DESIGN AND DEVELOPMENT OF MEMORY SYSTEM FOR 32-BIT 5 STAGE PIPELINE RISC: MEMORY SYSTEM INTEGRATION

BY

GOH DIH JIANN

A REPORT

SUBMITTED TO

UniversitiTunku Abdul Rahman

in partial fulfilment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONS)

COMPUTER ENGINEERING

Faculty of Information and Communication Technology (Perak Campus)

OCTOBER 2015

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: _____
_____
_____

Academic Session: _____

I _____
(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,

_____ _____
(Author's signature) (Supervisor's signature)

Address:
_____
_____
_____

_____
Supervisor's name

Date: _____ Date: _____

DESIGN AND DEVELOPMENT OF MEMORY SYSTEM FOR 32-BIT 5 STAGE PIPELINE RISC:
MEMORY SYSTEM INTEGRATION

BY
GOH DIH JIANN

A REPORT
SUBMITTED TO
UniversitiTunku Abdul Rahman
in partial fulfilment of the requirements
for the degree of
BACHELOR OF COMPUTER SCIENCE (HONS)
COMPUTER ENGINEERING
Faculty of Information and Communication Technology (Perak Campus)

OCTOBER 2015

DECLARATION OF ORIGINALITY

I declare that this report entitled "**DESIGN AND DEVELOPMENT OF MEMORY SYSTEM FOR 32-BIT 5 STAGE PIPELINE RISC: MEMORY SYSTEM INTEGRATION"** 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 : ____GOH DIH JIANN_____

Date : _____14/12/2015_____

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my gratitude to my final year project supervisor, Mr. Mok Kai Ming, who encourage me when I lost confidence, comfort me when I am stressed, and enlighten me when I lost my way. A million appreciation and thank for his guidance and wisdom during the entire course of this project. Lastly, I would like to say thanks to my parents for their unconditional support during my hard time throughout the course.

ABSTRACTS

This project is to enhance the current RISC32 architecture that developed in Universiti Tunku Abdul Rahman under Faculty of Information and Communication Technology by redesigning the memory system. After reviewing the previous work, the RISC32 processor memory system cache unit using write-through scheme which is able to improve more of it efficiency.

Hence, this project is initiated to redesign the cache unit into write-back cache and adding a write buffer(FIFO) in the cache unit to handling the data transferring back to SDRAM when read miss and write miss occur. Some modification on memory arbiter was done in order for the new cache unit worked in the memory system. This project is modelled using Verilog HDL and a test program will be developed in order to test the functionality and compatibility of the newly design write-back cache with the rest of memory system (memory arbiter, SDRAM controller, SDRAM).

TABLE OF CONTENTS

Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

MIPS	Microprocessor without Interlocked Pipeline Stages
RISC	Reduced Instruction Set Computing
CPU	Central Processing Unit
RTL	Register Transfer Level
I/O	Input output
FIFO	First In First Out
SOC	System On Chip

Chapter 1 Introduction

1.1 Background Information

The growing disparity between microprocessor and memory cause by the division of the semiconductor industry into CPU fields and memory fields which their technology have focus on different achievement, the first one has concentrated on increased in speed, while the latter one has concentrated on increased in capacity. Thus the improvement rate in microprocessor speed by far exceeds the one in memory. The continuous growing gap between CPU and memory speeds is a crucial flaw in the overall computer performance. Throughout the history, CPU speeds have been improving at an average of 55% per year, while memory latency has only been improving at 7% per year (Hennessy and Patterson 2007, p. 289).

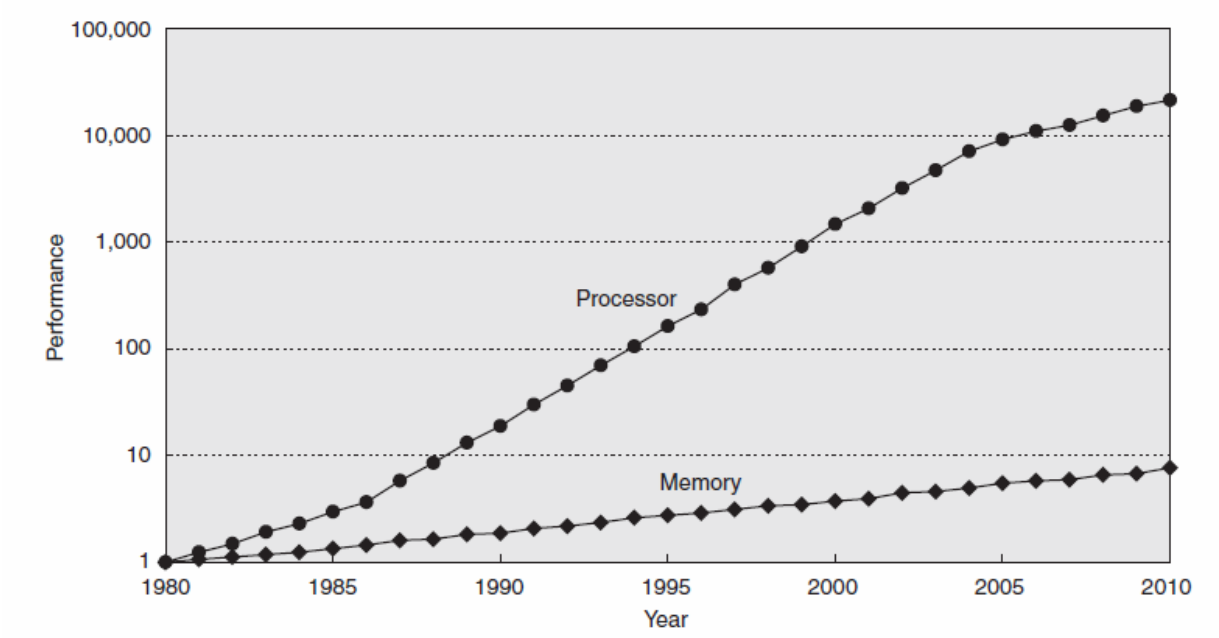


Figure 1-1-1 starting with 1980 performance as a baseline, the gap in performance between memory and processors is plotted over time.

The performance gap grows exponentially. This make increasing processor-memory performance gap is now the leading direction to improved computer system performance.

Memory Hierarchy was introduced in the late of sixties to provide decreased average latency and reduced bandwidth requirements to speed up memory system. The performance of a memory-hierarchy analyse through the average memory access time, using the following expression:

average memory access time = hit time + miss rate * miss penalty.

(Araújo 2002, p.146)

Thus the effort to decrease the performance gap between processor and physical memory has been concentrated on efficient implementations of a memory hierarchy to reduce miss rate, miss penalty and hit time.

1.2 Motivation and Problem Background

A 32-bit RISC processor has been developed in Faculty of Information and Communication Technology, University Tunku Abdul Rahman (UTAR). The project is based on Reduced Instruction Set Computing (RISC) architecture. There are several purposes to initiate this project.

- Microchip design companies develop microprocessor cores as IP (Intellectual Property) for commercial purposes only. This simply means that the microprocessor IP which includes information of the entire design process for front-end and back-end IC design are trade secrets of the company and certainly not available in market at affordable price. Hence, RISC32 project is started at University Tunku Abdul Rahman few years ago and still working to complete the design.

- There are several freely available microprocessor cores from open source such as OpenCores (opencores.org) which is the largest site for development of hardware IP cores as open source. However these processors are not complete and did not implement the entire MIPS Instruction Architecture (ISA). Furthermore, they are lack of comprehensive documentation which makes them not suitable for reuse and further customization.

- Verification is important for proving the functionality of any digital design. The microprocessors mentioned above are handicapped by incomplete and poorly developed verification specifications. This hampers the verification process, slowing down the overall design process.

- The lack of well-developed verification specifications for these microprocessor cores will certainly affect the physical design phase. A design need to be functionally proven before the physical design phase can proceed smoothly. Otherwise, if front-end design requires changing, the entire physical design needs to be redone.

1.3 Problem Statement

This project is aim to provide a solution to the above problems by creating a 32-bit RISC core-based development environment to assist research work in the area of soft-core and also application specific hardware modelling. Currently, a SDRAM Controller and SDRAM provided by MICRON Technology Inc. has been modelled at the Register Transfer Level (RTL) using Verilog HDL and both of them have been combined together and had gone through a series of simulation test. There is also a cache and a TLB modelled at RTL using Verilog HDL, both of them were integrated together with the SDRAM controller as a complete memory system.

Seniors of UTAR FICT computer engineering implemented cache unit, memory arbiter and SDRAM controller. In previous implementation, cache unit is a write-through 2-way set associative caches which it can be improved. Thus this project aim to redesign the cache unit into a write-back multiword direct mapped cache with write buffer (FIFO). The cache unit's protocol need to redesign because of the inclement of write-back ability in cache unit. After implemented the new cache unit, a little modification needs to be done in memory arbiter unit in order to compatible with the new cache unit. After that the functionality need to verify so that every unit is working as expected.

Chapter 2 Literature Review

2.1 Write-through Scheme vs Write-back Scheme

Write-through cache: Data are written into the cache and sent to the main memory (in this project is SDRAM) as operation is executed. This ensures that the contents of the cache and main memory are always the same, but it has downside that it experiences latency based on writing to SDRAM. This cache is good for application that writes and then re-read data frequently.

Write-back cache: Write-back cache keep stored data in the cache, and when a block that has been written is evicted from the cache, the contents of the block are then written back (copied) into the main memory (SDRAM). Write-back cache keep stored data in the cache, the main memory become the same after the contents of the block are written back (copied) into main memory. The disadvantage is there is data availability exposure risk because the only copy of the written data is in cache. Write-back cache is the best performing solution for mixed workloads as both read and write have similar response time levels. (Carter 2002)

This mean that if use write-through cache system performance is limited by memory speed whereas if use write-back cache the cache will get the full performance.

2.2 Write buffer

Data is not written to the main memory directly but into the write buffer first. Once the data is written into the write buffer and assuming cache hit, the CPU is done with the write, then the SDRAM controller will move the write buffer's contents to the real memory behind the scene. This work as long as the frequency of store is not too high.

2.2.1 Write Buffer Saturation

When store frequency approaching main memory write frequency it leads to write buffer saturation. In this case no matter how big the write buffer it is it will still overflow because data simply come in faster than it can empty it, thus CPU will running at main memory cycle time, which is very slow. The solution for write buffer saturation

is to get rid of this write buffer and replace this write through cache with a write back cache. (Mok KM 2009)

2.2.2 Write-back Scheme with Write Buffer

Write buffer allow cache to proceed as soon as data is placed in buffer rather than wait the full latency to write the data into memory. Write-back scheme write data to cache only. It makes main memory is not updated and allow cache and memory to be inconsistent. Since data in cache and memory is inconsistent, each block of data requires a dirty bit to indicate a block is modified. If block replacement happen in cache, only evicted dirty block is kept in a write buffer so that it can write-back to memory later. The drawback of this is it has complex hardware.

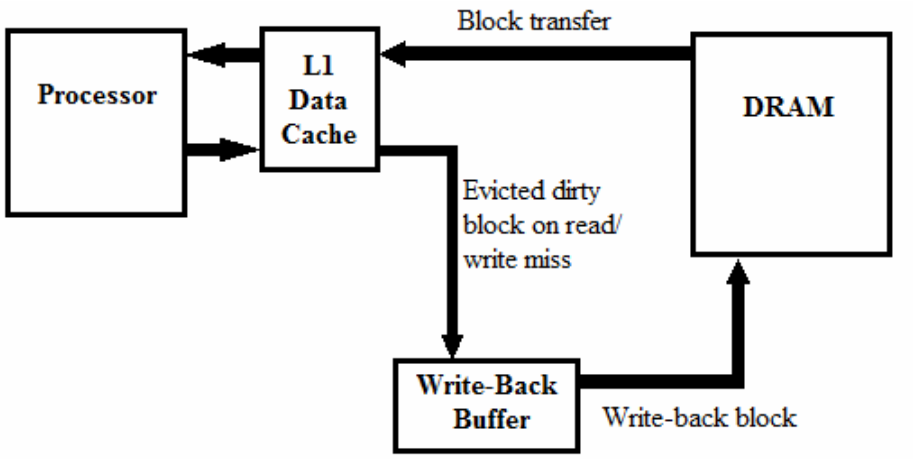


Figure 2-2-1 Write-back scheme with write buffer

2.3 Reduce Miss Rate via Larger Block Size: Multiword Block Direct Mapped Cache

Using multiword block direct mapped cache is the simplest way to reduce miss rate. This take advantage of spatial locality which mean if a word is accessed, nearby words are likely to be accessed soon, thus it is better to move more words per block from memory to cache. However when miss happen it takes more cycle to handle the miss (miss penalty increase).

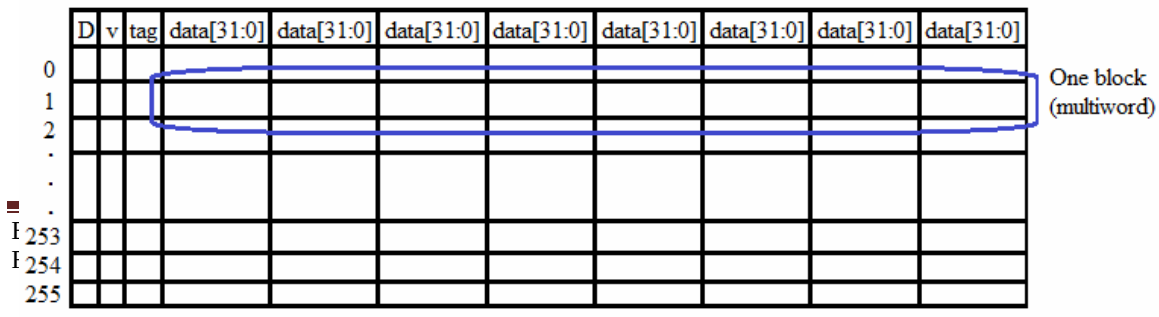


Figure 2-2-2 Multiword block direct mapped cache (block size = 32 bytes)

2.4 Cache Unit

A 2-way set associative write-through cache of 2MB has been modelled by Ching Li-lynn. This cache can be used as both Instruction Cache and Data Cache. Inside of cache unit consists of cache controller block and cache datapath block.

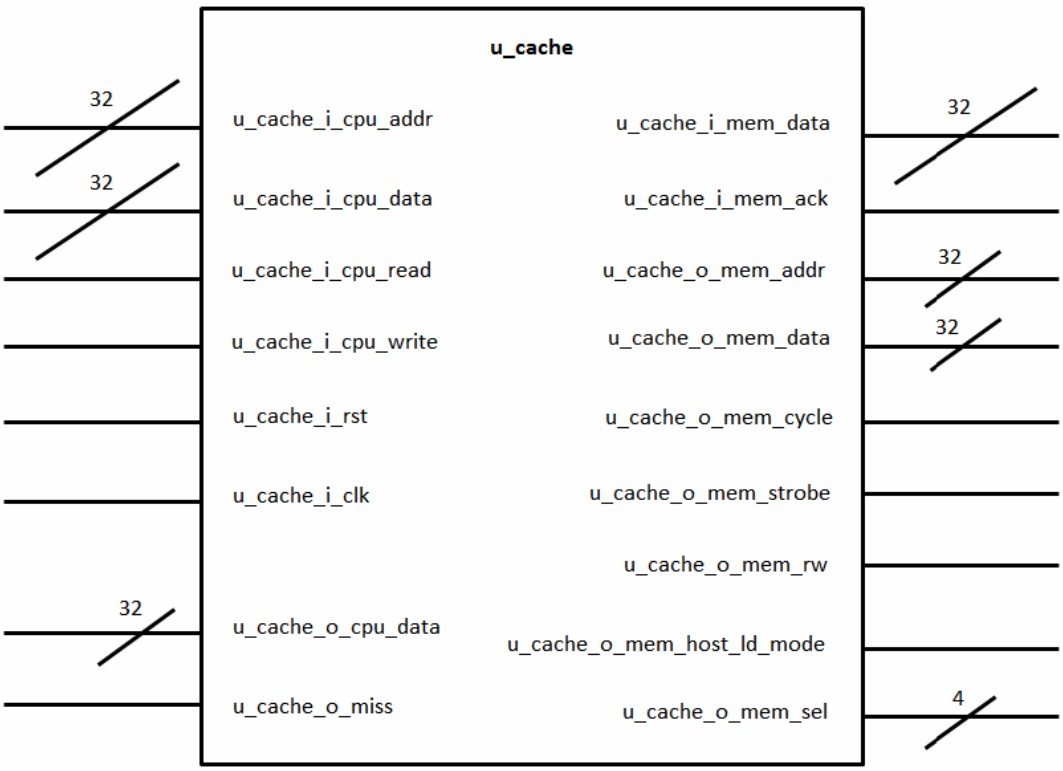


Figure 2-4-1 Cache Unit designed by Ching Li-lynn

2.4.1 Cache Associative

- The current cache is a 2-way set associative cache
- N-Way set associative - uses N cache, data RAMs and N cache-tag RAMs (built out of N RAMs and N comparators, a cache controller, and isolation buffers. It is actually separate the memory into different set of caches and ease the replacement and searching policy.

- 1-way set associative cache = direct mapped cache

2.4.2 Scenarios to Represent Cache Behaviours

Basically there are just 4 scenarios might be happened on cache, we need to decide what to do when these scenarios happen.

1. Read Miss

    - Receive physical address and instructions of read from the main controller of the CPU.
    - Check validity and tag for the index of the physical address points to. A miss signal is produced due to either it is invalid or the tag is different.
    - Cache controller asserts strobe, cycle, and read signals to SDRAM controller to fetch new black of data.
    - Meanwhile, the pipelines of the CPU are stalled.
    - Check LRU to determine which slot is least recently used, store the newly fetched block of data in it.
    - Set valid bit for the index pointed.
    - Update LRU.
    - Deassert the miss, strobe, cycle and read signal, the pipelines are un-stalled.

2. Read Hit

- Receive physical address and instruction of read from the main controller of CPU.
- Check validity and tag for index of the physical address points to. Miss signal is active low.
- Load the selected instruction or data by determining the byte offset to host.
- Update LRU.

3. Write Miss (For D-Cache only)

- Receive physical address, data, and instruction of write from the main controller of CPU.
- Check validity and tag for the index of the physical address points to. A miss signal is produced due to either it is invalid or the tag is different.
- Stall the pipelines.
- Check LRU to determine which is least recently used.
- Cache controller asserts strobe, cycle, and read to SDRAM controller to access the data in SDRAM.
- If the block of data was dirty, send the block of 8 words back to SDRAM.
- Fetch new block of data from SDRAM.
- After the new block is updated from SDRAM, strobe, cycle, read and miss signals are deasserted.
- Perform the write.
- Update LRU.

4. Write Hit (For D-Cache only)

- Receive physical address, data, and instruction of write from main controller of CPU.
- Check validity of tag for index of the physical address points to. Miss signal is active low.
- Update the selected instruction or data.
- Update LRU.

