, the selection of memory organization to be use in processor is important in deciding the performance of the processor.

Figure 2.2 below shows three types of available memory organizations which are one-word-wide memory, wide memory and interleaved memory organization. If a cache block of four words and in a) one-wide memory organization, it only can fetch one word per time. That is the main memory have to access 4 times to fetch all data require from the cache. In b) wide memory organization allows the require data fetch with parallel access in a widening bandwidth of bus system between memory and the processor. If a cache block of four words and c) interleaved memory organization, it is capable to fetch four words to access the main memory at once.

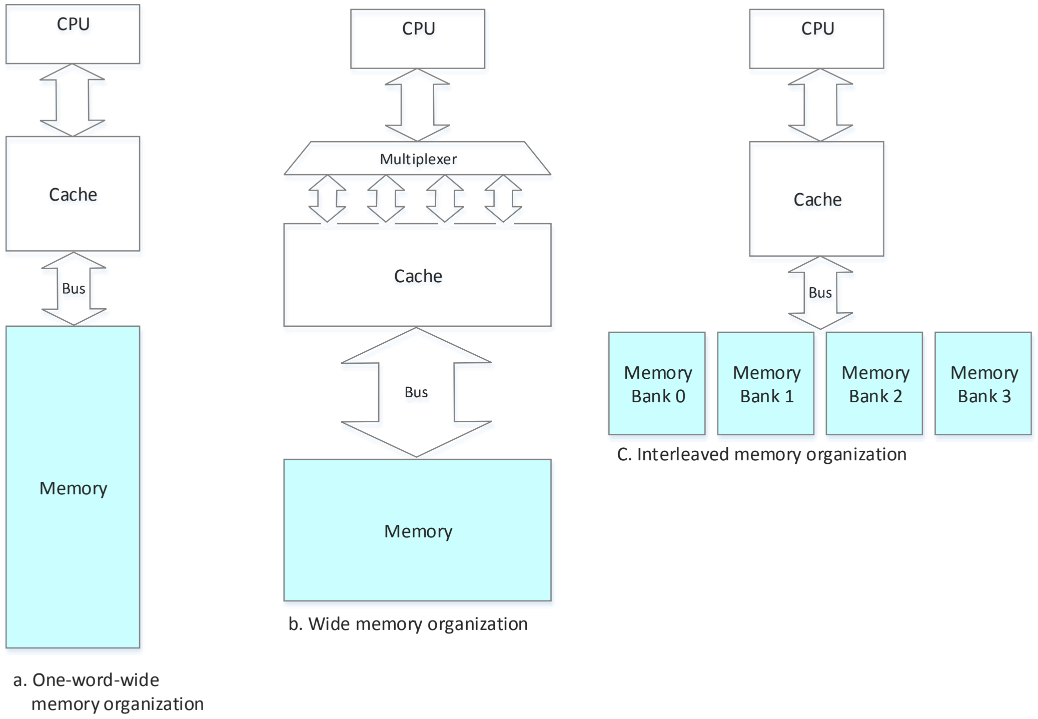


Figure 2.2: Memory Organization (Adapted from [4])

2.3: SDRAM Controller System Background

The overall figure of SDR SDRAM controller system is shown in the figure 2.3 below, which is describing a brief on how SDRAM controller can communicate with processor each other and interface with the SDRAM.

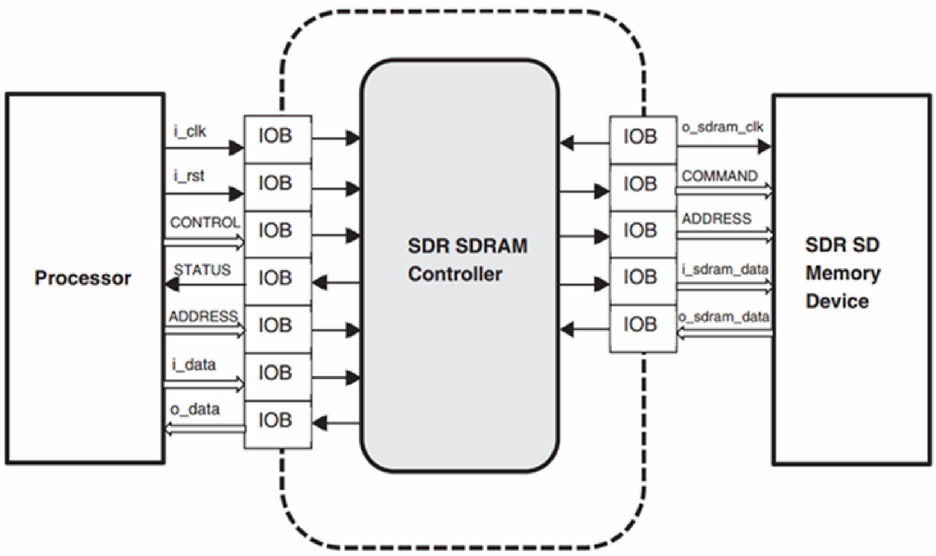


Figure 2.3: System block diagram (Adapted from [11])

2.4: SDRAM

Synchronous Dynamic Random Access Memory (SDRAM) is a type of DRAM that has a synchronous interface. There are two major types of SDRAM which can be distinguished by their data transfer rate. Single data rate (SDR) SDRAM transfers data on the rising edge of the clock, and double data rate (DDR) SDRAM transfers data on both rising and falling edge.

Figure 2.4 shows the pins for a conventional 1M x 32-bit x 4 banks SDRAM which is referring to the Micron. Pin ba(1:0) is used to select the 4 internal memory banks within the SDRAM while adr(11:0) is used as an input to send column address, row address and configuration setting to the SDRAM. The SDRAM has adopted bidirectional data line, dq, for write transfer and read transfer. This is because the SDRAM can only do one of the operations at a time. The granularity of a bus is defined as the smallest transfer can be done by that bus. According to [12], the granularity of a SDRAM is 8-bit. This is accomplished using the data masking pin,dqm(3:0). The data masking pin is used to select which byte of the 32-bit bidirectional data line, dq, is valid.

For example, if dqm = 0001 (binary), the valid 8-bit data is located at dq(7:0). Here is another example, if dqm = 1100 (binary), the valid 16-bit data is located at dq(31:16). As mentioned, since the smallest transfer is 8-bit, the granularity of this SDRAM is 8-bit. As a comparison, the customized SDRAM [11] has a granularity of 32-bit for its 32-bit write data line and 256-bit granularity for its 256-bit read data line. This also means that the customized SDRAM cannot support byte addressing.

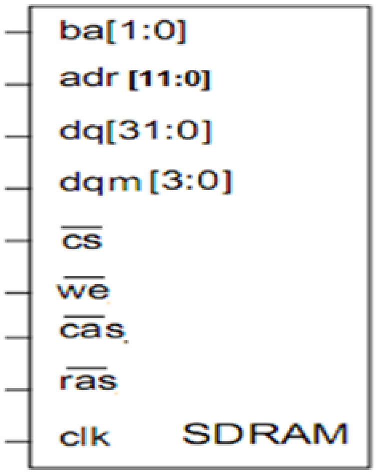
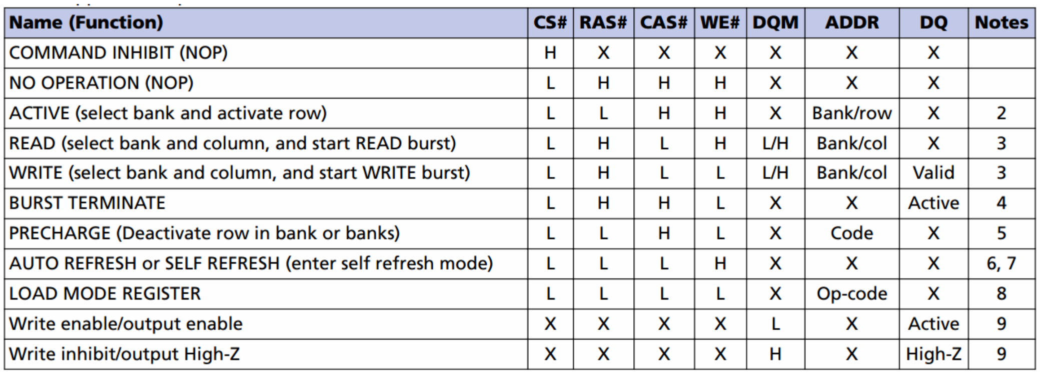


Figure 2.4: 128Mb banks SDRAM Block diagram (Adapted from [10])

To select the SDRAM, the cs (active low) pin is used. Meanwhile active low command signals (we, cas and ras) are used to request operations from the SDRAM. The list of commands available in SDRAM is shown in Table 2.4.



Name (Function)	CS#	RAS#	CAS#	WE#	DQM	ADDR	DQ	Notes
COMMAND INHIBIT (NOP)	H	X	X	X	X	X	X	
NO OPERATION (NOP)	L	H	H	H	X	X	X	
ACTIVE (select bank and activate row)	L	L	H	H	X	Bank/row	X	2
READ (select bank and column, and start READ burst)	L	H	L	H	L/H	Bank/col	X	3
WRITE (select bank and column, and start WRITE burst)	L	H	L	L	L/H	Bank/col	Valid	3
BURST TERMINATE	L	H	H	L	X	X	Active	4
PRECHARGE (Deactivate row in bank or banks)	L	L	H	L	X	Code	X	5
AUTO REFRESH or SELF REFRESH (enter self refresh mode)	L	L	L	H	X	X	X	6, 7
LOAD MODE REGISTER	L	L	L	L	X	Op-code	X	8
Write enable/output enable	X	X	X	X	L	X	Active	9
Write inhibit/output High-Z	X	X	X	X	H	X	High-Z	9

Table 2.4: Truth Table - Command and DQM operation (Adapted from [14])

2.5: SDRAM controller

The SDRAM Controller is located between SDRAM and the host, provide proper commands for SDRAM initialization, read/write accesses and memory refresh. The host can be either a microprocessor or a user's proprietary module interface. The SDRAM Controller has been previously modele\d based on industry standard WISHBONE SoC interface [10].

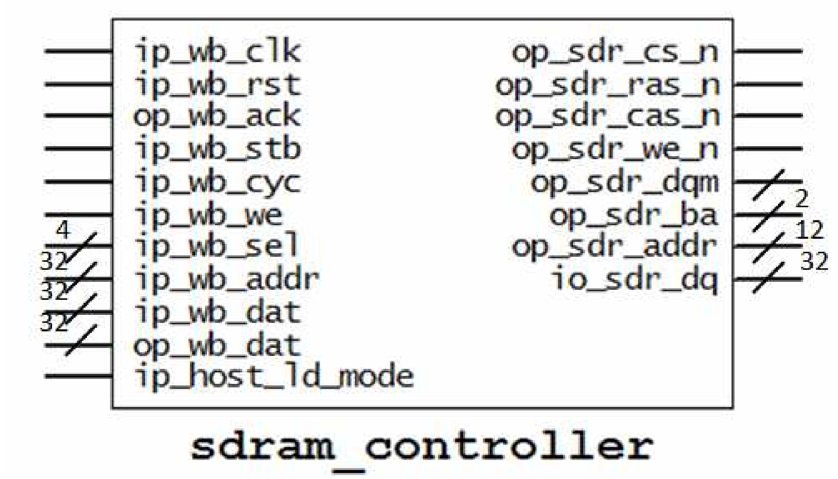


Figure 2.5: SDRAM Controller Block Diagram (Adapted from [10])

Pin name: ip_wb_clk **Path:** Memory Bus Clock -> SDRAM Controller **Description:** Wishbone Clock Input
Pin name: ip_wb_rst **Path:** System Reset -> SDRAM Controller **Description:** Wishbone Synchronous reset
Pin name: ip_wb_cyc **Path:** Host -> SDRAM Controller **Description:** When asserted, this pin indicates that a valid bus cycle is in progress.
Pin name: ip_wb_stb **Path:** Host -> SDRAM Controller **Description:** When asserted, this pin indicates that the SDRAM controller is selected.
Pin name: ip_wb_we **Path:** Host -> SDRAM Controller **Description:** When asserted, this pin indicates that the current cycle is READ. When deasserted, it indicates WRITE.
Pin name: op_wb_ack **Path:** SDRAM Controller -> Host **Description:** When asserted, it indicates that the current READ or WRITE is successful.
Pin name: ip_wb_sel **Path:** Host -> SDRAM Controller **Description:** This signal indicates where valid data is placed on the input data line

(ip_wb_dat) during WRITE cycle and where it should present on the output data line (op_wb_dat) during READ cycle. The array boundaries are determined by the granularity of a port. In this SDRAM controller, 8-bits granularity is used and all the data ports are 32-bits. Therefore, there would be 4 select signals with the boundaries of ip_wb_sel(3:0). Each individual select signal correlates to one of 4 active bytes on the 32-bits data port.

Pin name: ip_wb_addr
Path: Host -> SDRAM Controller
Description: The address input is used to pass the memory address from the host.

Pin name: ip_wb_dat
Path: Host-> SDRAM Controller
Description: This pin is used to pass WRITE data from the host.

Pin name: op_wb_dat
Path: SDRAM Controller -> Host
Description: This pin is used to output READ data from the SDRAM.

Pin name: ip_host_ld_mode
Path: SDRAM Controller -> Host
Description: This pin is asserted to load a new mode into the SDRAM.

Pin name: op_sdr_cs_n
Path: Host -> SDRAM
Description: SDRAM chip select

Pin name: op_sdr_ras_n
Path: Host -> SDRAM
Description: SDRAM row address select

Pin name: op_sdr_cas_n
Path: Host -> SDRAM
Description: SDRAM column address select

Pin name: op_sdr_we_n
Path: Host -> SDRAM
Description: SDRAM write enable.

Pin name: op_sdr_addr
Path: Host -> SDRAM
Description: This pin is used as an address output to the SDRAM. The address will be segmented into row, column and bank before being sent out through this pin.

Pin name: op_sdr_ba
Path: Host -> SDRAM
Description: This pin is used to select the bank within the SDRAM. There are a total of 4 banks within the SDRAM and each of them operates independently.

Pin name: op_sdr_dqm
Path: Host -> SDRAM
Description: This pin is used to select which bits of the data line (io_sdr_dq) to be masked.

Pin name: io_sdr_dq
Path: Host -> SDRAM
Description: This data line is a bidirectional line to receive READ data or send WRITE data.

2.5.1: Read/Write Cycle Timing diagram

Figure 2.5.1 indicates the timing diagram for writing a burst of four data words to the SDRAM. The wb_dat indicates the command received from host is in the idle state at the begining. At T1, the system places Address on the bus continue until T3. After SDRAM detects ACTIVE command and row address at T2 and after RAS-to-CAS delay (tRCD), SDRAM receives the WRITE command and the first data comes in. The four words burst write is done at T8.

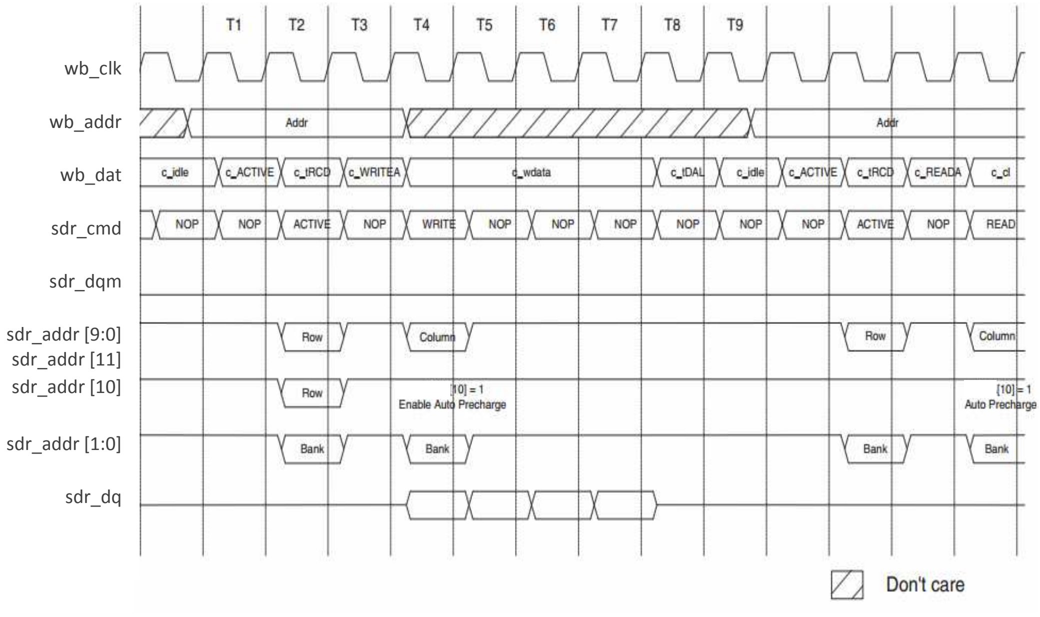


Figure 2.5.1: Write Timing Diagram

Figure 2.5.2 indicates the timing diagram for reading a burst of four data words to the SDRAM. At T1, the system places Address on the bus until T3. After SDRAM detects the ACTIVE command and row address at T2, and after RAS-to-CAS delay, SDRAM receives the READ command and the column address at T4. After CAS latency delay, the SDRAM starts to receive first data at T6. The four words burst read are completed in T9.

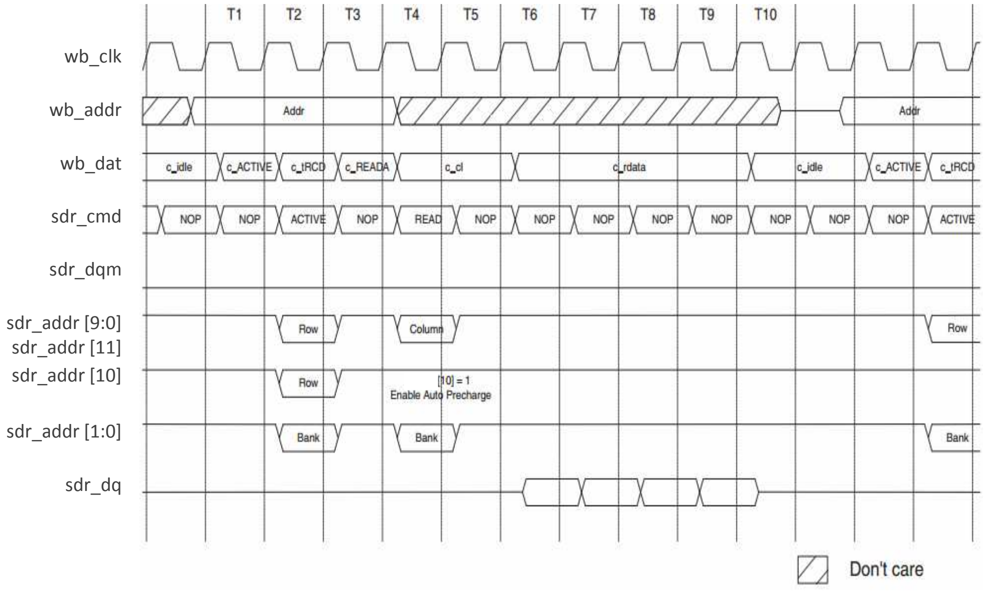


Figure 2.5.2: Read Timing Diagram

2.6: SDRAM Controller and Cache Controller Interfacing

The following figure 2.6 shows the interface of SDRAM controller to the cache unit.

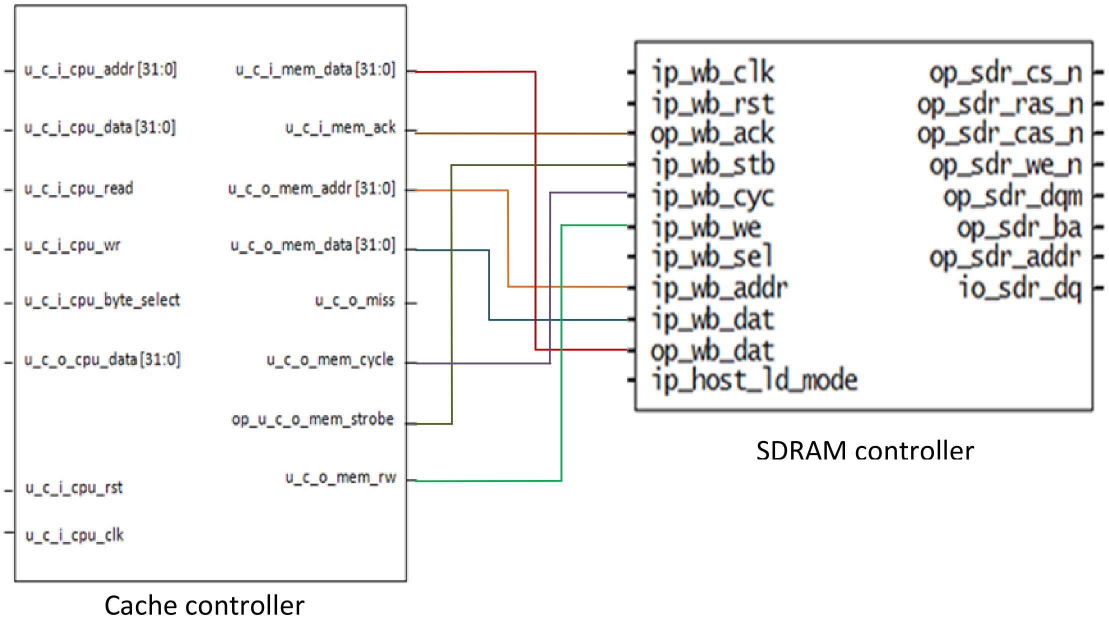


Figure 2.6: Connection between Cache controller and SDRAM controller

ip_host_ld_mode indicates as an enable pin to load new mode by passing write data from the host (ip_wb_dat). If the current load mode register (LMR) command is same with the previous mode, the register will retain the same configuration and not going to load any new mode to the SDRAM. But if both modes are differences, the ip_host_ld_mode will be asserted high to allow a new mode load to the SDRAM. This feature is required in order to reduce LMR time delayed whenever the same mode is appeared in the next stage.

2.7: Load Mode Register

The pins of the SDRAM adr[11:0]and command signals (cs, we, cas and ras) are used to configure the mode register which can define the specific mode of operation for SDRAM via the LOAD MODE REGISTER (LMR) command and the information stored will be retain until it has been reprogrammed or the device has been powered off. The definition includes the selection of burst length, burst type, CAS latency, operating mode and write burst mode. Burst indicates the technique used as continuous read or continuous write the data. An example of read operation with burst is when the burst length is set to be 4; the data will be read 4 times continuously. And the sequence of data will be read or write operation and either in a sequential or interleaved order. The figure 2.7 will show the data status to be configured.

The description of each Mode Register definition from figure 2.4.2 is listed as below:

Burst Length
To determine the maximum number of column locations that can be accessed for a given READ or WRITE command.

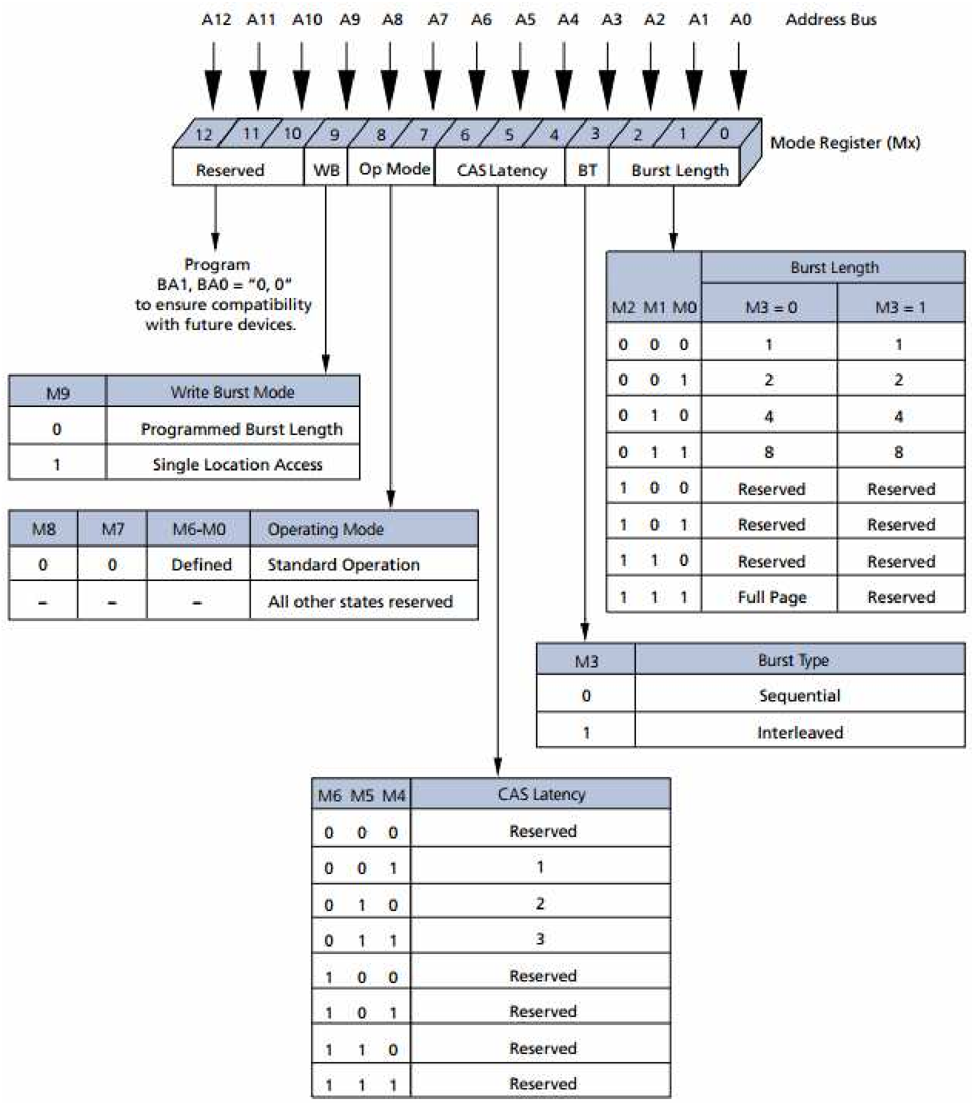
Burst Type
Access within a given burst can be programmed to be either sequential burst or interleaved burst to be adopted by SDRAM. The ordering of accesses within a burst is determined by burst length, burst type, and the starting column address.

CAS Latency
Delay in clock cycles between registration of a READ command and the availability of the first piece of output data. It can only be set to 2 or 3 clock cycles.

Operating Mode
To select the operating mode should be used in the SDRAM. Currently there is only normal operating mode is available for use.

Write Burst Mode

When the mode is asserted high, the burst length is programmed as READ burst or WRITE burst. If it is asserted low, the programmed burst length applies to READ burst, but WRITE access are single-location access (non-burst). The burst length that mentioned is referred to the M0-M2.



M2 M1 M0	Burst Length	
	M3 = 0	M3 = 1
0 0 0	1	1
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	Full Page	Reserved

M9	Write Burst Mode
0	Programmed Burst Length
1	Single Location Access

M8	M7	M6-M0	Operating Mode
0	0	Defined	Standard Operation
-	-	-	All other states reserved

M3	Burst Type
0	Sequential
1	Interleaved

M6 M5 M4	CAS Latency
0 0 0	Reserved
0 0 1	1
0 1 0	2
0 1 1	3
1 0 0	Reserved
1 0 1	Reserved
1 1 0	Reserved
1 1 1	Reserved

Figure 2.7: Mode Register Definition (Adapted from [14])

2.8: Memory Arbiter

The Figure 2.6 interface that shows in previously is merely an explanation on how SDRAM controller is connected with a cache. If there are independent requesting processor units connecting to the SDRAM, we required a memory arbiter to resolve the shared bus conflict. The memory arbiter allows one MASTER to access SDRAM controller at single time while the other MASTERs have to be waiting. It is given a pattern or ordering for each of the MASTER to access first. The shared bus usually uses a priority or a round robin arbiter. These grant the shared bus on a priority or equal basis. And a timeout is given to ensure that the bus does not remain locked at particular MASTER for duration greater than the time out period.

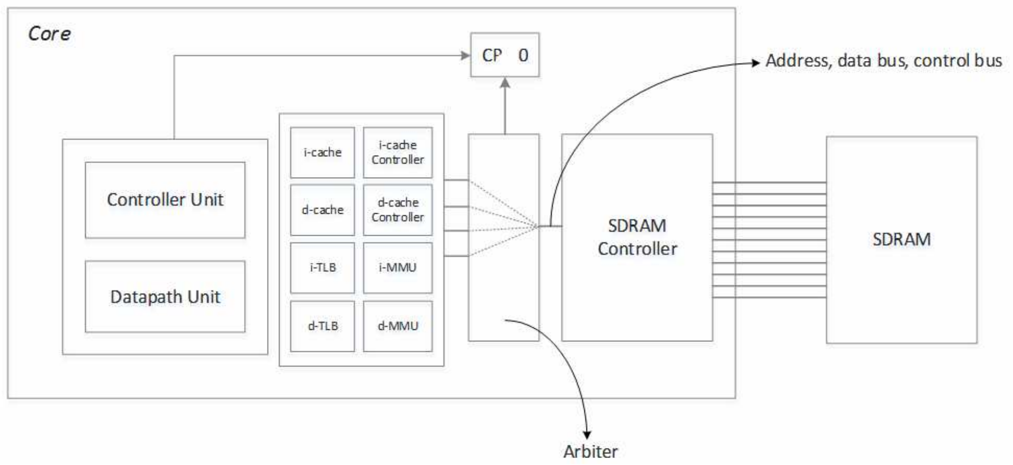


Figure 2.8.1: Micro-Architecture Level Design (Unit Level)

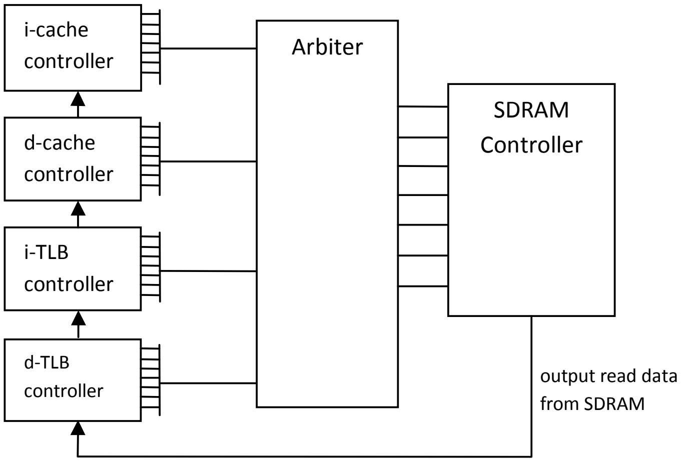


Figure 2.8.2: Interface of Memory Arbiter

2.9: Protocol Controller State Diagram

A 32-Bit Memory System of SDRAM controller was integrated by the previous work [10]. However there was an attempt to integrate this SDRAM controller, the design has its own readability issue.

The SDRAM controller was designed in the previous work has consequently leads to the difficulty of understanding how the design protocol works in SDRAM controller. The design has a combination of SDRAM initialization and SDRAM command in the finite state machine (FSM). Therefore this project is initiated to create a better and easier analyzing SDRAM controller. In the figure below shows the FSM of SDRAM protocol in previous work.

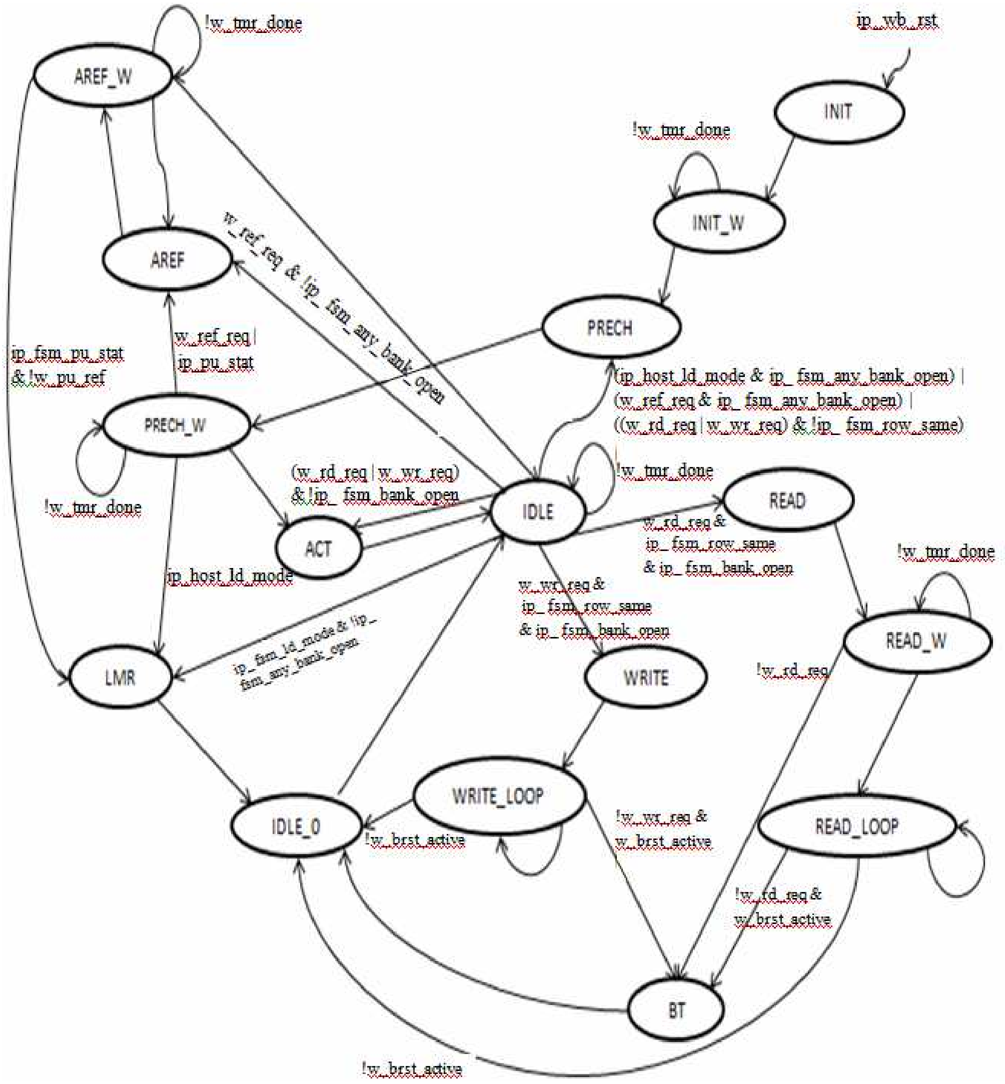


Figure 2.9: Initialization Protocol FSM (Adapted from [10])

Chapter 3: Project Scope and Objectives

3.1: Project Scope

This project is to redesign the existing interface of memory system and processor. A completed 32-bit memory system will be delivered. There are two parts of works required to be improved, which are the design of SDRAM controller compatible with Micron SDRAM and compatible with current memory system design.

3.2: Project Objectives

The project's objectives include:

- Analyze the 32-Bit Memory System organization for examining the scope of the integration done thus far. In addition, an appropriate test and testbench will be constructed to assist test analysis.
- Redesign the sub module of SDRAM controller Protocol Controller block Finite State Machine (PCB FSM) that compatible with Micron SDR SDRAM.
- Redesign the SDRAM controller to support multiple cached load mode configurations.
- Design of Memory Arbiter to allow the connection of differing caches to SDRAM controller.
- Verify the integration of the RISC32 processor and memory system by construct an appropriate test cases for direct test, integration test and random test.

3.3: Significance and Impact

As a synopsis to the problem statement, there is a lack of well-developed and well-founded 32-bit RISC microprocessor core-based development environment. The development environment refers to the availability of the following:

- A well-developed design document, which includes the chip specification, architecture specification and micro-architecture specification.
- A fully functional well-developed 32-bit RISC architecture core in the form of synthesis-ready RTL written in Verilog.

- A well-developed verification environment for the 32-bit RISC core. The verification specification should contain suitable verification methodology, verification techniques, test plans, testbench architectures etc.
- A complete physical design in FPGA with documented timing and resource usage information.

The project is an effort to develop the environment mentioned above: to be used as a multi-cycle pipelined RISC microprocessor core-based platform to support hardware modeling research work.

With the existing well-developed basic RTL model (which has been fully functionally verified), the verification environment and the design documents, a researcher can develop his research specific RTL model as part of the environment (whether directly modifying the internals of the processor or interface to the processor) and can quickly verify his model to obtain results, without having to worry about the development of the verification environment and the modeling environment. This can hasten the research work significantly. Relating exclusively to this project, the availability of a good methodology to help support memory system analysis makes it easier for any future improvement on the existing system.

