DESIGN AND DEVELOPMENT OF MEMORY SYSTEM FOR 32 BITS 5-STAGE PIPELINED PROCESSOR: MAIN MEMORY (DRAM) INTEGRATION

By

Kim Yuh Chang (09ACB04385)

A Proposal

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfilment of the requirements for the degree of

BACHELOR OF INFORMATION TECHNOLOGY (HONS)

COMPUTER ENGINEERING

Faculty of Information and Communication Technology

Department of Information Technology and Engineering

JAN 2013

DECLARATION OF ORIGINALITY

I declare that this report entitled "**Design and Development of Memory System for 32 bits 5-stage Pipelined Processor: Main Memory (DRAM) 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 : Kim Yuh Chang

Date : 2 April 2013

Abstract

This project is to enhance the RISC32 architecture that developed in UniversitiTunku Abdul Rahman under Faculty of Information and Communication Technology. After reviewing previous work, the RISC32 processor has a readily available SDRAM Controller and 128MB SDRAM functional model provided by Micron but it has not been integrated into the processor yet.

Therefore this project is initiated to integrate the main memory into the processor. The existing SDRAM Controller is build based on Wishbone Compatible Standard while the processor side is not. Therefore, a bus interface unit should be design in order to establish a communication platform for the processor and main memory.Other than that, caches design should be taken into consideration when we are designing the bus interface unit due to whenever there is a cache miss, the processor need to get the data or instruction from SDRAM. This design modeled using Verilog, High Level Description Language and connects to other component.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

Chapter 1: Introduction

1.1: Background

As modern day systems are gradually becoming more and more complex due to their wide functionalities, memory plays an important role in the performance of the system. Many computations executed on current machine are often limited by the response of the memory system rather than the speed of processor [1]. At early in the 1960's, it was the time cache memories were proposed and being introduced into the memory hierarchy as high speed memory buffers used to hold the contents of recently accessed main memory locations. It was already known at that time that recently used information such as instructions and data is likely to be used again in the near future[1-2]. With this method, although cache memory would only hold a small fraction of the contents of main memory, a disproportionate fraction of all memory references would be satisfied by information contained within the cache [5-7]. However, this introduction could not solve the problem perfectly since the size of the cache is inversely proportional to the speed of the memory. As the cache size reduced, miss rate which indicates the chance of data needed was not available inside cache will be increase. When cache miss happens, instruction or data has to be read from the main memory which indicates that several processes have to go through in order to handle cache miss[7]. This is unavoidable as long as we are using cache memory inside our system and this has comes to our topic, main memory integration which responsible to handle the data or instruction transfers between SDRAM and cache memory when cache miss happens.

1.2: Motivation and Problem Statement

Recently, a RISC32 project has been developed in the Faculty of Information and Communication Technology, UniversitiTunku Abdul Rahman. The project is based on the RISC architecture. The main reasons for initiating this project are:

- Microchip design companies develop microprocessors cores as IP for commercial purposes. The microprocessor IP includes information on the entire design process for the front-end (modeling and verification) and back-end (layout and physical design) IC design. These are trade secrets of a company and certainly not made available in the market at an affordable price for research purposes.
- Several freely available microprocessor cores can be found in [1]. Unfortunately, these processors do not implement the entire MIPS Instruction Set Architecture (ISA) and lack comprehensive documentation. This makes them unsuitable for reuse and customization.
- Verification is vital for proving the functionality of any digital design. The microprocessor cores 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 inevitably affect the physical design phase. A design needs to be functionally proven before the physical design phase can proceed smoothly. Otherwise, if the front-end design has to be changed, the physical design process has to be redone.

The RISC32 project will 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 modeling. Currently, a basic central processing unit (CPU) and SDRAM Controller and SDRAM providedby MICRON Technology Inc. has been modeled 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. However, several design issues were found in the existing RISC32

Memory System [9].One of the issues is although the SDRAM Controller and SDRAM has been modeled, it is not ready to integrate into basic CPU that has been modeled previously due to the outputs of CPU is not compatible to current SDRAM since the current RICS32 processor is using a 32 bits address which will cover up to 4GB of memory space. Hence, an additional circuit has been to add on to the current design which acts as a platform for current SDRAM, TLB, MMU, CACHE and others basic CPU to communicate with each others. With all these problems, it is imperative for us to reanalyze and refurbish the foundation of the Memory System before any memory integration can be done.

1.3: Project Scope

This project aims to integrate existing SDRAM Controller and conventional SDRAM into the 32 bits 5-stage pipelined RISC processor.

The scope of this project involves:

1) Designing a bus interface unit which compatible to SDRAM controller, 64MB SDRAM behavioral model provided by MICRON Technology Inc and CPU.
2) The implementation of an industry standard WISHBONE SoC interface in the bus interface unit design to ensure portability.
3) Verify its behavior and functionality at chip level together withTLB, MMU, CACHE, SDRAM controller, 64MB SDRAM behavioral model provided by MICRON Technology Inc and CPU. Timing analysis and synthesis is outside the scope of this project.

1.4: Project Objectives

The project's objectives include:

- Analyze the existing Memory System organization, interfacing and the functionality of a SDRAM and SDRAM Controller. Analysis on the existing MIPS Memory System will be done.
- SDRAM Bus Interface Unit Design – This part includes the development of chip specification and the microarchitecture specification of the SDRAM Bus Interface Unit based on WISHBONE Soc Interface.
- TLB Design – This part include the development of microarchitecture specification of the TLB which used to act as a cache for keeping page table entries.
- MMU Design – This part include the development of microarchitecture specification of MMU (Memor y Management Unit) which responsible to conduct a page table walk through.
- Integration with Cache – This part will include the integration of cache together with existing 64MB of SDRAM, SDRAM Controller, TLB and MMU.
- Verification – Test case will be developed to test the SDRAM and SDRAM controller as a whole by simulating Wishbone master interface signal based on Bus Functional Model and to test whether the design is workable, *lw* and *sw* instructions should be used inside the test case..

1.5: Significance and Impact

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

- A well-developed design document, which includes the chip specification, architecture specification and micro-architecture specification.
- A fully functional well-developed 32-bit RISC architecture core in the form of synthesis-ready RTL written in Verilog.
- A well-developed verification environment for the 32-bit RISC core. The verification specification should contain suitable verification methodology, verification techniques, test plans, testbench architectures etc.
- A complete physical design in FPGA with documented timing and resource usage information.

The RISC32 project is an effort to develop the environment mentioned above: to be used as a multi-cycle pipelined RISC microprocessor core-based platform to support hardware modeling research work.

With the existing well-developed basic RISC32 RTL model (which has been fully functionally verified), the verification environment and the design documents, a researcher can develop his research specific RTL model as part of the RISC32 environment (whether directly modifying the internals of the processor or interface to the processor) and can quickly verify his model to obtain results, without having to worry about the development of the verification environment and the modeling environment. This can hasten the research work significantly. Relating exclusively to this project, the establishment of a strong foundation of the Memory System is important. By building the SDRAM Bus Interface Unit which act as a communicator between SDRAM and CPU, a solid ground will be formed whereby the next designer can focus on fixing other parts of the Memory System.

Chapter 2: Literature Review

2.1: MIPS

MIPS (Microprocessor without Interlocked Pipelined Stage) is a RISC (Reduced Instruction Set Computers) processor which use hardware implementation to directlyexecute instructions, without microprogrammed control. MIPS is widely used in digitalconsumer, networking, personal entertainment, communications and business applications [2], such as Sony Playstation 2, Sony Playstation Portable (PSP) and *Linksys wireless router which primarily used in MIPS implementations. MIPS can be develop using Verilog* – a hardware description language (HDL).

2.2: Memory Hierarchy

When we are discussing about the performance issues in computer architectural design, algorithm predictions, and the low level programming constructs which involve locality of reference, the term, memory hierarchy will always been used in the computer architecture.

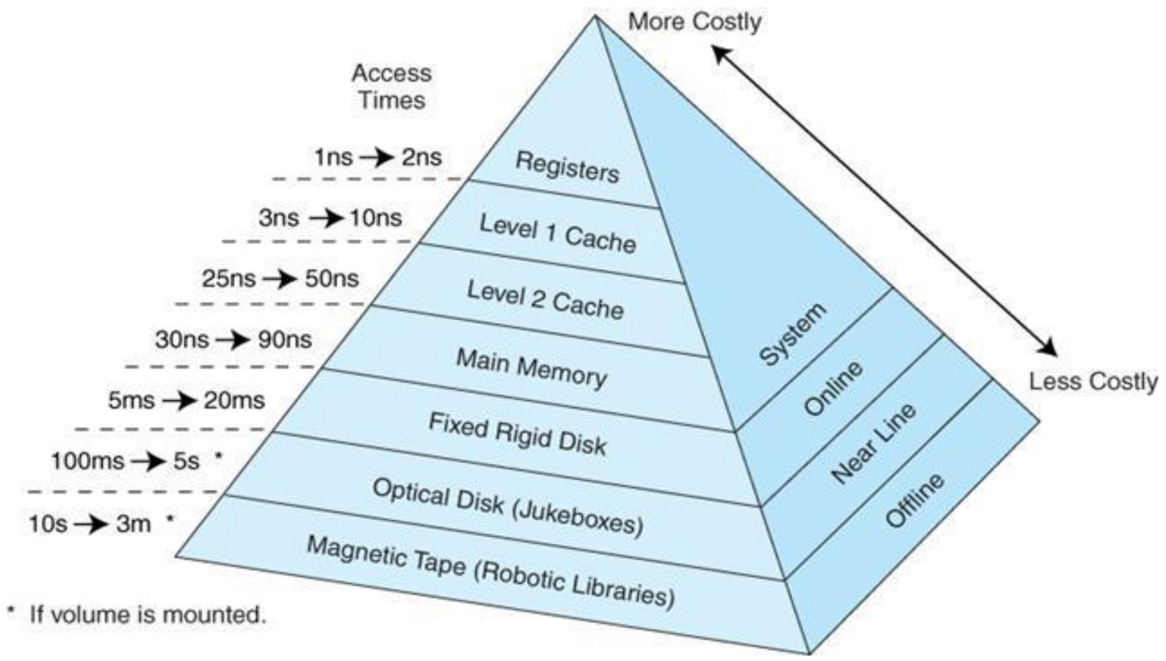


Figure 2.2.1: The Memory Hierarchy(Adapted from [6])

As shown in the diagram above, the memory hierarchy in computer storage is actually distinguishes each level by access time, cost per unit and capacity. Besides, in order to produce a faster access time memory, controlling technology plays an important role in it and therefore, each level of memory hierarchy also can be used to distinguish controlling technology [2,3,6,8].

2.3: Cache and Main Memory Interfacing

From [7], we know that processor is connected to the main memory by a bus system and the bandwidth of the bus system has a significant impact on miss penalty. This is because the clock rate for bus is usually much slower than the processor as much as a factor of 10. Therefore, selection of memory organization to be use in processor plays an important role in deciding the performance of the processor.

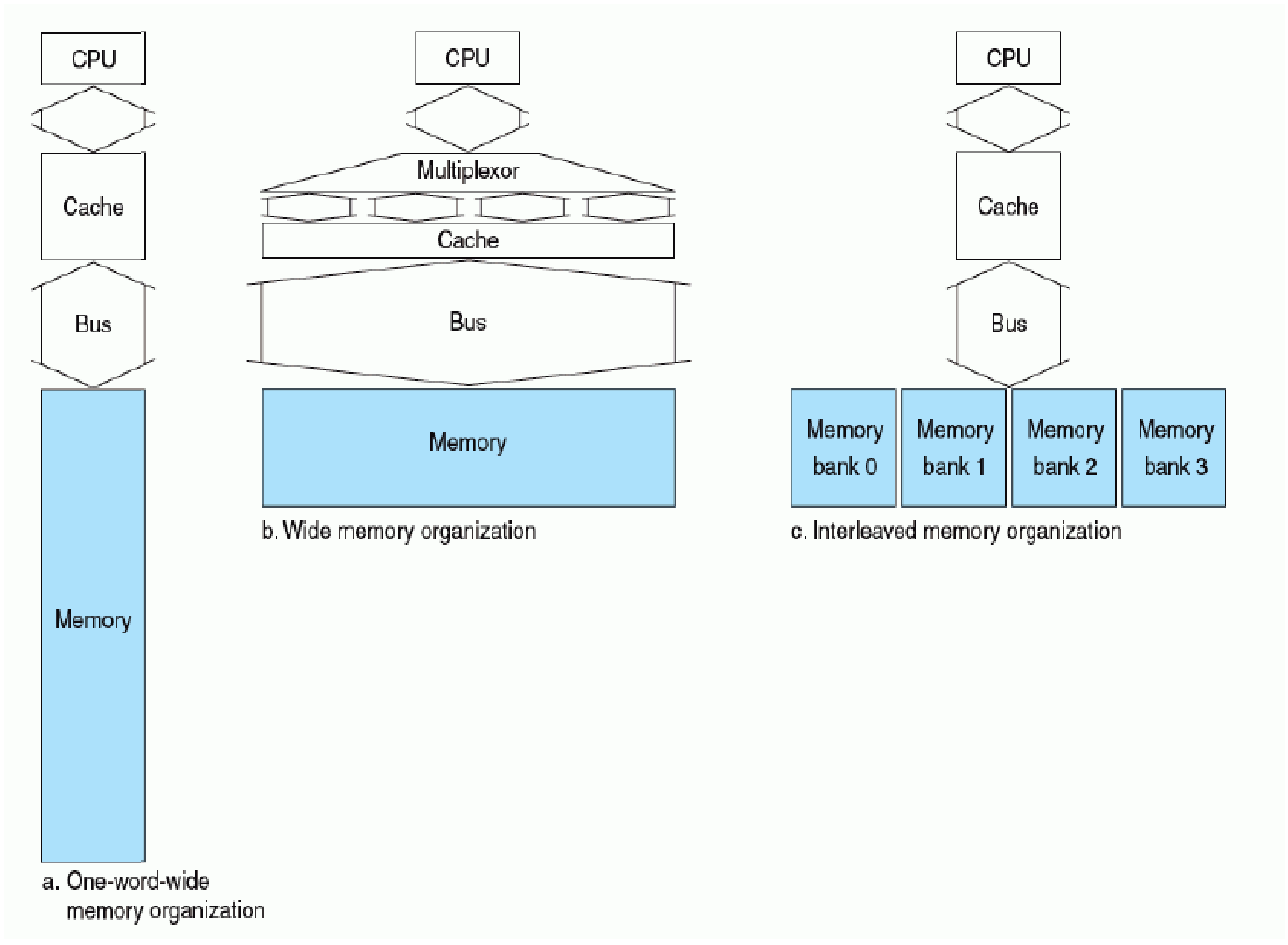


Figure 2.3.1: Memory Organization (Adapted from [7])

Figure on previous page shows three types of available memory organizations which are one-word-wide memory, wide memory and interleaved memory organization. To have a deeper understanding towards the memory organization, let us go through a simple example [7]. Assume that a processor need

- 1 memory bus clock cycle to send the address to main memory.
- 15 memory bus clock cycles for each DRAM access initiated.
- 1 memory bus clock cycle to send a word of data.

Assume that we are going to send 4 words from main memory to cache.

The miss penalty can be calculated by using the equation below:

Miss Penalty =

Send address (1 bus cycle) + Access 1 word in DRAM (15 bus cycles) + Send a word from DRAM to Cache (1 bus cycle)

With all the information given above, we can evaluate the performance of the memory organization shows in *Figure 2.3.1*.

For a one-word-wide memory organization, since it can only fetch one word per time, in another word, it means that the main memory needs to be access 4 times in order to fetch all the data require to the cache. Therefore,

Miss Penalty = 1 + (4 * 15) + (4 * 1) = 65 bus cycles.

For a wide memory organization, it is capable to fetch all the require data in one shot since it has a very high bandwidth of bus system. Therefore,

Miss Penalty = 1 + (1 * 15) + (1 * 1) = 17 bus cycles.

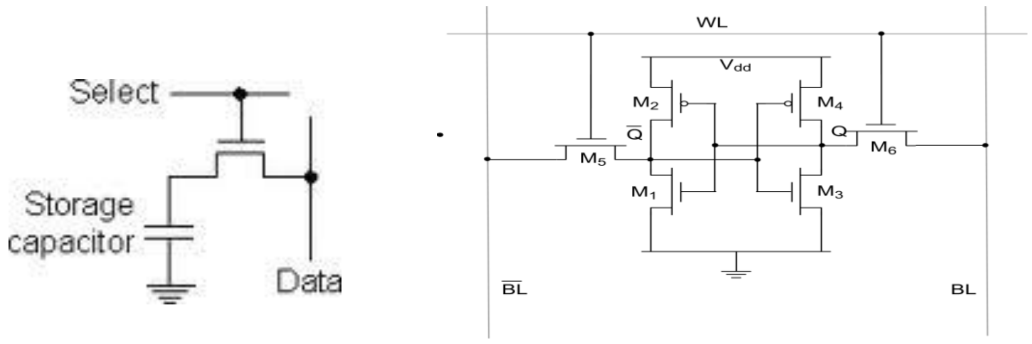
Lastly, a interleaved memory organization, which capable to read multiple words in main memory in a single bus cycle and transfer the data back word by word. Therefore,

Miss Penalty = 1 + (1 * 15) + (4 * 1) = 21 bus cycles.

The calculations above shows that a wide memory organization has the least miss penalty but keep in mind that a huge bus system is not easy to manage and it require a high cost to implement. For the interleaved memory organization, although it is slower than wide memory organization, it is using a shared bus system among the memory banks. This reduces the cost to implement but this will results in a similar performance with wide memory organization.

2.4: DRAM

Dynamic Random-Access Memory (DRAM) is a type of random access memory that will stores each bit of data in a separate of capacitor within an integrated circuit. It is a non-volatile memory that the data stored inside will be lost once the power supply been turned off. Due to the characteristic of capacitor which is charging and discharging, these states are taken to represent two values of bit which are 0 and 1. DRAM is always cost lesser than Static Random Access Memory (SRAM) due to its simple structural which only consists of one transistor and one capacitor per bit comparing to SRAM which is using 4 or 6 transistors depends on the design [6-8]. With this structure, DRAM can be designed to reach a very high density but as a tradeoff, the accessing time of DRAM is slower than SRAM. Other than that, since capacitors leak charge, the information stored inside will eventually fades unless the capacitor is being refreshed periodically.



DRAM ***SRAM***

Figure 2.4.1:The structure of DRAM and SRAM. (Adapted from [7])

2.5: SDRAM

Synchronous Dynamic Random Access Memory (SDRAM) is a DRAM that is synchronized with the system bus. The previous DRAM we had discussed has an asynchronous interface in which it responds as quickly as possible to changes in control input while SDRAM has a synchronous interface, meaning it will wait for a rising edge of clock signal before responding to control input[5].

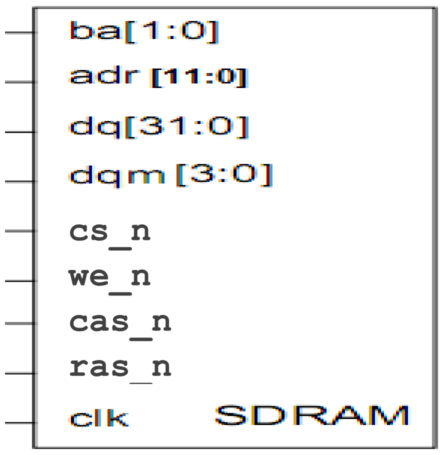


Figure 2.5.1: Block diagram of 128Mb banks SDRAM(Adapted from [9])

Pin Name	Size	Description
ba [1:0]	2 bits	Bank Address: Define to which device bank the ACTIVE, READ, WRITE or PRECHARGED is being applied.
adr [31:0]	12 bits	Address Bus: Used as an input to send column address, row address and configuration setting to the SDRAM.
dq [31:0]	32 bits	Data Line: 32 bits bidirectional data line to/from SDRAM.
dqm [4:0]	4 bits	Data Mask: Used to select which byte of the 32 bits bidirectional data line, dq, is valid.
cs_n	1 bits	Chip Select: When this signal is high, the chip ignores all other inputs except clock signal, and acts as if a NOP command is received.

we_n	1 bits	Write Enable: Along with /RAS and /CAS, this selects one of 8 commands. This generally distinguishes read-like commands from write-like commands.
cas_n	1 bits	Column Address Strobe: Along with /RAS and /WE, this selects one of 8 commands.
ras_n	1 bits	Row Address Strobe: Along with /CAS and /WE, this selects one of 8 commands.
clk	1 bits	Clock Signal: Used to synchronize with the CPU bus system.

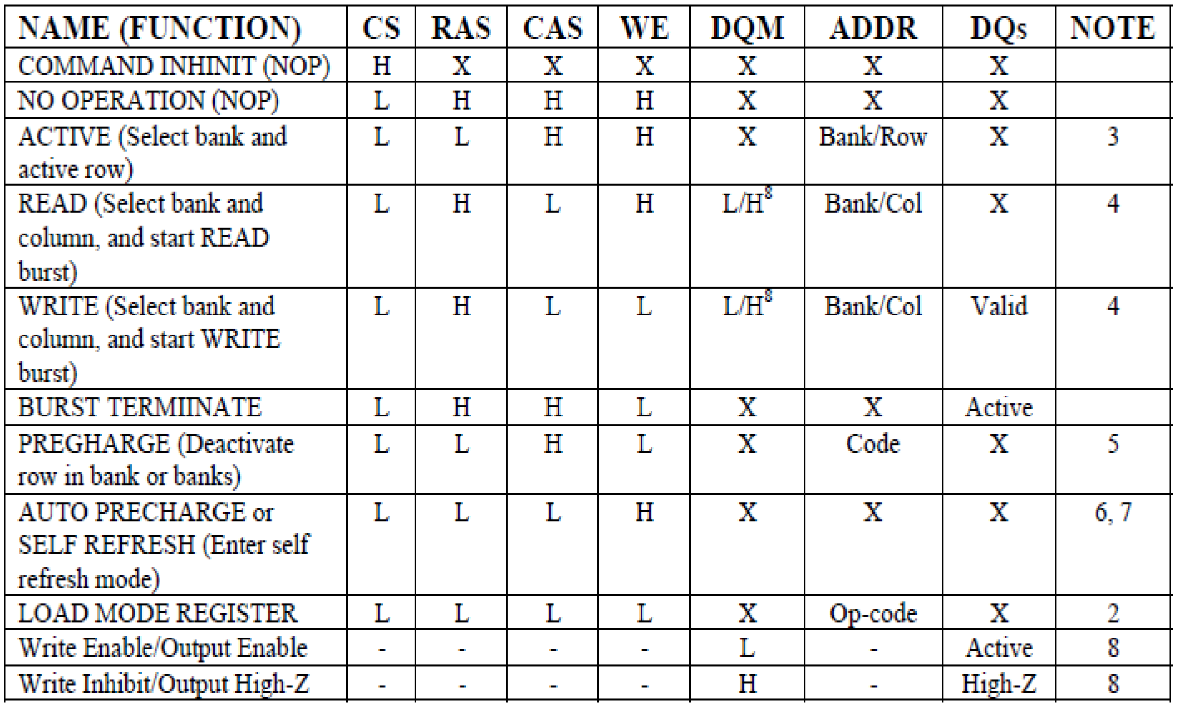
Table2.5.1: I/O description table of SDRAM.

The SDRAM has adopted bidirectional data line, dq, for write transfer and read transfer. This is because the SDRAM can only do one of the operations at a time. The granularity of a bus is defined as the smallest transfer can be done by that bus. This is accomplished using the data masking pin, dqm(3:0). The data masking pin is used to select which byte of the 32-bit bidirectional data line, dq, is valid.

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

There are several functions available to control the activity of SDRAM by varying the control signals such as cs_n, ras_n, cas_n, we_n. These control signals are normally issued by a SDRAM Controller.

The table below provided a quick reference of available command for SDRAM:

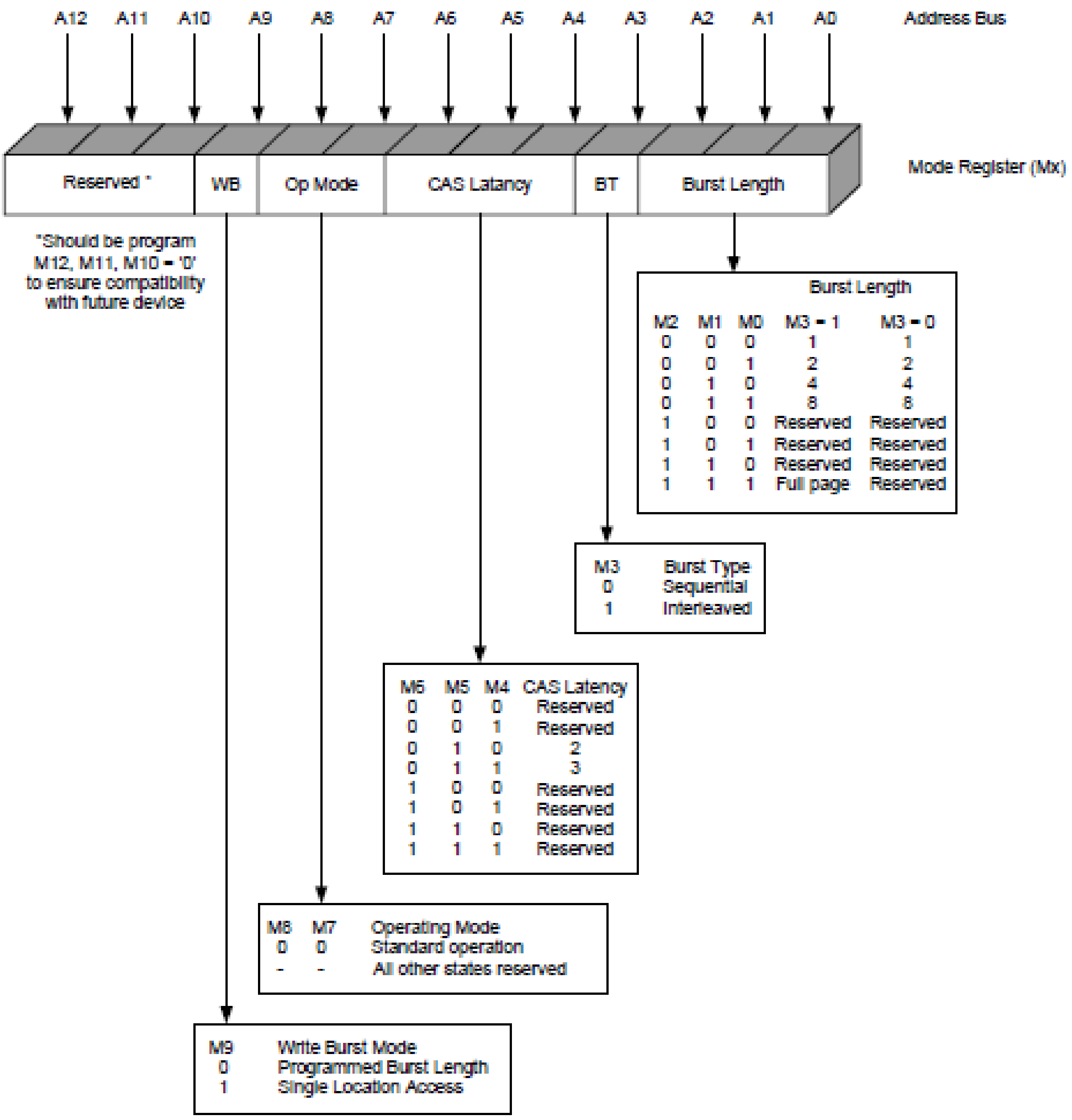


NAME (FUNCTION)	CS	RAS	CAS	WE	DQM	ADDR	DQs	NOTE
COMMAND INHINIT (NOP)	H	X	X	X	X	X	X	
NO OPERATION (NOP)	L	H	H	H	X	X	X	
ACTIVE (Select bank and active row)	L	L	H	H	X	Bank/Row	X	3
READ (Select bank and column, and start READ burst)	L	H	L	H	L/H[8]	Bank/Col	X	4
WRITE (Select bank and column, and start WRITE burst)	L	H	L	L	L/H[8]	Bank/Col	Valid	4
BURST TERMIINATE	L	H	H	L	X	X	Active	
PREGHARGE (Deactivate row in bank or banks)	L	L	H	L	X	Code	X	5
AUTO PRECHARGE 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	2
Write Enable/Output Enable	-	-	-	-	L	-	Active	8
Write Inhibit/Output High-Z	-	-	-	-	H	-	High-Z	8

Table2.5.2: Truth Table – Command and DQM operation. (Adapted from [4])

Other than that, by using adr[11:0] pin of the SDRAM, we can configure the mode register which used to define the specific mode of operation for SDRAM via the LOAD MODE REGISTER command and the information stored will be retain until it has been reprogrammed or the device has been powered off. The definition includes the selection of burst length, burst type, CAS latency, operating mode and write burst mode.

Burst is a technique used to continuous read or write data from the memory depends on the burst length. For example, if we set the burst length to be 4 and it is a READ operation, the data inside SDRAM will be read 4 times continuously. The sequences of the data read or write will be either in sequential or interleaved order which shows in *Table 2.4.3*.



			Burst Length	
M2	M1	M0	M3 = 1	M3 = 0
0	0	0	1	1
0	0	1	2	2
0	1	0	4	4
0	1	1	8	8
1	0	0	Reserved	Reserved
1	0	1	Reserved	Reserved
1	1	0	Reserved	Reserved
1	1	1	Full page	Reserved

M3	Burst Type
0	Sequential
1	Interleaved

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

M8	M7	Operating Mode
0	0	Standard operation
-	-	All other states reserved

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

Figure 2.5.2: Mode Register Definition Diagram. (Adapted from [4])

The description of each definition shown above will be discussed as below:

- Burst Length
  Used to determine maximum number of column locations that can be accessed for a given READ or Write command.

- Burst Type

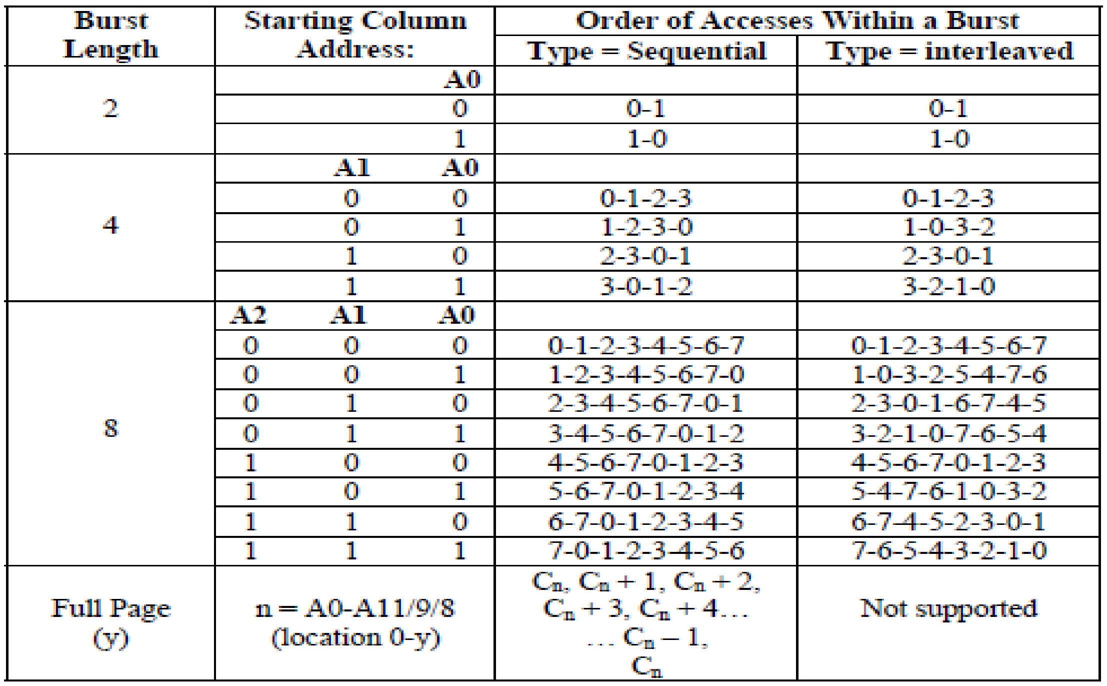
  Used to select either sequential or interleaved burst to be adopted by SDRAM. The ordering of accesses within a burst is determined by burst length, burst type, starting column address.
- CAS Latency

  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

  Used to 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.



Burst Length	Starting Column Address:			Order of Accesses Within a Burst	
				Type = Sequential	Type = interleaved
2			A0		
			0	0-1	0-1
			1	1-0	1-0
4		A1	A0		
		0	0	0-1-2-3	0-1-2-3
		0	1	1-2-3-0	1-0-3-2
		1	0	2-3-0-1	2-3-0-1
		1	1	3-0-1-2	3-2-1-0
8	A2	A1	A0		
	0	0	0	0-1-2-3-4-5-6-7	0-1-2-3-4-5-6-7
	0	0	1	1-2-3-4-5-6-7-0	1-0-3-2-5-4-7-6
	0	1	0	2-3-4-5-6-7-0-1	2-3-0-1-6-7-4-5
	0	1	1	3-4-5-6-7-0-1-2	3-2-1-0-7-6-5-4
	1	0	0	4-5-6-7-0-1-2-3	4-5-6-7-0-1-2-3
	1	0	1	5-6-7-0-1-2-3-4	5-4-7-6-1-0-3-2
	1	1	0	6-7-0-1-2-3-4-5	6-7-4-5-2-3-0-1
	1	1	1	7-0-1-2-3-4-5-6	7-6-5-4-3-2-1-0
Full Page (y)	n = A0-A11/9/8 (location 0-y)			C_n, $C_n + 1$, $C_n + 2$, $C_n + 3$, $C_n + 4$… … $C_n - 1$, C_n	Not supported

Table2.5.3: Burst Definition Table.(Adapted from [4])

2.6: SDRAM Controller

The SDRAM Controller is used to acts as a communicator between the host and SDRAM. As the SDRAM Controller receive the operation command from the host, it will interpret it and translate into a control signal which acts as an input to the SDRAM. The SDRAM Controller has been previously modeled based on industry standard WISHBONE SoC interface [9].

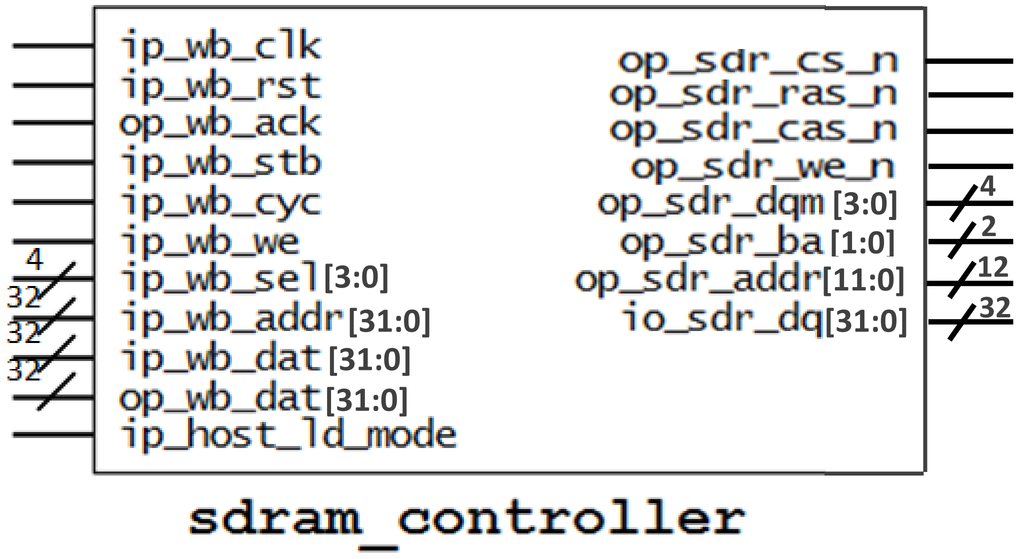


Figure 2.6.1: Block diagram of SDRAM Controller.
(Modified from [9])

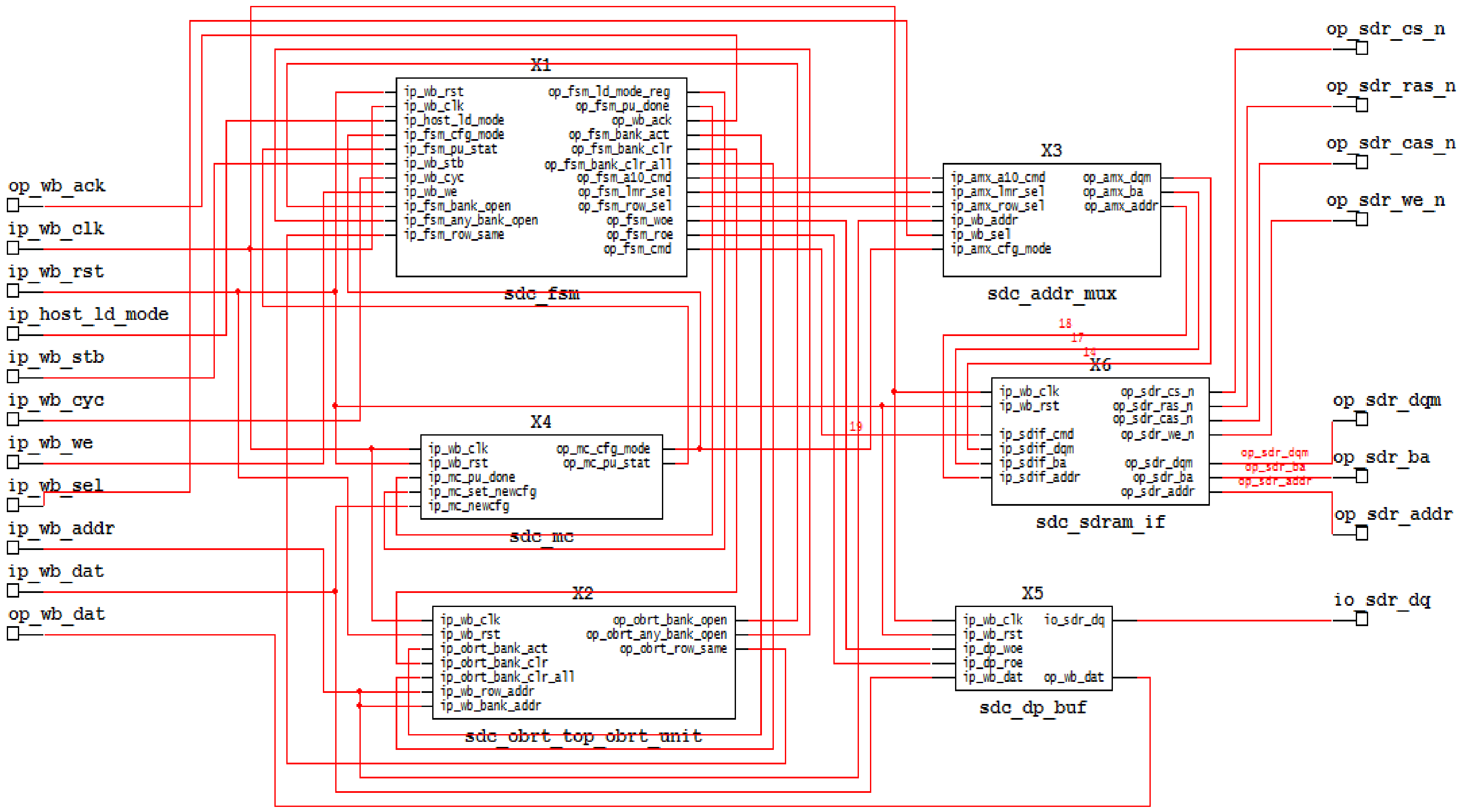
Pin Name	Size (bits)	Description
ip_wb_clk	1	Clock signal to synchronize to the system.
ip_wb_rst	1	Synchronous reset to reinitialize the system.
ip_wb_cyc	1	Asserted to indicate valid bus cycle is in progress.
ip_wb_stb	1	Asserted to indicate the SDRAM controller is selected.
ip_wb_we	1	Asserted to indicate that the current cycle is READ. Deasserted to indicate current cycle is WRITE.
op_wb_ack	1	Asserted to indicate that the current READ or WRITE operation is successful.

ip_wb_sel [3:0]	4	Used to indicate where valid data is placed on the input data line (ip_wb_dat) during WRITE cycle and where it should present on the output data line (op_wb_dat) during READ cycle.
ip_wb_addr [31:0]	32	Used to pass the memory address from the host.
ip_wb_dat [31:0]	32	Used to pass WRITE data from the host.
op_wb_dat [31:0]	32	Used to output READ data from the SDRAM.
ip_host_ld_mode	1	Asserted to load a new mode into the SDRAM.
op_sdr_cs_n	1	SDRAM chip select.
op_sdr_ras_n	1	SDRAM row address select.
op_sdr_cas_n	1	SDRAM column address select.
op_sdr_we_n	1	SDRAM write enable.
op_sdr_addr [11:0]	12	Address output to the SDRAM.
op_sdr_ba [1:0]	2	Bank Address output to SDRAM.
op_sdr_dqm [3:0]	4	Used to select which bits of the data line (io_sdr_dq) to be masked.
io_sdr_dq [31:0]	32	Bidirectional data line to receive READ data or send WRITE data.