2.4.3 Block Partitioning of Cache Unit

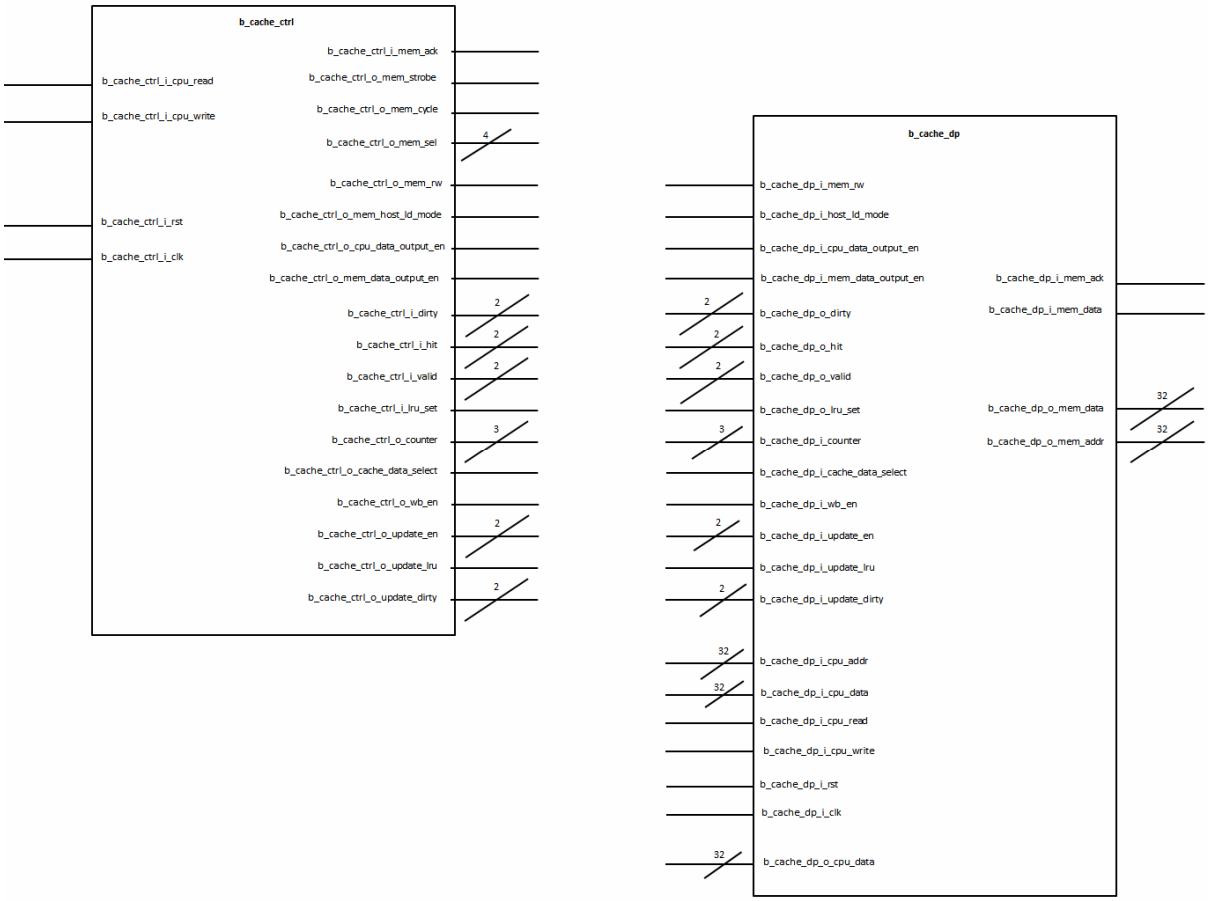


Figure 2-4-2 Block Partitioning of Cache Unit designed by Ching Li-lynn

2.5 SDRAM

Synchronous Dynamic Random Access Memory (SDRAM) is a type of DRAM that is synchronised with the system bus. This project uses a SDRAM that is provided by MICRON Technology Inc. It is MT48LC4M32B2, with 16MB of storage. (Micron datasheet, n.d.) SDRAM control by SDRAM controller modelled by Chin Chun Lek thus in this project just need to focus on function of SDRAM and it configuration – load mode definition.

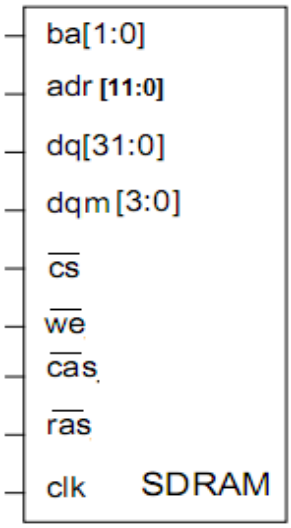


Figure 2-5-1 Block diagram of MT48LC4M32B2 (Oon Zhi Kang 2008)

The cs (active low) pin is used to select the SDRAM, while we, cas and ras are used to request operations from the SDRAM.

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 2-5-1 List of SDRAM commands and function. (Micron datasheet)

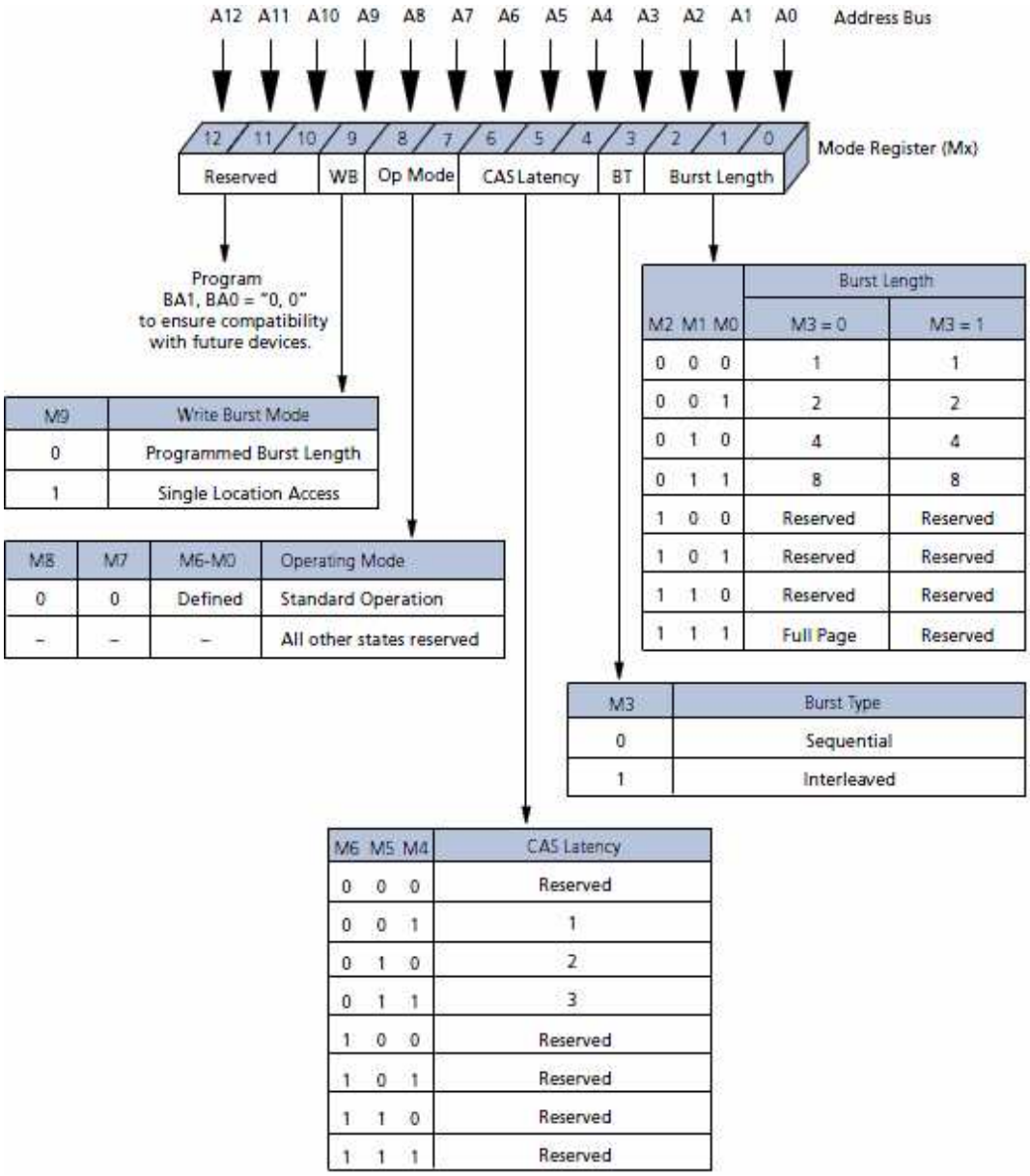


Figure 2-5-2 Mode Register definitions to configure SDRAM (Micron)

- Burst Length

  Determine the maximum number of column locations that can be accessed for a given READ or WRITE operation.

- Burst Type

  Select either sequential or interleaved burst to be adopted by SDRAM. The ordering of accesses within a burst is determined by burst length, burst type, 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

  Select which operating mode should the SDRAM be. Currently there is only normal operating mode is available for use.

- Writing Burst Mode

  When it is ‘0’, the burst length is programmed via M0-M2 applies to both READ and WRITE burst.

  When it is ‘1’, the programmed burst length applies to READ bursts, but write accesses are single-location (non-burst) accesses.

2.6 SDRAM Controller

A SDRAM controller had been modelled by Chin Chun Lek. The SDRAM controller acts as an intermediary between the SDRAM and the CPU. It handles SDRAM operations using some protocols. It has no longer been modeled based on Industry standard HOST SoC interface due to the current design needs.

The main features of SDRAM Controller are:

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

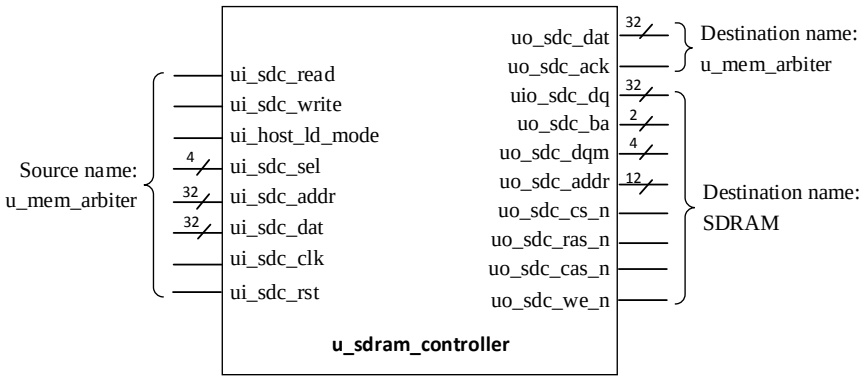


Figure 2-6-1: SDRAM Controller Block Diagram designed by Chin Chun Lek

2.6.1 Block partitioning of SDRAM Controller

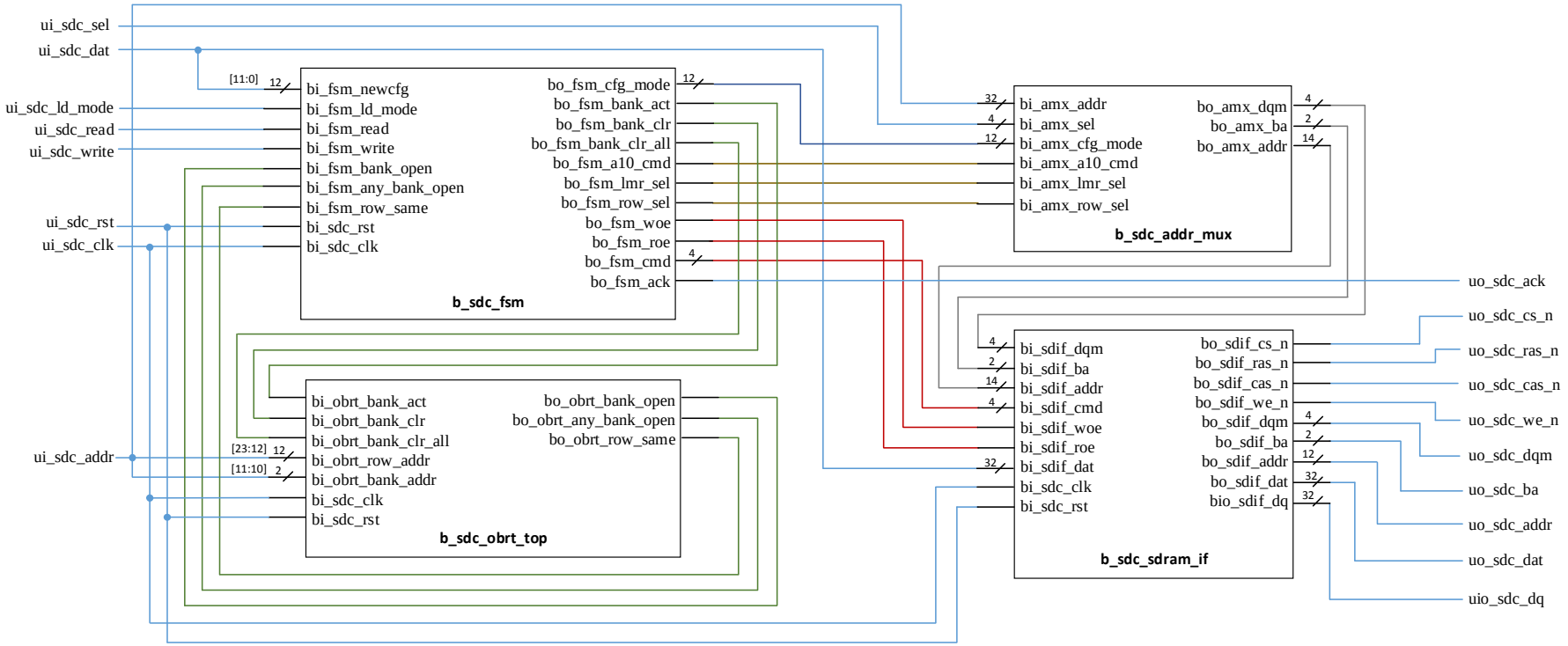


Figure 2-6-2: The Micro-Architecture of the SDRAM Controller designed by Chin Chun Lek

2.7 Memory Arbiter

Chin Chun Lek had modelled a new memory arbiter. This memory arbiter allows multiple caches to access single SDRAM by given priority. The block diagram below shows a memory arbiter that can support up to 4 caches. Some modification needs to be done after that in order to compatible with this project newly designed cache unit.

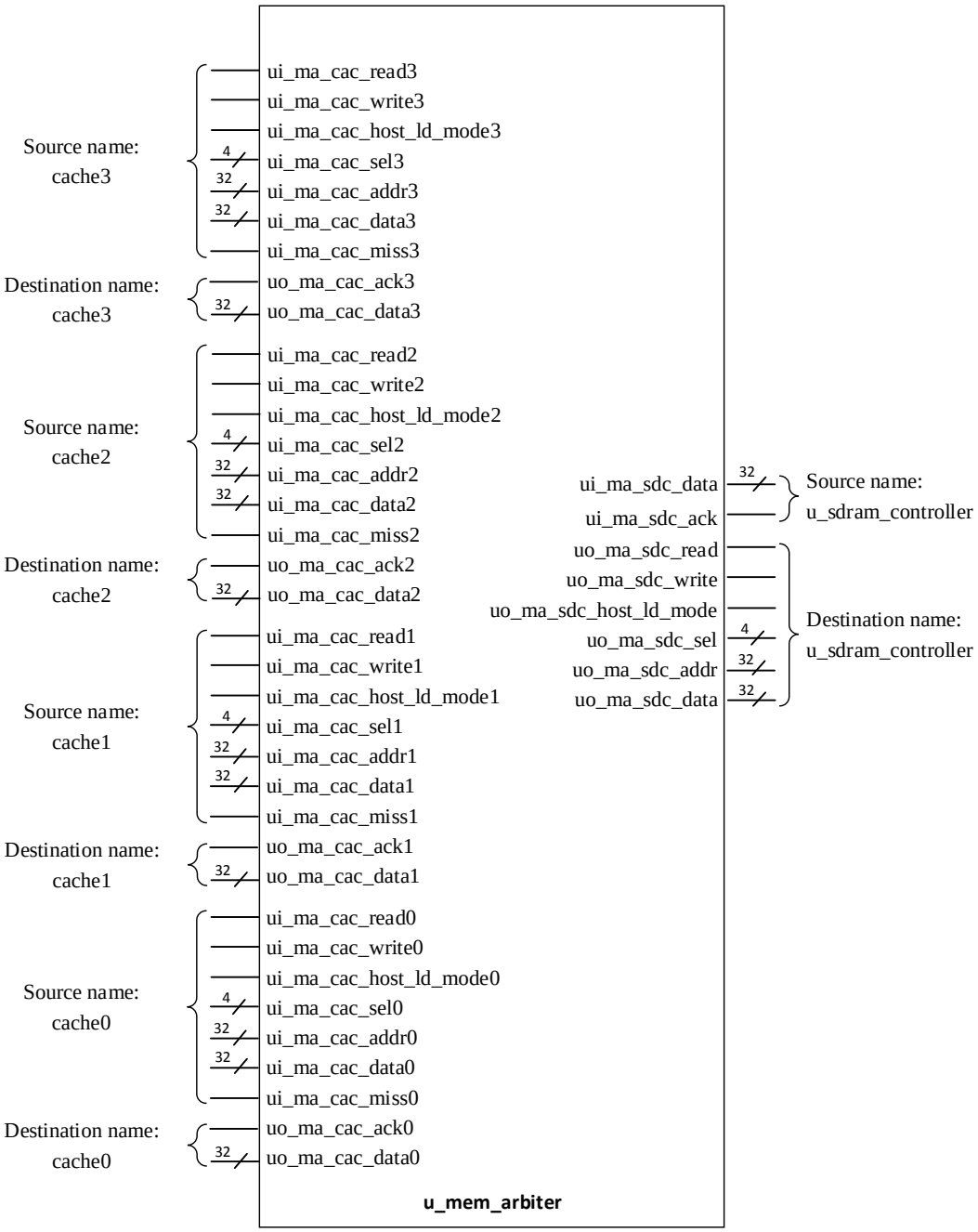


Figure 2-7-1: Memory Arbiter Block Diagram

2.7.1 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 2-7-1: Memory Arbiter I/O Descriptions

2.7.2 Memory Arbiter State Diagram

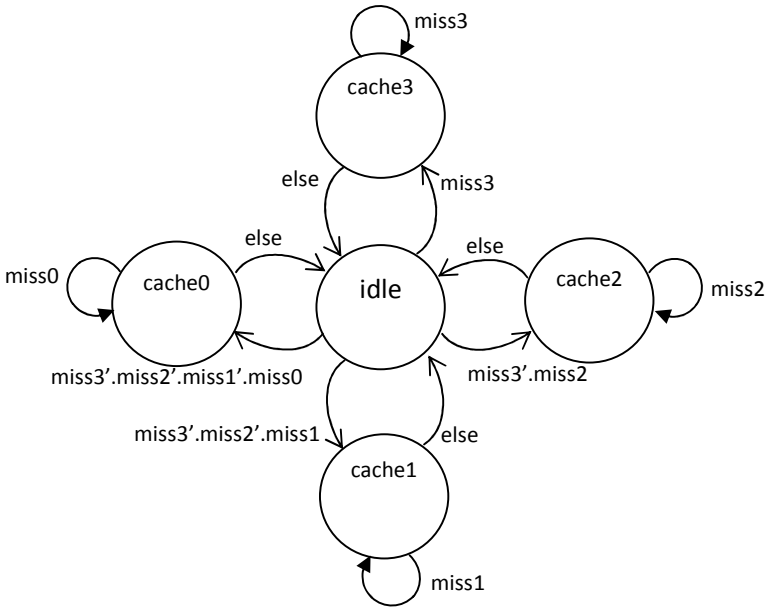


Figure 2-7-2: Memory Arbiter State Diagram

2.7.3 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 2-7-2: State Definition of Memory Arbiter

2.7.4 Output or Behaviors Corresponding to the States

State Name	Correspondence Output Behaviors
cache3	When ui_ma_cac_miss3 = 1, from cache3 to SDRAM controller: uo_ma_sdc_read = ui_ma_cac_read3, uo_ma_sdc_write = ui_ma_cac_write3, uo_ma_sdc_host_ld_mode = ui_ma_cac_host_ld_mode3 uo_ma_sdc_sel = ui_ma_cac_sel3, uo_ma_sdc_addr = ui_ma_cac_addr3, uo_ma_sdc_data = ui_ma_cac_data3 from SDRAM controller to cache3: ui_ma_sdc_ack = uo_ma_cac_ack3, ui_ma_sdc_data = uo_ma_cac_data3
cache2	When ui_ma_cac_miss3 = 0 and ui_ma_cac_miss2 = 1, from cache2 to SDRAM controller: uo_ma_sdc_read = ui_ma_cac_read2, uo_ma_sdc_write = ui_ma_cac_write2, uo_ma_sdc_host_ld_mode = ui_ma_cac_host_ld_mode2 uo_ma_sdc_sel = ui_ma_cac_sel2, uo_ma_sdc_addr = ui_ma_cac_addr2, uo_ma_sdc_data = ui_ma_cac_data2 from SDRAM controller to cache2: ui_ma_sdc_ack = uo_ma_cac_ack2, ui_ma_sdc_data = uo_ma_cac_data2
cache1	When ui_ma_cac_miss3 = 0 and

	ui_ma_cac_miss2 = 0 and ui_ma_cac_miss1 = 1, from cache1 to SDRAM controller: uo_ma_sdc_read = ui_ma_cac_read1, uo_ma_sdc_write = ui_ma_cac_write1, uo_ma_sdc_host_ld_mode = ui_ma_cac_host_ld_mode1 uo_ma_sdc_sel = ui_ma_cac_sel1, uo_ma_sdc_addr = ui_ma_cac_addr1, uo_ma_sdc_data = ui_ma_cac_data1 from SDRAM controller to cache1: ui_ma_sdc_ack = uo_ma_cac_ack1, ui_ma_sdc_data = uo_ma_cac_data1
cache0	When ui_ma_cac_miss3 = 0 and ui_ma_cac_miss2 = 0 and ui_ma_cac_miss1 = 0 and ui_ma_cac_miss0 = 1, from cache0 to SDRAM controller: uo_ma_sdc_read = ui_ma_cac_read0, uo_ma_sdc_write = ui_ma_cac_write0, uo_ma_sdc_host_ld_mode = ui_ma_cac_host_ld_mode0 uo_ma_sdc_sel = ui_ma_cac_sel0, uo_ma_sdc_addr = ui_ma_cac_addr0, uo_ma_sdc_data = ui_ma_cac_data0 from SDRAM controller to cache0: ui_ma_sdc_ack = uo_ma_cac_ack0, ui_ma_sdc_data = uo_ma_cac_data0
idle	All outputs are received zero.