Chapter 4: Methods/Technologies Involved

4.1: Design Methodology

Design Methodology basically refers to the method of development of a system. It provides us with a set of guidelines to successfully carry out the design work. A good design methodology needs to ensure the following [8]:

- Correct Functionality
- Satisfaction of performance and power goals
- Catching bugs early
- Good documentation

The ideal design flow for this project would be the top-down methodology as shown in figure 4.1:

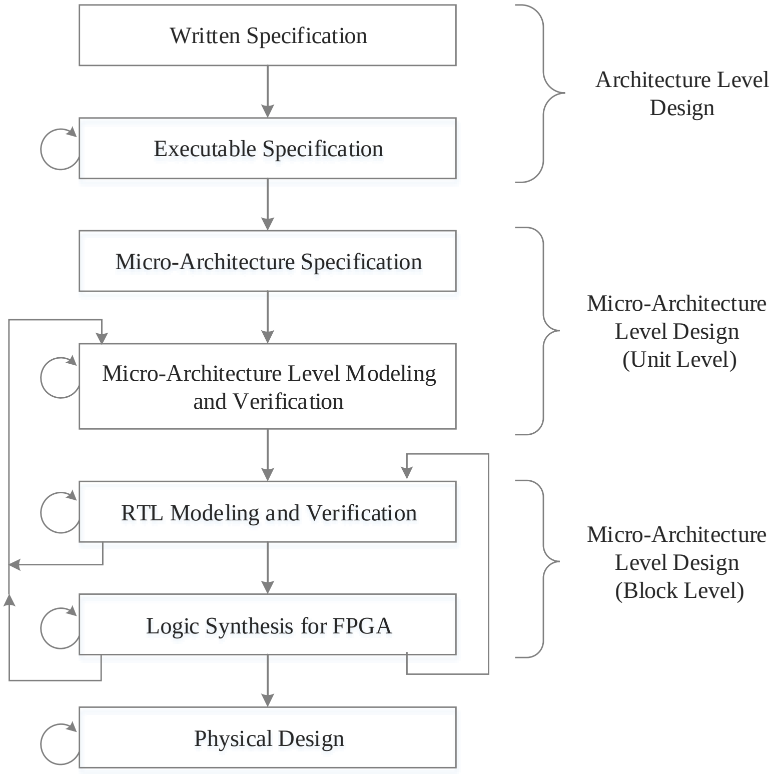

Written Specification
Executable Specification
Micro-Architecture Specification
Micro-Architecture Level Modeling and Verification
RTL Modeling and Verification
Logic Synthesis for FPGA
Physical Design
Architecture Level Design
Micro-Architecture Level Design (Unit Level)
Micro-Architecture Level Design (Block Level)

Figure 4.1: General Design Flow without Logic Synthesis and Physical Design.

Source: K.M. MOK [8]

4.1.1 Architecture Level Design

Architecture Level Design is level where chip specifications are being developed. The level design includes the following two types, *written specification* and *executable specification*, which carry (refer to Appendix A):

- functionality / features
- Operating procedures and application
- Naming convention
- Pipeline chip interface and I/O description
- Memory map
- System register
- Supported instruction set (machine language)
- Instruction formats
- Addressing modes

4.1.2 Micro-Architecture Level Design (Unit Level)

Micro-Architecture Level Design can categorize into 2 phases, *Micro-Architecture specification* and *Micro-Architecture Level Modeling and Verification***.** In the content of this level of design includes (refer to Appendix B):

- Design hierarchy
- Unit level functional partitioning (Datapath Unit, Instruction Fetch Unit, Control Unit, Instruction Memory Unit and Data Memory Unit)
- Worst case timing
- Full chip Verilog model
- Test plan
- Testbench

4.1.3 Micro-Architecture Level Design (Block Level)

In this level, RTL (Register Transfer Level) is developed. A micro-architecture specification of each unit, which used to describe the internal design of architecture block module. Micro-architecture specification may include information of:

- functionality / feature
- datapath unit interface and I/O pin description,

- internal operation, block / sub-block level functional partitioning (Register File Block, ALU Block, etc)
- Verilog model is later inserted
- Testbench and simulation result

After developed Micro-architecture Specification, RTL modeling with programming language can be start. Model can be simulate and verified with software. Verification includes development of test plan, timing verification and functionality verification. Hence designer can verify and modify the design to meet the chip specification.

4.2: Protocol Controller Block Design

Instead of the design protocol discussed in the previous work from [10], the FSM also can be separated into two by using one-hot encoding FSM, which shows in the following figure 4.2.1 and figure 4.2.2.

The INIT_FSM state machine from Figure 4.3.1 handles the SDRAM initialization. This initialization states begin with a NOP state, continued with PRECHARGE state, followed by AUTO REFRESH states, and then LOAD MODE REGISTER (LMR) states to configure SDRAM specific mode of operation. In each state consists of its delay time, and will be done by the timer. The auto refresh state use repeatedly [10] can be separated into two auto refresh to simplify the logic and state.

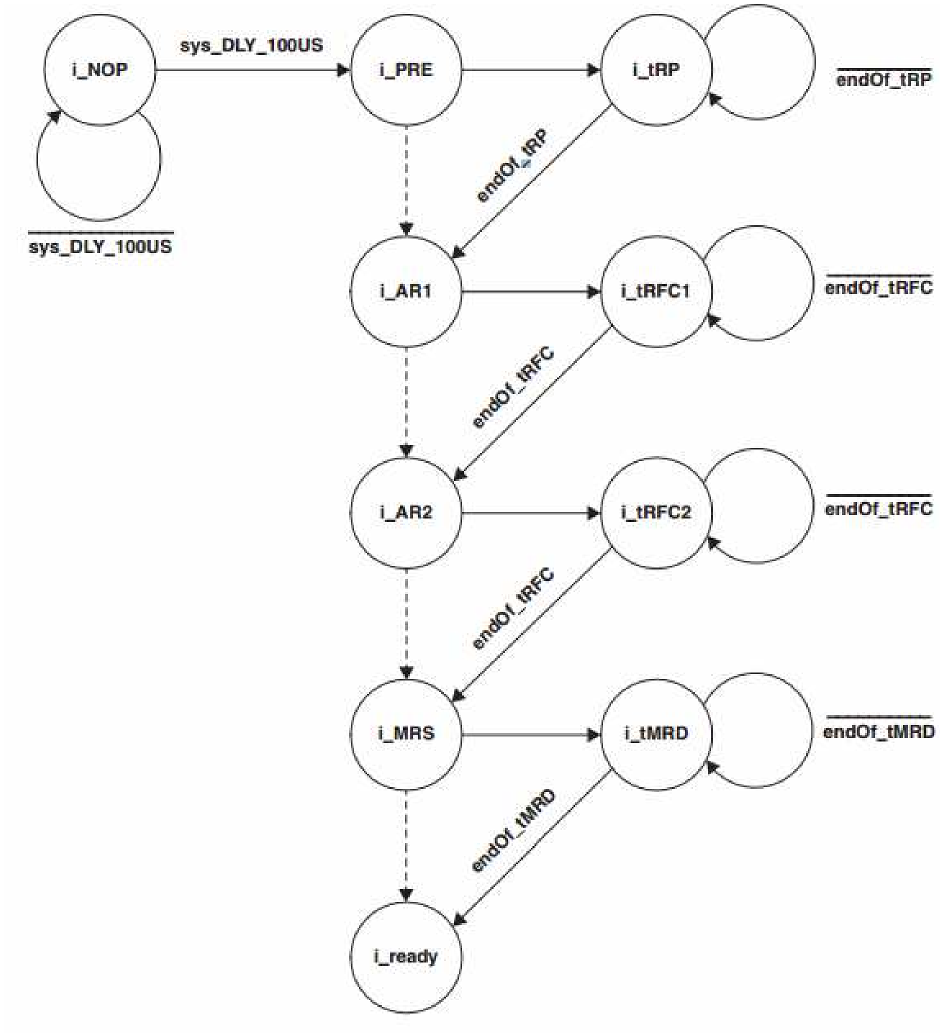


Figure 4.2.1: INIT_FSM (Adapted from [13])

The CMD_FSM state machine from Figure 4.2.2 handles commands such as read, write, and refresh of the SDRAM. The command FSM has its own auto refresh state, since the initialization and command FSM has been separated away. Other than that, the rest of the states are not much different with the previous work [10].

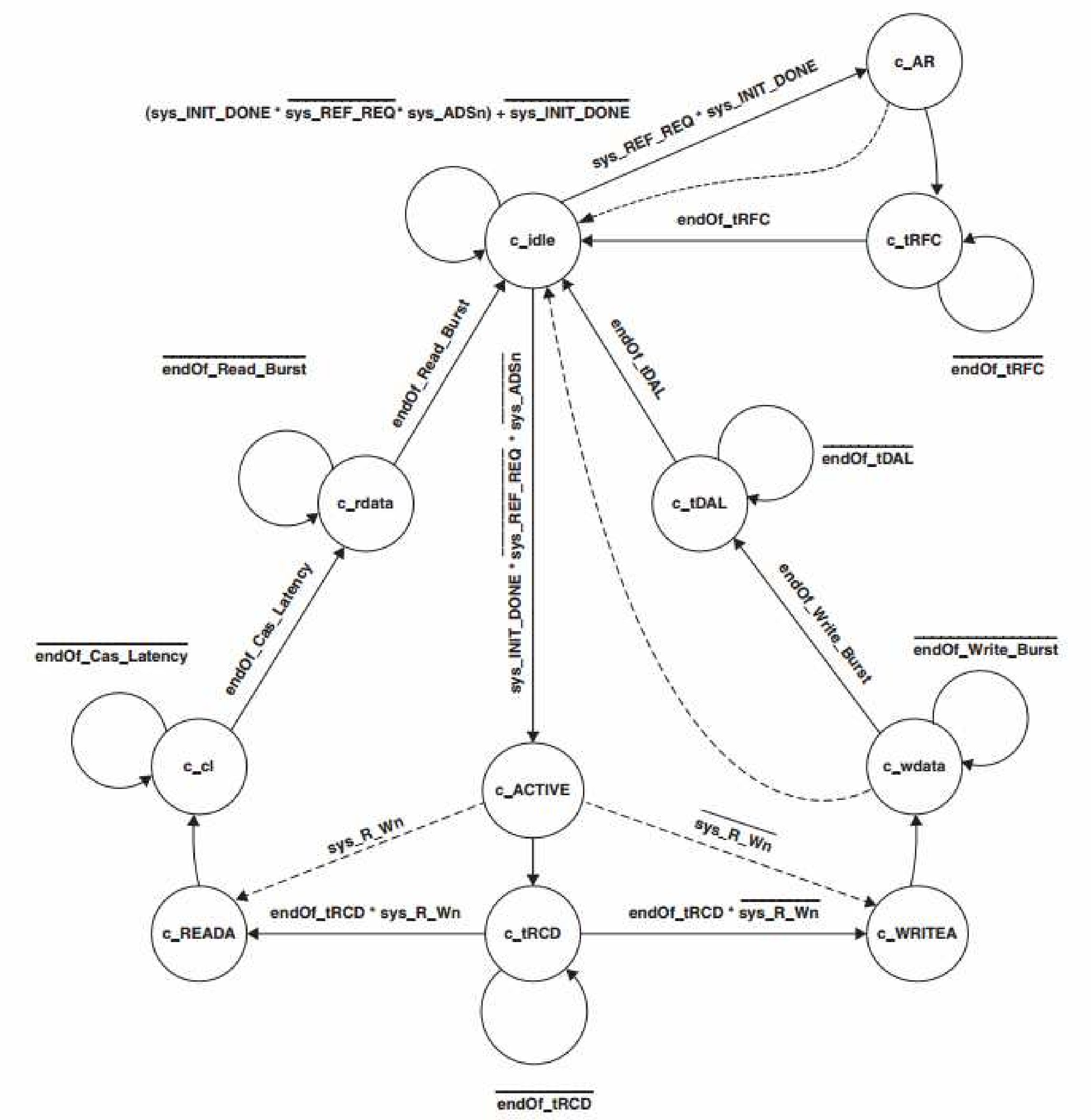


Figure 4.2.2: CMD_FSM (Adapted from [13])

The signal sys_DLY_100US from Figure 4.2.1 indicates the system clock delayed for 100 µs, which can be generated by the internal Phase-Locked Loop (PLL) by setting the proper PLL attributes (clock multiplication and division). An example of Clock divider with a 50% duty cycle can be generated as according to the following steps.

Firstly, the counts from N-1 to 0 count down counter must be created and always on the rising edge of input system clock. Secondly, toggle flip-flops TFFs are used and generate their enables. For an example the clock signal is divided by 3, TFF1 enable when count value is 1, TFF2 enable when count value is 2. Thirdly, the output of TFF1 (div1) triggered on rising edge of input clock whereas the output of TFF2 (div2) triggered on falling edge of input clock. Lastly, the final output signal is generated by the two clocks (div1 and div2) at half desired output frequency by undergoes XOR operation of the two waveforms together.

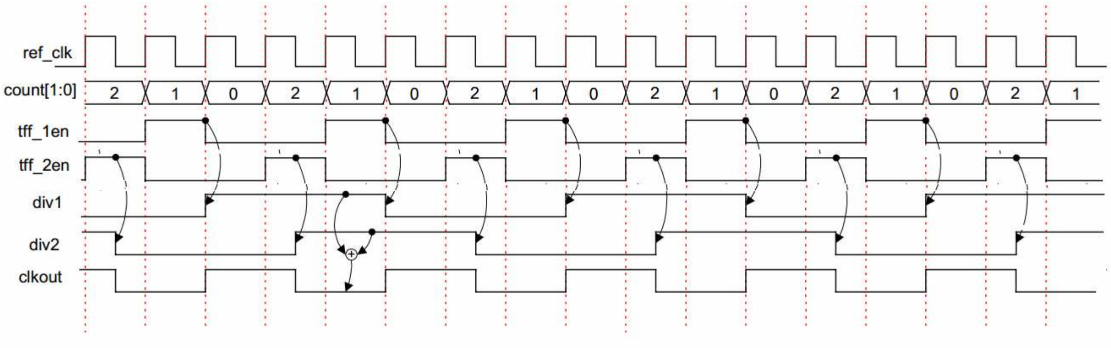


Figure 4.2.3: Timing diagram for Divide by 3 (N=2)

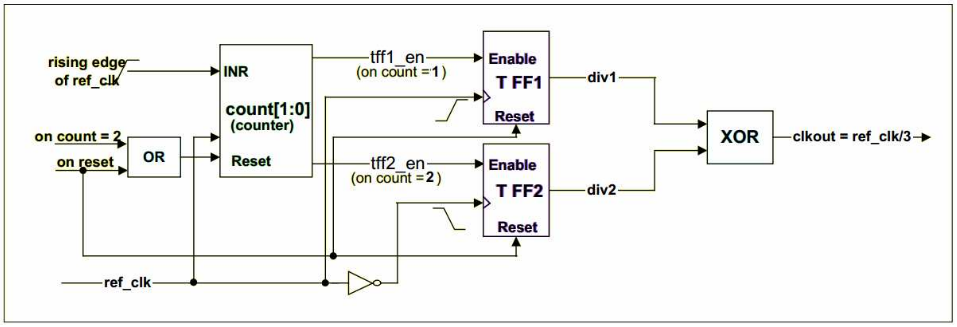


Figure 4.2.4: Divide by 3 using T Flip-flops

We can use the timer to create the exact delay time required for the SDRAM clock. To create the 50% duty cycle output clock delayed signal, we need to double up the input clock frequency use as referencing clock and perform the equation below:

$$count\ value = \frac{output\ clock\ frequency}{100\mu s\ delay\ time\ frequency}\ .$$

For an example, the system clock speed has 100 KHz (10μs per clock), thus count value will count down from 9 to 0. Each round of count, the output delayed clock signal will toggle its previous state, in order to obtain a half clock cycle of the output.

4.3: Load Mode Configuration with Multiple Cache

There is a problem that needs to resolve, which is to redesign the SDRAM controller so it can support Load Mode configuration and allow multiple caches to access. The ip_host_ld_mode is an enable pin to load new mode to the SDRAM. Caches need to share this pin. However, we can use one-hot method to separate the enable pin into individual pins and four individual load mode registers to store the configuration. For example if there are four caches, four ip_host_ld_mode enable pins are connected to the caches respectively, and each enable pin is controlling its own load mode register. The i-cache load mode enable pin will be controlling the i-cache load mode register. Thus there will be four registers need to be created. But this method will rather increasing the hardware complexity.

A more efficient way is using only one enable pin and the register just keep its previous configuration. To decide whether to load a new mode to the SDRAM, the SDRAM controller need to check out for the current data and the previous one is either same or not. From the figure below aids to architecture view of how multiple caches can be connected to SDRAM controller.

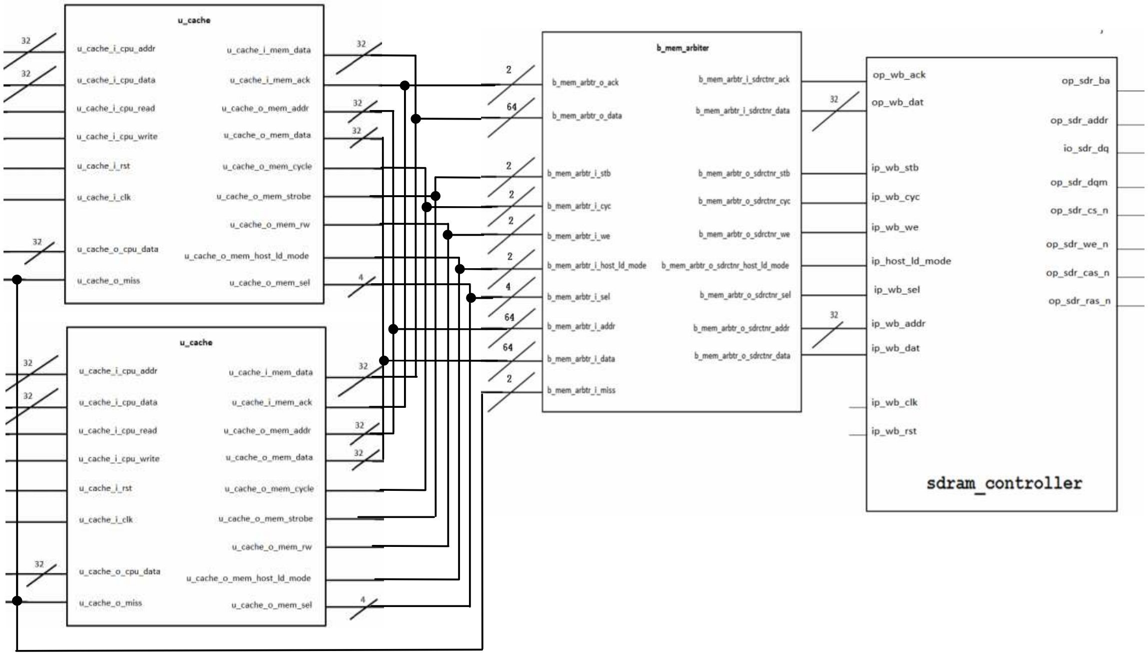


Figure 4.3: The interface of Arbiter and Two Cache units

4.4: Design of Memory Arbiter

There are four independent caches that need to access to the SDRAM. And the priority can be given in the order d-tlb > i-tlb > d-cache > i-cache. If four of those caches sent a miss signal at the same time, the d-tlb will first to access SDRAM, then i-tlb will take turn, and followed by the d-cache, and the i-cache will come to the end. The state diagram of memory arbiter can be designed as figure below:

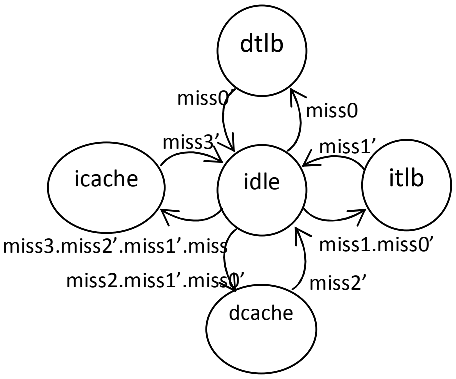


Figure 4.4 State diagram of Memory Arbiter

4.5: Designing Tools

Since this project is using Verilog, which is a Hardware Description Language (HDL). Simulations tools that support Verilog HDL is required, tools that provide simulation environment to verify the functional and timing models of the design, and the HDL source code. There are a lot HDL simulator created by different company, which has their own advantages and disadvantages. In order to choose most appropriate design tools for this project, some researches had been done and the choices has been narrow into three choices, which are the best HDL simulation tools available on the market, they are also known as the 'Big 3' simulators, three major signoff-grade simulators which qualified for application-specific integrated circuit (ASIC) (validation) sign-off at nearly all semiconductor fabrications. They are:

1. Incisive Enterprise Simulator by Cadence Design Systems
2. ModelSim by Mentor Graphic
3. Verilog Compiled code Simulator (VCS) by Synopsys

Simulator	Incisive Enterprise Simulator	ModelSim	VCS
Performance & functionality	high	moderate	High
Language Supported	VHDL-2002 V2001 SV2005	VHDL-2002 V2001 SV2005	VHDL-2002 V2001 SV2005
Simulation run speed	fastest	moderate	faster
Price	Expensive	Cost Saving and available for free SE edition	Expensive

Table 4.5 Comparison between 'Big 3' Simulators

Due to the availability, affordability, platform supported and performance requirement, the suitable simulator for this project is *Modelsim SE 10.3a* which is a freeware of student edition and is enough for the designing requirement. Other simulators may offer good features too, but no free license is provided to the students and the cost of each license is normally about $25000 and above which is unaffordable for a student.

4.6: Requirement Specification

This SDRAM controller is designed depends on Micron SDRAM MT48LC4M32B2 (1 Meg x 32 x 4 Banks). The entire design of SDRAM controller will need to fulfill the following requirements, which able to perform:

- Auto-refresh , 4096-cycles refresh (15.6µs/row)
- Auto-precharge, includes read, write and auto refresh mode
- Bank and row tracking for 4 banks
- Programmable burst length: 1,2,4,8 or full page
- Addressing controls
- I/O data buffer for read and write
- Supports CAS Latency (CL) of 1,2 and 3
- Self-refresh mode
- Command Generator to SDRAM
-

The SDRAM Controller design must provide input data for the Micron SDRAM as shown in the below:

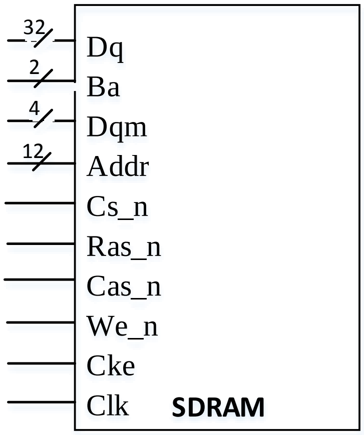
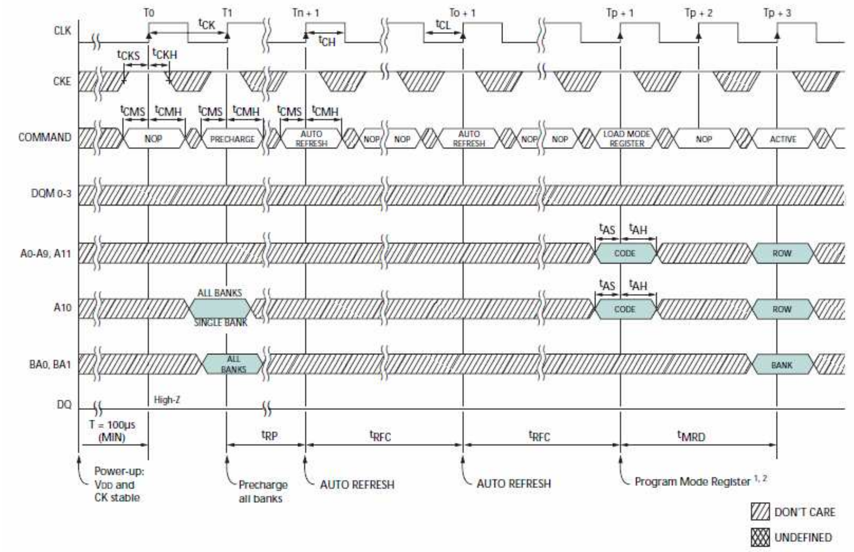


Figure 4.6.1: Micron SDRAM Block diagram

Additional timing diagram appear in the following requirement specification section; these timing diagrams provide better information for SDRAM controller design.

Initialize and Load Mode Register:



NOTE:
1. The Mode Register may be loaded prior to the AUTO REFRESH cycles if desired.
2. Outputs are guaranteed High-Z after command is issued.

Figure 4.6.F1: Initialize and Load Mode Register

Auto Refresh Mode:

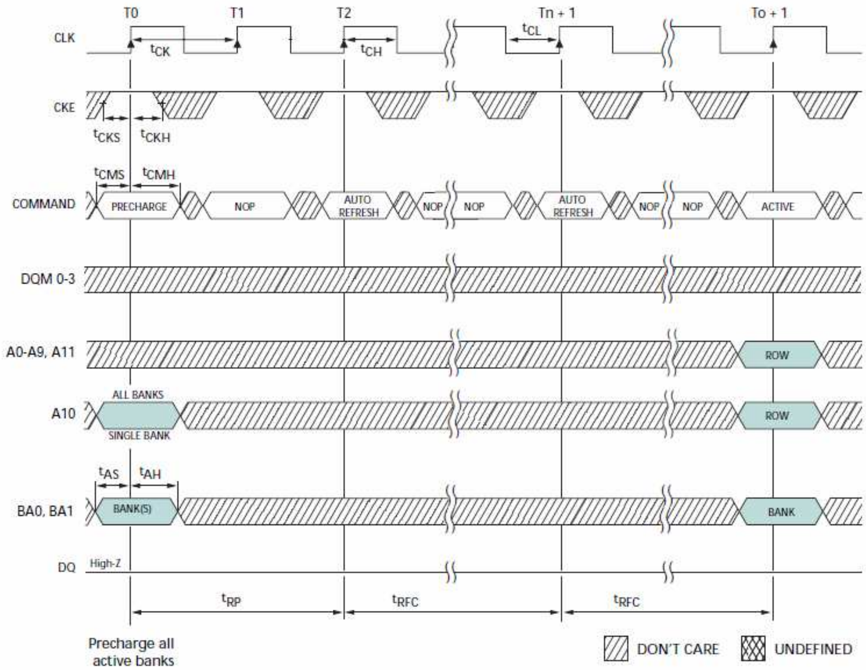


Figure 4.6.F2: Auto Refresh Mode

Self-Refresh Mode:

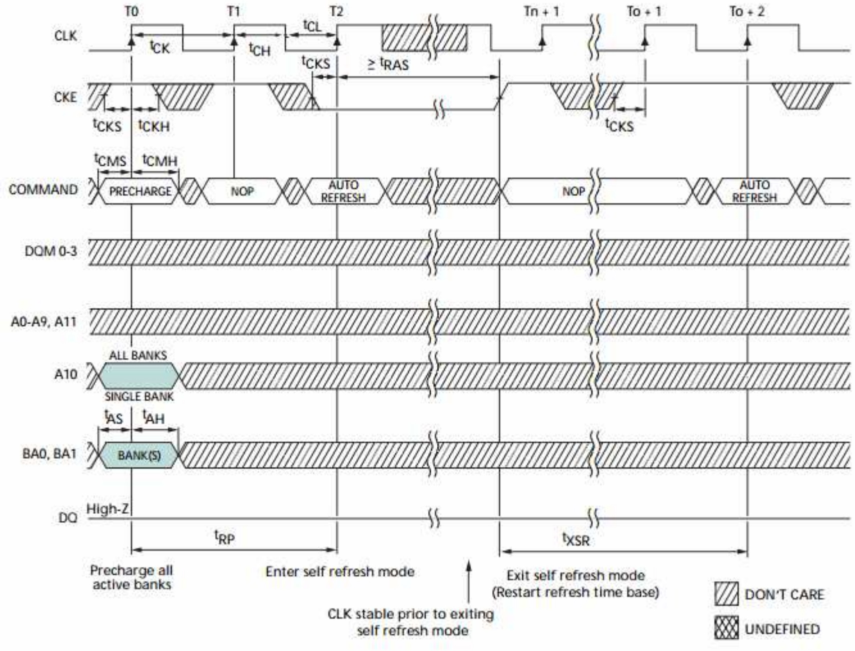


Figure 4.6.F3: Self-Refresh Mode

Single Read- Without Auto Precharge:

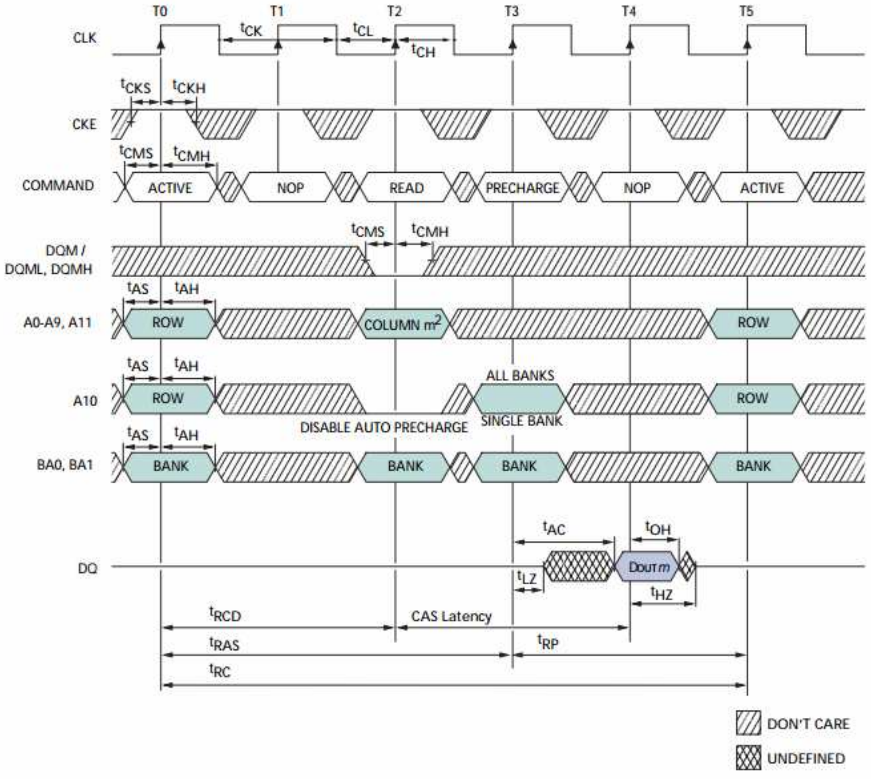


Figure 4.6.F4: Single Read- Without Auto Precharge

Read-With Auto Precharge:

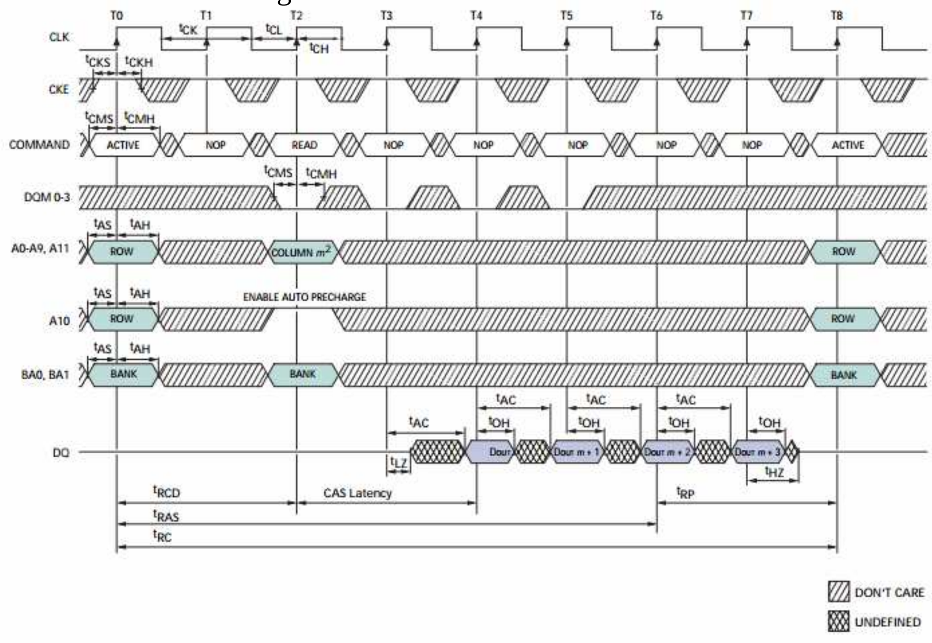
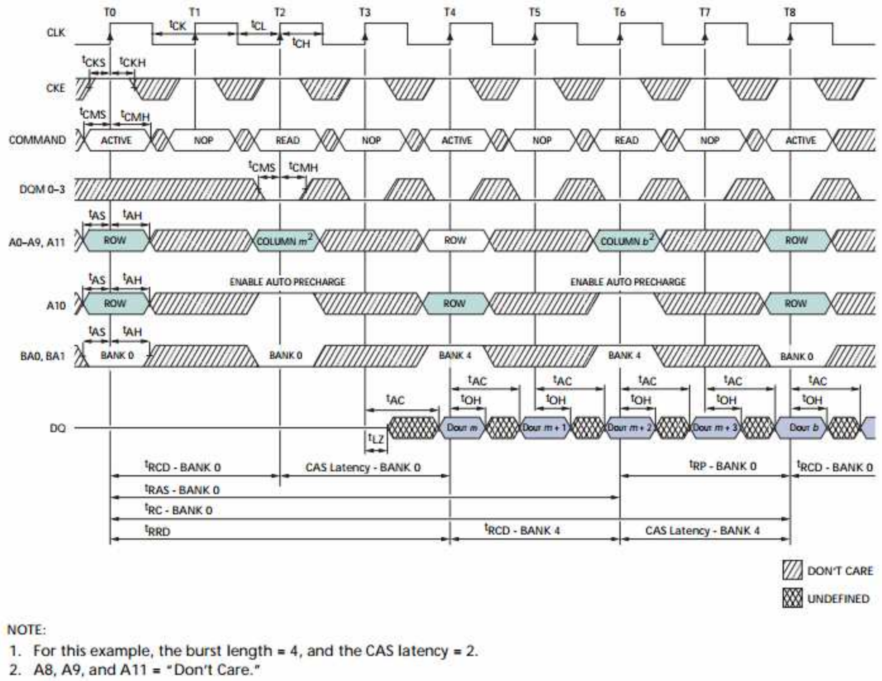


Figure 4.6.F5: Read- With Auto Precharge

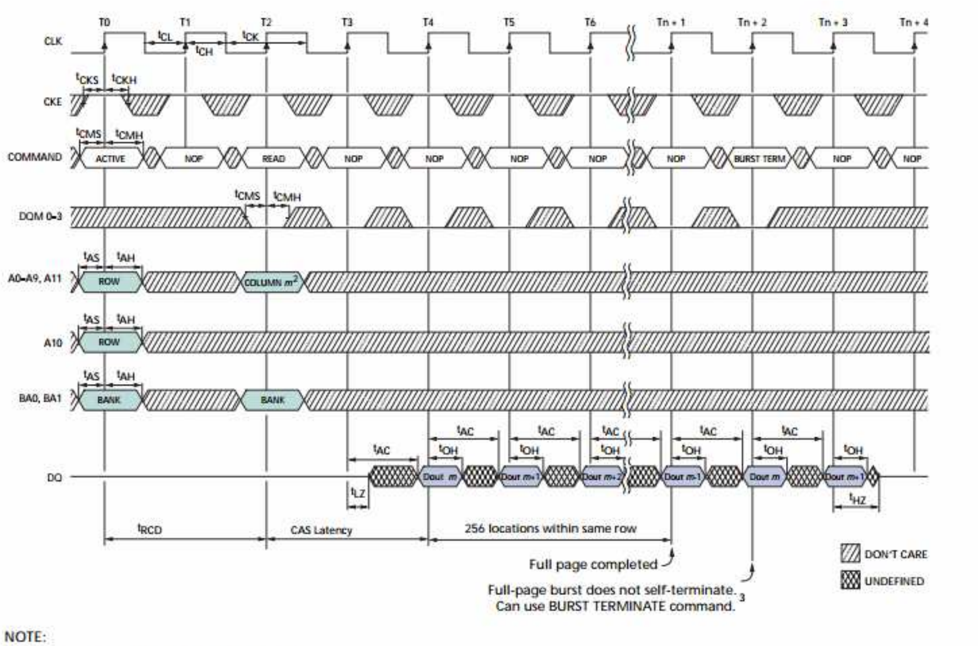
Alternating Bank Read Accesses:



NOTE:
1. For this example, the burst length = 4, and the CAS latency = 2.
2. A8, A9, and A11 = "Don't Care."