Table2.6.1: I/O pin description of SDRAM Controller.

**Note that ip represents input, op represents output, wb represents WISHBONE, sdr represents SDRAM.*

By using this SDRAM Controller, we can make a direct LOAD MODE REGISTER command straight from the host. To load the configuration to the SDRAM, the host nee to asserted for the pin ip_host_ld_mode. This can help in speeding upwhen configuring SDRAM since in the reality not only one SDRAM will be connected to this SDRAMController



Figures 2.6.2: The Microarchitecture of SDRAM Controller.

The figure on the previous page shows the microarhitecture of the SDRAM Controller. Inside the figure, the block *sdc_obrt_top_obrt_unit* is used to track the row status of all of the banks. Block *sdc_mc* is responsible to store the status of the SDRAM configuration and also the power up status to indicate if the SDRAM controller is executing the initialization protocol or not.The address multiplexer, *sdc_addr_mux* partitions the WISHBONE address input line into row address, bank address and column address. Then, it multiplexes the configuration mode, row address and column address. It also decodes the WISHBONE Select input pin and converts it to equivalent masking output.

Besides, block *sdc_dp_buf* is used to controls the flow of the data between SDRAM and Host while block *sdc_sdram_if*is the SDRAM Interface Block that synchronizes all the signals to the negative edge before sending them out the SDRAM.

Other than that, SDRAM Controller also responsible to instruct the SDRAM to initiate a precharge in order to maintain the information stored inside each cell. Otherwise, the information stored inside each cell will be lost due to the characteristic of capacitor which is the voltage will slowly leak off.The finite state machine below shows how the SDRAM Controllerhandles the timing and the state changes that forms the protocols of the SDRAM. It helps in decide which protocol to be executed and what commands to be sent to the SDRAM by using the *sdc_fsm* block.

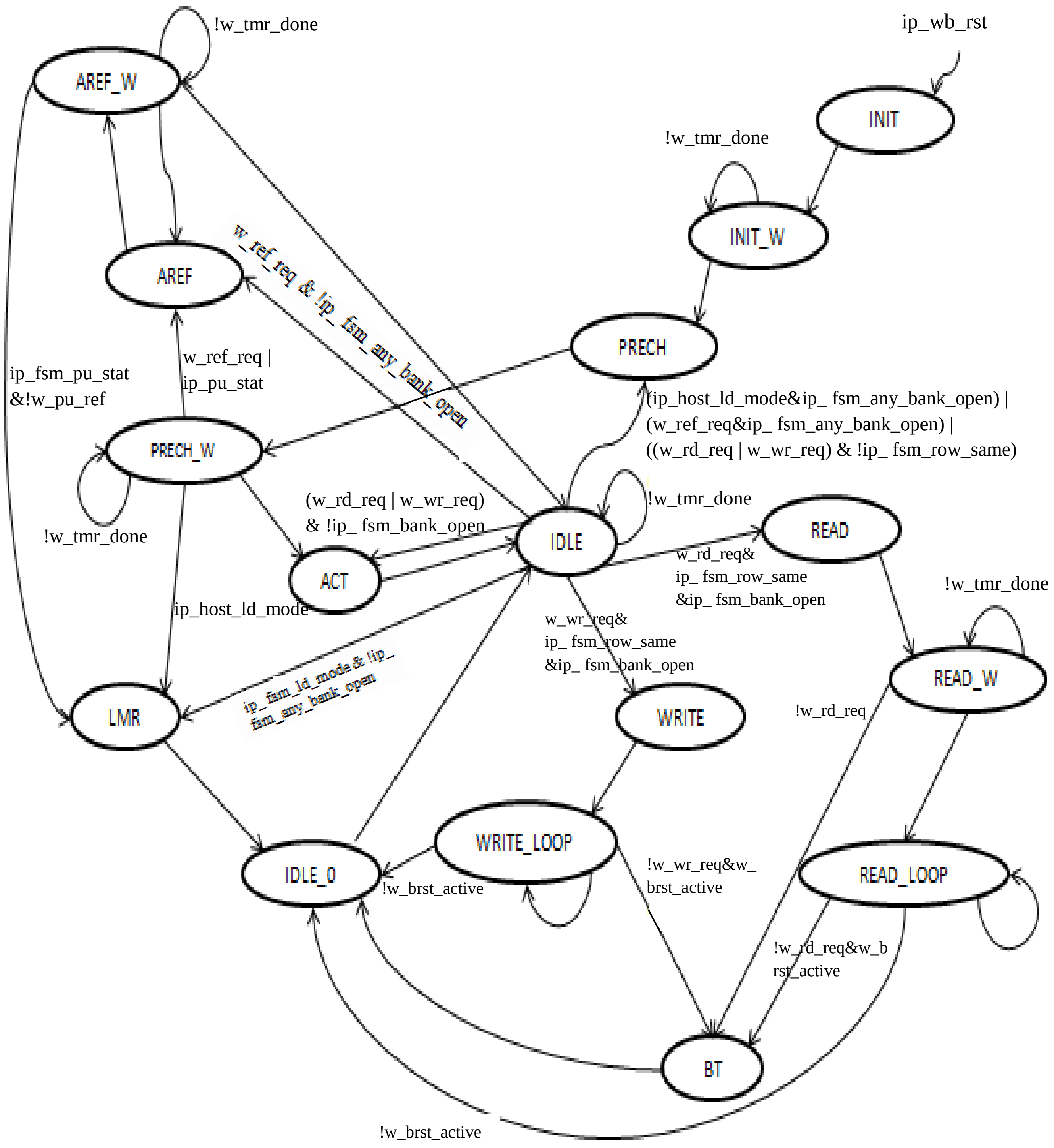


Figure 2.6.3: Sub Module of SDRAM Controller –

Protocol Controller Block Finite State Machine (Adapted from [9])

State Definitions of Protocol Controller Block

State Name	Definition
INIT	Initialization
INIT_W	Wait for power up delay. The delay needed is dependence on the SDRAM manufacturer
PRECH	Send Precharge command
PRECH_W	Wait row precharge delay time
AREF	Send Auto-Refresh command
AREF_W	Wait refresh delay time
LMR	Send Load Mode command
IDLE_0	Wait operation to complete
IDLE	Wait for new operation
ACT	Send Active command
WRITE	Send Write command
WRITE_LOOP	Write data
READ	Send Read Command
READ_W	Wait CAS Latency
READ_LOOP	Read data
BT	Send Burst Terminate command

Table 2.6.2: State Definitionsof Protocol Controller Block(Adapted from [9])

Output or Behaviors of Protocol Controller Block Corresponding to the States

State Name	Correspondence Output Behaviors
INIT	op_fsm_cmd<= `CMD_NOP; r_brst_cnt<= 0; r_pu_cnt<= 2; r_ri_cnt<= `REF_INTERVAL; r_tmr_val<= `WAIT_150us; op_wb_ack<= 0;
INIT_W	op_fsm_cmd<= `CMD_NOP;
PRECH	op_fsm_cmd<= `CMD_PRECH; op_fsm_bank_clr<= !(w_ref_req | ip_fsm_pu_stat); op_fsm_bank_clr_all<= (w_ref_req | ip_fsm_pu_stat | ip_host_ld_mode); op_fsm_a10_cmd <= (w_ref_req | ip_fsm_pu_stat | ip_host_ld_mode); r_tmr_val<= `TRP_DEF – 13’d1;

PRECH_W	op_fsm_ld_mode_req<= ip_host_ld_mode;
	op_fsm_cmd<= `CMD_NOP;

AREF	op_fsm_cmd<= `CMD_AREF; r_pu_cnt<= ip_fsm_pu_stat? r_pu_cnt – 1: r_pu_cnt; r_ri_cnt<= `REF_INTERVAL; r_tmr_val<= tRFC constant – 1;
AREF_W	op_fsm_cmd<= `CMD_NOP; r_ri_cnt<= `REF_INTERVAL;
LMR	op_fsm_cmd<= `CMD_LMR; op_fsm_lmr_sel<= 1; op_fsm_pu_done<= ip_fsm_pu_stat? 1: 0; r_tmr_val<= {2'b00, `TMR_DEF} – 13'd1; op_wb_ack<= ip_wb_cyc&ip_wb_stb&ip_host_ld_mode;
IDLE_0	op_fsm_cmd<= `CMD_NOP;
IDLE	op_fsm_cmd<= `CMD_NOP;
ACT	op_fsm_cmd<= `CMD_ACT; op_fsm_bank_act<= 1 op_fsm_row_sel<= 1; r_tmr_val<= {1'b0,`TRCD_DEF} – 13'd1;
WRITE	op_fsm_cmd<= `CMD_WR; r_brst_cnt<= r_brst_val – 1; r_tmr_val<= {2'b00,`TWR_DEF} – 13'd1; op_fsm_woe<= 1; op_wb_ack<= ip_wb_cyc&ip_wb_stb;
WRITE_LOOP	op_fsm_cmd<= `CMD_NOP; r_brst_cnt<= r_brst_cnt – 1; r_tmr_val<= {2'b00,`TWR_DEF} – 13'd1; op_fsm_woe<= 1; op_wb_ack<= ip_wb_cyc&ip_wb_stb;

READ	op_fsm_cmd<= `CMD_RD;
	r_brst_cnt<= r_brst_val;
	r_tmr_val<= {1'b0,ip_fsm_cfg_mode[6:4]} – 13'd1;

READ_W	op_fsm_cmd<= `CMD_NOP;

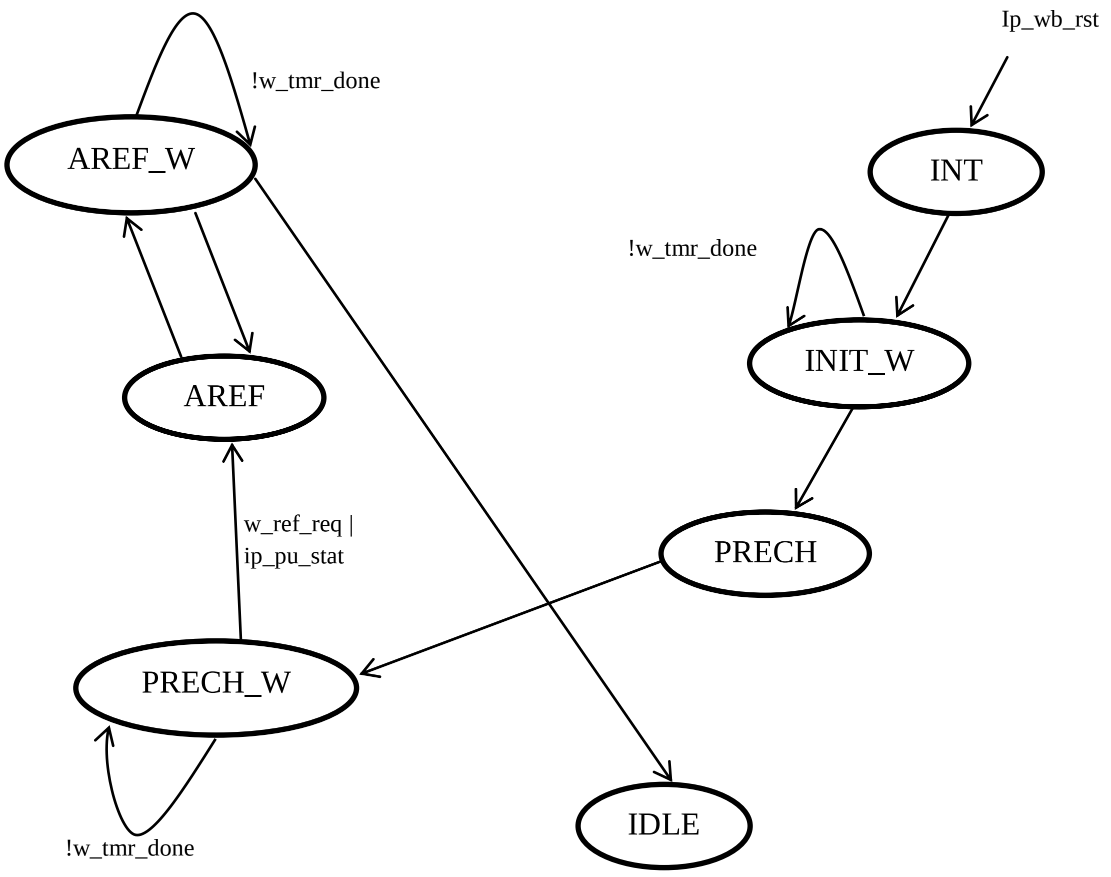
READ_LOOP	op_wb_ack<= ip_wb_cyc&ip_wb_stb&r_roe;
	op_fsm_cmd<= `CMD_NOP;
	r_brst_cnt<= r_brst_cnt – 1;
	r_roe<= 1;

BT	op_fsm_cmd<= `CMD_BT;
	r_brst_cnt<= 0;

Table 2.6.3: Output or Behaviors of Protocol Controller BlockCorresponding to the States (Adapted from [9])

With the help of the protocol controller block, all the states and operations need to be done by SDRAM have been fully specify and been show clearly. With the aid of this sub module, the SDRAM Controller can initiate a refreshing circuit whenever it is necessary without receiving any command from the CPU. Due to the complexity of finite state machine, in order to have a ease way to understand what Protocol Controller do, process of understanding how the protocol controller had been conducted and as a result, individual process has been successfully been figured out.

Initialization Protocol



Figures 2.6.4: This protocol follows the recommended SDRAM initialization requirement given by MICRON.

Keep Bank and Row Open Access Protocol

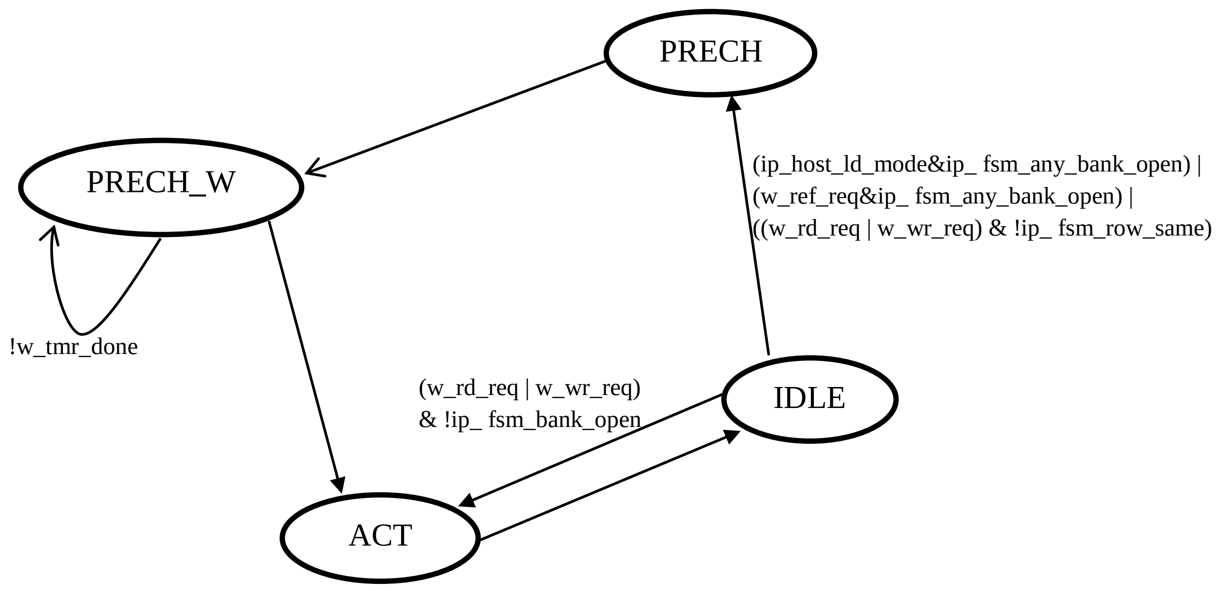

PRECH
PRECH_W
IDLE
ACT
(ip_host_ld_mode&ip_ fsm_any_bank_open) |
(w_ref_req&ip_ fsm_any_bank_open) |
((w_rd_req | w_wr_req) & !ip_ fsm_row_same)
!w_tmr_done
(w_rd_req | w_wr_req)
& !ip_ fsm_bank_open

Figure 2.6.5: Keep Bank and Row Open Access Protocol to to achieve fast access cycle for same row accesses.

Load Mode Protocol (Initialization Stage)

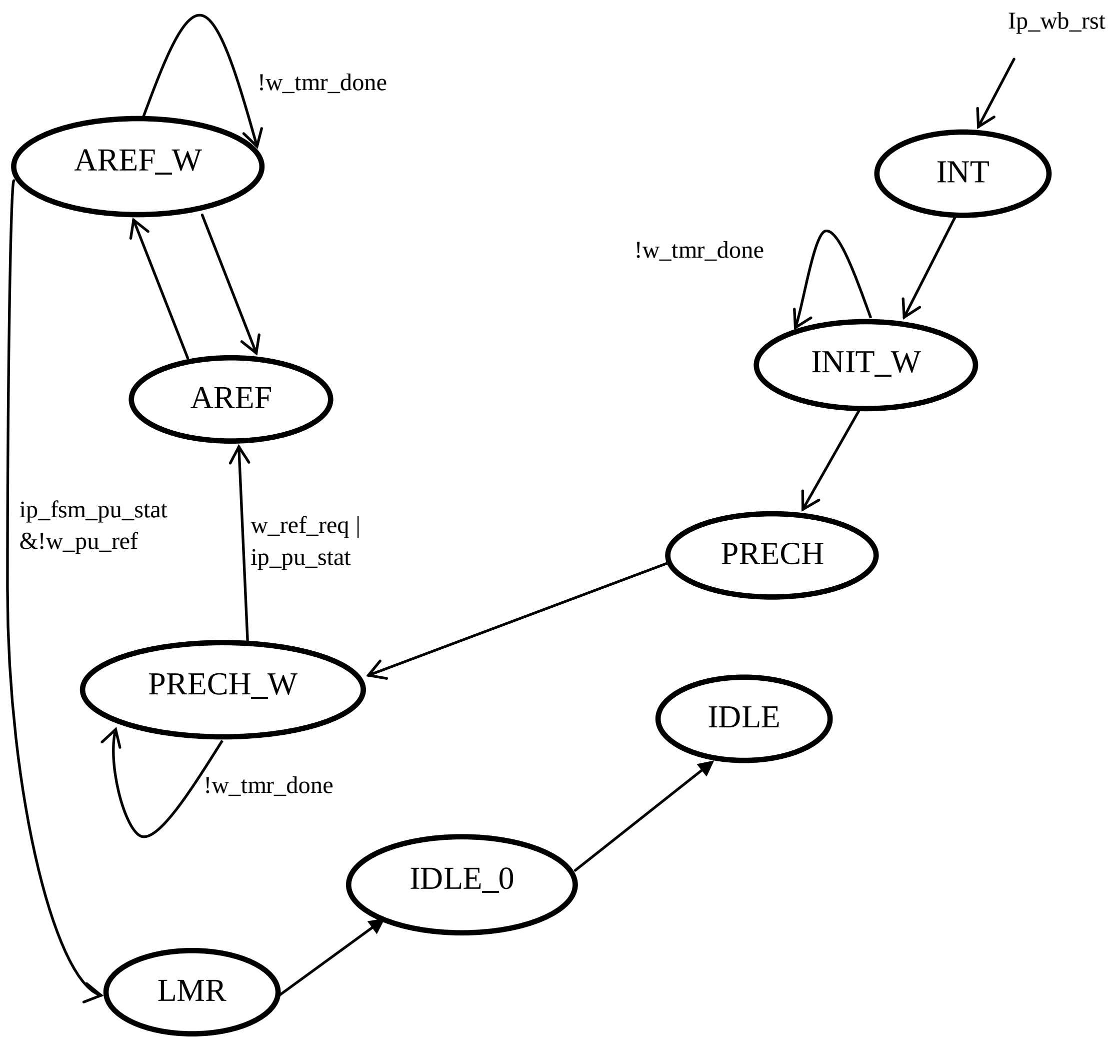


Figure 2.6.6: Load Mode Protocol when in the initialization stage.

Load Mode Protocol (Post Initialization Stage)

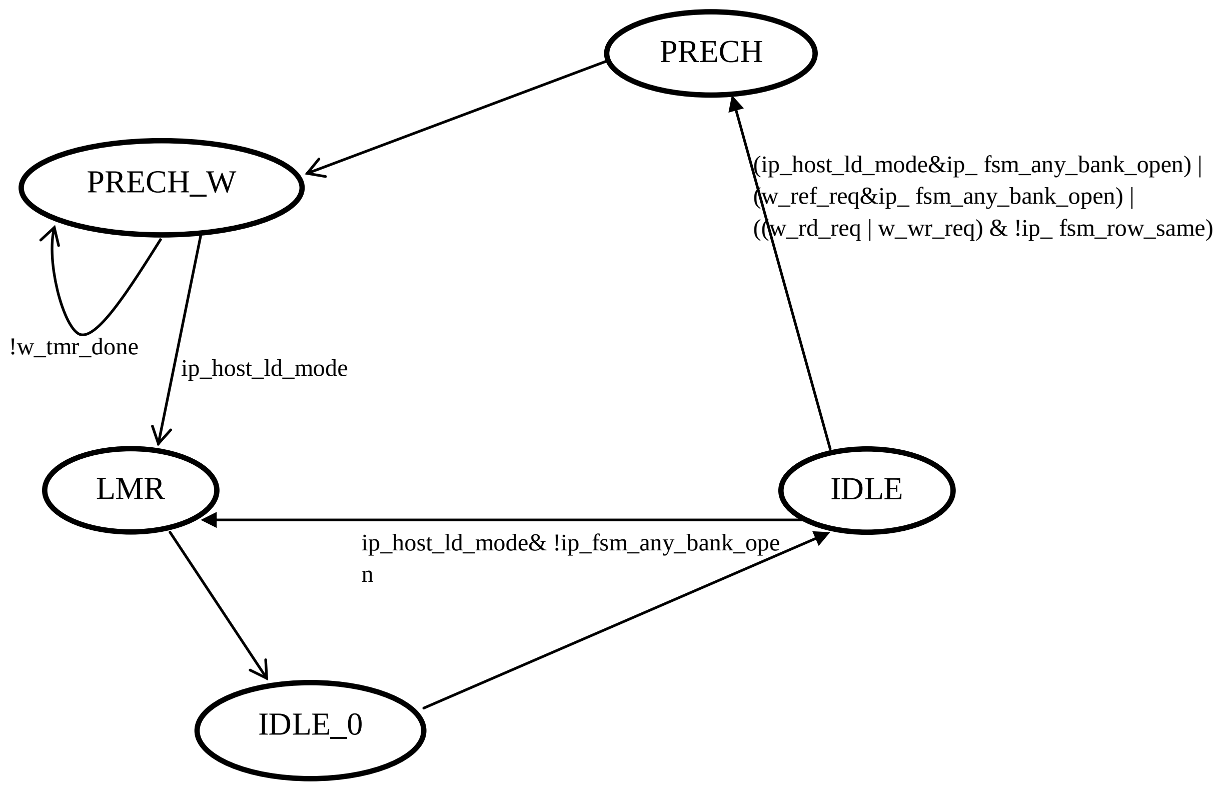

PRECH
PRECH_W
(ip_host_ld_mode&ip_ fsm_any_bank_open) |
(w_ref_req&ip_ fsm_any_bank_open) |
((w_rd_req | w_wr_req) & !ip_ fsm_row_same)
!w_tmr_done
ip_host_ld_mode
LMR
IDLE
ip_host_ld_mode& !ip_fsm_any_bank_ope
n
IDLE_0

Figure 2.6.7: Load Mode Protocol when in the post initialization stage.

Auto Refresh Protocol (Post Initialization Stage)

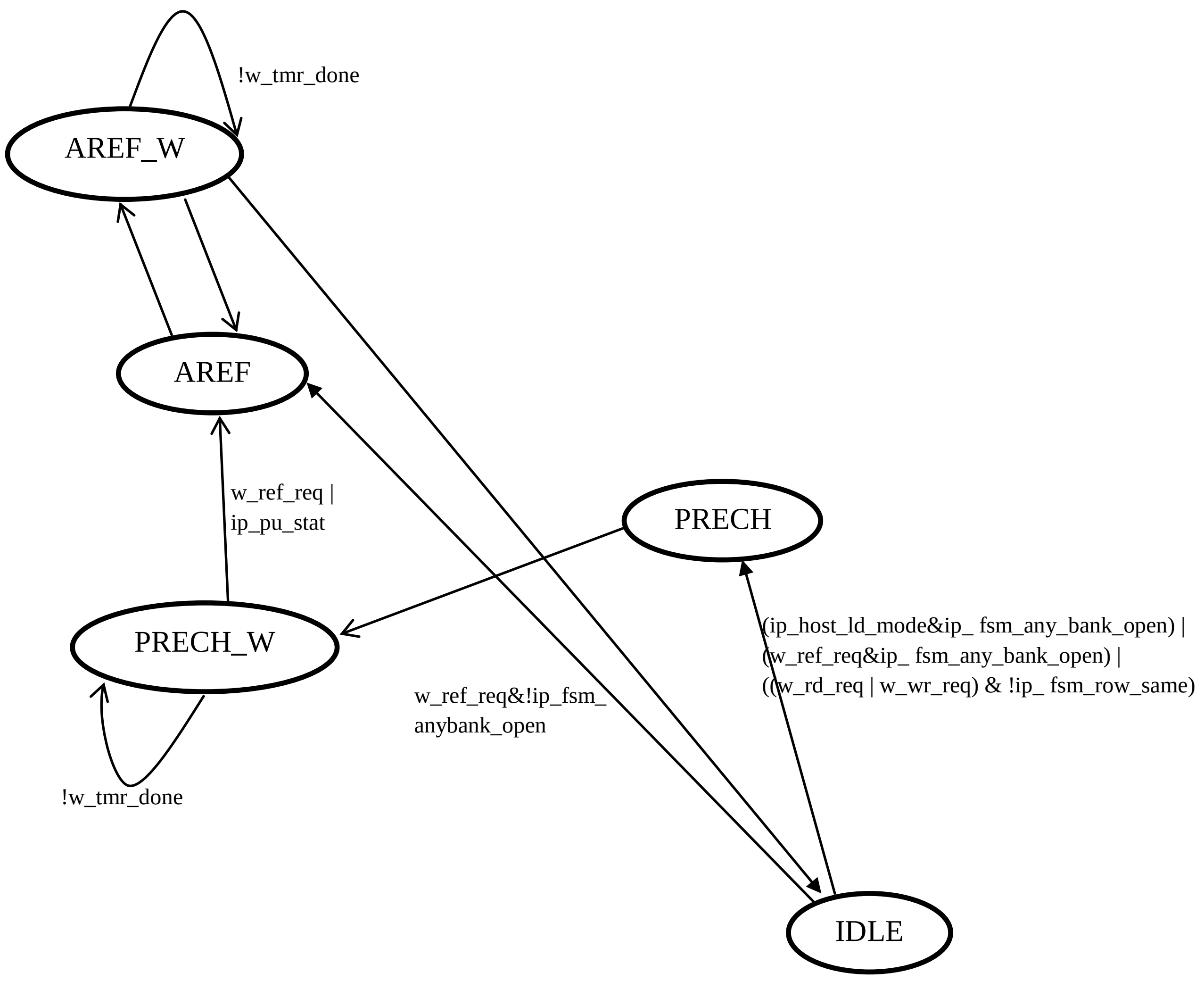

AREF_W
!w_tmr_done
AREF
w_ref_req |
ip_pu_stat
PRECH
PRECH_W
!w_tmr_done
(ip_host_ld_mode&ip_ fsm_any_bank_open) |
(w_ref_req&ip_ fsm_any_bank_open) |
((w_rd_req | w_wr_req) & !ip_ fsm_row_same)
w_ref_req&!ip_fsm_
anybank_open
IDLE

Figure 2.6.8: Auto Refresh Protocol when in the post initialization stage.

Read Protocol

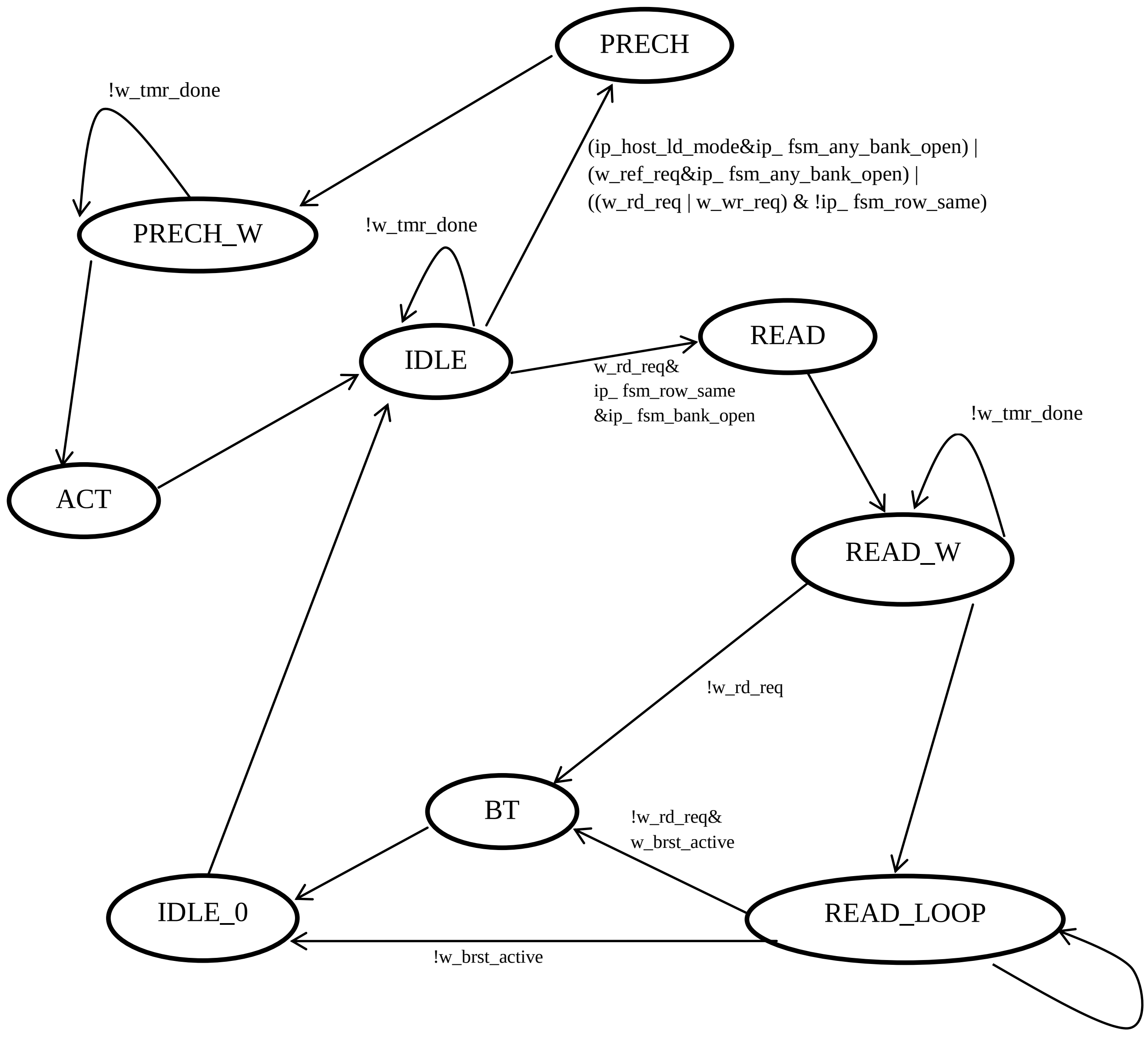


Figure 2.6.9: Read Protocol.

Write Protocol

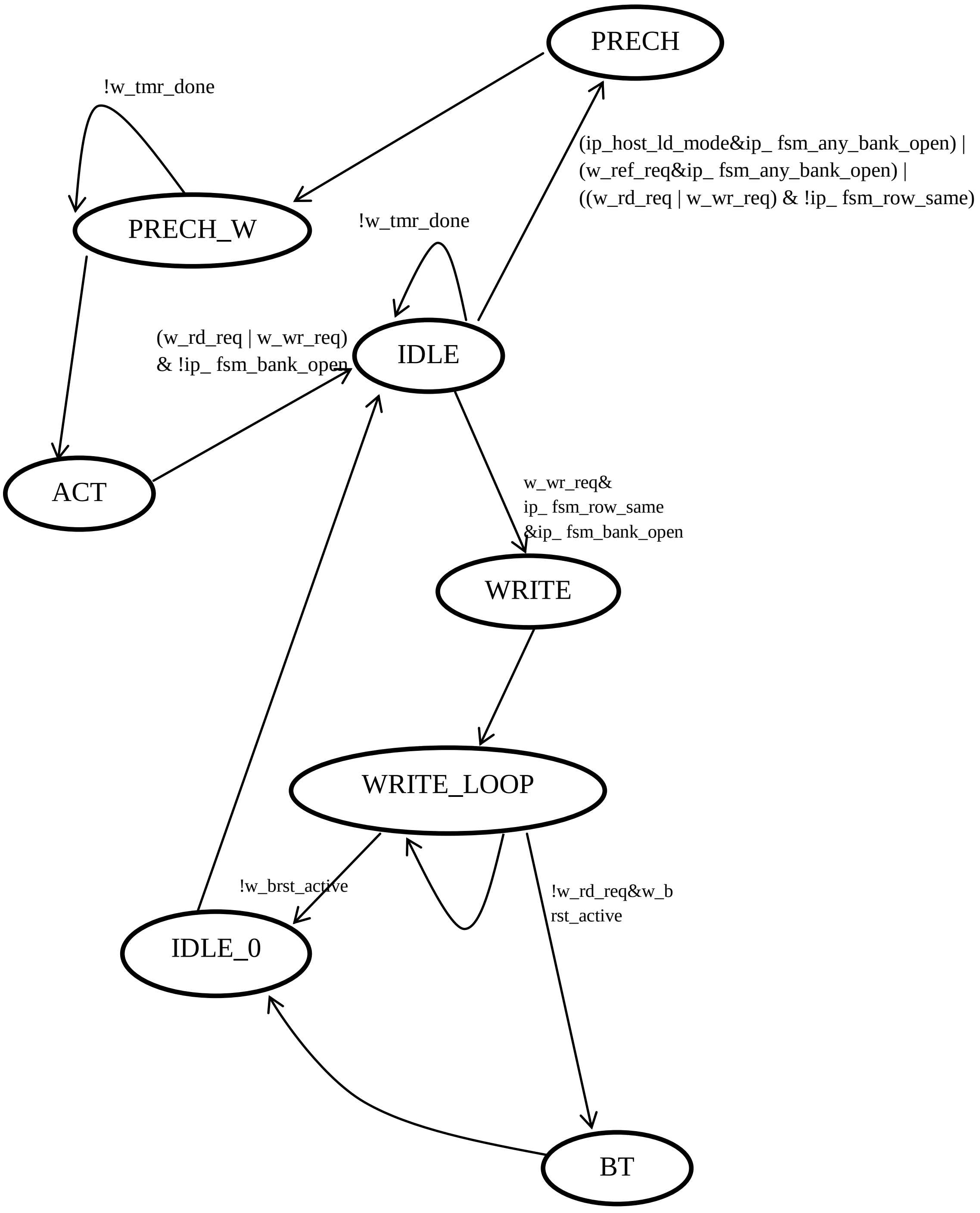


Figure 2.6.10: Write Protocol.

2.7: Problem in Existing Memory System

For the existing memory system, they are actually using physical address to access the information resides in either SDRAM or caches. For this design, it is only capable to work with a single user program. The problem arises when,

- ➢ Run multiple programs simultaneously.
    - o For example, when UserA start up a process and UserB also start a process, how are we going to manage both of the memory spaces required by both of the process to ensure they are not overlaying each others?

- ➢ Run a program in which its size is larger than SDRAM.
    - o For example, the size of main memory used in the memory system is 64MB, so how are going to start a process when the process required more than 64MB of memory?

      **Noted that all the process that is currently running need to be in main memory.*

To solve the problems, we can enlarge our main memory or the programmer needs to bear the responsibility to divide the program that they had written into few sections and transfer them into main memory. As the program proceeds, new sections will be added into main memory by replacing those sections that are currently unused. There is some disadvantage for both of solution which is

- ➢ Cost of enlarging main memory.

- ➢ As program become more and more complex, it is impossible for programmers to handle the division of the program.