Table 2-7-4: Memory Arbiter Output or Behaviours Corresponding to the States

Chapter 3 Project Scope and Objectives

This project aims to redesign existing memory system by changing write-through scheme to write-back scheme by adding a write buffer (FIFO) to improve the efficiency of previous memory system. A fully functionality verified and synthesis-ready model will be modelled in RTL using the Verilog HDL at the end of this project including the development of test specification, test plan, test vector and testbench which are written in Verilog HDL to ensure functional correctness and the performance.

3.1 Project Objectives

This project's objectives include:

- Design the write-back scheme direct mapped cache unit.
- Design the protocol of cache unit (cache controller block).
- Design the write buffer (FIFO).
- Design the protocol of write buffer (FIFO controller block).
- Modification on memory arbiter to compatible with new cache unit.
- Integration of cache unit, memory arbiter, SDRAM controller and SDRAM.
- Verified the functionality of the integrated unit (cache unit, memory arbiter, SDRAM controller and SDRAM) by construct proper test cases.

3.2 Impact and Significance

As a summary 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 HDL.
- 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 Field Programmable Gate Array (FPGA) with documented timing and resource usage information.

With the available well-developed basic 32-bit RISC RTL model (which has been fully functional verified), the verification environment and the design documents, researchers can develop their own specific RTL model as part of the development environment (whether directly modifying the internals of the processor or interface to the processor) and can quickly verify their model to obtain results, without having to worry about the development of the verification environment and the modeling environment. This can speed up the research work significantly. For example, a researcher may have developed an image-processing algorithm and modified the algorithm to obtain a structure that suits the hardware implementation. The structure can be modeled in Verilog as part of a specialized datapath or as a coprocessor interfacing to the RISC processor.

Chapter 4 Method and Technologies Involved

4.1 Design Methodology

There are several types of design methodologies for design process:

- Top-down design methodology
- Bottom-up design methodology
- Mixed design methodology

A top down design approach was adopted as the main design methodology in this project as shown in the following figure.

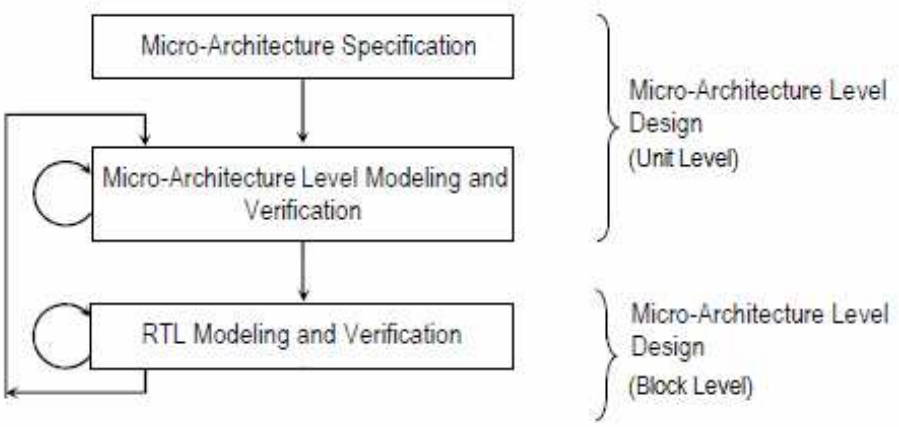


Figure 4-1-1 General Design Flow without Synthesis and Physical Design

This methodology put design partition reduces a complex design into smaller and a manageable piece thus provides step to step guideline that leading to a good design work and development of systems A good design methodology can ensure that functionality correctness in design, satisfaction in term of performance and power goals, can catches bugs at early stage, and provide good documentation for future references (Wolf 2004, p.22).

This project only involved in micro-Architecture level design (Unit Level and Block Level) since higher architecture level had been complete and waiting for integration only.

4.1.1 Micro-architecture Level Design (Unit Level)

The alternate appellation of this level is RTL (Register Transfer Level). This level describes the internal design of architecture unit module with data flow. The unit module is partition into several blocks which each block have its own functionality to carry out the sub-function of the unit module to reduce complexity of design process.

4.1.2 Micro-architecture Level Design (Block Level)

This level further describes each partition from previous level which is block. Their specification are written in this level, normally carry following information such as:

- Functionality / Feature
- Block interface and I/O pin description
- Internal operation which include function table
- Schematic and block diagram
- Test plan
- Timing requirement

Once done with the micro-architecture specification, with the information in the specification, RTL modelling with High Level Language or Hardware Description Language (HDL) can be start. It is combination of behaviour and data flow synthesizable HDL model. Throughout the RTL modelling, Verilog will be use as the design language in this project. The model can be simulate and synthesis. The model is then need to go through verification process which verify the functionality of the design which need to meet the micro-architecture specification. Verification includes development of testbench, timing verification and functionality verification.

4.2 Design Tools

4.2.1 Verilog HDL Simulator - Mentor Graphics ModelSim SE-64 10.1c

Develop using Verilog Hardware Description Language (HDL) require a simulator tool that can provide simulation environment to verify the functional behaviours and waveform simulation. With multiple choices of HDL simulator in the market, a research had been to choose the most appropriate design tools for this project which affect by language supported, availability, price and etc. From the consideration above, ModelSim from Mentor Graphic is the best choice as a design tools for this project as they offer a free license for Student Edition, can found in internet and support Microsoft Windows platform. Although with some limitation, which is slower simulation speed than full version and have code limitation, but it is sufficient for this project as the scope of this project would not reach the limit.

Chapter 5 Memory System Specification

5.1 Partitioning and Design Hierarchy

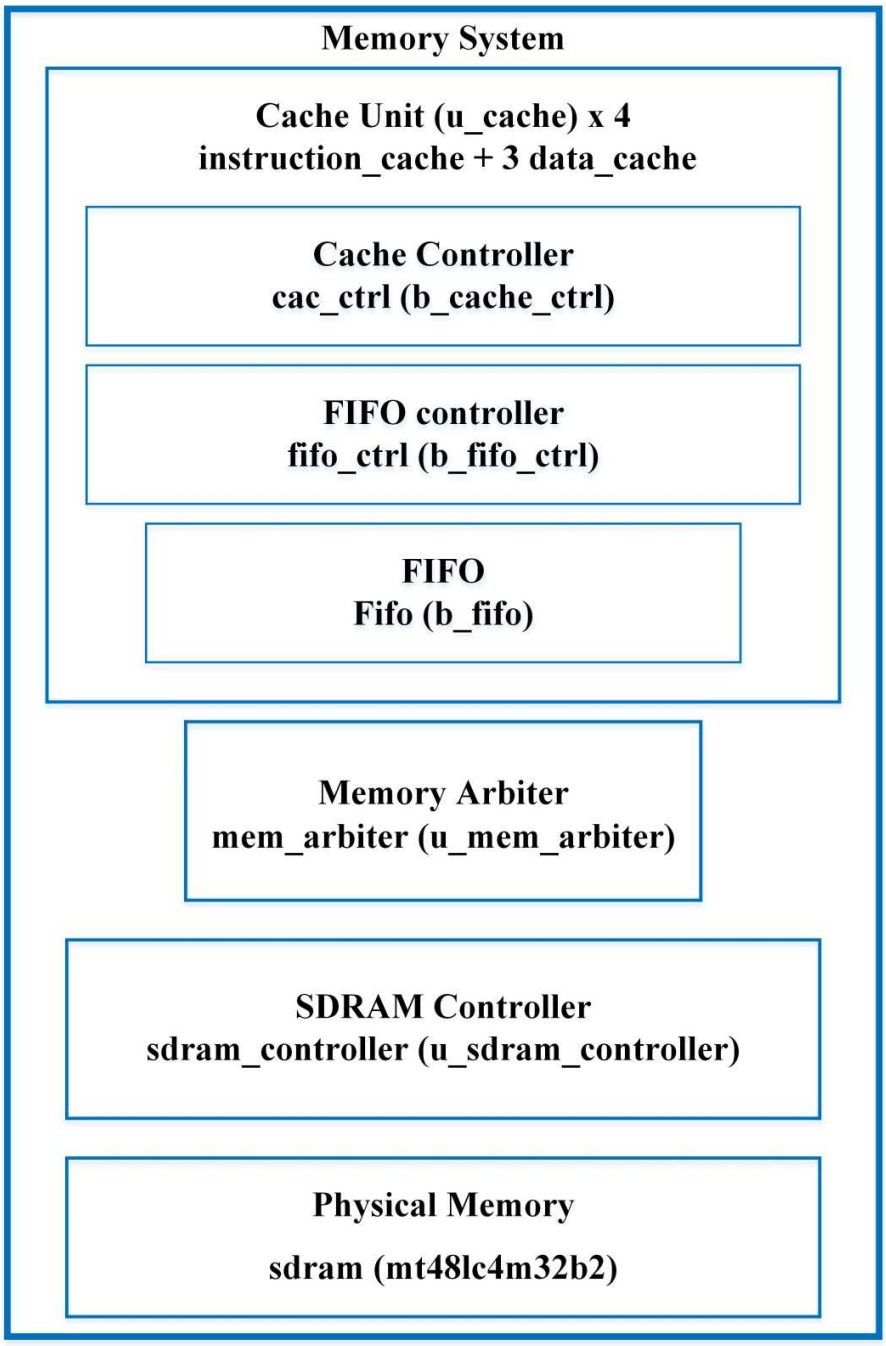


Figure 5-1-1 Memory System Partitioning

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 data)**	**b_cache_ctrl**
		b_fifo_ctrl
		b_fifo
	u_cache (for instruction)	**b_cache_ctrl**
		b_fifo_ctrl
		b_fifo
	u_mem_arbiter	-
	u_sdram_controller	b_sdc_fsm
		b_sdc_sdram_if
		b_sdc_addr_mux
		b_sdc_obrt_top
	sdram (mt48lc4m32b2)	-

Table 5-1-1 Design hierarchy for 32-bit Memory System

5.2 Memory System Specifications

	RISC32 with Integrated Main Memory
SDRAM	16MB
Instruction Cache	Direct mapped write-back cache, 2MB
Data Cache	Direct mapped write-back cache, 2MB
Data Bus Width	32-bits
Instruction Width	32-bits

Table 5-2-1 Specifications of the Memory System

5.3 Memory Map

Segment	Address	Purpose
kseg2 – 1GB	0xFFFF FFFF 0xC000 0000	Kernel module, Page Table allocated here
kseg1 – 512MB	0xBFFF FFFF 0xA000 0000	Boot Rom I/O Register (if below 512MB)
kseg0 – 512MB	0x9FFF FFFF 0x8000 0000	Direct view of memory to 512MB kernel code and data. Exception and Page Table Base Register allocated here.
kuseg – 2GB	0x7FFF FFFF 0x1000 8000	**Stack Segment** starts from the ending address and expand down. **Heap Segment** starts from the starting address and expand top.
	0x1000 7FFF 0x1000 0000	**Data segment** and Dynamic library code.
	0x09FFF FFFF 0x0040 0000	**Code Segment**, where the main program stored.
	0x003F FFFF 0x0000 0000	Reserved

Table 5-3-1 Virtual memory map of 32-bits MIPS

- **Stack Segment**
  - Use for storing automatic variables, which are variables that allocated and de-allocated automatically when program flow.
- **Heap Segment**
  - Use for dynamic memory allocation such as malloc(), realloc() and free().
- **Data Segment**
  - Use for storing global or static variables that initialize by programmer.
- **Code Segment**

- Use for storing codes of main program or main program instructions.

5.4 Architecture of Memory System

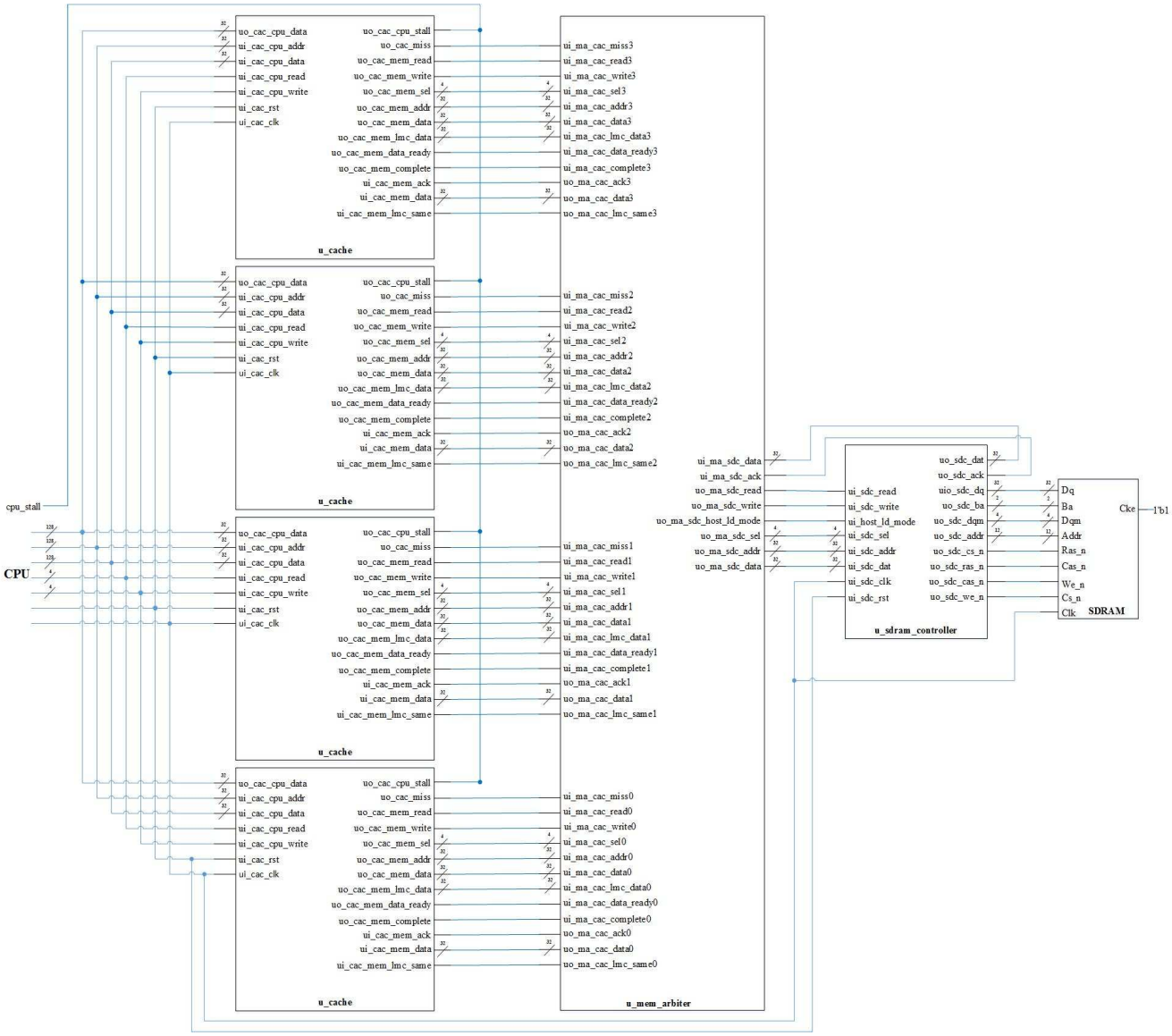


Figure 5-4-1 Architecture of Memory System

Chapter 6 Micro-Architecture Specification

6.1 Cache Unit

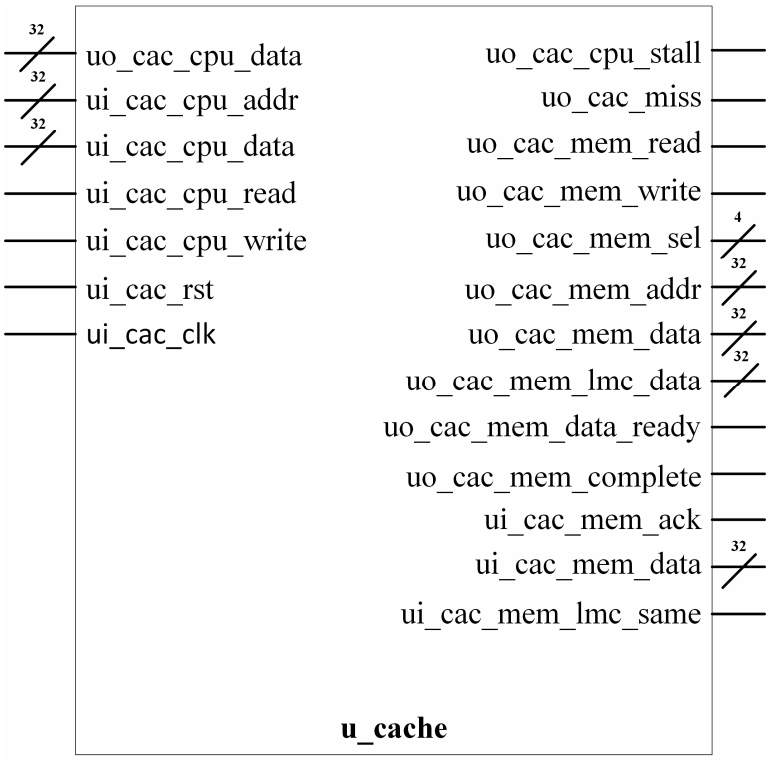


Figure 6-1-1 Block diagram of cache unit

This is a direct mapped write-back cache with write buffer. The functionalities of Cache Unit are:

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.