Figure 4.6.F6: Alternating Bank Read Accesses

Read – Full page Burst:

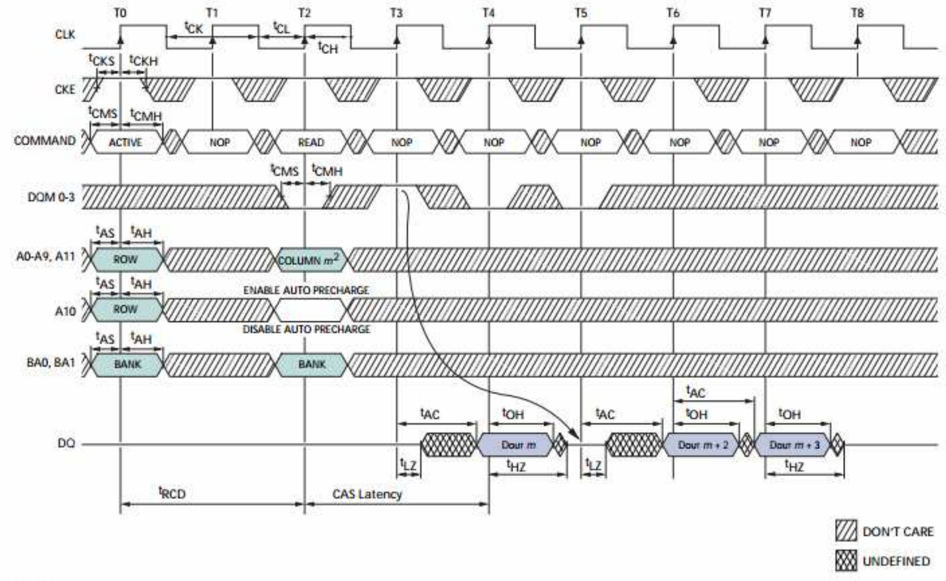


NOTE:

1. For this example, the CAS latency = 2.
2. A8, A9, and A11 = "Don't Care."
3. Page left open; no tRP.

Figure 4.6.F7: Read – Full page Burst

Read – DQM operation:

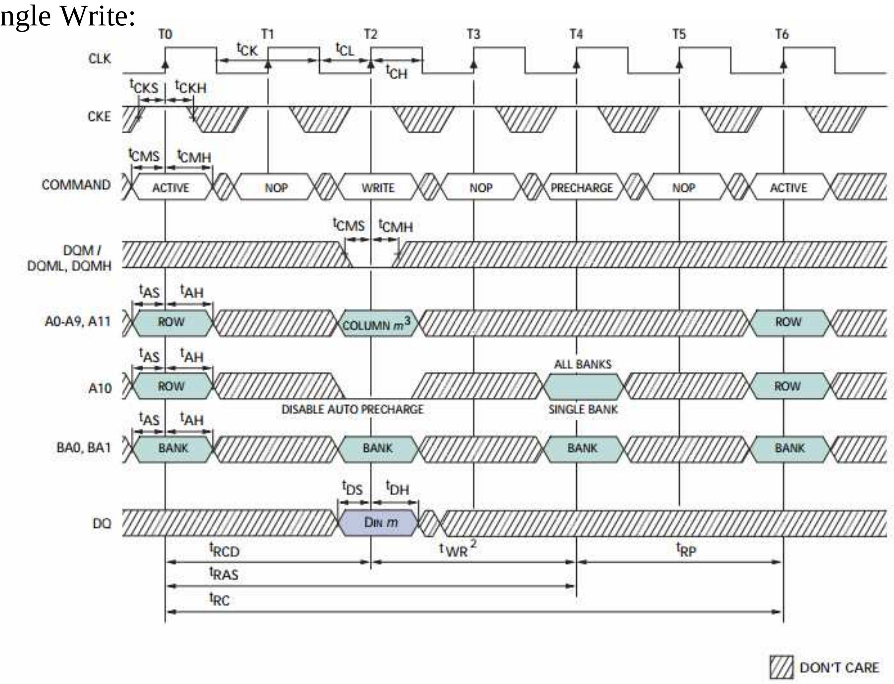


NOTE:

1. For this example, the CAS latency = 2.
2. A8, A9, and A11 = "Don't Care."

Figure 4.6.F8: Read – DQM operation

Single Write:

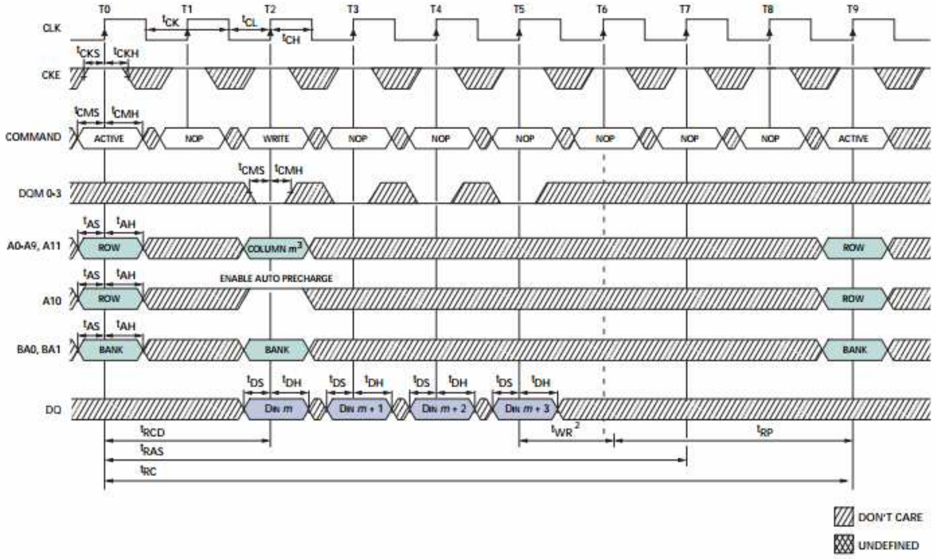


NOTE:

1. For this example, the burst length = 1, and the WRITE burst is followed by a "manual" PRECHARGE.
2. tWR is required between <Din m> and the PRECHARGE command, regardless of frequency.
3. A8, A9, and A11 = "Don't Care."

Figure 4.6.F9: Single Write

Write – With Auto Precharge:

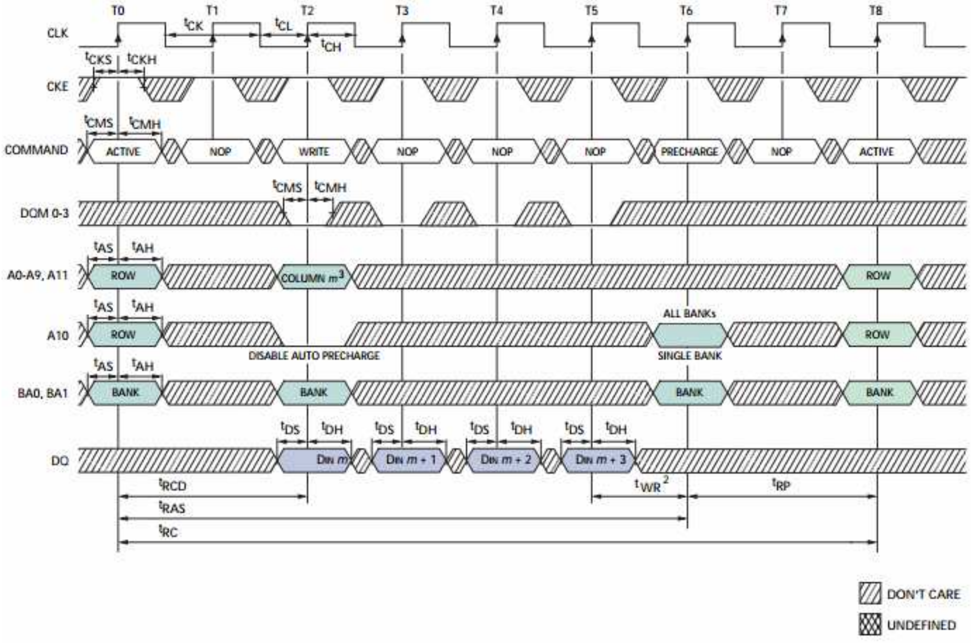


NOTE:

1. For this example, the burst length = 4.
2. Faster frequencies require two clocks (when tWR > tCK).
3. A8, A9, and A11 = "Don't Care."

Figure 4.6.F10: Write – With Auto Precharge

Write – Without Auto Precharge:

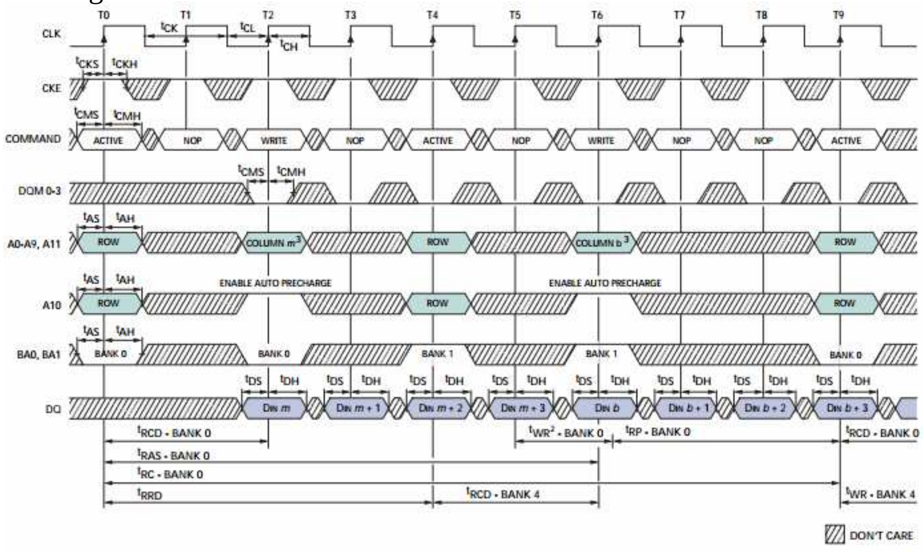


NOTE:

1. For this example, the burst length = 4, and the WRITE burst is followed by a "manual" PRECHARGE.
2. Faster frequencies require two clocks (when tWR > tCK).
3. A8, A9, and A11 = "Don't Care."
4. tWR of 1 CLK available if running 100 MHz or slower. Check factory for availability.

Figure 4.6.F11: Write – Without Auto Precharge

Alternating Bank Write Accesses:

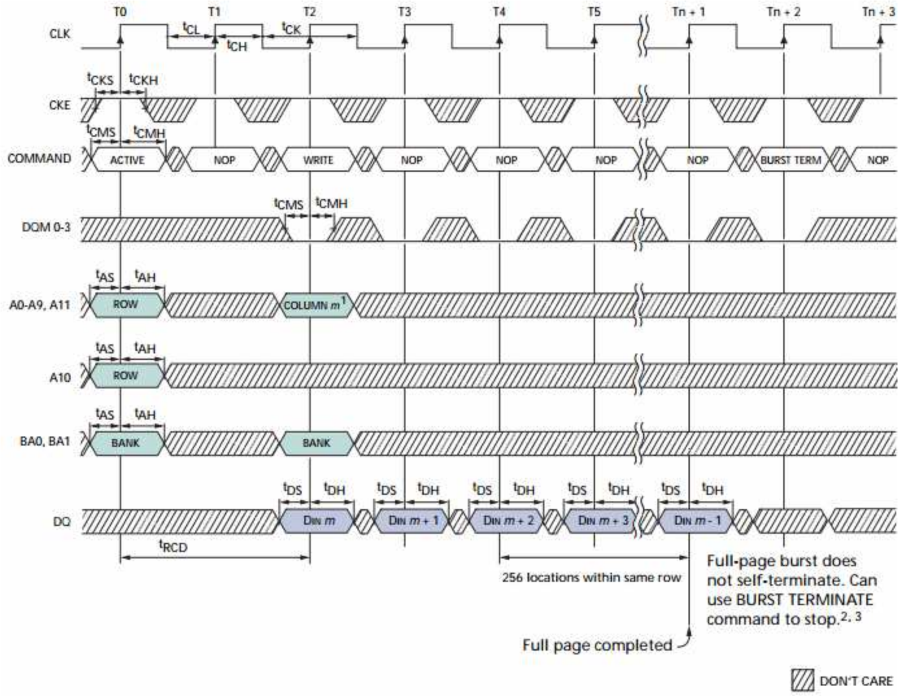


NOTE:

1. For this example, the burst length = 4.
2. Faster frequencies require two clocks (when tWR > tCK).
3. A8, A9, and A11 = "Don't Care."

Figure 4.6.F12: Alternating Bank Write Accesses

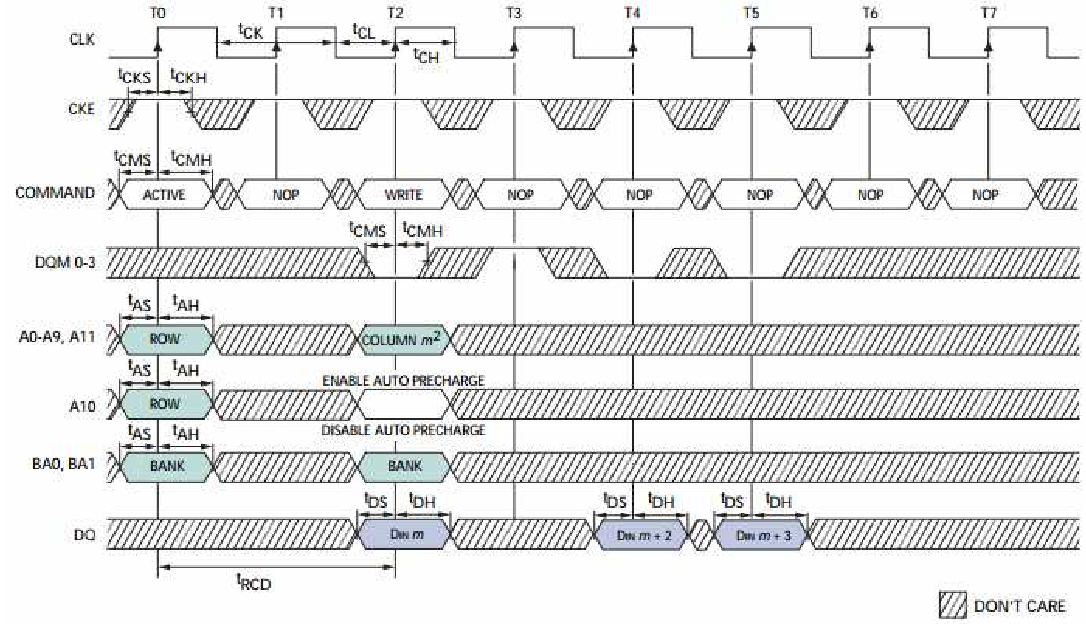
Write – Full Page Burst:



NOTE:
1. A8, A9, and A11 = "Don't Care."
2. tWR must be satisfied prior to PRECHARGE command.
3. Page left open; no tRP.

Figure 4.6.F13: Write – Full Page Burst

Write – DQM Operation:



NOTE:
1. For this example, the burst length = 4.
2. A8, A9, and A11 = "Don't Care."

Figure 4.6.F14: Write – DQM Operation

Consecutive Read Burst:

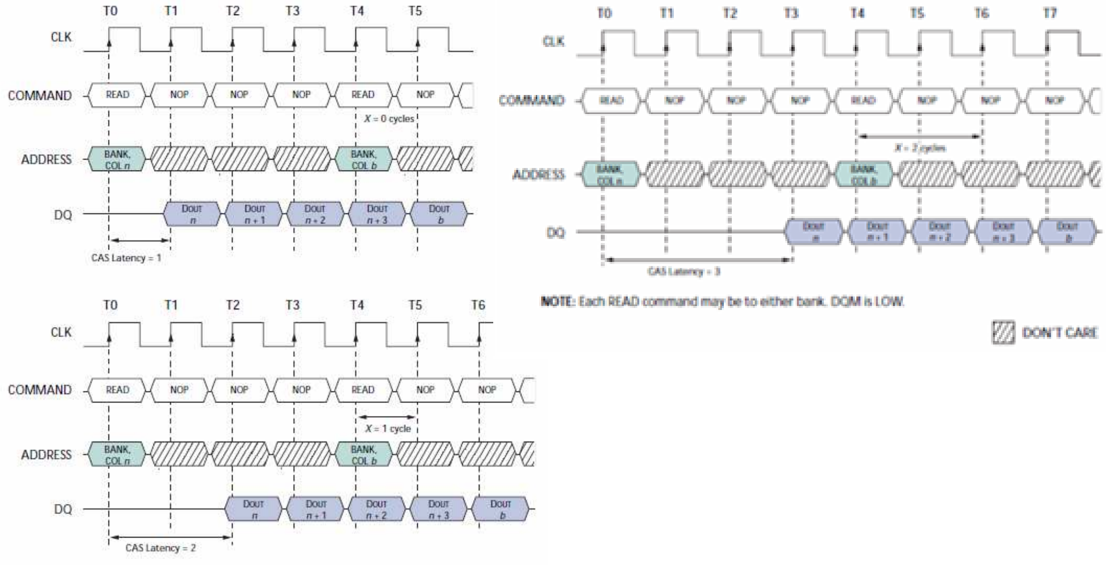


Figure 4.6.F15: Consecutive Read Burst

Terminating a Read Burst:

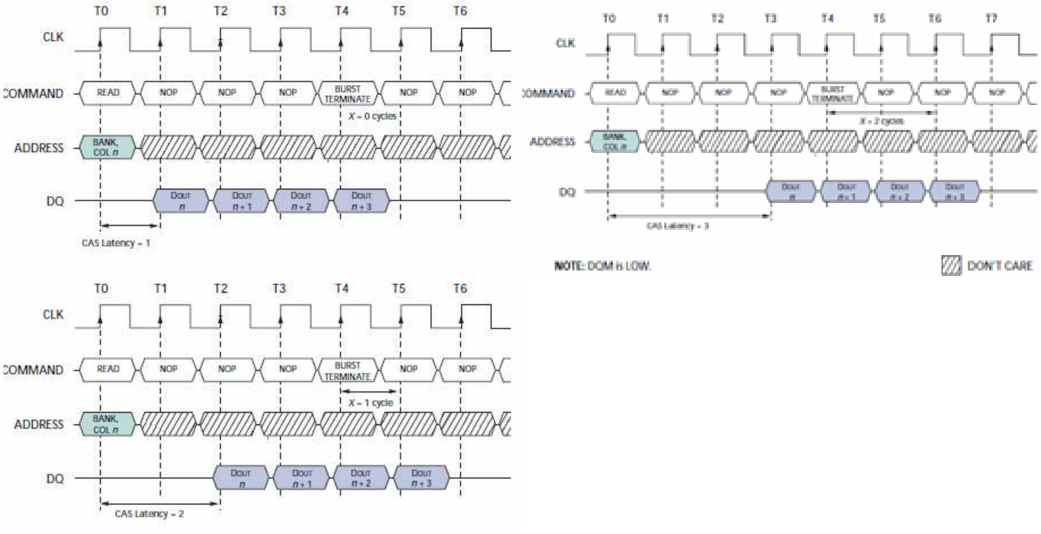


Figure 4.6.F16: Terminating a Read Burst

4.7: Timeline

Task Name	Duration	Start Date	End Date	week													
	(weeks)			1	2	3	4	5	6	7	8	9	10	11	12	13	14
Study the existing work that being developed	2	2/6/14	14/6/14	░	■	🟥											
Develop test for the existing RISC 32 pipeline processor	4	9/6/14	3/7/14		░	■	■	🟥	🟥								
Review the previous work of SDRAM controller	3	16/6/14	6/7/14			░	■	■	■								
Perform a deeper Literature Reviews	2	30/6/14	9/7/14					░	■	■							
Research and Fact Findings																	
***analyze the interface of SDRAM and Cache**	3	9/7/14	25/7/14						░	■	■	🟥					
Develop a Methodology and provide solutions																	
***Protocol Controller Block design**	2	26/7/14	3/7/14							░	░	■	🟥				
***improve SDRAM controller to support multiple cache**	3	27/7/14	5/8/14								░	■	■	🟥			
Verify the integration of the controller of cache and SDRAM by																	
***Develop an appropriate test**	3	28/7/14	18/8/14									░	■	■	■	🟩	
Meet with Supervisor weekly	**14**	**26/5/14**	**25/8/14**	■	■	■	■	■	■	■	■	■	■	■	🟩	🟩	🟩
Submission of proposal report			11/8/14								░				🟩		
Project I presentation			25/8/14														🟩
End of Project I			29/8/14														🟩

░ According to schedule

🟥 Completed beyond time

■ Completed

🟩 Planning

Table 4.7.1 Gantt chart for Project I

Task Name	Duration	week													
	(weeks)	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Specification and development															
i) Develop Architecture Specification	1	🟩													
ii) Develop Microarchitecture Specification	2		🟩	🟩											
iii) Develop Verification Specification	2			🟩	🟩										
Develop Test case and Verification	2							🟩	🟩	🟩					
Documentation Report Writing	1									🟩	🟩				
Meet with Supervisor weekly	14	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Submission of proposal report												🟩			
Project II presentation															🟩
End of Project II															🟩

Table 4.7.2 Planning Gantt chart for Project II

Chapter 5: Micro-architecture of Memory System

5.1: Memory System Micro-Architecture and its Partitioning

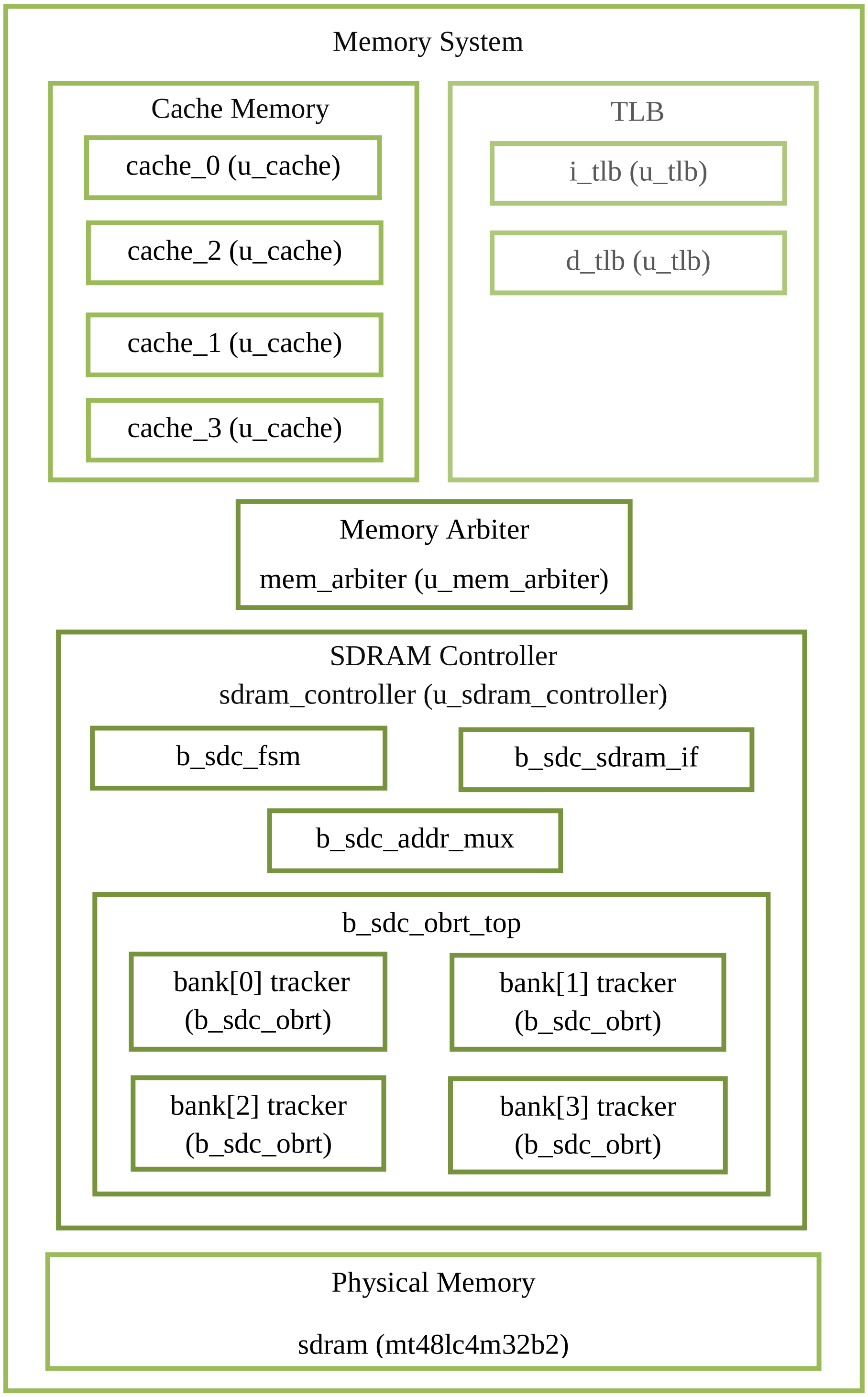


Figure 5.0: Memory System Micro-Architecture and its Partitioning

5.2: Design Hierarchy

Cache is involved for the project purpose, to verify the compatibility of memory system and SDRAM controller. However, the Translation Lookaside Buffer (TLB) is not included in this design since memory initialization can be done by the testbench.

Chip Partitioning at Architecture level	Unit Partitioning at Micro-Architecture Level	Block and Functional Block Partitioning at RTL level (Micro-Architecture level)
Memory System unit	u_cache (for instruction)	b_cache_ctrl
	u_cache (for data)	b_cache_ctrl
	u_mem_arbiter	**-**
	u_sdram_controller	**b_sdc_fsm**
		b_sdc_sdram_if
		b_sdc_addr_mux
		b_sdc_obrt_top
	sdram (mt48lc4m32b2)	-

Table 5.1: Formation of a design hierarchy for 32-bit Memory System

Unit Partitioning of Memory System

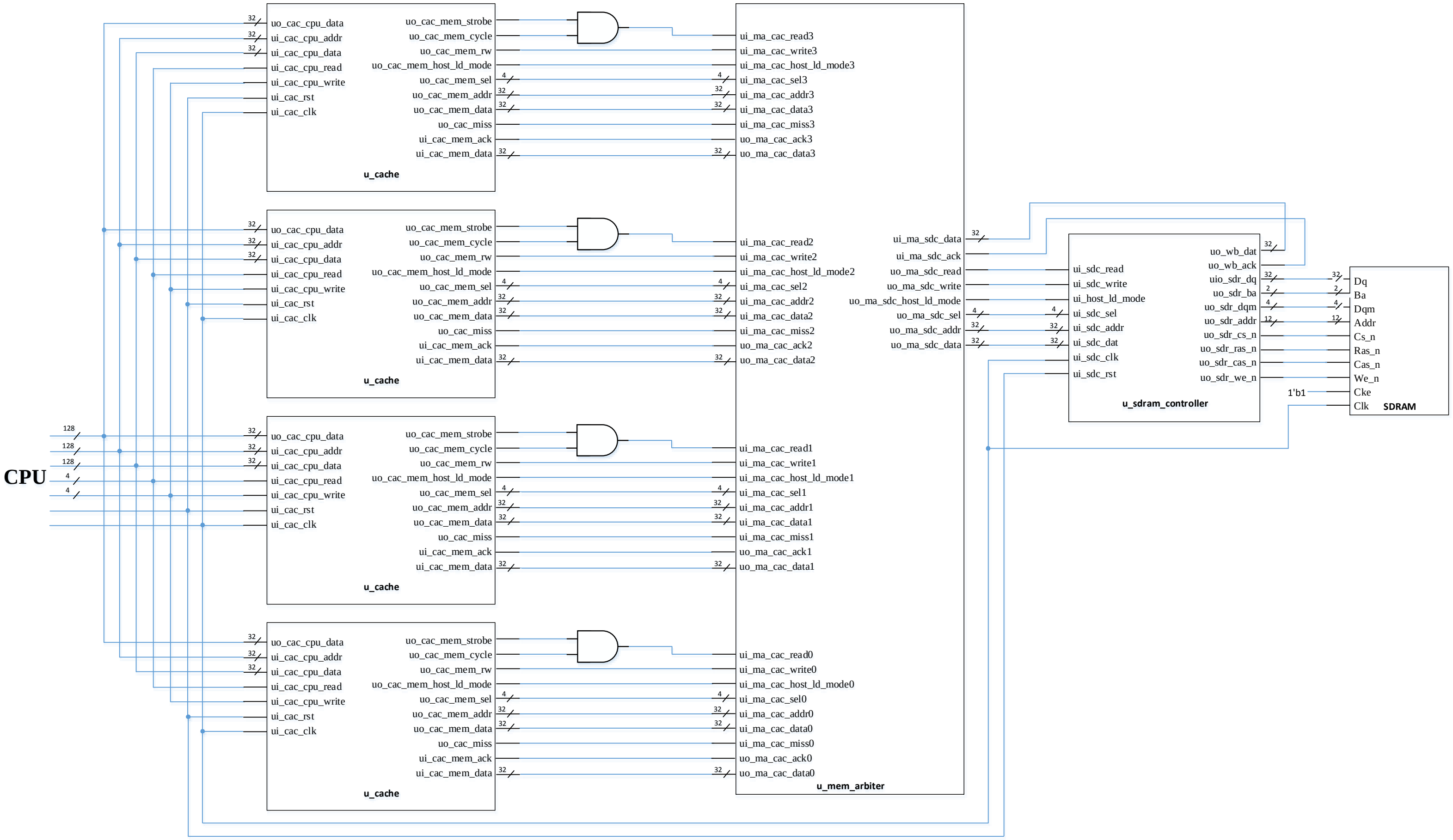


Figure 6: Unit Partitioning of Memory System

6.1: Cache Unit

This is a 2-way set associative cache. Functionalities of Cache Unit:

1. Store a small fraction of data (for D-Cache) or instructions (for I-Cache) of main memory.
2. Output desired data or instruction to CPU when it issues a READ.
3. Write data into desired location as instructed by CPU (D-Cache only).
4. Send signal to stall the CPU when read miss or write miss.
5. Communicate with SDRAM Controller to write back 'dirty' block of data back into SDRAM and fetch new block of data from it.

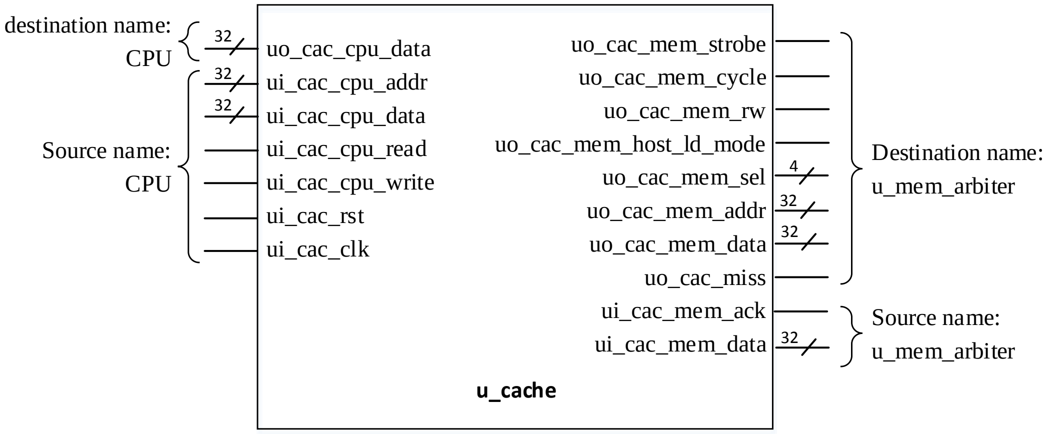


Figure 6.1: Cache Unit Block Diagram

This design includes Wishbone bus output signals, which are strobe and cycle, indicate that a valid bus cycle in progress and chip selected. However, the SDRAM controller does not use any Wishbone interfaces. The design is unnecessary for the cache and should be removed in future development. And yet it uses to test for the compatibility of new SDRAM controller only.

6.1.1: I/O Description

Pin name: ui_cac_clk **Pin class:** Global **Path:** External → Cache **Description:** System clock signal.
Pin name: ui_cac_rst **Pin class:** Global **Path:** External → Cache **Description:** System reset signal.

Pin name: ui_cac_cpu_data
Pin class: Data
Path: CPU→ Cache
Description: 32-bits data from CPU that to be written into the cache.

Pin name: ui_cac_cpu_addr
Pin class: Address
Path: CPU→ Cache
Description: 32-bits address from CPU that indicates a certain location that to be accessed.

Pin name: ui_cac_cpu_read
Pin class: Control
Path: CPU→ Cache
Description: A control signal that enables the read from cache from given address when it is asserted (HIGH).

Pin name: ui_cac_cpu_write
Pin class: Control
Path: CPU→ Cache
Description: A control signal that enables the write of data into a certain location in cache when it is asserted (HIGH).

Pin name: uo_cac_cpu_data
Pin class: Data
Path: Cache→ CPU
Description: 32-bits data that to be output to CPU.

Pin name: uo_cac_mem_strobe
Pin class: Control
Path: Cache→ Memory Arbiter
Description: Strobe signal that goes into SDRAM Controller.

Pin name: uo_cac_mem_cycle
Pin class: Control
Path: Cache→ Memory Arbiter
Description: Cycle signal that goes into SDRAM Controller.

Pin name: uo_cac_mem_rw
Pin class: Control
Path: Cache→ Memory Arbiter
Description: A read or write signal that goes into SDRAM Controller.
When ‘1’, write.
When ‘0’, read.

Pin name: uo_cac_mem_host_ld_mode
Pin class: Control
Path: Cache→ Memory Arbiter
Description: Assert (HIGH) this signal to configure the operating mode of SDRAM

Pin name: uo_cac_mem_sel
Pin class: Control
Path: Cache→ Memory Arbiter
Description: 4-bits control signals to mask which byte of the 4 bytes (32-bits) data goes in or comes out from SDRAM.

When it is ‘1’, the corresponding byte will enable. When it is ‘0’, the corresponding byte will be masked and the output becomes ‘z’.
Pin name: uo_cac_mem_addr **Pin class:** Address **Path:** Cache→ Memory Arbiter **Description:** 32-bits address that indicates which location in the SDRAM to be accessed.
Pin name: uo_cac_mem_data **Pin class:** Data **Path:** Cache→ Memory Arbiter **Description:** 32-bits data that to be written in to the SDRAM. When in host load mode, it contains the valid mode value for configuration.
Pin name: uo_cac_miss **Pin class:** Control **Path:** Cache→ Memory Arbiter **Description:** A status signal indicates cache miss. It is to stall the pipelines.
Pin name: ui_cac_mem_ack **Pin class:** Control **Path:** Memory Arbiter → Cache **Description:** Acknowledge signal (active HIGH) to indicate read or write to SDRAM is done.
Pin name: ui_cac_mem_data **Pin class:** Data **Path:** Memory Arbiter → Cache **Description:** 32-bits data that is read from SDRAM.

Table 6.1.1: Cache Unit I/O Descriptions

6.2: Memory Arbiter

The memory arbiter allows multiple caches or TLB to access single SDRAM. In order to do that, different priorities are given to d_TLB, i_TLB, d_Cache and i_Cache. The block diagram below shows a memory arbiter that can support up to 4 caches.

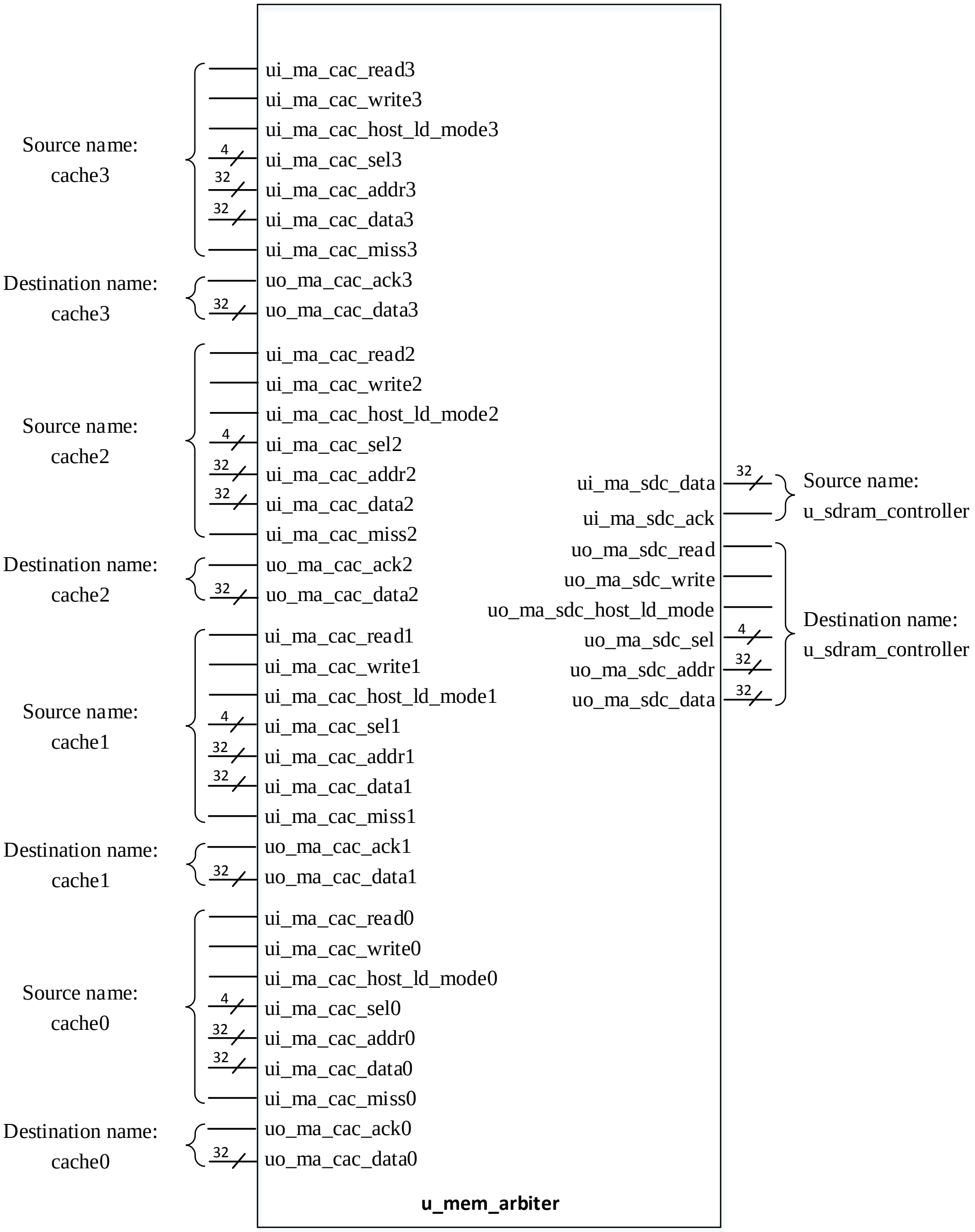


Figure 6.2: Memory Arbiter Block Diagram

6.2.1: I/O Description

Pin name: ui_ma_cac_read
Pin class: Control
Path: TLB or Cache → Memory Arbiter
Description: read signals from the TLBs and Caches.