2.8: Introduction of Virtual Memory

To solve the problems discussed in the previous session, the best solution is using a virtual memory which is a technique that used main memory, also called as physical memory to act as a "cache" for disk. As what we had been discussed earlier, the access time is increasing as going down from the memory hierarchy like what is illustrated by the figure below,

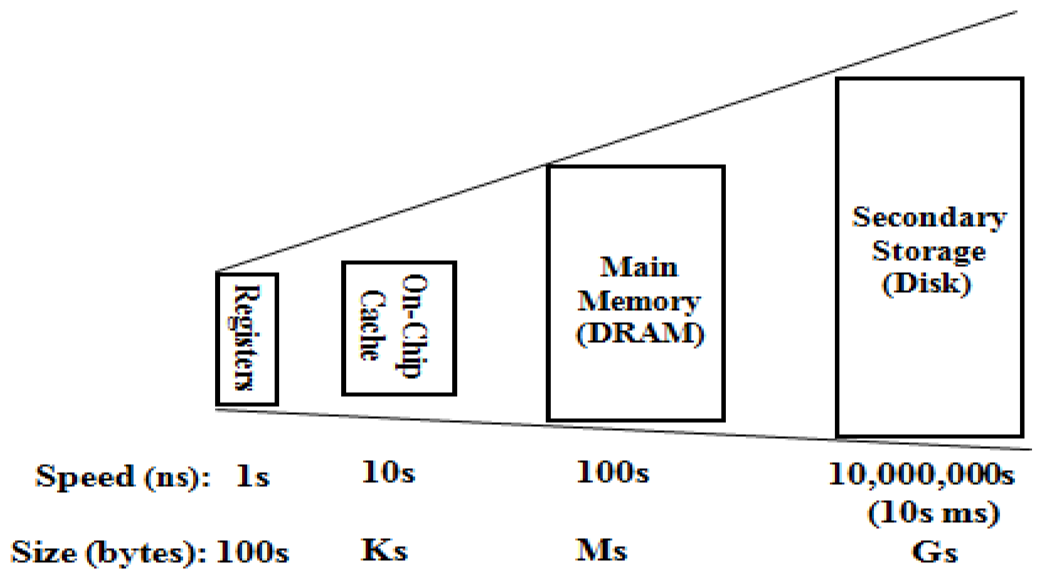


Figure 2.8.1:Access time and size of memory as going down from memory hierarchy.

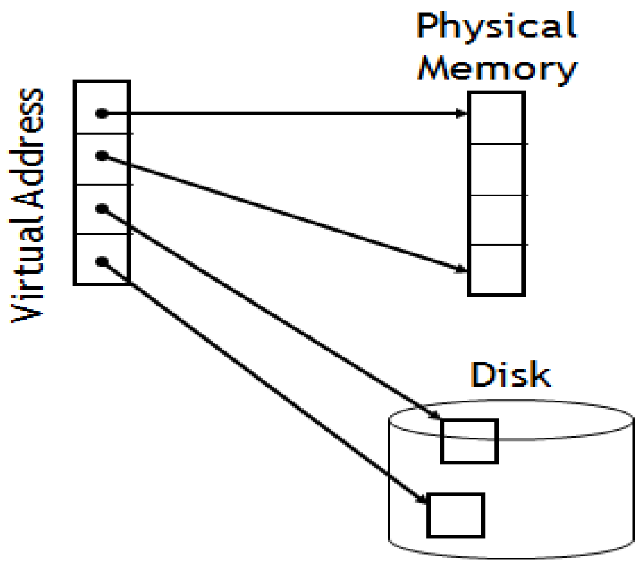


Figure 2.8.2: The basic concept of virtual memory.

Previously, as the size of physical memory grows, the access time is becoming slower and slower. Therefore, cache has been introduced to solve this problem which a portion of the data in main memory will be stored inside cache. Same theory we apply on the disk, we use the main memory to act as a cache for disk in order to speed up the processing speed. In this design, the address used will be virtual address and it need to go through address translation before it can be to access memory.

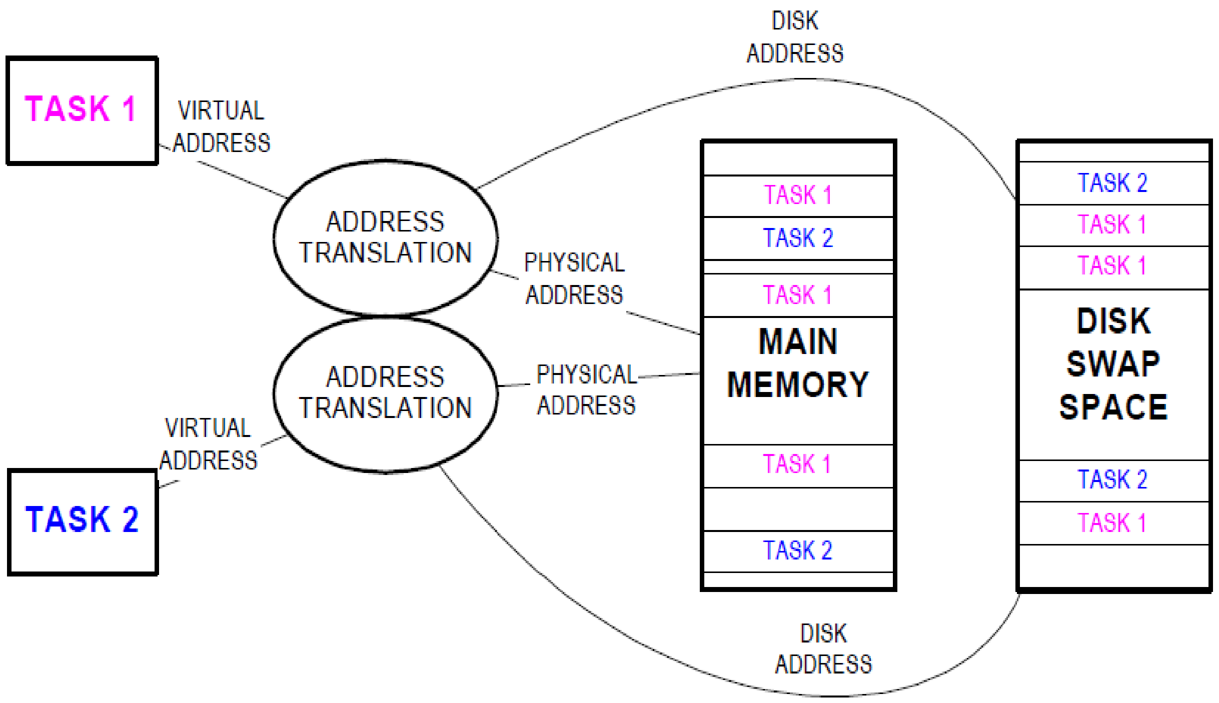
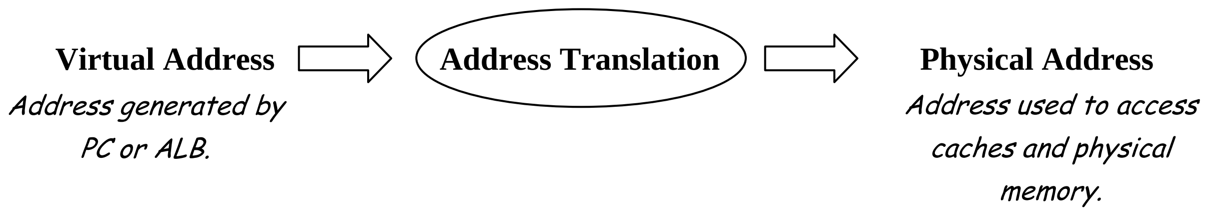


Figure 2.8.3: The overall picture of how virtual memory works.

2.9: Overview of Virtual Address Space

For main memory and caches access, both of them must receive a physical address in order to proceed. When we adopt the virtual memory, all of the address generated by the program counter will become a virtual address and translation of address need to be made in order to access physical memory and cache.


Virtual Address
Address generated by PC or ALB.
Address Translation
Physical Address
Address used to access caches and physical memory.

For virtual memory, the memory space is divided into a few segments as shown in figure below,

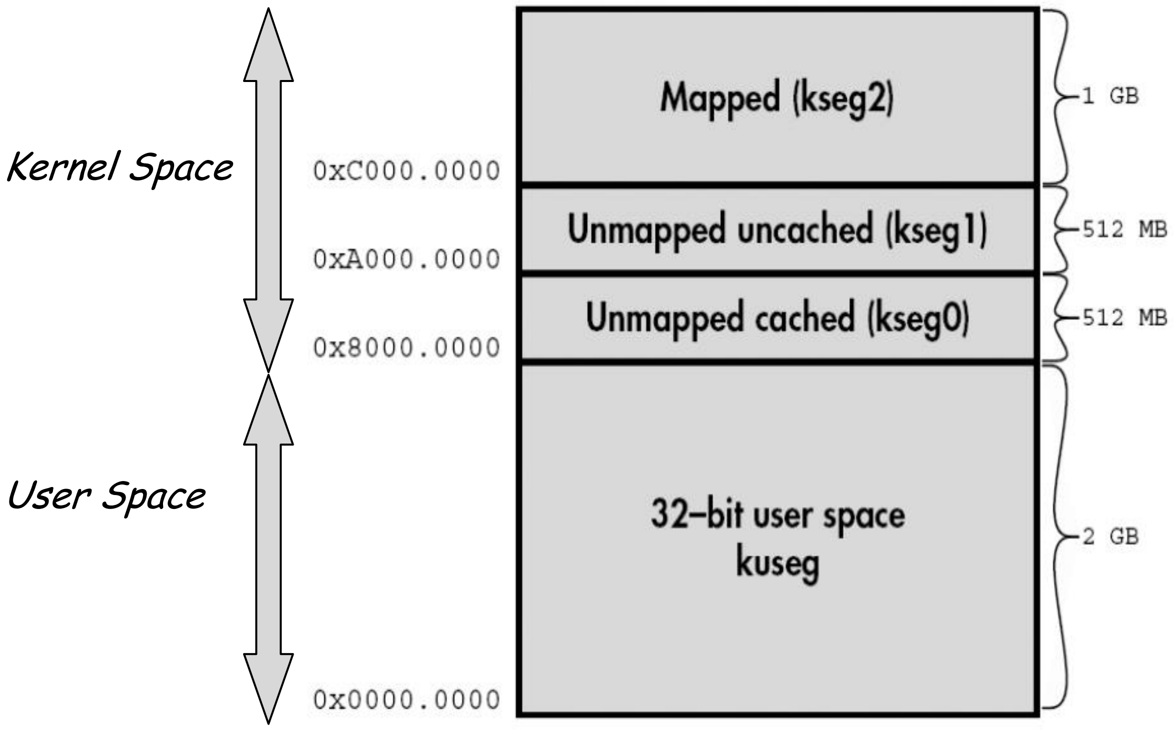


Figure 2.9.1: The virtual address space based on MIPS.

**Note that,*

- *kseg2 is mapped and cacheable. It is used for kernel data structures such as page table.*
- *kseg1 is unmapped and uncacheable. Access to this space doesn't go through Translation Lookaside Buffer, TLB. It is used for disk buffer, I/O register and ROM code.*
- *kseg0 is unmapped and cacheable. It is used for kernel instruction and data.*
- *kuseg is mapped and cacheable. It is used for current user process.*

2.10: Concept of Address Translation

Address Translation is a process which converts virtual address generated by CPU to physical address. Although the concept at work in virtual memory and in caches are the same, their different historical roots have led to different terminology in which the virtual memory block is called as a page while virtual memory miss is called as page fault.

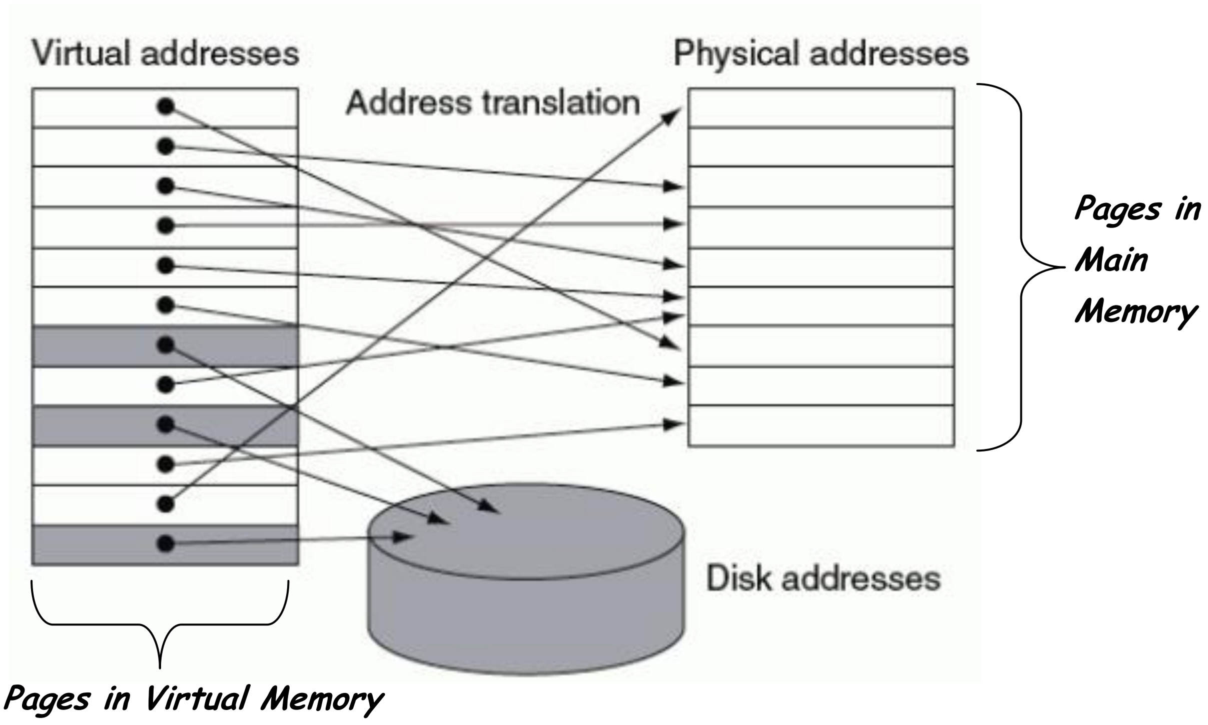


Figure 2.10.1: Address translation flow between pages in virtual memory and pages in main memory.

Based on the figure shown, we can actually notice that both virtual memory and physical memory are broken into pages so that the virtual page can exactly mapped to the physical page. As we all known, the size of virtual memory is actually larger than size of main memory. Therefore, it is possible for a page to be absent which means the virtual page is not mapping to a page inside physical memory, mapped instead on disk. It is possible for two virtual pages points to the same physical page and with this capability, it allows two different programs to share data or codes.

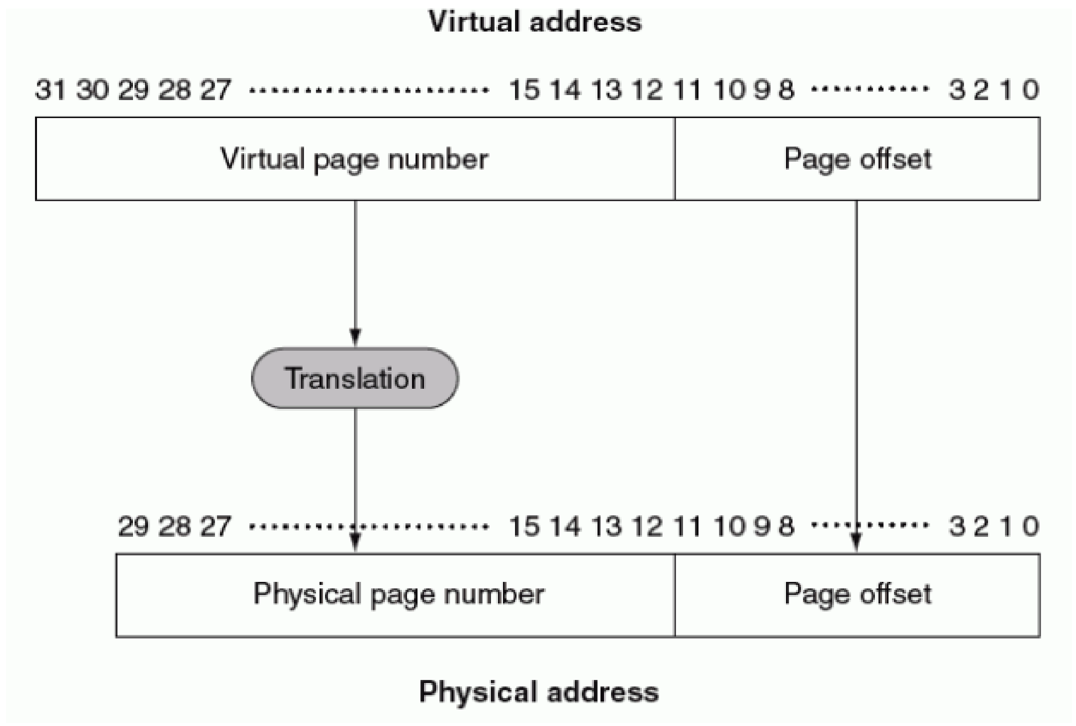


Figure 2.10.2: An example address translation mechanism.

**Note that,*

- *Virtual Page Number (VPN) is used to index a page table to find out appropriate Physical Page Number (PPN) for that particular virtual address.*

- *Page offset is representing the Page Size.*
  - *For example in this case,*

Number of bits used as page offset	=	*12 bits*
Page Size	=	*2 ^ 12*
	=	*4KB*

- *By observing the length of Physical Page Number, we can actually compute the size of main memory they are using which is*

Number of page in main memory	=	*2^18*	*= 256K physical page*
Page Size	=	*4KB*	
Size of main memory	=	*256K x 4KB*	
	=	*512MB*	

2.11: Introduction of Page Table

For the previous session, we keep on discuss about address translation, a process to convert virtual address to physical address but what is the procedure for the translation? In order to map VPN to PPN, page table, which is a table of entries contain the information required for the translation is used.

Valid	Physical Page Number

Figure 2.11.1: The contents of page table entry.

**Note that,*

- *Valid, is used to show the location of the page reside.*
    - *'1' indicate the page reside in physical memory.*
    - *'0' indicate the page reside in disk.*
- *Physical Page Number is a part of physical address to be output to concatenate with the page offset.*

By using page table, we can compute the physical address based on a given virtual address from kuseg. Below shows the example of how to do address translation using page table.

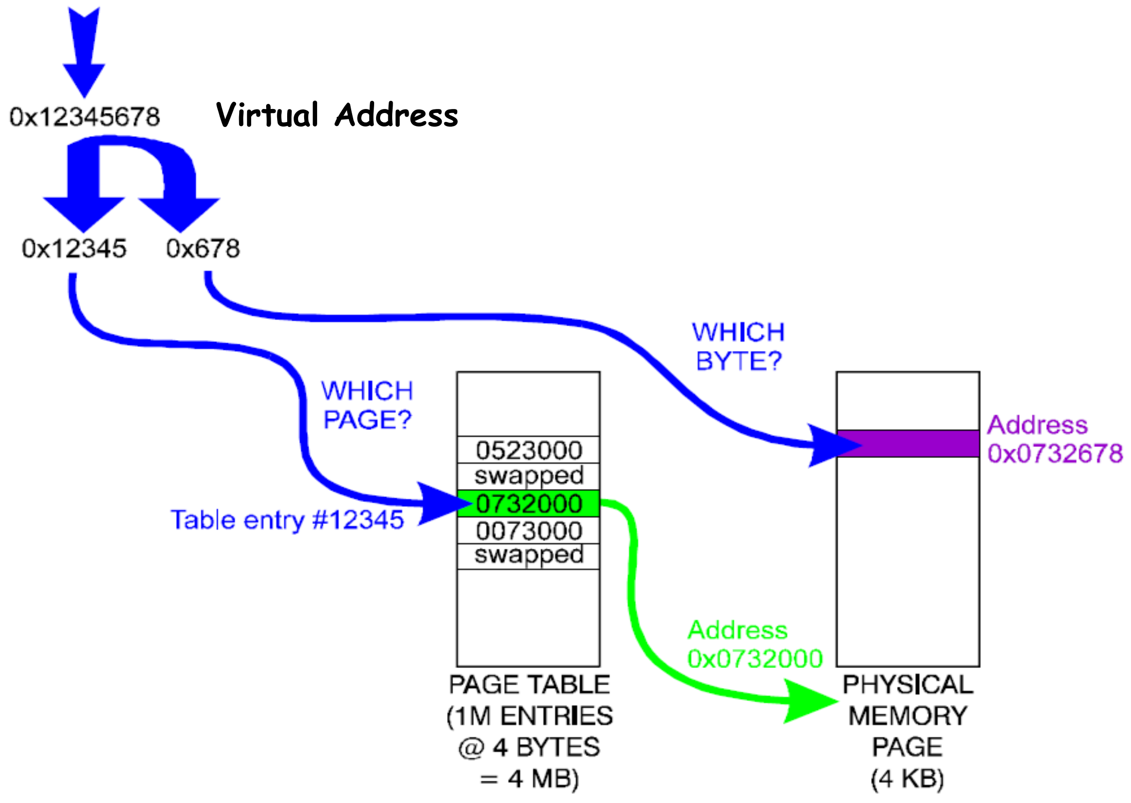


Figure 2.11.2: The usage of page table in address translation.

Inside the figure, since we are using just only one page table which all called as 1-level page table, therefore, the size of the page table will be

Number of entries in page table	=	2 ^20
	=	1M
Size of each entry in page table	=	4B
Max. Size of page table	=	4B x 1M
	=	4MB

That is waste of memory in which too much of spaces are wasted to build up a page table. Therefore, another technique is used to reduce the wastage of memory which called as 2-level page table. The concept of using 2-level page table is the first level of page table is will contain the page table entries as below

Valid	Page Table Base Register

Figure 2.11.3: The contents of first level page table entry.

**Note that,*

- *Valid, is used to show the location of the page reside.*
  - *'1' indicate the second level page table reside in physical memory.*
  - *'0' indicate the second level page table reside in disk.*
- *Page Table Base Register is a pointer to the second level page table.*

Valid	Physical Page Number

Figure 2.11.4: The contents second level page table entry.

**Note that,*

- *Valid, is used to show the location of the page reside.*
  - *'1' indicate the page reside in physical memory.*
  - *'0' indicate the page reside in disk.*
- *Physical Page Number is a part of physical address to be output to concatenate with the page offset.*

By using the 2-level page table technique, we need to segment out the virtual address into,

Virtual Page Number		
1st Level Page Table Index (10 bits)	2nd Level Page Table Index (10 bits)	**Page Offset** (12 bits)

Figure 2.11.5: Segmentation of virtual address.

**Note that,*

- *1st Level Page Table Index is used to locate the address of 2nd level Page Table.*
- *2nd Level Page Table Index is used to select the appropriate page table entries.*

By segmenting the virtual page number into 1st level page table index and 2nd level page table index, we will be able to locate desired page table entries as below,

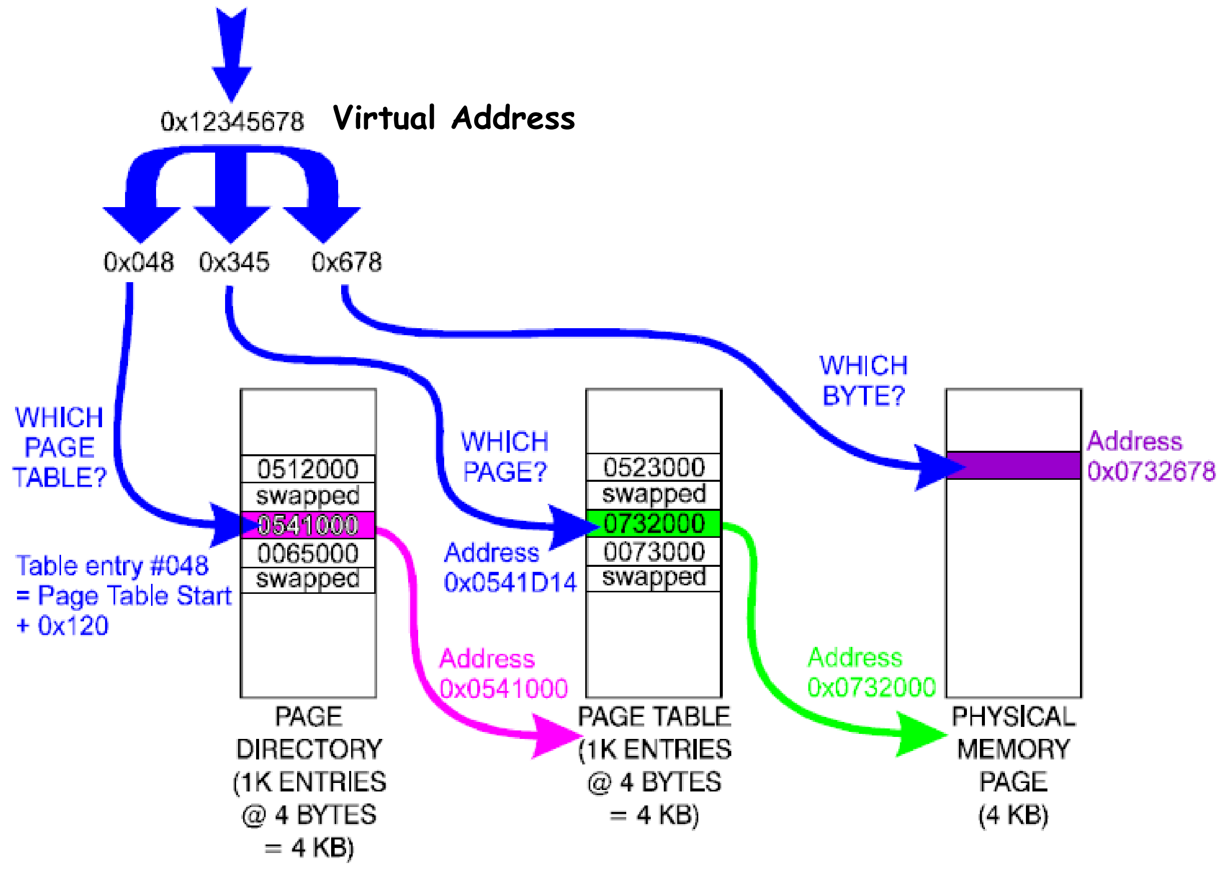


Figure 2.11.6: The usage of 2-level page table in address translation.

As shown in figure, the size of page table has been increase significantly compare with the 1-level page table. A more detail calculation shown as below

Number of entries in 1st level page table	=	2 ^10
	=	1K
Number of entries in 2nd level page table	=	2 ^10
	=	1K
Size of each entry in page table	=	4B
Size of each level page table	=	4B x 1K
	=	4KB
Total size of page table	=	4KB + 4KB
	=	8KB

Previously if we are using the 1-level page table, we need to allocate 4MB space for the page table for each of the process. On the other hand, when we are using 2-level page table, we just need to allocate 8KB space for page table and the page table can be created based on demand. Besides, by using this mechanism, the size of page table will be uniform with the page size whether in virtual memory or physical memory.

2.12: Introduction of Translation Lookaside Buffer

For previous sessions, we had discussed how to use a page table to allocate pages that reside in the physical memory. By using the 2-level page table, although we can save the memory spaces that required to store the page table, the access time in order to get the physical pages is becoming longer compare with 1-level page table.

1-level Page Table

i. *Given a virtual address.*
ii. *Use VPN to find out the PPN which used to concatenate with the page offset to form physical address.*
iii. *Use physical address get data for physical memory.*

2-level Page Table

i. *Given a virtual address.*
ii. *Use 1st level page table index to allocate the address of 2nd level page table.*
iii. *Use 2nd level page table index to find out the PPN which is used to concatenate with the page offset to form physical address.*
iv. *Use physical address to get data from physical memory.*

Based on both of the scenario discussed above, we can notice that 2-level page table need one more access to the physical memory compare with the 1-level page level. As we are increasing the level of page table, although the size of page table required for each process will decrease, the number of access to physical memory will increase. This is very inefficient and therefore, Translation Lookaside Buffer (TLB) is used to solve this problem.

The key to improving the performance is to rely the locality of reference to the page table. When a translation for a virtual page number is used, it will probably be needed again in the near future. With this concept, TLB has been introduced which is a special cache for translation that whole part of the page table entries in order to speed up the address translation. In order to enable a faster access table, TLB usually only contain very less entries which is around 48-128 entries and due to this, TLB usually be implemented as a fully associative cache which all of the entries inside TLB will be compare in one shot. This will result in a faster searching speed but it may require a lot of hardware support in order to build it.

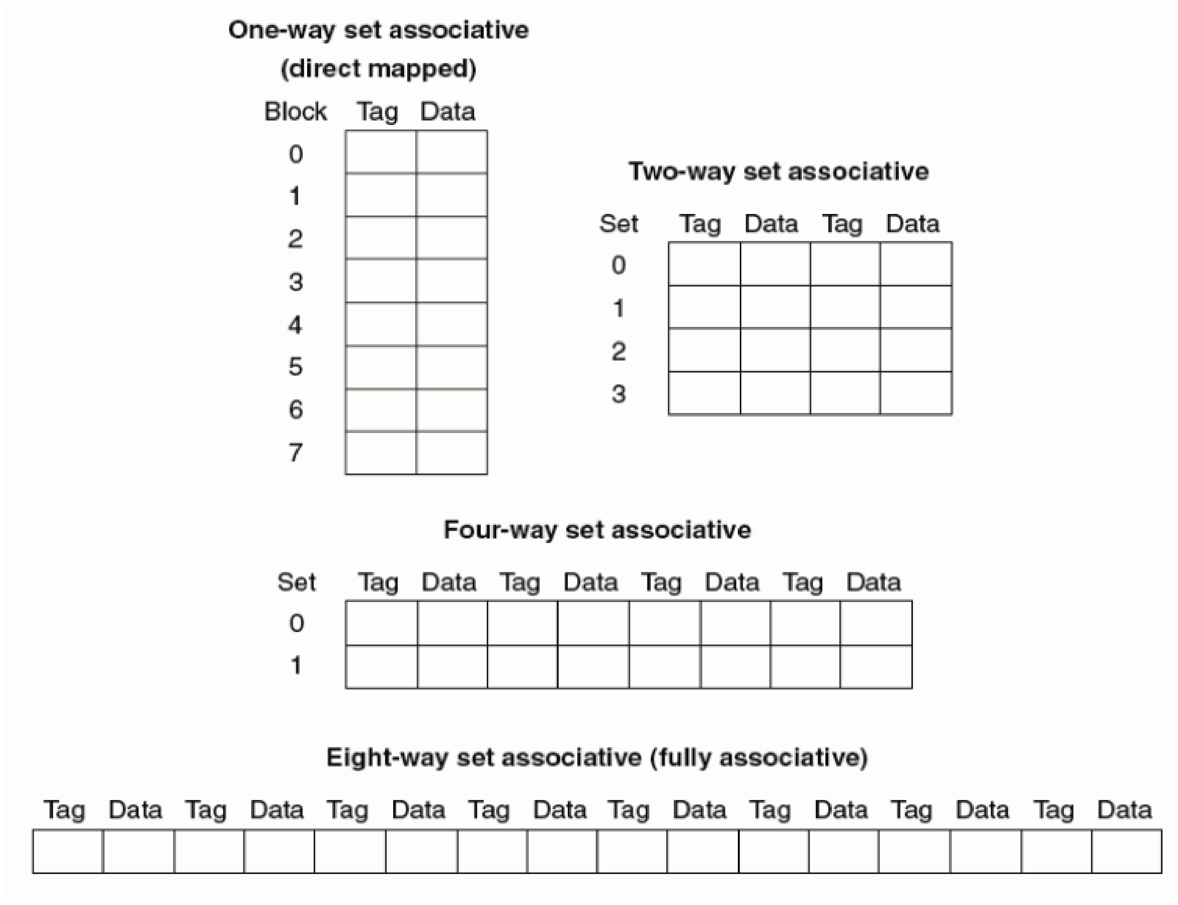


Figure 2.12.1:Example of how an eight-block cache configure as direct mapped, two-way set associative, four-way set associative and fully associative cache.

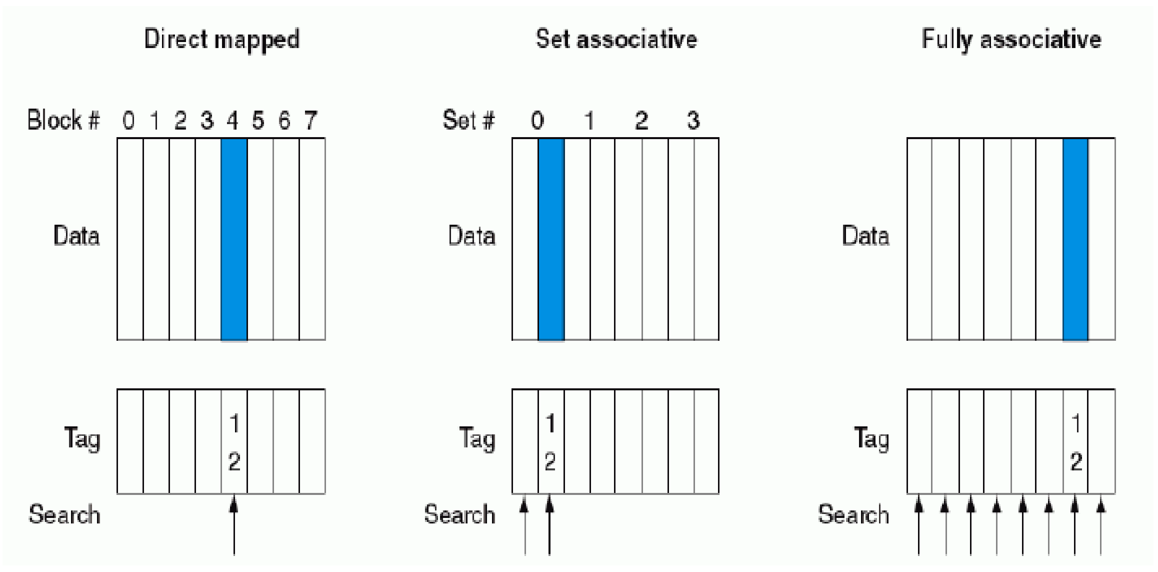


Figure 2.12.2: Example of how a searching works on eight-block cache based on direct mapped, two-way set associative and fully associative configuration.

Now, for us to start implementing TLB, the first thing we need to do is identify the contents of each entry in TLB. For a basic TLB, we must have VPN, PPN and also some control bits used to indicate the status of each entry such as, valid bit, dirty bit and so on based on the design needs.

Virtual Page Number (20 bits)	Control Bits	Physical Page Number (20 bits)

Figure 2.12.3: The contents TLB entry.

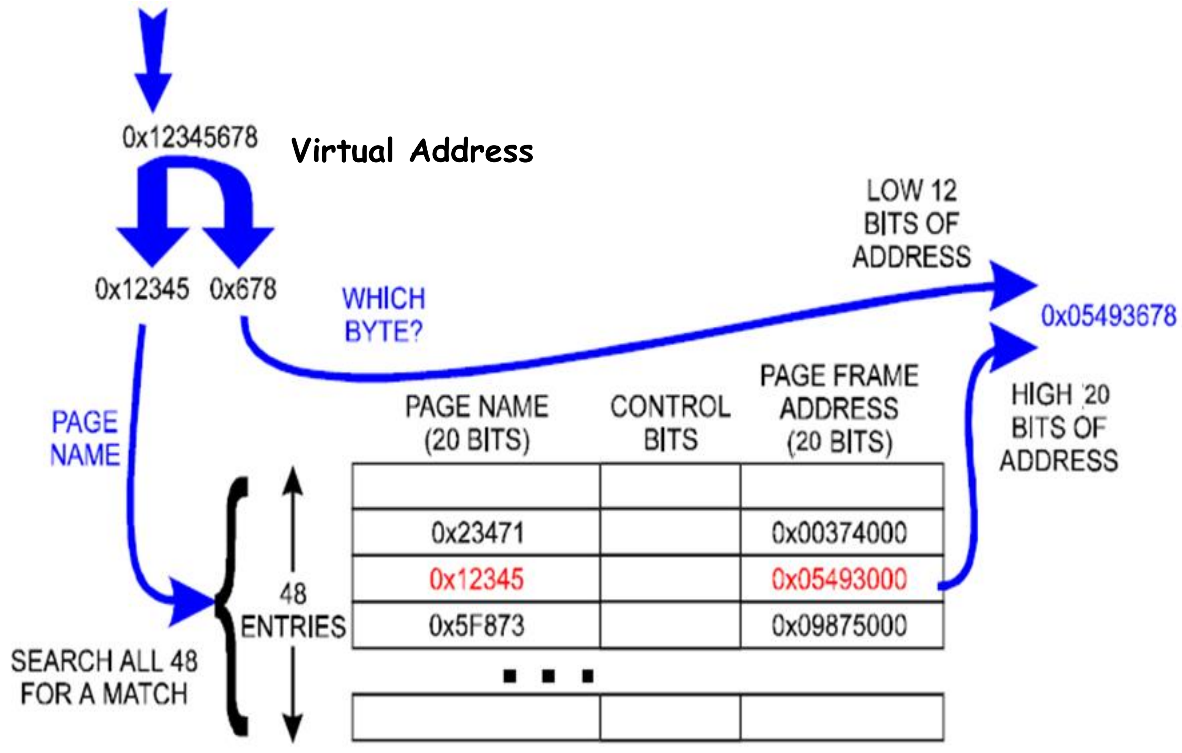


Figure 2.12.4: Usage of TLB in address translation by using 48 entries and fully associative TLB.

**Note that,*

- *VPN is included inside as part of the TLB entry contents which is different from the page table entry.*
- *VPN doesn't segment into 1st page table index and 2nd page table index. This is because when we are using TLB, it is containing the information in 2nd level page table only.*
- *Control bits can be any bits which used to represent the status of each entry based on the design needs.*
    - *Example of control bits will be*
        - *Valid Bit, which used to represent the location of the page whether in physical memory or disk.*
        - *Dirty bit, which used to represent whether the entry has been modified or not. Usually used for write back policy in cache.*
        - *Ref bit, which is a LRU status where the entry with the smallest ref will be replace when the CPU going to bring in a new page from disk or physical memory.*

2.13: Virtually Addressed and Physically Addressed Cache

The placement of TLB can be either in series with caches or parallel with caches. Both of the design have their pros and con. When we set the TLB in front of the cache, this will mean that all of the address need to be translates into physical address before access into cache. By using this design, the processing speed will be reduced because we need to access to TLB first then only can access cache which means we need to times two the access time to a cache. Although the processing speed will be reduce, this method will be much simpler compare with a virtually addressed cache which will be discussed later.