6.4 Cache Unit I/O Description

Input pins
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[31:0] **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[31:0] **Pin class:** Address **Path:** CPU→ Cache **Description:** 32-bits address from CPU that indicates the 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 based on ui_cac_cpu_addr[31:0] 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 cache based on ui_cac_cpu_addr[31:0] when asserted (HIGH).
Pin name: ui_cac_mem_ack **Pin class:** Control **Path:** Memory Arbiter → Cache

Description: Acknowledge signal (active HIGH) to indicate read data is ready from SDRAM (read from SDRAM) or SDRAM prepare to receive data (write to SDRAM).
Pin name: ui_cac_mem_data[31:0] **Pin class:** Data **Path:** Memory Arbiter → Cache **Description:** 32-bits data that is read from SDRAM.
Pin name: ui_cac_mem_lmc_same **Pin class:** Status **Path:** Memory Arbiter → Cache **Description:** Indicate the configuration of SDRAM is same when asserted (HIGH).
Output pins
Pin name: uo_cac_cpu_data[31:0] **Pin class:** Data **Path:** Cache→ CPU **Description:** 32-bits data that to be output to CPU.
Pin name: uo_cac_cpu_stall **Pin class:** Control **Path:** Cache→ CPU **Description:** A status signal that used to stall the pipelines.
Pin name: uo_cac_miss **Pin class:** Status **Path:** Cache→ Memory Arbiter **Description:** A status signal indicates cache miss.
Pin name: uo_cac_mem_read **Pin class:** Control **Path:** Cache→ Memory Arbiter **Description:** Read signal that indicate need read from SDRAM.
Pin name: uo_cac_mem_write **Pin class:** Control **Path:** Cache→ Memory Arbiter **Description:** Write signal that indicate need write data into SDRAM.

Pin name: uo_cac_mem_sel[3:0] **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[31:0] **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[31:0] **Pin class:** Data **Path:** Cache→ Memory Arbiter **Description:** 32-bits data that to be written in to the SDRAM.
Pin name: uo_cac_mem_lmc_data[31:0] **Pin class:** Data **Path:** Cache→ Memory Arbiter **Description:** 32-bits data that configure the SDRAM.
Pin name: uo_cac_mem_data_ready **Pin class:** Status **Path:** Cache→ Memory Arbiter **Description:** When asserted (HIGH), data is ready write back from FIFO to SDRAM.
Pin name: uo_cac_mem_complete **Pin class:** Status **Path:** Cache→ Memory Arbiter **Description:** Indicates one block of data was written into SDRAM when HIGH.

Table 6-4-1: Cache Unit I/O Descriptions

6.5 Block Partitioning of Cache Unit

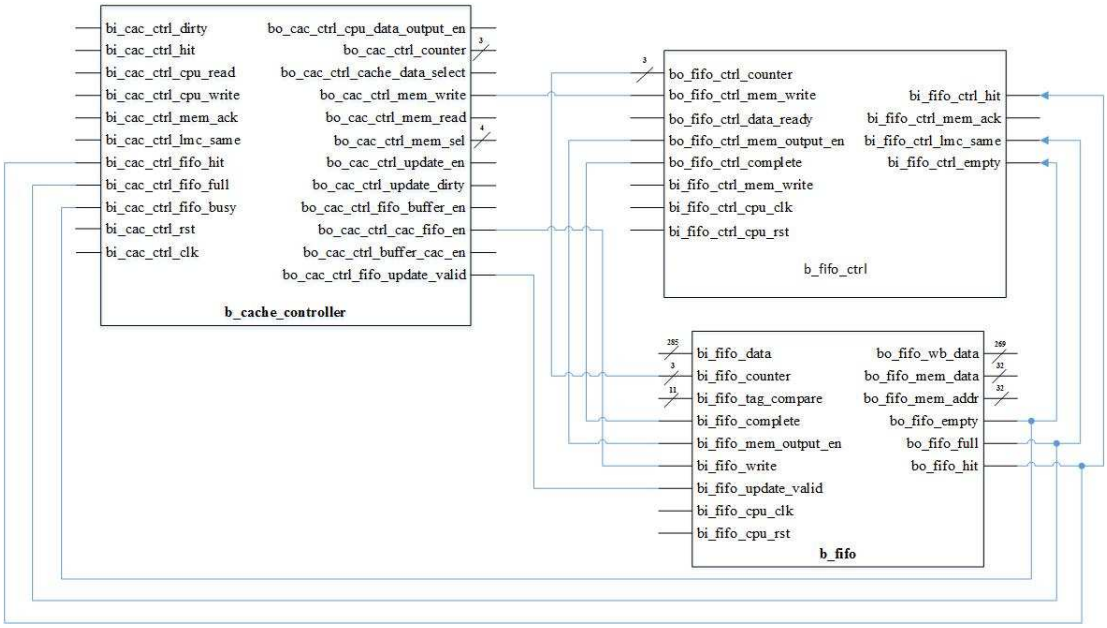


Figure 6-5-1 Block Partition of Cache Unit

6.6 Cache Controller Block

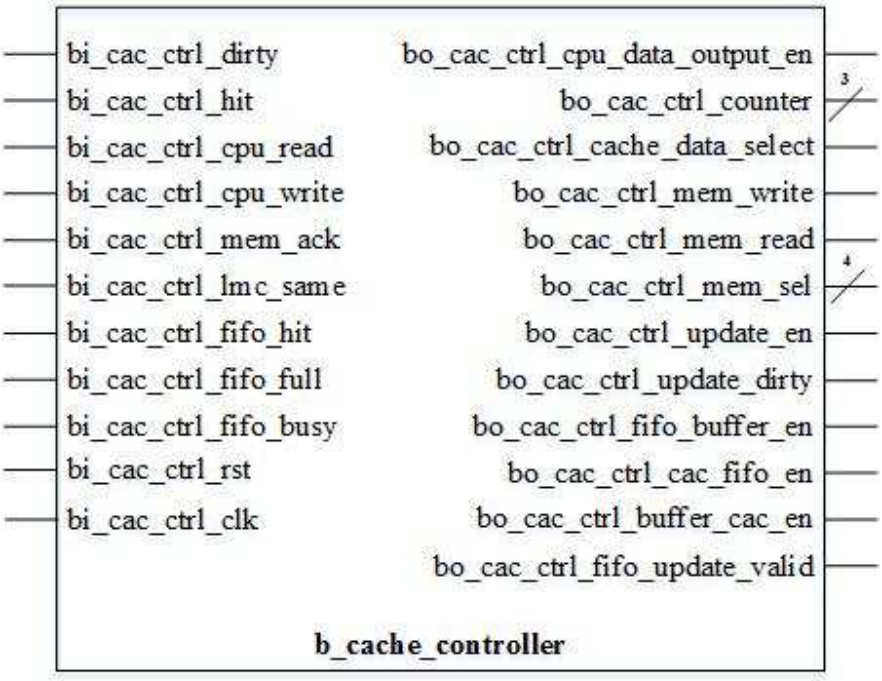


Figure 6-6-1 Block diagram of Cache Controller Block

Functionalities of Cache Controller:

1. Control main activity of cache unit.
2. Determine data to read when read hit.
3. Determine data to be updated when write hit.
4. Determine data to read from SDRAM when miss.
5. Output control signal and status signal to write back data from FIFO to cache.
6. Output control signal to move dirty data from cache to FIFO.
7. Output control signal and status signal out to CPU and SDRAM.

6.6.1 Cache Controller block I/O description

Input pins
Pin name: bi_cac_ctrl_clk **Pin class:** Global **Path:** External → Cache → Cache Controller **Description:** System clock signal.
Pin name: bi_cac_ctrl_rst **Pin class:** Global **Path:** External → Cache → Cache Controller **Description:** System reset signal.
Pin name: bi_cac_ctrl_lmc_same **Pin class:** Status **Path:** Memory Arbiter → Cache → Cache Controller **Description:** Indicates the configuration of SDRAM is same when asserted (HIGH).
Pin name: bi_cac_ctrl_mem_ack **Pin class:** Control **Path:** SDRAM controller → Memory Arbiter → Cache → Cache Controller **Description:** Acknowledge signal (active HIGH) to indicate read data is ready from SDRAM(read from SDRAM) or SDRAM prepare to receive data (write to SDRAM).

Pin name: bi_cac_ctrl_cpu_write **Pin class:** Control **Path:** CPU→ Cache → Cache Controller **Description:** A control signal that enables the write of data into cache based on ui_cac_cpu_addr[31:0] when asserted (HIGH).
Pin name: bi_cac_ctrl_cpu_read **Pin class:** Control **Path:** CPU→ Cache → Cache Controller **Description:** A control signal that enables the read from cache based on ui_cac_cpu_addr[31:0] when it is asserted (HIGH).
Pin name: bi_cac_ctrl_hit **Pin class:** Status **Path:** Cache → Cache Controller **Description:** Asserted when (tag == tag_ram) && (valid_ram == 1).
Pin name: bi_cac_ctrl_dirty **Pin class:** Status **Path:** Cache → Cache Controller **Description:** Asserted when dirty_ram == 1.
Pin name: bi_cac_ctrl_fifo_busy **Pin class:** Status **Path:** FIFO → Cache Controller **Description:** HIGH when FIFO is writing into SDRAM.
Pin name: bi_cac_ctrl_fifo_full **Pin class:** Status **Path:** FIFO → Cache Controller **Description:** Status signal that indicate FIFO is full.
Pin name: bi_cac_ctrl_fifo_hit **Pin class:** Status **Path:** FIFO → Cache Controller **Description:** Status Signal that FIFO contain same tag and index with the physical address tag and index.

Output pins
Pin name: bo_cac_ctrl_cpu_data_output_en **Pin class:** Control **Path:** Cache Controller→ Cache **Description:** When asserted (HIGH), data is enabled to be output to CPU.
Pin name: bo_cac_ctrl_counter[2:0] **Pin class:** Control **Path:** Cache Controller→ Cache **Description:** 3-bits counter value. This is used to count the data when transferring a whole block (8 words) of data.
Pin name: bo_cac_ctrl_cache_data_select **Pin class:** Control **Path:** Cache→ Cache Controller→ Cache **Description:** Instruct the cache datapath which data (data from cpu or data from SDRAM) to be written into. When HIGH, choose data from SDRAM. When LOW, choose data from CPU.
Pin name: bo_cac_ctrl_mem_read **Pin class:** Control **Path:** Cache Controller→Cache→Memory Arbiter→SDRAM Controller →SDRAM **Description:** Read signal that indicate need read from SDRAM.
Pin name: bo_cac_ctrl_mem_write **Pin class:** Control **Path:** Cache Controller→FIFO controller **Description:** Write signal that indicate need write data into SDRAM.
Pin name: bo_cac_ctrl_mem_sel [3:0] **Pin class:** Control **Path:** Cache Controller→ 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: bo_cac_ctrl_update_en **Pin class:** Control **Path:** Cache Controller→ Cache **Description:** Enables the update of cache when asserted (HIGH).
Pin name: bo_cac_ctrl_update_dirty **Pin class:** Control **Path:** Cache Controller→ Cache **Description:** Enables the update of ‘Dirty’ when asserted (HIGH).
Pin name: bo_cac_ctrl_fifo_buffer_en **Pin class:** Control **Path:** Cache Controller→ Cache **Description:** Enable to move write back data from FIFO to temporary buffer.
Pin name: bo_cac_ctrl_cac_fifo_en **Pin class:** Control **Path:** Cache Controller→ Cache **Description:** Enable to move cache data to FIFO.
Pin name: bo_cac_ctrl_buffer_cac_en **Pin class:** Control **Path:** Cache Controller→ Cache **Description:** Enable to move write back data from temporary buffer to cache.
Pin name: bo_cac_ctrl_fifo_update_valid **Pin class:** Control **Path:** Cache Controller→ FIFO **Description:** Control signal that update the valid bit in FIFO.

Table 6-6-1: Cache Controller Block I/O Descriptions

6.6.2 Cache Controller State Diagram

Figure 6-6-2 State Diagram of Cache Controller

6.7 FIFO Controller Block

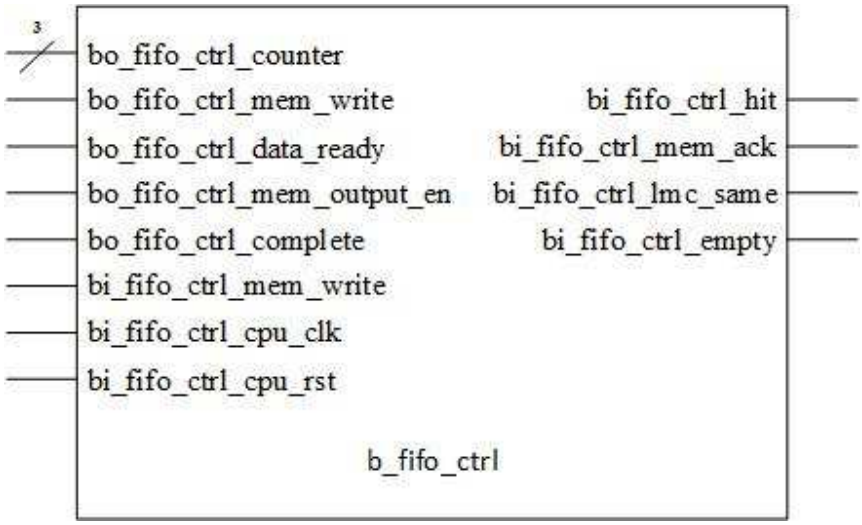


Figure 6-7-1 Block diagram of FIFO Controller Block

Functionalities of FIFO Controller:

1. Control main activity of FIFO block.

2. Send control signal to FIFO to write data back to SDRAM behind the scene.

6.7.1 FIFO Controller block I/O description

Input pins
Pin name: bi_fifo_ctrl_cpu_clk **Pin class:** Global **Path:** External → Cache → FIFO Controller **Description:** System clock signal.
Pin name: bi_fifo_ctrl_cpu_rst **Pin class:** Global **Path:** External → Cache → FIFO Controller **Description:** System reset signal.
Pin name: bi_fifo_ctrl_hit **Pin class:** Status **Path:** FIFO → FIFO Controller **Description:** Status Signal that FIFO contain same tag and index with the physical

address tag and index.
Pin name: bi_fifo_ctrl_mem_write **Pin class:** Control **Path:** Cache Controller → FIFO controller **Description:** Write signal that indicate need write data into SDRAM
Pin name: bi_fifo_ctrl_mem_ack **Pin class:** Control **Path:** SDRAM controller → Memory Arbiter → Cache → FIFO Controller **Description:** Acknowledge signal (active HIGH) to indicate read data is ready from SDRAM(read from SDRAM) or SDRAM prepare to receive data (write to SDRAM).
Pin name: bi_fifo_ctrl_lmc_same **Pin class:** Status **Path:** Memory Arbiter → FIFO Controller **Description:** Indicate the configuration of SDRAM is same when asserted (HIGH).
Pin name: bi_fifo_ctrl_empty **Pin class:** Status **Path:** FIFO → FIFO Controller **Description:** When asserted, it indicate FIFO is empty.
Output pins
Pin name: bo_fifo_ctrl_counter [2:0] **Pin class:** Control **Path:** FIFO Controller→ FIFO **Description:** 3-bits counter value. This is used to count the data when transferring a whole block (8 words) of data.
Pin name: bo_fifo_ctrl_mem_write **Pin class:** Control **Path:** FIFO Controller→ Memory Arbiter **Description:** Write signal that indicate need write data from FIFO into SDRAM.
Pin name: bo_fifo_ctrl_data_ready **Pin class:** Status **Path:** FIFO Controller →Memory Arbiter

Description: When asserted (HIGH), data is ready write back from FIFO to SDRAM.
Pin name: bo_fifo_ctrl_mem_output_en **Pin class:** Control **Path:** FIFO Controller → FIFO **Description:** Enable data in FIFO to be written into SDRAM
Pin name: bo_fifo_ctrl_complete **Pin class:** Control **Path:** FIFO Controller→ Memory Arbiter **Description:** Indicates one block of data was written into SDRAM when HIGH.

Table 6-7-1: Cache Controller Block I/O Descriptions

6.7.2 FIFO Controller State Diagram

Figure 6-7-2 State Diagram of Cache Controller

6.8 FIFO Block

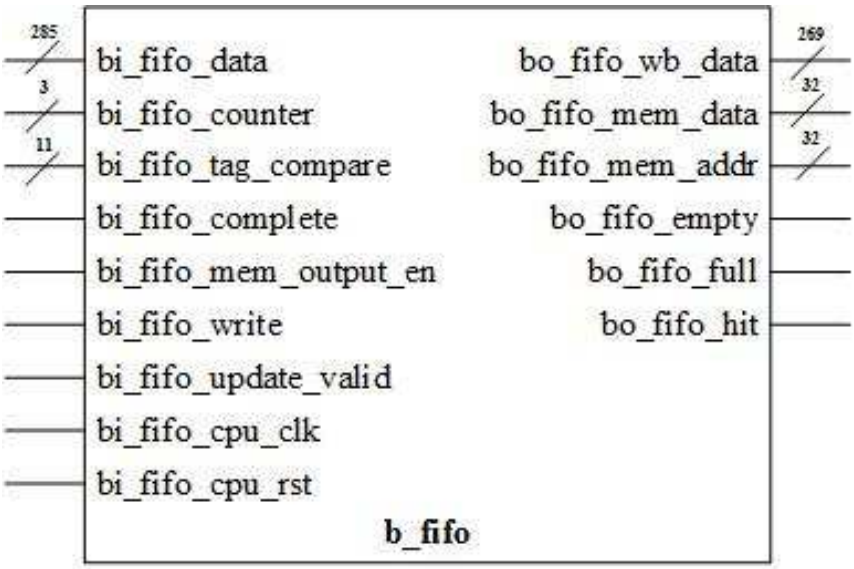


Figure 6-8-1 Block diagram of FIFO Block

This FIFO block consists of 4 entries to store data block from cache. The functionalities of FIFO block are:

1. Store dirty block from cache that need to written back to SDRAM
2. Data able to written back to cache or back to SDRAM.
3. Communicate with SDRAM to written data back to SDRAM when SDRAM is free.
4. Compare tag and index to indicate whether same block of data need to accessed next in cache.
5. Output a full signal when 4 entries are used.
6. Output an empty signal when FIFO contains no data.

