

**EE 6313**

**Advanced Microprocessor Systems**

**Project 2 on**

**Interfacing cache controller to the 32-bit RISC microprocessor**

**Fall 2019**

**Done By**

**Kumar Ananda Chaya 1001671413**

**Surabhi Chythanya Kumar 1001678544**

**Chandrashekar Mohan 1001657329**

**CONTENTS**

|  |  |
| --- | --- |
| **TOPICS** | **PAGE NUMBERS** |
| INTRODUCTION | 3 |
| PROJECT OVERVIEW | 4 |
| L1 INSTRUCTION AND DATA CACHE | 6 |
| SDRAM | 7 |
| MUX DESIGN | 11 |
| OPERATION OF CACHE | 18 |
| LRU OPERATION | 21 |

**INTRODUCTION:**

In project 1, we had designed the 32-bit RISC processor with four stages of pipeline that interfaced with main memory making it a Von Neumann architecture. Now we are interfacing cache memory in order to increase the speed of operation and thus making it Harvard architecture.

**SPECIFICATIONS:**

* A single SDRAM memory interface with 32bit data bus and burst length of 4 is supported
* All instructions are aligned to a quad byte boundary
* No data access crosses an aligned quad byte boundary
* The processor transaction bus XACT\_IF will terminate on the L1 instruction cache
* The processor transaction buses XACT\_FO and XACT\_WB will terminate on the L1 data cache
* 64kiB of instruction cache and 64kiB of data cache are needed implemented as static RAM
* The cache controller design is write-back, 2-way set-associative with burst-length 4 for SDRAM.

**PROJECT OVERVIEW:**

**Architecture**

ARB 2

MUX I

XACT\_IF

0

S

D

R

A

M

C

T

R

L

L1 I

CACHE

128

0

32

128

SDRAM

ARB 1

128

MUX D

L1 D CACHE

XACT\_FO

1

1

1

XACT\_WB

2

2

128

1 IF(XACT\_FO)

2 IF(XACT\_WB)

0 IF(XACT\_IF)

1 IF(XACT\_FO||XACT\_WB)

* The 32-bit RISC processor generates the signal XACT\_IF, XACT\_FO and XACT\_WB to the main memory.
* Now cache is interfaced between main memory (SDRAM) and the processor to speed up the data transfer.
* SDRAM has a burst length (BL) of 4. It produces the 32-bit data in each burst making it 128-bit of data.
* SDRAM controller controls the data flow between SDRAM and the cache.
* ARB2 selects address or data either from L1 I cache or from L1 D cache based on the priority.
* There are two cache each with 64KiB size used for instruction (L1 I) and the other for data (L1 D).
* ARB1 selects address or data either from XACT\_FO or from XACT\_WB according to the priority.
* MUX I selects 32bits data from L1 I cache to the IF stage.
* MUX D selects 32 bits and transfers data according to opcode and BE signals.

**Truth table for priority encoder**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PRIORITY** | **I2** | **I1** | **I0** | **STAGE** |
| 0 | 1 | 0 | 0 | XACT\_IF |
| 1 | X | 1 | 0 | XACT\_FO |
| 2 | X | X | 1 | XACT\_WB |

**Logic for ARB1**

|  |  |  |  |
| --- | --- | --- | --- |
| **PRIORITY** | **I1** | **I0** | **STAGE** |
| 1 | 1 | 0 | XACT\_FO |
| 2 | X | 1 | XACT\_WB |

**Logic for ARB2**

|  |  |  |  |
| --- | --- | --- | --- |
| **PRIORITY** | **I1** | **I0** | **STAGE** |
| 0 | 1 | 0 | XACT\_IF |
| 1 | X | 1 | XACT\_WB|XACT\_FO |

**DONE signal from MUXI, ARB1 and ARB2**

DONE|ARB1 = HIT (L1 cache has data)

DONE|ARB2 = MEM\_DATA(data from SDRAM)

DONE|MUXI = HIT (L1 I cache has data)

DONE= DONE|ARB1 or DONE|ARB2 or DONE|MUXI

When any of the XACT signals are selected based on the priority, DONE=0 until it has made a hit condition.