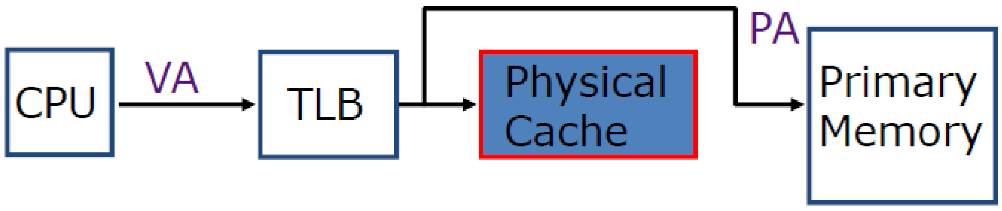


Figure 2.13.1: The design of physically addressed cache.

**Note that all of the virtual addresses have to be translated by TLB before accessing cache or main memory.*

There is another design of the placement of TLB which is the TLB works parallel with the caches. This will reduce the processing time because the address translation and the data searching can be done in parallel. Although this method can enhance the efficiency of the processor, the design is more complex compare with physically address cache because the lower 12 bits, page offset is used to search the data in cache and the tag inside cache entries is output from the cache to compare with the PFN output from TLB to determine whether it is a cache hit or miss. Problem arises when we have two cache entries with the same page offset, which will cause an aliasing effect. Therefore, additional logic needs to be added to eliminate this problem.

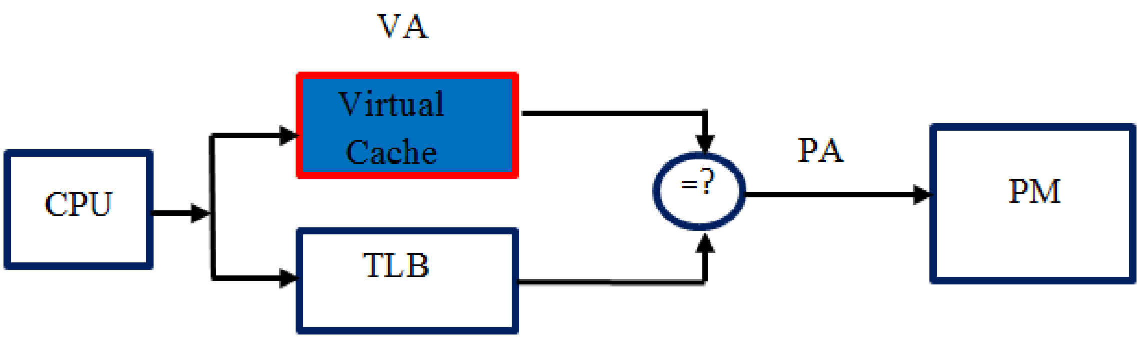


Figure 2.13.2:The design of virtually addressed cache.

**Note that the virtual address output from CPU is directly input to cache and TLB.*

Chapter 3 – Methodology& Development tools

3.1: Methodology

A top down design approach was adopted as the main design methodology in this project. In this project, more focuses were put onto the functionality of the design. In the earlier phase of the project, a study was done on the performance analysis and the behavioral correctness of the previous memory system. However, from the analysis, we have found out the need to build a new Memory System Bus Interface Unitin order to integrate the current memory system to the basic CPU.This requires us to implement the system by using top down methodology.

In the top down methodology, the first step involves the gathering of the requirements of the SDRAM and SDRAM Controller. The requirements gathered will be analyzed and studied so that a specification can be created. This specification describes the input/outputs, registers, functions, and the constraints of the design. The requirements can be obtained from users, market demands and datasheets. In this project, the requirements are mainly defined from the SDRAM datasheets [12].

The reason for this is to ensure that the integration of memory system to32 bits RISC pipelined processor can be successfully done. Besides, studies were done on the ways to maximize the utilization of the 4 banks in the SDRAM. These studies were elaborated Chapter 2 literature review.

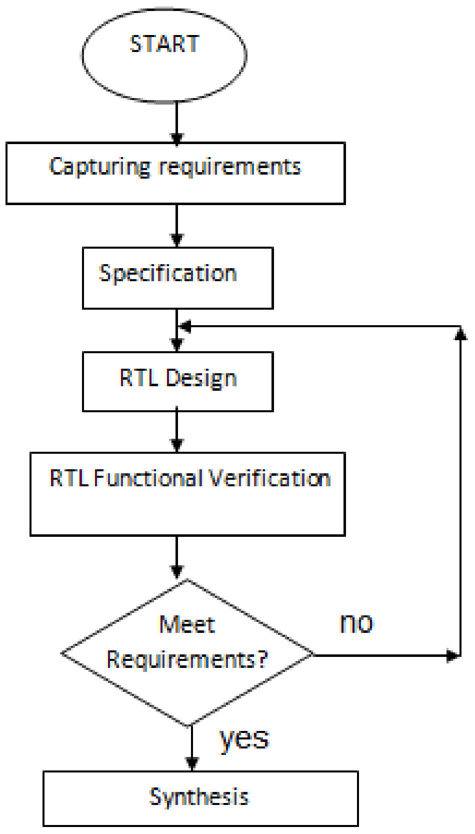


Figure 3.1.1: The top down approach adopted in this project

After capturing the requirements of the design, a specification is build. This specification specifies the functions of all the modules, data flows between input pins, output pins, registers and such. Basically, it is a detailed description of the design in Register Transfer Level (RTL). Logic is described in terms of data flow and algorithms.

From the requirements, RTL codes are written. These codes are then simulated to verify their functionality up to clock cycle accuracy. Sub-blocks that don't perform as specified are to be debugged and have their RTL codes fixed the requirements are met.

After the main task of defining the functionality is completed, the design will synthesize into gate-level representation. Design synthesis is outside the scope of this project thus will not be pursued.

3.2: Development Tools

ModelSim XE 3 – Starter 6.4b will be used to code the RTL model of the design. Besides, it will also be used to carry out the functional and timing simulation. ModelSim provides an user friendly debug environments. Graphical waveform to display the simulation results is integrated into ModelSim.

The starter edition placed a 10000 lines limit to the code. Based on the scope of this project, it is expected that this limit will not be reached. Besides, it is free thus being chosen as the main tool for this project.

Chapter 4: Handling Virtual Memory

4.1: Address Translation to Instruction Cache without TLB

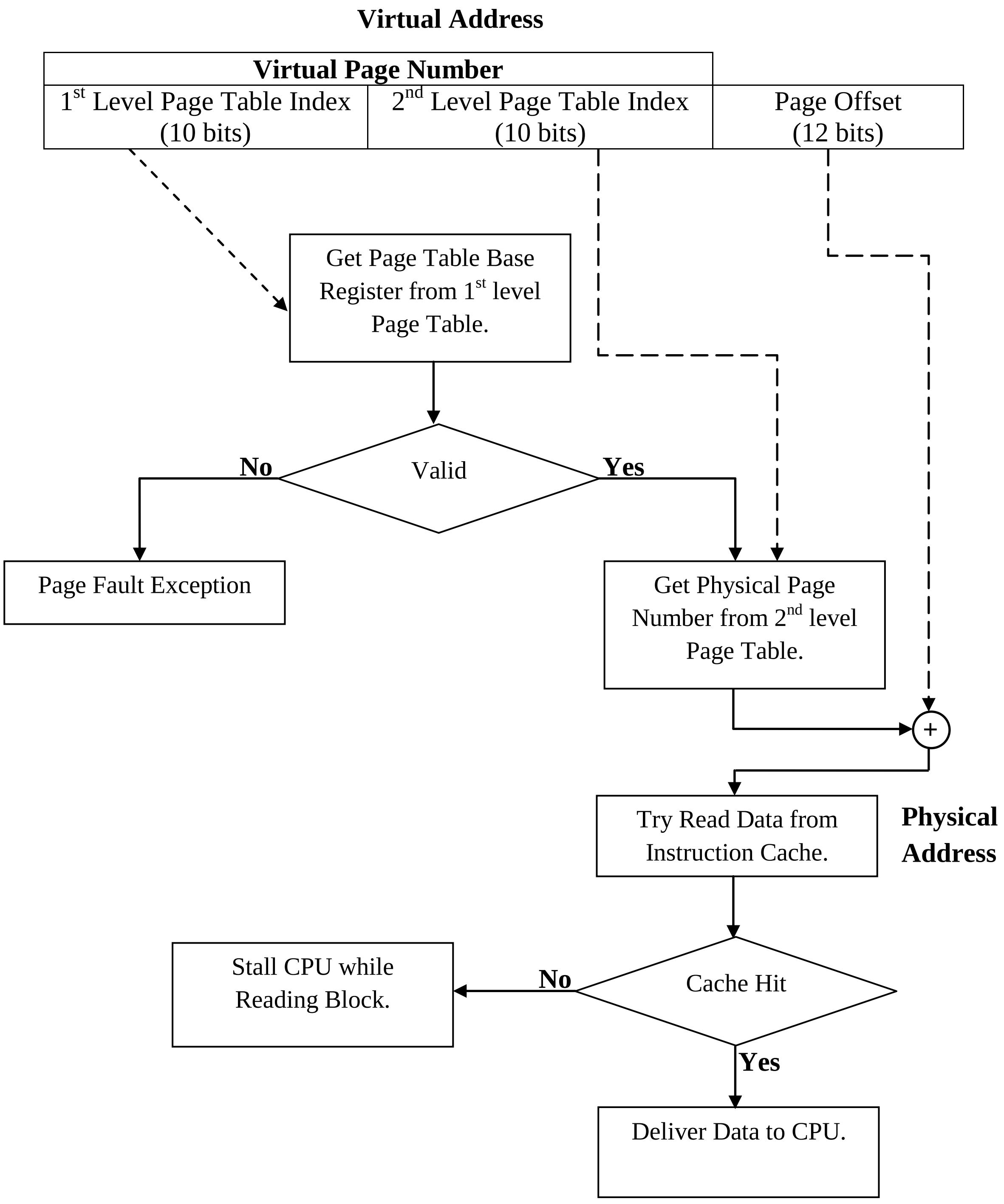

Virtual Address
Virtual Page Number
1st Level Page Table Index (10 bits)
2nd Level Page Table Index (10 bits)
Page Offset (12 bits)
Get Page Table Base Register from 1st level Page Table.
Valid
No
Yes
Page Fault Exception
Get Physical Page Number from 2nd level Page Table.
+
Try Read Data from Instruction Cache.
Physical Address
Cache Hit
No
Stall CPU while Reading Block.
Yes
Deliver Data to CPU.

Figure 4.1.1:Address Translation to Instruction Cache without TLB.

4.2: Address Translation to Data Cache without TLB

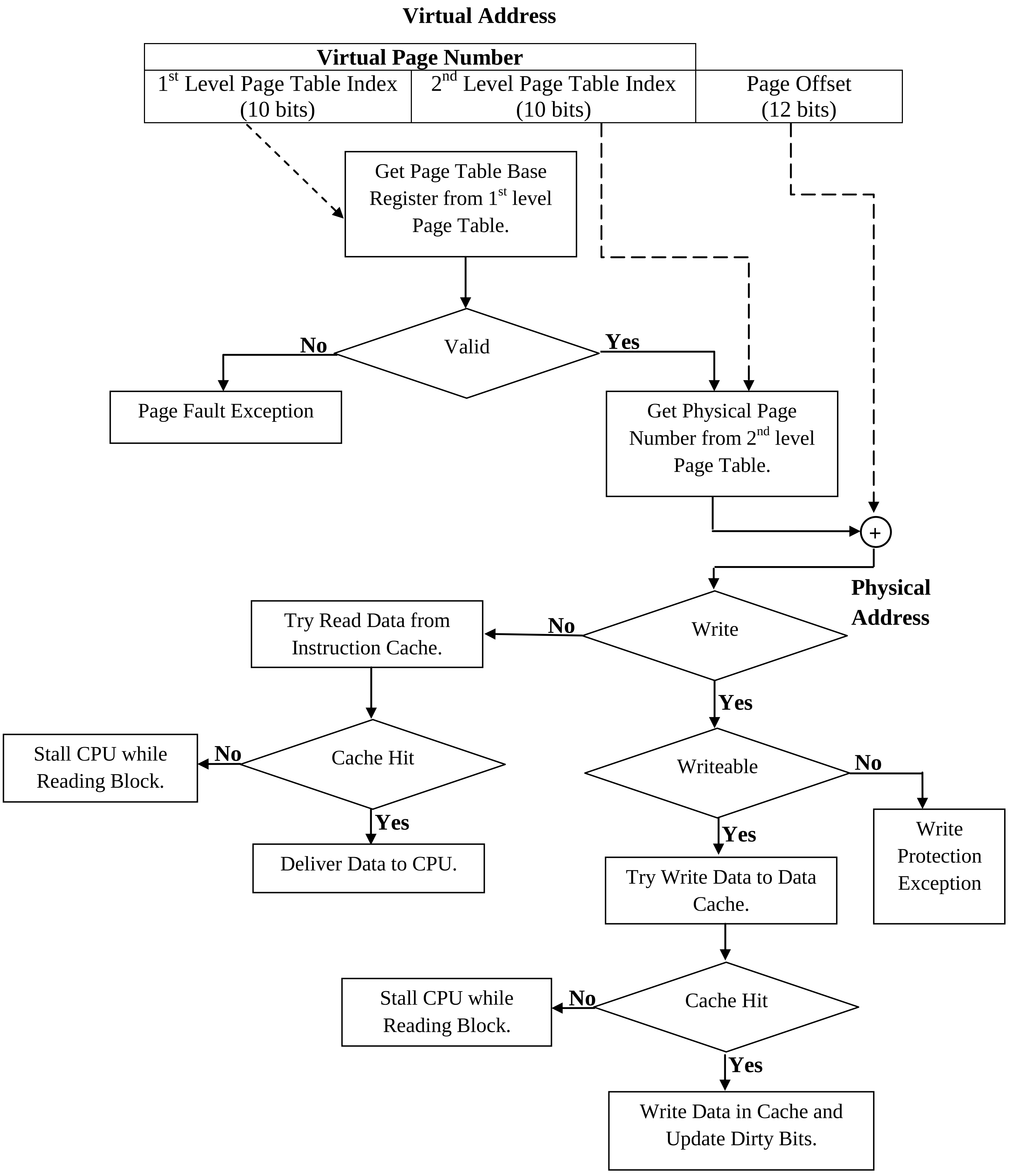

Virtual Address
Virtual Page Number
1st Level Page Table Index (10 bits)
2nd Level Page Table Index (10 bits)
Page Offset (12 bits)
Get Page Table Base Register from 1st level Page Table.
Valid
No
Yes
Page Fault Exception
Get Physical Page Number from 2nd level Page Table.
+
Physical Address
Write
No
Try Read Data from Instruction Cache.
Yes
Cache Hit
No
Stall CPU while Reading Block.
Yes
Deliver Data to CPU.
Writeable
No
Write Protection Exception
Yes
Try Write Data to Data Cache.
Cache Hit
No
Stall CPU while Reading Block.
Yes
Write Data in Cache and Update Dirty Bits.

Figure 4.2.1:Address Translation to Data Cache without TLB.

4.3: Address Translation to Instruction Cache with TLB

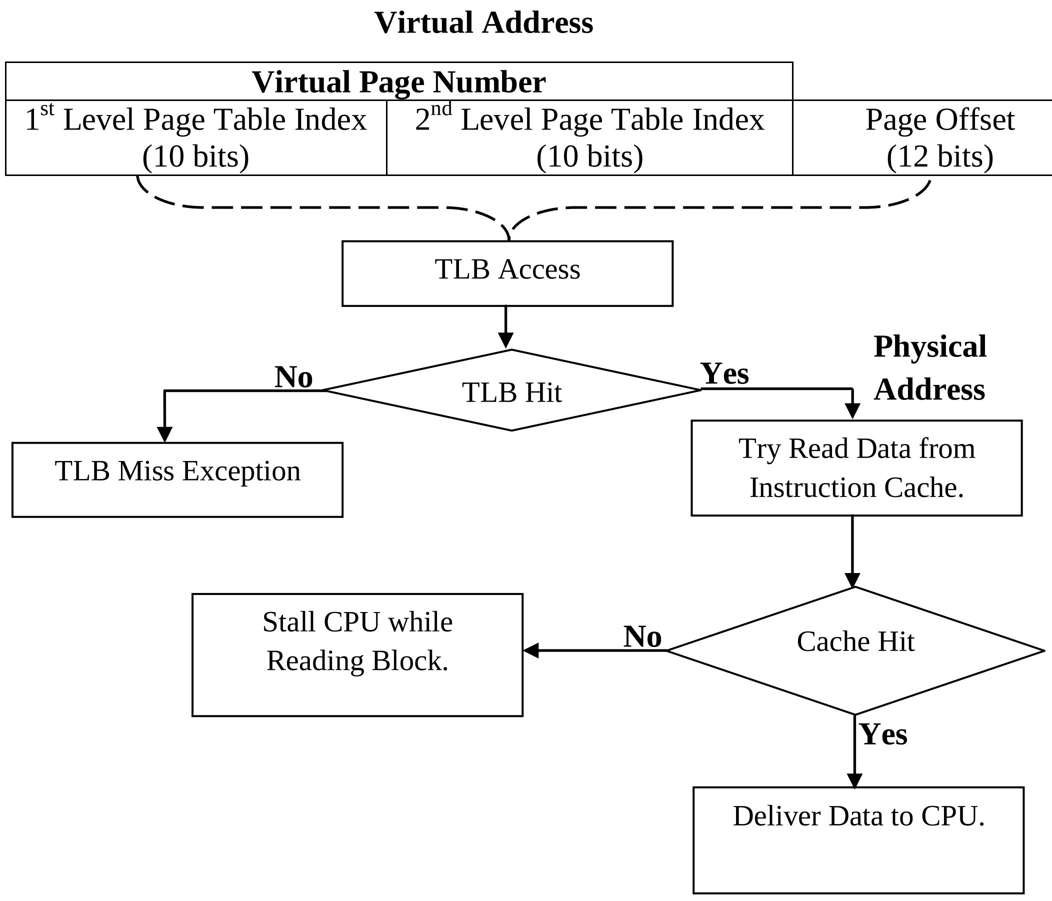


Figure 4.3.1: Address Translation to Instruction Cache with TLB.

4.4: Address Translation to Instruction Cache with TLB

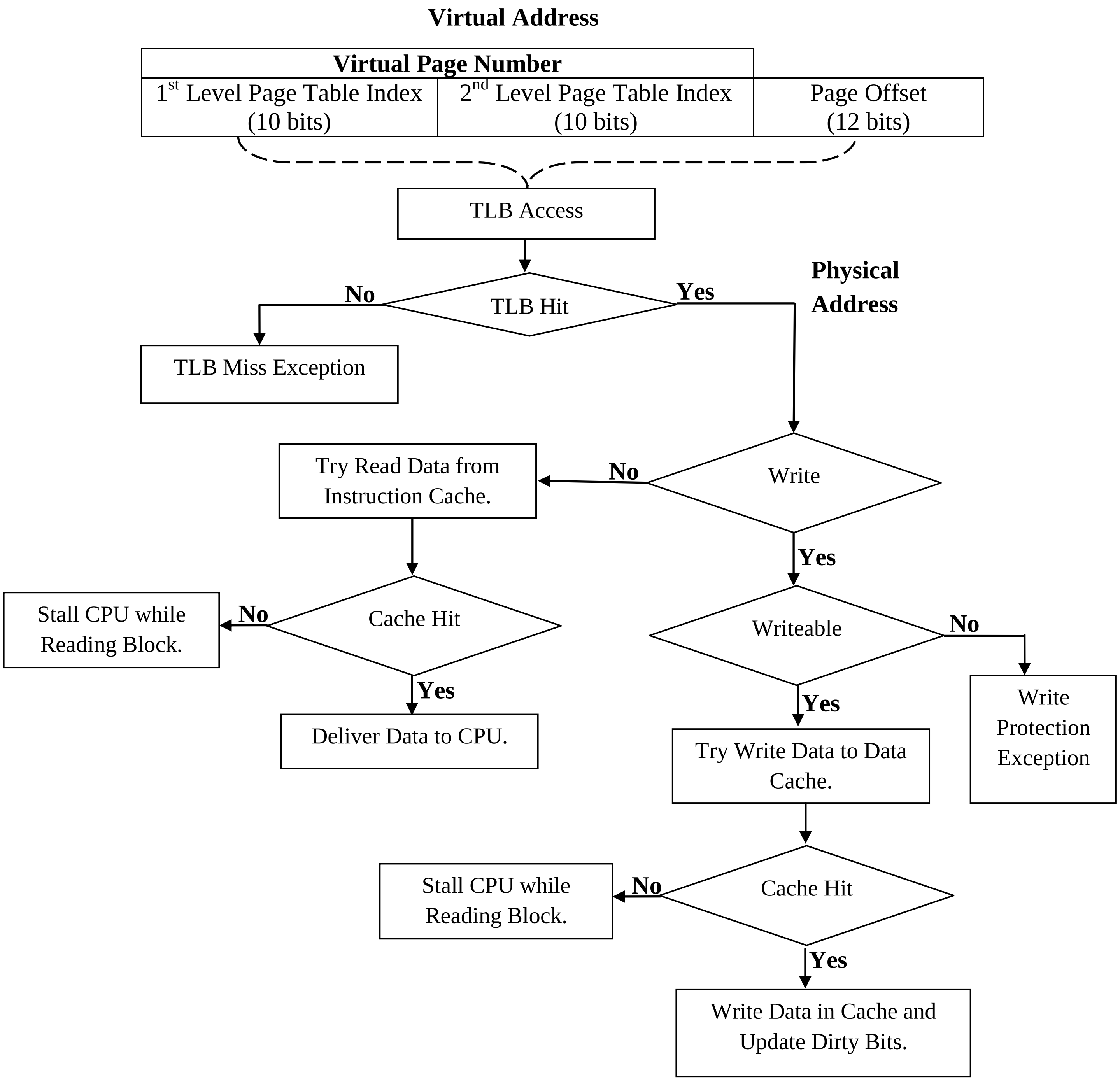

Virtual Address
Virtual Page Number
1st Level Page Table Index (10 bits)
2nd Level Page Table Index (10 bits)
Page Offset (12 bits)
TLB Access
TLB Hit
No
Yes
Physical Address
TLB Miss Exception
Write
No
Yes
Try Read Data from Instruction Cache.
Cache Hit
No
Yes
Stall CPU while Reading Block.
Deliver Data to CPU.
Writeable
No
Yes
Write Protection Exception
Try Write Data to Data Cache.
Cache Hit
No
Yes
Stall CPU while Reading Block.
Write Data in Cache and Update Dirty Bits.

Figure 4.4.1: Address Translation to Data Cache with TLB.

Chapter 5: Memory System Specification

5.1: Features of Memory System

	RISC32 with Integrated Main Memory
SDRAM	***Yes, 64MB***
Instruction TLB	***Yes, 64 entries***
Data TLB	***Yes, 64 entries***
Instruction Cache	***2MB***
Data Cache	***2MB***
Data Bus Width	32bits
Instruction Width	32bits

Table 5.1.1: The features of recent RISC32.

5.2: Naming Convention

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

Abbreviation:

	Description	Case	Available	Remark
lvl	level	lower	c : Chip u : Unit b : Block	
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 5.2.1: Naming convention.

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:The memory map used in this project.

**Note that,*

- ***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: Memory Unit Interface

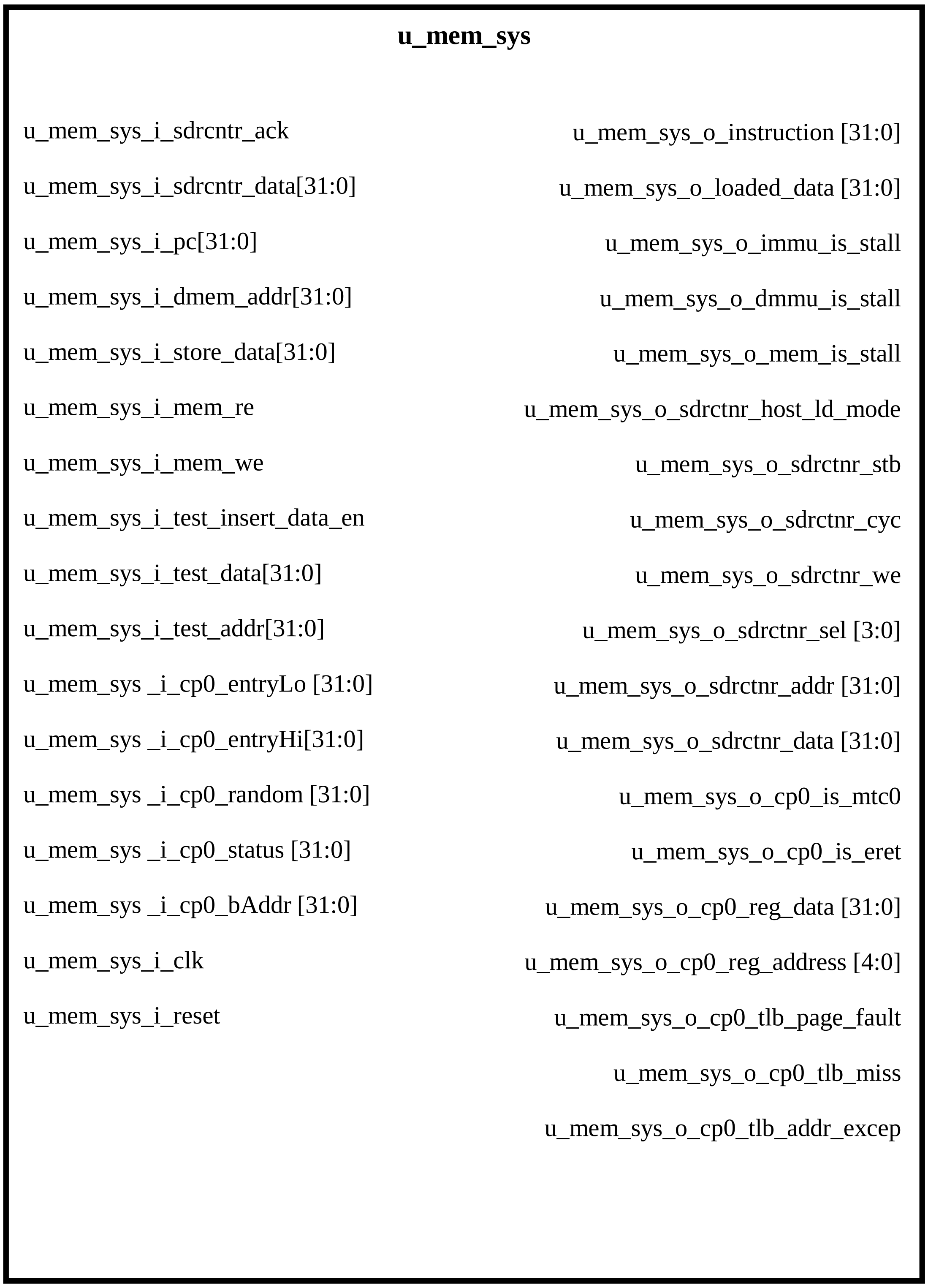

u_mem_sys
u_mem_sys_i_sdrcntr_ack
u_mem_sys_i_sdrcntr_data[31:0]
u_mem_sys_i_pc[31:0]
u_mem_sys_i_dmem_addr[31:0]
u_mem_sys_i_store_data[31:0]
u_mem_sys_i_mem_re
u_mem_sys_i_mem_we
u_mem_sys_i_test_insert_data_en
u_mem_sys_i_test_data[31:0]
u_mem_sys_i_test_addr[31:0]
u_mem_sys _i_cp0_entryLo [31:0]
u_mem_sys _i_cp0_entryHi[31:0]
u_mem_sys _i_cp0_random [31:0]
u_mem_sys _i_cp0_status [31:0]
u_mem_sys _i_cp0_bAddr [31:0]
u_mem_sys_i_clk
u_mem_sys_i_reset
u_mem_sys_o_instruction [31:0]
u_mem_sys_o_loaded_data [31:0]
u_mem_sys_o_immu_is_stall
u_mem_sys_o_dmmu_is_stall
u_mem_sys_o_mem_is_stall
u_mem_sys_o_sdrctnr_host_ld_mode
u_mem_sys_o_sdrctnr_stb
u_mem_sys_o_sdrctnr_cyc
u_mem_sys_o_sdrctnr_we
u_mem_sys_o_sdrctnr_sel [3:0]
u_mem_sys_o_sdrctnr_addr [31:0]
u_mem_sys_o_sdrctnr_data [31:0]
u_mem_sys_o_cp0_is_mtc0
u_mem_sys_o_cp0_is_eret
u_mem_sys_o_cp0_reg_data [31:0]
u_mem_sys_o_cp0_reg_address [4:0]
u_mem_sys_o_cp0_tlb_page_fault
u_mem_sys_o_cp0_tlb_miss
u_mem_sys_o_cp0_tlb_addr_excep

Figure 5.4.1:The block diagram of memory system.

I/O Description

Memory System's Input Pin Description

Pin Name: u_mem_sys_i_sdrcntr_ack	Source → Destination: SDRAM CNTR→ Memory System	Registered: No
Pin Function: Acknowledge signal to indicate read or write to SDRAM is done.		
Pin Name: u_mem_sys_i_sdrcntr_data [31:0]	Source → Destination: SDRAM CNTR→ Memory System	Registered: No
Pin Function: 32 bit data read from SDRAM.		
Pin Name: u_mem_sys_i_pc [31:0]	Source → Destination: Data Path Unit→Memory System	Registered: No
Pin Function: 32 bits virtual address from program counter.		
Pin Name: u_mem_sys_i_dmem_addr [31:0]	Source → Destination: Data Path Unit→Memory System	Registered: No
Pin Function: 32 bits virtual address from ALB.		
Pin Name: u_mem_sys_i_store_data [31:0]	Source → Destination: Data Path Unit→Memory System	Registered: No
Pin Function: 32 bits data to be store in data cache or SDRAM.		
Pin Name: mem_sys_i_mem_re	Source → Destination: Data Path Unit→ Memory System	Registered: No
Pin Function: Data cache read control signal. 0: Read Disable 1: Read Enable		
Pin Name: u_mem_sys_i_mem_we	Source → Destination: Data Path Unit→ Memory System	Registered: No
Pin Function: Data cache write control signal. 0: Write Disable 1: Write Enable		
Pin Name: u_mem_sys_i_test_insert_data_en	Source → Destination: External→ Memory System	Registered: No
Pin Function:		

Control signal to allow data input into SDRAM manually. 0: Data input Disable. 1: Data input Enable.		
Pin Name: u_mem_sys_i_test_data [31:0]	Source → Destination: External→ Memory System	Registered: No
Pin Function: 32 bits TEST data to be write into SDRAM.		
Pin Name: u_mem_sys_i_test_addr [31:0]	Source → Destination: SDRAM CNTR→ Memory System	Registered: No
Pin Function: 32 bits TEST address to indicate location to store TEST data.		
Pin Name: u_mem_sys _i_cp0_entryLo [31:0]	Source → Destination: CP0 →Memory System	Registered: No
Pin Function: 32 bits EntryLo register from CP0.		
Pin Name: u_mem_sys _i_cp0_entryHi[31:0]	Source → Destination: CP0 →Memory System	Registered: No
Pin Function: 32 bits EntryHi register from CP0.		
Pin Name: u_mem_sys _i_cp0_random [31:0]	Source → Destination: CP0 → Memory System	Registered: No
Pin Function: 32 bits Random register from CP0.		
Pin Name: u_mem_sys _i_cp0_status [31:0]	Source → Destination: CP0 → Memory System	Registered: No
Pin Function: 32 bits Status register from CP0.		
Pin Name: u_mem_sys _i_cp0_bAddr [31:0]	Source → Destination: CP0 → Memory System	Registered: No
Pin Function: 32 bits bAddr register from CP0.		

Pin Name: u_mem_sys_i_clk	Source → Destination: System Clock→Memory System	Registered: No
Pin Function: System clock signal.		
Pin Name: u_mem_sys_i_reset	Source → Destination: System Reset→Memory System	Registered: No
Pin Function: System reset signal.		

Table 5.4.2: Memory System's Input Pin Description

Memory System's Output Pin Description

Pin Name: u_mem_sys_o_instruction [31:0]	Source → Destination: Memory System→ Data Path Unit	Registered: No
Pin Function: 32 bits instruction read from instruction cache.		
Pin Name:u_mem_sys_o_loaded_data [31:0]	Source → Destination: Memory System→ Data Path Unit	Registered: No
Pin Function: 32 bit data read from data cache.		
Pin Name: u_mem_sys_o_immu_is_stall	Source → Destination: Memory System→ Control Unit	Registered: No
Pin Function: Stall signal for CPU when ITLB miss. 0: Stall Disable 1: Stall Enable		
Pin Name: u_mem_sys_o_dmmu_is_stall	Source → Destination: Memory System→ Control Unit	Registered: No
Pin Function: Stall signal for CPU when DTLB miss. 0: Stall Disable 1: Stall Enable		

Pin Name: u_mem_sys_o_mem_is_stall	Source → Destination: Memory System→ Control Unit	Registered: No
Pin Function: Stall signal for CPU when icache and dcache miss. 0: Stall Disable 1: Stall Enable		
Pin Name: u_mem_sys_o_sdrctnr_host_ld_mode	Source → Destination: Memory System → SDRAM CNTR	Registered: No
Pin Function: Asserted to load a new mode into the SDRAM.		
Pin Name: u_mem_sys_o_sdrctnr_stb	Source → Destination: Memory System → SDRAM CNTR	Registered: No
Pin Function: Asserted to indicate the SDRAM controller is selected.		
Pin Name: u_mem_sys_o_sdrctnr_cyc	Source → Destination: Memory System → SDRAM CNTR	Registered: No
Pin Function: Asserted to indicate valid bus cycle is in progress.		
Pin Name: u_mem_sys_o_sdrctnr_we	Source → Destination: Memory System → SDRAM CNTR	Registered: Yes
Pin Function: Asserted to indicate write cycle, deasserted to indicate read cycle.		
Pin Name: u_mem_sys_o_sdrctnr_sel [3:0]	Source → Destination: Memory System → SDRAM CNTR	Registered: No
Pin Function: Used to indicate where valid data is placed on the input data line during WRITE cycle and where it should present on the output data line during READ cycle.		

Pin Name: u_mem_sys_o_sdrctnr_addr [31:0]	Source → Destination: Memory System → SDRAM CNTR	Registered: No
Pin Function: 32-bit addresses to SDRAM Controller for read or write.		
Pin Name: u_mem_sys_o_sdrctnr_data [31:0]	Source → Destination: Memory System → SDRAM CNTR	Registered: No
Pin Function: 32-bit data to be written into SDRAM.		
Pin Name: u_mem_sys_o_cp0_is_mtc0	Source → Destination: Memory System →CP0	Registered: No
Pin Function: Write enable signal to CP0. 0: Write Disable. 1: Write Enable.		
Pin Name: u_mem_sys_o_cp0_is_eret	Source → Destination: Memory System →CP0	Registered: Yes
Pin Function: Restart instruction signal for CP0. 0: Normal operation. 1: Restart exception instruction.		
Pin Name: u_mem_sys_o_cp0_reg_data [31:0]	Source → Destination: Memory System →CP0	Registered: No
Pin Function: 32 bits data to be written into CP0 register.		
Pin Name: u_mem_sys_o_cp0_reg_address [4:0]	Source → Destination: Memory System →CP0	Registered: No
Pin Function: 5 bits address to indicate which register of CP0 should be update.		
Pin Name: u_mem_sys_o_cp0_tlb_page_fault	Source → Destination: Memory System →CP0	Registered: No
Pin Function: Page fault signal for CP0 to update CAUSE register.		

Pin Name: u_mem_sys_o_cp0_tlb_miss	Source → Destination: Memory System →CP0	Registered: No
Pin Function: Status signal to indicate tlb miss.		
Pin Name: u_mem_sys_o_cp0_tlb_addr_excep	Source → Destination: Memory System →CP0	Registered: No
Pin Function: Status signal to indicate address exception occur in TLB.		

Ttable 5.4.3: Memory System's Output Pin Description

5.5: Memory System Operating Procedure

1. Start the system
2. Porting appropriate instruction, data, first level page table, second level page table into SDRAM.
3. Reset the system for at least 2 clocks
4. While release the reset, the system will automatically run the program inside instruction cache
5. Observe the waveform from the development tools.

Chapter 6: Architecture Specification

6.1: Unit Partition of Memory System

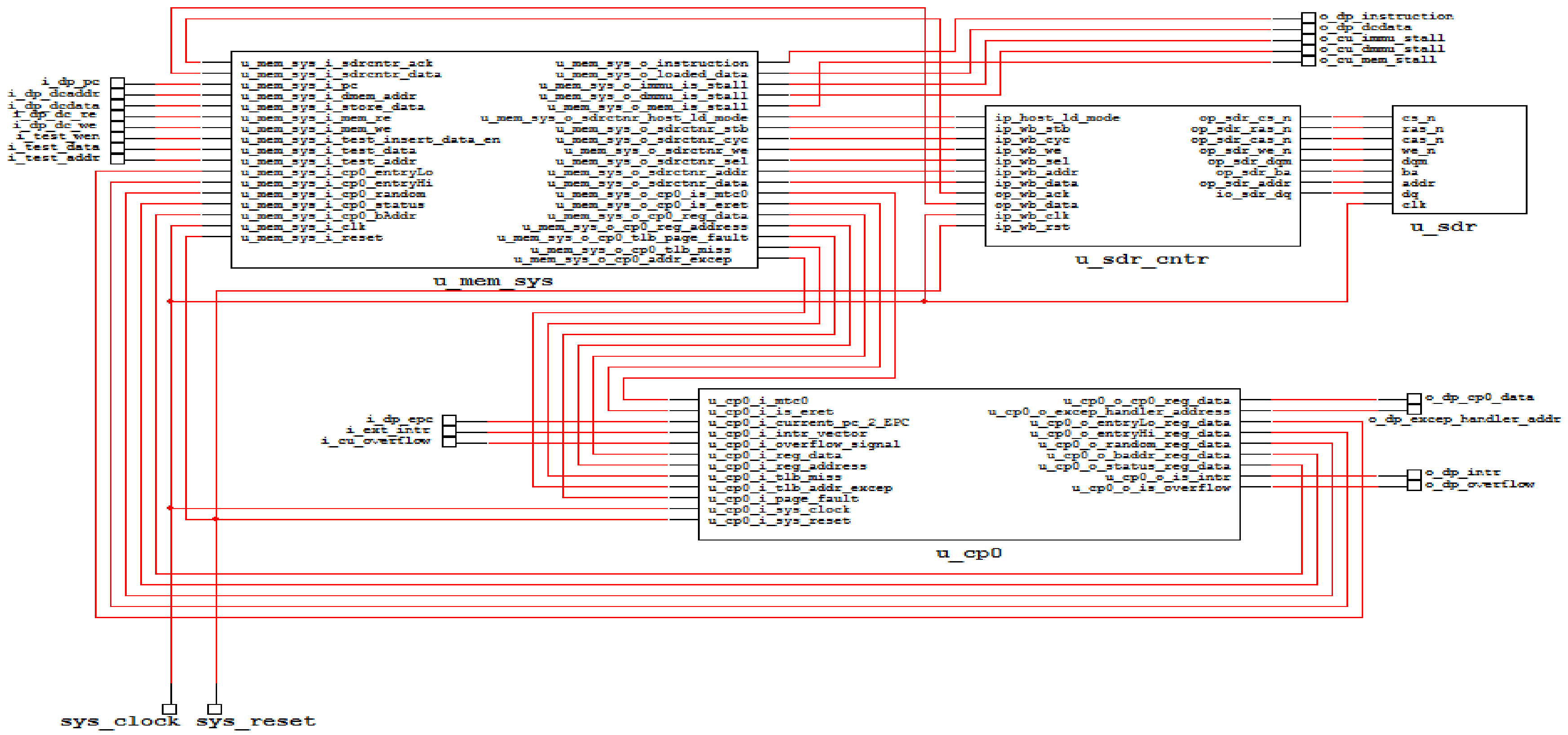


Figure 6.1.1: Unit partition of memory system.