6.8.1 FIFO Controller block I/O description

Input pins
Pin name: bi_fifo_cpu_clk **Pin class:** Global **Path:** External → Cache → FIFO **Description:** System clock signal.
Pin name: bi_fifo_cpu_rst **Pin class:** Global **Path:** External → Cache → FIFO **Description:** System reset signal.
Pin name: bi_fifo_update_valid **Pin class:** Control **Path:** Cache Controller → FIFO **Description:** Control signal that update the valid bit in FIFO.
Pin name: bi_fifo_write **Pin class:** Control **Path:** Cache Controller → FIFO **Description:** Write signal that indicate data write from cache to FIFO.
Pin name: bi_fifo_mem_output_en **Pin class:** Control **Path:** FIFO controller → FIFO **Description:** Enable data in FIFO to be written into SDRAM
Pin name: bi_fifo_complete **Pin class:** Status **Path:** FIFO controller → FIFO **Description:** Indicates one block of data was written into SDRAM when HIGH.
Pin name: bi_fifo_tag_compare[10:0] **Pin class:** Address **Path:** Cache → FIFO **Description:** Tag from physical address that used to compare FIFO_hit signal

Pin name: bi_fifo_counter [2:0] **Pin class:** Control **Path:** FIFO Controller→ FIFO **Description:** 3-bits counter value. This is used to count the data when transferring a whole block (8 words) of data.
Pin name: bi_fifo_data [284:0] **Pin class:** Data **Path:** Cache→ FIFO **Description:** contain index from physical address, tag_ram, data_ram and byte_ram from cache.
Output pins
Pin name: bo_fifo_hit **Pin class:** Status **Path:** FIFO Controller→ Cache Controller **Description:** Status Signal that FIFO contain same tag and index with the physical address tag and index.
Pin name: bo_fifo_full **Pin class:** Status **Path:** FIFO → Cache Controller and FIFO Controller **Description:** Status signal that indicate FIFO is full.
Pin name: bo_fifo_empty **Pin class:** Status **Path:** FIFO → Cache Controller **Description:** When asserted, it indicate FIFO is empty.
Pin name: bo_fifo_mem_addr[31:0] **Pin class:** Address **Path:** FIFO → Memory Arbiter → SDRAM controller → SDRAM **Description:** 32-bits address that indicates which location in the SDRAM to be accessed.
Pin name: bo_fifo_mem_data [31:0] **Pin class:** Data

Path: FIFO → Memory Arbiter → SDRAM controller → SDRAM **Description: :** 32-bits data that to be written in to the SDRAM.
Pin name: bo_fifo_wb_data [268:0] **Pin class:** Data **Path:** FIFO → Cache **Description:** Contain all data that need to write back to cache (data, tag and byte).

Table 6-8-1: FIFO Block I/O Descriptions

Chapter 7 Verification

7.1 Test Plan

Function To be Tested	Test Case
Test 1: System Reset	tb_r_rst is asserted to high at least one clock cycle

Test 2: Testing Cache priority and reading in different burst length	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;
	
	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;

Test 3 : Write Hit in Cache 3 and continuous Write Hit	First write instruction,
	tb_r_cpu_cac_data3 = 32'h07070707;
	tb_r_cpu_cac_addr3 = 32'h00567004;
	
	tb_r_cpu_cac_read3 = 0;
	tb_r_cpu_cac_write3 = 1;
	
	Second write instruction,
	tb_r_cpu_cac_data3 = 32'h04404404;
	tb_r_cpu_cac_addr3 = 32'h00567000;
	
	tb_r_cpu_cac_read3 = 0;
	tb_r_cpu_cac_write3 = 1;

Test 4: Read Hit in Cache 3 and continuous Read Hit	First read instruction,
	tb_r_cpu_cac_data3 = 32'h0;
	tb_r_cpu_cac_addr3 = 32'h00567004;

	tb_r_cpu_cac_read3 = 1;
	tb_r_cpu_cac_write3 = 0;
	
	Second read instruction,
	tb_r_cpu_cac_data3 = 32'h0;
	tb_r_cpu_cac_addr3 = 32'h00567000;
	
	tb_r_cpu_cac_read3 = 1;
	tb_r_cpu_cac_write3 = 0;

Test 5: Write Miss with FIFO miss in Cache 3	First read a data from SDRAM by trying write miss in @89A00 (where valid = 0),
	tb_r_cpu_cac_data3 = 32'h00B00177;
	tb_r_cpu_cac_addr3 = 32'h0089A000;
	
	tb_r_cpu_cac_read3 = 0;
	tb_r_cpu_cac_write3 = 1;
	
	Then try to write a data with same index but different tag with @56700, (tag different),
	tb_r_cpu_cac_data3 = 32'h06070809;
	tb_r_cpu_cac_addr3 = 32'h00167000;
	
	tb_r_cpu_cac_read3 = 0;
	tb_r_cpu_cac_write3 = 1;
	
	with FIFO miss,@56700 data evict to FIFO

Test 6: Write Miss with FIFO hit in Cache 3	FIFO hit,@56700 data write back from FIFO
	tb_r_cpu_cac_data3 = 32'hF1FA0000;
	tb_r_cpu_cac_addr3 = 32'h00567000;
	
	tb_r_cpu_cac_read3 = 0;
	tb_r_cpu_cac_write3 = 1;
	
	@16700 move to FIFO

Test 7: Auto Write Back to SDRAM in Cache 3 with FIFO busy	Give an instruction that give hit cache for 6 clock cycle
	tb_r_cpu_cac_data3 = 32'h0;
	tb_r_cpu_cac_addr3 = 32'h00567004;
	
	tb_r_cpu_cac_read3 = 1;
	tb_r_cpu_cac_write3 = 0;
	
	@16700 move from FIFO to SDRAM

	Then give miss cache instruction, cache controller wait for FIFO finish writing

Test 8: Read Miss with FIFO miss in Cache 3	tb_r_cpu_cac_data3 = 32'h0;
	tb_r_cpu_cac_addr3 = 32'h00E9A000;
	
	tb_r_cpu_cac_read3 = 1;
	tb_r_cpu_cac_write3 = 0;
	
	Data read back from SDRAM,@89A00 move to FIFO (same index different tag)

Test 9: Read Miss with FIFO hit in Cache 3	tb_r_cpu_cac_data3 = 32'h0;
	tb_r_cpu_cac_addr3 = 32'h0089A000;
	
	tb_r_cpu_cac_read3 = 1;
	tb_r_cpu_cac_write3 = 0;
	
	Since previous instruction is read only so dirty is 0. @E9A00 did not move to FIFO

Test 10: Miss happen and FIFO full	//FIFO status: *,*,*,*
	Try a write miss instruction where valid = 0,
	tb_r_cpu_cac_data3 = 32'h26100AAA;
	tb_r_cpu_cac_addr3 = 32'h00261000;
	
	tb_r_cpu_cac_read3 = 0;
	tb_r_cpu_cac_write3 = 1;
	
	Write miss and @26100 move to FIFO,
	tb_r_cpu_cac_data3 = 32'h46100BBB;
	tb_r_cpu_cac_addr3 = 32'h00461000;
	
	tb_r_cpu_cac_read3 = 0;
	tb_r_cpu_cac_write3 = 1;
	//FIFO after this: 26100,*,*,*
	
	Write miss and @46100 move to FIFO,
	tb_r_cpu_cac_data3 = 32'h66100CCC;
	tb_r_cpu_cac_addr3 = 32'h00661000;
	
	tb_r_cpu_cac_read3 = 0;
	tb_r_cpu_cac_write3 = 1;
	//FIFO after this: 26100,46100,*,*
	
	Write miss and @66100 move to FIFO,
	tb_r_cpu_cac_data3 = 32'h86100DDD;
	tb_r_cpu_cac_addr3 = 32'h00861000;

	tb_r_cpu_cac_read3 = 0;
	tb_r_cpu_cac_write3 = 1;
	//FIFO after this: 26100,46100,66100,*
	
	Write miss and @86100 move to FIFO,
	tb_r_cpu_cac_data3 = 32'hA6100EEE;
	tb_r_cpu_cac_addr3 = 32'h00A61000;
	
	tb_r_cpu_cac_read3 = 0;
	tb_r_cpu_cac_write3 = 1;
	//FIFO after this: 26100,46100,66100,86100
	
	Write miss and FIFO is full, @ 26100 write back to SDRAM, after that @A6100 move to FIFO, and cache resumes write operation
	tb_r_cpu_cac_data3 = 32'hC6100FFF;
	tb_r_cpu_cac_addr3 = 32'h00C61000;
	
	tb_r_cpu_cac_read3 = 0;
	tb_r_cpu_cac_write3 = 1;
	//FIFO after this: A6100,46100,66100,86100

Table 7-1-1: Memory system Full Chip Test Plan

7.2 Testbench Verilog Code

```
`include "./../util/sdc_macro.v"
`timescale 1ns / 10ps
module tb_cac_ma_sdc();
//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_data_ready3,
                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_lmc_data3;
wire            w_ma_cac_complete3;
```

```
reg     [31:0]  r_ma_cac_i_data3;
wire            w_cac_mem_ack3;
wire            w_cac_mem_lmc_same3;
//cache2
wire            w_ma_cac_read2,
                w_ma_cac_write2,
                w_data_ready2,
                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_lmc_data2;
wire            w_ma_cac_complete2;
reg     [31:0]  r_ma_cac_i_data2;
wire            w_cac_mem_ack2;
wire            w_cac_mem_lmc_same2;
//cache1
wire            w_ma_cac_read1,
                w_ma_cac_write1,
                w_data_ready1,
                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_lmc_data1;
wire            w_ma_cac_complete1;
reg     [31:0]  r_ma_cac_i_data1;
wire            w_cac_mem_ack1;
wire            w_cac_mem_lmc_same1;
//cache0
wire           w_ma_cac_read0,
               w_ma_cac_write0,
               w_data_ready0,
               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_lmc_data0;
wire            w_ma_cac_complete0;
reg     [31:0] r_ma_cac_i_data0;
wire            w_cac_mem_ack0;
wire            w_cac_mem_lmc_same0;

//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;

//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 [300:0]     status;

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

u_cache cache_3
 (//memory arbiter connection
  .uo_cac_mem_addr(w_ma_cac_addr3),
  .uo_cac_mem_data(w_i_data3),
  .uo_cac_mem_lmc_data(),
  .uo_cac_miss(w_ma_cac_miss3),
  .uo_cac_mem_read(w_ma_cac_read3),
  .uo_cac_mem_write(w_ma_cac_write3),
  .uo_cac_mem_data_ready(w_data_ready3),
  .uo_cac_mem_sel(w_ma_cac_sel3),
  .uo_cac_mem_complete(w_ma_cac_complete3),
  .ui_cac_mem_data(w_ma_cac_o_data3),
  .ui_cac_mem_ack(w_cac_mem_ack3),
  .ui_cac_mem_lmc_same(w_cac_mem_lmc_same3),
  // CPU connection
  .uo_cac_cpu_stall(),
```

```
 .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_mem_lmc_data(),
 .uo_cac_miss(w_ma_cac_miss2),
 .uo_cac_mem_read(w_ma_cac_read2),
 .uo_cac_mem_write(w_ma_cac_write2),
 .uo_cac_mem_data_ready(w_data_ready2),
 .uo_cac_mem_sel(w_ma_cac_sel2),
 .uo_cac_mem_complete(w_ma_cac_complete2),
 .ui_cac_mem_data(w_ma_cac_o_data2),
 .ui_cac_mem_ack(w_cac_mem_ack2),
 .ui_cac_mem_lmc_same(w_cac_mem_lmc_same2),
 // CPU connection
 .uo_cac_cpu_stall(),
 .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_mem_lmc_data(),
  .uo_cac_miss(w_ma_cac_miss1),
  .uo_cac_mem_read(w_ma_cac_read1),
  .uo_cac_mem_write(w_ma_cac_write1),
  .uo_cac_mem_data_ready(w_data_ready1),
  .uo_cac_mem_sel(w_ma_cac_sel1),
  .uo_cac_mem_complete(w_ma_cac_complete1),
  .ui_cac_mem_data(w_ma_cac_o_data1),
  .ui_cac_mem_ack(w_cac_mem_ack1),
  .ui_cac_mem_lmc_same(w_cac_mem_lmc_same1),
```

```
 // CPU connection
 .uo_cac_cpu_stall(),
 .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_mem_lmc_data(),
 .uo_cac_miss(w_ma_cac_miss0),
 .uo_cac_mem_read(w_ma_cac_read0),
 .uo_cac_mem_write(w_ma_cac_write0),
 .uo_cac_mem_data_ready(w_data_ready0),
 .uo_cac_mem_sel(w_ma_cac_sel0),
 .uo_cac_mem_complete(w_ma_cac_complete0),
 .ui_cac_mem_data(w_ma_cac_o_data0),
 .ui_cac_mem_ack(w_cac_mem_ack0),
 .ui_cac_mem_lmc_same(w_cac_mem_lmc_same0),
 // CPU connection
 .uo_cac_cpu_stall(),
 .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_miss3(w_ma_cac_miss3),
 .ui_ma_cac_data_ready3(w_data_ready3),
 .ui_ma_cac_read3(w_ma_cac_read3),
 .ui_ma_cac_write3(w_ma_cac_write3),
 .ui_ma_cac_sel3(w_ma_cac_sel3),
 .ui_ma_cac_addr3(w_ma_cac_addr3),
 .ui_ma_cac_data3(w_i_data3),
 .ui_ma_cac_lmc_data3(r_ma_cac_lmc_data3),
 .ui_ma_cac_complete3(w_ma_cac_complete3),
```

```
.uo_ma_cac_ack3(w_cac_mem_ack3),
.uo_ma_cac_lmc_same3(w_cac_mem_lmc_same3),
.uo_ma_cac_data3(w_ma_cac_o_data3),
//cache2
.ui_ma_cac_miss2(w_ma_cac_miss2),
.ui_ma_cac_data_ready2(w_data_ready2),
.ui_ma_cac_read2(w_ma_cac_read2),
.ui_ma_cac_write2(w_ma_cac_write2),
.ui_ma_cac_sel2(w_ma_cac_sel2),
.ui_ma_cac_addr2(w_ma_cac_addr2),
.ui_ma_cac_data2(w_i_data2),
.ui_ma_cac_lmc_data2(r_ma_cac_lmc_data2),
.ui_ma_cac_complete2(w_ma_cac_complete2),
.uo_ma_cac_ack2(w_cac_mem_ack2),
.uo_ma_cac_lmc_same2(w_cac_mem_lmc_same2),
.uo_ma_cac_data2(w_ma_cac_o_data2),
//cache1
.ui_ma_cac_miss1(w_ma_cac_miss1),
.ui_ma_cac_data_ready1(w_data_ready1),
.ui_ma_cac_read1(w_ma_cac_read1),
.ui_ma_cac_write1(w_ma_cac_write1),
.ui_ma_cac_sel1(w_ma_cac_sel1),
.ui_ma_cac_addr1(w_ma_cac_addr1),
.ui_ma_cac_data1(w_i_data1),
.ui_ma_cac_lmc_data1(r_ma_cac_lmc_data1),
.ui_ma_cac_complete1(w_ma_cac_complete1),
.uo_ma_cac_ack1(w_cac_mem_ack1),
.uo_ma_cac_lmc_same1(w_cac_mem_lmc_same1),
.uo_ma_cac_data1(w_ma_cac_o_data1),
//cache0
.ui_ma_cac_miss0(w_ma_cac_miss0),
.ui_ma_cac_data_ready0(w_data_ready0),
.ui_ma_cac_read0(w_ma_cac_read0),
.ui_ma_cac_write0(w_ma_cac_write0),
.ui_ma_cac_sel0(w_ma_cac_sel0),
.ui_ma_cac_addr0(w_ma_cac_addr0),
.ui_ma_cac_data0(w_i_data0),
.ui_ma_cac_lmc_data0(r_ma_cac_lmc_data0),
.ui_ma_cac_complete0(w_ma_cac_complete0),
.uo_ma_cac_ack0(w_cac_mem_ack0),
.uo_ma_cac_lmc_same0(w_cac_mem_lmc_same0),
.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),
 .ui_ma_clk(tb_r_clk),
 .ui_ma_rst(tb_r_rst));

u_sdram_controller u_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));

//initialize clock signal
initial tb_r_clk = 1;
```

```
always #10 tb_r_clk = ~tb_r_clk;

always@* begin
  r_ma_cac_lmc_data3 = {29'h4,tb_r_BL_sel[3]};
  r_ma_cac_lmc_data2 = {29'h4,tb_r_BL_sel[2]};
  r_ma_cac_lmc_data1 = {29'h4,tb_r_BL_sel[1]};
  r_ma_cac_lmc_data0 = {29'h4,tb_r_BL_sel[0]};
end

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);

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 //Test 1: 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("rtl/micron SDRAM/sdram_bank0_data.txt", sdram.Bank0) ;

  status = "Read data (Cache3->Cache2->Cache1->Cache0)";

        //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;

 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Test 2: Testing Cache priority and reading in different burst length
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // All 4 cache read misses in same clock cycle
    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 cache misses
    // Wait until they are done
    while(w_ma_cac_miss3||w_ma_cac_miss2||w_ma_cac_miss1||w_ma_cac_miss0
    ||w_data_ready3||w_data_ready2||w_data_ready1||w_data_ready0)
    @(posedge tb_r_clk);
```

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//Test 3: Write Hit in Cache 3
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   status = "Write Hit";
   tb_r_cpu_cac_data3  = 32'h07070707;
   tb_r_cpu_cac_addr3  = 32'h00567004;

   tb_r_cpu_cac_read3  = 0;
   tb_r_cpu_cac_write3 = 1;

   @(posedge tb_r_clk);
   status = "Write Hit";
   tb_r_cpu_cac_data3  = 32'h04404404;
   tb_r_cpu_cac_addr3  = 32'h00567000;

   tb_r_cpu_cac_read3  = 0;
   tb_r_cpu_cac_write3 = 1;

   /*@56700
     0440_4404
     0707_0707
     24A6_0004
     0004_1080
     00C2_3021
     0020_0900
     0100_0750
     3402_000A*/

   @(posedge tb_r_clk);

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//Test 4: Read Hit in Cache 3
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   status = "Read Hit";
   tb_r_cpu_cac_data3  = 32'h0;
   tb_r_cpu_cac_addr3  = 32'h00567004;

   tb_r_cpu_cac_read3  = 1;
   tb_r_cpu_cac_write3 = 0;

   @(posedge tb_r_clk);
   status = "Read Hit";
   tb_r_cpu_cac_data3  = 32'h0;
   tb_r_cpu_cac_addr3  = 32'h00567000;
```

```
    tb_r_cpu_cac_read3  = 1;
    tb_r_cpu_cac_write3 = 0;

  //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//Test 5: Write Miss with FIFO miss in Cache 3
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    @(posedge tb_r_clk);
    status = "Write Miss"; // with dirty = 0; after write dirty = 1;

    tb_r_cpu_cac_data3  = 32'h00B00177;
    tb_r_cpu_cac_addr3  = 32'h0089A000;

    tb_r_cpu_cac_read3  = 0;
    tb_r_cpu_cac_write3 = 1;

    /*@89A00
      00B0_0177
      1234_ABCD
      5678_7654
      3456_789A
      9876_3210
      FAFA_FAFA
      BEEF_BEEF
      DEAD_DEAD*/

    @(posedge tb_r_clk);
    while(w_ma_cac_miss3)@(posedge tb_r_clk);

    status = "Check";
    tb_r_cpu_cac_data3  = 32'h0;
    tb_r_cpu_cac_addr3  = 32'h0089A000;

    tb_r_cpu_cac_read3  = 1;
    tb_r_cpu_cac_write3 = 0;

    @(posedge tb_r_clk);
    status = "Write Miss"; // dirty =1; with FIFO miss,@56700 data erect to FIFO
    tb_r_cpu_cac_data3  = 32'h06070809;
    tb_r_cpu_cac_addr3  = 32'h00167000; //same index different tag (@56700)

    tb_r_cpu_cac_read3  = 0;
    tb_r_cpu_cac_write3 = 1;

    /*@16700
      0607_0809
      5201_314B
```

```
  5201_314C
  5201_314D
  5201_314E
  5201_314F
  5201_3140
  5201_315A
  5201_315B */

  @(posedge tb_r_clk);
  while(w_ma_cac_miss3)@(posedge tb_r_clk);

  status = "Check";
  tb_r_cpu_cac_data3  = 32'h0;
  tb_r_cpu_cac_addr3  = 32'h00167000;

  tb_r_cpu_cac_read3  = 1;
  tb_r_cpu_cac_write3 = 0;
  @(posedge tb_r_clk);