**WAIT signal**

If miss,

WAIT = !DONE

* Cache can receive the next XACT\_IF or XACT\_FO or XACT\_WB signal according to the priority only if DONE signal is generated from MUXI or ARB1.
* If WAIT signal in MUXI or ARB1, the processor has to wait until data is read from memory and written to cache block to make it a hit condition so that DONE is generated.
* If WAIT signal in ARB2, the cache has to wait until SDRAM controller sends the data from memory or write to the memory.

**L1 INSTRUCTION AND DATA CACHE:**

The interfacing of cache between the processor and SDRAM reduces the access time.

L1 Instruction cache is only for read and data cache is for both read and write.

**DESIGN OF CACHE:**

* The size of the cache memory is 64KiB.
* Set associative cache is used.

**Specifications of cache:**

* Data width (DW) = 32 bits, Address width (AW) = 32 bits, Space of cache(S)=64KiB, N = 2 ways
* Block size (B) = BL\*DW = 4\*32 = 128 bits
* L = S/(BW\*N) = 64KiB/(128\*2) = 2048 lines

**Set associative cache**

N = 2 ways

L

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D0 | V0 | LRU | TAG0 | B0 | D1 | V1 | LRU | TAG1 | B1 |
|  | | | | | | | | | |

**Features**

* Dirty bit(D): It tells whether the data in cache and SDRAM are coherent or not. If the data in cache is updated, D bit becomes 1 which indicates to write new updated data to SDRAM.
* Valid bit (V): The valid bit tells whether the data in block is valid or not
* LRU: LRU is used to replace the old data with new data during hit or miss condition and increment the necessary LRU by 1.
* TAG: TAG is last 17 bits of the address. It tells where is the data in memory. TAG is used to identify hit or miss condition.
* Block(B): Block is 16 bytes size and holds the data.

**SDRAM**

SDRAM has 32 bit data bus and LMR is programmed with a burst length of 4.

ARB2

L1 I CACHE

0

A0-11

128

BA0

BA1

WE

SDRAM

CONTROLLER

SDRAM

RAS

128

D0-32

CAS

DQML

L1 D CACHE

DQMH

1

CKE

128

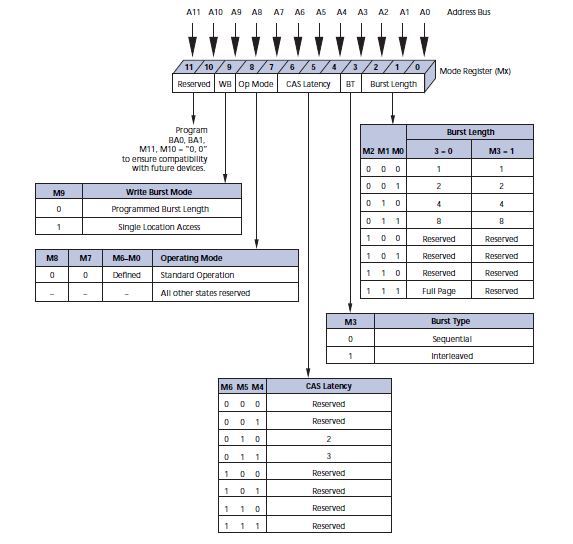
SDRAM CLK

0 IF(XACT\_IF)

1 IF(XACT\_FO||XACT\_WB)

**LOAD MODE REGISTER**

LMR is programmed to give a burst length of 4.