Pin name: ui_ma_cac_write
Pin class: Control
Path: TLB or Cache → Memory Arbiter
Description: write signal from the TLBs and Caches.

Pin name: ui_ma_cac_host_ld_mode
Pin class: Control
Path: TLB or Cache → Memory Arbiter
Description: Host Load Mode signals from the TLBs and Caches.

Pin name: ui_ma_cac_sel
Pin class: Control
Path: TLB or Cache → Memory Arbiter
Description: Byte Select signals from the TLBs and Caches.

Pin name: ui_ma_cac_addr
Pin class: Address
Path: TLB or Cache → Memory Arbiter
Description: Addresses from the TLBs and Caches.

Pin name: ui_ma_cac_data
Pin class: Data
Path: TLB or Cache → Memory Arbiter
Description: Data from the TLBs and Caches.

Pin name: ui_ma_cac_miss
Pin class: Control
Path: TLB or Cache → Memory Arbiter
Description: Miss signals from the TLBs and Caches.

Pin name: uo_ma_cac_ack
Pin class: Control
Path: Memory Arbiter → TLB or Cache
Description: Acknowledge signal (active HIGH) to indicate read or write to SDRAM is done, and send to Caches or TLB.

Pin name: uo_ma_cac_data
Pin class: Data
Path: Memory Arbiter → TLB or Cache
Description: 32-bits data that goes to Cache or TLB.

Pin name: ui_ma_sdc_data
Pin class: Data
Path: Memory Arbiter → SDRAM Controller
Description: 32-bits data that comes from SDRAM.

Pin name: ui_ma_sdc_ack
Pin class: control
Path: Memory Arbiter → SDRAM Controller

Description: Acknowledge signal (active HIGH) to indicate read or write to SDRAM is done.
Pin name: uo_ma_sdc_host_ld_mode **Pin class:** control **Path:** Memory Arbiter → SDRAM Controller **Description:** Host Load Mode signals that send to SDRAM Controller.
Pin name: uo_ma_sdc_read **Pin class:** control **Path:** Memory Arbiter → SDRAM Controller **Description:** read signal that goes to SDRAM Controller
Pin name: uo_ma_sdc_write **Pin class:** control **Path:** Memory Arbiter → SDRAM Controller **Description:** Write signal that goes to SDRAM Controller.
Pin name: uo_ma_sdc_sel **Pin class:** control **Path:** Memory Arbiter → SDRAM Controller **Description:** 4-bits control signals to mask which byte of the 4 bytes (32-bits) data goes in or comes out from SDRAM. When it is ‘1’, the corresponding byte will enable. When it is ‘0’, the corresponding byte will be masked and the output becomes ‘z’.
Pin name: uo_ma_sdc_addr **Pin class:** control **Path:** SDRAM Controller → Memory Arbiter **Description:** 32-bits address to indicate which location in the SDRAM to be accessed.
Pin name: uo_ma_sdc_data **Pin class:** control **Path:** SDRAM Controller → Memory Arbiter **Description:** 32-bits data that goes into the SDRAM. When wants to configure the operating mode of the SDRAM, the configuration values goes into SDRAM via this port too.

Table 6.2.1: Memory Arbiter I/O Descriptions

6.2.2: Memory Arbiter State Diagram

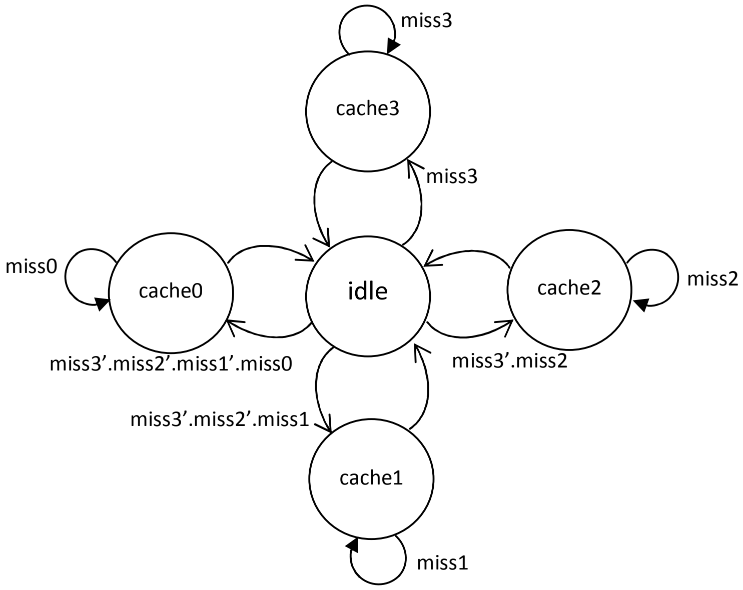


Figure 6.2.2: Memory Arbiter State Diagram

6.2.3 State Definition

	State Name	Definition
Memory Arbiter	cache3	First priority cache given to perform operation
	cache2	Second priority cache given to perform operation
	cache1	Third priority cache given to perform operation
	cache0	Last priority cache given to perform operation
	idle	Wait for new operation

Table 6.2.3: State Definition

6.3: SDRAM Controller

The SDRAM controller acts as an intermediary between the SDRAM and the host. It handles SDRAM operations using the protocols which will be explained section 6.4.1 Protocol Controller. And it has no longer been modeled based on Industry standard HOST SoC interface due to the current design needs.

Some of the main features are:

1) Burst transfers and burst termination
2) SDRAM initialization support
3) Performance optimization by leaving active rows open
4) Load mode control

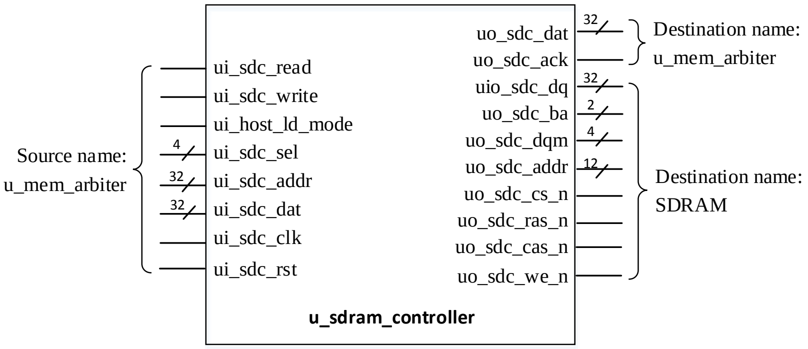


Figure 6.3: SDRAM Controller Block Diagram

6.3.1: I/O Pin Descriptions

Pin name: ui_sdc_clk
Pin class: Global
Path: Memory Bus Clock → SDRAM Controller
Description: SDRAM Controller Clock Input

Pin name: ui_sdc_rst
Pin class: Global
Path: System Reset → SDRAM Controller
Description: SDRAM Controller Reset

Pin name: ui_sdc_read
Pin class: Control
Path: Memory Arbiter → SDRAM Controller
Description: This pin indicates that the current cycle is READ when it asserted high.

Pin name: ui_sdc_we
Pin class: Control
Path: Memory Arbiter → SDRAM Controller
Description: This pin indicates that the current cycle is WRITE when it asserted high.

Pin name: uo_sdc_ack
Pin class: Control
Path: SDRAM Controller → Memory Arbiter
Description: When asserted high, it indicates that the current READ or WRITE is successful. When asserted low, it indicates the operation is not completed yet or no operation is processing now.

Pin name: ui_sdc_sel
Pin class: Control
Path: Memory Arbiter → SDRAM Controller
Description: This signal indicates where valid data is placed on the input data line (ui_wb_dat) during WRITE cycle and where it should present on the output data line (uo_wb_dat) during READ cycle. The array boundaries are determined by the granularity of a port. In this SDRAM controller, 8-bits granularity is used and all the data ports are 32-bits. Therefore, there would be 4 select signals with the boundaries of ui_wb_sel(3:0). Each individual select signal correlates to one of 4 active bytes on the 32-bits data port.

Pin name: ui_sdc_addr
Pin class: Address
Path: Memory Arbiter → SDRAM Controller
Description: The address input is used to pass the memory address from the host.

Pin name: ui_sdc_dat
Pin class: Data
Path: Memory Arbiter → SDRAM Controller
Description: This pin is used to pass WRITE data from the host.

Pin name: uo_sdc_dat
Pin class: Data
Path: SDRAM Controller → Memory Arbiter
Description: This pin is used to output READ data from the SDRAM.

Pin name: ui_host_ld_mode
Pin class: Control
Path: SDRAM Controller → Memory Arbiter
Description: This pin is asserted to load a new mode into the SDRAM.

Pin name: uo_sdc_cs_n
Pin class: Control
Path: Memory Arbiter → SDRAM
Description: SDRAM chip select

Pin name: uo_sdc_ras_n
Pin class: Control
Path: Memory Arbiter → SDRAM
Description: SDRAM row address select

Pin name: uo_sdc_cas_n

Pin class: Control **Path:** Memory Arbiter → SDRAM **Description:** SDRAM column address select
Pin name: uo_sdc_we_n **Pin class:** Control **Path:** Memory Arbiter → SDRAM **Description:** SDRAM write enable.
Pin name: uo_sdc_addr **Pin class:** Address **Path:** Memory Arbiter → SDRAM **Description:** This pin is used as an address output to the SDRAM. The address will be segmented into row, column and bank before being sent out through this pin.
Pin name: uo_sdc_ba **Pin class:** Control **Path:** Memory Arbiter → SDRAM **Description:** This pin is used to select the bank within the SDRAM. There are a total of 4 banks within the SDRAM and each of them operates independently.
Pin name: uo_sdc_dqm **Pin class:** Control **Path:** Memory Arbiter → SDRAM **Description:** This pin is used to select which bits of the data line (uio_sdr_dq) to be masked.
Pin name: uio_sdc_dq **Pin class:** Data **Path:** Memory Arbiter → SDRAM **Description:** This data line is a bidirectional line to receive READ data or send WRITE data.

Table 6.3.1: SDRAM I/O Descriptions

6.4: Block partitioning of SDRAM Controller

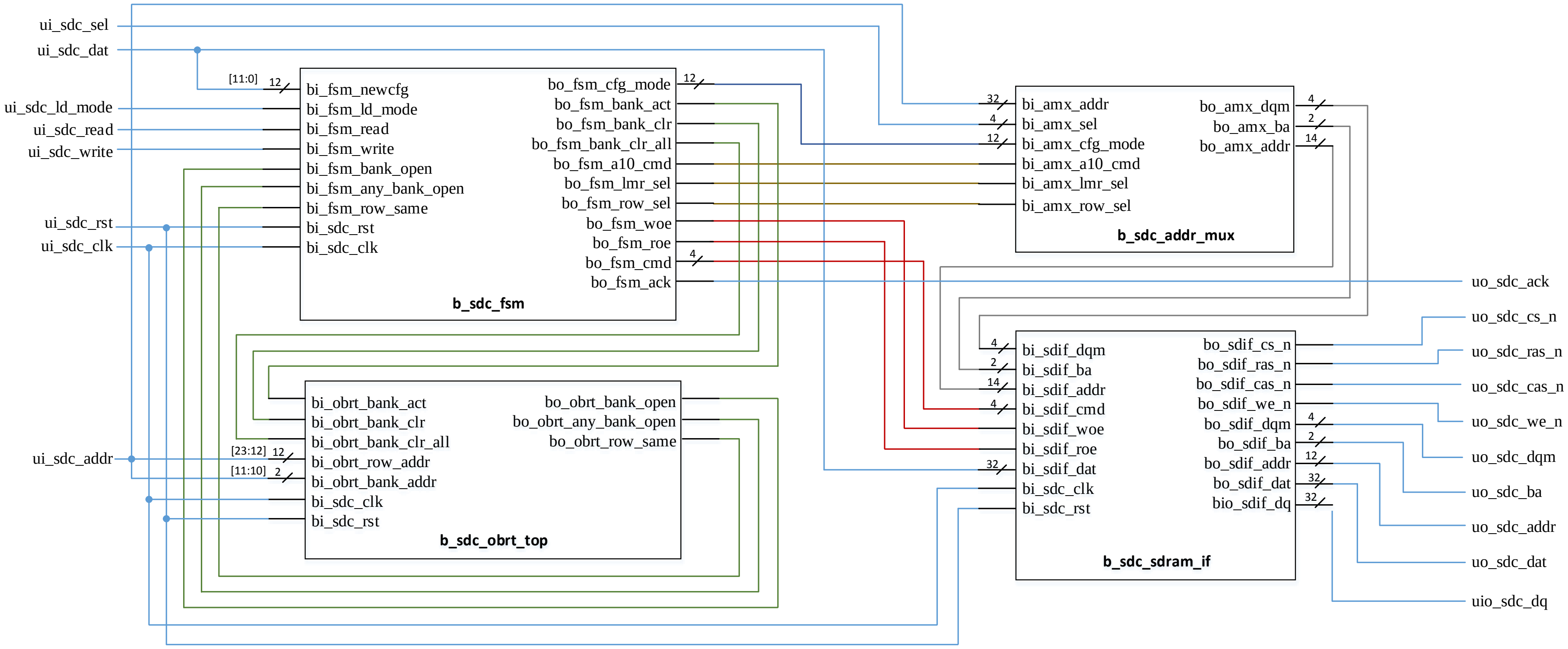


Figure 6.4: The Micro-Architecture of the SDRAM Controller

6.4.1: Protocol Controller

This block handles the timing and the state changes that forms the protocols of the SDRAM. It decides which protocol to be executed and what commands to be sent to the SDRAM. This block performs simple decoding on the HOST signals and uses them as input controls for the states.

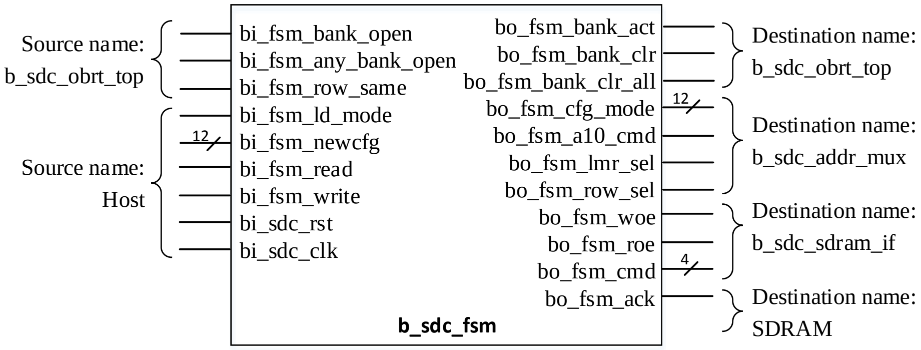


Figure 6.4.1: Protocol Controller Block Diagram

6.4.1.1: I/O Pin Descriptions

Pin Name : bi_sdc_clk **Pin class:** Global **Path**: Host → Protocol Controller **Description:** Clock Input
Pin Name : bi_sdc_rst **Pin class:** Global **Path**: Host → Protocol Controller **Description:** Synchronous reset
Pin Name : bi_fsm_read **Pin class:** Control **Path**: Host → Protocol Controller **Description:** When asserted high, this pin indicates that the current cycle is READ.
Pin Name : bi_fsm_write **Pin class:** Control **Path**: Host → Protocol Controller **Description:** When asserted high, this pin indicates that the current cycle is WRITE.
Pin Name : bi_fsm_ack **Pin class:** Control **Path**: Host → Protocol Controller **Description:** Acknowledge signal is activated after read or write is done.
Pin Name : bi_fsm_ld_mode **Pin class:** Control **Path**: Host → Protocol Controller **Description:** This pin is asserted to request for load mode.

Pin Name : bi_ fsm_newcfg
Pin class: Control
Path: Host → Protocol Controller
Description: 12-bits mode configuration status

Pin Name : bi_fsm_bank_open
Pin class: Control
Path: OBRT → Protocol Controller
Description: If deasserted, row status is “row closed”.
If asserted, row status is “row opened”.

Pin Name : bi_fsm_any_bank_open
Pin class: Control
Path: OBRT → Protocol Controller
Description: If deasserted, the row status for all banks is “row closed”.
If asserted, there is at least one bank with the status of “row opened”.

Pin Name : bi_fsm_row_same
Pin class: Data
Path: OBRT → Protocol Controller
Description: If asserted, the existing row is the same as the opened row in the selected bank.

Pin Name : bo_fsm_bank_act
Pin class: Control
Path: Protocol Controller → OBRT
Description: If asserted, Protocol Controller requests OBRT to update the bank status of the selected bank to “row opened”.

Pin Name : bo_fsm_bank_clr
Pin class: Control
Path: Protocol Controller → OBRT
Description: If asserted, Protocol Controller requests OBRT to update the bank status of the selected bank to “row closed”.

Pin Name : bo_fsm_bank_clr_all
Pin class: Control
Path: Protocol Controller → OBRT
Description: Asserted to set all the bank statuses in OBRT to “row clear”.

Pin Name : bo_ fsm_cfg_mode
Pin class: Control
Path: Protocol Controller → Address Multiplexer
Description: 12-bits mode configuration status

Pin Name : bo_fsm_a10_cmd
Pin class: Control
Path: Protocol Controller → Address Multiplexer
Description: Signal to be sent out to the address (10) of the SDRAM. During a row precharge, the assertion of this pin indicates precharge all banks.

Pin Name : bo_fsm_lmr_sel **Pin class:** Control **Path**: Protocol Controller → Address Multiplexer **Description:** Select load mode configuration.
Pin Name : bo_fsm_row_sel **Pin class:** Control **Path**: Protocol Controller → Address Multiplexer **Description:** Select row address
Pin Name : bo_fsm_woe **Pin class:** Control **Path**: Protocol Controller → Data Buffer **Description:** Write output buffer enable.
Pin Name : bo_fsm_roe **Pin class:** Control **Path**: Protocol Controller → Data Buffer **Description:** Read output buffer enable.
Pin Name : bo_fsm_cmd **Pin class:** Control **Path**: Protocol Controller → SDRAM Interface **Description:** Output SDRAM commands

Table 6.4.1.1: Protocol Controller Input/ Output Pin Descriptions

6.4.1.2: Protocol Controller State Diagram

This section details the state diagram of the Protocol Controller block. Figure 6.4.1.2.F1 shows the simplified view of the Protocol Controller FSM model, followed by the state diagram shown in Figure 6.4.1.2.F2 and Figure 6.4.1.2.F3.

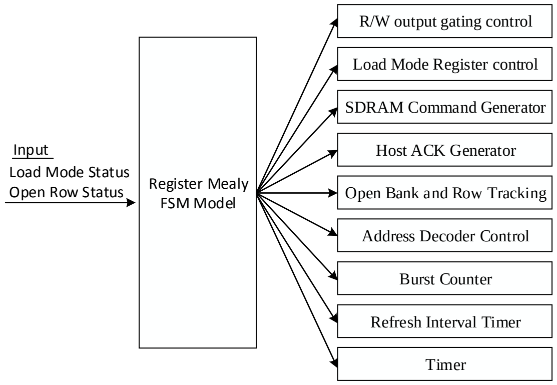


Figure 6.4.1.2.F1: A simplified view on the Protocol Block

The Protocol Controller is designed using a registered mealy model. There are 9 different output generators driven by the same FSM. Each of this output generators serves different micro functions at different state.

Initialization of SDRAM controller occurred when asynchronous reset signal is asserted high. SDRAM needs to perform power and clock stabilization. Before issuing read or write command, it has to go through at least 100µs delay during initialization wait state (INIT_W). The default timer value in INIT_W, b_tmr_done is set to 150µs, but the value has been scaled down for the testing purpose. The system is then throughout the wait states of pre-charge, reset, and load mode according to specific timing values. These delays can be modified by designer and decided by referring to the SDRAM types that has been chosen. After the initialization, state machine go directly to idle command state.

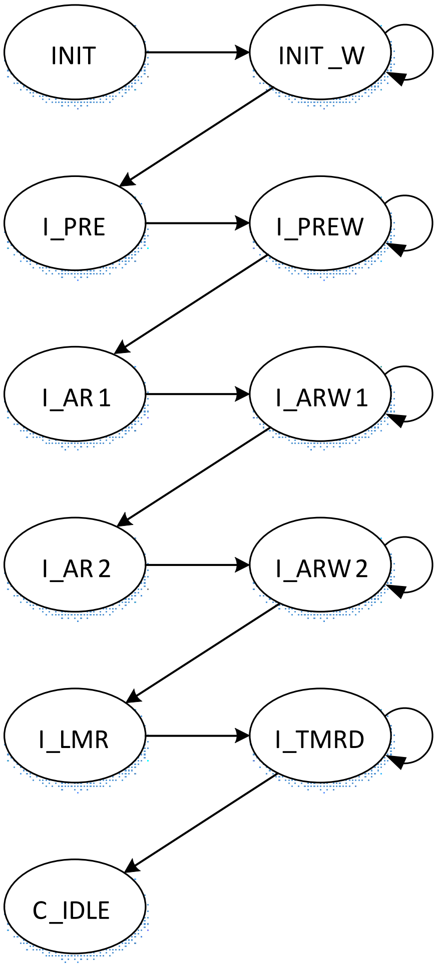


Figure 6.4.1.2.F2: Initialization Protocol FSM

If the chosen bank is not open during the read or write cycle time, the active command (C_ACT) will be issued to close the particular bank and open the chosen bank. If the chosen row is different, pre-charge will took place. Otherwise, read or write can be executed directly.

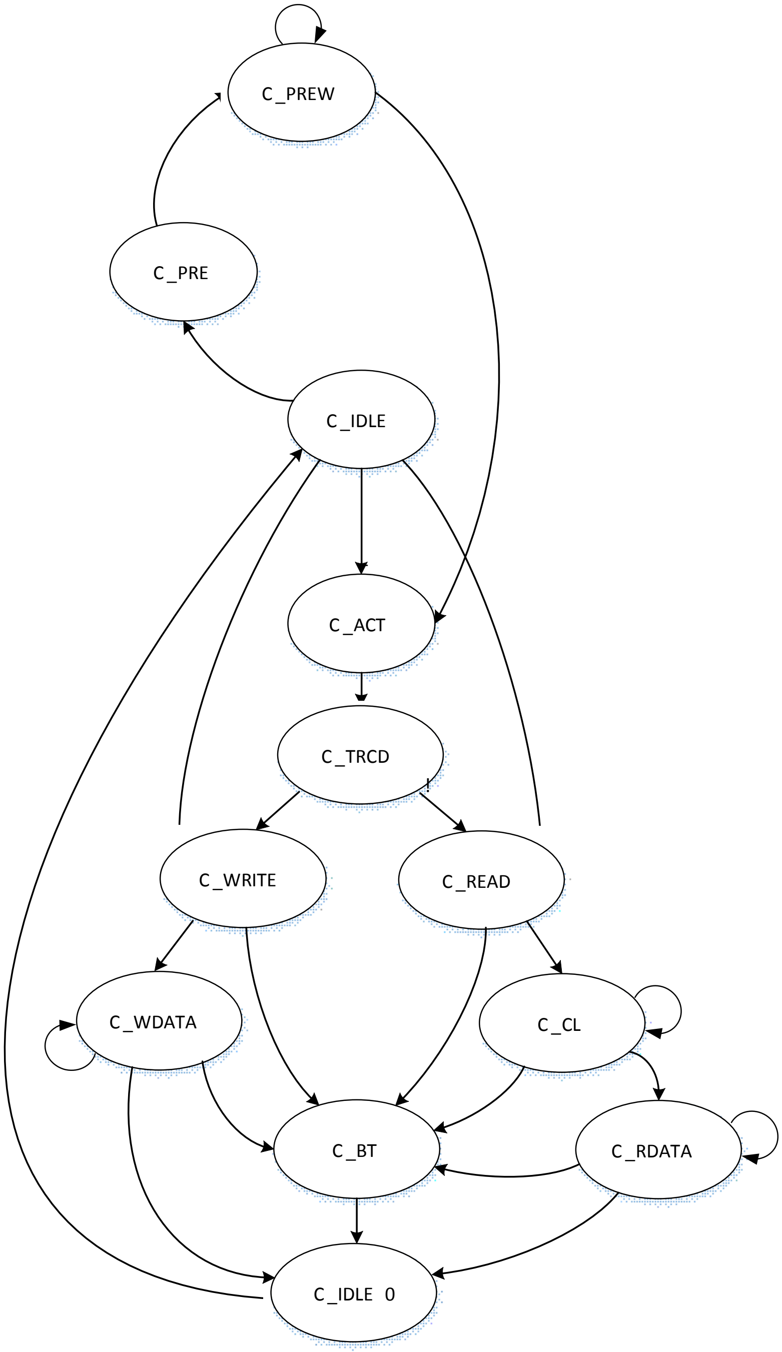


Figure 6.4.1.2.F6: Open Bank and Row Tracking Control sub-FSM

6.4.2: Open Bank and Row Tracking (OBRT) Top

This block is used to keep track of the row statuses for all the banks. It has 4 sub-blocks of OBRT instantiated within it to store and compare the row status (activated or precharged) of each bank. This block will select which sub-blocks row statuses to be updated. It also selects which sub-block's row statuses to be output to the protocol controller block.

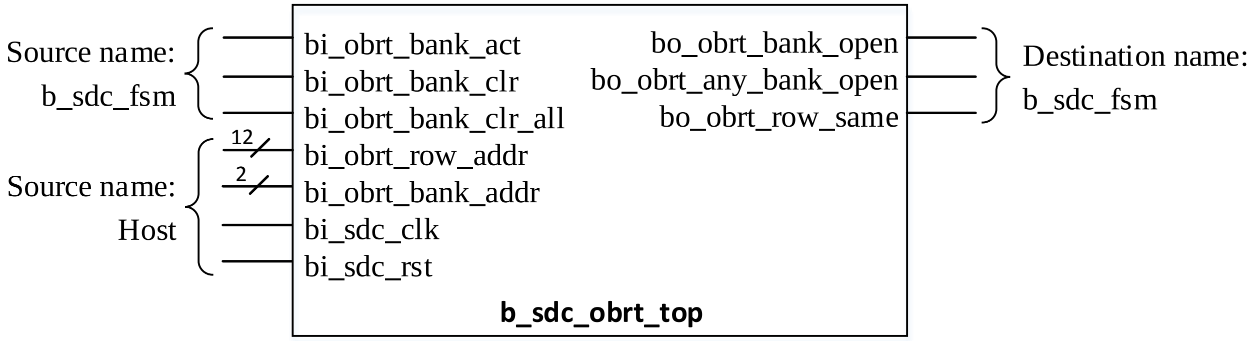


Figure 6.4.2: OBRT Top Block diagram

6.4.2.1: I/O Pin Descriptions

Pin Name : bi_sdc_clk
Pin class: Global
Path: Host → OBRT
Description: Clock Input

Pin Name : bi_sdc_rst
Pin class: Global
Path: Host → OBRT
Description: Synchronous reset

Pin Name : bi_ obrt_bank_addr
Pin class: Address
Path: Host → OBRT
Description: Input of bank address to select which bank's status to be updated or checked.

Pin Name : bi_ obrt_row_addr
Pin class: Address
Path: Host → OBRT
Description: Input row address to be compared with the activated row of the selected bank.

Pin Name : bi_obrt_bank_act
Pin class: Control
Path: Protocol Controller → OBRT
Description: Set the status of the selected bank as "Row Active"

Pin Name : bi_ obrt_bank_clr **Pin class:** Control **Path**: Protocol Controller → OBRT **Description:** Clear the status of the selected bank to indicate “Row Closed”
Pin Name : bi_obrt_bank_clr_all **Pin class:** Control **Path**: Protocol Controller → OBRT **Description:** Clear the status of all banks
Pin Name : bo_obrt_bank_open **Pin class:** Control **Path**: OBRT → Protocol Controller **Description:** Indicates “Row Active” status if asserted and “Row Closed” if de-asserted.
Pin Name : bo_obrt_any_bank_open **Pin class:** Control **Path**: OBRT → Protocol Controller **Description:** When asserted, it indicates if there is any bank with “Row Active” status.
Pin Name : bo_ obrt_row_same **Pin class:** Control **Path**: OBRT → Protocol Controller **Description:** Indicate “Row Same” status when the currently accessed row is the same as the activated row.

Table 6.4.2.1: OBRT Top Input /Output Pin Descriptions

6.4.2.3: Block Partitioning of OBRT Top

This sub-block is generated 4 times within the OBRT_Top Block. Each of these sub-blocks stores the row status (precharged, activated) of each bank.

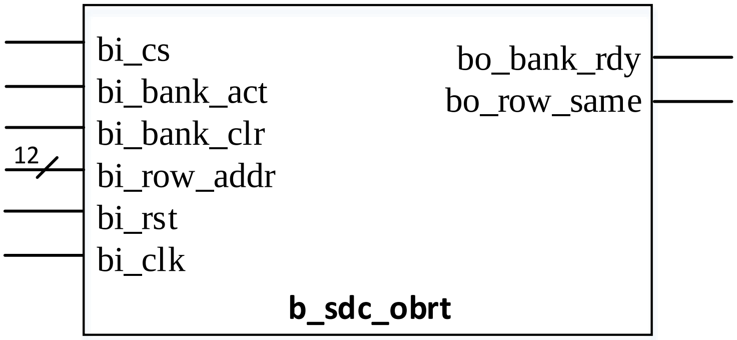


Figure 6.4.2.3: OBRT Sub-block Diagram

6.4.2.4: I/O Pin Descriptions

Pin Name : bi_clk
Pin class: System
Path: OBRT_Top → OBRT
Description: Clock Input

Pin Name : bi_ rst
Pin class: System
Path: OBRT_Top → OBRT
Description: Synchronous reset

Pin Name : bi_cs
Pin class: Control
Path: OBRT_Top → OBRT
Description: Chip select. This sub-block will not react to all input signals, with the exception of bi_rst, if this pin is not asserted.

Pin Name : bi_ bank_act
Pin class: Control
Path: OBRT_Top → OBRT
Description: If asserted, set bank status as "row active"

Pin Name : bi_ bank_clr
Pin class: Control
Path: OBRT_Top → OBRT
Description: If deasserted, set bank status as "row closed".

Pin Name : bi_row_addr
Pin class: Address
Path: OBRT_Top → OBRT
Description: Input row address to be compared with stored activated row address.

Pin Name : bo_bank_rdy

Pin class: Control
Path: OBRT → OBRT _Top
Description: Row active status. Indicate the row stored in the selected bank is ready. At reset, this register is initialized to 0. If asserted, indicates “row active”.
If deasserted, indicates “row closed”.

Pin Name : bo_row_same
Pin class: Control
Path: OBRT → OBRT _Top
Description: If asserted, indicates the input address is same as the stored address.

Table 6.4.2.4: OBRT Input/ Output Pins Descriptions

6.4.2.5 Important Registers in OBRT

There are a total of 4 trackers to track the row status of each bank. Within each tracker, there are 2 important registers used to track the row address and its activation status.

Pin Name : b_row_previous
Pin class: Register
Description: Stores the activated row to be compared with the input row address from bi_wb_row_addr. At reset, this register is initialized to 0.

Pin Name : bo_bank_rdy
Pin class: Register
Description: Indicate the row stored in the selected bank is ready. At reset, this register is initialized to 0.
If asserted, it indicates “row active”.If deasserted, it indicates “row closed”.

Table 6.4.2.5: OBRT Important Registers

6.4.3: Address Multiplexer

The address multiplexer (MUX) partitions the HOST address input line into row address, bank address and column address. Then, it multiplexes the configuration mode, row address and column address. It also decodes the HOST Select input pin and converts it to equivalent masking output.

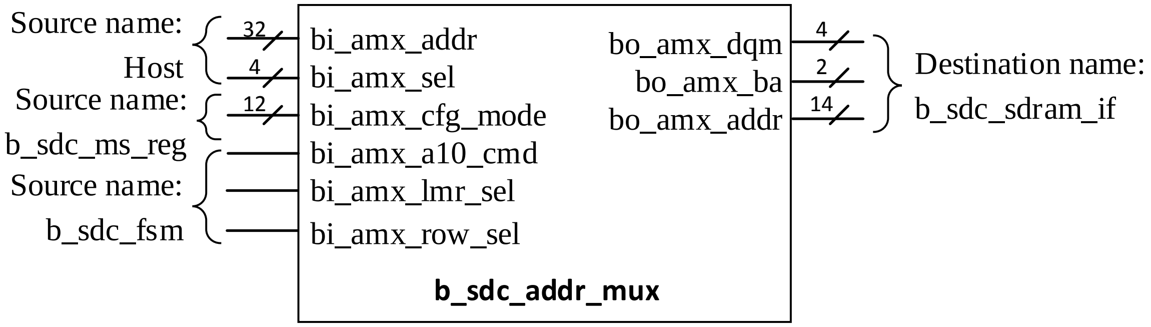


Figure 6.4.3: Address Multiplexer Block Diagram

6.4.3.1: I/O Descriptions

Pin Name : bi_amx_addr
Pin class: Address
Path: Host → Address Multiplexer
Description: Host address input. This input line is used to get the address of the host connected to the SDRAM controller

Pin Name : bi_amx_sel
Pin class: Control
Path: Host → Address Multiplexer
Description: Host Select Input. This input line is used to select which data on the 32-bit data line is valid. Can be used for the purpose of byte access, half-word access or word access.

Pin Name : bi_amx_cfg_mode
Pin class: Control
Path: CSR → Address Multiplexer
Description: This input is used to read the status of the configured mode. The status will be used as the value to configure the SDRAM when load mode protocol is executed.

Pin Name : bi_amx_a10_cmd
Pin class: Control
Path: Protocol Controller → Address Multiplexer
Description: Address bit-10 control signal

Pin Name : bi_amx_lmr_sel
Pin class: Control

Path: Protocol Controller → Address Multiplexer **Description:** Load mode select input
Pin Name : bi_amx_row_sel **Pin class:** Control **Path**: Protocol Controller → Address Multiplexer **Description:** Row address select input
Pin Name : bo_amx_dqm **Pin class:** Control **Path**: Address Multiplexer → SDRAM Interface **Description:** Masking output. Used to select which data line of the SDRAM to be masked. Refer to [12] for further details.
Pin Name : bo_amx_ba **Pin class:** Control **Path**: Address Multiplexer → SDRAM Interface **Description:** Bank address output
Pin Name : bo_amx_addr **Pin class:** Address **Path**: Address Multiplexer → SDRAM Interface **Description:** Multiplexer address output

Table 6.4.3.1: Address Multiplexer Input/ Output Pin Descriptions

6.4.4 SDRAM Interface Block Specification

The SDRAM Interface Block synchronizes all the signals to negative edge for the write cycle and positive edge for the read cycle, before sending them out the SDRAM. Within the host and SDRAM, there are two tri-states buffers used as the gating mechanism to enable the data to flow in or out.

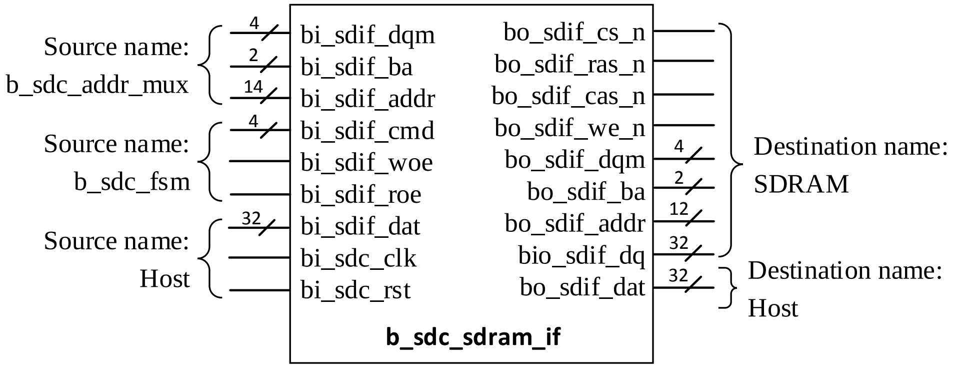


Figure 6.4.4: SDRAM Interface Block Diagram

6.4.4.1: I/O pin descriptions

Pin Name : bi_sdc_clk
Pin class: Global
Path: Host → SDRAM Interface
Description: Clock Input

Pin Name : bi_sdc_rst
Pin class: Global
Path: Host → SDRAM Interface
Description: Synchronous reset

Pin Name : bi_sdif_cmd
Pin class: Control
Path: Protocol Controller → SDRAM Interface
Description: This pin receives the command sent out by the Protocol Controller.

Pin Name : bi_sdif_dqm
Pin class: Control
Path: Address Multiplexer → SDRAM Interface
Description: This pin receives the data mask from the address multiplexer so that it can be passed to the SDRAM at the next negative edge of the clock through bo_sdr_dqm.

Pin Name : bi_sdif_ba
Pin class: Control
Path: Address Multiplexer → SDRAM Interface
Description: This pin receives the bank address.