6.2 Design hierarchy

Chip Partitioning at System Level	Unit Partitioning at Architecture Level	Block and Functional Block Partitioning at RTL Level (Micro-Architecture Level)
c_risc32_full	u_data_path_full	b_reg_file
		b_alb_32
		b_mult_32
		b_branch_pred
	u_ctrl_path_full	b_alb_ctrl
		b_iag_ctrl
		b_main_ctrl
		b_fwrd
		b_itl_ctrl
	u_mem_sys	**b_cache (for instruction)**
		b_cache (for data)
		b_tlb (for instruction)
		b_tlb (for data)
		b_mmu (for instruction)
		b_mmu (for data)
	u_cp0	b_cp0_dc
		b_cp0_regfile
Structural description	Structural description/Behavioral description	Behavioral description

Table 6.1.1: Formation of a design hierarchy for Full RISC32 microprocessor through top down design methodology

**Note that this design is provided as a mindset for future improvement.*

**Since the memory system is not ready yet to connect with current RISC32, the following will be discussing what had been done in this project.*

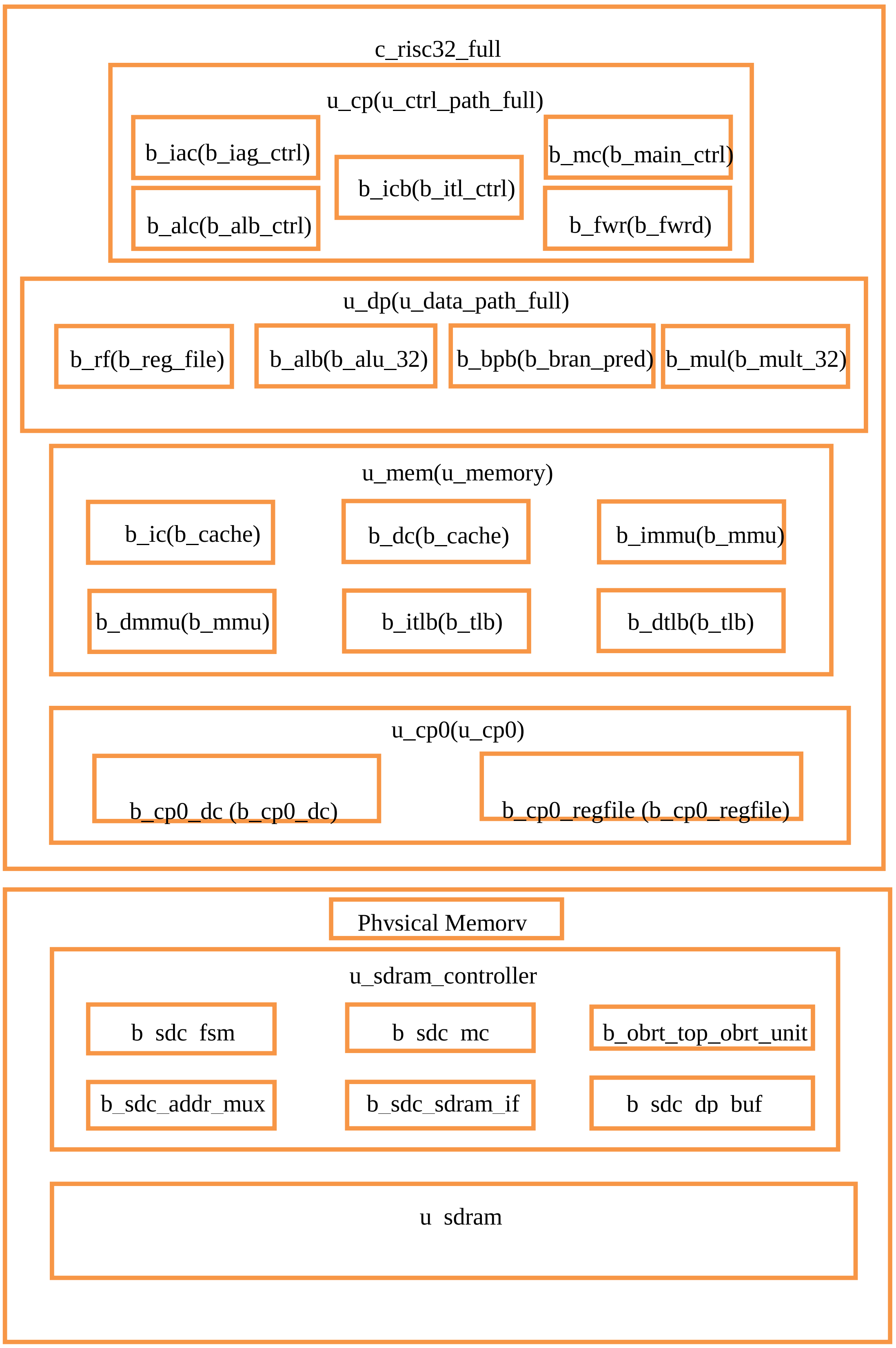


Figure 6.2.1: Full RISC32's Architecture and Micro-Architecture Partitioning

6.3: Memory Unit

**Refer to chapter 4, Memory System Specification.*

6.4: CP0 unit

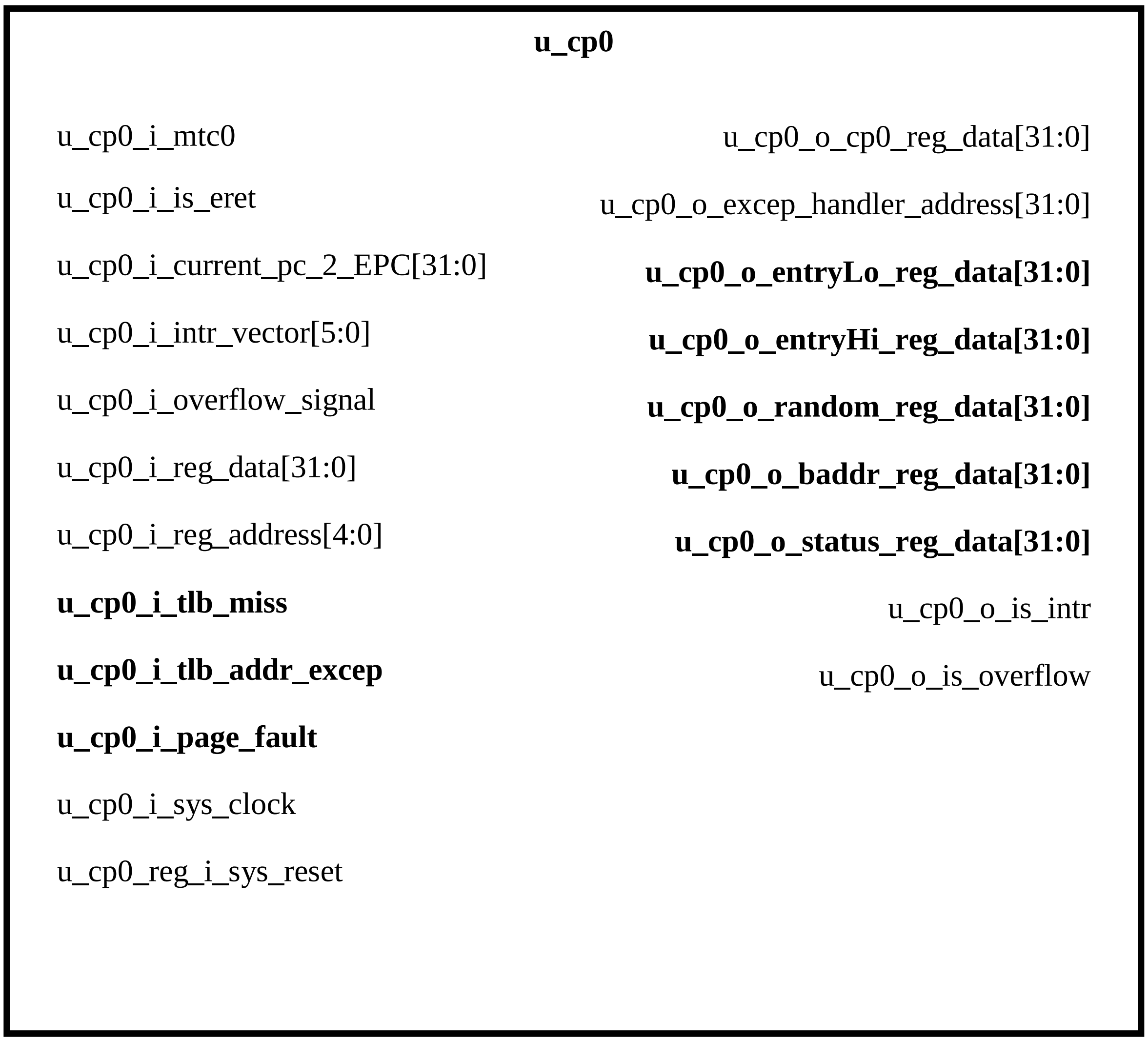


Figure 6.4.1: Block diagram for co-processor 0 which used to process and store exception/interrupt information.

Overview of CP0's register used in Memory System

1. Random Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0																		SLOT						0							

Figure 6.4.2: Random register structure.

**Note that,*

- *SLOT – 6 bits value used to choose which TLB entries to be overwrite when TLB miss occurs. This value is increment every clock cycle.*

2. Status Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
d	c	b	a	0					B	T	E	M	Z	S	I	H	H	H	H	H	H	F	F	0		KUo	IEo	KUp	IEp	KUc	IEc

Figure 6.4.3: Statusregister structure.

**Note that,*

- *abcdTEMZSI–Not related in this project. Set to 0.*
- *B - Boot flag.*
- *H - Hardware interrupt enable bit, lines 0-5.*
- *F - Software interrupt enable bit, lines 0-1.*
- *KU - 1 if user mode, 0 if kernel mode.*
- *IE - 1 if interrupts enabled, 0 if disabled. See below regarding o/p/c.*

3. EntryLo Register

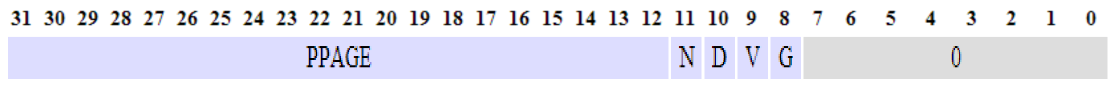


Figure 6.4.4: EntryLo register structure.

**Note that,*

- *PPAGE - Physical page number for TLB entry.*
- *N - Noncached; if set, accesses via TLB entry will be uncached.*
- *D - Dirty; if set, write accesses via TLB entry will be permitted; otherwise exception occurs.*
- *V - Valid; if set, accesses via TLB entry will be permitted; otherwise exception occurs.*
- *G - Global; if set, the ASID field will be ignored when matching TLB entry.*

4. EntryHi Register

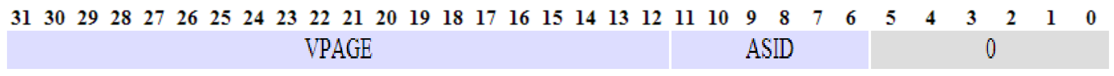


Figure 6.4.5: EntryHi register structure.

**Note that,*

- *VPAGE - virtual page number for TLB entry.*
- *ASID - address space ID for TLB entry.*

5. Baddr Register

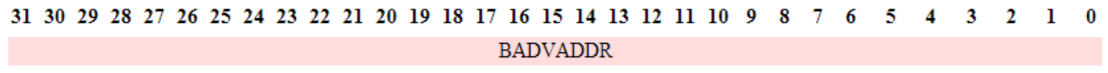


Figure 6.4.6: Baddrregister structure.

**Note that Baddr register is used to store PC value where exception occurs.*

I/O Description

CP0's Input Pin Description

Pin Name: u_cp0_i_mtc0	Source → Destination:ControlPath Unit → Co-Processor 0 Unit	Registered: No
Pin Function: 1 bit flag indicate instruction mtc0. 0: Not mtc0 instruction 1: mtc0 instruction		
Pin Name: u_cp0_i_is_eret	Source → Destination:Control PathUnit→ Co-Processor 0 Unit	Registered: No
Pin Function: 1 bit flag indicate instruction eret. 0: not eret instruction 1: eret instruction		
Pin Name: u_cp0_i_current_pc_2_EPC [31:0]	Source → Destination:Datapath Unit→ Co-Processor 0 Unit	Registered: No
Pin Function: 32 bit of current Program Counter (PC) value.		
Pin Name: u_cp0_i_intr_vector [5:0]	Source → Destination:Externaldevice → Co-Processor 0 Unit	Registered: No
Pin Function: Each bit of this input is indicating interrupt signal from external device.		

Pin Name: u_cp0_i_overflow_signal	Source → Destination:Datapath Unit→ Co-Processor 0 Unit	Registered: No
Pin Function: 1 bit flag indicate overflow happen. 0: no overflow happen 1: overflow happened		
Pin Name: u_cp0_i_reg_data [31:0]	Source → Destination:Datapath Unit→ Co-Processor 0 Unit	Registered: No
Pin Function: 32 bit data to be store in CP0 register file.		
Pin Name: u_cp0_i_reg_address [4:0]	Source → Destination:Datapath Unit→ Co-Processor 0 Unit	Registered: No
Pin Function: Address indicates CP0 register file location.		
Pin Name: **u_cp0_i_tlb_miss**	**Source → Destination: Memory Unit → Co-Processor 0 Unit**	**Registered:** **No**
Pin Function: **1 bit flag to indicate TLB miss.** **0: No TLB miss occurs.** **1: TLB miss occurs.**		
Pin Name: **u_cp0_i_tlb_addr_excep**	**Source → Destination: Memory Unit → Co-Processor 0 Unit**	**Registered:** **No**
Pin Function: **1 bit flag to indicate TLB address exception.** **0: No TLB address exception occurs.** **1: TLB address exception occurs.**		

Pin Name: **u_cp0_i_page_fault**	**Source → Destination:** **Memory Unit → Co-Processor 0 Unit**	**Registered:** **No**
Pin Function: **1 bit flag to indicate page fault.** **0: No page fault occurs.** **1: Page fault occurs.**		
Pin Name: u_cp0_i_sys_clock	Source → Destination: Micro-processor → Co-Processor 0 Unit	Registered: No
Pin Function: Synchronous System clock.		
Pin Name: u_cp0_reg_i_sys_reset	Source → Destination: Micro-processor → Co-Processor 0 Unit	Registered: No
Pin Function: Global reset signal.		

Table 6.4.1: CP0's Input Pin Description

<u>CP0's Output Pin Description</u>

Pin Name: u_cp0_o_cp0_reg_data [31:0]	Source → Destination: Co-processor 0 Unit→Datapath Unit	Registered: No
Pin Function: 32 bit Co-processor 0 registers value to be store in main Register File.		
Pin Name: u_cp0_o_excep_handler_address [31:0]	Source → Destination: Co-processor 0 Unit→Datapath Unit	Registered: No
Pin Function: 32 bit Program Counter (PC) address.		

Pin Name: **u_cp0_o_entryLo_reg_data [31:0]**	**Source → Destination:** **Co-processor 0 Unit→ Memory Unit**	**Registered:** **No**
Pin Function: **32 bits EntryLo register data.**		
Pin Name: **u_cp0_o_entryHi_reg_data [31:0]**	**Source → Destination:** **Co-processor 0 Unit→ Memory Unit**	**Registered:** **No**
Pin Function: **32 bits EntryHi register data.**		
Pin Name: **u_cp0_o_random_reg_data [31:0]**	**Source → Destination:** **Co-processor 0 Unit→ Memory Unit**	**Registered:** **No**
Pin Function: **32 bits Random Register Data.**		
Pin Name: **u_cp0_o_baddr_reg_data[31:0]**	**Source → Destination:** **Co-processor 0 Unit→ Memory Unit**	**Registered:** **No**
Pin Function: **32 bits Baddr register data.**		
Pin Name: **u_cp0_o_status_reg_data [31:0]**	**Source → Destination:** **Co-processor 0 Unit→ Memory Unit**	**Registered:** **No**
Pin Function: **32 bits Status register data.**		

Pin Name: u_cp0_o_is_intr	Source → Destination: Co-processor 0 Unit→ControlPath Unit	Registered: No
Pin Function: 1 bit signal to Control Unit to indicate interrupt happen. 0: No interrupt 1: Interrupt happened		
Pin Name: u_cp0_o_is_overflow	Source → Destination: Co-processor 0 Unit→ControlPath Unit	Registered: No
Pin Function: 1 bit signal to Control Unit to indicate overflow happen. 0: No Overflow 1: Overflow happened		

Table 6.4.2: CP0's Output Pin Description

6.5: SDRAM Controller

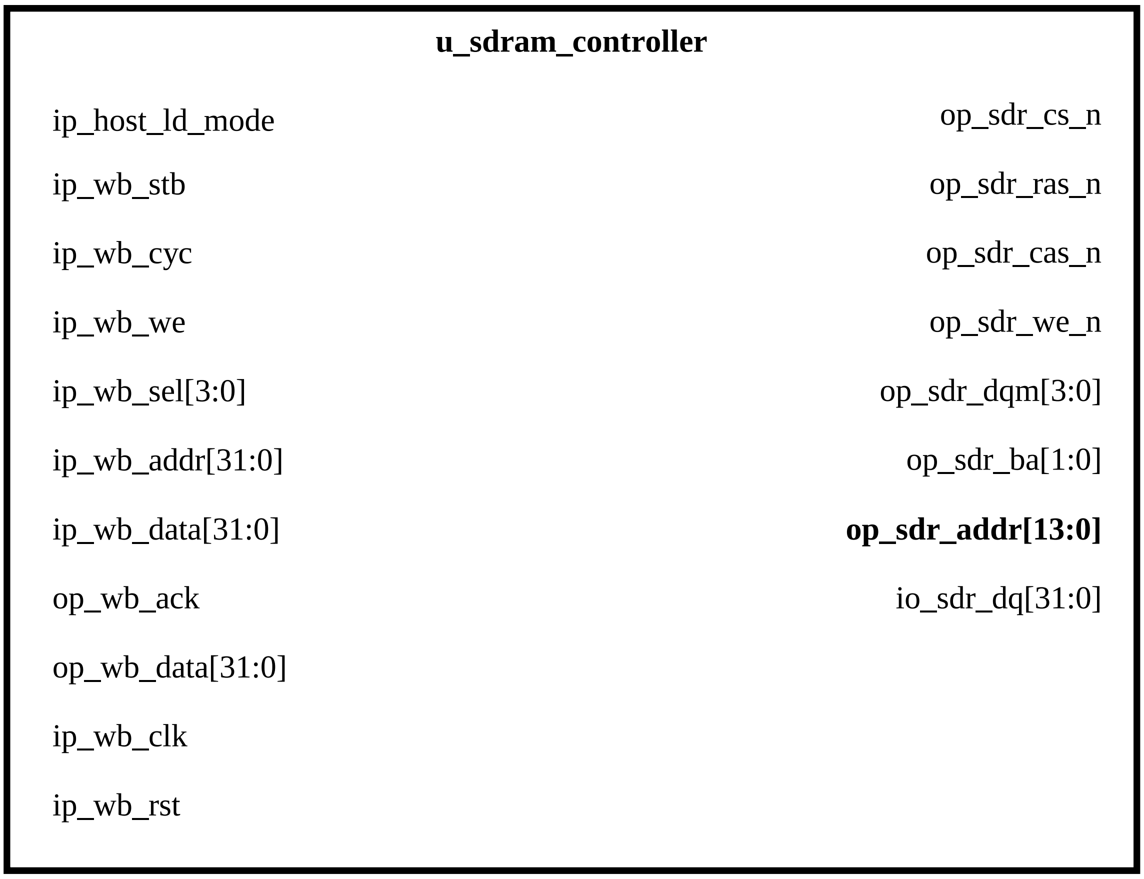


Figure 6.5.1: Block diagram for SDRAM controller.[Modified from [9]]

**Note that for the previous design of SDRAM Controller is based on 16MB of SDRAM provided by Micron. In order to communicate with a 64MB SDRAM, some modification had been made.*

I/O Description

SDRAM Controller's Input Pin Description

Pin Name: ip_host_ld_mode	Source → Destination: Memory Unit → SDRAM Controller	Registered: No
Pin Function: Asserted to load a new mode into the SDRAM.		
Pin Name: ip_wb_stb	Source → Destination: Memory Unit → SDRAM Controller	Registered: No
Pin Function: Asserted to indicate the SDRAM controller is selected.		
Pin Name: ip_wb_cyc	Source → Destination: Memory Unit → SDRAM Controller	Registered: No
Pin Function: Asserted to indicate valid bus cycle is in progress.		
Pin Name: ip_wb_we	Source → Destination: Memory Unit → SDRAM Controller	Registered: No
Pin Function: Asserted to indicate that the current cycle is READ. Deasserted to indicate current cycle is WRITE.		
Pin Name: ip_wb_sel[3:0]	Source → Destination: Memory Unit → SDRAM Controller	Registered: No
Pin Function: Used to indicate where valid data is placed on the input data line (ip_wb_dat) during WRITE cycle and where it should present on the output data line (op_wb_dat) during READ cycle.		
Pin Name: ip_wb_addr[31:0]	Source → Destination: Memory Unit → SDRAM Controller	Registered: No
Pin Function: Used to pass the memory address from the host.		

Pin Name: ip_wb_data[31:0]	Source → Destination: Memory Unit → SDRAM Controller	Registered: No
Pin Function: Used to pass WRITE data from the host.		
Pin Name: ip_wb_clk	Source → Destination: Memory Unit → SDRAM Controller	Registered: No
Pin Function: Clock signal to synchronize to the system.		
Pin Name: ip_wb_rst	Source → Destination: System Clock→ SDRAM Controller	Registered: No
Pin Function: Synchronous reset to reinitialize the system.		

Table 6.5.1: SDRAM Controller's Input Pin Description

SDRAM Controller's Output Pin Description

Pin Name: op_wb_ack	Source → Destination: SDRAM Controller → Memory Unit	Registered: No
Pin Function: Asserted to indicate that the current READ or WRITE operation is successful.		
Pin Name: op_wb_data[31:0]	Source → Destination: SDRAM Controller → Memory Unit	Registered: No
Pin Function: Used to output READ data from the SDRAM.		
Pin Name: op_sdr_cs_n	Source → Destination: SDRAM Controller → SDRAM	Registered: No
Pin Function: SDRAM chip select.		

Pin Name: op_sdr_ras_n	Source → Destination: SDRAM Controller → SDRAM	Registered: No
Pin Function: SDRAM row address select.		
Pin Name: op_sdr_cas_n	Source → Destination: SDRAM Controller → SDRAM	Registered: No
Pin Function: SDRAM column address select.		
Pin Name: op_sdr_we_n	Source → Destination: SDRAM Controller → SDRAM	Registered: No
Pin Function: SDRAM write enable.		
Pin Name: op_sdr_dqm[3:0]	Source → Destination: SDRAM Controller → SDRAM	Registered: No
Pin Function: Used to select which bits of the data line (io_sdr_dq) to be masked.		
Pin Name: op_sdr_ba[1:0]	Source → Destination: SDRAM Controller → SDRAM	Registered: No
Pin Function: Bank Address output to SDRAM.		
Pin Name: **op_sdr_addr[13:0]**	**Source → Destination:** **SDRAM Controller →** **SDRAM**	**Registered:** **No**
Pin Function: **14 bits address output to the SDRAM.**		
Pin Name: io_sdr_dq[31:0]	Source → Destination: SDRAM Controller → SDRAM	Registered: No
Pin Function: Bidirectional data line to receive READ data or send WRITE data.		

Table 6.5.2: SDRAM Controller's Output Pin Description

6.6: 64 MB SDRAM

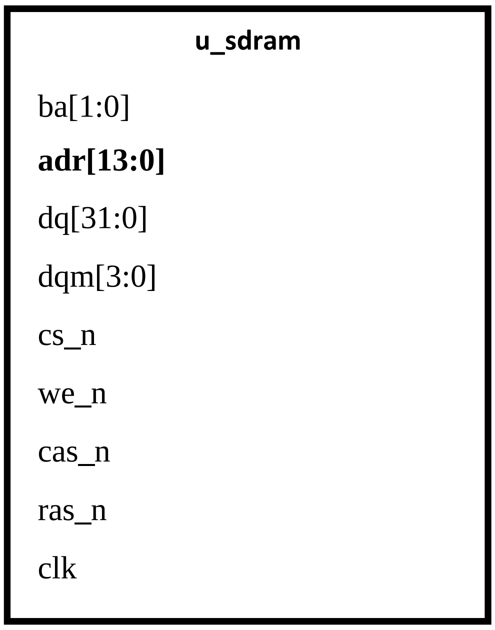


Figure 6.6.1: Block diagram for SDRAM. .[Modified from [9]]

<u>I/O Description</u>

<u>SDRAM's Input Pin Description</u>

Pin Name: op_sdr_cs_n	Source → Destination: SDRAM Controller → SDRAM	Registered: No
Pin Function: SDRAM chip select.		
Pin Name: op_sdr_ras_n	Source → Destination: SDRAM Controller → SDRAM	Registered: No
Pin Function: SDRAM row address select.		
Pin Name: op_sdr_cas_n	Source → Destination: SDRAM Controller → SDRAM	Registered: No
Pin Function: SDRAM column address select.		

Pin Name: op_sdr_we_n	Source → Destination: SDRAM Controller → SDRAM	Registered: No
Pin Function: SDRAM write enable.		
Pin Name: op_sdr_dqm[3:0]	Source → Destination: SDRAM Controller → SDRAM	Registered: No
Pin Function: Used to select which bits of the data line (io_sdr_dq) to be masked.		
Pin Name: op_sdr_ba[1:0]	Source → Destination: SDRAM Controller → SDRAM	Registered: No
Pin Function: Bank Address output to SDRAM.		
Pin Name: **op_sdr_addr[13:0]**	**Source → Destination:** **SDRAM Controller → SDRAM**	**Registered:** **No**
Pin Function: **14 bits address output to the SDRAM.**		
Pin Name: io_sdr_dq[31:0]	Source → Destination: SDRAM Controller → SDRAM	Registered: No
Pin Function: Bidirectional data line to receive READ data or send WRITE data.		

Table 6.6.1: SDRAM's Input Pin Description

Chapter 7: Micro-Architecture Specification

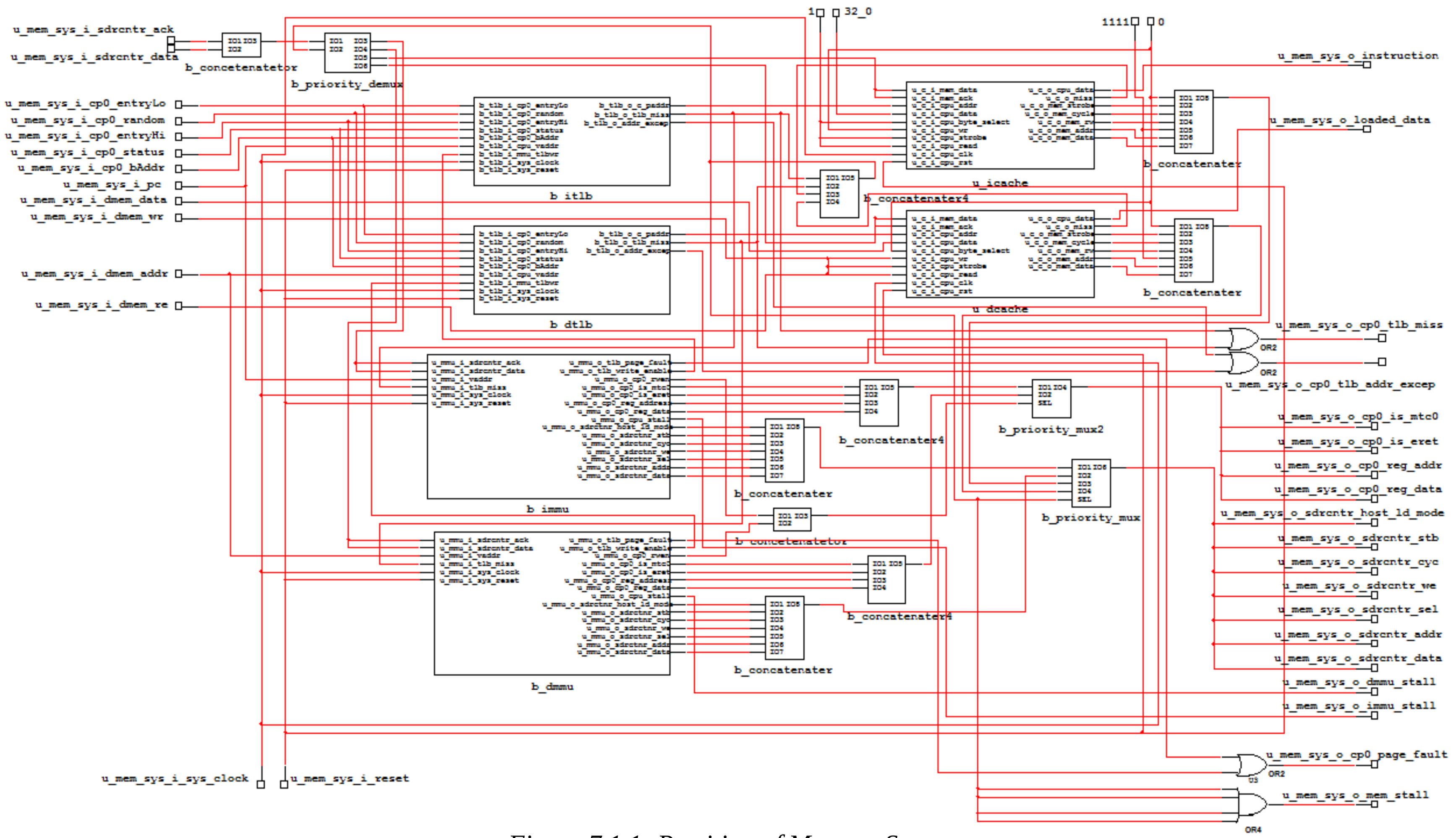


Figure 7.1.1: Partition of Memory System

7.1 Translation Lookaside Buffer (TLB)

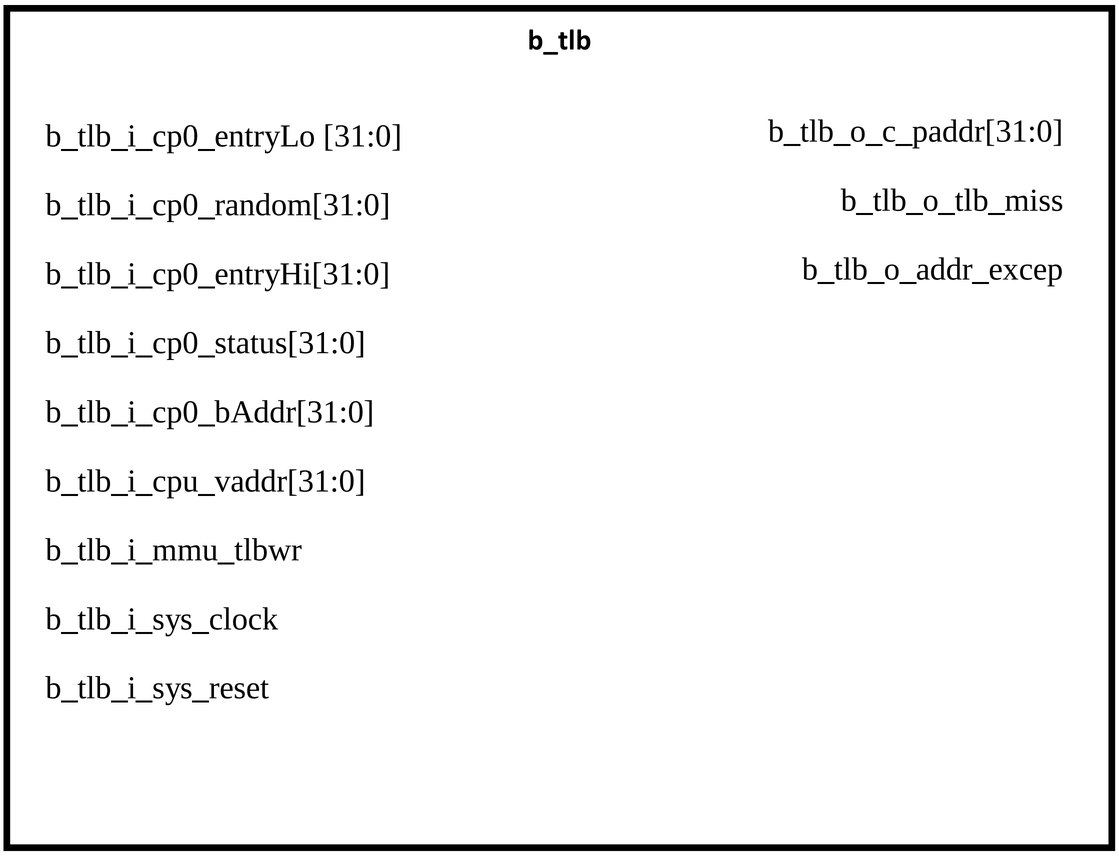


Figure 7.1.2:Block diagram for TLB.

Translation Lookaside Buffer is just like a cache which holds some of the page table entries which can be reside either in physical memory or disk. Its responsibility including translate virtual address given by CPU into a physical address and ensure each user process does not able to access to kernel segment. In this project, *assume that instruction TLB and data TLB is the same.*

Feature:

1. Consist of 64 entries.
2. Fully associative.
3. Capable to handle TLB Miss together with MMU (Memory Management Unit).

Address Translation Scenario

Based on what we had discussed in previous chapter, we know that the address translation is important for us to get the physical address which used to either write or read data. Figure 5.3.1 and 5.4.1 give a clearer picture which told us that the cache miss and TLB miss are the independent event that a cache miss only can occur when there is a TLB hit. On the other way is means that the data must be present inside the main memory only we can access to cache. To further discuss about this, the table below provide us a simplest way to examine the relationship between cache and TLB.

TLB	Page Table	Cache	Events Possible? If so, under what circumstance?
hit	hit	miss	Possible, although the page table is never really check after TLB hits.
miss	hit	hit	Possible, although TLB misses, entry found in page table; after retry, data found in cache.
miss	hit	miss	Possible, although TLB misses, entry found in page table; after retry, data misses in cache.
miss	miss	miss	Possible, TLB misses follow by page fault, data must misses in cache.
hit	hit	hit	Possible, although the page table is never really check after TLB hits.
hit	miss	miss	Impossible, TLB must misses if page is not present in main memory.
hit	miss	hit	Impossible, TLB must misses if page is not present in main memory.
miss	miss	hit	Impossible, data must misses in cache if page is not present in main memory.

Table 7.1.1: Possible combinations of events in the TLB, virtual memory system and cache.

I/O Description

TLB's Input Pin Description

Pin Name: b_tlb_i_cp0_entryLo [31:0]	Source → Destination: CP0 → TLB	Registered: No
Pin Function: 32 bits EntryLo register from CP0.		
Pin Name: b_tlb_i_cp0_random [31:0]	Source → Destination: CP0 → TLB	Registered: No
Pin Function: 32 bits Random register from CP0.		
Pin Name: b_tlb_i_cp0_entryHi [31:0]	Source → Destination: CP0 → TLB	Registered: No
Pin Function: 32 bits EntryHi register from CP0.		
Pin Name: b_tlb_i_cp0_status [31:0]	Source → Destination: CP0 → TLB	Registered: No
Pin Function: 32 bits Status register from CP0.		
Pin Name: b_tlb_i_cp0_bAddr [31:0]	Source → Destination: CP0 → TLB	Registered: No
Pin Function: 32 bits Baddr register from CP0.		
Pin Name: b_tlb_i_cpu_vaddr [31:0]	Source → Destination: CPU → TLB	Registered: No
Pin Function: 32 bits address virtual address from CPU.		

Pin Name: b_tlb_i_mmu_tlbwr	Source → Destination: MMU → TLB	Registered: No
Pin Function: 1 bit flag to enable write to TLB entry. 0: Write Disable. 1: Write Enable.		
Pin Name: b_tlb_i_sys_clock	Source → Destination: System Clock → TLB	Registered: No
Pin Function: System clock signal		
Pin Name: b_tlb_i_sys_reset	Source → Destination: System Reset → TLB	Registered: No
Pin Function: System reset signal		

Table 7.1.2: TLB's Input Pin Description

<u>TLB's Output Pin Description</u>

Pin Name: b_tlb_o_c_paddr [31:0]	Source → Destination: TLB → Cache	Registered: No
Pin Function: 32 bits physical address output to cache.		
Pin Name: b_tlb_o_tlb_miss	Source → Destination: TLB→CP0 & MMU	Registered: No
Pin Function: 1 bit flag to indicate TLB miss. 0: No TLB miss. 1: TLB miss.		

Pin Name: b_tlb_o_addr_excep	Source → Destination: TLB→CP0	Registered: No
Pin Function: 1 bit flag to indicate TLB address exception. 0: No TLB address exception. 1: TLB address exception.		

Table 7.1.3: TLB's Output Pin Description

Functionality

1. Compare with Status register to determine TLB address exception.
2. Able to translation virtual address to physical address based on the TLB entries.
3. Send out miss signal when there are no entries matched.