The LMR register contains:

* Burst length= 4
* Burst type= sequential
* CAS Latency = 2
* Operating mode= Standard operation
* Write Burst Mode= Programmed Burst Length

Hence the value of the LMR is 0x0022

**ADDRESS GENERATION FROM SDRAM**

12

8X3 MUX

AUTO REFRESH

12

LMR VALUE

PRECHARGE

ACTIVE

12

12

12

A11-0

READ

WRITE

12

12

NO OPERATION

BUS TERMINATE

12

12

3

CMD

**CMD:**

READ

BUS TERMINATE

PRECHARGE

AUTO REFRESH

WRITE

ACTIVE

LMR

NOP

|  |  |  |
| --- | --- | --- |
| **RAS#** | **CAS#** | **WE#** |
| **0** | **0** | **0** |
| **0** | **0** | **1** |
| **0** | **1** | **0** |
| **0** | **1** | **1** |
| **1** | **0** | **0** |
| **1** | **0** | **1** |
| **1** | **1** | **0** |
| **1** | **1** | **1** |

**BE signals from SDRAM producing 32-bits in each burst to get 128 bits in total**

|  |  |  |  |
| --- | --- | --- | --- |
| BE0 | BE1 | BE2 | BE3 |
| 0 | 0 | 0 | 0 |

D0-127

**DQM OPERATION -** DQMisalways zero as 128 bits are always read or written. We need all the 4 burst length data to make it 128 bits.

4X1 MUX

BE0#

BE1# DQM

BE2#

BE3#

S0 S1

|  |  |
| --- | --- |
| S0 | S1 |
| 0 | 0 |
| 0 | 1 |
| 1 | 0 |
| 1 | 1 |

BE3#

BE2#

BE1#

BE0#

**ADDRESS MAP FROM CACHE TO SDRAM:**

|  |  |  |  |
| --- | --- | --- | --- |
| A31 A25 | A24 A23 | A22 A11 | A10 A2 |

|  |  |  |  |
| --- | --- | --- | --- |
| DECODER | BANK | ROW | COLUMN |

9 bits

12 bits

2 bits

7 bits

**ADDRESS MAP FROM SDRAM TO CACHE:**

|  |  |  |
| --- | --- | --- |
| A31 A15 | A14 A4 | A3 A0 |

|  |  |  |
| --- | --- | --- |
| TAG | LINE | XXXX |

4 bits

11 bits

17 bits

**MUX DESIGN**

**MUX I**

MUX I selects 32bits data from L1 I cache to the IF stage

Processor

L1 I cache

D96-127

32 bits

D64-95

D32-63

D0-31

Fn(bit3,bit2)

D96-127

D64-95

D32-63`

D0-31

|  |  |
| --- | --- |
| Bit3 | Bit2 |
| 0 | 0 |
| 0 | 1 |
| 1 | 0 |
| 1 | 1 |

The 4 bits from LSB of the address

|  |  |  |  |
| --- | --- | --- | --- |
| BIT3 | BIT2 | X | X |

Bit1

Bit0

**MUX D**

MUX D selects 32 bits and data transfer occurs according to the opcode and BE signal.

64KiB L1 Cache

216 bytes

16 Bytes 16 Bytes

|  |  |
| --- | --- |
| **B0** | **B1** |
|  |  |

**MUX D**

211

lines

128

Processor BE

32 bit data

SDRAM BE

0000

Fn(bit1 and bit0 from address and OPCODE)

Fn(bit3 and bit2) from address

The 4 bits from LSB of the address

|  |  |  |  |
| --- | --- | --- | --- |
| BIT3 | BIT2 | BIT1 | BIT0 |

Bit3 and bit2 selects 32 bit data as shown below, and bit 1 and bit 0 is used to select necessary data according to the opcode and BE.