Pin Name : bi_sdif_addr **Pin class:** Address **Path**: Address Multiplexer → SDRAM Interface **Description:** This pin receives the multiplexed SDRAM address.
Pin Name : bi_sdif_dat **Pin class:** Data **Path**: Host → SDRAM Interface **Description:** Host input data bus
Pin Name : bi_sdif_woe **Pin class:** Control **Path**: Protocol Controller → SDRAM Interface **Description:** Write output enable
Pin Name : bi_sdif_roe **Pin class:** Control **Path**: Protocol Controller → SDRAM Interface **Description:** Read output enable
Pin Name : bio_sdif_dq **Pin class:** data **Path**: SDRAM Interface → SDRAM SDRAM → SDRAM Interface **Description:** SDRAM bidirectional data bus
Pin Name : bo_sdif_we_n **Pin class:** Control **Path**: SDRAM Interface → SDRAM **Description:** This pin outputs the SDRAM write enable signal.
Pin Name : bo_sdif_dqm **Pin class:** Control **Path**: SDRAM Interface → SDRAM **Description:** This pin sends out the data line mask.
Pin Name : bo_sdif_ba **Pin class:** Control **Path**: SDRAM Interface → SDRAM **Description:** This pin sends out the SDRAM bank address.
Pin Name : bo_sdif_addr **Pin class:** Address **Path**: SDRAM Interface → SDRAM **Description:** This pin sends out the multiplexed address to the SDRAM
Pin Name : bo_sdif_dat **Pin class:** Data **Path**: SDRAM Interface →Host **Description:** SDRAM output data bus

Table 6.4.4.1: SDRAM Interface I/ O pin descriptions

Chapter 7: Test and Verification

7.1: SDRAM Controller

7.1.1: Test Plan

Function To be Tested	Test Case
Test 1: Reset and Initialization	ui_wb_rst is asserted to high at least one clock cycle
Test 2: Single WRITE (inactive banks)	for (i = 0; i < 4; i = i + 1)begin - load row address = 2 - load bank address = i - load column address = 15 - load select = 4'b1111 - load data = 2001 + i - execute "write" - execute "idle" end
Test 3 : Force Pre-charging Reset	ui_wb_rst is asserted to high at least one clock cycle after read or write.
Test 4: Single READ (inactive banks)	- execute "reset" ← to deactivate all activated banks for (i = 0; i < 4; i = i + 1)begin - load row address = 2 - load bank address = i - load column address = 15 - load select = 4'b1111 - load data = Hi-Z - execute "read" - execute "idle" end
Test 5: Single Write (active bank/ same row)	for (i = 0; i < 4; i = i + 1)begin - load row address = 2 - load bank address = i - load column address = 15 - load select = 4'b1111 - load data = 4000 + i - execute "write" - execute "idle" end
Test 6: Single READ (active bank/ same row)	for (i = 0; i < 4; i = i + 1)begin - load row address = 2 - load bank address = i - load column address = 15 - load select = 4'b1111 - load data = Hi-Z - execute "read" - execute "idle"

	end
Test 7: Single WRITE (active bank/ row differs)	for (i = 0; i < 4; i = i + 1)begin - load row address = 0 - load bank address = i - load column address = 4 - load select = 4’b1111 - load data = 6000 + i - execute “write” - execute “idle” end
Test 8: Programming Mode Register (Burst Length 8)	- load data = {22’d0, `WR_BRST, `OPMODE, `CAS_2, `BT_0, `BL_8} - execute “change mode”
Test 9: Burst Write 8	for(i = 0; i < 8; i= i+1)begin - load row address = 8; - load bank address = 1; - load column address = 9; - load select = 4'b1111; - load data = 32'd900 + i; - execute “write” end execute “idle”
Test 10: Same Programming Mode Register (Burst Length 8)	- execute “reset” - load data = {22’d0, `WR_BRST, `OPMODE, `CAS_2, `BT_0, `BL_8} - execute “change mode”
Test 11: Burst Read 8	for(i = 0; i < 8; i= i+1)begin - load row address = 8; - load bank address = 1; - load column address = 9; - load select = 4'b1111; - load data = Hi-Z - execute “read” end - execute “idle”
Test 12: Programming Mode Register (Burst Length 4)	- load data = {22’d0, `WR_BRST, `OPMODE, `CAS_2, `BT_0, `BL_4} - execute “change mode”
Test 13: Burst Read 4	for(i = 0; i < 4; i= i+1)begin - load row address = 4; - load bank address = 1; - load column address = 5; - load select = 4'b1111; - load data = Hi-Z - execute “read” end - execute “idle”
Test 14: Programming Mode Register (Burst Length 2)	- execute “reset” - load data = {22’d0, `WR_BRST,

	`OPMODE, `CAS_2, `BT_0, `BL_2} - execute “change mode”
Test 15: Burst Read 2	for(i = 0; i < 2; i= i+1)begin - load row address = 2; - load bank address = 1; - load column address = 3; - load select = 4'b1111; - load data = Hi-Z - execute “read” end - execute “idle”
Test 16: Programming Mode Register (Burst Length 1)	- load data = {22’d0, `WR_BRST, `OPMODE, `CAS_2, `BT_0, `BL_1} - execute “change mode”
Test 17: Burst Read 1	- load row address = 1; - load bank address = 1; - load column address = 2; - load select = 4'b1111; - load data = Hi-Z - execute “read” - execute “idle”
Test 18: Programming Mode Register (Default)	- load data = `DEFAULT_MODE - execute “change mode”
Test 19: Single READ (active bank/ row differs)	for (i = 0; i < 4; i = i + 1)begin - load row address = 2 - load bank address = i - load column address = 15 - load select = 4’b1111 - load data = Hi-Z - execute “read” - execute “idle” end
Test 20: Programming Mode Register (Burst Length 8)	- load data = {22’d0, `WR_BRST, `OPMODE, `CAS_2, `BT_0, `BL_8} - execute “change mode”
Test 21: Bus Termination (Write)	for(i = 0; i < 2; i= i+1)begin - load row address = 0; - load bank address = 0; - load column address = 11; - load select = 4’b1111; - load data = 11000 end - execute “idle”
Test 22: Bus Termination (Read)	for(i = 0; i < 2; i= i+1)begin - load row address = 8; - load bank address = 1; - load column address = 9; - load select = 4'b1111; - load data = 32'hz;

	- execute “read” end - execute “idle”
Test 23a: Data Masking Simulate “store byte”	for(i = 0 ; i < 5; i = i+1)begin - load row address = 8; - load bank address = 3; - load column address = 8; - load select = 4'b1111; - load data = 32'hffffffff; - execute “write” - load select = 4'h0; - if(i < 4) load select[i] = 1'b1; - load data = 0 - execute “write” end
Test 23b: Data Masking Simulate “store half”	for(i = 0 ; i < 2; i = i+1)begin - load row address = 8; - load bank address = 3; - load column address = 8; - load select = 4'b1111; - load data = 32'hffffffff; - execute “write” - load select = 0; if(i === 0) load select[1:0] = 2’b11 else if (i === 1) load select[3:2] = 3’b11 else load select = 0; - load data = 0; - execute “write” - execute “idle”
Test 24: Auto-Refresh	- execute “idle” - do nothing until Auto-refresh is requested

Table 7.1.1: SDRAM Controller Full Chip Test Plan

7.1.2: Testbench Verilog code

```
//############################################################
/*
Module       : tb_u_sdc_sdram
File name    : tb_u_sdc_sdram.v
Date Created : 21.1.2015
Author       : Chin Chun Lek
Code Type    : Verilog
Description  : Testbench for sdram controller and sdram
*/
//############################################################
`include "./util/sdc_macro.v"
`timescale 1ns / 10ps

module tb_u_sdc_sdram;

//SDRAM to CPU
reg                         tb_ui_clk;
reg                         tb_ui_rst;
reg                         tb_ui_host_ld_mode;
reg                         tb_ui_write,
                            tb_ui_read;
reg       [3:0]             tb_ui_sel;
reg       [31:0]            tb_ui_addr;
reg       [31:0]            tb_ui_data;
wire      [31:0]            tb_uo_data;
wire                        tb_uo_ack;

//between sdram controller and sdram
wire      [31:0]            u_sdc_dq;
wire      [11:0]            u_sdc_addr;
wire      [1:0]             u_sdc_ba;
wire                        u_sdc_cs_n;
wire                        u_sdc_ras_n;
wire                        u_sdc_cas_n;
wire                        u_sdc_we_n;
wire      [3:0]             u_sdc_dqm;

//display test status
reg       [255:0]   status;
integer i;

//To generate ASCII value in the waveform to ease debugging
bfm_wave_monitor bfm_monitor();

 u_sdram_controller u_sdram_controller
  (.ui_sdc_clk(tb_ui_clk),
   .ui_sdc_rst(tb_ui_rst),
   .ui_host_ld_mode(tb_ui_host_ld_mode),
   .ui_sdc_write(tb_ui_write),
   .ui_sdc_read(tb_ui_read),
   .ui_sdc_sel(tb_ui_sel),
   .ui_sdc_addr(tb_ui_addr),
   .ui_sdc_dat(tb_ui_data),
   .uo_sdc_dat(tb_uo_data),
   .uo_sdc_ack(tb_uo_ack),
   .uio_sdc_dq(u_sdc_dq),
   .uo_sdc_ba(u_sdc_ba),
   .uo_sdc_dqm(u_sdc_dqm),
   .uo_sdc_addr(u_sdc_addr),
   .uo_sdc_cs_n(u_sdc_cs_n),
   .uo_sdc_ras_n(u_sdc_ras_n),
   .uo_sdc_cas_n(u_sdc_cas_n),
   .uo_sdc_we_n(u_sdc_we_n) ) ;

  //MICRON SDRAM Instantiation
  mt48lc4m32b2 sdram(
  .Dq(u_sdc_dq),
  .Addr(u_sdc_addr),
  .Ba(u_sdc_ba),
  .Clk(tb_ui_clk),
  .Cke(1'b1), //cke always activated
  .Cs_n(u_sdc_cs_n),
  .Ras_n(u_sdc_ras_n),
  .Cas_n(u_sdc_cas_n),
  .We_n(u_sdc_we_n),
  .Dqm(u_sdc_dqm));

  //initialize clock signal
  initial tb_ui_clk = 1;
  always #10 tb_ui_clk = ~tb_ui_clk;

  initial begin
//**********Test 1: Reset and Initialization**********
$display("Test 1: Reset and Initialization");
//do idle
tb_ui_rst = 0;
tb_ui_write  = 0;
tb_ui_read = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
status  = "TEST 1: INIT";

//do reset
tb_ui_addr          = 32'b0;
tb_ui_data          = 32'b0;
tb_ui_sel           = 4'b1111;
tb_ui_rst = 1;
@(posedge tb_ui_clk)#1;

//do idle
tb_ui_rst = 0;
tb_ui_write  = 0;
tb_ui_read = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
            while(!u_sdram_controller.b_sdc_fsm.b_present[10])@(posedge tb_ui_clk);

//**********Test 2: Single WRITE into inactive banks**********
$display("Test 2: Single WRITE into inactive banks");
status  = "TEST 2: SWRITE - !BANK";
for(i = 0; i < 4; i= i+1)begin
            tb_ui_addr                  = 0;
            tb_ui_addr[23:12]           = 2;
            tb_ui_addr[11:10]           = i;
```

```
        tb_ui_addr[9:2]         = 15;
        tb_ui_sel               = 4'b1111;
        tb_ui_data              = 32'd2001 + i;

        //do write
        tb_ui_write = 1;

        //wait acknowledge
        while(~tb_uo_ack)       @(posedge tb_ui_clk);

        //do idle
        tb_ui_rst = 0;
        tb_ui_write  = 0;
        tb_ui_read  = 0;
        tb_ui_host_ld_mode = 0;
        @(posedge tb_ui_clk)#1;
end
//do idle
tb_ui_rst = 0;
tb_ui_write  = 0;
tb_ui_read  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
repeat(5) @(posedge tb_ui_clk);

//**********Test 3: Force Precharging Using Reset**********
$display("Test 3: Force Precharging Using Reset");
status  = "Test 3: Force Precharging Reset";
//do reset
tb_ui_addr      = 32'b0;
tb_ui_data      = 32'b0;
tb_ui_sel       = 4'b1111;
tb_ui_rst = 1;
@(posedge tb_ui_clk)#1;

//do idle
tb_ui_rst = 0;
tb_ui_write  = 0;
tb_ui_read  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;

while(!u_sdram_controller.b_sdc_fsm.b_present[10])@(posedge tb_ui_clk);

//********Test 4: Single READ from inactive banks********
$display("Test 4: Single READ from inactive banks");
        status  = "TEST 4: SREAD - !BANK";
for(i = 0; i < 4; i= i+1)begin
        tb_ui_addr              = 0;
        tb_ui_addr[23:12]       = 2;
        tb_ui_addr[11:10]       = i;
        tb_ui_addr[9:2]         = 15;
        tb_ui_sel               = 4'b1111;
        tb_ui_data              = 32'hz;

        //do read
        tb_ui_read = 1;

        //wait acknowledge
        while(~tb_uo_ack)          @(posedge tb_ui_clk);

        //do idle
        tb_ui_rst = 0;
        tb_ui_read  = 0;
        tb_ui_host_ld_mode = 0;
        @(posedge tb_ui_clk)#1;
end
//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
repeat(5) @(posedge tb_ui_clk);

//****Test 5: Single WRITE into active banks (same row)*****
$display("Test 5: Single WRITE into active banks (same row)");
        status  = "TEST 5: SWRITE - BANK ROW";
for(i = 0; i < 4; i= i+1)begin
        tb_ui_addr                 = 0;
        tb_ui_addr[23:12]          = 2;
        tb_ui_addr[11:10]          = i;
        tb_ui_addr[9:2]            = 15;
        tb_ui_sel                  = 4'b1111;
        tb_ui_data                 = 32'd4000 + i;

        //do write
        tb_ui_write = 1;

        //wait acknowledge
        while(~tb_uo_ack) @(posedge tb_ui_clk);
        //$display("ACK detected");

        //do idle
tb_ui_rst = 0;
tb_ui_write  = 0;
tb_ui_read  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
end
//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;

//****Test 6: Single READ from active banks (same row)******
$display("Test 6: Single READ from active banks (same row)");
        status  = "TEST 6: SREAD - BANK ROW";
for(i = 0; i < 4; i= i+1)begin
        tb_ui_addr                 = 0;
        tb_ui_addr[23:12]          = 2;
        tb_ui_addr[11:10]          = i;
        tb_ui_addr[9:2]            = 15;
        tb_ui_sel                  = 4'b1111;
        tb_ui_data                 = 32'hz;
        //do read
        tb_ui_read          = 1;

        //wait acknowledge
        while(~tb_uo_ack)@(posedge tb_ui_clk);

//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
end
//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;

//******Test 7: Single WRITE into active banks (row differs)*******
$display("Test 7: Single WRITE into active banks (row differs)");
        status  = "TEST 7: SWRITE - BANK !ROW";
for(i = 0; i < 4; i= i+1)begin
        tb_ui_addr                  = 0;
        tb_ui_addr[23:12]           = 3;
        tb_ui_addr[11:10]           = i;
        tb_ui_addr[9:2]             = 4;
        tb_ui_sel                   = 4'b1111;
        tb_ui_data                  = 32'h6000+i;

        //do write
        tb_ui_write = 1;

        //wait acknowledge
        while(~tb_uo_ack)           @(posedge tb_ui_clk);

//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
end
//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
repeat(5) @(posedge tb_ui_clk);

//*******Test 8: Programming Mode Register BL8**************
$display("Test 8: Programming Mode Register BL8");
        status  = "TEST 8: LMR BL8";
tb_ui_data          =           {20'b0, 2'b00,`WB_BRST, `OPMODE, `CAS_2, `BT_0, `BL_8};

//change mode
@(posedge tb_ui_clk)#1;
```

```
tb_ui_host_ld_mode =      1'b1;

//wait acknowledge
while(~tb_uo_ack)@(posedge tb_ui_clk);

//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;

//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
repeat(5) @(posedge tb_ui_clk);

//****************Test 9: Burst Write********************
$display("Test 9: Burst Write");
        status  = "TEST 9: BURST WRITE";
for(i = 0; i < 8; i= i+1)begin
        tb_ui_addr              = 0;
        tb_ui_addr[23:12]       = 8;
        tb_ui_addr[11:10]       = 1;
        tb_ui_addr[9:2]         = 9;
        tb_ui_sel               = 4'b1111;
        tb_ui_data              = 32'd900+i;

        //do write
        tb_ui_write = 1;

        //wait acknowledge
        @(posedge tb_ui_clk);
        while(~tb_uo_ack)       @(posedge tb_ui_clk);
end
//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
repeat(5) @(posedge tb_ui_clk);

//*******Test 10: Programming Mode Register BL8 same*************
$display("Test 10: Programming Mode Register BL8 same");
        status  = "TEST 10: LMR BL8 (same)";
tb_ui_data      =       {20'b0, 2'b00,`WB_BRST, `OPMODE, `CAS_2, `BT_0, `BL_8};

//change mode
@(posedge tb_ui_clk)#1;
tb_ui_host_ld_mode =      1'b1;

//wait acknowledge
while(~tb_uo_ack)@(posedge tb_ui_clk);

//do idle
```

```
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;

//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
repeat(5) @(posedge tb_ui_clk);

//******************Test 11: Burst READ 8****************
$display("Test 11: Burst READ 8");
        status  = "TEST 11: BURST READ 8";
for(i = 0; i < 8; i= i+1)begin
        tb_ui_addr              = 0;
        tb_ui_addr[23:12]       = 8;
        tb_ui_addr[11:10]       = 1;
        tb_ui_addr[9:2]         = 9;
        tb_ui_sel               = 4'b1111;
        tb_ui_data              = 32'hz;

        //do read
        tb_ui_read = 1;

        //wait acknowledge
        @(posedge tb_ui_clk);
        while(~tb_uo_ack)@(posedge tb_ui_clk);
end
//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
repeat(5) @(posedge tb_ui_clk);

//*******Test 12: Programming Mode Register BL4******************
$display("Test 12: Programming Mode Register BL4");
        status  = "TEST 12: LMR BL4";
tb_ui_data      =       {20'b0, 2'b00,`WB_BRST, `OPMODE, `CAS_2, `BT_0, `BL_4};

//change mode
@(posedge tb_ui_clk)#1;
tb_ui_host_ld_mode =    1'b1;

//wait acknowledge
while(~tb_uo_ack)@(posedge tb_ui_clk);

//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
```

```
//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
repeat(5) @(posedge tb_ui_clk);

//******************Test 13: Burst READ 4*******************
$display("Test 13: Burst READ 4");
            status  = "TEST 13: BURST READ 4";
for(i = 0; i < 4; i= i+1) begin
            tb_ui_addr                  = 0;
            tb_ui_addr[23:12]           = 8;
            tb_ui_addr[11:10]           = 1;
            tb_ui_addr[9:2]             = 9;
            tb_ui_sel                   = 4'b1111;
            tb_ui_data                  = 32'hz;

            //do read
            tb_ui_read = 1;

            //wait acknowledge
            @(posedge tb_ui_clk);
            while(~tb_uo_ack)@(posedge tb_ui_clk);
end
//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
repeat(5) @(posedge tb_ui_clk);

//*******Test 14: Programming Mode Register BL2******************
$display("Test 14: Programming Mode Register BL2");
            status  = "TEST 14: LMR BL2";
tb_ui_data          =           {20'b0, 2'b00,`WB_BRST, `OPMODE, `CAS_2, `BT_0, `BL_2};

//change mode
@(posedge tb_ui_clk)#1;
tb_ui_host_ld_mode =       1'b1;

//wait acknowledge
while(~tb_uo_ack)@(posedge tb_ui_clk);

//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;

//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
```

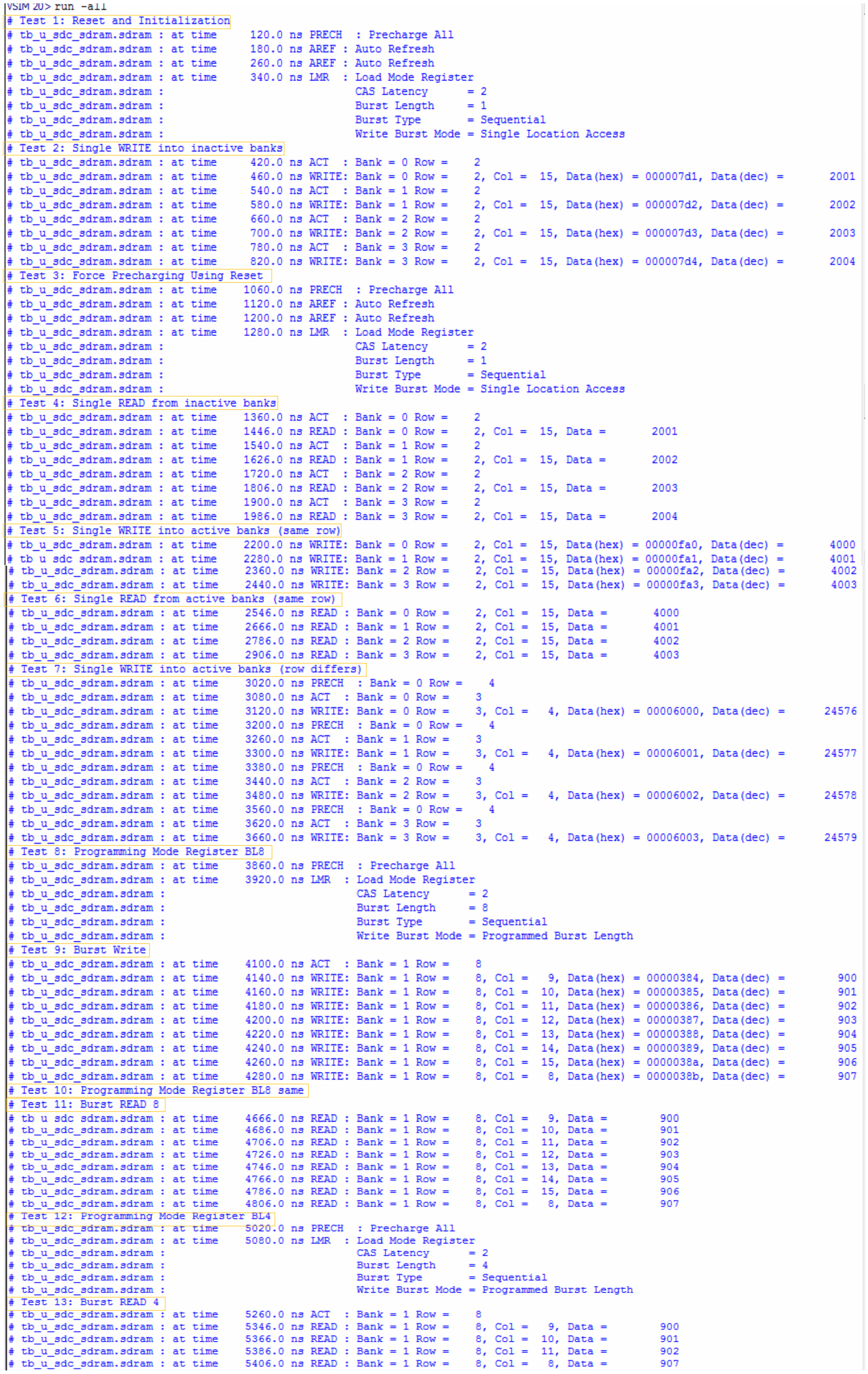
```
repeat(5) @(posedge tb_ui_clk);

		//******************Test 15: Burst READ 2*******************
$display("Test 15: Burst READ 2");
		status  = "TEST 15: BURST READ 2";
for(i = 0; i < 2; i= i+1)begin
		tb_ui_addr			= 0;
		tb_ui_addr[23:12]		= 8;
		tb_ui_addr[11:10]		= 1;
		tb_ui_addr[9:2]		= 9;
		tb_ui_sel			= 4'b1111;
		tb_ui_data			= 32'hz;

		//do read
		tb_ui_read = 1;

		//wait acknowledge
		@(posedge tb_ui_clk);
		while(~tb_uo_ack)@(posedge tb_ui_clk);
end
//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
repeat(5) @(posedge tb_ui_clk);

//*******Test 16: Programming Mode Register BL1******************
$display("Test 16: Programming Mode Register BL1");
		status  = "TEST 16: LMR BL1";
tb_ui_data		=		{20'b0, 2'b00,`WB_BRST, `OPMODE, `CAS_2, `BT_0, `BL_1};

//change mode
@(posedge tb_ui_clk)#1;
tb_ui_host_ld_mode =	1'b1;

//wait acknowledge
while(~tb_uo_ack)		@(posedge tb_ui_clk);

//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;

//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
repeat(5) @(posedge tb_ui_clk);

//******************Test 17: Burst READ 1*******************
$display("Test 17: Burst READ 1");
		status  = "TEST 17: BURST READ 1";
tb_ui_addr			= 0;
```

```
tb_ui_addr[23:12]          = 8;
tb_ui_addr[11:10]          = 1;
tb_ui_addr[9:2]            = 9;
tb_ui_sel                  = 4'b1111;
tb_ui_data                 = 32'hz;

//do read
tb_ui_read = 1;

//wait acknowledge
@(posedge tb_ui_clk);
while(~tb_uo_ack)@(posedge tb_ui_clk);

//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
repeat(5) @(posedge tb_ui_clk);

//**********Test 18: Programming Mode Register default**********
$display("Test 18: Programming Mode Register to default");
        status  = "TEST 18: LMR default";
tb_ui_data         =          `DEFAULT_MODE;

//change mode
@(posedge tb_ui_clk)#1;
tb_ui_host_ld_mode =       1'b1;

//wait acknowledge
while(~tb_uo_ack)          @(posedge tb_ui_clk);

//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;

//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
repeat(5) @(posedge tb_ui_clk);

//****Test 19: Single READ from active bank (row differs)********
$display("Test 19: Single READ from active bank (row differs)");
        status  = "TEST 19: SREAD - BANK !ROW";
for(i = 0; i < 4; i= i+1)begin
        tb_ui_addr                 = 0;
        tb_ui_addr[23:12]          = 2;
        tb_ui_addr[11:10]          = i;
        tb_ui_addr[9:2]            = 15;
        tb_ui_sel                  = 4'b1111;
        tb_ui_data                 = 32'hz;
        //do read
        tb_ui_read = 1;

        //wait acknowledge
        while(~tb_uo_ack)           @(posedge tb_ui_clk);

//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
end
//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
repeat(5) @(posedge tb_ui_clk);

//*******Test 20: Programming Mode Register BL8**********
$display("Test 20: Programming Mode Register BL8");
        status  = "TEST 20: LMR BL8";
tb_ui_data          =           {20'b0, 2'b00,`WB_BRST, `OPMODE, `CAS_2, `BT_0, `BL_8};

//change mode
@(posedge tb_ui_clk)#1;
tb_ui_host_ld_mode =        1'b1;

//wait acknowledge
while(~tb_uo_ack)           @(posedge tb_ui_clk);

//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;

//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
repeat(5) @(posedge tb_ui_clk);

//*****Test 21: Bus Termination for Write Cycle********
$display("Test 21: Bus Termination for Write Cycle");
        status  = "TEST 21: BT WRITE";
for(i = 0; i < 2; i= i+1)begin
        tb_ui_addr                  = 0;
        tb_ui_addr[23:12]           = 0;
        tb_ui_addr[11:10]           = 0;
        tb_ui_addr[9:2]             = 11;
        tb_ui_sel                   = 4'b1111;
        tb_ui_data                  = 32'd11000;
```

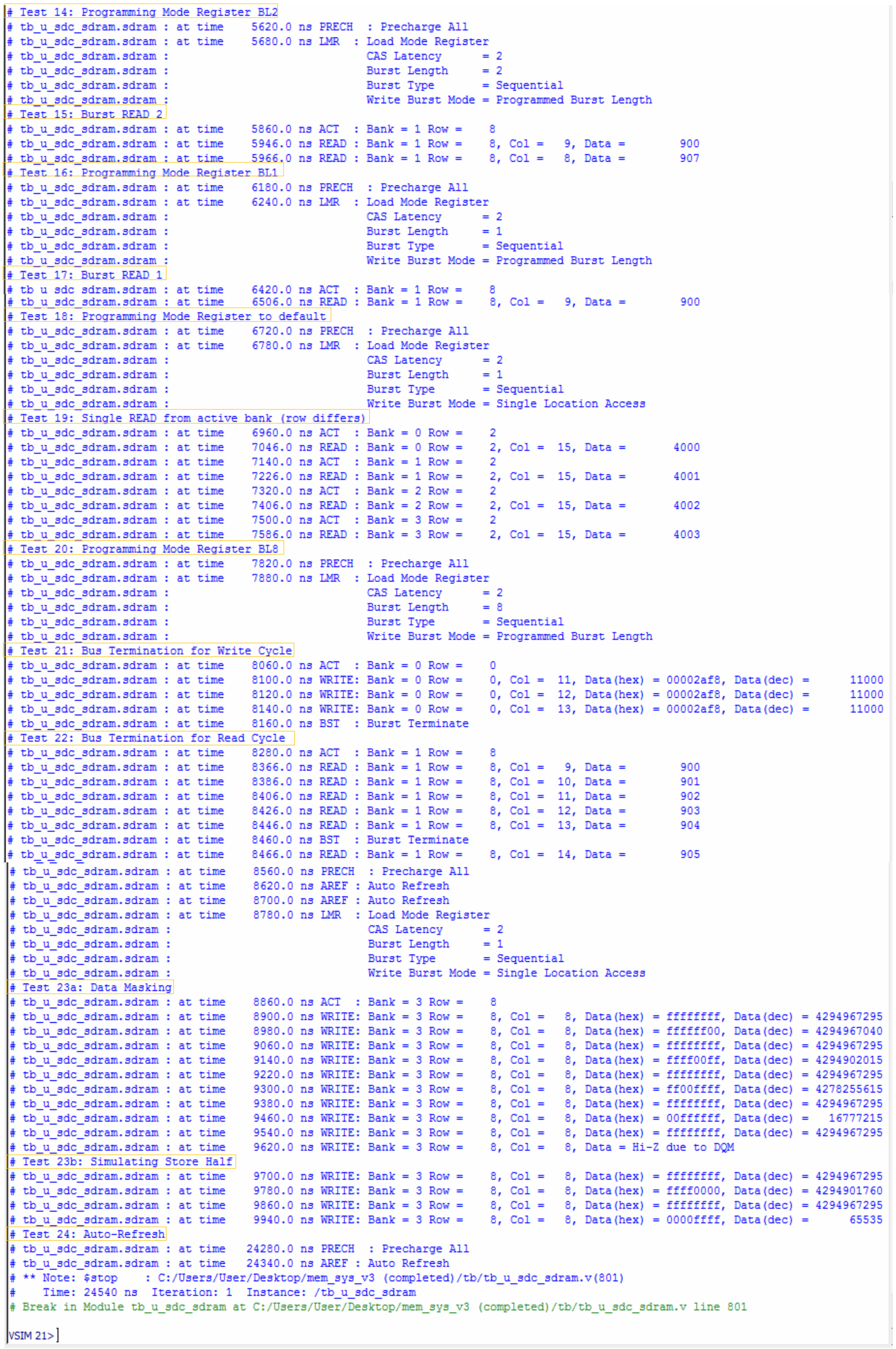
```
    //do write
    tb_ui_write = 1;

    //wait acknowledge
    @(posedge tb_ui_clk);
    while(~tb_uo_ack)       @(posedge tb_ui_clk);
end
//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
repeat(5) @(posedge tb_ui_clk);

//*********Test 22: Bus Termination for Read Cycle**********
$display("Test 22: Bus Termination for Read Cycle");
    status  = "TEST 22: BT READ";
for(i = 0; i < 2; i= i+1)begin
    tb_ui_addr              = 0;
    tb_ui_addr[23:12]       = 8;
    tb_ui_addr[11:10]       = 1;
    tb_ui_addr[9:2]         = 9;
    tb_ui_sel               = 4'b1111;
    tb_ui_data              = 32'hz;

    //do read
    tb_ui_read = 1;

    //wait acknowledge
    @(posedge tb_ui_clk);
    while(~tb_uo_ack)       @(posedge tb_ui_clk);
end
//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
@(posedge tb_ui_clk);

//Normally, masking is only used for single direct read or write. Since read is executed in block (burst)
to the cache, single write is used to simulate write through without buffer.

//do reset
tb_ui_addr      = 32'b0;
tb_ui_data      = 32'b0;
tb_ui_sel       = 4'b1111;
tb_ui_rst= 1;
@(posedge tb_ui_clk)#1;

//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
while(!u_sdram_controller.b_sdc_fsm.b_present[10])@(posedge tb_ui_clk);
```

```
//*****************Test 23: Data Masking********************
$display("Test 23a: Data Masking");
        status  = "TEST 23a: MASK Simulating Store Byte";
for(i = 0; i < 5; i= i+1)begin
        tb_ui_addr              = 0;
        tb_ui_addr[23:12]       = 8;
        tb_ui_addr[11:10]       = 3;
        tb_ui_addr[9:2]         = 8;
        tb_ui_sel               = 4'b1111;
        tb_ui_data              = 32'hffffffff;

        //do write
        tb_ui_write = 1;

        //wait acknowledge
        @(posedge tb_ui_clk);
        while(~tb_uo_ack)@(posedge tb_ui_clk);
        @(posedge tb_ui_clk);

        tb_ui_addr              = 0;
        tb_ui_addr[23:12]       = 8;
        tb_ui_addr[11:10]       = 3;
        tb_ui_addr[9:2]         = 8;
        tb_ui_sel               = 4'h0;
        if(i < 4)
                tb_ui_sel[i]    = 1'b1;
        else
                tb_ui_sel       = 4'h0;
        tb_ui_data = 32'd0;

        //do write
        tb_ui_write = 1;

        //wait acknowledge
        @(posedge tb_ui_clk);
        while(~tb_uo_ack)@(posedge tb_ui_clk);
        @(posedge tb_ui_clk);
end

$display("Test 23b: Simulating Store Half");
        status  = "Test 23b: Simulating Store Half";
for(i = 0; i < 2; i= i+1)begin
        tb_ui_addr              = 0;
        tb_ui_addr[23:12]       = 8;
        tb_ui_addr[11:10]       = 3;
        tb_ui_addr[9:2]         = 8;
        tb_ui_sel               = 4'b1111;
        tb_ui_data              = 32'hffffffff;

        //do write
        tb_ui_write = 1;

        //wait acknowledge
        @(posedge tb_ui_clk);
        while(~tb_uo_ack)@(posedge tb_ui_clk);
        @(posedge tb_ui_clk);

        tb_ui_addr              = 0;
        tb_ui_addr[23:12]       = 8;
        tb_ui_addr[11:10]           = 3;
        tb_ui_addr[9:2]             = 8;
        tb_ui_sel                   = 4'h0;
        if(i === 0)
                tb_ui_sel[1:0]      = 2'b11;
        else if(i === 1)
                tb_ui_sel[3:2]      = 2'b11;
        else
                tb_ui_sel           = 4'h0;
        tb_ui_data = 32'd0;

        //do write
        tb_ui_write = 1;

        //wait acknowledge
        @(posedge tb_ui_clk);
        while(~tb_uo_ack)@(posedge tb_ui_clk);
        @(posedge tb_ui_clk);
end
//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;

//****************Test 24: Auto-Refresh********************
$display("Test 24: Auto-Refresh");
status  = "TEST 24: AREF";
//do idle
tb_ui_rst = 0;
tb_ui_read  = 0;
tb_ui_write  = 0;
tb_ui_host_ld_mode = 0;
@(posedge tb_ui_clk)#1;
while(!u_sdram_controller.b_sdc_fsm.b_present[11])@(posedge tb_ui_clk);
repeat(10) @(posedge tb_ui_clk);


$stop;
  end


endmodule
```