//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//Test 6: Write Miss with FIFO hit in Cache 3
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  status = "Write Miss"; //with FIFO hit,@56700 data write back from FIFO
  tb_r_cpu_cac_data3  = 32'hF1FA0000;
  tb_r_cpu_cac_addr3  = 32'h00567000; // @16700 move to FIFO

  tb_r_cpu_cac_read3  = 0;
  tb_r_cpu_cac_write3 = 1;

  /*@56700
    F1FA_0000
    0707_0707
    24A6_0004
    0004_1080
    00C2_3021
    0020_0900
    0100_0750
    3402_000A*/

  @(posedge tb_r_clk);
  while(w_ma_cac_miss3)@(posedge tb_r_clk);

  status = "Check";
  tb_r_cpu_cac_data3  = 32'h0;
  tb_r_cpu_cac_addr3  = 32'h00567000;
```

```
    tb_r_cpu_cac_read3  = 1;
    tb_r_cpu_cac_write3 = 0;


//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//Test 7: Auto Write Back to SDRAM in Cache 3 with FIFO busy
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    @(posedge tb_r_clk);
    status = "FIFO WB to SDRAM"; //(@16700 move from FIFO to SDRAM)
    tb_r_cpu_cac_data3  = 32'h0;
    tb_r_cpu_cac_addr3  = 32'h00567004;

    tb_r_cpu_cac_read3  = 1;
    tb_r_cpu_cac_write3 = 0;

    @(posedge tb_r_clk);
    tb_r_cpu_cac_data3  = 32'h0;
    tb_r_cpu_cac_addr3  = 32'h00567008;

    tb_r_cpu_cac_read3  = 1;
    tb_r_cpu_cac_write3 = 0;

    @(posedge tb_r_clk);
    tb_r_cpu_cac_data3  = 32'h0;
    tb_r_cpu_cac_addr3  = 32'h0056700C;

    tb_r_cpu_cac_read3  = 1;
    tb_r_cpu_cac_write3 = 0;

    @(posedge tb_r_clk);
    tb_r_cpu_cac_data3  = 32'h0;
    tb_r_cpu_cac_addr3  = 32'h00567010;

    tb_r_cpu_cac_read3  = 1;
    tb_r_cpu_cac_write3 = 0;

    @(posedge tb_r_clk);
    tb_r_cpu_cac_data3  = 32'h0;
    tb_r_cpu_cac_addr3  = 32'h00567014;

    tb_r_cpu_cac_read3  = 1;
    tb_r_cpu_cac_write3 = 0;

    @(posedge tb_r_clk);
    tb_r_cpu_cac_data3  = 32'h0;
```

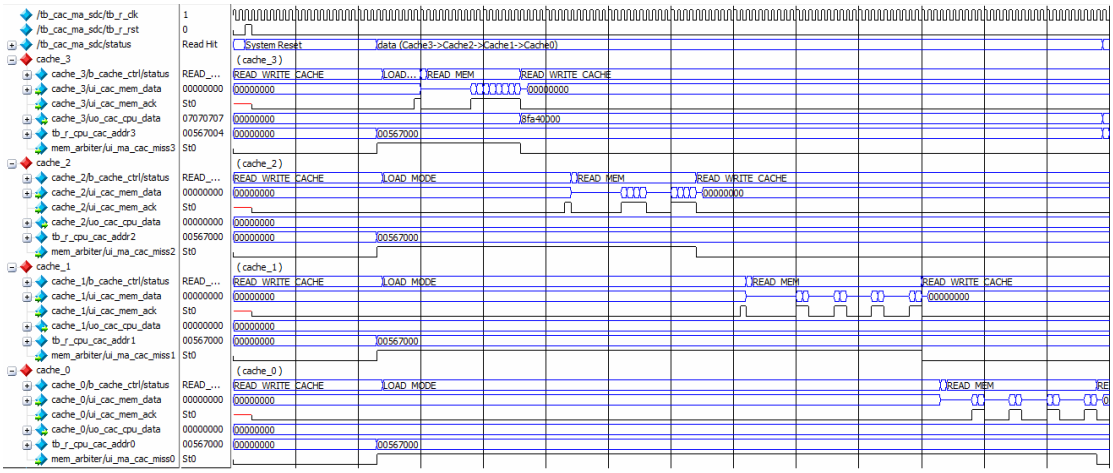
```
   tb_r_cpu_cac_addr3  = 32'h00567018;

   tb_r_cpu_cac_read3  = 1;
   tb_r_cpu_cac_write3 = 0;

   @(posedge tb_r_clk);
   tb_r_cpu_cac_data3  = 32'h0;
   tb_r_cpu_cac_addr3  = 32'h0056701C;

   tb_r_cpu_cac_read3  = 1;
   tb_r_cpu_cac_write3 = 0;

   @(posedge tb_r_clk);

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//Test 8: Read Miss with FIFO miss in Cache 3
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   status = "Read Miss"; //data read back from SDRAM,@89A00 move to FIFO
   tb_r_cpu_cac_data3  = 32'h0;
   tb_r_cpu_cac_addr3  = 32'h00E9A000;

   tb_r_cpu_cac_read3  = 1;
   tb_r_cpu_cac_write3 = 0;

   /*@89A00      @E9A00
     00B0_0177    39A8_776F
     1234_ABCD    5555_5555
     5678_7654    7777_7777
     3456_789A    FFFF_FFFF
     9876_3210    1212_3434
     FAFA_FAFA    0000_0001
     BEEF_BEEF    BAD0_ADD8
     DEAD_DEAD    2345_5432*/

   @(posedge tb_r_clk);
   while(w_ma_cac_miss3)@(posedge tb_r_clk);


//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//Test 9: Read Miss with FIFO hit in Cache 3
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   status = "Read Miss"; //@89A00 move from FIFO to cache
   tb_r_cpu_cac_data3  = 32'h0;
   tb_r_cpu_cac_addr3  = 32'h0089A000;

   tb_r_cpu_cac_read3  = 1;
```

```
    tb_r_cpu_cac_write3 = 0;

    @(posedge tb_r_clk);
    while(w_ma_cac_miss3)@(posedge tb_r_clk);


//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//Test 10: Miss happen and FIFO full
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    status = "FIFO Full,Write Miss1"; //FIFO status: *,*,*,*
    tb_r_cpu_cac_data3  = 32'h26100AAA;
    tb_r_cpu_cac_addr3  = 32'h00261000;

    tb_r_cpu_cac_read3  = 0;
    tb_r_cpu_cac_write3 = 1;

    @(posedge tb_r_clk);
    while(w_ma_cac_miss3)@(posedge tb_r_clk);
    status = "Write  Miss2"; //FIFO after this: 26100,*,*,*
    tb_r_cpu_cac_data3  = 32'h46100BBB;
    tb_r_cpu_cac_addr3  = 32'h00461000;

    tb_r_cpu_cac_read3  = 0;
    tb_r_cpu_cac_write3 = 1;

    @(posedge tb_r_clk);
    while(w_ma_cac_miss3)@(posedge tb_r_clk);
    status = "Write  Miss3"; //FIFO after this: 26100,46100,*,*
    tb_r_cpu_cac_data3  = 32'h66100CCC;
    tb_r_cpu_cac_addr3  = 32'h00661000;

    tb_r_cpu_cac_read3  = 0;
    tb_r_cpu_cac_write3 = 1;

    @(posedge tb_r_clk);
    while(w_ma_cac_miss3)@(posedge tb_r_clk);
    status = "Write Miss4"; //FIFO after this: 26100,46100,66100,*
    tb_r_cpu_cac_data3  = 32'h86100DDD;
    tb_r_cpu_cac_addr3  = 32'h00861000;

    tb_r_cpu_cac_read3  = 0;
    tb_r_cpu_cac_write3 = 1;

    @(posedge tb_r_clk);
    while(w_ma_cac_miss3)@(posedge tb_r_clk);
    status = "Write Miss5"; //FIFO after this: 26100,46100,66100,86100
```

```
    tb_r_cpu_cac_data3  = 32'hA6100EEE;
    tb_r_cpu_cac_addr3  = 32'h00A61000;

    tb_r_cpu_cac_read3  = 0;
    tb_r_cpu_cac_write3 = 1;

    @(posedge tb_r_clk);
    while(w_ma_cac_miss3)@(posedge tb_r_clk); //@26100 wb to SDRAM
    status = "Write Miss6"; //FIFO after this: A6100,46100,66100,86100
    tb_r_cpu_cac_data3  = 32'hC6100FFF;
    tb_r_cpu_cac_addr3  = 32'h00C61000;

    tb_r_cpu_cac_read3  = 0;
    tb_r_cpu_cac_write3 = 1;

    @(posedge tb_r_clk);
    while(w_ma_cac_miss3)@(posedge tb_r_clk);
    repeat(5) @(posedge tb_r_clk);
    $stop;
end