D96-127

D64-95

D32-63

D0-31

|  |  |
| --- | --- |
| Bit 3 | Bit 2 |
| 0 | 0 |
| 0 | 1 |
| 1 | 0 |
| 1 | 1 |

From Data cache

Microprocessor

8/16/32

|  |
| --- |
| BE3 |
| BE2 |
| BE1 |
| BE0 |

Fn ( bit 1 and bit 0 and OPCODE)

32

|  |  |
| --- | --- |
| Bit 1 | Bit 0 |
| 0 | 0 |
| 0 | 1 |
| 1 | 0 |
| 1 | 1 |

BE0

BE1

BE2

BE3

OPCODE= LDU8/LDS8

If ((OPCODE = 19|| 20) && LSB of address bits is 0 0 0 0)

TO PROCESSOR

FROM CACHE

3:1

Z

BE3

D24-31

S

3:1

BE2

Z

S

BE1

5:1

Z

4:1

BE0

D0-7

* Here OPCODE is of 8 bit data.
* Since bit 3 and bit 2 are 0 0, D0-31 is selected
* Since bit 1 and bit 0 are 0 0, 8 bit data will flow from BE0.
* Signed or zero extension is done according to the opcode.

OPCODE= LDU16/LDS16

If ((OPCODE = 17|| 18) && LSB of address bits is 0 0 0 1)

FROM CACHE

TO PROCESSOR

3:1

Z

BE3

S

3:1

BE2

Z

D16-23

S

BE1

5:1

Z

D8-15

4:1

BE0

* Here OPCODE is of 16 bit data.
* Since bit 3 and bit 2 are 0 0, D0-31 is selected
* Since bit 1 and bit 0 are 0 1, 16 bit data will flow from BE1 and BE2.
* Signed or zero extension is done according to the opcode.

OPCODE=LD32

If ((OPCODE = 16 && LSB of address bits is 0 0 X X)

MUX D

CACHE

3:1

Z

BE3

D24-31

S

3:1

BE2

Z

D16-23

S

BE1

5:1

D8-15

Z

4:1

BE0

D0-7

* Here OPCODE is of 32 bit data.
* Since bit 3 and bit 2 are 0 0, D0-31 is selected.
* Bit 1 and bit 0 are X X.
* Signed or zero extension is done according to the opcode.

OPCODE=LDU8/LDS8

If ((OPCODE = 19|| 20) && LSB of address bits is 0 1 0 0)

MUX D

CACHE

3:1

Z

BE3

D54-63

S

3:1

BE2

Z

S

BE1

5:1

Z

4:1

BE0

D32-39

* Here OPCODE is of 8 bit data.
* Since bit 3 and bit 2 are 0 1, D32-63 is selected.
* Since bit 1 and bit 0 are 0 0, 8 bit data will flow from BE0.
* Signed or zero extension is done according to the opcode.

**OPERATION OF CACHE**

Set associative cache

N = 2 ways

1080

2047

0

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D0 | V0 | LRU0 | TAG0 | B0 | D1 | V1 | LRU1 | TAG1 | B1 |
| 0 | 0 |  |  |  | 0 | 0 |  |  |  |
| 0 | 0 |  |  |  | 0 | 0 |  |  |  |
| 0 | 0 |  |  |  | 0 | 0 |  |  |  |
| 0 0 0 0 | | | | | | | | | |

Assuming line number 1080 to be accessed as shown in the above figure.

During read operation by the processor from cache, the tag number in the address bus must match with the tag number in the cache. If the tag number in the line 1080 doesn’t match with the tag number in the address bus, then condition is a miss. Hence the address is passed on to SDRAM. The SDRAM fetches the data according to the line number and tag number and it is moved to cache with the same tag number and line number, thus making it a hit condition. As soon as hit condition occurs, DONE changes from 0 to 1. Assuming way=1 has a hit condition as shown below:

N = 2 ways

1080

2047

0

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D0 | V0 | LRU0 | TAG0 | B0 | D1 | V1 | LRU1 | TAG1 | B1 |
| 0 | 0 |  |  |  | 0 | 0 |  |  |  |
| 0 | 0 |  |  |  | 0 | 0 |  |  |  |
| 0 | 0 |  |  |  | 0 | 0 |  |  |  |
| 0 0 0 1 0 | | | | | | | | | |

Now the valid bit (V1) is 1, LRU bit is made 0 as it is new, block B1 will have data and dirty bit (D1) will remain 0.