7.1.3: Verification Result



```
VSIM 20> run -all
# Test 1: Reset and Initialization
# tb_u_sdc_sdram.sdram : at time     120.0 ns PRECH  : Precharge All
# tb_u_sdc_sdram.sdram : at time     180.0 ns AREF : Auto Refresh
# tb_u_sdc_sdram.sdram : at time     260.0 ns AREF : Auto Refresh
# tb_u_sdc_sdram.sdram : at time     340.0 ns LMR  : Load Mode Register
# tb_u_sdc_sdram.sdram :                            CAS Latency      = 2
# tb_u_sdc_sdram.sdram :                            Burst Length     = 1
# tb_u_sdc_sdram.sdram :                            Burst Type       = Sequential
# tb_u_sdc_sdram.sdram :                            Write Burst Mode = Single Location Access
# Test 2: Single WRITE into inactive banks
# tb_u_sdc_sdram.sdram : at time     420.0 ns ACT  : Bank = 0 Row =    2
# tb_u_sdc_sdram.sdram : at time     460.0 ns WRITE: Bank = 0 Row =    2, Col =  15, Data(hex) = 000007d1, Data(dec) =       2001
# tb_u_sdc_sdram.sdram : at time     540.0 ns ACT  : Bank = 1 Row =    2
# tb_u_sdc_sdram.sdram : at time     580.0 ns WRITE: Bank = 1 Row =    2, Col =  15, Data(hex) = 000007d2, Data(dec) =       2002
# tb_u_sdc_sdram.sdram : at time     660.0 ns ACT  : Bank = 2 Row =    2
# tb_u_sdc_sdram.sdram : at time     700.0 ns WRITE: Bank = 2 Row =    2, Col =  15, Data(hex) = 000007d3, Data(dec) =       2003
# tb_u_sdc_sdram.sdram : at time     780.0 ns ACT  : Bank = 3 Row =    2
# tb_u_sdc_sdram.sdram : at time     820.0 ns WRITE: Bank = 3 Row =    2, Col =  15, Data(hex) = 000007d4, Data(dec) =       2004
# Test 3: Force Precharging Using Reset
# tb_u_sdc_sdram.sdram : at time    1060.0 ns PRECH  : Precharge All
# tb_u_sdc_sdram.sdram : at time    1120.0 ns AREF : Auto Refresh
# tb_u_sdc_sdram.sdram : at time    1200.0 ns AREF : Auto Refresh
# tb_u_sdc_sdram.sdram : at time    1280.0 ns LMR  : Load Mode Register
# tb_u_sdc_sdram.sdram :                            CAS Latency      = 2
# tb_u_sdc_sdram.sdram :                            Burst Length     = 1
# tb_u_sdc_sdram.sdram :                            Burst Type       = Sequential
# tb_u_sdc_sdram.sdram :                            Write Burst Mode = Single Location Access
# Test 4: Single READ from inactive banks
# tb_u_sdc_sdram.sdram : at time    1360.0 ns ACT  : Bank = 0 Row =    2
# tb_u_sdc_sdram.sdram : at time    1446.0 ns READ : Bank = 0 Row =    2, Col =  15, Data =       2001
# tb_u_sdc_sdram.sdram : at time    1540.0 ns ACT  : Bank = 1 Row =    2
# tb_u_sdc_sdram.sdram : at time    1626.0 ns READ : Bank = 1 Row =    2, Col =  15, Data =       2002
# tb_u_sdc_sdram.sdram : at time    1720.0 ns ACT  : Bank = 2 Row =    2
# tb_u_sdc_sdram.sdram : at time    1806.0 ns READ : Bank = 2 Row =    2, Col =  15, Data =       2003
# tb_u_sdc_sdram.sdram : at time    1900.0 ns ACT  : Bank = 3 Row =    2
# tb_u_sdc_sdram.sdram : at time    1986.0 ns READ : Bank = 3 Row =    2, Col =  15, Data =       2004
# Test 5: Single WRITE into active banks (same row)
# tb_u_sdc_sdram.sdram : at time    2200.0 ns WRITE: Bank = 0 Row =    2, Col =  15, Data(hex) = 00000fa0, Data(dec) =       4000
# tb u sdc sdram.sdram : at time    2280.0 ns WRITE: Bank = 1 Row =    2, Col =  15, Data(hex) = 00000fa1, Data(dec) =       4001
# tb_u_sdc_sdram.sdram : at time    2360.0 ns WRITE: Bank = 2 Row =    2, Col =  15, Data(hex) = 00000fa2, Data(dec) =       4002
# tb_u_sdc_sdram.sdram : at time    2440.0 ns WRITE: Bank = 3 Row =    2, Col =  15, Data(hex) = 00000fa3, Data(dec) =       4003
# Test 6: Single READ from active banks (same row)
# tb_u_sdc_sdram.sdram : at time    2546.0 ns READ : Bank = 0 Row =    2, Col =  15, Data =       4000
# tb_u_sdc_sdram.sdram : at time    2666.0 ns READ : Bank = 1 Row =    2, Col =  15, Data =       4001
# tb_u_sdc_sdram.sdram : at time    2786.0 ns READ : Bank = 2 Row =    2, Col =  15, Data =       4002
# tb_u_sdc_sdram.sdram : at time    2906.0 ns READ : Bank = 3 Row =    2, Col =  15, Data =       4003
# Test 7: Single WRITE into active banks (row differs)
# tb_u_sdc_sdram.sdram : at time    3020.0 ns PRECH  : Bank = 0 Row =    4
# tb_u_sdc_sdram.sdram : at time    3080.0 ns ACT  : Bank = 0 Row =    3
# tb_u_sdc_sdram.sdram : at time    3120.0 ns WRITE: Bank = 0 Row =    3, Col =   4, Data(hex) = 00006000, Data(dec) =      24576
# tb_u_sdc_sdram.sdram : at time    3200.0 ns PRECH  : Bank = 0 Row =    4
# tb_u_sdc_sdram.sdram : at time    3260.0 ns ACT  : Bank = 1 Row =    3
# tb_u_sdc_sdram.sdram : at time    3300.0 ns WRITE: Bank = 1 Row =    3, Col =   4, Data(hex) = 00006001, Data(dec) =      24577
# tb_u_sdc_sdram.sdram : at time    3380.0 ns PRECH  : Bank = 0 Row =    4
# tb_u_sdc_sdram.sdram : at time    3440.0 ns ACT  : Bank = 2 Row =    3
# tb_u_sdc_sdram.sdram : at time    3480.0 ns WRITE: Bank = 2 Row =    3, Col =   4, Data(hex) = 00006002, Data(dec) =      24578
# tb_u_sdc_sdram.sdram : at time    3560.0 ns PRECH  : Bank = 0 Row =    4
# tb_u_sdc_sdram.sdram : at time    3620.0 ns ACT  : Bank = 3 Row =    3
# tb_u_sdc_sdram.sdram : at time    3660.0 ns WRITE: Bank = 3 Row =    3, Col =   4, Data(hex) = 00006003, Data(dec) =      24579
# Test 8: Programming Mode Register BL8
# tb_u_sdc_sdram.sdram : at time    3860.0 ns PRECH  : Precharge All
# tb_u_sdc_sdram.sdram : at time    3920.0 ns LMR  : Load Mode Register
# tb_u_sdc_sdram.sdram :                            CAS Latency      = 2
# tb_u_sdc_sdram.sdram :                            Burst Length     = 8
# tb_u_sdc_sdram.sdram :                            Burst Type       = Sequential
# tb_u_sdc_sdram.sdram :                            Write Burst Mode = Programmed Burst Length
# Test 9: Burst Write
# tb_u_sdc_sdram.sdram : at time    4100.0 ns ACT  : Bank = 1 Row =    8
# tb_u_sdc_sdram.sdram : at time    4140.0 ns WRITE: Bank = 1 Row =    8, Col =   9, Data(hex) = 00000384, Data(dec) =        900
# tb_u_sdc_sdram.sdram : at time    4160.0 ns WRITE: Bank = 1 Row =    8, Col =  10, Data(hex) = 00000385, Data(dec) =        901
# tb_u_sdc_sdram.sdram : at time    4180.0 ns WRITE: Bank = 1 Row =    8, Col =  11, Data(hex) = 00000386, Data(dec) =        902
# tb_u_sdc_sdram.sdram : at time    4200.0 ns WRITE: Bank = 1 Row =    8, Col =  12, Data(hex) = 00000387, Data(dec) =        903
# tb_u_sdc_sdram.sdram : at time    4220.0 ns WRITE: Bank = 1 Row =    8, Col =  13, Data(hex) = 00000388, Data(dec) =        904
# tb_u_sdc_sdram.sdram : at time    4240.0 ns WRITE: Bank = 1 Row =    8, Col =  14, Data(hex) = 00000389, Data(dec) =        905
# tb_u_sdc_sdram.sdram : at time    4260.0 ns WRITE: Bank = 1 Row =    8, Col =  15, Data(hex) = 0000038a, Data(dec) =        906
# tb_u_sdc_sdram.sdram : at time    4280.0 ns WRITE: Bank = 1 Row =    8, Col =   8, Data(hex) = 0000038b, Data(dec) =        907
# Test 10: Programming Mode Register BL8 same
# Test 11: Burst READ 8
# tb u sdc sdram.sdram : at time    4666.0 ns READ : Bank = 1 Row =    8, Col =   9, Data =        900
# tb_u_sdc_sdram.sdram : at time    4686.0 ns READ : Bank = 1 Row =    8, Col =  10, Data =        901
# tb_u_sdc_sdram.sdram : at time    4706.0 ns READ : Bank = 1 Row =    8, Col =  11, Data =        902
# tb_u_sdc_sdram.sdram : at time    4726.0 ns READ : Bank = 1 Row =    8, Col =  12, Data =        903
# tb_u_sdc_sdram.sdram : at time    4746.0 ns READ : Bank = 1 Row =    8, Col =  13, Data =        904
# tb_u_sdc_sdram.sdram : at time    4766.0 ns READ : Bank = 1 Row =    8, Col =  14, Data =        905
# tb_u_sdc_sdram.sdram : at time    4786.0 ns READ : Bank = 1 Row =    8, Col =  15, Data =        906
# tb_u_sdc_sdram.sdram : at time    4806.0 ns READ : Bank = 1 Row =    8, Col =   8, Data =        907
# Test 12: Programming Mode Register BL4
# tb_u_sdc_sdram.sdram : at time    5020.0 ns PRECH  : Precharge All
# tb_u_sdc_sdram.sdram : at time    5080.0 ns LMR  : Load Mode Register
# tb_u_sdc_sdram.sdram :                            CAS Latency      = 2
# tb_u_sdc_sdram.sdram :                            Burst Length     = 4
# tb_u_sdc_sdram.sdram :                            Burst Type       = Sequential
# tb_u_sdc_sdram.sdram :                            Write Burst Mode = Programmed Burst Length
# Test 13: Burst READ 4
# tb_u_sdc_sdram.sdram : at time    5260.0 ns ACT  : Bank = 1 Row =    8
# tb_u_sdc_sdram.sdram : at time    5346.0 ns READ : Bank = 1 Row =    8, Col =   9, Data =        900
# tb_u_sdc_sdram.sdram : at time    5366.0 ns READ : Bank = 1 Row =    8, Col =  10, Data =        901
# tb_u_sdc_sdram.sdram : at time    5386.0 ns READ : Bank = 1 Row =    8, Col =  11, Data =        902
# tb_u_sdc_sdram.sdram : at time    5406.0 ns READ : Bank = 1 Row =    8, Col =   8, Data =        907
```

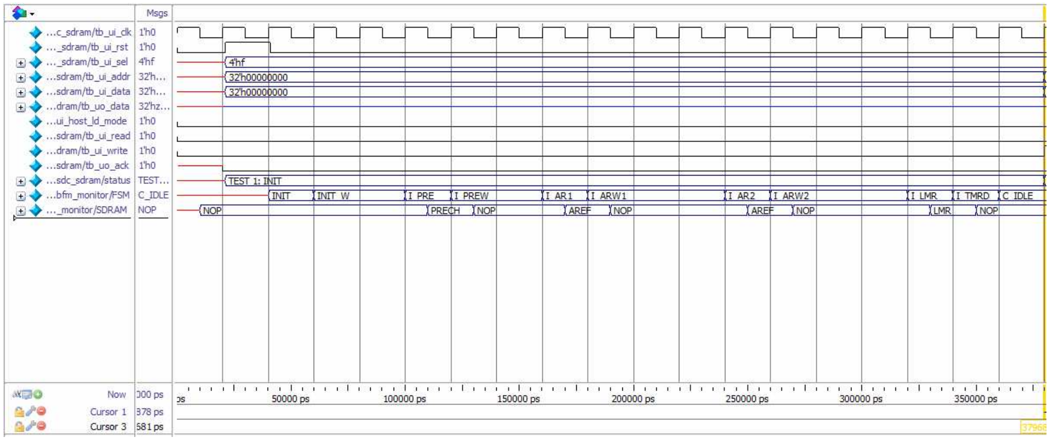


```
# Test 14: Programming Mode Register BL2
# tb_u_sdc_sdram.sdram : at time    5620.0 ns PRECH  : Precharge All
# tb_u_sdc_sdram.sdram : at time    5680.0 ns LMR  : Load Mode Register
# tb_u_sdc_sdram.sdram :                             CAS Latency      = 2
# tb_u_sdc_sdram.sdram :                             Burst Length     = 2
# tb_u_sdc_sdram.sdram :                             Burst Type       = Sequential
# tb_u_sdc_sdram.sdram :                             Write Burst Mode = Programmed Burst Length
# Test 15: Burst READ 2
# tb_u_sdc_sdram.sdram : at time    5860.0 ns ACT  : Bank = 1 Row =    8
# tb_u_sdc_sdram.sdram : at time    5946.0 ns READ : Bank = 1 Row =    8, Col =   9, Data =        900
# tb_u_sdc_sdram.sdram : at time    5966.0 ns READ : Bank = 1 Row =    8, Col =   8, Data =        907
# Test 16: Programming Mode Register BL1
# tb_u_sdc_sdram.sdram : at time    6180.0 ns PRECH  : Precharge All
# tb_u_sdc_sdram.sdram : at time    6240.0 ns LMR  : Load Mode Register
# tb_u_sdc_sdram.sdram :                             CAS Latency      = 2
# tb_u_sdc_sdram.sdram :                             Burst Length     = 1
# tb_u_sdc_sdram.sdram :                             Burst Type       = Sequential
# tb_u_sdc_sdram.sdram :                             Write Burst Mode = Programmed Burst Length
# Test 17: Burst READ 1
# tb u sdc sdram.sdram : at time    6420.0 ns ACT  : Bank = 1 Row =    8
# tb_u_sdc_sdram.sdram : at time    6506.0 ns READ : Bank = 1 Row =    8, Col =   9, Data =        900
# Test 18: Programming Mode Register to default
# tb_u_sdc_sdram.sdram : at time    6720.0 ns PRECH  : Precharge All
# tb_u_sdc_sdram.sdram : at time    6780.0 ns LMR  : Load Mode Register
# tb_u_sdc_sdram.sdram :                             CAS Latency      = 2
# tb_u_sdc_sdram.sdram :                             Burst Length     = 1
# tb_u_sdc_sdram.sdram :                             Burst Type       = Sequential
# tb_u_sdc_sdram.sdram :                             Write Burst Mode = Single Location Access
# Test 19: Single READ from active bank (row differs)
# tb_u_sdc_sdram.sdram : at time    6960.0 ns ACT  : Bank = 0 Row =    2
# tb_u_sdc_sdram.sdram : at time    7046.0 ns READ : Bank = 0 Row =    2, Col =  15, Data =       4000
# tb_u_sdc_sdram.sdram : at time    7140.0 ns ACT  : Bank = 1 Row =    2
# tb_u_sdc_sdram.sdram : at time    7226.0 ns READ : Bank = 1 Row =    2, Col =  15, Data =       4001
# tb_u_sdc_sdram.sdram : at time    7320.0 ns ACT  : Bank = 2 Row =    2
# tb_u_sdc_sdram.sdram : at time    7406.0 ns READ : Bank = 2 Row =    2, Col =  15, Data =       4002
# tb_u_sdc_sdram.sdram : at time    7500.0 ns ACT  : Bank = 3 Row =    2
# tb_u_sdc_sdram.sdram : at time    7586.0 ns READ : Bank = 3 Row =    2, Col =  15, Data =       4003
# Test 20: Programming Mode Register BL8
# tb_u_sdc_sdram.sdram : at time    7820.0 ns PRECH  : Precharge All
# tb_u_sdc_sdram.sdram : at time    7880.0 ns LMR  : Load Mode Register
# tb_u_sdc_sdram.sdram :                             CAS Latency      = 2
# tb_u_sdc_sdram.sdram :                             Burst Length     = 8
# tb_u_sdc_sdram.sdram :                             Burst Type       = Sequential
# tb_u_sdc_sdram.sdram :                             Write Burst Mode = Programmed Burst Length
# Test 21: Bus Termination for Write Cycle
# tb_u_sdc_sdram.sdram : at time    8060.0 ns ACT  : Bank = 0 Row =    0
# tb_u_sdc_sdram.sdram : at time    8100.0 ns WRITE: Bank = 0 Row =    0, Col =  11, Data(hex) = 00002af8, Data(dec) =      11000
# tb_u_sdc_sdram.sdram : at time    8120.0 ns WRITE: Bank = 0 Row =    0, Col =  12, Data(hex) = 00002af8, Data(dec) =      11000
# tb_u_sdc_sdram.sdram : at time    8140.0 ns WRITE: Bank = 0 Row =    0, Col =  13, Data(hex) = 00002af8, Data(dec) =      11000
# tb_u_sdc_sdram.sdram : at time    8160.0 ns BST  : Burst Terminate
# Test 22: Bus Termination for Read Cycle
# tb_u_sdc_sdram.sdram : at time    8280.0 ns ACT  : Bank = 1 Row =    8
# tb_u_sdc_sdram.sdram : at time    8366.0 ns READ : Bank = 1 Row =    8, Col =   9, Data =        900
# tb_u_sdc_sdram.sdram : at time    8386.0 ns READ : Bank = 1 Row =    8, Col =  10, Data =        901
# tb_u_sdc_sdram.sdram : at time    8406.0 ns READ : Bank = 1 Row =    8, Col =  11, Data =        902
# tb_u_sdc_sdram.sdram : at time    8426.0 ns READ : Bank = 1 Row =    8, Col =  12, Data =        903
# tb_u_sdc_sdram.sdram : at time    8446.0 ns READ : Bank = 1 Row =    8, Col =  13, Data =        904
# tb_u_sdc_sdram.sdram : at time    8460.0 ns BST  : Burst Terminate
# tb_u_sdc_sdram.sdram : at time    8466.0 ns READ : Bank = 1 Row =    8, Col =  14, Data =        905
# tb_u_sdc_sdram.sdram : at time    8560.0 ns PRECH  : Precharge All
# tb_u_sdc_sdram.sdram : at time    8620.0 ns AREF : Auto Refresh
# tb_u_sdc_sdram.sdram : at time    8700.0 ns AREF : Auto Refresh
# tb_u_sdc_sdram.sdram : at time    8780.0 ns LMR  : Load Mode Register
# tb_u_sdc_sdram.sdram :                             CAS Latency      = 2
# tb_u_sdc_sdram.sdram :                             Burst Length     = 1
# tb_u_sdc_sdram.sdram :                             Burst Type       = Sequential
# tb_u_sdc_sdram.sdram :                             Write Burst Mode = Single Location Access
# Test 23a: Data Masking
# tb_u_sdc_sdram.sdram : at time    8860.0 ns ACT  : Bank = 3 Row =    8
# tb_u_sdc_sdram.sdram : at time    8900.0 ns WRITE: Bank = 3 Row =    8, Col =   8, Data(hex) = ffffffff, Data(dec) = 4294967295
# tb_u_sdc_sdram.sdram : at time    8980.0 ns WRITE: Bank = 3 Row =    8, Col =   8, Data(hex) = ffffff00, Data(dec) = 4294967040
# tb_u_sdc_sdram.sdram : at time    9060.0 ns WRITE: Bank = 3 Row =    8, Col =   8, Data(hex) = ffffffff, Data(dec) = 4294967295
# tb_u_sdc_sdram.sdram : at time    9140.0 ns WRITE: Bank = 3 Row =    8, Col =   8, Data(hex) = ffff00ff, Data(dec) = 4294902015
# tb_u_sdc_sdram.sdram : at time    9220.0 ns WRITE: Bank = 3 Row =    8, Col =   8, Data(hex) = ffffffff, Data(dec) = 4294967295
# tb_u_sdc_sdram.sdram : at time    9300.0 ns WRITE: Bank = 3 Row =    8, Col =   8, Data(hex) = ff00ffff, Data(dec) = 4278255615
# tb_u_sdc_sdram.sdram : at time    9380.0 ns WRITE: Bank = 3 Row =    8, Col =   8, Data(hex) = ffffffff, Data(dec) = 4294967295
# tb_u_sdc_sdram.sdram : at time    9460.0 ns WRITE: Bank = 3 Row =    8, Col =   8, Data(hex) = 00ffffff, Data(dec) =   16777215
# tb_u_sdc_sdram.sdram : at time    9540.0 ns WRITE: Bank = 3 Row =    8, Col =   8, Data(hex) = ffffffff, Data(dec) = 4294967295
# tb_u_sdc_sdram.sdram : at time    9620.0 ns WRITE: Bank = 3 Row =    8, Col =   8, Data = Hi-Z due to DQM
# Test 23b: Simulating Store Half
# tb_u_sdc_sdram.sdram : at time    9700.0 ns WRITE: Bank = 3 Row =    8, Col =   8, Data(hex) = ffffffff, Data(dec) = 4294967295
# tb_u_sdc_sdram.sdram : at time    9780.0 ns WRITE: Bank = 3 Row =    8, Col =   8, Data(hex) = ffff0000, Data(dec) = 4294901760
# tb_u_sdc_sdram.sdram : at time    9860.0 ns WRITE: Bank = 3 Row =    8, Col =   8, Data(hex) = ffffffff, Data(dec) = 4294967295
# tb_u_sdc_sdram.sdram : at time    9940.0 ns WRITE: Bank = 3 Row =    8, Col =   8, Data(hex) = 0000ffff, Data(dec) =      65535
# Test 24: Auto-Refresh
# tb_u_sdc_sdram.sdram : at time   24280.0 ns PRECH  : Precharge All
# tb_u_sdc_sdram.sdram : at time   24340.0 ns AREF : Auto Refresh
# ** Note: $stop    : C:/Users/User/Desktop/mem_sys_v3 (completed)/tb/tb_u_sdc_sdram.v(801)
#    Time: 24540 ns  Iteration: 1  Instance: /tb_u_sdc_sdram
# Break in Module tb_u_sdc_sdram at C:/Users/User/Desktop/mem_sys_v3 (completed)/tb/tb_u_sdc_sdram.v line 801

VSIM 21>
```

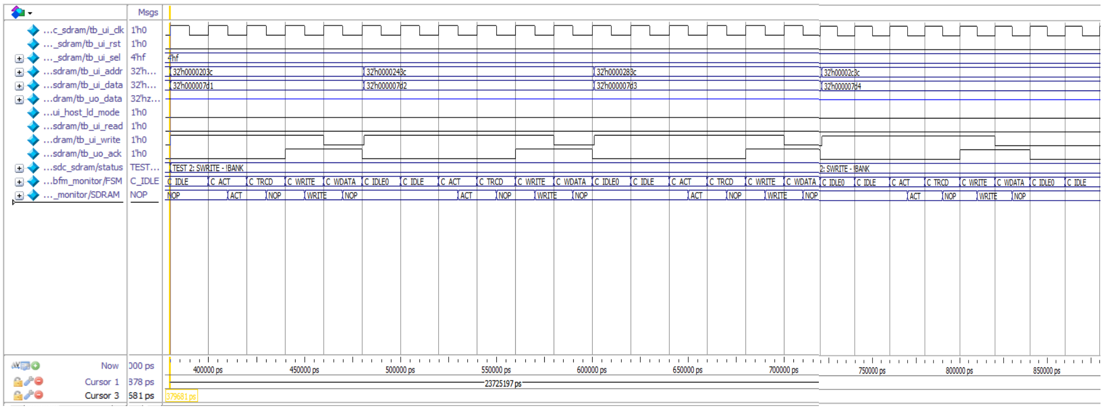
Figure 7.1.2 SDRAM Controller Verification Result

7.1.3: Simulation Result (Timing Diagram)

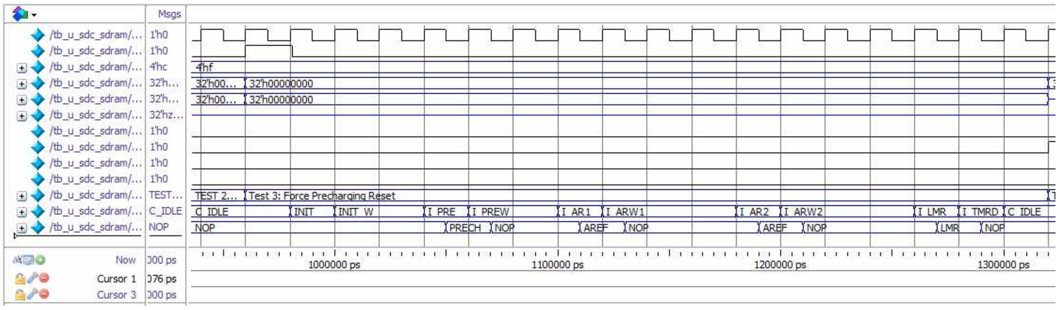
Result 1: Initialization and Reset



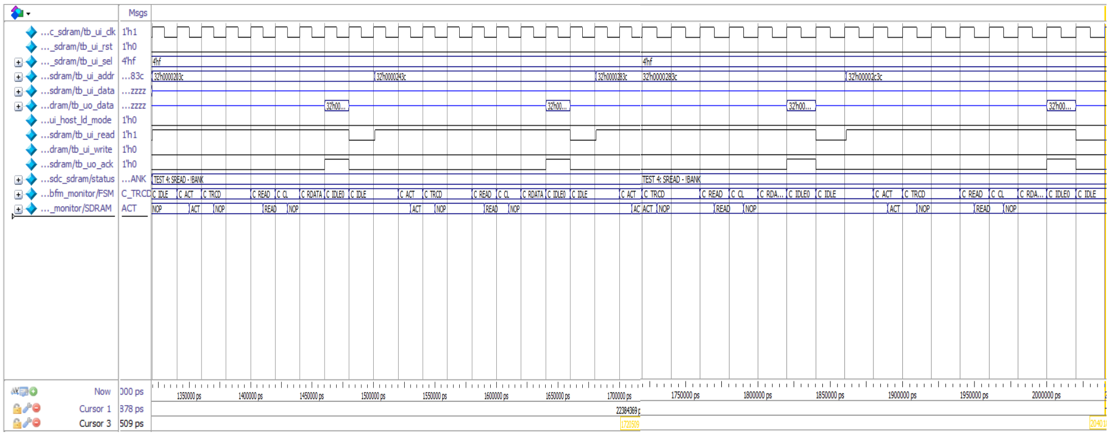
Result 2: Single Write and inactive Bank



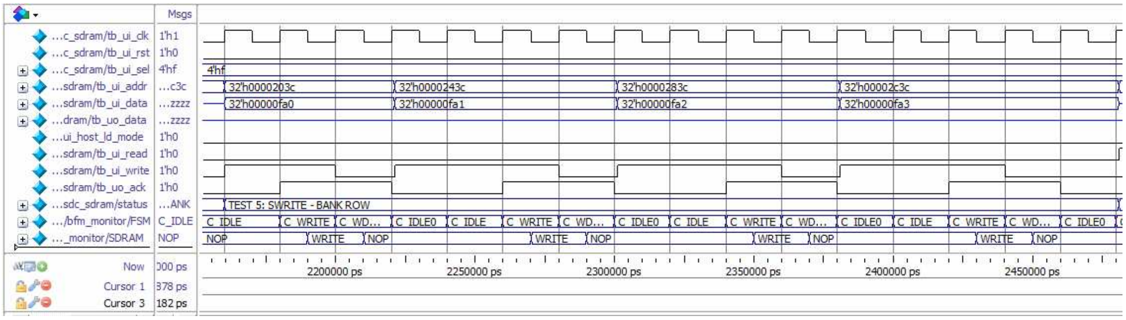
Result 3: Force Pre-charging reset



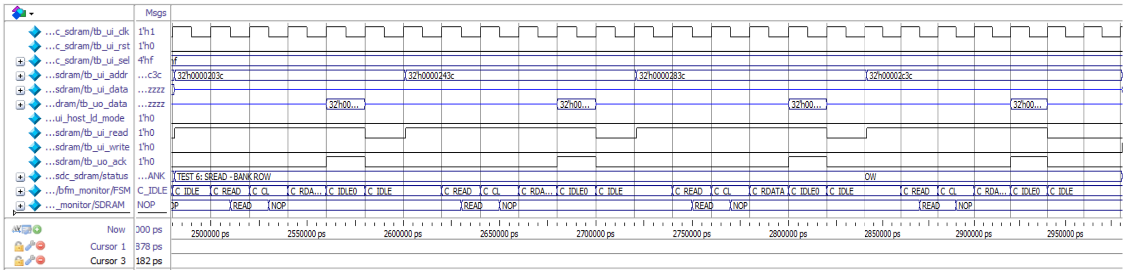
Result 4: Single Read and inactive Bank



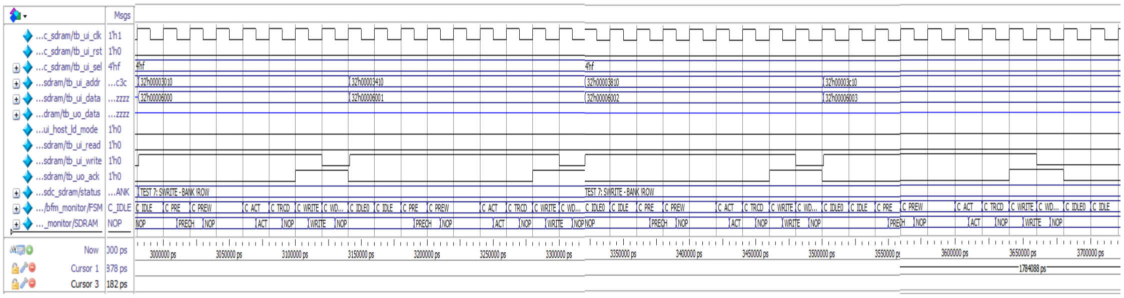
Result 5: Single Write and Active Bank (Same Row)



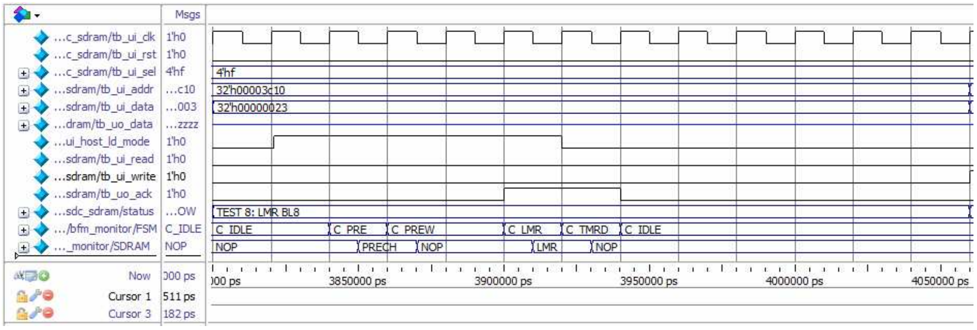
Result 6: Single Read and Active Bank (Same Row)



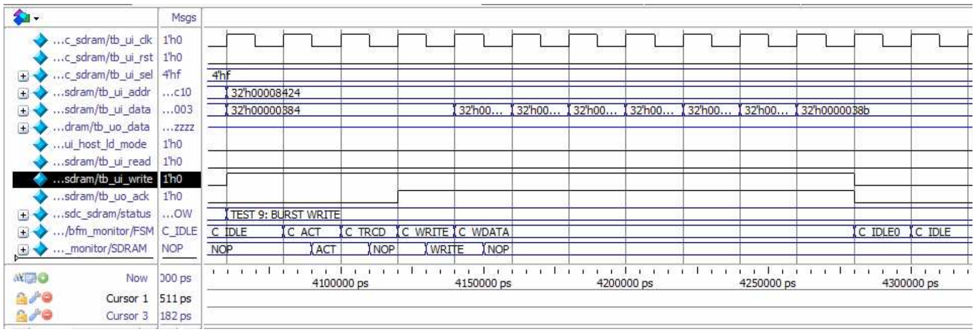
Result 7: Single Write and Active Bank (Different Row)


Msgs
...c_sdram/tb_ui_clk 1'h1
...c_sdram/tb_ui_rst 1'h0
...c_sdram/tb_ui_sel 4'hf
...sdram/tb_ui_addr ...c3c
...sdram/tb_ui_data ...zzzz
...dram/tb_uo_data ...zzzz
...ui_host_ld_mode 1'h0
...sdram/tb_ui_read 1'h0
...sdram/tb_ui_write 1'h0
...sdram/tb_uo_ack 1'h0
...sdc_sdram/status ...ANK
.../bfm_monitor/FSM C_IDLE
..._monitor/SDRAM NOP
TEST 7: SWRITE - BANK ROW
Now
Cursor 1
Cursor 3

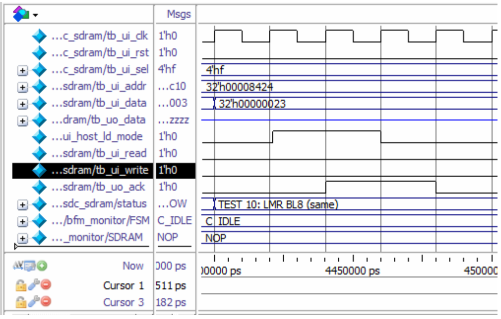
Result 8: Programming Load Mode Register (Burst length 8)


Msgs
...c_sdram/tb_ui_clk 1'h0
...c_sdram/tb_ui_rst 1'h0
...c_sdram/tb_ui_sel 4'hf
...sdram/tb_ui_addr ...c10
...sdram/tb_ui_data ...003
...dram/tb_uo_data ...zzzz
...ui_host_ld_mode 1'h0
...sdram/tb_ui_read 1'h0
...sdram/tb_ui_write 1'h0
...sdram/tb_uo_ack 1'h0
...sdc_sdram/status ...OW
.../bfm_monitor/FSM C_IDLE
..._monitor/SDRAM NOP
4'hf
32'h00003c10
32'h00000023
TEST 8: LMR BL8
C_IDLE C_PRE C_PREW C_LMR C_TMRD C_IDLE
NOP PRECH NOP LMR NOP
Now 000 ps
Cursor 1 511 ps
Cursor 3 182 ps
00 ps 3850000 ps 3900000 ps 3950000 ps 4000000 ps 4050000 ps

Result 9: Burst Write

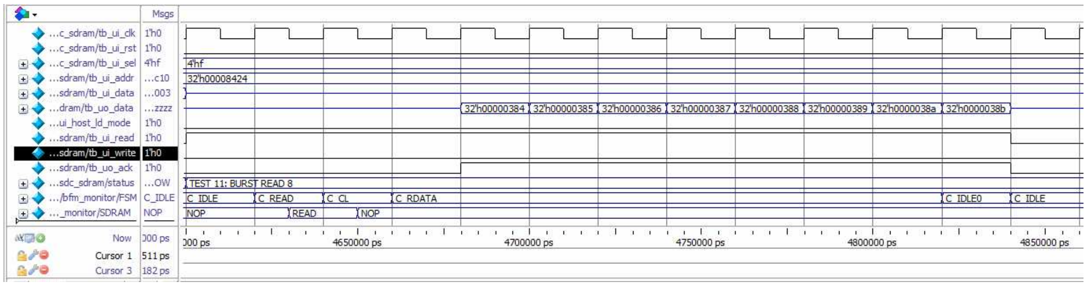

Msgs
...c_sdram/tb_ui_clk 1'h0
...c_sdram/tb_ui_rst 1'h0
...c_sdram/tb_ui_sel 4'hf
...sdram/tb_ui_addr ...c10
...sdram/tb_ui_data ...003
...dram/tb_uo_data ...zzzz
...ui_host_ld_mode 1'h0
...sdram/tb_ui_read 1'h0
...sdram/tb_ui_write 1'h0
...sdram/tb_uo_ack 1'h0
...sdc_sdram/status ...OW
.../bfm_monitor/FSM C_IDLE
..._monitor/SDRAM NOP
4'hf
32'h00008424
32'h00000384 32'h00... 32'h00... 32'h00... 32'h00... 32'h00... 32'h00... 32'h0000038b
TEST 9: BURST WRITE
C_IDLE C_ACT C_TRCD C_WRITE C_WDATA C_IDLE0 C_IDLE
NOP ACT NOP WRITE NOP
Now 000 ps
Cursor 1 511 ps
Cursor 3 182 ps
4100000 ps 4150000 ps 4200000 ps 4250000 ps 4300000 ps

Result 10: Programming Load Mode Register Same (Burst length 8)

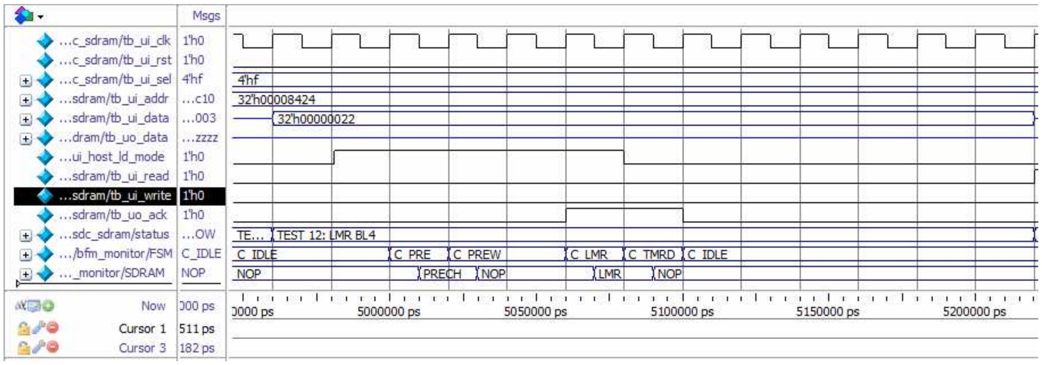

Msgs
...c_sdram/tb_ui_clk 1'h0
...c_sdram/tb_ui_rst 1'h0
...c_sdram/tb_ui_sel 4'hf
...sdram/tb_ui_addr ...c10
...sdram/tb_ui_data ...003
...dram/tb_uo_data ...zzzz
...ui_host_ld_mode 1'h0
...sdram/tb_ui_read 1'h0
...sdram/tb_ui_write 1'h0
...sdram/tb_uo_ack 1'h0
...sdc_sdram/status ...OW
.../bfm_monitor/FSM C_IDLE
..._monitor/SDRAM NOP
4'hf
32'h00008424
32'h00000023
TEST 10: LMR BL8 (same)
C_IDLE
NOP
Now 000 ps
Cursor 1 511 ps
Cursor 3 182 ps
0000 ps 4450000 ps 450000

Result 11: Burst

Read

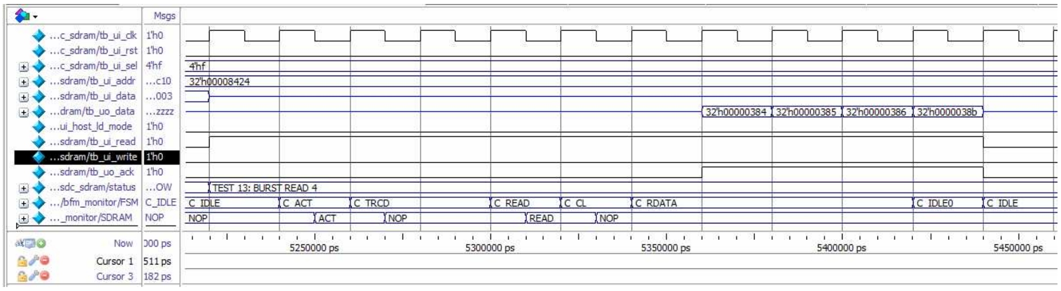


Result 12: Programming Load Mode Register (Burst length 4)