7.2 Memory Management Unit (MMU)

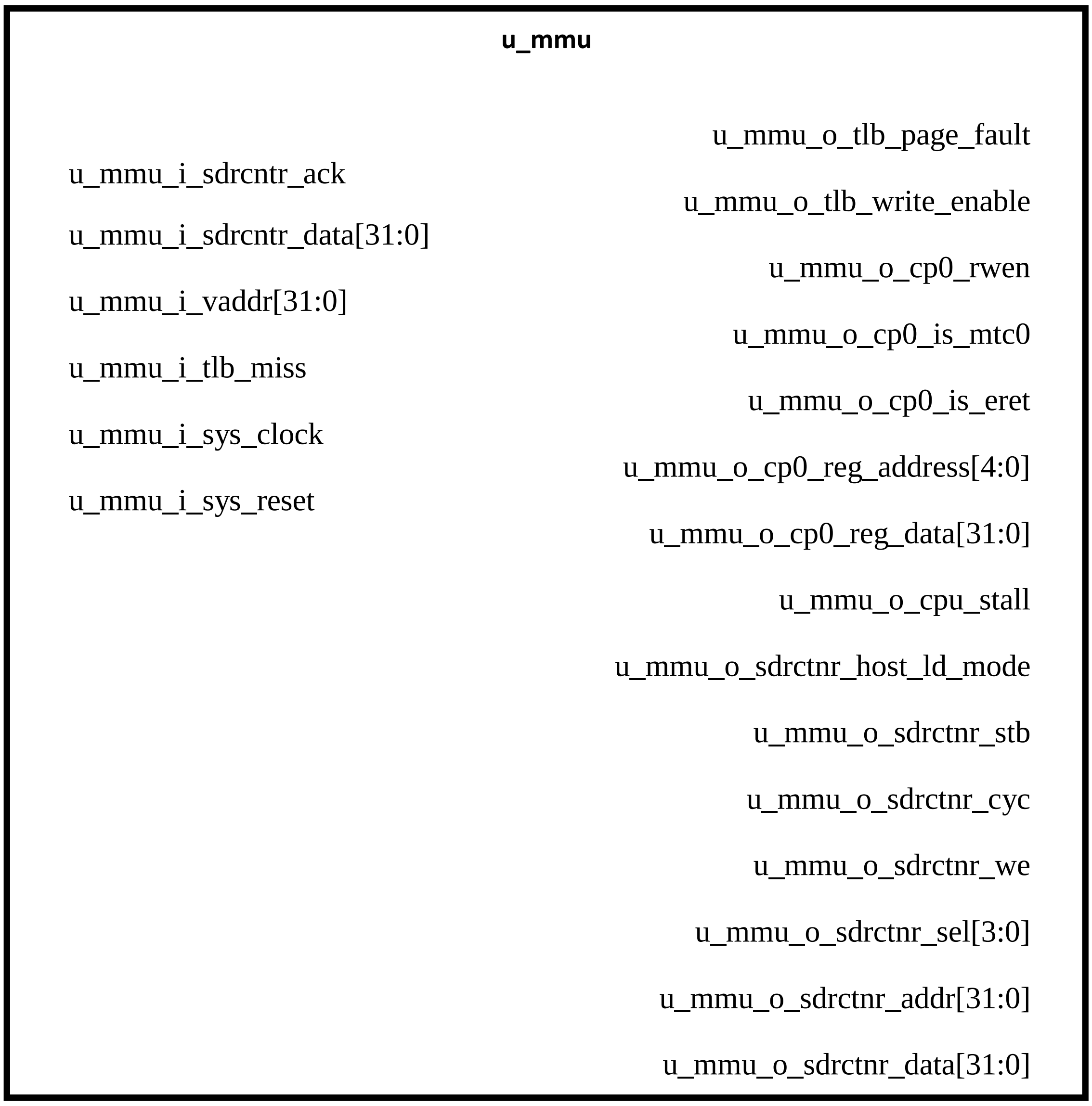


Figure 7.2.1: Block diagram for MMU.

Memory Management Unit is responsible to handle the page table walk through when TLB Miss occurs. In this project, two-level page table is used. Therefore, for each time TLB miss and invoke MMU to handle Page Table Entries (PTE) transfer, physical memory has to be access twice to get the appropriate PTE.

I/O Description

MMU’s Input Pin Description

Pin Name: u_mmu_i_sdrcntr_ack	Source → Destination: SDRAM Controller→MMU	Registered: No
Pin Function: Acknowledge signal from SDRAM Controller asserted to indicate whether completion of read or write operation.		
Pin Name: u_mmu_i_sdrcntr_data[31:0]	Source → Destination: SDRAM Controller→MMU	Registered: No
Pin Function: 32 bits read data from SDRAM Controller.		
Pin Name: u_mmu_i_vaddr[31:0]	Source → Destination: Data Path Unit→MMU	Registered: No
Pin Function: 32 bits virtual address from data path unit.		
Pin Name: u_mmu_i_tlb_miss	Source → Destination: TLB→MMU	Registered: No
Pin Function: TLB miss signal from TLB.		
Pin Name: u_mmu_i_sys_clock	Source → Destination: System Clock→MMU	Registered: No
Pin Function: System Clock Signal.		
Pin Name: u_mmu_i_sys_reset	Source → Destination: System Reset→MMU	Registered: No
Pin Function: System Reset Signal.		

Table 7.2.1: MMU’s Input Pin Description

MMU's Output Pin Description

Pin Name: u_mmu_o_tlb_page_fault	Source → Destination: MMU→ CP0	Registered: No
Pin Function: 1 bit signal asserted to indicate page fault happen.		
Pin Name: u_mmu_o_tlb_write_enable	Source → Destination: MMU→ TLB	Registered: No
Pin Function: 1 bit signal asserted toenable write in TLB.		
Pin Name: u_mmu_o_cp0_rwen	Source → Destination: MMU→ Multiplexer	Registered: No
Pin Function: 1 bit signal to select which data should go to CP0 between IMMU and DMMU.		
Pin Name: u_mmu_o_cp0_is_mtc0	Source → Destination: MMU→ CP0	Registered: No
Pin Function: Instruction signal to insert data into CP0 register file.		
Pin Name: u_mmu_o_cp0_is_eret	Source → Destination: MMU→ CP0	Registered: No
Pin Function: 1 bit signal to indicate end of TLB miss by sending the signal to CP0 and CP0 will restart the instruction by loading address store in EPC register.		
Pin Name: u_mmu_o_cp0_reg_address[4: 0]	Source → Destination: MMU→ CP0	Registered: No
Pin Function: 5 bits register address to be update.		

Pin Name: u_mmu_o_cp0_reg_data[31:0]	Source → Destination: MMU→ CP0	Registered: No
Pin Function: 32 bits register data to be update in CP0 register file.		
Pin Name: u_mmu_o_cpu_stall	Source → Destination: MMU→ Control Unit	Registered: No
Pin Function: Stall signal to control unit when TLB miss.		
Pin Name: u_mmu_o_sdrctnr_host_ld_mode	Source → Destination: MMU→ SDRAM Controller	Registered: No
Pin Function: Asserted to load a new mode into the SDRAM.		
Pin Name: u_mmu_o_sdrctnr_stb	Source → Destination: MMU→ SDRAM Controller	Registered: No
Pin Function: Asserted to indicate the SDRAM controller is selected.		
Pin Name: u_mmu_o_sdrctnr_cyc	Source → Destination: MMU→ SDRAM Controller	Registered: No
Pin Function: Asserted to indicate valid bus cycle is in progress.		
Pin Name: u_mmu_o_sdrctnr_we	Source → Destination: MMU→ SDRAM Controller	Registered: No
Pin Function: Asserted to indicate that the current cycle is READ. Deasserted to indicate current cycle is WRITE.		

Pin Name: u_mmu_o_sdrctnr_sel[3:0] 	Source → Destination: MMU→ SDRAM Controller	Registered: No
Pin Function: Used to indicate where valid data is placed on the input data line (ip_wb_dat) during WRITE cycle and where it should present on the output data line (op_wb_dat) during READ cycle.		
Pin Name: u_mmu_o_sdrctnr_addr[31:0]	Source → Destination: MMU→ SDRAM Controller	Registered: No
Pin Function: 32 bits address to read or write from SDRAM.		
Pin Name: u_mmu_o_sdrctnr_data[31:0]	Source → Destination: MMU→ SDRAM Controller	Registered: No
Pin Function: 32 bits data to be write into SDRAM.		

Table 7.2.2: MMU's Output Pin Description

Memory Management Unit (MMU) Protocol

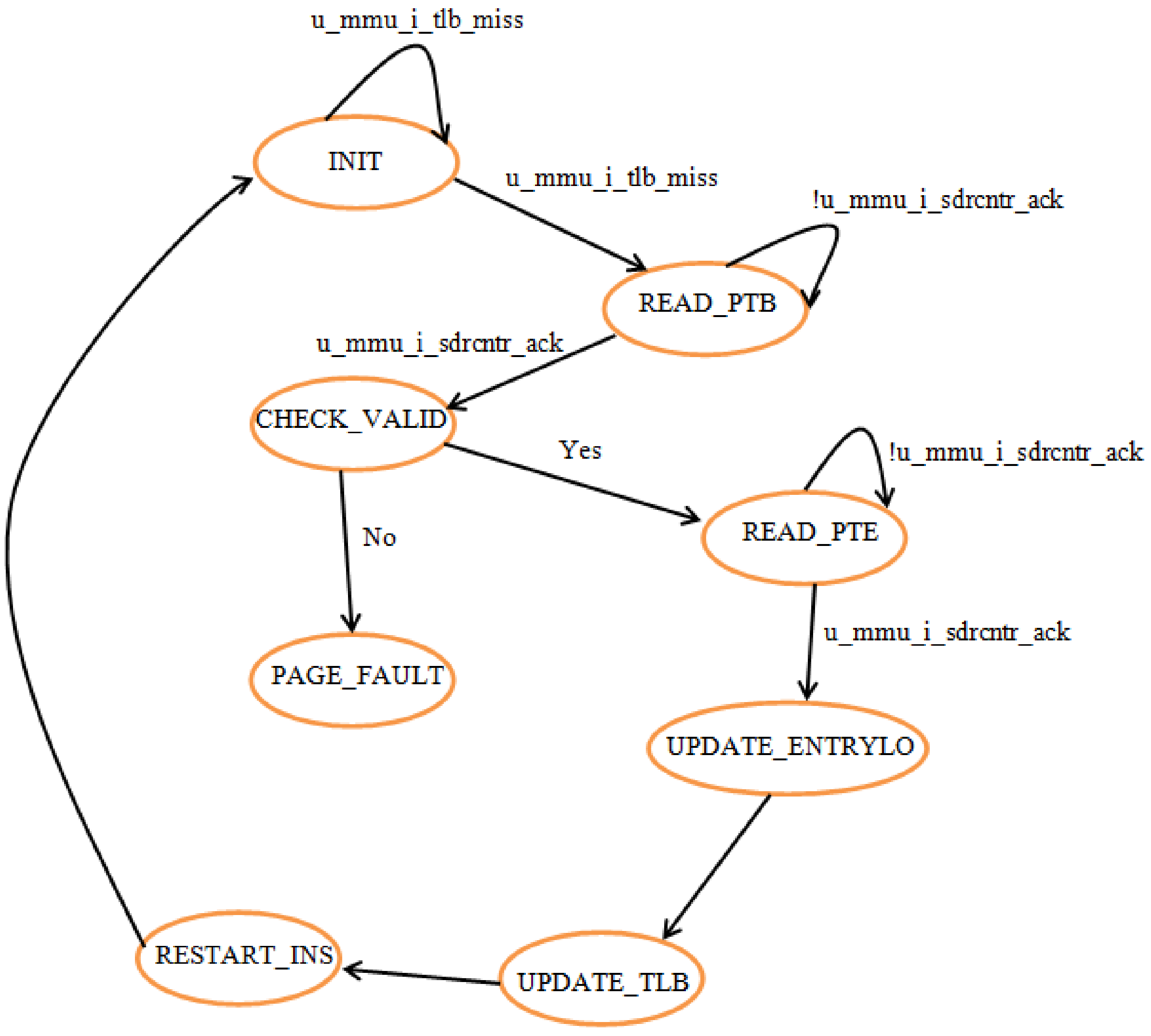


Figure 7.2.2: MMU protocol.

Output for each state in MMU protocol

State	Output
INIT	`u_mmu_o_tlb_page_fault       <=  1'b0;`
	`u_mmu_o_tlb_write_enable     <=  1'b0;`
	`u_mmu_o_cp0_rwen             <=  1'b0;`
	`u_mmu_o_cp0_is_eret          <=  1'b0;`
	`u_mmu_o_cp0_is_mtc0          <=  1'b0;`
	`u_mmu_o_cp0_reg_address      <=  5'bz;`
	`u_mmu_o_cp0_reg_data         <= 32'bz;`
	`u_mmu_o_cpu_stall            <=  1'b0;`
	`u_mmu_o_sdrctnr_host_ld_mode <=  1'b0;`
	`u_mmu_o_sdrctnr_stb          <=  1'b0;`
	`u_mmu_o_sdrctnr_cyc          <=  1'b0;`
	`u_mmu_o_sdrctnr_we           <=  1'b0;`
	`u_mmu_o_sdrctnr_sel          <=  4'b0;`
	`u_mmu_o_sdrctnr_addr         <= 32'bz;`
	`u_mmu_o_sdrctnr_data         <= 32'bz;`
READ_PTBR	`u_mmu_o_tlb_page_fault       <=  1'b0;`
	`u_mmu_o_tlb_write_enable     <=  1'b0;`
	`u_mmu_o_cp0_rwen             <=  1'b0;`
	`u_mmu_o_cp0_is_eret          <=  1'b0;`
	`u_mmu_o_cp0_is_mtc0          <=  1'b0;`
	`u_mmu_o_cp0_reg_address      <=  5'bz;`
	`u_mmu_o_cp0_reg_data         <=  32'bz;`
	`u_mmu_o_cpu_stall            <=  1'b1;`
	`u_mmu_o_sdrctnr_host_ld_mode <=  1'b0;`
	`u_mmu_o_sdrctnr_stb          <=  1'b1;`
	`u_mmu_o_sdrctnr_cyc          <=  1'b1;`
	`u_mmu_o_sdrctnr_we           <=  1'b0;`
	`u_mmu_o_sdrctnr_sel          <=  4'b1111;`
	`u_mmu_o_sdrctnr_addr         <= {6'b0,14'b00_0000_0000_0000,u_mmu_i_vaddr[31:22],2'b0};`
	`u_mmu_o_sdrctnr_data         <= 32'bz;`
CHECK_VALID	`u_mmu_o_tlb_page_fault       <=  1'b0;`
	`u_mmu_o_tlb_write_enable     <=  1'b0;`
	`u_mmu_o_cp0_rwen             <=  1'b0;`
	`u_mmu_o_cp0_is_eret          <=  1'b0;`
	`u_mmu_o_cp0_is_mtc0          <=  1'b0;`
	`u_mmu_o_cp0_reg_address      <=  5'bz;`
	`u_mmu_o_cp0_reg_data         <= 32'bz;`
	`u_mmu_o_cpu_stall            <=  1'b1;`
	`u_mmu_o_sdrctnr_host_ld_mode <=  1'b0;`
	`u_mmu_o_sdrctnr_stb          <=  1'b0;`
	`u_mmu_o_sdrctnr_cyc          <=  1'b0;`
	`u_mmu_o_sdrctnr_we           <=  1'b0;`
	`u_mmu_o_sdrctnr_sel          <=  4'b0;`
	`u_mmu_o_sdrctnr_addr         <= 32'bz;`
	`u_mmu_o_sdrctnr_data         <= 32'bz;`
PAGE_FAULT	`u_mmu_o_tlb_page_fault       <=  1'b1;`
	`u_mmu_o_tlb_write_enable     <=  1'b0;`

	u_mmu_o_cp0_rwen <= 1'b0;
	u_mmu_o_cp0_is_eret <= 1'b0;
	u_mmu_o_cp0_is_mtc0 <= 1'b0;
	u_mmu_o_cp0_reg_address <= 5'bz;
	u_mmu_o_cp0_reg_data <= 32'bz;
	u_mmu_o_cpu_stall <= 1'b1;
	u_mmu_o_sdrctnr_host_ld_mode <= 1'b0;
	u_mmu_o_sdrctnr_stb <= 1'b0;
	u_mmu_o_sdrctnr_cyc <= 1'b0;
	u_mmu_o_sdrctnr_we <= 1'b0;
	u_mmu_o_sdrctnr_sel <= 4'b0;
	u_mmu_o_sdrctnr_addr <= 32'bz;
	u_mmu_o_sdrctnr_data <= 32'bz;
READ_PTE	u_mmu_o_tlb_page_fault <= 1'b0;
	u_mmu_o_tlb_write_enable <= 1'b0;
	u_mmu_o_cp0_rwen <= 1'b0;
	u_mmu_o_cp0_is_eret <= 1'b0;
	u_mmu_o_cp0_is_mtc0 <= 1'b0;
	u_mmu_o_cp0_reg_address <= 5'bz;
	u_mmu_o_cp0_reg_data <= 32'bz;
	u_mmu_o_cpu_stall <= 1'b1;
	u_mmu_o_sdrctnr_host_ld_mode <= 1'b0;
	u_mmu_o_sdrctnr_stb <= 1'b1;
	u_mmu_o_sdrctnr_cyc <= 1'b1;
	u_mmu_o_sdrctnr_we <= 1'b0;
	u_mmu_o_sdrctnr_sel <= 4'b1111;
	u_mmu_o_sdrctnr_addr <= {6'b0,u_mmu_r_buffer[13:0],u_mmu_i_vaddr[21:12],2'b0};
	u_mmu_o_sdrctnr_data <= 32'bz;
UPDATE_ENTRYLO	u_mmu_o_tlb_page_fault <= 1'b0;
	u_mmu_o_tlb_write_enable <= 1'b0;
	u_mmu_o_cp0_rwen <= 1'b1;
	u_mmu_o_cp0_is_eret <= 1'b0;
	u_mmu_o_cp0_is_mtc0 <= 1'b1;
	u_mmu_o_cp0_reg_address <= 5'b00010;
	u_mmu_o_cp0_reg_data <= {u_mmu_r_buffer[19:0],u_mmu_r_buffer[23:20],8'b0};
	u_mmu_o_cpu_stall <= 1'b1;
	u_mmu_o_sdrctnr_host_ld_mode <= 1'b0;
	u_mmu_o_sdrctnr_stb <= 1'b0;
	u_mmu_o_sdrctnr_cyc <= 1'b0;
	u_mmu_o_sdrctnr_we <= 1'b0;
	u_mmu_o_sdrctnr_sel <= 4'b0;
	u_mmu_o_sdrctnr_addr <= 32'bz;
	u_mmu_o_sdrctnr_data <= 32'bz;
UPDATE_TLB	u_mmu_o_tlb_page_fault <= 1'b0;
	u_mmu_o_tlb_write_enable <= 1'b1;
	u_mmu_o_cp0_rwen <= 1'b0;
	u_mmu_o_cp0_is_eret <= 1'b0;
	u_mmu_o_cp0_is_mtc0 <= 1'b0;
	u_mmu_o_cp0_reg_address <= 5'b0;
	u_mmu_o_cp0_reg_data <= 32'bz;

	u_mmu_o_cpu_stall <= 1'b1;
	u_mmu_o_sdrctnr_host_ld_mode <= 1'b0;
	u_mmu_o_sdrctnr_stb <= 1'b0;
	u_mmu_o_sdrctnr_cyc <= 1'b0;
	u_mmu_o_sdrctnr_we <= 1'b0;
	u_mmu_o_sdrctnr_sel <= 4'b0;
	u_mmu_o_sdrctnr_addr <= 32'bz;
	u_mmu_o_sdrctnr_data <= 32'bz;

RESTART_INS	u_mmu_o_tlb_page_fault <= 1'b0;
	u_mmu_o_tlb_write_enable <= 1'b0;
	u_mmu_o_cp0_rwen <= 1'b1;
	u_mmu_o_cp0_is_eret <= 1'b1;
	u_mmu_o_cp0_is_mtc0 <= 1'b0;
	u_mmu_o_cp0_reg_address <= 5'b0;
	u_mmu_o_cp0_reg_data <= 32'bz;
	u_mmu_o_cpu_stall <= 1'b1;
	u_mmu_o_sdrctnr_host_ld_mode <= 1'b0;
	u_mmu_o_sdrctnr_stb <= 1'b0;
	u_mmu_o_sdrctnr_cyc <= 1'b0;
	u_mmu_o_sdrctnr_we <= 1'b0;
	u_mmu_o_sdrctnr_sel <= 4'b0;
	u_mmu_o_sdrctnr_addr <= 32'bz;
	u_mmu_o_sdrctnr_data <= 32'bz;

Table 7.2.3: Output for each state in MMU protocol

Chapter 8: Verification Specification

8.1: Test Plan of Memory Unit

Test Case	Expected Result
Load Page Table and Page to SDRAM.	• Observe from SDRAM read/write transcript to ensure the data had been successfully written into SDRAM.
Instruction TLB Miss	• Instruction MMU read First Level Page Table Entry. • Instruction MMU read Second Level Page Table Enrty. • IMMU stall signal deasserted.
Data TLB Miss	• Data MMU read First Level Page Table Entry. • Data MMU read Second Level Page Table Enrty. • DMMU stall signal deasserted.
Instruction Cache Miss	• Instruction Cache sends address to read from SDRAM. • SDRAM response by sending back data and acknowledge signal. • Repeat Step 1 and 2 for 8 times. • Instruction output from instruction cache.
Data Cache Miss	• Data Cache sends address to read from SDRAM. • SDRAM response by sending back data and acknowledge signal. • Repeat Step 1 and 2 for 8 times. • Data output from data cache.

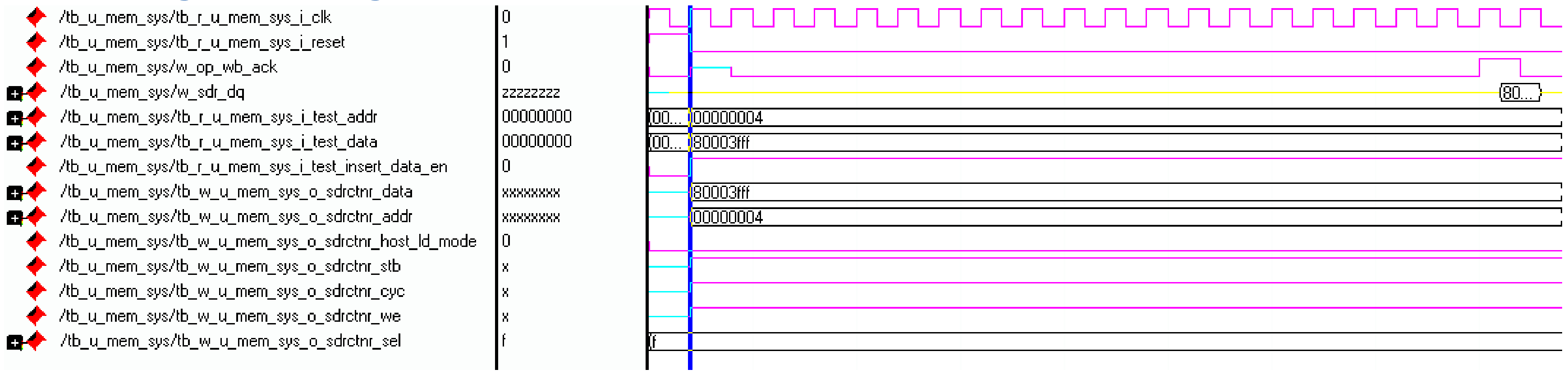
Table 8.1.1: Test Plan of Memory Unit

8.1.1: Test Procedure

1. System reset.
2. Porting appropriate data to CP0 registers.
3. Insert data into SDRAM by using the test signal, *u_mem_sys_test_insert_data_en*, *u_mem_sys_i_test_data* and *u_mem_sys_i_test_addr*.
    - Atleast 3 data needed to get the memory system run.
        1. First level Page Table.
        2. Second level Page Table.
        3. 8 sequential data to be read by cache when cache misses.
4. System reset and let the memory system run itself.

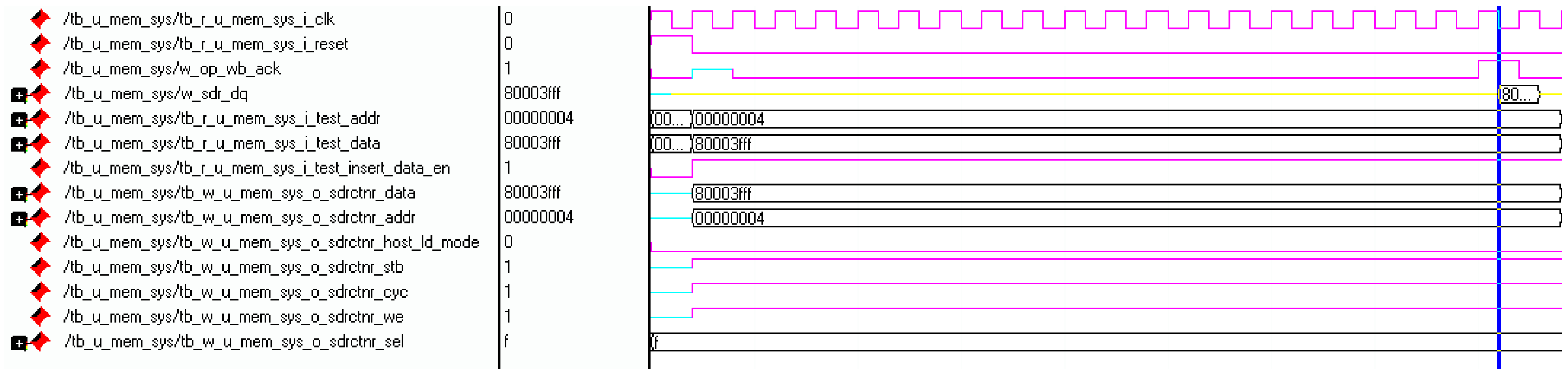
8.2: Simulation Result for Memory System

8.2.1: Load Page Table and Page to SDRAM



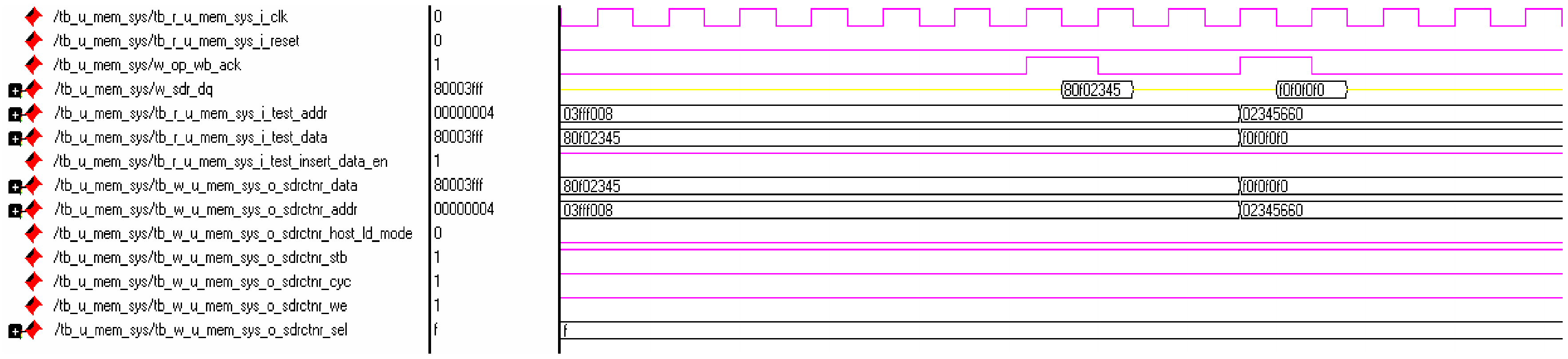
```
/tb_u_mem_sys/tb_r_u_mem_sys_i_clk                   0
/tb_u_mem_sys/tb_r_u_mem_sys_i_reset                 1
/tb_u_mem_sys/w_op_wb_ack                            0
/tb_u_mem_sys/w_sdr_dq                               zzzzzzzz
/tb_u_mem_sys/tb_r_u_mem_sys_i_test_addr             00000000
/tb_u_mem_sys/tb_r_u_mem_sys_i_test_data             00000000
/tb_u_mem_sys/tb_r_u_mem_sys_i_test_insert_data_en   0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_data          xxxxxxxx
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_addr          xxxxxxxx
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_host_ld_mode  0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_stb           x
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_cyc           x
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_we            x
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_sel           f
```

Figure 8.2.1: System Reset, follow by loading First Level Page Table Entry into SDRAM.



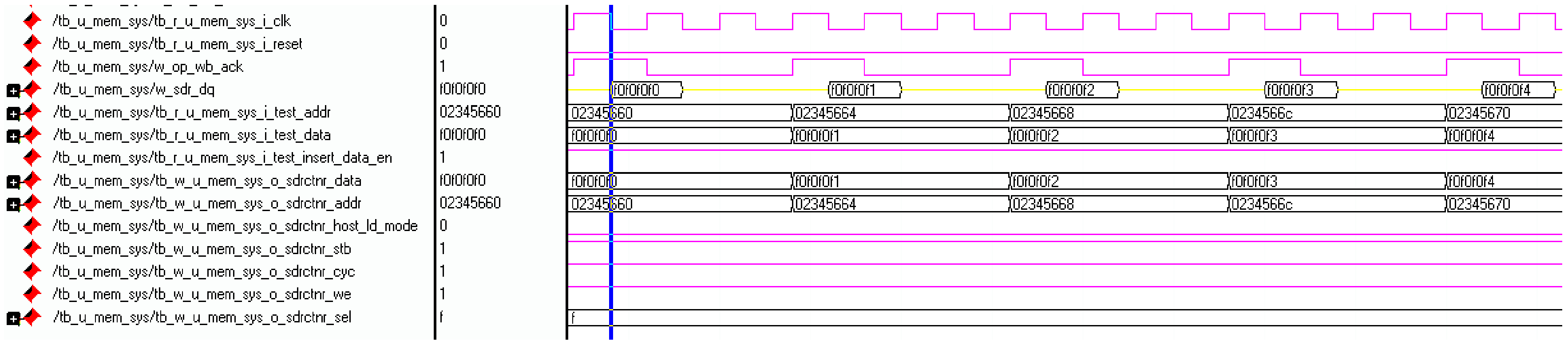
```
/tb_u_mem_sys/tb_r_u_mem_sys_i_clk                   0
/tb_u_mem_sys/tb_r_u_mem_sys_i_reset                 0
/tb_u_mem_sys/w_op_wb_ack                            1
/tb_u_mem_sys/w_sdr_dq                               80003fff
/tb_u_mem_sys/tb_r_u_mem_sys_i_test_addr             00000004
/tb_u_mem_sys/tb_r_u_mem_sys_i_test_data             80003fff
/tb_u_mem_sys/tb_r_u_mem_sys_i_test_insert_data_en   1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_data          80003fff
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_addr          00000004
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_host_ld_mode  0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_stb           1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_cyc           1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_we            1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_sel           f
```

Figure 8.2.2: After 19 clock cycles, First Level Page Table Entry successfully loaded into SDRAM.



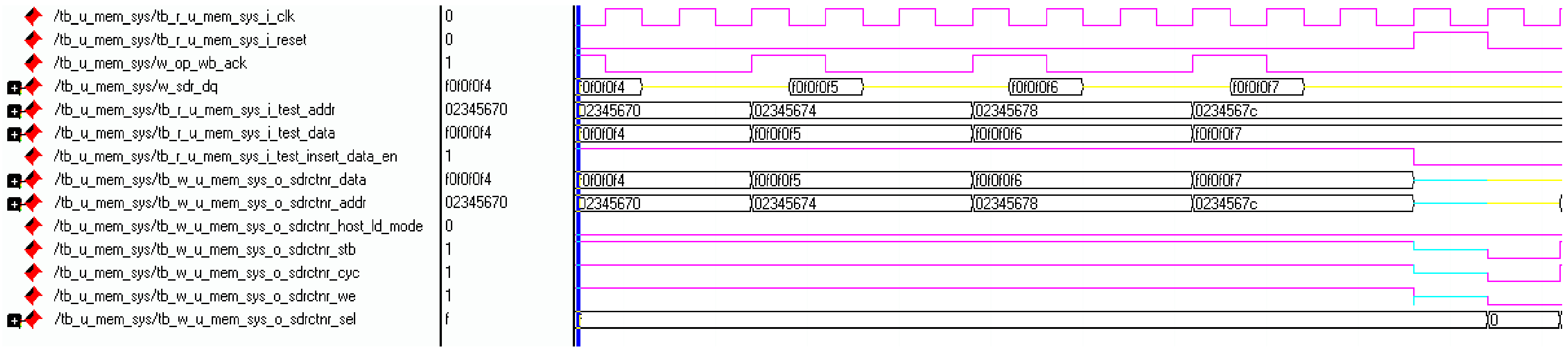
```
/tb_u_mem_sys/tb_r_u_mem_sys_i_clk                        0
/tb_u_mem_sys/tb_r_u_mem_sys_i_reset                      0
/tb_u_mem_sys/w_op_wb_ack                                 1
/tb_u_mem_sys/w_sdr_dq                                    80003fff
/tb_u_mem_sys/tb_r_u_mem_sys_i_test_addr                  00000004
/tb_u_mem_sys/tb_r_u_mem_sys_i_test_data                  80003fff
/tb_u_mem_sys/tb_r_u_mem_sys_i_test_insert_data_en        1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_data               80003fff
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_addr               00000004
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_host_ld_mode       0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_stb                1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_cyc                1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_we                 1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_sel                f
```

Figure 8.2.3: Loading Second Level Page Table Entry into SDRAM.



```
/tb_u_mem_sys/tb_r_u_mem_sys_i_clk                        0
/tb_u_mem_sys/tb_r_u_mem_sys_i_reset                      0
/tb_u_mem_sys/w_op_wb_ack                                 1
/tb_u_mem_sys/w_sdr_dq                                    f0f0f0f0
/tb_u_mem_sys/tb_r_u_mem_sys_i_test_addr                  02345660
/tb_u_mem_sys/tb_r_u_mem_sys_i_test_data                  f0f0f0f0
/tb_u_mem_sys/tb_r_u_mem_sys_i_test_insert_data_en        1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_data               f0f0f0f0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_addr               02345660
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_host_ld_mode       0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_stb                1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_cyc                1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_we                 1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_sel                f
```

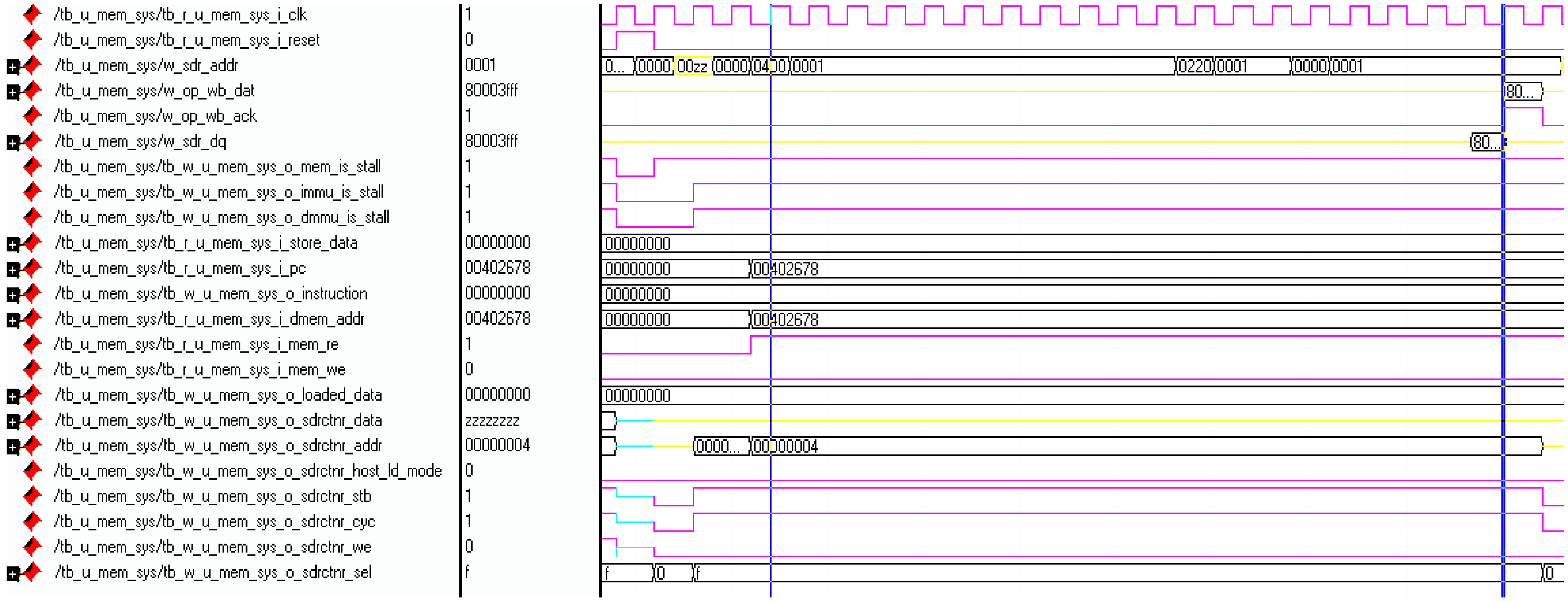
Figure 8.2.4: Loading Pages into SDRAM (Part 1).



```
/tb_u_mem_sys/tb_r_u_mem_sys_i_clk                      0
/tb_u_mem_sys/tb_r_u_mem_sys_i_reset                    0
/tb_u_mem_sys/w_op_wb_ack                               1
/tb_u_mem_sys/w_sdr_dq                                  f0f0f0f4
/tb_u_mem_sys/tb_r_u_mem_sys_i_test_addr                02345670
/tb_u_mem_sys/tb_r_u_mem_sys_i_test_data                f0f0f0f4
/tb_u_mem_sys/tb_r_u_mem_sys_i_test_insert_data_en      1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_data             f0f0f0f4
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_addr             02345670
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_host_ld_mode     0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_stb              1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_cyc              1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_we               1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_sel              f
```

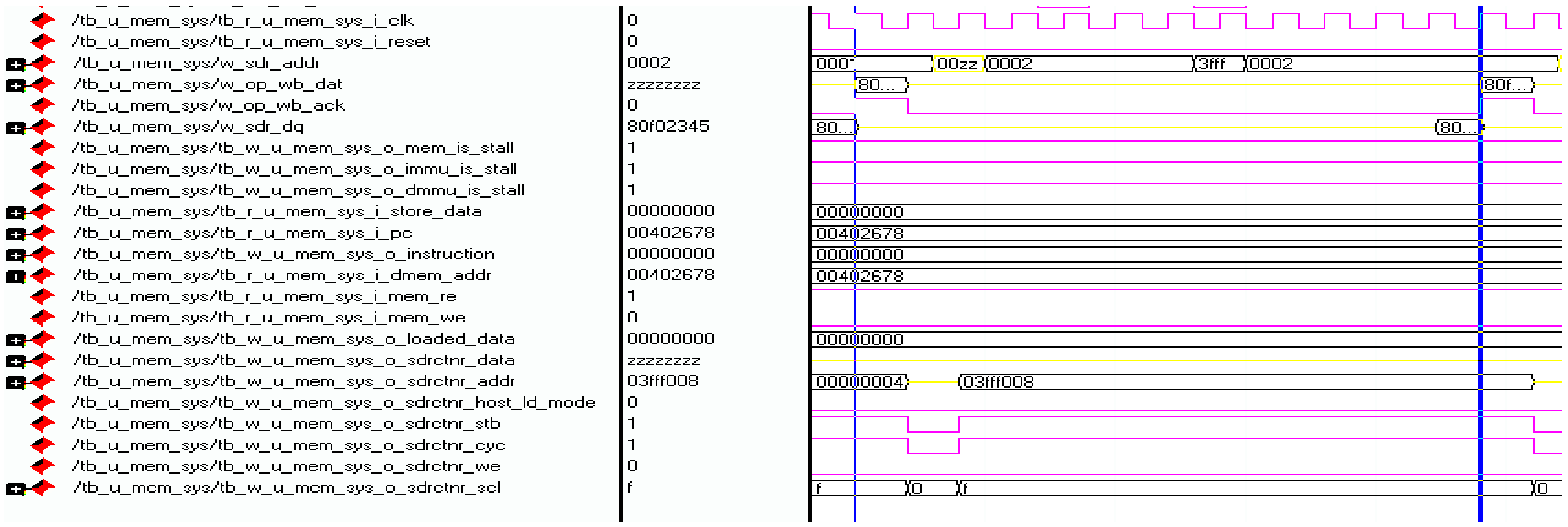
Figure 8.2.5: Loading Pages into SDRAM (Part 2) follow by System Reset to initiate the system.