endmodule
```

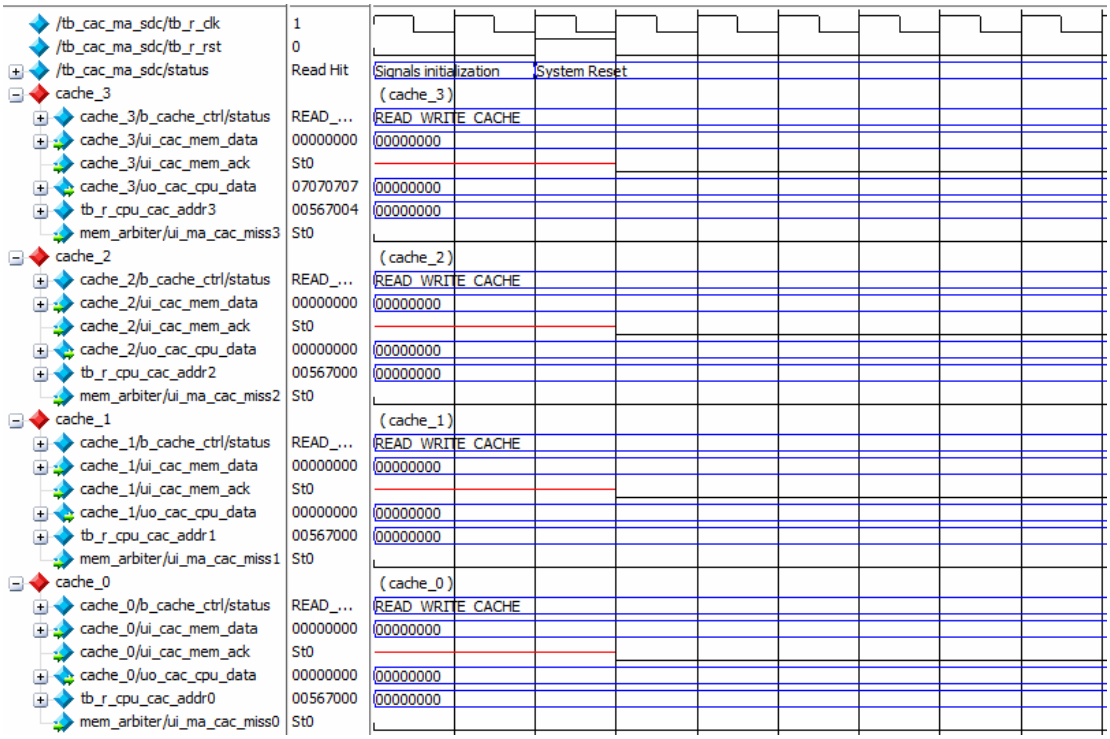
7.3 Simulation Result

Test 1 and Test 2 overall Timing Diagram



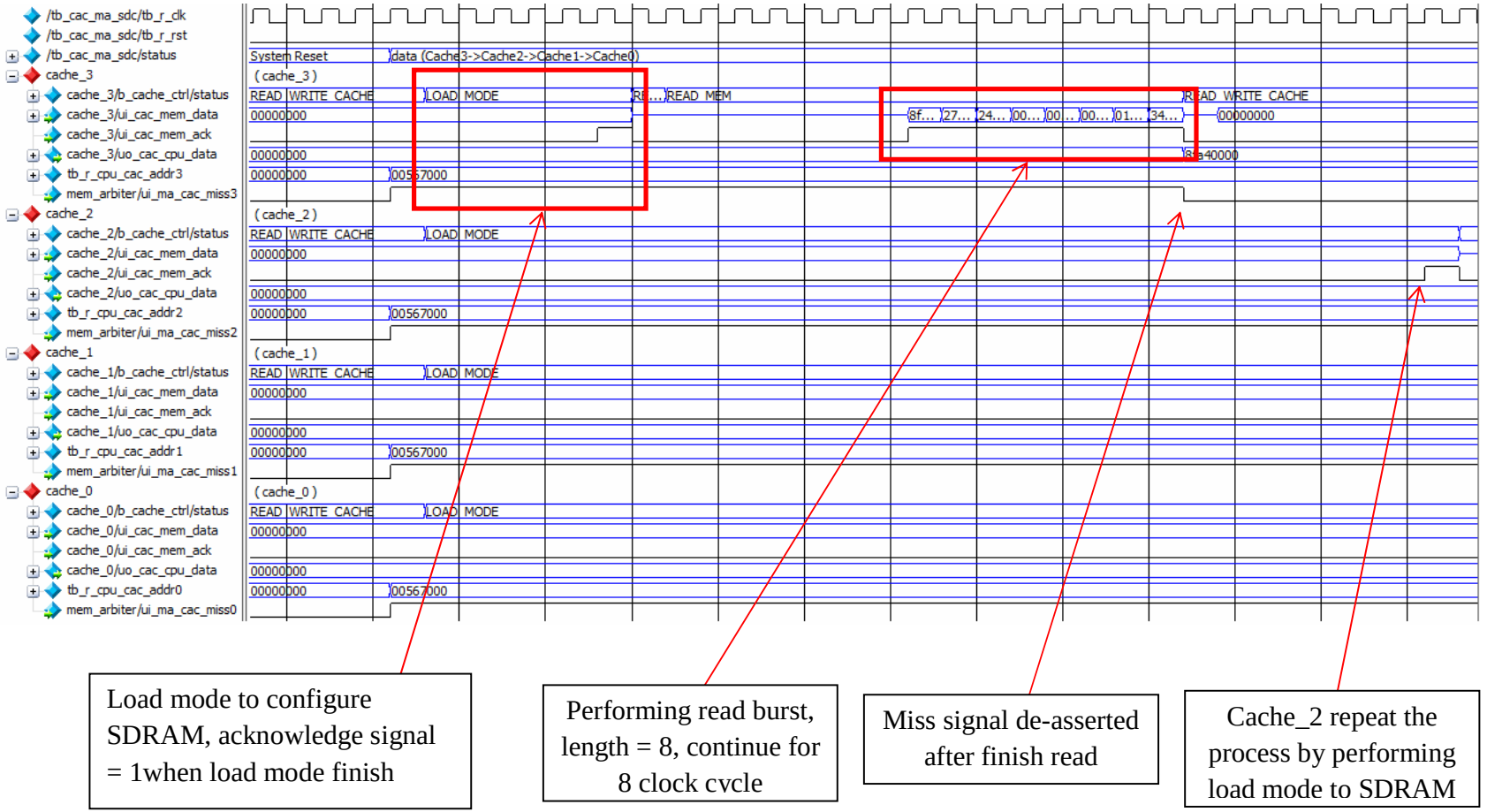
Test 1: System Reset

Signal Initialization and System Reset

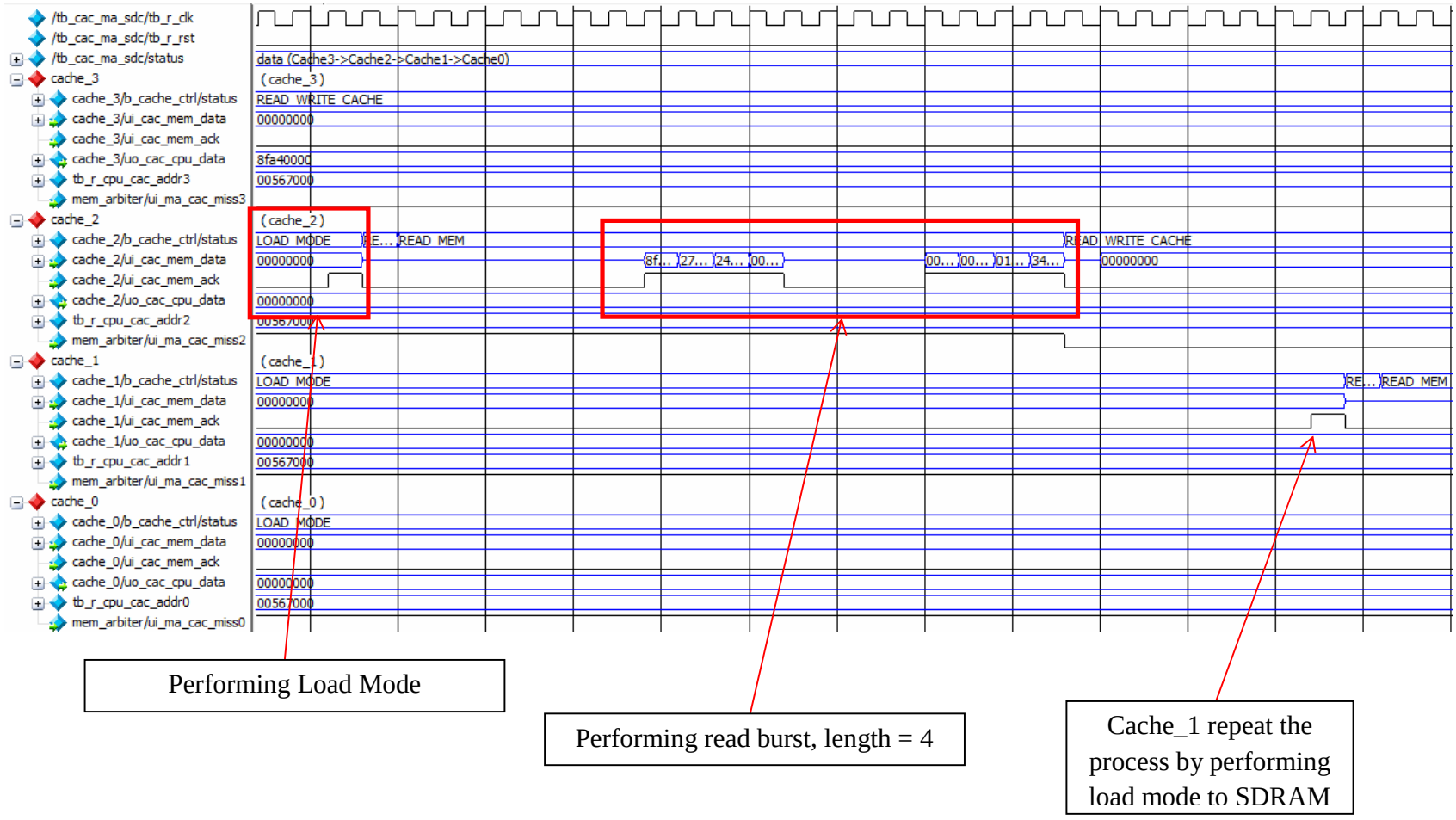


Test 2: Testing Cache priority and reading in different burst length

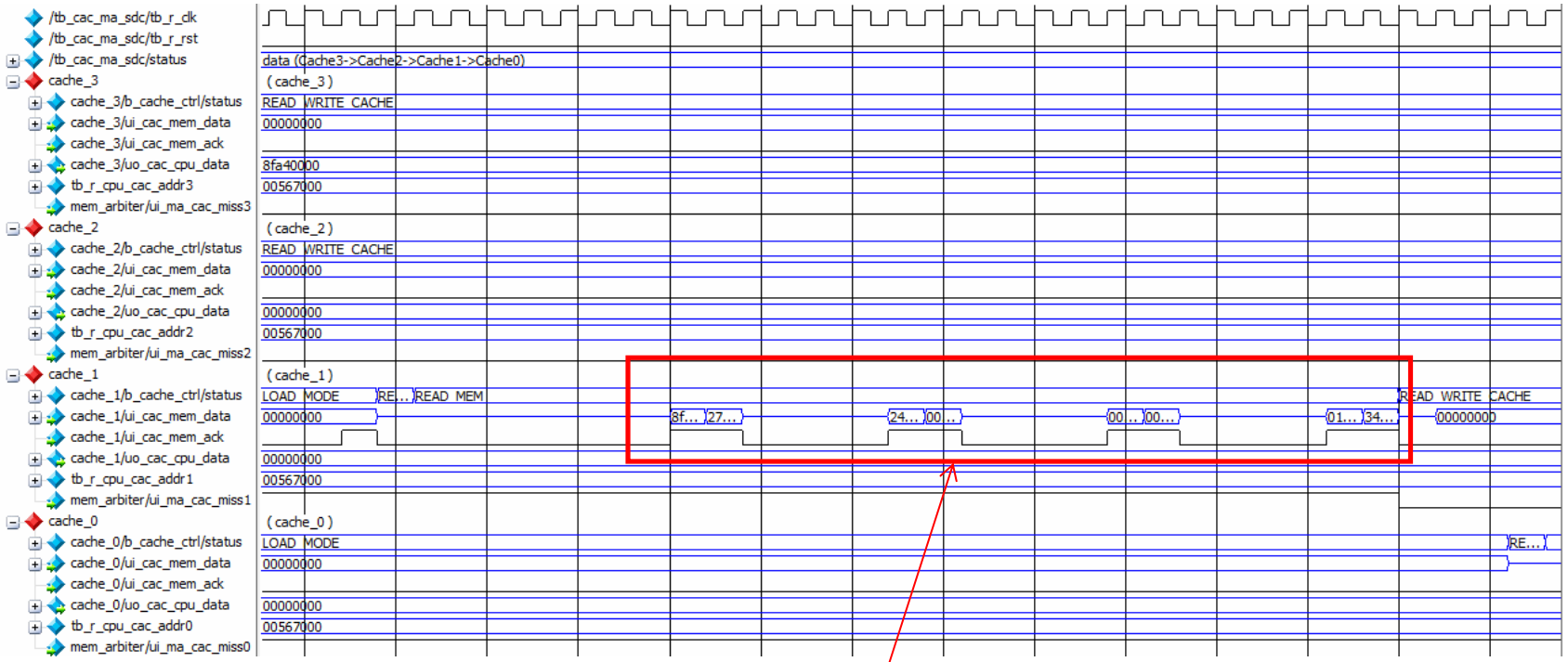
Priority is given to cache_3 to run first according to the priority arrangement in Memory Arbiter. Here SDRAM configuration is burst length = 8.



Then, priority is given to cache_2 to run. SDRAM configuration is burst length = 4.

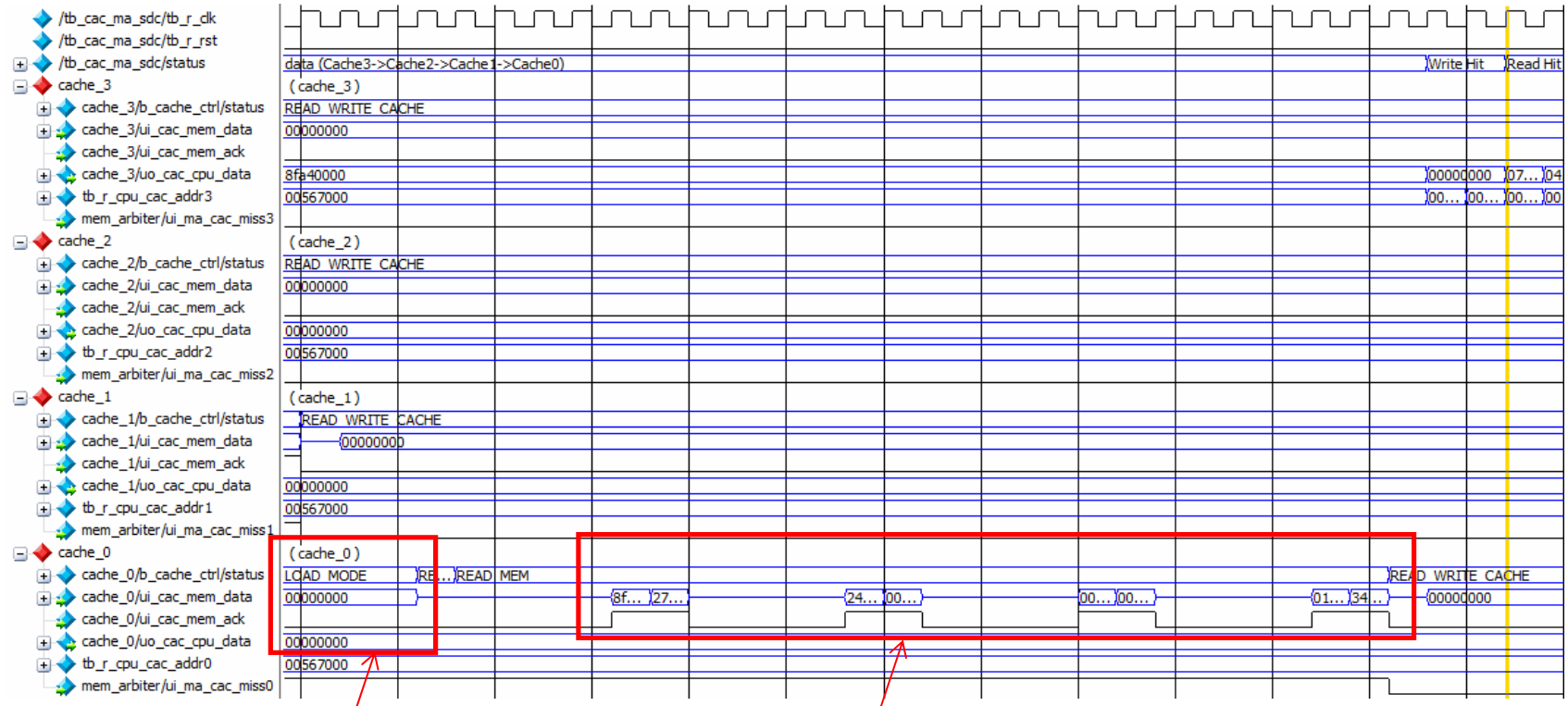


Then, priority is given to cache_1 to run. SDRAM configuration is burst length = 2.



Performing read burst, length = 2

Then, priority is given to cache_0 to run. SDRAM configuration is burst length = 2. The configuration same as previous thus SDRAM no need to load mode again.

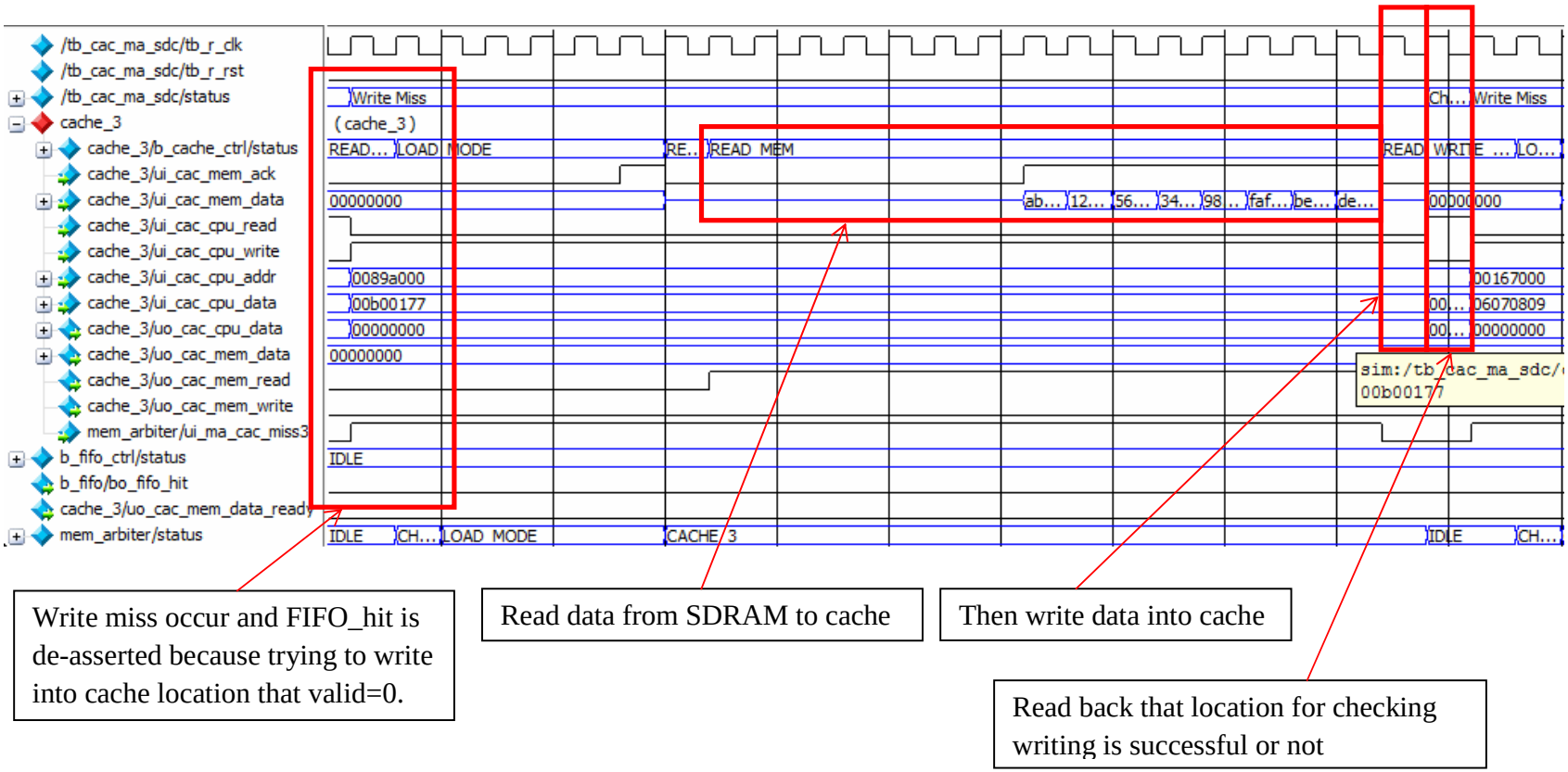


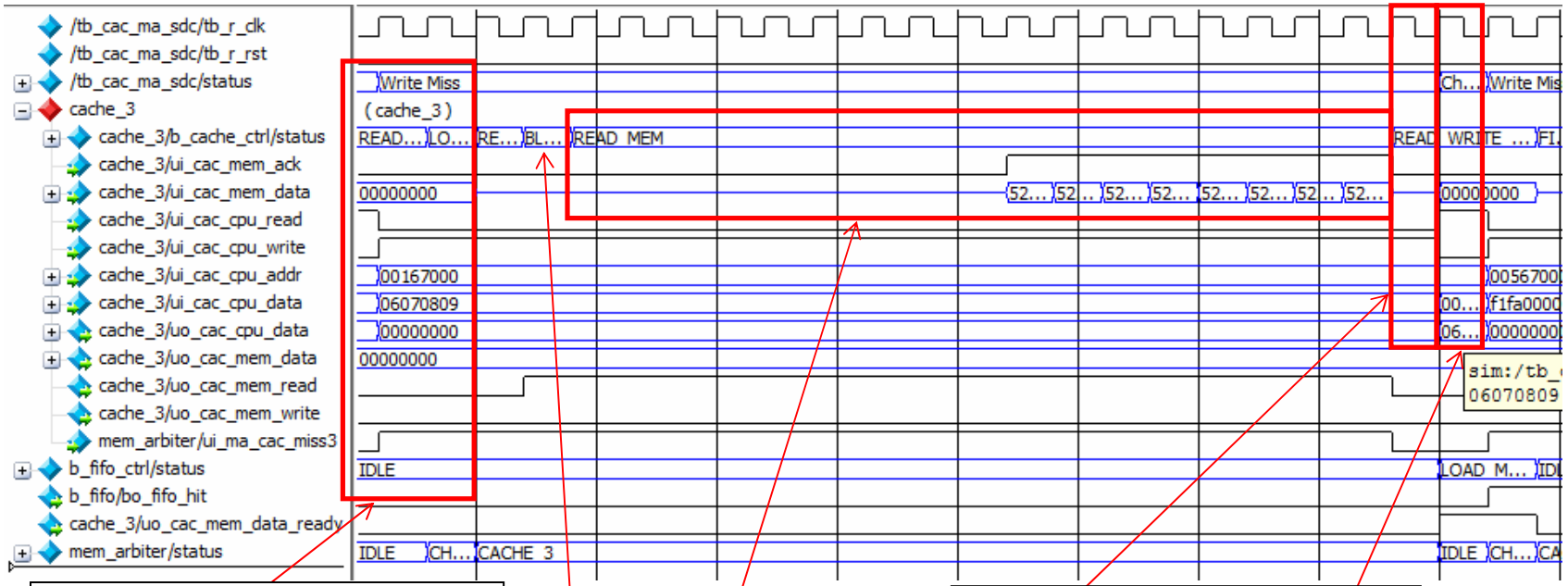
Test 3: Write Hit in Cache 3 and continuous Write Hit and

Test 4: Read Hit in Cache 3 and continuous Read Hit

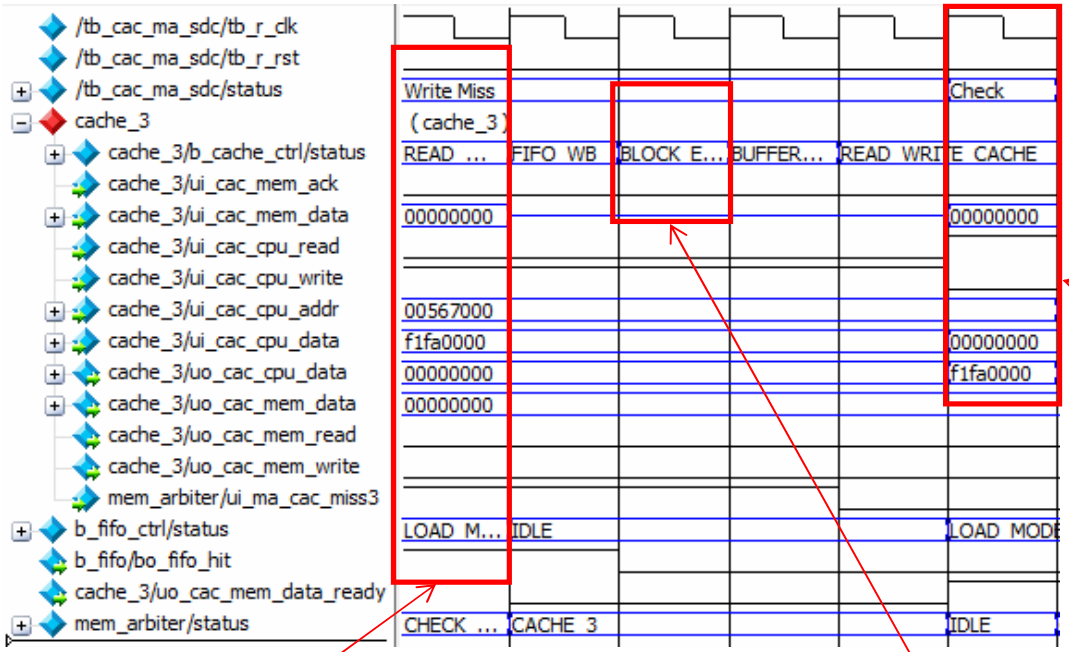
Data had been written into cache in previous test. Thus write hit occur here with same tag and index. Data were written into cache continuously (data become dirty because not updated to SDRAM) and then for next two clock cycle data were read out to uo_cac_cpu_data.

Test 5: Write Miss with FIFO misses in Cache 3




/tb_cac_ma_sdc/tb_r_clk
/tb_cac_ma_sdc/tb_r_rst
/tb_cac_ma_sdc/status
cache_3
cache_3/b_cache_ctrl/status
cache_3/ui_cac_mem_ack
cache_3/ui_cac_mem_data
cache_3/ui_cac_cpu_read
cache_3/ui_cac_cpu_write
cache_3/ui_cac_cpu_addr
cache_3/ui_cac_cpu_data
cache_3/uo_cac_cpu_data
cache_3/uo_cac_mem_data
cache_3/uo_cac_mem_read
cache_3/uo_cac_mem_write
mem_arbiter/ui_ma_cac_miss3
b_fifo_ctrl/status
b_fifo/bo_fifo_hit
cache_3/uo_cac_mem_data_ready
mem_arbiter/status
Write Miss
(cache_3)
READ...
LO...
RE...
BL...
READ_MEM
00000000
00167000
06070809
00000000
00000000
IDLE
IDLE
CH...
CACHE_3
Ch...
Write Mis
READ_WRITE ...
FI...
00000000
00056700
f1fa0000
00000000
sim:/tb_
06070809
LOAD_M...
IDLE
CH...
Write miss occur and FIFO_hit is de-asserted because trying to write into cache location that tag is different.
Read data from SDRAM to cache.
Then write data into cache. Dirty = 1
Since cache location that need to be written is dirty the block is evict and copy to FIFO.
Read back that location for checking writing is successful or not.
FIFO entries: @56700, *, *, *

Test 6: Write Miss with FIFO hit in Cache 3



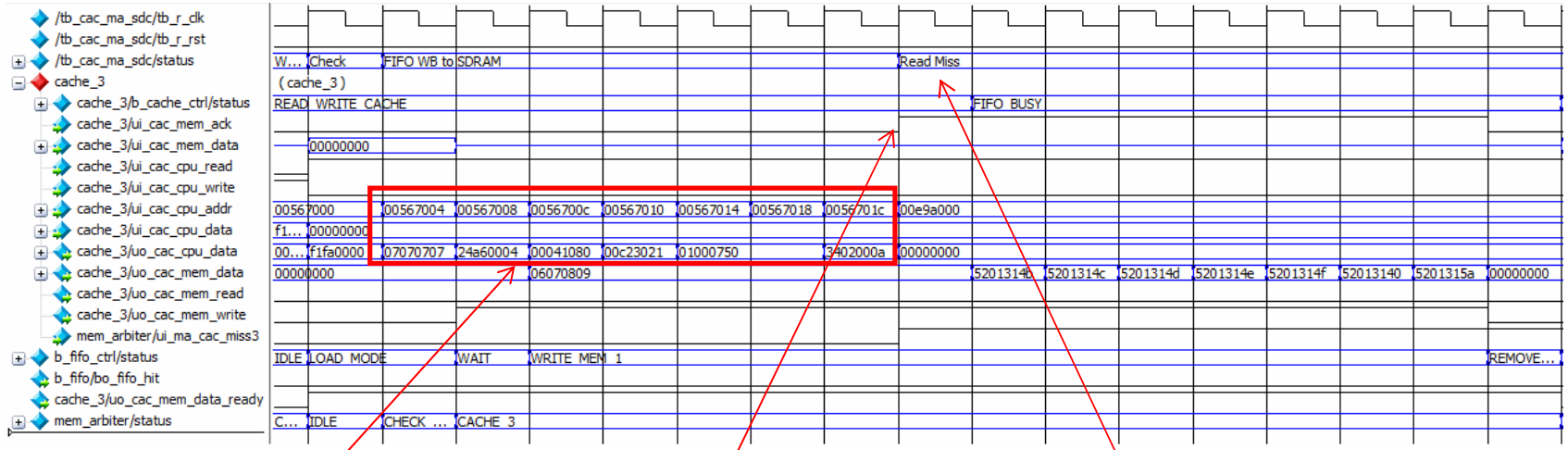
Read back that cache location for checking writing is successful or not.

Write miss occur and FIFO_hit is asserted because trying to write into cache location that tag is different.

Since it is FIFO hit, data written back from FIFO to cache. The location in cache to be written is dirty thus move that data block from cache to FIFO.

FIFO entries: @16700, *, *, *

Test 7: Auto Write Back to SDRAM in Cache 3 with FIFO busy

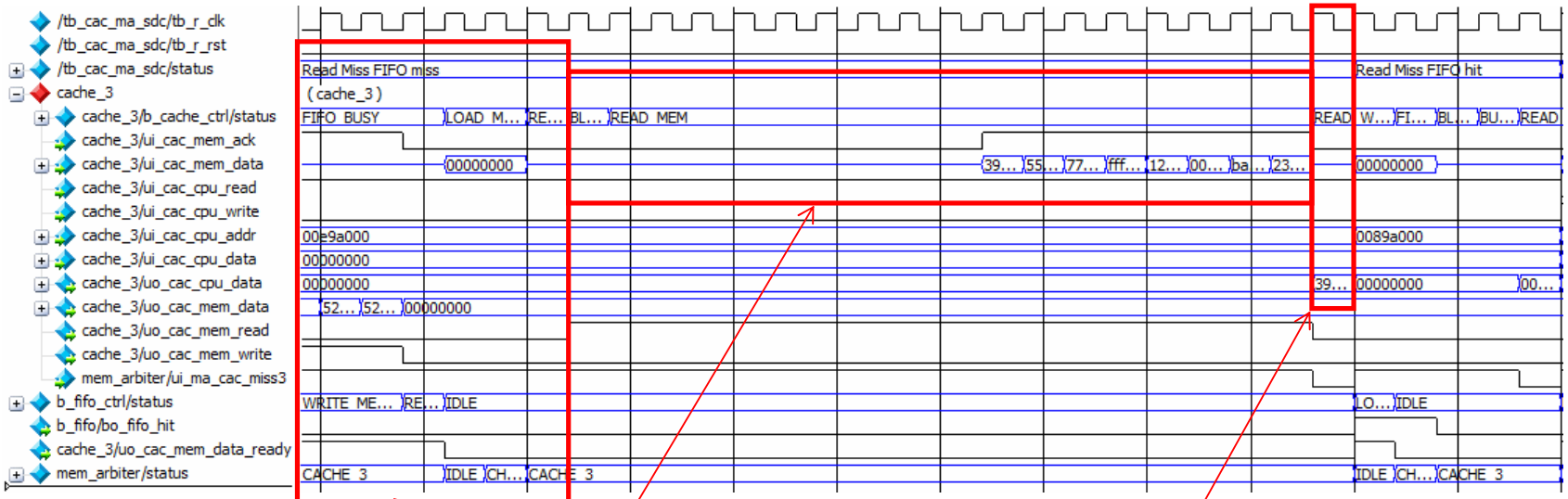


A series of read hit operation is given and now SDRAM is free. FIFO is written back to SDRAM while read operation is in progress.

FIFO wait for SDRAM to prepare receive data, then when ack signal is asserted data were written back to SDRAM.

During writing into SDRAM, read miss occur, thus pipeline was stalled until data block is finish written into

Test 8: Read Miss with FIFO misses in Cache 3



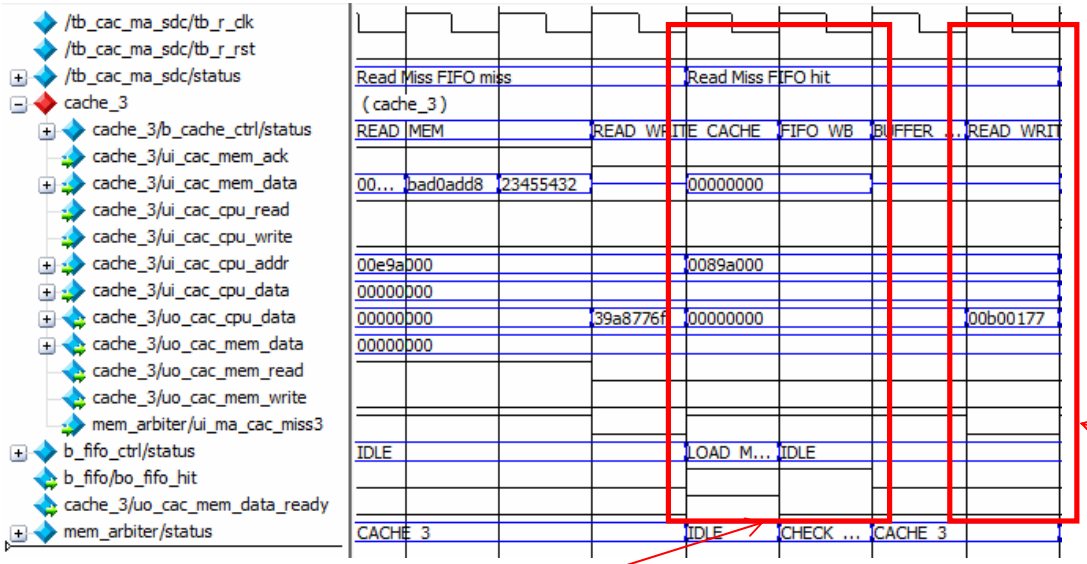
After wait for data written finish into SDRAM, read instruction was resume.

Read miss and FIFO miss (fifo_hit de-asserted) happen since tag is different; data need read back from SDRAM to cache and @89A00 move to FIFO (Test 5 had made dirty to 1).

Data is then read out to CPU

FIFO entries: @89A00, *, *, *

Test 9: Read Miss with FIFO hit in Cache 3



Data in respective location

@89A00	**@E9A00**
00B0_0177	39A8_776F
1234_ABCD	5555_5555
5678_7654	7777_7777
3456_789A	FFFF_FFFF
9876_3210	1212_3434
FAFA_FAFA	0000_0001
BEEF_BEEF	BAD0_ADD8
DEAD_DEAD	2345_5432

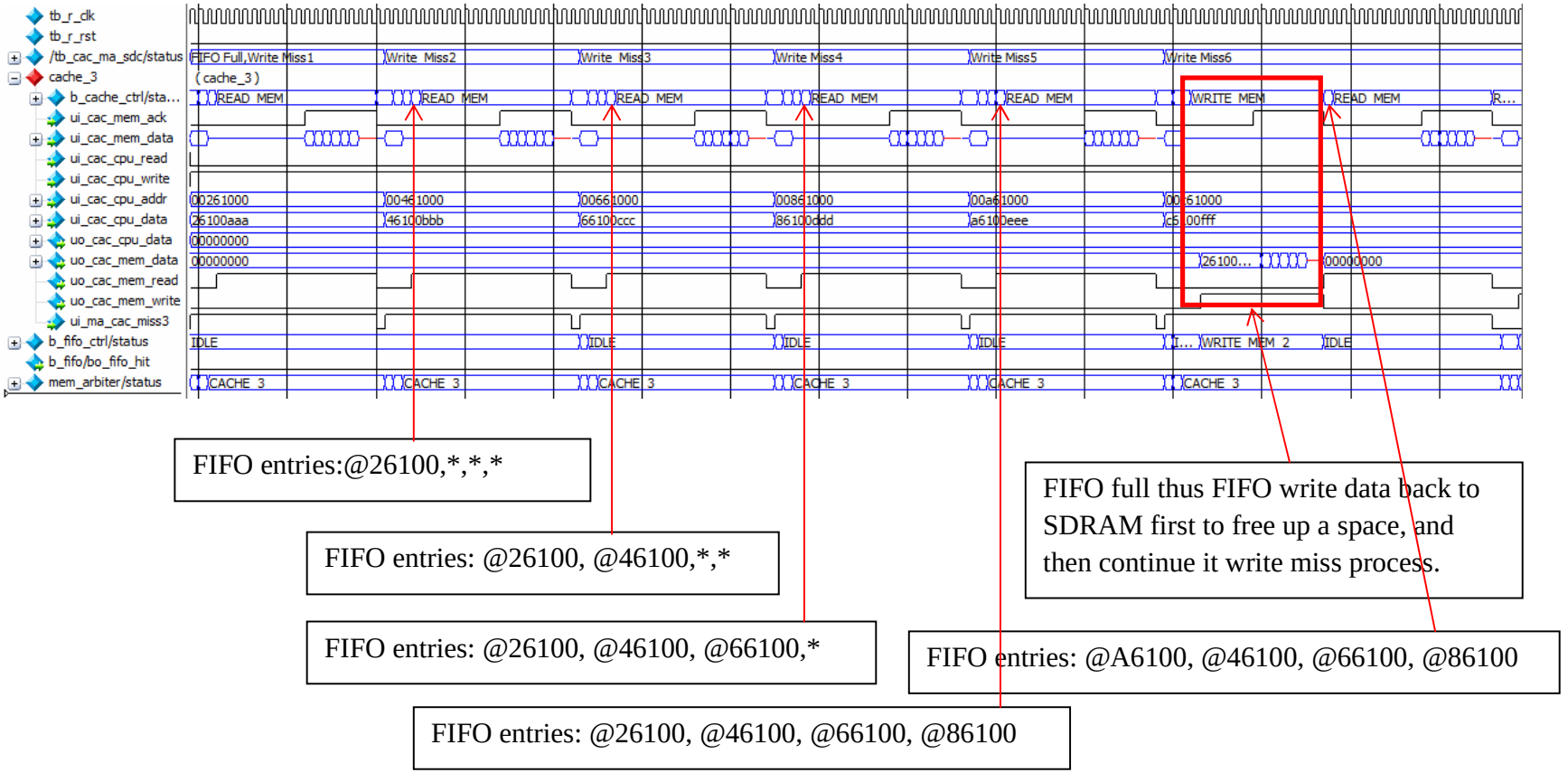
Data is then read out to CPU

Read miss because tag is different and fifo_hit asserted, data written back from FIFO to SDRAM