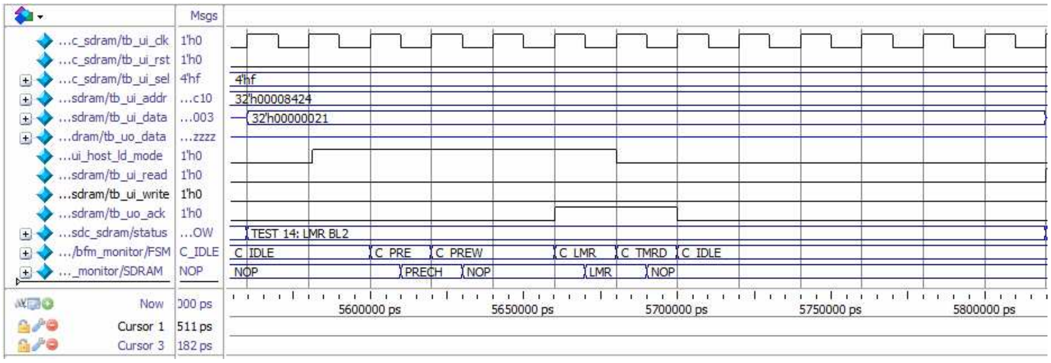


Result 13: Burst

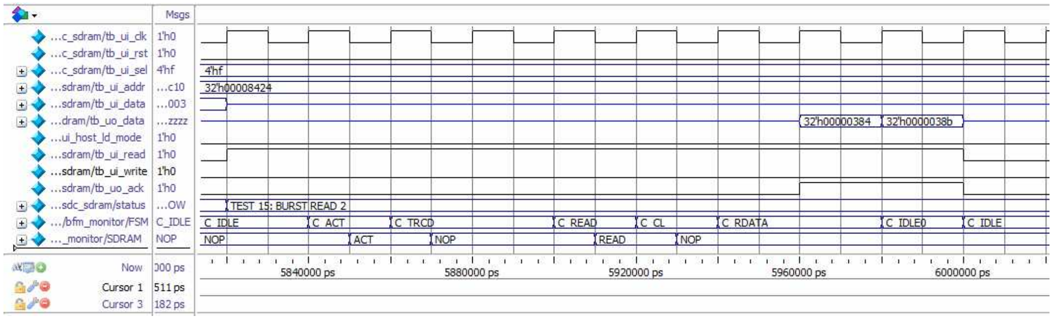
Read



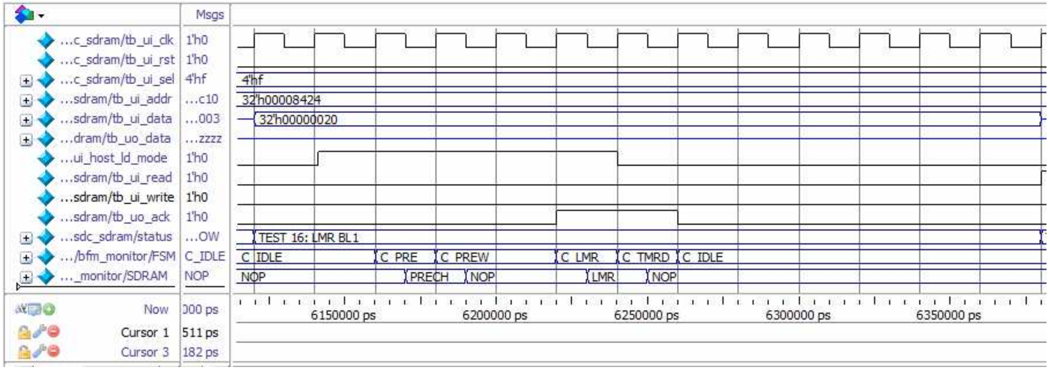
Result 14: Programming Load Mode Register (Burst length 2)



Result 15: Burst Read

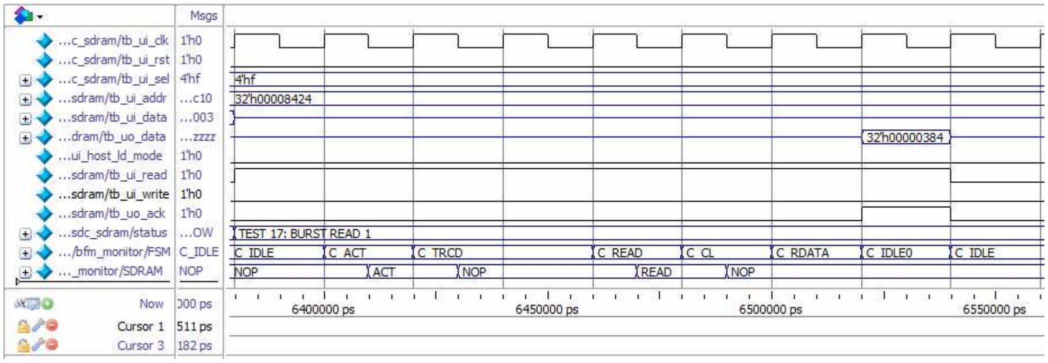


Result 16: Programming Load Mode Register (Burst length 1)

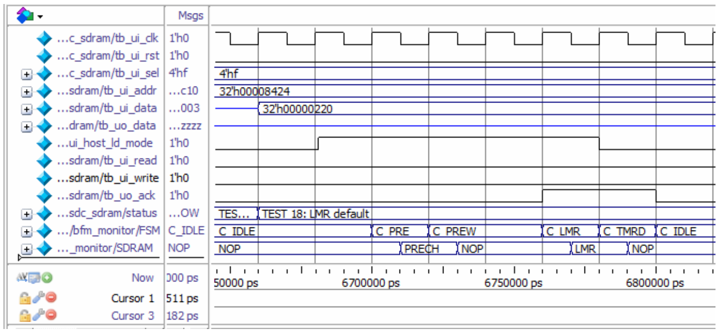


Result 17: Burst

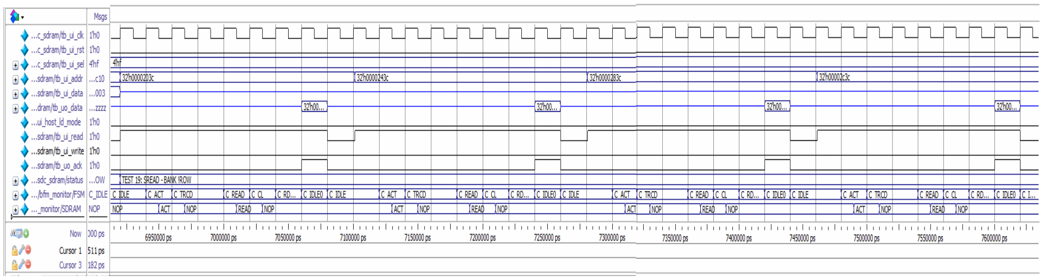
Read



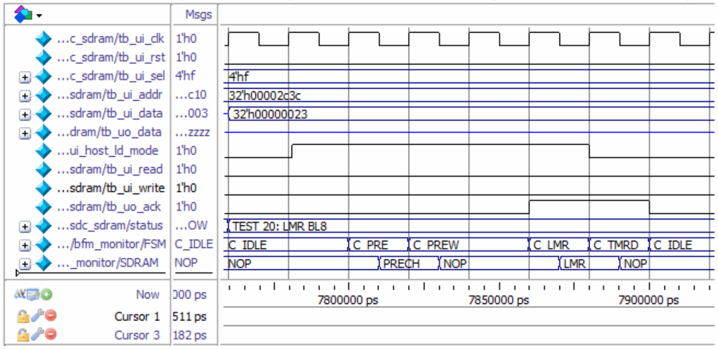
Result 18: Programming Load Mode Register (Default)



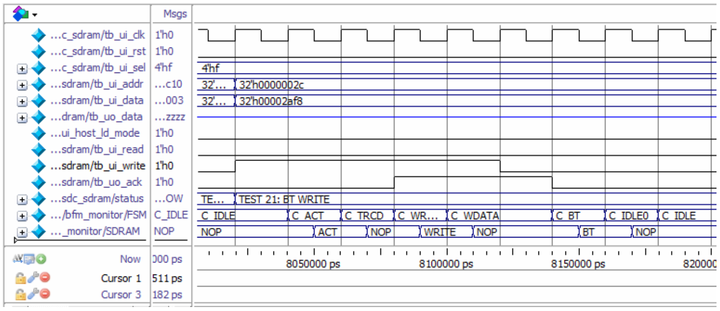
Result 19: Single Read and Active Bank (Different row) read 4 times



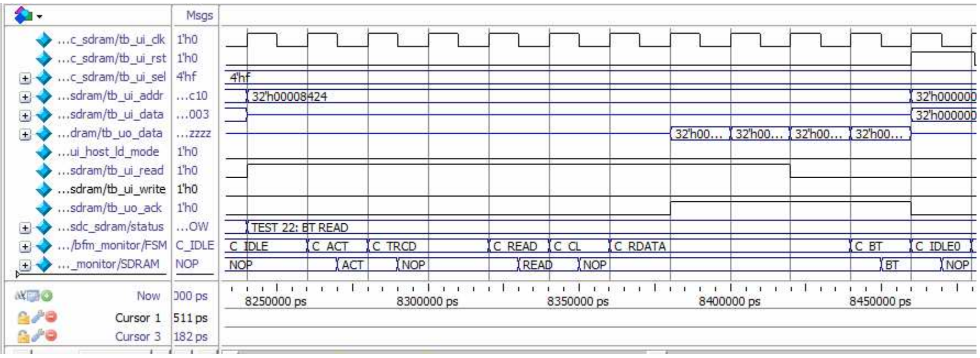
Result 20: Programming Load Mode Register (Burst length 8)



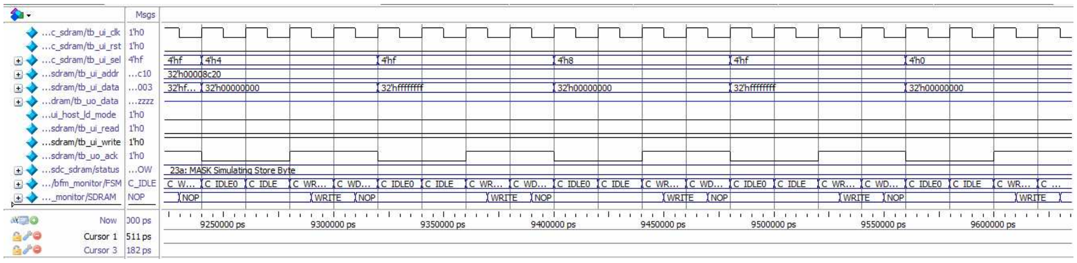
Result 21: Burst Terminal (Write)


Msgs
...c_sdram/tb_ui_clk 1'h0
...c_sdram/tb_ui_rst 1'h0
...c_sdram/tb_ui_sel 4'hf
...sdram/tb_ui_addr ...c10
...sdram/tb_ui_data ...003
...dram/tb_uo_data ...zzzz
...ui_host_ld_mode 1'h0
...sdram/tb_ui_read 1'h0
...sdram/tb_ui_write 1'h0
...sdram/tb_uo_ack 1'h0
...sdc_sdram/status ...OW
.../bfm_monitor/FSM C_IDLE
..._monitor/SDRAM NOP
4'hf
32'... 32'h0000002c
32'... 32'h00002af8
TE... TEST 21: BT WRITE
C_IDLE C_ACT C_TRCD C_WR... C_WDATA C_BT C_IDLE0 C_IDLE
NOP ACT NOP WRITE NOP BT NOP
Now 000 ps
Cursor 1 511 ps
Cursor 3 182 ps
8050000 ps
8100000 ps
8150000 ps
820000

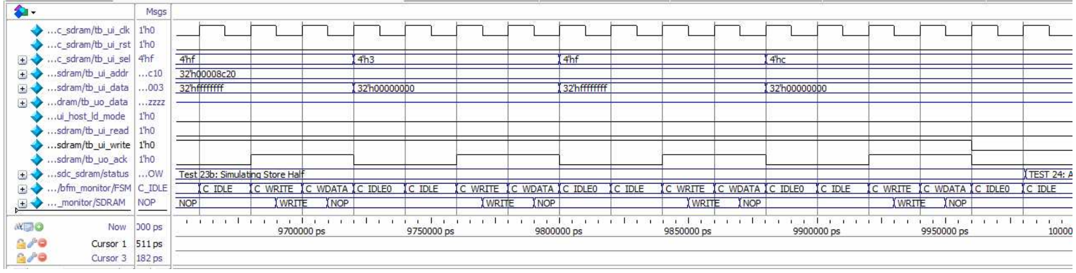
Result 22: Burst Terminal (Read)


Msgs
...c_sdram/tb_ui_clk 1'h0
...c_sdram/tb_ui_rst 1'h0
...c_sdram/tb_ui_sel 4'hf
...sdram/tb_ui_addr ...c10
...sdram/tb_ui_data ...003
...dram/tb_uo_data ...zzzz
...ui_host_ld_mode 1'h0
...sdram/tb_ui_read 1'h0
...sdram/tb_ui_write 1'h0
...sdram/tb_uo_ack 1'h0
...sdc_sdram/status ...OW
.../bfm_monitor/FSM C_IDLE
..._monitor/SDRAM NOP
4'hf
32'h00008424
32'h000000
32'h000000
32'h00... 32'h00... 32'h00... 32'h00...
TEST 22: BT READ
C_IDLE C_ACT C_TRCD C_READ C_CL C_RDATA C_BT C_IDLE0
NOP ACT NOP READ NOP BT NOP
Now 000 ps
Cursor 1 511 ps
Cursor 3 182 ps
8250000 ps
8300000 ps
8350000 ps
8400000 ps
8450000 ps

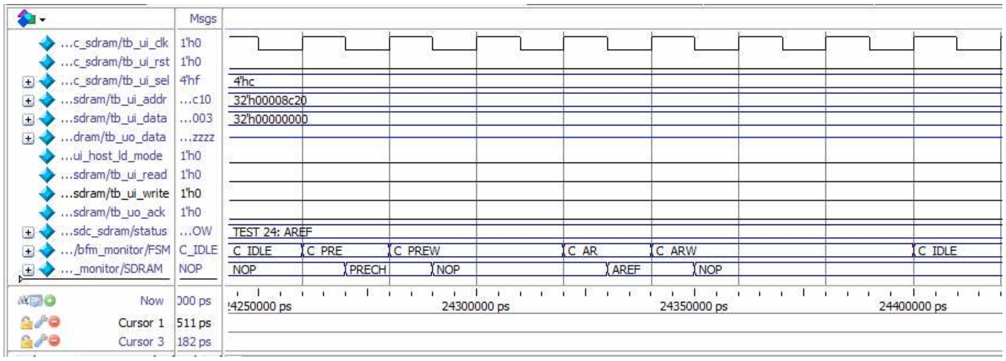
Result 23a: Mask Simulating Store Byte


Msgs
...c_sdram/tb_ui_clk 1'h0
...c_sdram/tb_ui_rst 1'h0
...c_sdram/tb_ui_sel 4'hf
...sdram/tb_ui_addr ...c10
...sdram/tb_ui_data ...003
...dram/tb_uo_data ...zzzz
...ui_host_ld_mode 1'h0
...sdram/tb_ui_read 1'h0
...sdram/tb_ui_write 1'h0
...sdram/tb_uo_ack 1'h0
...sdc_sdram/status ...OW
.../bfm_monitor/FSM C_IDLE
..._monitor/SDRAM NOP
4'hf 4'h4 4'hf 4'h8 4'hf 4'h0
32'h00008c20
32'h00000000 32'hffffffff
23a: MASK Simulating Store Byte
C_IDLE0 C_IDLE C_WR... C_WD...
NOP WRITE
Now Cursor 1 511 ps Cursor 3 182 ps
9250000 ps 9300000 ps 9350000 ps 9400000 ps 9450000 ps 9500000 ps 9550000 ps 9600000 ps

Result 23b: Mask Simulating Store Half Byte


Msgs
...c_sdram/tb_ui_clk 1'h0
...c_sdram/tb_ui_rst 1'h0
...c_sdram/tb_ui_sel 4'hf
...sdram/tb_ui_addr ...c10
...sdram/tb_ui_data ...003
...dram/tb_uo_data ...zzzz
...ui_host_ld_mode 1'h0
...sdram/tb_ui_read 1'h0
...sdram/tb_ui_write 1'h0
...sdram/tb_uo_ack 1'h0
...sdc_sdram/status ...OW
.../bfm_monitor/FSM C_IDLE
..._monitor/SDRAM NOP
4'hf 4'h3 4'hf 4'hc
32'h00008c20
32'hffffffff 32'h00000000
Test 23b: Simulating Store Half
TEST 24: A
C_IDLE C_WRITE C_WDATA C_IDLE0
NOP WRITE
Now Cursor 1 511 ps Cursor 3 182 ps
9700000 ps 9750000 ps 9800000 ps 9850000 ps 9900000 ps 9950000 ps 10000

Result 24: Wait for Auto Refresh


Msgs
...c_sdram/tb_ui_clk 1'h0
...c_sdram/tb_ui_rst 1'h0
...c_sdram/tb_ui_sel 4'hf
...sdram/tb_ui_addr ...c10
...sdram/tb_ui_data ...003
...dram/tb_uo_data ...zzzz
...ui_host_ld_mode 1'h0
...sdram/tb_ui_read 1'h0
...sdram/tb_ui_write 1'h0
...sdram/tb_uo_ack 1'h0
...sdc_sdram/status ...OW
.../bfm_monitor/FSM C_IDLE
..._monitor/SDRAM NOP
4'hc
32'h00008c20
32'h00000000
TEST 24: AREF
C_IDLE C_PRE C_PREW C_AR C_ARW C_IDLE
NOP PRECH NOP AREF NOP
Now 000 ps Cursor 1 511 ps Cursor 3 182 ps
24250000 ps 24300000 ps 24350000 ps 24400000 ps

7.2: Memory System

The following test is to make sure that SDRAM Controller and Memory Arbiter that implemented can be support four caches, two i-caches and two d-cache. And this memory arbiter will allow caches to access SDRAM accordingly to the priority given. Since this test does not involve any TLB, plus the cache only have a fixed 8 burst length mode, an appropriate test for this different load mode configuration are not able to carry out.

Thus tb_r_BL_sel is assigned to change the cache output into different burst length (acts like TLB) for testing. If the SDRAM is able to receive the load mode configuration from cache and the read address, SDRAM should be sending back the data according to the address from cache.

7.2.1: Test Plan

Function To be Tested	Test Case

Different load mode configuration with burst length 1, 2, 4 and 8.	tb_r_BL_sel[3] = 3'd3;//burst length = 8
	tb_r_BL_sel[2] = 3'd2; ;//burst length = 4
	tb_r_BL_sel[1] = 3'd1; ;//burst length = 2
	tb_r_BL_sel[0] = 3'd1; ;//burst length = 2
	tb_r_cpu_cac_addr3 = 32'h00567000 ;
	tb_r_cpu_cac_addr2 = 32'h00567000 ;
	tb_r_cpu_cac_addr1 = 32'h00567000 ;
	tb_r_cpu_cac_addr0 = 32'h00567000

7.2.2: Testbench Verilog code

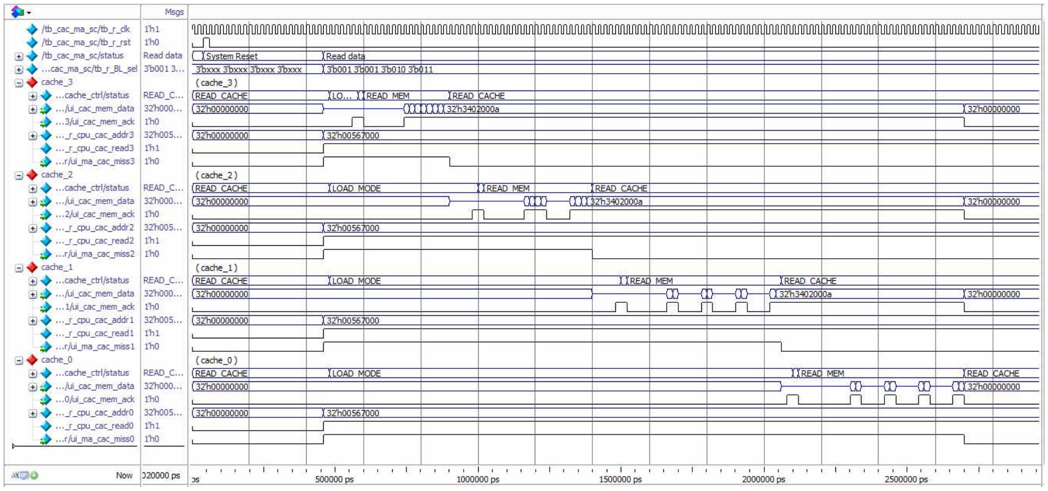
```
`include "./../util/sdc_macro.v"
`timescale 1ns / 10ps
module tb_cac_ma_sc();
//CPU to 4 caches
//cache3
wire    [31:0]  tb_w_cpu_cac_data3;
reg     [31:0]  tb_r_cpu_cac_addr3,
                tb_r_cpu_cac_data3;
reg             tb_r_cpu_cac_read3,
                tb_r_cpu_cac_write3;
//cache2
wire    [31:0]  tb_w_cpu_cac_data2;
reg     [31:0]  tb_r_cpu_cac_addr2,
                tb_r_cpu_cac_data2;
reg             tb_r_cpu_cac_read2,
                tb_r_cpu_cac_write2;
//cache1
wire    [31:0]  tb_w_cpu_cac_data1;
reg     [31:0]  tb_r_cpu_cac_addr1,
                tb_r_cpu_cac_data1;
reg             tb_r_cpu_cac_read1,
                tb_r_cpu_cac_write1;
//cache0
wire    [31:0]  tb_w_cpu_cac_data0;
reg     [31:0]  tb_r_cpu_cac_addr0,
                tb_r_cpu_cac_data0;
reg             tb_r_cpu_cac_read0,
                tb_r_cpu_cac_write0;
reg             tb_r_clk;
reg             tb_r_rst;

//between caches and memory arbiter
//4 caches
//cache3
wire            w_ma_cac_read3,
                w_ma_cac_write3,
                w_ma_cac_host_ld_mode3,
                w_ma_cac_miss3;
wire    [3:0]   w_ma_cac_sel3;
wire    [31:0]  w_ma_cac_addr3,
                w_ma_cac_o_data3;
reg     [31:0]  r_ma_cac_i_data3;
wire            w_ma_cac_ack3;
//cache2
wire            w_ma_cac_read2,
                w_ma_cac_write2,
                w_ma_cac_host_ld_mode2,
                w_ma_cac_miss2;
wire    [3:0]   w_ma_cac_sel2;
wire    [31:0]  w_ma_cac_addr2,
                w_ma_cac_o_data2;
reg     [31:0]  r_ma_cac_i_data2;
wire            w_ma_cac_ack2;
//cache1
wire            w_ma_cac_read1,
                w_ma_cac_write1,
                w_ma_cac_host_ld_mode1,
                w_ma_cac_miss1;
wire    [3:0]   w_ma_cac_sel1;
wire    [31:0]  w_ma_cac_addr1,
                w_ma_cac_o_data1;
reg     [31:0]  r_ma_cac_i_data1;
wire            w_ma_cac_ack1;
//cache0
wire            w_ma_cac_read0,
                w_ma_cac_write0,
                w_ma_cac_host_ld_mode0,
                w_ma_cac_miss0;
wire    [3:0]   w_ma_cac_sel0;
wire    [31:0]  w_ma_cac_addr0,
                w_ma_cac_o_data0;
reg     [31:0]  r_ma_cac_i_data0;
wire            w_ma_cac_ack0;

//between memory arbiter and sdram controller
wire            w_ma_sdc_host_ld_mode,
                w_ma_sdc_read,
                w_ma_sdc_write;
wire    [3:0]   w_ma_sdc_sel;
wire    [31:0]  w_ma_sdc_addr,
                w_ma_sdc_i_data,
                w_ma_sdc_o_data;
wire            w_ma_sdc_ack;

//between sdram controller and sdram
wire    [31:0]  w_sc_sdc_dq;
wire    [11:0]  w_sc_sdc_addr;
wire    [1:0]   w_sc_sdc_ba;
wire            w_sc_sdc_cs_n;
wire            w_sc_sdc_ras_n;
wire            w_sc_sdc_cas_n;
wire            w_sc_sdc_we_n;
wire    [3:0]   w_sc_sdc_dqm;

//wishbone standard signal from caches output
wire    [3:0]   w_cycle,
                w_strobe;

//Change burst length of caches to test different mode configuration
reg [2:0]       tb_r_BL_sel[0:3];
wire    [31:0]  w_i_data3,
                w_i_data2,
                w_i_data1,
                w_i_data0;

//indicates current test status in waveform
reg [255:0]     status;

u_cache cache_3
 (//memory arbiter connection
  .uo_cac_mem_addr(w_ma_cac_addr3),
  .uo_cac_mem_data(w_i_data3),
  .uo_cac_miss(w_ma_cac_miss3),
  .uo_cac_mem_cycle(w_cycle[3]),
  .uo_cac_mem_strobe(w_strobe[3]),
  .uo_cac_mem_rw(w_ma_cac_we3),
  .uo_cac_mem_host_ld_mode(w_ma_cac_host_ld_mode3),
  .uo_cac_mem_sel(w_ma_cac_sel3),
  .ui_cac_mem_data(w_ma_cac_o_data3),
  .ui_cac_mem_ack(w_ma_cac_ack3),
  // CPU connection
  .uo_cac_cpu_data(tb_w_cpu_cac_data3),
  .ui_cac_cpu_addr(tb_r_cpu_cac_addr3),
  .ui_cac_cpu_data(tb_r_cpu_cac_data3),
  .ui_cac_cpu_read(tb_r_cpu_cac_read3),
  .ui_cac_cpu_write(tb_r_cpu_cac_write3),
  .ui_cac_rst(tb_r_rst),
  .ui_cac_clk(tb_r_clk) ) ;

 u_cache cache_2
 (//memory arbiter connection
  .uo_cac_mem_addr(w_ma_cac_addr2),
  .uo_cac_mem_data(w_i_data2),
  .uo_cac_miss(w_ma_cac_miss2),
  .uo_cac_mem_cycle(w_cycle[2]),
  .uo_cac_mem_strobe(w_strobe[2]),
  .uo_cac_mem_rw(w_ma_cac_we2),
  .uo_cac_mem_host_ld_mode(w_ma_cac_host_ld_mode2),
  .uo_cac_mem_sel(w_ma_cac_sel2),
  .ui_cac_mem_data(w_ma_cac_o_data2),
  .ui_cac_mem_ack(w_ma_cac_ack2),
  // CPU connection
  .uo_cac_cpu_data(tb_w_cpu_cac_data2),
  .ui_cac_cpu_addr(tb_r_cpu_cac_addr2),
  .ui_cac_cpu_data(tb_r_cpu_cac_data2),
  .ui_cac_cpu_read(tb_r_cpu_cac_read2),
  .ui_cac_cpu_write(tb_r_cpu_cac_write2),
  .ui_cac_rst(tb_r_rst),
  .ui_cac_clk(tb_r_clk) ) ;

u_cache cache_1
 (//memory arbiter connection
  .uo_cac_mem_addr(w_ma_cac_addr1),
  .uo_cac_mem_data(w_i_data1),
  .uo_cac_miss(w_ma_cac_miss1),
  .uo_cac_mem_cycle(w_cycle[1]),
  .uo_cac_mem_strobe(w_strobe[1]),
  .uo_cac_mem_rw(w_ma_cac_we1),
  .uo_cac_mem_host_ld_mode(w_ma_cac_host_ld_mode1),
  .uo_cac_mem_sel(w_ma_cac_sel1),
  .ui_cac_mem_data(w_ma_cac_o_data1),
  .ui_cac_mem_ack(w_ma_cac_ack1),
  // CPU connection
  .uo_cac_cpu_data(tb_w_cpu_cac_data1),
  .ui_cac_cpu_addr(tb_r_cpu_cac_addr1),
  .ui_cac_cpu_data(tb_r_cpu_cac_data1),
  .ui_cac_cpu_read(tb_r_cpu_cac_read1),
  .ui_cac_cpu_write(tb_r_cpu_cac_write1),
  .ui_cac_rst(tb_r_rst),
  .ui_cac_clk(tb_r_clk) ) ;

 u_cache cache_0
 (//memory arbiter connection
  .uo_cac_mem_addr(w_ma_cac_addr0),
  .uo_cac_mem_data(w_i_data0),
  .uo_cac_miss(w_ma_cac_miss0),
  .uo_cac_mem_cycle(w_cycle[0]),
  .uo_cac_mem_strobe(w_strobe[0]),
  .uo_cac_mem_rw(w_ma_cac_we0),
  .uo_cac_mem_host_ld_mode(w_ma_cac_host_ld_mode0),
  .uo_cac_mem_sel(w_ma_cac_sel0),
  .ui_cac_mem_data(w_ma_cac_o_data0),
  .ui_cac_mem_ack(w_ma_cac_ack0),
  // CPU connection
  .uo_cac_cpu_data(tb_w_cpu_cac_data0),
  .ui_cac_cpu_addr(tb_r_cpu_cac_addr0),
  .ui_cac_cpu_data(tb_r_cpu_cac_data0),
  .ui_cac_cpu_read(tb_r_cpu_cac_read0),
  .ui_cac_cpu_write(tb_r_cpu_cac_write0),
  .ui_cac_rst(tb_r_rst),
  .ui_cac_clk(tb_r_clk) ) ;

 u_mem_arbiter mem_arbiter
 (//caches connection
  //cache3
  .ui_ma_cac_read3(w_ma_cac_read3),
  .ui_ma_cac_write3(w_ma_cac_write3),
  .ui_ma_cac_host_ld_mode3(w_ma_cac_host_ld_mode3),
  .ui_ma_cac_sel3(w_ma_cac_sel3),
  .ui_ma_cac_addr3(w_ma_cac_addr3),
  .ui_ma_cac_data3(r_ma_cac_i_data3),
  .ui_ma_cac_miss3(w_ma_cac_miss3),
  .uo_ma_cac_ack3(w_ma_cac_ack3),
  .uo_ma_cac_data3(w_ma_cac_o_data3),
  //cache2
  .ui_ma_cac_read2(w_ma_cac_read2),
  .ui_ma_cac_write2(w_ma_cac_write2),
  .ui_ma_cac_host_ld_mode2(w_ma_cac_host_ld_mode2),
  .ui_ma_cac_sel2(w_ma_cac_sel2),
  .ui_ma_cac_addr2(w_ma_cac_addr2),
  .ui_ma_cac_data2(r_ma_cac_i_data2),
  .ui_ma_cac_miss2(w_ma_cac_miss2),
  .uo_ma_cac_ack2(w_ma_cac_ack2),
  .uo_ma_cac_data2(w_ma_cac_o_data2),
  //cache1
  .ui_ma_cac_read1(w_ma_cac_read1),
  .ui_ma_cac_write1(w_ma_cac_write1),
  .ui_ma_cac_host_ld_mode1(w_ma_cac_host_ld_mode1),
  .ui_ma_cac_sel1(w_ma_cac_sel1),
  .ui_ma_cac_addr1(w_ma_cac_addr1),
  .ui_ma_cac_data1(r_ma_cac_i_data1),
  .ui_ma_cac_miss1(w_ma_cac_miss1),
  .uo_ma_cac_ack1(w_ma_cac_ack1),
  .uo_ma_cac_data1(w_ma_cac_o_data1),
  //cache0
  .ui_ma_cac_read0(w_ma_cac_read0),
  .ui_ma_cac_write0(w_ma_cac_write0),
  .ui_ma_cac_host_ld_mode0(w_ma_cac_host_ld_mode0),
  .ui_ma_cac_sel0(w_ma_cac_sel0),
  .ui_ma_cac_addr0(w_ma_cac_addr0),
  .ui_ma_cac_data0(r_ma_cac_i_data0),
  .ui_ma_cac_miss0(w_ma_cac_miss0),
  .uo_ma_cac_ack0(w_ma_cac_ack0),
  .uo_ma_cac_data0(w_ma_cac_o_data0),

  //sdram controller connection
  .ui_ma_sdc_ack(w_ma_sdc_ack),
  .ui_ma_sdc_data(w_ma_sdc_i_data),
  .uo_ma_sdc_read(w_ma_sdc_read),
  .uo_ma_sdc_write(w_ma_sdc_write),
  .uo_ma_sdc_host_ld_mode(w_ma_sdc_host_ld_mode),
  .uo_ma_sdc_sel(w_ma_sdc_sel),
  .uo_ma_sdc_addr(w_ma_sdc_addr),
  .uo_ma_sdc_data(w_ma_sdc_o_data));

 u_sdram_controller sdram_controller
  (.ui_sdc_clk(tb_r_clk),
   .ui_sdc_rst(tb_r_rst),
   //memory arbiter connection
   .ui_host_ld_mode(w_ma_sdc_host_ld_mode),
   .ui_sdc_read(w_ma_sdc_read),
   .ui_sdc_write(w_ma_sdc_write),
   .ui_sdc_sel(w_ma_sdc_sel),
   .ui_sdc_addr(w_ma_sdc_addr),
   .ui_sdc_dat(w_ma_sdc_o_data),
   .uo_sdc_dat(w_ma_sdc_i_data),
   .uo_sdc_ack(w_ma_sdc_ack),
   //sdram connection
   .uio_sdc_dq(w_sc_sdc_dq),
   .uo_sdc_ba(w_sc_sdc_ba),
   .uo_sdc_dqm(w_sc_sdc_dqm),
   .uo_sdc_addr(w_sc_sdc_addr),
   .uo_sdc_cs_n(w_sc_sdc_cs_n),
   .uo_sdc_ras_n(w_sc_sdc_ras_n),
   .uo_sdc_cas_n(w_sc_sdc_cas_n),
   .uo_sdc_we_n(w_sc_sdc_we_n) ) ;

  //MICRON SDRAM Instantiation
  mt48lc4m32b2 sdram(
  .Dq(w_sc_sdc_dq),
  .Addr(w_sc_sdc_addr),
  .Ba(w_sc_sdc_ba),
  .Clk(tb_r_clk),
  .Cke(1'b1), //cke always activated
  .Cs_n(w_sc_sdc_cs_n),
  .Ras_n(w_sc_sdc_ras_n),
  .Cas_n(w_sc_sdc_cas_n),
  .We_n(w_sc_sdc_we_n),
  .Dqm(w_sc_sdc_dqm));

//generate READ enable signal from caches to memory arbiter
assign w_ma_cac_read3 = w_cycle[3]&w_strobe[3];
assign w_ma_cac_read2 = w_cycle[2]&w_strobe[2];
assign w_ma_cac_read1 = w_cycle[1]&w_strobe[1];
assign w_ma_cac_read0 = w_cycle[0]&w_strobe[0];

//self LMR programable test
always@(*)begin

if(w_ma_cac_host_ld_mode3)
            r_ma_cac_i_data3 = {w_i_data3[31:3],tb_r_BL_sel[3]};
```

```
else
        r_ma_cac_i_data3 = w_i_data3;

if(w_ma_cac_host_ld_mode2)
        r_ma_cac_i_data2 = {w_i_data2[31:3],tb_r_BL_sel[2]};
else
        r_ma_cac_i_data2 = w_i_data2;

if(w_ma_cac_host_ld_mode1)
        r_ma_cac_i_data1 = {w_i_data1[31:3],tb_r_BL_sel[1]};
else
        r_ma_cac_i_data1 = w_i_data1;

if(w_ma_cac_host_ld_mode0)
        r_ma_cac_i_data0 = {w_i_data0[31:3],tb_r_BL_sel[0]};
else
        r_ma_cac_i_data0 = w_i_data0;

end

 //initialize clock signal
 initial tb_r_clk = 1;
 always #10 tb_r_clk = ~tb_r_clk;

initial begin
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//Signals initialization
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  status = "Signals initialization";
  tb_r_cpu_cac_addr3     = 32'b0;
  tb_r_cpu_cac_data3    = 32'b0;
  tb_r_cpu_cac_write3    = 1'b0;
  tb_r_cpu_cac_read3      = 1'b0;

  tb_r_cpu_cac_addr2     = 32'b0;
  tb_r_cpu_cac_data2    = 32'b0;
  tb_r_cpu_cac_write2    = 1'b0;
  tb_r_cpu_cac_read2      = 1'b0;

  tb_r_cpu_cac_addr1     = 32'b0;
  tb_r_cpu_cac_data1    = 32'b0;
  tb_r_cpu_cac_write1    = 1'b0;
  tb_r_cpu_cac_read1      = 1'b0;

  tb_r_cpu_cac_addr0     = 32'b0;
  tb_r_cpu_cac_data0    = 32'b0;
  tb_r_cpu_cac_write0    = 1'b0;
  tb_r_cpu_cac_read0      = 1'b0;
  tb_r_rst                        = 0;
  repeat(2) @(posedge tb_r_clk);

 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 //System Reset
 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  status = "System Reset";
  tb_r_rst  = 1;
  repeat(1) @(posedge tb_r_clk);

  tb_r_rst  = 0;
```