During write to cache, and it is a hit condition to way=0. The valid bit (V0) is 1 and dirty bit (D0) will be 1, LRU bit (LRU0) will be 0 (new) and LRU bit (LRU1) of way=1 will be incremented by 1 (old) as shown below.

N = 2 ways

1080

2047

0

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D0 | V0 | LRU0 | TAG0 | B0 | D1 | V1 | LRU1 | TAG1 | B1 |
| 0 | 0 |  |  |  | 0 | 0 |  |  |  |
| 0 | 0 |  |  |  | 0 | 0 |  |  |  |
| 0 | 0 |  |  |  | 0 | 0 |  |  |  |
| 1 1 0 0 1 1 | | | | | | | | | |

**OPERATION OF MISS AND HIT**

In cache, block being replaced

Is d=1, V=1?

NO

ADD

Find a place in cache

Is in cache?

Read block into cache. Mark V=1

YES

YES

DATA

Read cache

Ready

Writeback cache to memory

Set d=0, V=0

* If any of the XACT signal is selected according to the priority, DONE=0.
* During read, if the tag number in the address from the processor and the tag number in the cache matches, then it is a hit condition and cache is ready and data is sent to processor and done signal is generated i.e., DONE=1.
* During write to cache and if it’s a hit condition, the LRU of the way that is hit will be made 0 and the LRUs of the ways that are less than the LRU of the way that was hit will be incremented by 1.
* If the tag number in the address from the processor, and the tag number in the cache does not match, then it is a miss condition. Then, it tries to find a block with the oldest LRU value and checks if d=1, v=1 to replace that block with new data. If d and v are 1 then cache will writeback data to memory and set d=0, v=0.
* Then, the matching tag location is read from memory to cache block and set v=1 and the new tag number is written and the LRU is made 0(new).The remaining LRU’s age will be incremented by 1.
* Now, the cache is ready and the data is read by the processor from cache.

If d=1, v=1 when the block is being replaced in cache:

Average Memory Access Time (AMAT) = Thit + Twriteback,dirtyblock + Tread new block

If d!=1 when the block is being replaced in cache:

Average Memory Access Time (AMAT) = Thit + Tread new block

* TAGADD xor TAGCACHE = TAGCMP

NORing TAGCMP bits with each other gives MATCH

TAGADD

TAGCMP0-16

TAGCACHE

TAGCMP0

MATCHN

TAGCMP16

* If MATCH=1, then it is a hit.

If MATCH=0, then it is a miss

**LRU operation**

**Search algorithm of LRU bits**

for Ɐ i Є {0,1,…,N-1}

if (LRUi < LRUway)

LRUi++

LRUway (one which is replaced or changed) = 0

**For N=2ways,**

**if (miss)**

Way 0 1

LRU 1(oldest) 0(new)

0(new) 1(oldest)

* Replace way 0 with LRU=0(new)
* Age way1 LRU from 0 to 1

**if (hit)**

assuming way number 1 that matches the tag

way 0 1

LRU 0(new) 1(oldest)

1(oldest) 0(new)

* Hit is on way 1. Change LRU of way1 to 0(new)
* Age all LRU with way< 1

**LOGIC DIAGRAM**

For N = 2 ways,

|  |  |  |
| --- | --- | --- |
| TAG | LINE | XXXX |

LRU BLOCK

LRU0

LRU1

log22 = 1 bit

Fn(hit or miss)

MATCH0

Tag 0

Way

MATCH1

Tag 1

Hit or miss

**MATCH LOGIC**

TAGADD

TAGCMP0-16

TAGCACHE

TAGCMP0

MATCHN

TAGCMP16

* The TAG from processor is compared with TAG from cache by XORing it.
* The XORed TAG is NORed bitwise to determine MATCH condition.
* Finally the MATCH bits will be ORed to determine hit or miss condition
* The LRU block chooses appropriate LRU depending on hit or miss condition.
* If MATCH=1, its respective WAY will be selected and will be assigned with respective LRUs according to hit condition and will be aged accordingly.
* If miss condition, all the MATCH=0, and the way with the oldest LRU will be selected.