Since previous instruction is read only so dirty is 0. @E9A00 did not move to FIFO

FIFO entries: *, *, *, *

Test 10: Miss happen and FIFO full



Chapter 8 Conclusion

8.1 Conclusion

Cache unit had successfully redesigned with write-back scheme and write buffer (FIFO) from previous work. With this cache unit, data no longer always need to written back to SDRAM since SDRAM accessing taking 40 to 50 cycles

Now with the new cache unit dirty data able to written back to SDRAM if SDRAM is free while CPU is can do other process. In order to suit in this new ability, a little modification on memory arbiter was made while still keeping the same good feature and functionality of memory arbiter modelled by Chin Chun Lek.

At the end, all the objective of this project is achieved. The cache unit is developed in RTL (Register Transfer Level) form and modeled in synthesizable Verilog. A series of test cases and scenarios has been carried to verified memory system functionality. All the expected results are obtained.

8.2 Discussion and Future Work

With the newly designed cache unit, data no longer always need to written back to SDRAM. In worst case scenario if a miss happen, cache need to access SDRAM twice by writing the dirty data into SDRAM and read another data from SDRAM. With write-back write buffer (FIFO) it can reduce to only read data from SDRAM since dirty data was written into FIFO. Also, if data found in write buffer (FIFO) data can always write back from write buffer (FIFO) and skip the writing from SDRAM. Now with the new cache unit dirty data in FIFO able to written back to SDRAM if SDRAM is free while CPU is can do other process, thus it increase the efficiency use of clock cycle.

Some modifications need to be done in the future work. One is in SDRAM, the acknowledgement signal had two functions in one signal, it indicates load mode is done and data was ready. It is better in to split in two signals to prevent confusion. Next is implementation of Load Mode Instruction in CPU since now did not have a method to change the configuration mode of SDRAM. This need look into pipeline and cache unit and modified both of them.

References

- Araújo J.P.(2002) *Intelligent RAM's: a Radical Solution?* In Proceedings of the 3rd Internal Conference on Computer Architecture, Universidade do Minho http://gec.di.uminho.pt/discip/minf/ac0102/1600IRAM.pdf

- Chin Chun Lek (2015) "32-Bit Memory System Design: Design of Memory Controller for Micron SDR SDRAM" University of Tunku Abdul Rahman, Faculty of Information and Communication Technology

- Ching Yi-lynn (2008) "Memory System Design: Integration of Caches, Translation Lookaside Buffer (TLB) and SDRAM" University of Tunku Abdul Rahman, Faculty of Information and Communication Technology

- Hennessy, J.L. and Patterson, D.A. (2007) *Computer Architecture: A Quantitative Approach*, 4th ed., San Francisco, California: Morgan Kauffmann Publisher

- Micron (n.d) 128Mb x 32 Synchronous DRAM [online] Available at: https://www.micron.com/~/media/documents/products/data-sheet/dram/128mb_x32_sdram.pdf

- MOK, K. M. (2009) *Computer organization and architecture 201210 – memory-basic cache design note,* University of Tunku Abdul Rahman, Faculty of Information and Communication Technology

- Nick Carter (2002) *Schaum's Outline of Theory and Problems of Computer Architecture First Edition*, McGraw-Hill Education Publisher

- Oon Zhi Kang (2008) "SDRAM Enhancement: Design of a SDRAM Controller WISHBONE Industrial Standard" University of Tunku Abdul Rahman, Faculty of Information and Communication Technology

- Wolf, W.(2004), *FPGA-Based System Design*, Upper Saddle River, New Jersey: Prentice-Hall

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

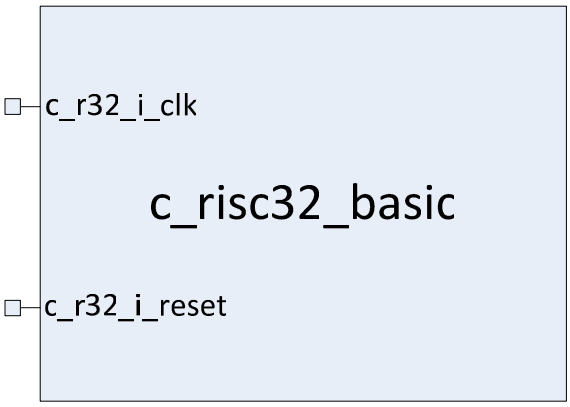


Figure A.3 Block diagram for RISC32-basic processor

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

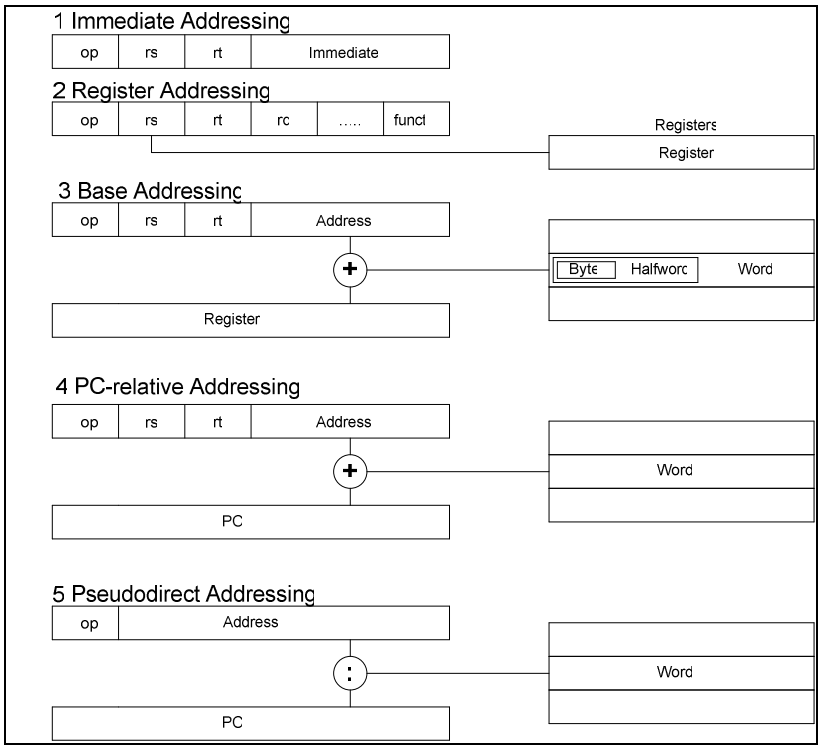


Figure A-6 RISC32 Addressing Mode.

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]) +	addiu $rt, $rs,	no

							U(ZE(Imm))	imm	
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

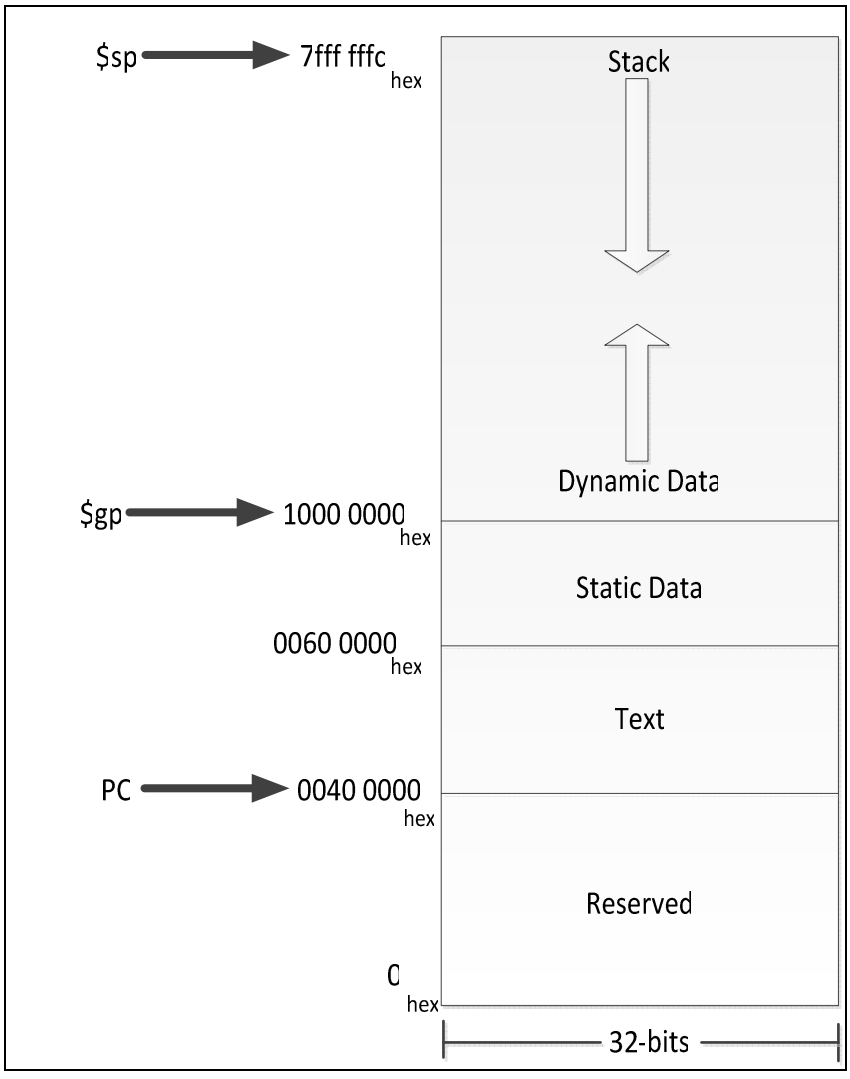


Figure A.8 Memory map for Kuseg section, accessible without CP0

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.