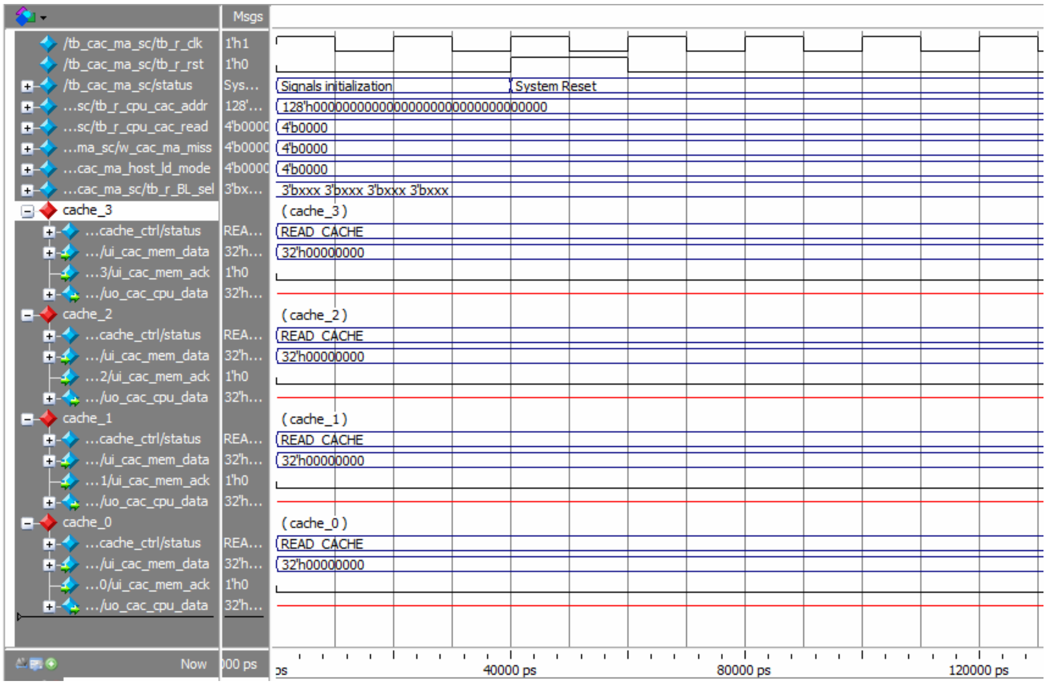
```
   repeat(20) @(posedge tb_r_clk);

 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 // Prepare data in sdram
    $readmemh("micron SDRAM/sdram_bank0_data.txt", sdram.Bank0) ;

status = "Read data";
// MEM stage
            //select brust length 0,1,2,3 = 1,2,4,8
            tb_r_BL_sel[3] = 3'd3;
            tb_r_BL_sel[2] = 3'd2;
            tb_r_BL_sel[1] = 3'd1;
            tb_r_BL_sel[0] = 3'd1;

            //NOTED: burst length 1 test failed
    // Read a data from 0x1000_000
    tb_r_cpu_cac_data3 = 0;
    tb_r_cpu_cac_data2 = 0;
    tb_r_cpu_cac_data1 = 0;
    tb_r_cpu_cac_data0 = 0;

    tb_r_cpu_cac_addr3 = 32'h00567000 ;
    tb_r_cpu_cac_addr2 = 32'h00567000 ;
    tb_r_cpu_cac_addr1 = 32'h00567000 ;
    tb_r_cpu_cac_addr0 = 32'h00567000 ;

    tb_r_cpu_cac_read3 = 1 ;
    tb_r_cpu_cac_write3 = 0;
    tb_r_cpu_cac_read2 = 1 ;
    tb_r_cpu_cac_write2 = 0;
    tb_r_cpu_cac_read1 = 1 ;
    tb_r_cpu_cac_write1 = 0;
    tb_r_cpu_cac_read0 = 1 ;
    tb_r_cpu_cac_write0 = 0;

    @(posedge tb_r_clk) ;
            // Expecting dtlb and dcache misses
// Wait until they are done
    while(w_ma_cac_miss3||w_ma_cac_miss2||w_ma_cac_miss1||w_ma_cac_miss0) @(posedge
tb_r_clk) ;


  repeat(15) @(posedge tb_r_clk);
   $stop;

   end

endmodule
```

7.2.3: Simulation Result (Timing Diagram)

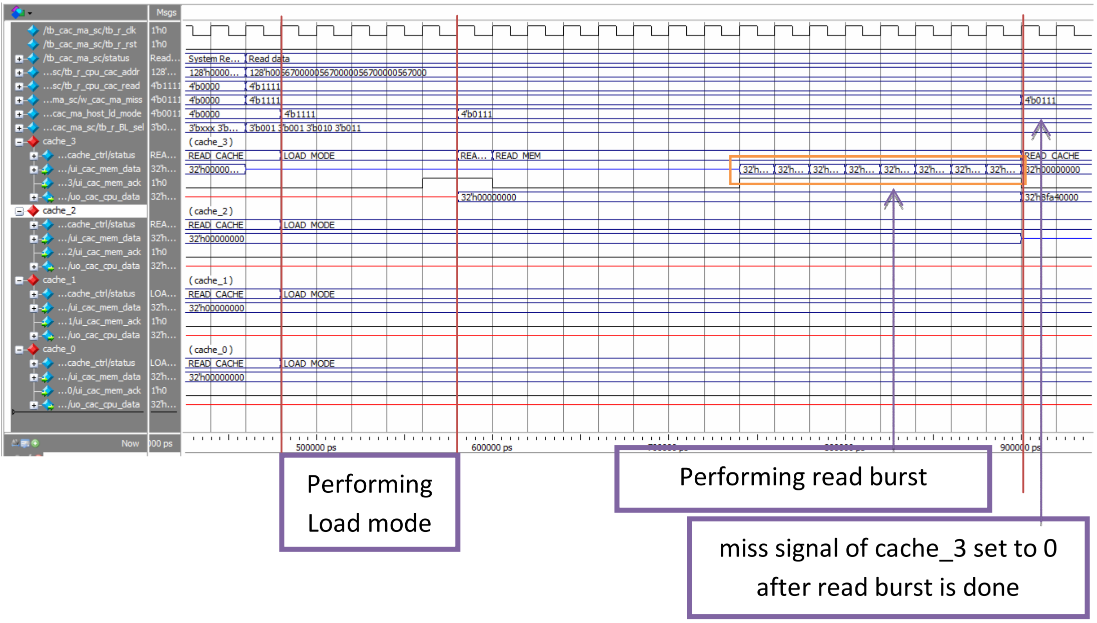
Overall Test Timing Diagram


Msgs
/tb_cac_ma_sc/tb_r_clk 1'h1
/tb_cac_ma_sc/tb_r_rst 1'h0
/tb_cac_ma_sc/status Read data
...cac_ma_sc/tb_r_BL_sel 3'b001 3...
cache_3
...cache_ctrl/status READ_C...
.../ui_cac_mem_data 32'h000...
...3/ui_cac_mem_ack 1'h0
..._r_cpu_cac_addr3 32'h005...
..._r_cpu_cac_read3 1'h1
...r/ui_ma_cac_miss3 1'h0
cache_2
...cache_ctrl/status READ_C...
.../ui_cac_mem_data 32'h000...
...2/ui_cac_mem_ack 1'h0
..._r_cpu_cac_addr2 32'h005...
..._r_cpu_cac_read2 1'h1
...r/ui_ma_cac_miss2 1'h0
cache_1
...cache_ctrl/status READ_C...
.../ui_cac_mem_data 32'h000...
...1/ui_cac_mem_ack 1'h0
..._r_cpu_cac_addr1 32'h005...
..._r_cpu_cac_read1 1'h1
...r/ui_ma_cac_miss1 1'h0
cache_0
...cache_ctrl/status READ_C...
.../ui_cac_mem_data 32'h000...
...0/ui_cac_mem_ack 1'h0
..._r_cpu_cac_addr0 32'h005...
..._r_cpu_cac_read0 1'h1
...r/ui_ma_cac_miss0 1'h0
System Reset
Read data
3'bxxx 3'bxxx 3'bxxx 3'bxxx
3'b001 3'b001 3'b010 3'b011
(cache_3)
READ_CACHE
LO...
READ_MEM
READ_CACHE
32'h00000000
32'h3402000a
32'h00567000
(cache_2)
LOAD_MODE
(cache_1)
(cache_0)
Now
020000 ps
ps
500000 ps
1000000 ps
1500000 ps
2000000 ps
2500000 ps

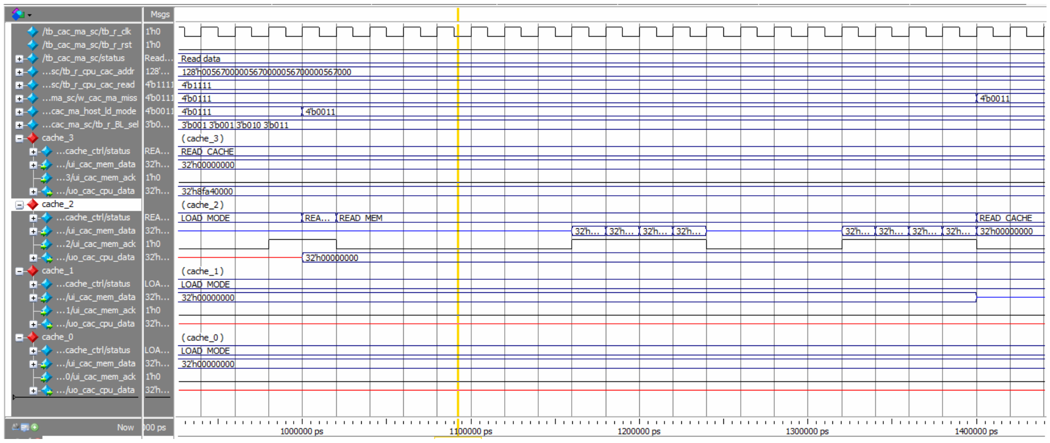
Signal Initialization and System Reset


Msgs
/tb_cac_ma_sc/tb_r_clk 1'h1
/tb_cac_ma_sc/tb_r_rst 1'h0
/tb_cac_ma_sc/status Sys...
...sc/tb_r_cpu_cac_addr 128'...
...sc/tb_r_cpu_cac_read 4'b0000
...ma_sc/w_cac_ma_miss 4'b0000
...cac_ma_host_ld_mode 4'b0000
...cac_ma_sc/tb_r_BL_sel 3'bx...
cache_3
...cache_ctrl/status REA...
.../ui_cac_mem_data 32'h...
...3/ui_cac_mem_ack 1'h0
.../uo_cac_cpu_data 32'h...
cache_2
...cache_ctrl/status REA...
.../ui_cac_mem_data 32'h...
...2/ui_cac_mem_ack 1'h0
.../uo_cac_cpu_data 32'h...
cache_1
...cache_ctrl/status REA...
.../ui_cac_mem_data 32'h...
...1/ui_cac_mem_ack 1'h0
.../uo_cac_cpu_data 32'h...
cache_0
...cache_ctrl/status REA...
.../ui_cac_mem_data 32'h...
...0/ui_cac_mem_ack 1'h0
.../uo_cac_cpu_data 32'h...
Signals initialization
System Reset
128'h00000000000000000000000000000000
4'b0000
4'b0000
4'b0000
3'bxxx 3'bxxx 3'bxxx 3'bxxx
(cache_3)
READ_CACHE
32'h00000000
(cache_2)
READ_CACHE
32'h00000000
(cache_1)
READ_CACHE
32'h00000000
(cache_0)
READ_CACHE
32'h00000000
Now
00 ps
ps
40000 ps
80000 ps
120000 ps

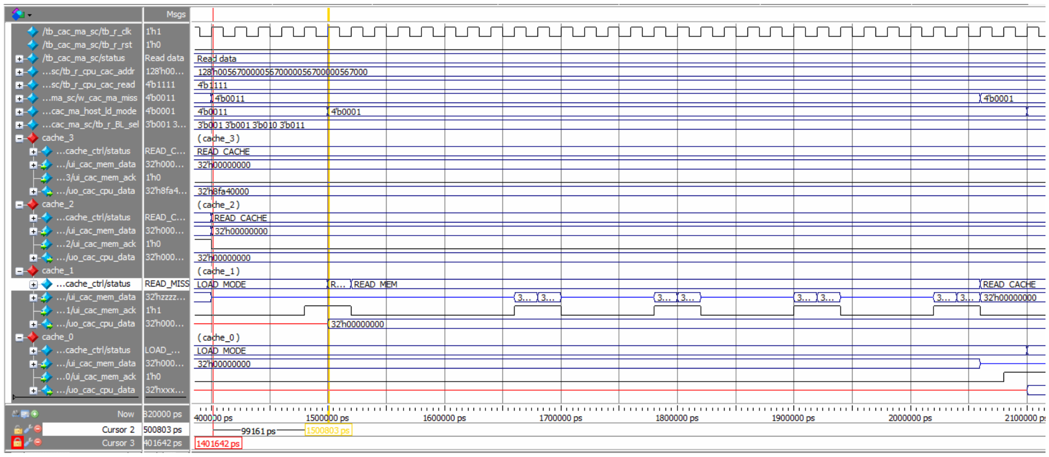
Priority given to cache_3 to run first according to the pin assigned in Memory Arbiter. tb_r_BL_sel assigned to burst length =8, indicates burst length of SDRAM is set to eight.



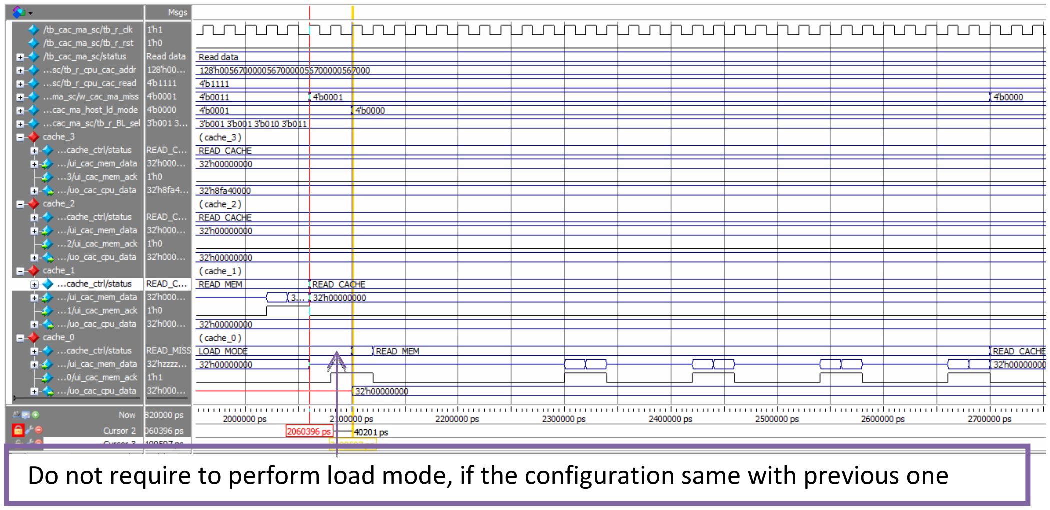
Next, the priority is given to cache_2 and tb_r_BL_sel assigned to burst length =4, indicates that burst length of SDRAM is set to four.



Next, the priority is given to cache_1 and tb_r_BL_sel assigned to burst length =2, indicates that burst length of SDRAM is set to two.



tb_r_BL_sel assigned to burst length =2, indicates that burst length of SDRAM is set to two same with the previous programmable mode.



Do not require to perform load mode, if the configuration same with previous one

Chapter 8: Discussions and Conclusion

8.1: Discussions

SDRAM controller can be directly connected to the processor but accessing SDRAM once can take up 40 to 50 clock cycles. Read or write from cache or TLB is only required 2 to 3 clock cycles. Thus cache or TLB is implemented to increase the performance of memory system. The memory arbiter is then come by to support multiple caches accessing to the DRAM.

The SDRAM controller is successfully redesigned from the previous work [10]. The memory controller is no longer in wishbone standards. Since the strobe and cycle signal are removed, write and read cannot using a share pin. A read signal is added to enable read operation. In addition, the protocol controller block is modified to a simplified form of FSM. In other parts of sub-modules, the power up control has been removed since it is not strictly necessary to functioning in the system. And some of the sub-modules are combined to eliminate unnecessary circuitry that may cause performance redundancy.

The memory arbiter is implemented and worked nicely. SDRAM is now allowed multiple cached interfacing with the presence of memory arbiter. In the Chapter 7 has shown the memory arbiter has been tested and it is working fine.

On the other hand, the SDRAM controller now has better support in different load mode control. Normally, it takes up to 7 clock cycles or more to perform load mode cycles. But it spends 2 to 3 clock cycles only when same configuration is detected as previous one. And no configuration will be loaded to the SDRAM in this time. Hence, the overall performance is improved due to the reducing time of load mode cycle.

At the end, a series of test cases has been carried to justify the SDRAM controller design is either compatible with the memory system or not. And no flaws are found from the result. All the expected results are obtained.

8.2: Conclusion

The SDRAM controller is successfully redesigned from the previous work [10]. Next, more detailed tests also have been provided and been verified that the SDRAM

Controller is compatible with the MICRON MT48LC4M32B2 SDRAM. After that, the SDRAM controller design was further developed to allow more caches to access to SDRAM by using memory shared bus arbiter and with an improved version of load mode configurations control. Now, a more thorough analysis for the test integration of memory system is provided which can be determined from Chapter 7. The implemented tests are able to obtain with the desired results.

8.3: Future Work

A more thorough analysis needs to be done on the cache interfacing, exception handling (to handle delay caused by miss) and the address distribution. Besides, it is crucial for the future designer to keep byte addressability and half-word addressability in mind when building future memory module for the MIPS unit. Apart from that, a study needs to be conducted to see how the SDRAM controller and the MIPS Processor are connected. Last but not least, this SDR SDRAM controller design can also be modified for DDR SDRAM controller due to its similarities. Other than the data transfer phase, the different power-on initialization and mode register definitions; these two SDRAMs share same command sets and basic design concepts. The future designer can obtain the idea to implement DDR memory controller from this design and thus reduce the overall time of implementation.

References

[1] John L. Hennessy , David A. Patterson, "Computer architecture (2nd ed.): a quantitative approach", Morgan Kaufmann Publishers Inc., San Francisco, CA, 1996

[2] Mittra, S. (1995) IEEE Xplore/IEL. *A Virtual Memory Management Scheme For Simulation Enviroment*, 1 (2012), p.114,115,116.

[3] John L. Hennessy , David A. Patterson, "Computer architecture (2nd ed.): a quantitative approach", Morgan Kaufmann Publishers Inc., San Francisco, CA, 1996

[4] David A. Patterson and John L. Hennessy, "Computer Organization and Design the hardware/software interface, 3rd edition", *India*: Morgan Kaufman Publishers, 2004.

[5] Awang Aizzuddin Sulong B Awang Sabli,"*Modelling and verification of 32 Megabyte synchronous dynamic random access memory using verilog,"* University of Tunku Abdul Rahman, Faculty of Information and Communication Technology, 2008.

[6] Kim Yuh Chang, "Design and Development of Memory System for 32 bits 5-stage Pipelined Processor: Main Memory (DRAM) Integration" University of Tunku Abdul Rahman, Faculty of Information and Communication Technology, 2012.

[7] Ruchir P. and Jun Gu, *"An Efficient Algorithm to Search for Minimal Closed Covers in Sequential Machines"*. [online] Available at: http://www.research.ibm.com/da/publications/pap1.pdf [Accessed: 18 AUGUST 2013].

[8] K.M Mok, *Digital System Design Notes*, University of Tunku Abdul Rahman, Faculty of Information and Communication Technology, 2013.

[9] Herveille, R. (2010) *Wishbone B4* . 4th ed. United State: OpenCores Organization.

[10] Zhi Kang Oon, "SDRAM Enhancement: Design of a SDRAM Controller WISHBONE Industrial Standard" University of Tunku Abdul Rahman, Faculty of Information and Communication Technology, 2008.

[11] Lattice Semiconductor (2014) SDR SDRAM Controller Reference Design RD1174 [online] Available at : http://www.latticesemi.com/~/media/Documents/ReferenceDesigns/SZ/SDRSSDRSDRAMContro-Documentation.pdf?document_id=49626 [Accessed: 21 July 2014]

[12] Xilinx (2000) Synthesizable High Performance SDRAM Controller [online] Available at: http://wenku.baidu.com/view/a25cfd7002768e9951e73851.html [Accessed : 21 July 2014]

[13] Lattice Semiconductor (2014) SDR SDRAM Controller Reference Design RD1010 [online] Available at : http://www.latticesemi.com/~/media/Documents/ReferenceDesigns/1D/AdvancedSDRSDRAMController-DesignDocumentation.pdf?document_id=3467 [Accessed: 21 July 2014]

[14] Micron (n.d) 128Mb x 32 Synchronous DRAM [online] Available at: http://pdf1.alldatasheet.com/datasheet-pdf/view/75877/MICRON/MT48LC4M32B2.html [Accessed: 21 July 2014]

Appendices

Appendix A: System Specification

Chip level design: RISC32 processor

A.1 Feature

	Basic RISC32	Full RISC32
Dummy Instruction Cache (KB)	16	16
Dummy Data Cache (KB)	16	16
Data width (bits)	32	32
Instruction width (bits)	32	32
General Purpose Register	32	32
Special Purpose Register	HILO, PC	HILO, PC
Pipelined Stage	5	5
Hazard Handling	No	Yes
Interlock Handling	No	Yes
Data Dependency Forwarding	No	Yes
Branch Prediction	Fixed – always invalid	Dynamic – 2bits scheme
Multiplication (size of multiplier and multiplicand)	yes – 32bits	yes – 32 bits
Branch Delay Slot	Not supported	Not supported
Instruction supported	38	38

Table A.1 RISC32 features

A.2 Naming Convention

Module – [lvl]_[mod. name]

Instantiation – [lvl]_[abbr. mod. name]

Pin – [lvl] [Type] _[abbr. mod. name] _ [pin name]

– [lvl]_[abbr. mod. name]_[Type]_[stage]_[pin name]

Abbreviation:

	Description	Case	Available	Remark
lvl	level	lower	c : Chip u : Unit b : Block tb: Test Bench	
mod. name	Module Name	lower all	any	
abbr. mod. name	Abbreviated module name	lower all	any	maximum 3 characters
Type	Pin type	lower	o : output i : input r : register w : wire f- :function	
stage	Stage name	lower all	if, id, ex, mem, wb	
pin name	Pin name	lower all	any	Several word separate by "_"

Table A.2 Naming Convention

A.3 Basic RISC32 processor

A.3.1 Processor Interface

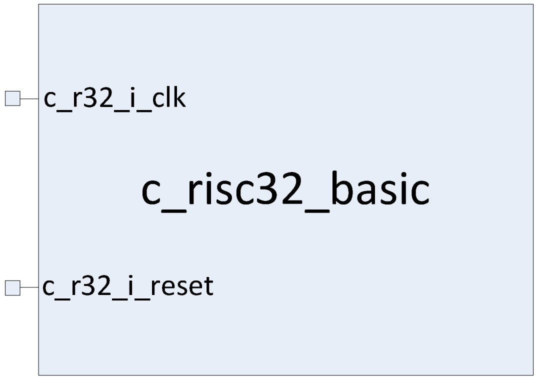


Figure A.3 Block diagram for RISC32-basic processor

A.3.2 I/O Pin Description

Pin Name: c_r32_i_reset	Source → Destination: External Source → RISC32 processor	Registered: No
Pin Function: System reset for the RISC32 microprocessor. It is synchronous to the system clock.		
Pin Name: c_r32_i_clk	Source → Destination: External Source → RISC32 processor	Registered: No
Pin Function: System clock for the RISC32 microprocessor.		

Table A.3 Basic RISC32 Input Pins Description

A.4 System Register

A.4.1 General Purpose Register

Width : 32-bits

Size : 32 units

Retrieving method : 5-bits address as index

Name	Address	Use	Preserved Across A Call?
$zero	0	Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0 - $v1	2 - 3	Value for Function Results and Expression Evaluation	No
$a0 - $a3	4 - 7	Arguments	No
$t0 - $t7	8 – 15	Temporaries	No
$s0 - $s7	16 - 23	Saved temporaries	Yes
$t8 - $t9	24 – 25	Temporaries	No
$k0 - $k1	26 -27	Reserved for OS kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

Table A.4.1 Register file

A.4.2 Special Purpose Register

Width : 32-bits

Size : 2-units

Retrieving method : access using MFHI, MTHI, MFLO, MTLO, MULT and MULTU instructions

Name	definition	location in double [64:0]
HI	Most Significant Word	Double [63:32]
LO	Least Significant Word	Double [31:0]

Table A.4.2 HILO Register

A.4.3 Program Counter Register

Width : 32-bits

Size : 1 unit

Retrieving method : Control by instruction address generator control

A.5 Instruction Format

R-type (Register)					
Op [31:26]	Rs [25:21]	Rt [20:16]	Rd [15:11]	Shamt [10:6]	Funct [5:0]
I-type (Immediate)					
Op [31:26]	Rs [25:21]	Rt [20:16]	Immediate [15:0]		
J-type (Jump)					
Op [31:26]	Target [25:0]				

Table A.5 Instruction Type

Abbreviation:

	Definition	width
op	Operation code (instruction)	6
rs	Source register	5
rt	Target(source/destination) or branch	5
immediate	Immediate, branch displacement or address displacement	16
target	Jump target address	26
rd	Destination register	5
shamt	Shift amount	5
funct	Function field	6

A.6 Addressing Mode

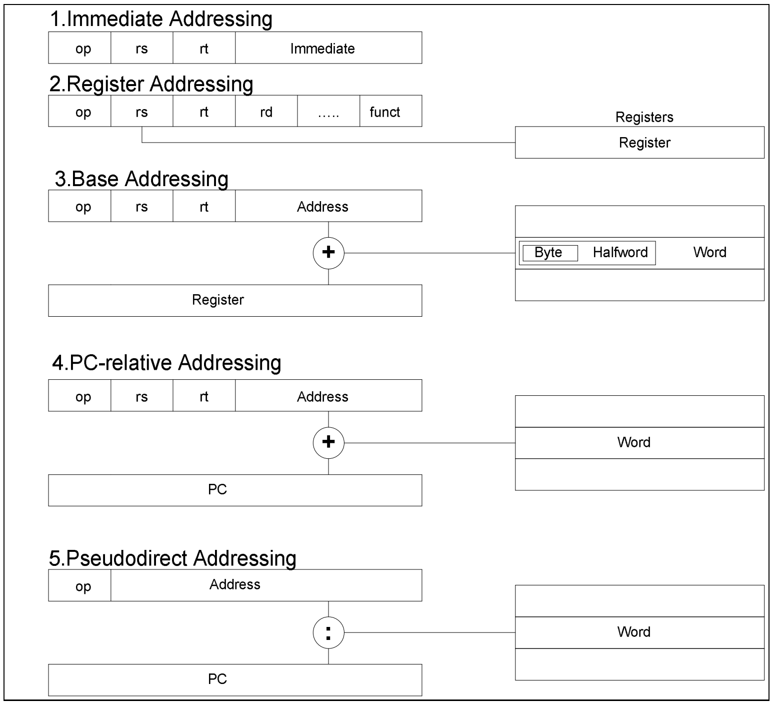


Figure A.6 RISC32 Addressing Mode.

1. *Immediate Addressing*, where operand is constant within the instruction itself

2. *Register Addressing*, where operand is a register

3. *Based Displacement Addressing*, where operand is at the memory location whose address is the sum of a register and a constant in the instruction

4. *PC-relative Addressing*, where branch address s the sum of the PC and a constant in the instruction

5. *Pseudodirect Addressing*, where the jump address is the 26-bits of the instruction concatenated with the upper bits of the PC.

A.7 Instruction Set and Description

Instruction / Assembly	Format	Addr. Mode	Machine Language						Register Transfer Notation	Assembly Format	Overflow
			OpCode	Rs	Rt	Rd	Shamt	Func			
nop	R	Register	0x00	0	0	0	0	0x00	NOP	sll $zero, $zero, 0	no
sll	R	Register	0x00	0	$rt	$rd	n	0x01	R[rd] =R[rs] << n	sll $rd, $rt, n	no
srl	R	Register	0x00	0	$rt	$rd	n	0x03	R[rd] =R[rs] >> n	srl $rd, $rt, n	no
sra	R	Register	0x00	0	$rt	$rd	n	0x04	R[rd] =R[rs] >>> n	sra $rd, $rt, n	no
jr	R	Register	0x00	$rs	0	0	0	0x0A	PC = R[rs]	jr $rs	no
jalr	R	Register	0x00	$rs	0	0	0	0x0B	PC = R[rs] R[31] = PC + 4	jalr $rs	no
mfhi	R	Register	0x00	0	0	$rd	0	0x10	R[rd] = HI	mfhi $rd	no
mthi	R	Register	0x00	$rs	0	0	0	0x11	HI = R[rs]	mthi $rs	no
mflo	R	Register	0x00	0	0	$rd	0	0x12	R[rd] = LO	mflo $rd	no
mtlo	R	Register	0x00	$rs	0	0	0	0x13	LO = R[rs]	mtlo $rs	no
mult	R	Register	0x00	$rs	$rt	0	0	0x24	HILO = R[rs] * R[rt]	mult $rs, $rt	no
multu	R	Register	0x00	$rs	$rt	0	0	0x24	HILO = U(R[rs]) * U(R[rt])	multu $rs, $rt	no
add	R	Register	0x00	$rs	$rt	$rd	0	0x20	R[rd] = R[rs] + R[rt]	add $rd, $rs, $rt	yes
addu	R	Register	0x00	$rs	$rt	$rd	0	0x21	R[rd] = U(R[rs]) + U(R[rt])	addu $rd, $rs, $rt	no
sub	R	Register	0x00	$rs	$rt	$rd	0	0x22	R[rd] = R[rs] - R[rt]	sub $rd, $rs, $rt	yes
subu	R	Register	0x00	$rs	$rt	$rd	0	0x23	R[rd] = U(R[rs]) - U(R[rt])	subu $rd, $rs, $rt	no
and	R	Register	0x00	$rs	$rt	$rd	0	0x24	R[rd] = R[rs] & R[rt]	and $rd, $rs, $rt	no
or	R	Register	0x00	$rs	$rt	$rd	0	0x25	R[rd] = R[rs] | R[rt]	or $rd, $rs, $rt	no
xor	R	Register	0x00	$rs	$rt	$rd	0	0x26	R[rd] = R[rs] ^ R[rt]	xor $rd, $rs, $rt	no
nor	R	Register	0x00	$rs	$rt	$rd	0	0x27	R[rd] = ~(R[rs] | R[rt])	nor $rd, $rs, $rt	no
slt	R	Register	0x00	$rs	$rt	$rd	0	0x2A	R[rd] = (R[rs] < R[rt]) ? 1 : 0	slt $rd, $rs, $rt	no
sltu	R	Register	0x00	$rs	$rt	$rd	0	0x2B	R[rd] = (U(R[rs]) < U(R[rt])) ? 1 : 0	sltu $rd, $rs, $rt	no
j	J	Pseudo-Direct	0x02	JumpAddr (Label)					PC = {(PC+4) [31:28], JumpAddr, 2'b00}	j label	no
jal	J	Pseudo-Direct	0x03	JumpAddr (Label)					PC = {(PC+4) [31:28], JumpAddr, 2'b00} R[31] = PC + 4	jal label	no

beq	I	PC-Relative	0x04	$rs	$rt	BranchAddr (Label)	PC = (R[rs] == R[rt]) ? (PC + 4 + (SE(BranchAddr)<<2)) : (PC + 4)	beq $rs, $rt, label	no
bne	I	PC-Relative	0x05	$rs	$rt	BranchAddr (Label)	PC = (R[rs] != R[rt]) ? (PC + 4 + (SE(BranchAddr)<<2)) : (PC + 4)	bne $rs, $rt, label	no
blez	I	PC-Relative	0x06	$rs	0	BranchAddr (Label)	PC = (R[rs] <=0) ? (PC + 4 + (SE(BranchAddr)<<2)) : (PC + 4)	blez $rs, $rt, label	no
bgtz	I	PC-Relative	0x07	$rs	0	BranchAddr (Label)	PC = (R[rs] > 0) ? (PC + 4 + (SE(BranchAddr)<<2)) : (PC + 4)	bgtz $rs, $rt, label	no
addi	I	Immediate	0x08	$rs	$rt	Imm	R[rt] = R[rs] + SE(Imm)	addi $rt, $rs, imm	yes
addiu	I	Immediate	0x09	$rs	$rt	Imm	R[rt] = U(R[rs]) + U(ZE(Imm))	addiu $rt, $rs, imm	no
slti	I	Immediate	0x0A	$rs	$rt	Imm	R[rt] = (R[rs] < SE(Imm)) ? 1 : 0	slti $rt, $rs, imm	no
sltiu	I	Immediate	0x0B	$rs	$rt	Imm	R[rt] = (U(R[rs]) < U(SE(Imm))) ? 1 : 0	sltiu $rt, $rs, imm	no
andi	I	Immediate	0x0C	$rs	$rt	Imm	R[rt] = R[rs] & ZE(Imm)	andi $rt, $rs, imm	no
ori	I	Immediate	0x0D	$rs	$rt	Imm	R[rt] = R[rs] | ZE(Imm)	ori $rt, $rs, imm	no
xori	I	Immediate	0x0E	$rs	$rt	Imm	R[rt] = R[rs] ^ ZE(Imm)	xori $rt, $rs, imm	no
lui	I	Immediate	0x0F	$rs	$rt	Imm	R[rt] = Imm << 16	lui $rt, imm	no
lw	I	Based-Displacement	0x23	$rs	$rt	Imm	R[rt] = MEM[R[rs] + SE(Imm)]	lw $rt, imm($rs)	no
sw	I	Based-Displacement	0x2B	$rs	$rt	Imm	MEM[R[rs] + SE(Imm)] = R[rt]	sw $rt, imm($rs)	no

Table A.7 RISC32 Instruction set

A.8 Memory Map

Purpose	start address	Direction	Segment
Kernel module	0xC000 0000	Up	Kseg2
Boot Rom		Up	Kseg1
i/o register(if below 512MB)	0xA000 0000	Up	
Direct view of memory to 512MB linux kernel code and data		Up	Kseg0
Exception Entry point	0x8000 0000	Up	
Stack	0x7fff ffff	Down	Kuseg
Program heap	0x1000 8000	Up	
Dynamic library code and data	0x1000 0000	Up	
Main program	0x0040 0000	Up	
Reserved	0x0000 0000	Up	

Table A.8 Memory Map

Memory map description

Kernel module

- Accessible by kernel*

Boot Rom

- Start up ROM which keep the system configuration*

I/O registers (if below 512MB)

- External IO device register*

Direct view of memory to 512MB linux kernel code and data

- *

Exception Entry point

- Software exception handling *

Stack

- Use for argument passing

Program heap

- Dynamic memory allocation such as malloc()

Dynamic library code and data

- Data segment which is access by

Main program

- Text segment which contain the main program

Reserved

Note *: required CP0

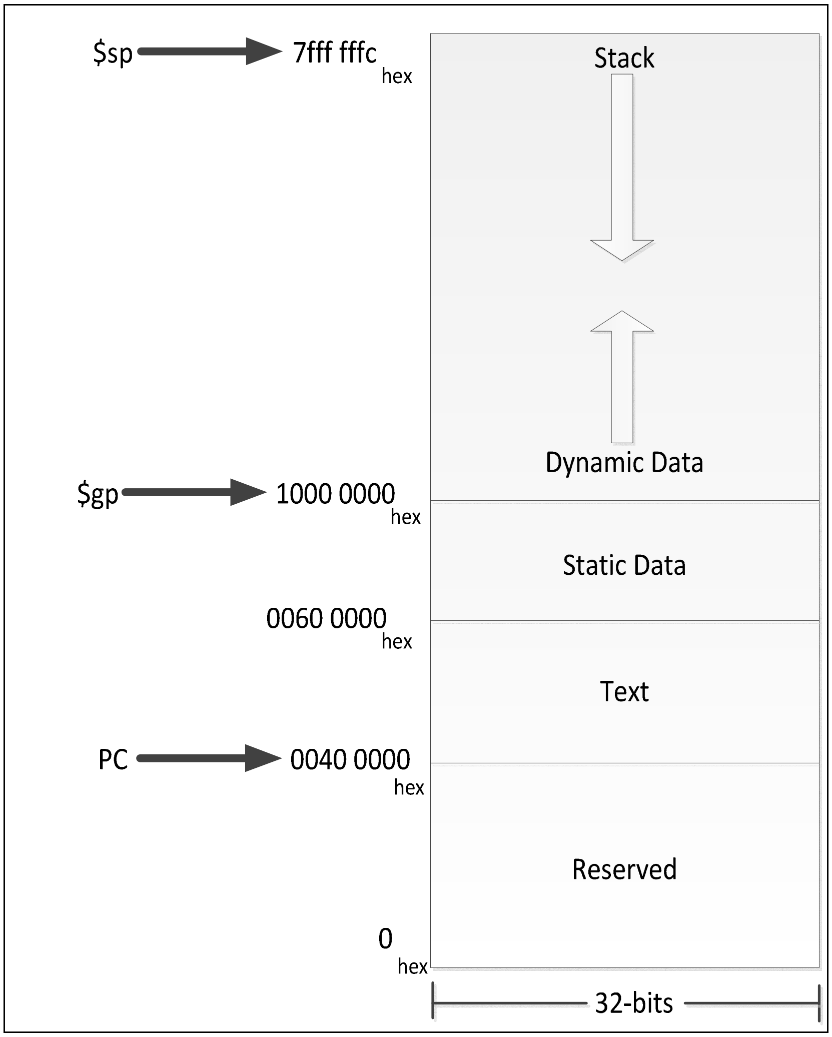


Figure A.8 Memory map for Kuseg section, accessible without CP0

A.9 Operating Procedure

- Start the system
- Porting sequence of instruction into cache (instruction or data)
- Reset the system for at least 2 clocks
- While release the reset, the system will automatically run the program inside instruction cache
- Observe the waveform from the development tools.