8.2.2: ITLB MISS



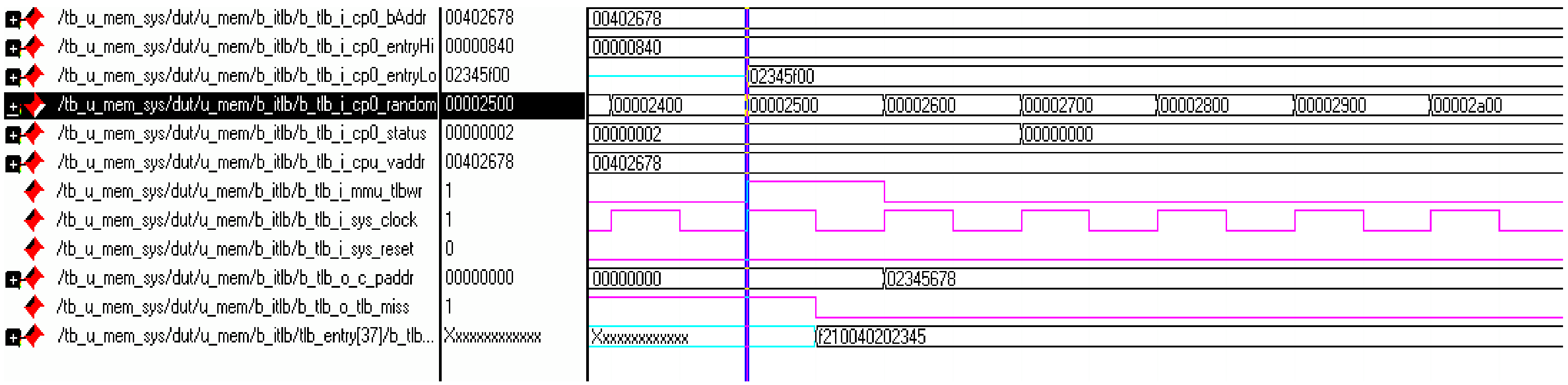
```
/tb_u_mem_sys/tb_r_u_mem_sys_i_clk                    1
/tb_u_mem_sys/tb_r_u_mem_sys_i_reset                  0
/tb_u_mem_sys/w_sdr_addr                              0001
/tb_u_mem_sys/w_op_wb_dat                             80003fff
/tb_u_mem_sys/w_op_wb_ack                             1
/tb_u_mem_sys/w_sdr_dq                                80003fff
/tb_u_mem_sys/tb_w_u_mem_sys_o_mem_is_stall           1
/tb_u_mem_sys/tb_w_u_mem_sys_o_immu_is_stall          1
/tb_u_mem_sys/tb_w_u_mem_sys_o_dmmu_is_stall          1
/tb_u_mem_sys/tb_r_u_mem_sys_i_store_data             00000000
/tb_u_mem_sys/tb_r_u_mem_sys_i_pc                     00402678
/tb_u_mem_sys/tb_w_u_mem_sys_o_instruction            00000000
/tb_u_mem_sys/tb_r_u_mem_sys_i_dmem_addr              00402678
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_re                 1
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_we                 0
/tb_u_mem_sys/tb_w_u_mem_sys_o_loaded_data            00000000
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_data           zzzzzzzz
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_addr           00000004
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_host_ld_mode   0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_stb            1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_cyc            1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_we             0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_sel            f
```

Figure 8.2.6: ITLB miss occurs, IMMU take over to fetch First Level Page Table Entry from SDRAM. (Take 19 clock cycles)



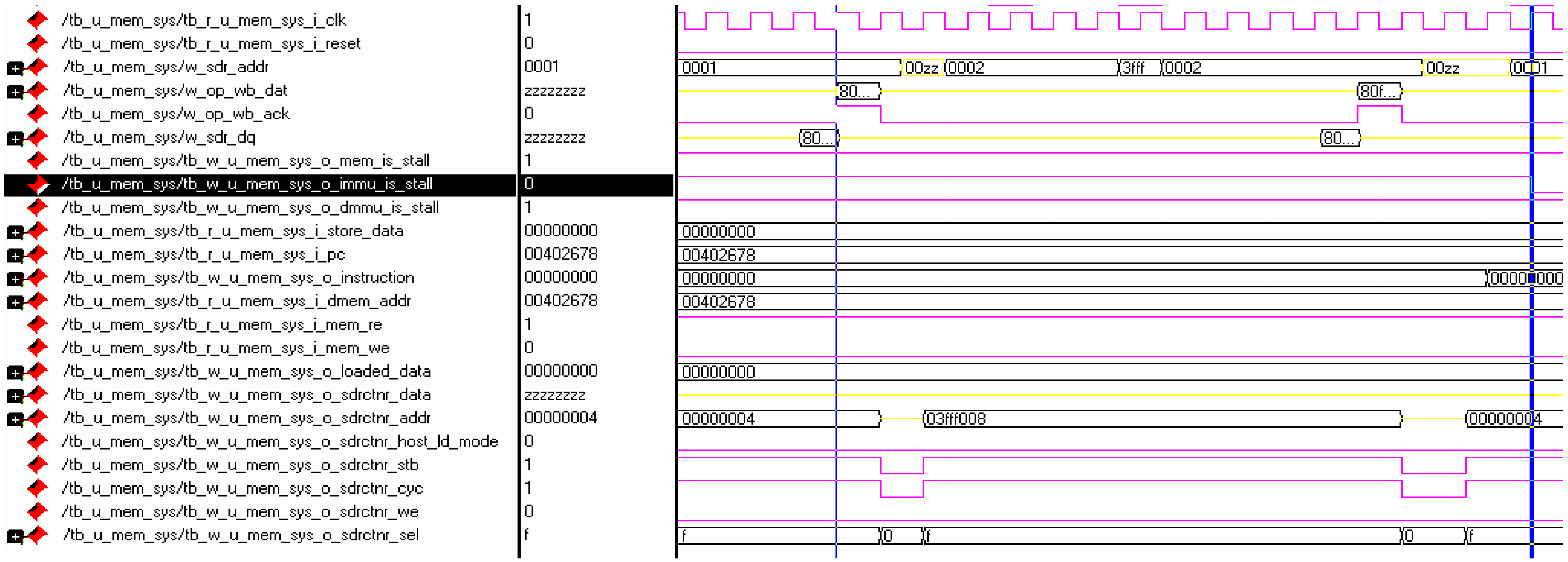
```
/tb_u_mem_sys/tb_r_u_mem_sys_i_clk                     0
/tb_u_mem_sys/tb_r_u_mem_sys_i_reset                   0
/tb_u_mem_sys/w_sdr_addr                               0002
/tb_u_mem_sys/w_op_wb_dat                              zzzzzzzz
/tb_u_mem_sys/w_op_wb_ack                              0
/tb_u_mem_sys/w_sdr_dq                                 80f02345
/tb_u_mem_sys/tb_w_u_mem_sys_o_mem_is_stall            1
/tb_u_mem_sys/tb_w_u_mem_sys_o_immu_is_stall           1
/tb_u_mem_sys/tb_w_u_mem_sys_o_dmmu_is_stall           1
/tb_u_mem_sys/tb_r_u_mem_sys_i_store_data              00000000
/tb_u_mem_sys/tb_r_u_mem_sys_i_pc                      00402678
/tb_u_mem_sys/tb_w_u_mem_sys_o_instruction             00000000
/tb_u_mem_sys/tb_r_u_mem_sys_i_dmem_addr               00402678
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_re                  1
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_we                  0
/tb_u_mem_sys/tb_w_u_mem_sys_o_loaded_data             00000000
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_data            zzzzzzzz
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_addr            03fff008
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_host_ld_mode    0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_stb             1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_cyc             1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_we              0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_sel             f
```

Figure 8.2.7: IMMU take over to fetch Second Level Page Table Entry from SDRAM. (Take 12 clock cycles)



```
/tb_u_mem_sys/dut/u_mem/b_itlb/b_tlb_i_cp0_bAddr     00402678
/tb_u_mem_sys/dut/u_mem/b_itlb/b_tlb_i_cp0_entryHi   00000840
/tb_u_mem_sys/dut/u_mem/b_itlb/b_tlb_i_cp0_entryLo   02345f00
/tb_u_mem_sys/dut/u_mem/b_itlb/b_tlb_i_cp0_random    00002500
/tb_u_mem_sys/dut/u_mem/b_itlb/b_tlb_i_cp0_status    00000002
/tb_u_mem_sys/dut/u_mem/b_itlb/b_tlb_i_cpu_vaddr     00402678
/tb_u_mem_sys/dut/u_mem/b_itlb/b_tlb_i_mmu_tlbwr     1
/tb_u_mem_sys/dut/u_mem/b_itlb/b_tlb_i_sys_clock     1
/tb_u_mem_sys/dut/u_mem/b_itlb/b_tlb_i_sys_reset     0
/tb_u_mem_sys/dut/u_mem/b_itlb/b_tlb_o_c_paddr       00000000
/tb_u_mem_sys/dut/u_mem/b_itlb/b_tlb_o_tlb_miss      1
/tb_u_mem_sys/dut/u_mem/b_itlb/tlb_entry[37]/b_tlb...  Xxxxxxxxxxxx
```

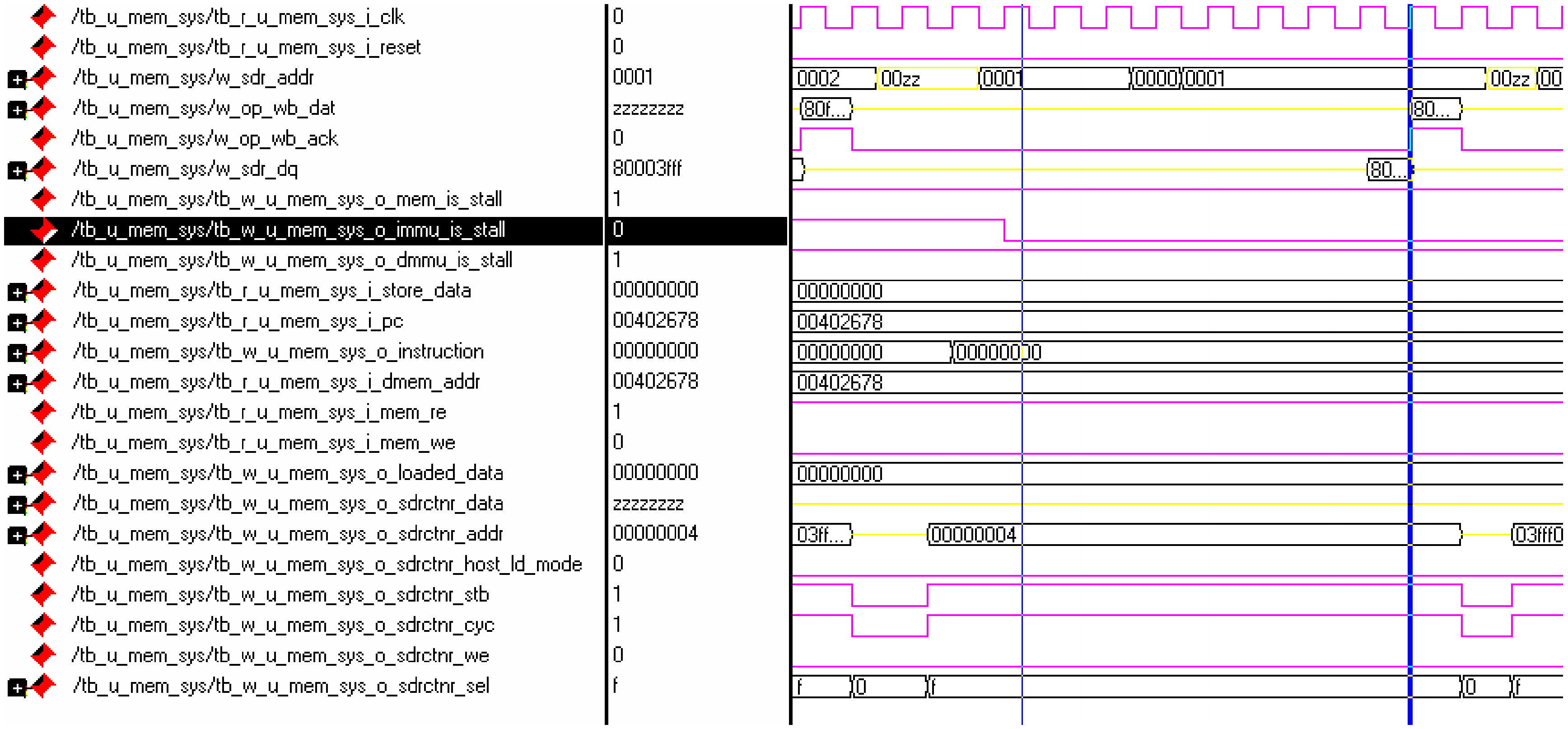
Figure 8.2.8: Updating ITLB Entry.



```
/tb_u_mem_sys/tb_r_u_mem_sys_i_clk                      1
/tb_u_mem_sys/tb_r_u_mem_sys_i_reset                    0
/tb_u_mem_sys/w_sdr_addr                                0001
/tb_u_mem_sys/w_op_wb_dat                               zzzzzzzz
/tb_u_mem_sys/w_op_wb_ack                               0
/tb_u_mem_sys/w_sdr_dq                                  zzzzzzzz
/tb_u_mem_sys/tb_w_u_mem_sys_o_mem_is_stall             1
/tb_u_mem_sys/tb_w_u_mem_sys_o_immu_is_stall            0
/tb_u_mem_sys/tb_w_u_mem_sys_o_dmmu_is_stall            1
/tb_u_mem_sys/tb_r_u_mem_sys_i_store_data               00000000
/tb_u_mem_sys/tb_r_u_mem_sys_i_pc                       00402678
/tb_u_mem_sys/tb_w_u_mem_sys_o_instruction              00000000
/tb_u_mem_sys/tb_r_u_mem_sys_i_dmem_addr                00402678
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_re                   1
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_we                   0
/tb_u_mem_sys/tb_w_u_mem_sys_o_loaded_data              00000000
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_data             zzzzzzzz
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_addr             00000004
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_host_ld_mode     0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_stb              1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_cyc              1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_we               0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_sel              f
```

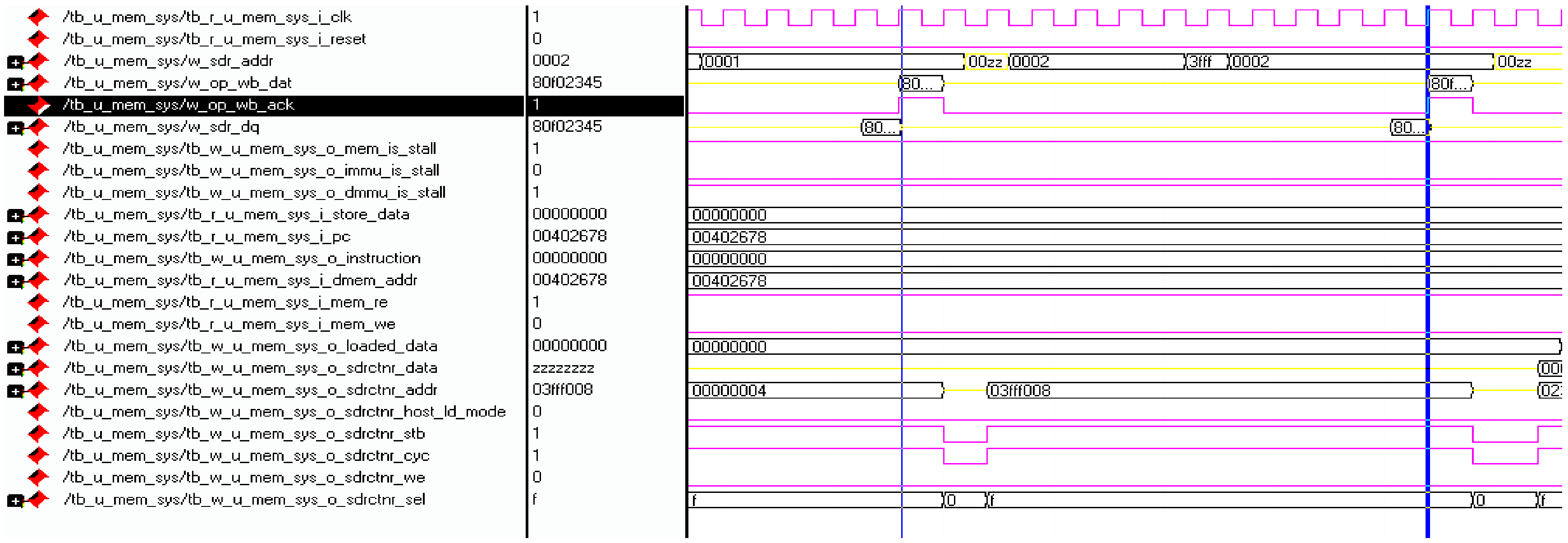
Figure 8.2.9: ITLB HIT.

8.2.3: DTLB MISS



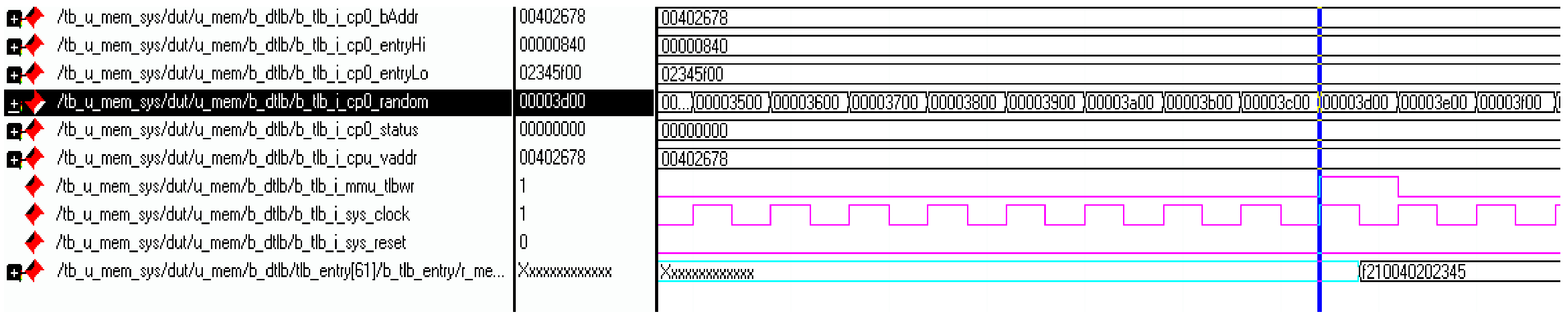
```
/tb_u_mem_sys/tb_r_u_mem_sys_i_clk                    0
/tb_u_mem_sys/tb_r_u_mem_sys_i_reset                  0
/tb_u_mem_sys/w_sdr_addr                              0001
/tb_u_mem_sys/w_op_wb_dat                             zzzzzzzz
/tb_u_mem_sys/w_op_wb_ack                             0
/tb_u_mem_sys/w_sdr_dq                                80003fff
/tb_u_mem_sys/tb_w_u_mem_sys_o_mem_is_stall           1
/tb_u_mem_sys/tb_w_u_mem_sys_o_immu_is_stall          0
/tb_u_mem_sys/tb_w_u_mem_sys_o_dmmu_is_stall          1
/tb_u_mem_sys/tb_r_u_mem_sys_i_store_data             00000000
/tb_u_mem_sys/tb_r_u_mem_sys_i_pc                     00402678
/tb_u_mem_sys/tb_w_u_mem_sys_o_instruction            00000000
/tb_u_mem_sys/tb_r_u_mem_sys_i_dmem_addr              00402678
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_re                 1
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_we                 0
/tb_u_mem_sys/tb_w_u_mem_sys_o_loaded_data            00000000
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_data           zzzzzzzz
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_addr           00000004
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_host_ld_mode   0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_stb            1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_cyc            1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_we             0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_sel            f
```

Figure 8.2.10: DTLB miss occurs, DMMU take over to fetch First Level Page Table Entry from SDRAM. (Take 9 clock cycles)



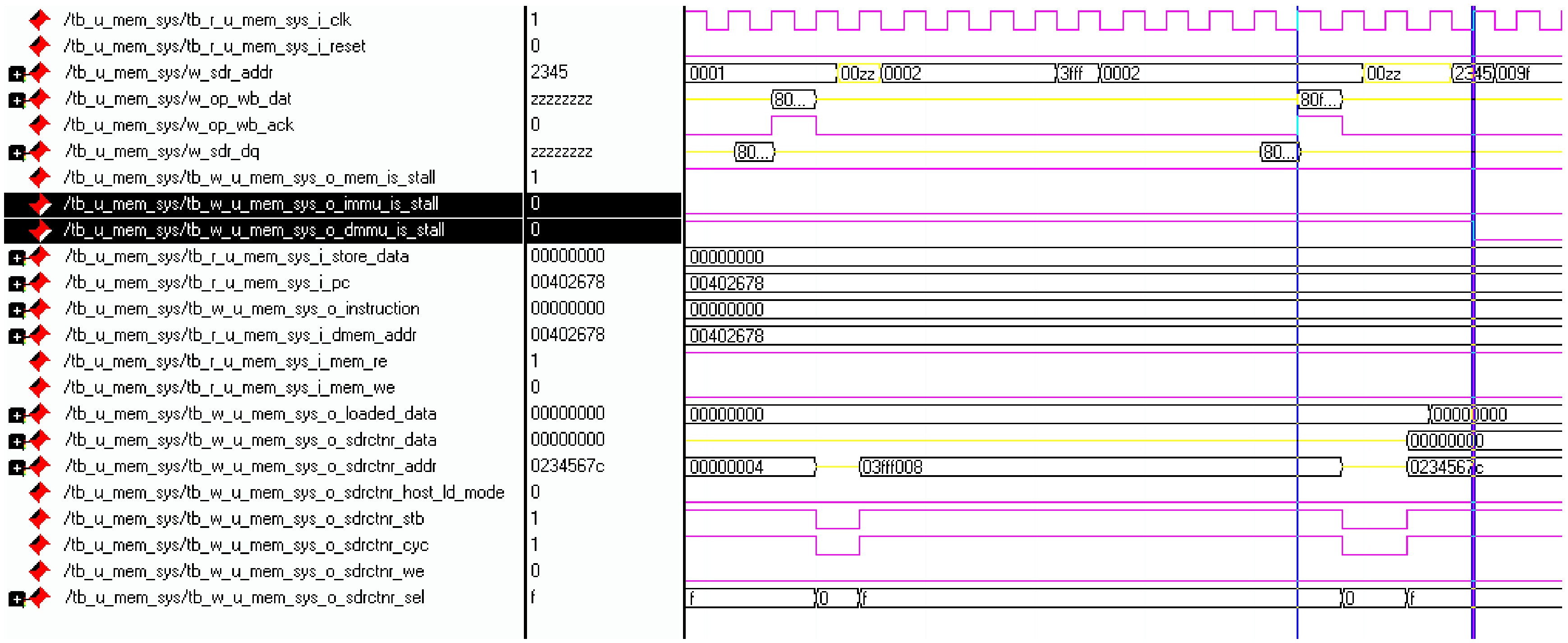
/tb_u_mem_sys/tb_r_u_mem_sys_i_clk	1
/tb_u_mem_sys/tb_r_u_mem_sys_i_reset	0
/tb_u_mem_sys/w_sdr_addr	0002
/tb_u_mem_sys/w_op_wb_dat	80f02345
/tb_u_mem_sys/w_op_wb_ack	1
/tb_u_mem_sys/w_sdr_dq	80f02345
/tb_u_mem_sys/tb_w_u_mem_sys_o_mem_is_stall	1
/tb_u_mem_sys/tb_w_u_mem_sys_o_immu_is_stall	0
/tb_u_mem_sys/tb_w_u_mem_sys_o_dmmu_is_stall	1
/tb_u_mem_sys/tb_r_u_mem_sys_i_store_data	00000000
/tb_u_mem_sys/tb_r_u_mem_sys_i_pc	00402678
/tb_u_mem_sys/tb_w_u_mem_sys_o_instruction	00000000
/tb_u_mem_sys/tb_r_u_mem_sys_i_dmem_addr	00402678
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_re	1
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_we	0
/tb_u_mem_sys/tb_w_u_mem_sys_o_loaded_data	00000000
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_data	zzzzzzzz
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_addr	03fff008
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_host_ld_mode	0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_stb	1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_cyc	1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_we	0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_sel	f

Figure 8.2.11: DMMU take over to fetch Second Level Page Table Entry from SDRAM. (Take 9 clock cycles)



/tb_u_mem_sys/dut/u_mem/b_dtlb/b_tlb_i_cp0_bAddr	00402678
/tb_u_mem_sys/dut/u_mem/b_dtlb/b_tlb_i_cp0_entryHi	00000840
/tb_u_mem_sys/dut/u_mem/b_dtlb/b_tlb_i_cp0_entryLo	02345f00
/tb_u_mem_sys/dut/u_mem/b_dtlb/b_tlb_i_cp0_random	00003d00
/tb_u_mem_sys/dut/u_mem/b_dtlb/b_tlb_i_cp0_status	00000000
/tb_u_mem_sys/dut/u_mem/b_dtlb/b_tlb_i_cpu_vaddr	00402678
/tb_u_mem_sys/dut/u_mem/b_dtlb/b_tlb_i_mmu_tlbwr	1
/tb_u_mem_sys/dut/u_mem/b_dtlb/b_tlb_i_sys_clock	1
/tb_u_mem_sys/dut/u_mem/b_dtlb/b_tlb_i_sys_reset	0
/tb_u_mem_sys/dut/u_mem/b_dtlb/tlb_entry[61]/b_tlb_entry/r_me...	Xxxxxxxxxxxxx

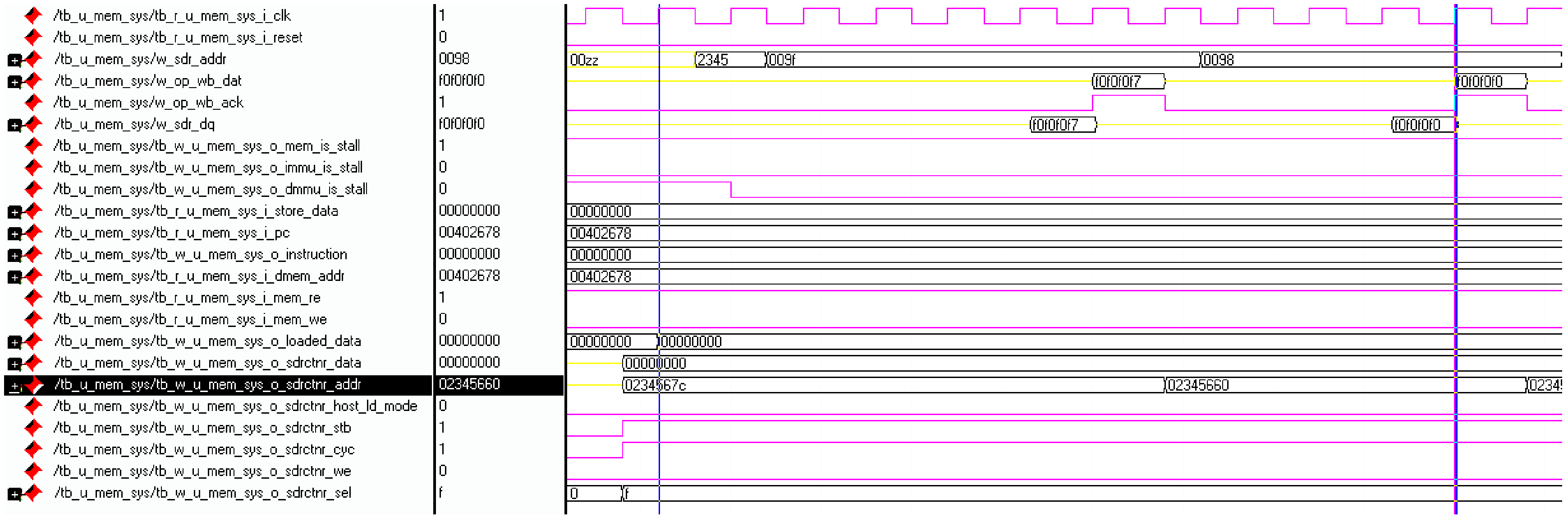
Figure8.2.12: Updating DTLB Entry.



```
/tb_u_mem_sys/tb_r_u_mem_sys_i_clk                    1
/tb_u_mem_sys/tb_r_u_mem_sys_i_reset                  0
/tb_u_mem_sys/w_sdr_addr                              2345
/tb_u_mem_sys/w_op_wb_dat                             zzzzzzzz
/tb_u_mem_sys/w_op_wb_ack                             0
/tb_u_mem_sys/w_sdr_dq                                zzzzzzzz
/tb_u_mem_sys/tb_w_u_mem_sys_o_mem_is_stall           1
/tb_u_mem_sys/tb_w_u_mem_sys_o_immu_is_stall          0
/tb_u_mem_sys/tb_w_u_mem_sys_o_dmmu_is_stall          0
/tb_u_mem_sys/tb_r_u_mem_sys_i_store_data             00000000
/tb_u_mem_sys/tb_r_u_mem_sys_i_pc                     00402678
/tb_u_mem_sys/tb_w_u_mem_sys_o_instruction            00000000
/tb_u_mem_sys/tb_r_u_mem_sys_i_dmem_addr              00402678
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_re                 1
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_we                 0
/tb_u_mem_sys/tb_w_u_mem_sys_o_loaded_data            00000000
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_data           00000000
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_addr           0234567c
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_host_ld_mode   0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_stb            1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_cyc            1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_we             0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_sel            f
```

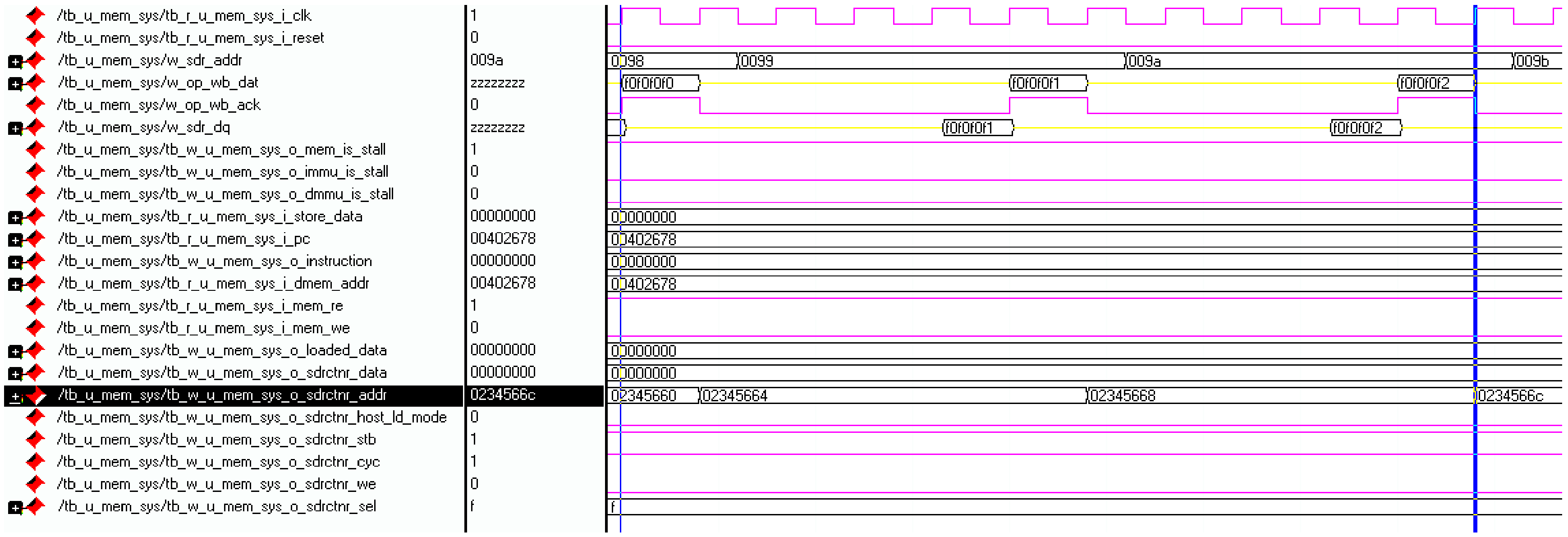
Figure 8.2.13: DTLB HIT.

8.2.4: Instruction Cache Miss



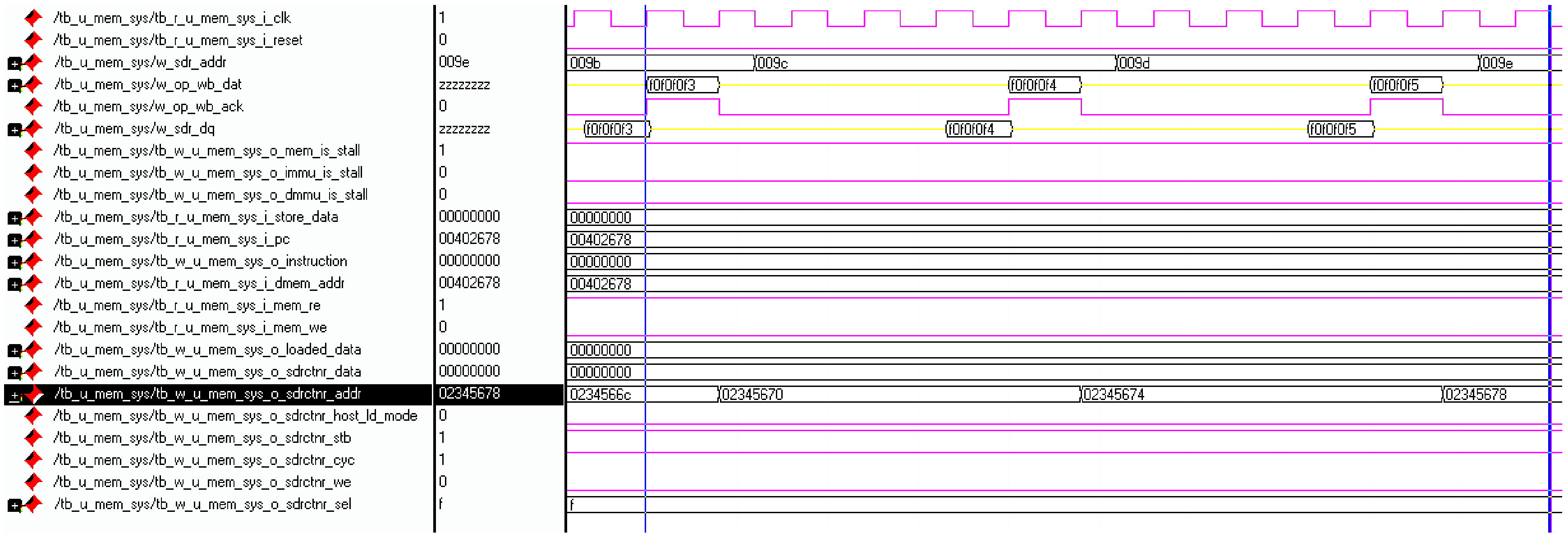
/tb_u_mem_sys/tb_r_u_mem_sys_i_clk	1
/tb_u_mem_sys/tb_r_u_mem_sys_i_reset	0
/tb_u_mem_sys/w_sdr_addr	0098
/tb_u_mem_sys/w_op_wb_dat	f0f0f0f0
/tb_u_mem_sys/w_op_wb_ack	1
/tb_u_mem_sys/w_sdr_dq	f0f0f0f0
/tb_u_mem_sys/tb_w_u_mem_sys_o_mem_is_stall	1
/tb_u_mem_sys/tb_w_u_mem_sys_o_immu_is_stall	0
/tb_u_mem_sys/tb_w_u_mem_sys_o_dmmu_is_stall	0
/tb_u_mem_sys/tb_r_u_mem_sys_i_store_data	00000000
/tb_u_mem_sys/tb_r_u_mem_sys_i_pc	00402678
/tb_u_mem_sys/tb_w_u_mem_sys_o_instruction	00000000
/tb_u_mem_sys/tb_r_u_mem_sys_i_dmem_addr	00402678
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_re	1
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_we	0
/tb_u_mem_sys/tb_w_u_mem_sys_o_loaded_data	00000000
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_data	00000000
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_addr	02345660
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_host_ld_mode	0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_stb	1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_cyc	1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_we	0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_sel	f

Figure 8.2.14: Instruction cache misses, transferring block from SDRAM (Part 1). (Take 6 clock cycles for first access)



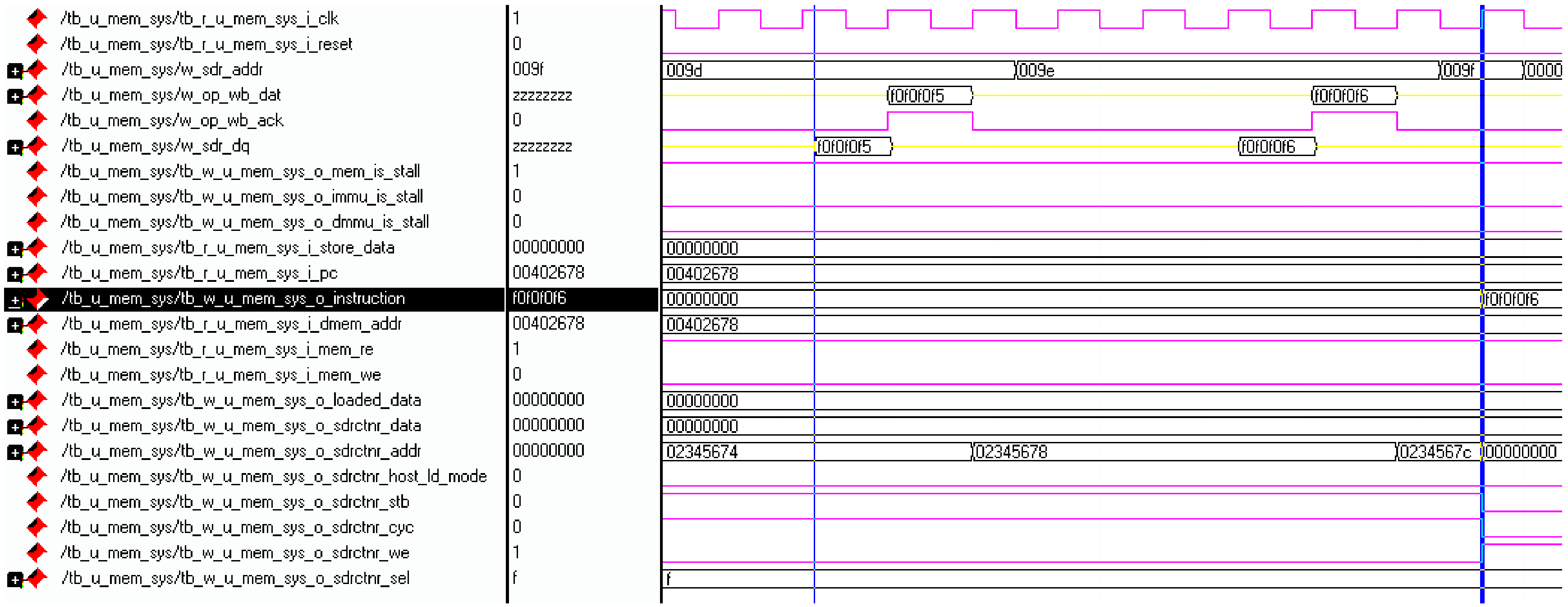
/tb_u_mem_sys/tb_r_u_mem_sys_i_clk	1
/tb_u_mem_sys/tb_r_u_mem_sys_i_reset	0
/tb_u_mem_sys/w_sdr_addr	009a
/tb_u_mem_sys/w_op_wb_dat	zzzzzzzz
/tb_u_mem_sys/w_op_wb_ack	0
/tb_u_mem_sys/w_sdr_dq	zzzzzzzz
/tb_u_mem_sys/tb_w_u_mem_sys_o_mem_is_stall	1
/tb_u_mem_sys/tb_w_u_mem_sys_o_immu_is_stall	0
/tb_u_mem_sys/tb_w_u_mem_sys_o_dmmu_is_stall	0
/tb_u_mem_sys/tb_r_u_mem_sys_i_store_data	00000000
/tb_u_mem_sys/tb_r_u_mem_sys_i_pc	00402678
/tb_u_mem_sys/tb_w_u_mem_sys_o_instruction	00000000
/tb_u_mem_sys/tb_r_u_mem_sys_i_dmem_addr	00402678
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_re	1
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_we	0
/tb_u_mem_sys/tb_w_u_mem_sys_o_loaded_data	00000000
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_data	00000000
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_addr	0234566c
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_host_ld_mode	0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_stb	1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_cyc	1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_we	0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_sel	f

Figure 8.2.15: Instruction cache misses, transferring block from SDRAM (Part 2). (Take 4 clock cycles for subsequence access)



```
/tb_u_mem_sys/tb_r_u_mem_sys_i_clk                    1
/tb_u_mem_sys/tb_r_u_mem_sys_i_reset                  0
/tb_u_mem_sys/w_sdr_addr                              009e
/tb_u_mem_sys/w_op_wb_dat                             zzzzzzzz
/tb_u_mem_sys/w_op_wb_ack                             0
/tb_u_mem_sys/w_sdr_dq                                zzzzzzzz
/tb_u_mem_sys/tb_w_u_mem_sys_o_mem_is_stall           1
/tb_u_mem_sys/tb_w_u_mem_sys_o_immu_is_stall          0
/tb_u_mem_sys/tb_w_u_mem_sys_o_dmmu_is_stall          0
/tb_u_mem_sys/tb_r_u_mem_sys_i_store_data             00000000
/tb_u_mem_sys/tb_r_u_mem_sys_i_pc                     00402678
/tb_u_mem_sys/tb_w_u_mem_sys_o_instruction            00000000
/tb_u_mem_sys/tb_r_u_mem_sys_i_dmem_addr              00402678
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_re                 1
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_we                 0
/tb_u_mem_sys/tb_w_u_mem_sys_o_loaded_data            00000000
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_data           00000000
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_addr           02345678
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_host_ld_mode   0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_stb            1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_cyc            1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_we             0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_sel            f
```

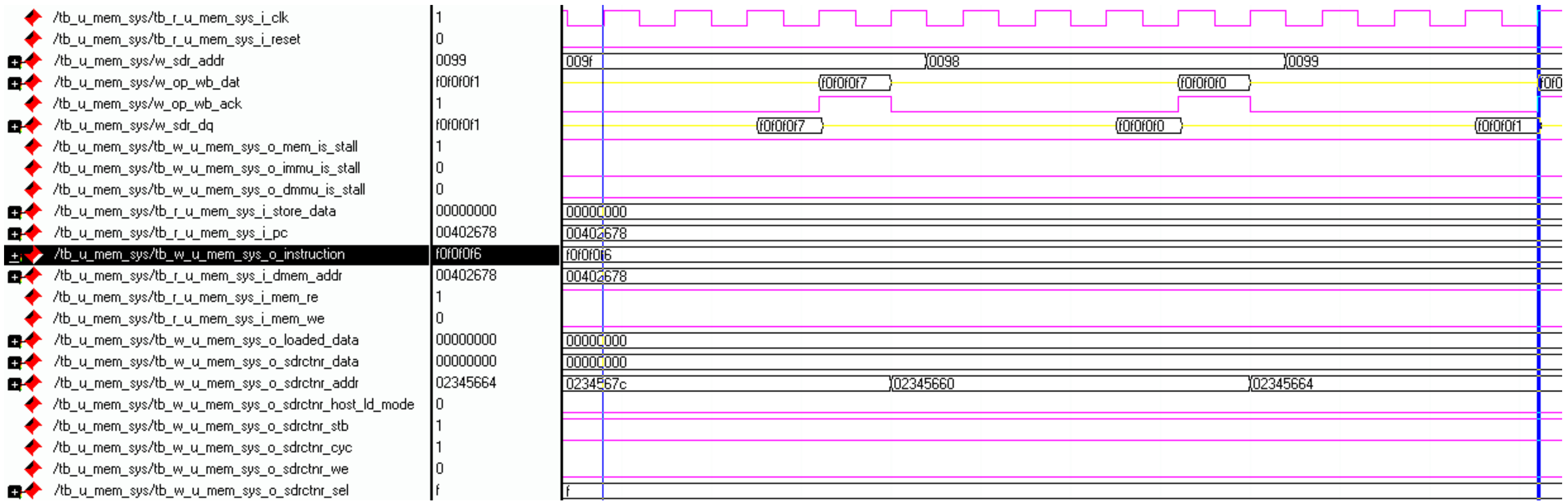
Figure 8.2.16: Instruction cache misses, transferring block from SDRAM (Part 3). (Take 4 clock cycles for subsequence access)



```
/tb_u_mem_sys/tb_r_u_mem_sys_i_clk                      1
/tb_u_mem_sys/tb_r_u_mem_sys_i_reset                    0
/tb_u_mem_sys/w_sdr_addr                                009f
/tb_u_mem_sys/w_op_wb_dat                               zzzzzzzz
/tb_u_mem_sys/w_op_wb_ack                               0
/tb_u_mem_sys/w_sdr_dq                                  zzzzzzzz
/tb_u_mem_sys/tb_w_u_mem_sys_o_mem_is_stall             1
/tb_u_mem_sys/tb_w_u_mem_sys_o_immu_is_stall            0
/tb_u_mem_sys/tb_w_u_mem_sys_o_dmmu_is_stall            0
/tb_u_mem_sys/tb_r_u_mem_sys_i_store_data               00000000
/tb_u_mem_sys/tb_r_u_mem_sys_i_pc                       00402678
/tb_u_mem_sys/tb_w_u_mem_sys_o_instruction              f0f0f0f6
/tb_u_mem_sys/tb_r_u_mem_sys_i_dmem_addr                00402678
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_re                   1
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_we                   0
/tb_u_mem_sys/tb_w_u_mem_sys_o_loaded_data              00000000
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_data             00000000
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_addr             00000000
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_host_ld_mode     0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_stb              0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_cyc              0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_we               1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_sel              f
```

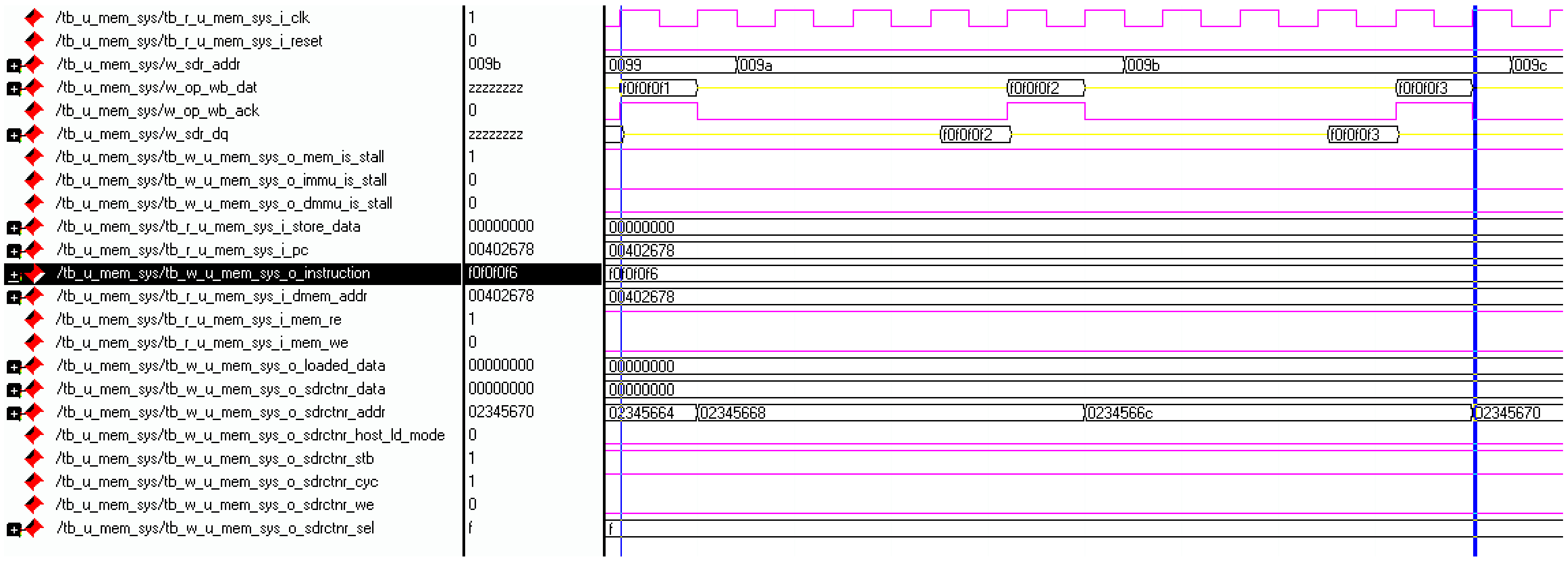
Figure 8.2.17: Instruction cache Hit, instruction is successfully read from cache.

8.2.5: Data Cache Miss



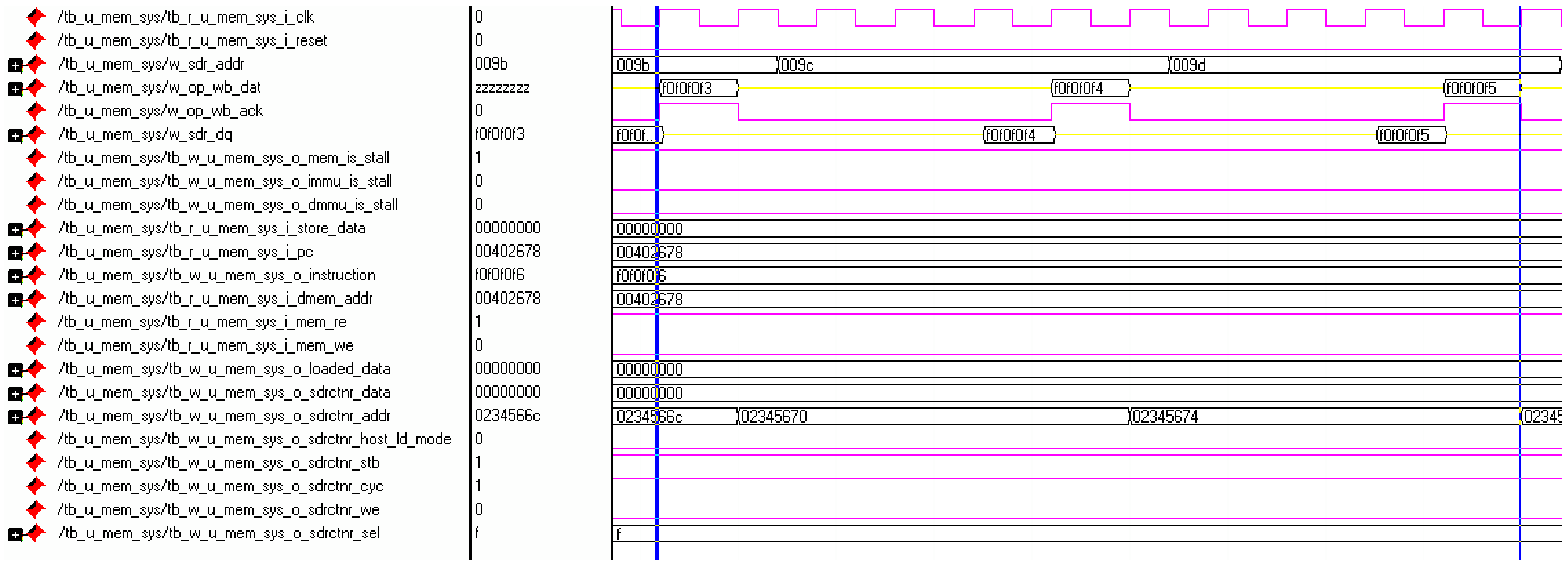
```
/tb_u_mem_sys/tb_r_u_mem_sys_i_clk                       1
/tb_u_mem_sys/tb_r_u_mem_sys_i_reset                     0
/tb_u_mem_sys/w_sdr_addr                                 0099
/tb_u_mem_sys/w_op_wb_dat                                f0f0f0f1
/tb_u_mem_sys/w_op_wb_ack                                1
/tb_u_mem_sys/w_sdr_dq                                   f0f0f0f1
/tb_u_mem_sys/tb_w_u_mem_sys_o_mem_is_stall              1
/tb_u_mem_sys/tb_w_u_mem_sys_o_immu_is_stall             0
/tb_u_mem_sys/tb_w_u_mem_sys_o_dmmu_is_stall             0
/tb_u_mem_sys/tb_r_u_mem_sys_i_store_data                00000000
/tb_u_mem_sys/tb_r_u_mem_sys_i_pc                        00402678
/tb_u_mem_sys/tb_w_u_mem_sys_o_instruction               f0f0f0f6
/tb_u_mem_sys/tb_r_u_mem_sys_i_dmem_addr                 00402678
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_re                    1
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_we                    0
/tb_u_mem_sys/tb_w_u_mem_sys_o_loaded_data               00000000
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_data              00000000
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_addr              02345664
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_host_ld_mode      0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_stb               1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_cyc               1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_we                0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_sel               f
```

Figure 8.2.18: Data cache misses, transferring block from SDRAM (Part 1). (Take 6 clock cycles for first access)



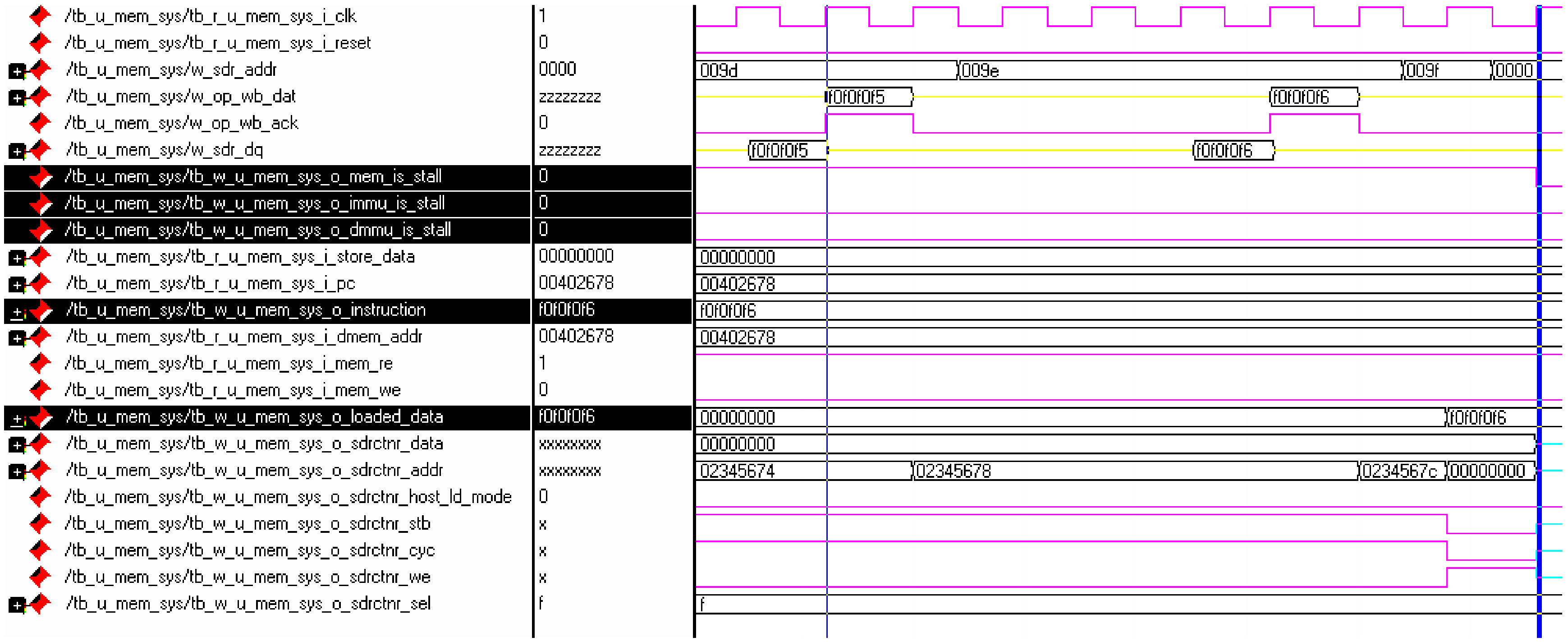
/tb_u_mem_sys/tb_r_u_mem_sys_i_clk	1
/tb_u_mem_sys/tb_r_u_mem_sys_i_reset	0
/tb_u_mem_sys/w_sdr_addr	009b
/tb_u_mem_sys/w_op_wb_dat	zzzzzzzz
/tb_u_mem_sys/w_op_wb_ack	0
/tb_u_mem_sys/w_sdr_dq	zzzzzzzz
/tb_u_mem_sys/tb_w_u_mem_sys_o_mem_is_stall	1
/tb_u_mem_sys/tb_w_u_mem_sys_o_immu_is_stall	0
/tb_u_mem_sys/tb_w_u_mem_sys_o_dmmu_is_stall	0
/tb_u_mem_sys/tb_r_u_mem_sys_i_store_data	00000000
/tb_u_mem_sys/tb_r_u_mem_sys_i_pc	00402678
/tb_u_mem_sys/tb_w_u_mem_sys_o_instruction	f0f0f0f6
/tb_u_mem_sys/tb_r_u_mem_sys_i_dmem_addr	00402678
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_re	1
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_we	0
/tb_u_mem_sys/tb_w_u_mem_sys_o_loaded_data	00000000
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_data	00000000
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_addr	02345670
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_host_ld_mode	0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_stb	1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_cyc	1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_we	0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_sel	f

Figure 8.2.19: Data cache misses, transferring block from SDRAM (Part 2). (Take 4 clock cycles for subsequence access)



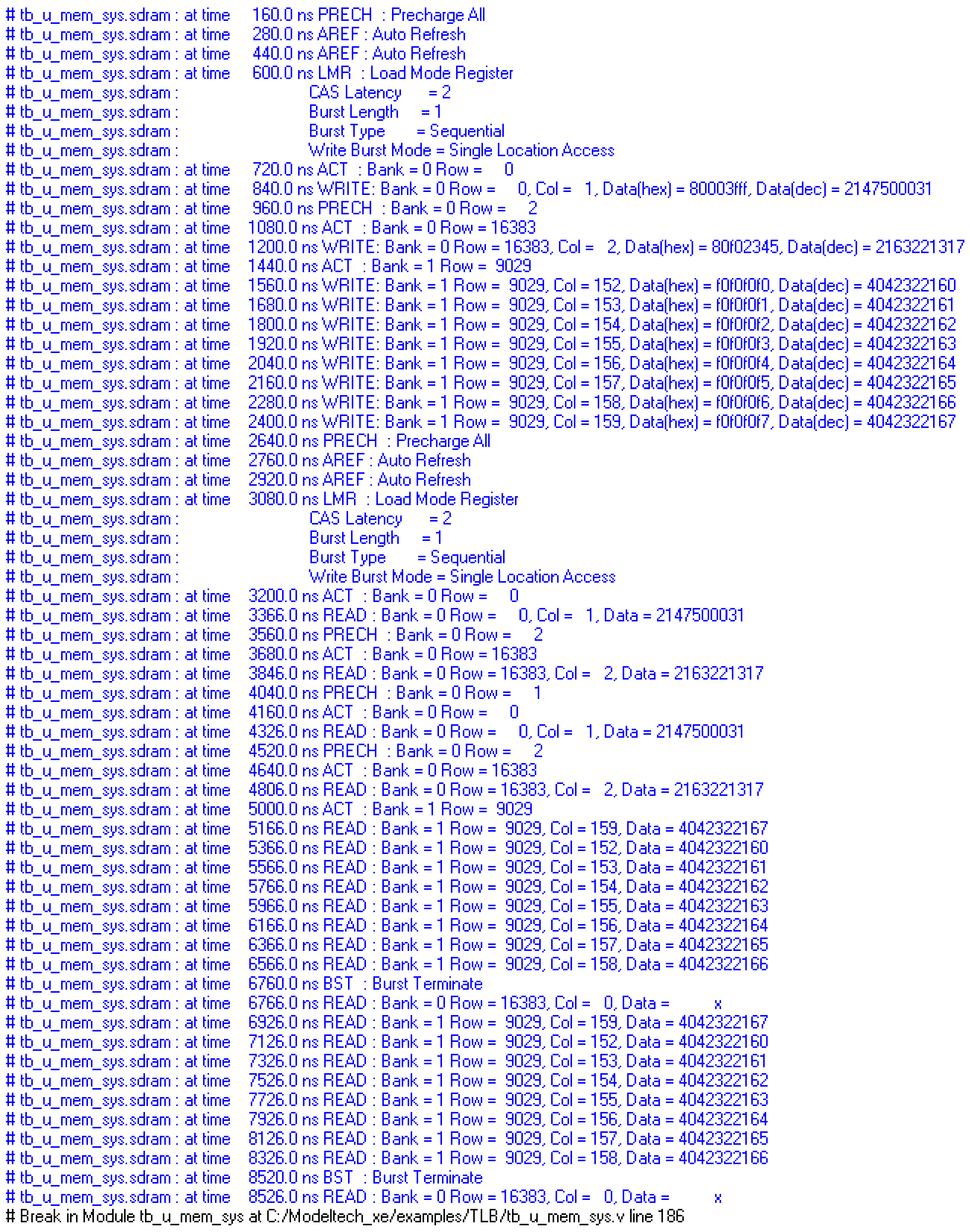
```
/tb_u_mem_sys/tb_r_u_mem_sys_i_clk                        0
/tb_u_mem_sys/tb_r_u_mem_sys_i_reset                      0
/tb_u_mem_sys/w_sdr_addr                                  009b
/tb_u_mem_sys/w_op_wb_dat                                 zzzzzzzz
/tb_u_mem_sys/w_op_wb_ack                                 0
/tb_u_mem_sys/w_sdr_dq                                    f0f0f0f3
/tb_u_mem_sys/tb_w_u_mem_sys_o_mem_is_stall               1
/tb_u_mem_sys/tb_w_u_mem_sys_o_immu_is_stall              0
/tb_u_mem_sys/tb_w_u_mem_sys_o_dmmu_is_stall              0
/tb_u_mem_sys/tb_r_u_mem_sys_i_store_data                 00000000
/tb_u_mem_sys/tb_r_u_mem_sys_i_pc                         00402678
/tb_u_mem_sys/tb_w_u_mem_sys_o_instruction                f0f0f0f6
/tb_u_mem_sys/tb_r_u_mem_sys_i_dmem_addr                  00402678
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_re                     1
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_we                     0
/tb_u_mem_sys/tb_w_u_mem_sys_o_loaded_data                00000000
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_data               00000000
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_addr               0234566c
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_host_ld_mode       0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_stb                1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_cyc                1
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_we                 0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_sel                f
```

Figure 8.2.20: Data cache misses, transferring block from SDRAM (Part 3). (Take 4 clock cycles for subsequence access)



/tb_u_mem_sys/tb_r_u_mem_sys_i_clk	1
/tb_u_mem_sys/tb_r_u_mem_sys_i_reset	0
/tb_u_mem_sys/w_sdr_addr	0000
/tb_u_mem_sys/w_op_wb_dat	zzzzzzzz
/tb_u_mem_sys/w_op_wb_ack	0
/tb_u_mem_sys/w_sdr_dq	zzzzzzzz
/tb_u_mem_sys/tb_w_u_mem_sys_o_mem_is_stall	0
/tb_u_mem_sys/tb_w_u_mem_sys_o_immu_is_stall	0
/tb_u_mem_sys/tb_w_u_mem_sys_o_dmmu_is_stall	0
/tb_u_mem_sys/tb_r_u_mem_sys_i_store_data	00000000
/tb_u_mem_sys/tb_r_u_mem_sys_i_pc	00402678
/tb_u_mem_sys/tb_w_u_mem_sys_o_instruction	f0f0f0f6
/tb_u_mem_sys/tb_r_u_mem_sys_i_dmem_addr	00402678
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_re	1
/tb_u_mem_sys/tb_r_u_mem_sys_i_mem_we	0
/tb_u_mem_sys/tb_w_u_mem_sys_o_loaded_data	f0f0f0f6
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_data	xxxxxxxx
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_addr	xxxxxxxx
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_host_ld_mode	0
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_stb	x
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_cyc	x
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_we	x
/tb_u_mem_sys/tb_w_u_mem_sys_o_sdrctnr_sel	f

Figure 8.2.21: ITLB, DTLB, ICACHE, DCACHE hit. Data and instruction successfully loaded.



```
# tb_u_mem_sys.sdram : at time    160.0 ns PRECH  : Precharge All
# tb_u_mem_sys.sdram : at time    280.0 ns AREF : Auto Refresh
# tb_u_mem_sys.sdram : at time    440.0 ns AREF : Auto Refresh
# tb_u_mem_sys.sdram : at time    600.0 ns LMR  : Load Mode Register
# tb_u_mem_sys.sdram :                        CAS Latency      = 2
# tb_u_mem_sys.sdram :                        Burst Length     = 1
# tb_u_mem_sys.sdram :                        Burst Type       = Sequential
# tb_u_mem_sys.sdram :                        Write Burst Mode = Single Location Access
# tb_u_mem_sys.sdram : at time    720.0 ns ACT  : Bank = 0 Row =     0
# tb_u_mem_sys.sdram : at time    840.0 ns WRITE: Bank = 0 Row =     0, Col =   1, Data(hex) = 80003fff, Data(dec) = 2147500031
# tb_u_mem_sys.sdram : at time    960.0 ns PRECH  : Bank = 0 Row =     2
# tb_u_mem_sys.sdram : at time   1080.0 ns ACT  : Bank = 0 Row = 16383
# tb_u_mem_sys.sdram : at time   1200.0 ns WRITE: Bank = 0 Row = 16383, Col =   2, Data(hex) = 80f02345, Data(dec) = 2163221317
# tb_u_mem_sys.sdram : at time   1440.0 ns ACT  : Bank = 1 Row =  9029
# tb_u_mem_sys.sdram : at time   1560.0 ns WRITE: Bank = 1 Row =  9029, Col = 152, Data(hex) = f0f0f0f0, Data(dec) = 4042322160
# tb_u_mem_sys.sdram : at time   1680.0 ns WRITE: Bank = 1 Row =  9029, Col = 153, Data(hex) = f0f0f0f1, Data(dec) = 4042322161
# tb_u_mem_sys.sdram : at time   1800.0 ns WRITE: Bank = 1 Row =  9029, Col = 154, Data(hex) = f0f0f0f2, Data(dec) = 4042322162
# tb_u_mem_sys.sdram : at time   1920.0 ns WRITE: Bank = 1 Row =  9029, Col = 155, Data(hex) = f0f0f0f3, Data(dec) = 4042322163
# tb_u_mem_sys.sdram : at time   2040.0 ns WRITE: Bank = 1 Row =  9029, Col = 156, Data(hex) = f0f0f0f4, Data(dec) = 4042322164
# tb_u_mem_sys.sdram : at time   2160.0 ns WRITE: Bank = 1 Row =  9029, Col = 157, Data(hex) = f0f0f0f5, Data(dec) = 4042322165
# tb_u_mem_sys.sdram : at time   2280.0 ns WRITE: Bank = 1 Row =  9029, Col = 158, Data(hex) = f0f0f0f6, Data(dec) = 4042322166
# tb_u_mem_sys.sdram : at time   2400.0 ns WRITE: Bank = 1 Row =  9029, Col = 159, Data(hex) = f0f0f0f7, Data(dec) = 4042322167
# tb_u_mem_sys.sdram : at time   2640.0 ns PRECH  : Precharge All
# tb_u_mem_sys.sdram : at time   2760.0 ns AREF : Auto Refresh
# tb_u_mem_sys.sdram : at time   2920.0 ns AREF : Auto Refresh
# tb_u_mem_sys.sdram : at time   3080.0 ns LMR  : Load Mode Register
# tb_u_mem_sys.sdram :                        CAS Latency      = 2
# tb_u_mem_sys.sdram :                        Burst Length     = 1
# tb_u_mem_sys.sdram :                        Burst Type       = Sequential
# tb_u_mem_sys.sdram :                        Write Burst Mode = Single Location Access
# tb_u_mem_sys.sdram : at time   3200.0 ns ACT  : Bank = 0 Row =     0
# tb_u_mem_sys.sdram : at time   3366.0 ns READ : Bank = 0 Row =     0, Col =   1, Data = 2147500031
# tb_u_mem_sys.sdram : at time   3560.0 ns PRECH  : Bank = 0 Row =     2
# tb_u_mem_sys.sdram : at time   3680.0 ns ACT  : Bank = 0 Row = 16383
# tb_u_mem_sys.sdram : at time   3846.0 ns READ : Bank = 0 Row = 16383, Col =   2, Data = 2163221317
# tb_u_mem_sys.sdram : at time   4040.0 ns PRECH  : Bank = 0 Row =     1
# tb_u_mem_sys.sdram : at time   4160.0 ns ACT  : Bank = 0 Row =     0
# tb_u_mem_sys.sdram : at time   4326.0 ns READ : Bank = 0 Row =     0, Col =   1, Data = 2147500031
# tb_u_mem_sys.sdram : at time   4520.0 ns PRECH  : Bank = 0 Row =     2
# tb_u_mem_sys.sdram : at time   4640.0 ns ACT  : Bank = 0 Row = 16383
# tb_u_mem_sys.sdram : at time   4806.0 ns READ : Bank = 0 Row = 16383, Col =   2, Data = 2163221317
# tb_u_mem_sys.sdram : at time   5000.0 ns ACT  : Bank = 1 Row =  9029
# tb_u_mem_sys.sdram : at time   5166.0 ns READ : Bank = 1 Row =  9029, Col = 159, Data = 4042322167
# tb_u_mem_sys.sdram : at time   5366.0 ns READ : Bank = 1 Row =  9029, Col = 152, Data = 4042322160
# tb_u_mem_sys.sdram : at time   5566.0 ns READ : Bank = 1 Row =  9029, Col = 153, Data = 4042322161
# tb_u_mem_sys.sdram : at time   5766.0 ns READ : Bank = 1 Row =  9029, Col = 154, Data = 4042322162
# tb_u_mem_sys.sdram : at time   5966.0 ns READ : Bank = 1 Row =  9029, Col = 155, Data = 4042322163
# tb_u_mem_sys.sdram : at time   6166.0 ns READ : Bank = 1 Row =  9029, Col = 156, Data = 4042322164
# tb_u_mem_sys.sdram : at time   6366.0 ns READ : Bank = 1 Row =  9029, Col = 157, Data = 4042322165
# tb_u_mem_sys.sdram : at time   6566.0 ns READ : Bank = 1 Row =  9029, Col = 158, Data = 4042322166
# tb_u_mem_sys.sdram : at time   6760.0 ns BST  : Burst Terminate
# tb_u_mem_sys.sdram : at time   6766.0 ns READ : Bank = 0 Row = 16383, Col =   0, Data =          x
# tb_u_mem_sys.sdram : at time   6926.0 ns READ : Bank = 1 Row =  9029, Col = 159, Data = 4042322167
# tb_u_mem_sys.sdram : at time   7126.0 ns READ : Bank = 1 Row =  9029, Col = 152, Data = 4042322160
# tb_u_mem_sys.sdram : at time   7326.0 ns READ : Bank = 1 Row =  9029, Col = 153, Data = 4042322161
# tb_u_mem_sys.sdram : at time   7526.0 ns READ : Bank = 1 Row =  9029, Col = 154, Data = 4042322162
# tb_u_mem_sys.sdram : at time   7726.0 ns READ : Bank = 1 Row =  9029, Col = 155, Data = 4042322163
# tb_u_mem_sys.sdram : at time   7926.0 ns READ : Bank = 1 Row =  9029, Col = 156, Data = 4042322164
# tb_u_mem_sys.sdram : at time   8126.0 ns READ : Bank = 1 Row =  9029, Col = 157, Data = 4042322165
# tb_u_mem_sys.sdram : at time   8326.0 ns READ : Bank = 1 Row =  9029, Col = 158, Data = 4042322166
# tb_u_mem_sys.sdram : at time   8520.0 ns BST  : Burst Terminate
# tb_u_mem_sys.sdram : at time   8526.0 ns READ : Bank = 0 Row = 16383, Col =   0, Data =          x
# Break in Module tb_u_mem_sys at C:/Modeltech_xe/examples/TLB/tb_u_mem_sys.v line 186
```

Figure 8.2.22: SDRAM read/write transcript.

Chapter 9: Discussion and Conclusion

9.1: Discussion & Conclusion

Virtual Memory system is the memory management technique which unavoidable, every processor has to use it due to limitation of the size for physical memory. When we adopt virtual memory, Translation Lookaside Buffer plays an important role to determine the speed of the processor. Although without the existence of Translation Lookaside Buffer, processor still can run as usual. Just that for each time of address translation, the processor has to access SDRAM twice if we are using two level hierarchy page tables. Imagine that for each instruction, we have to spend around 40-50 clock cycles to process it, how slow will the processor be. Therefore, Translation Lookaside Buffer is implemented to solve this problem.

Translation Lookaside Buffer is mainly used to store some of the page table entries reside in physical memory. Whenever there is a TLB miss, page table walkthrough need to be conducted to fetch the page table entry out from physical memory and update TLB. To handle this situation, either software, TLB miss is handling by a series of kernel process or hardware, page table walk through is conducted by using hardware. In this project, we are using hardware method in which Memory Management Unit has been implemented in this project to take care about page table walk through.

In this project, 64 entries TLB, MMU, 2MB Cache, 64MB SDRAM has been successfully connected and its behavior has been test during the verification stage. However, there is some error occurs at cache in which it will sending one time more address to SDRAM which causes an invalid READ operation at SDRAM. Although all the data had been successfully read into cache, one of the entry does not write into cache memory which might causes data loss.

The following list is the outcome of this project:-

	Status
TLB	Enhanced & Verified
MMU	Enhanced & Verified
SDRAM	Enhanced & Verified
Memory Unit	Enhanced & Verified

Table 9.1.1: outcome of this project

9.1: Future Works

Memory Unit has been completed and verified. It seems like the cache is not operating as expected. This might cause data loss or stalling effect when integrate into RISC32 processor and therefore, Memory Unit is not yet integrates into RISC32 processor. Improvement and fix needed to overcome this problem by conducting a deep study on current memory system to figure out the root cause of the problem to ensure a workable memory system to be successfully integrated into RISC32 processor.

Other than that, during the verification stage, we need to manually load the page table information by our own due to absent of operating system. To overcome this problem, an operating system should be implemented which will responsible for creation of page table and address mapping process. Therefore, it is necessary for future designer to understand how the memory system works before starting the design of operating system.

References

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

[2] JAY SMITH, A. (2010) IEEE Xplore/IEL. *A Comparative Study of Set Associative Memory Mapping Algorithms and Their Use for Cache and Main Memory*, p.121-128.

[3] Akesson, B. et al. (2007) IEEE Xplore/IEL. *Predator: A Predictable SDRAM Memory Controller*, p.251-256.

[4] Design Gateway Corporation (2002) *SDRAM CONTROLLER DESIGN MANUAL*. [online] Available at: http://www.dgway.com/products/IP/IP-SDRAMCTL.pdf [Accessed: 20 July 2012].

[5] G. Cragon, H. (1996) *Memory Systems and Pipelined Processors Jones and Bartlett Books in Computer Science*. Jones & Bartlett Publishers, Inc..

[6] H.Roth, C. (2012) *Digital System Design Using VHDL*. Boston: PWS Publishing Company.

[7] L. Hennessy, J. and A. Patterson , D. (2004) *Computer Organization and Design the hardware/software interface*. 3rd ed. India: Morgan Kaufman Publishers.

[8] Sweetman, D. (2012) *See MIPS Run* . 2nd ed. Boston: Denise E. M. Penrose.

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

[10] Chun Jin Teoh, “RISC32 Interrupt Handling or Enhanced RISC32 Architecture” University of Tunku Abdul Rahman, Faculty of Information and Communication Technology, 2012.