數位系統設計 HW4: Cache

B04901060 電機三 黃文璁

1. General specifications

- **Direct mapped**
  - **Size**: 32 words (8 blocks x 4 words)
  - **Placement**: direct mapped
  - **Details for each block**:
    - **Total**: 155b/block
    - Valid bit: 1b
    - Dirty bit: 1b
    - Tag: 25b
    - Data: 128b (32b x 4)

- **2-way set associative**
  - **Size**: 32 words (4 sets x 2 ways x 4 words)
  - **Placement**: 2-way set associative
  - **Details for each block**:
    - **Total**: 156b/block
    - Valid bit: 1b
    - Dirty bit: 1b
    - Tag: 26b
    - Data: 128b (32b x 4)

2. Read/write policy

- **Direct mapped**
  - **Write policy**: write back + write allocate
  - **Write buffer**: yes (see bonus)
  - **Read preloader**: yes (see bonus)
  - **Block replacement policy**: none (only one candidate)

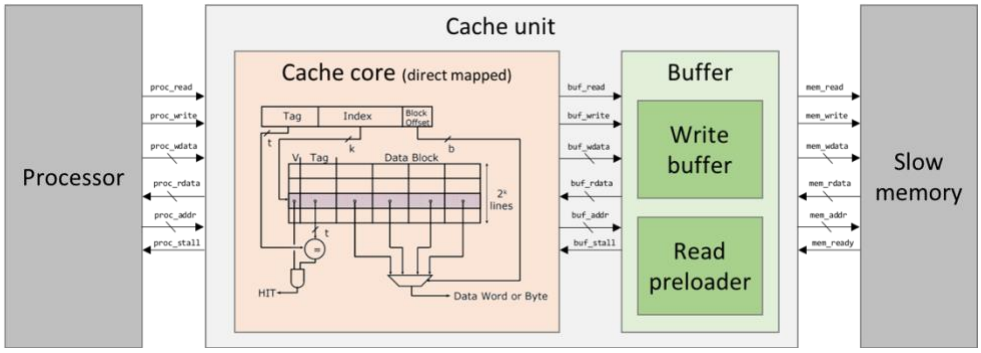
- **2-way set associative**
  - **Write policy**: write back + write allocate
  - **Write buffer**: yes (see bonus)
  - **Read preloader**: yes (see bonus)
  - **Block replacement policy**: round-robin

3. Design architecture or the finite state machine

以下為兩種做法的架構和 FSM，其中 Cache core 和投影片上的架構基本相同，圖中主要展示各模組間的 I/O 介面。另外本次雖然實作 write back 但仍然在 cache 和 memory 間加上了 buffer 提升效能，buffer 的細節和 FSM 請參考 bonus。

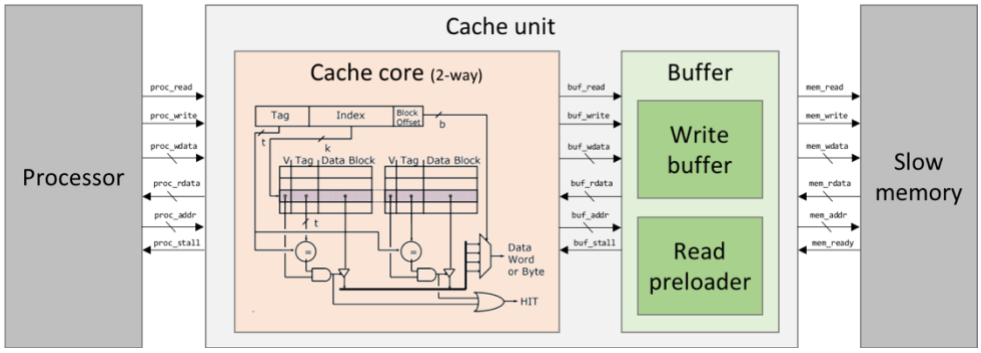
■ **Direct mapped**

- **Architecture**



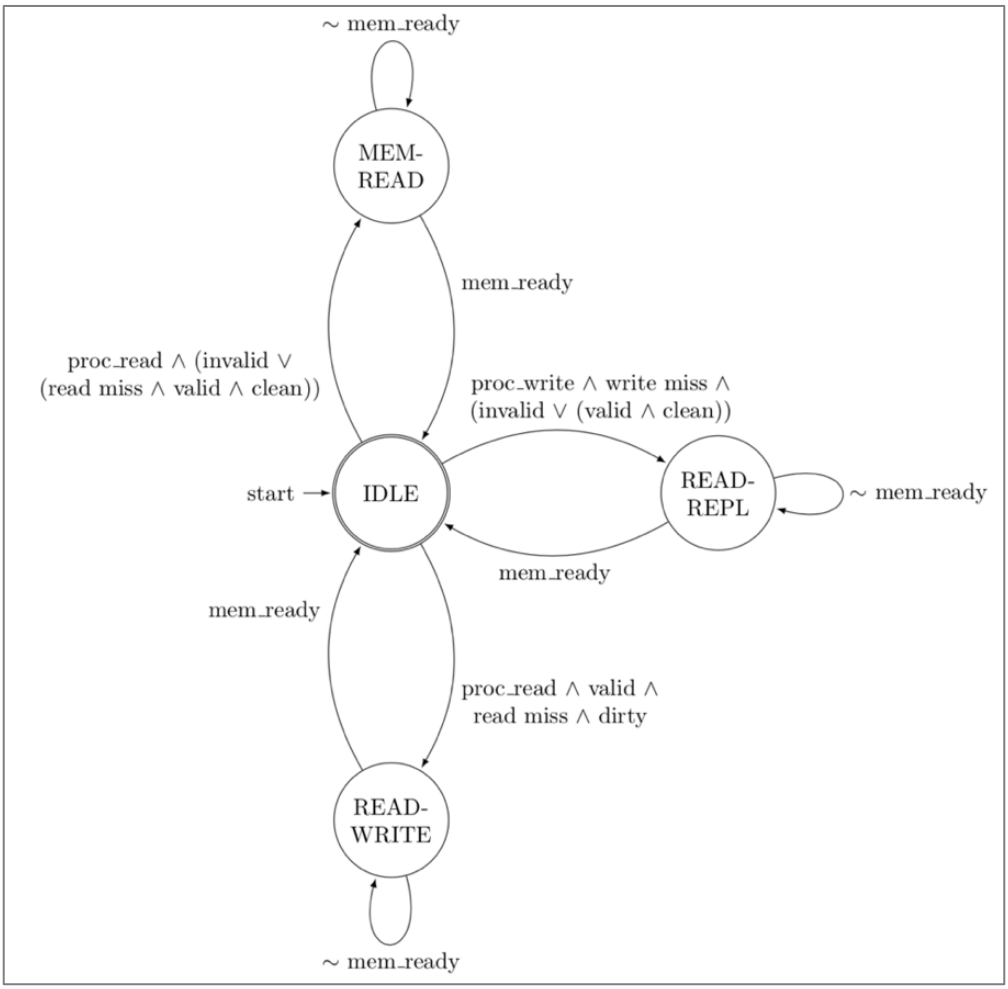
■ **2-way set associative**

- **Architecture**



■ **Finite state machine (for both architectures)**

本次作業中兩種架構使用的 FSM 相同，如下圖：



總共 4 個狀態，其中以下 3 種情況需要分別轉換到 IDLE 以外的另外三個狀態：

1. **MEM_READ**：
   當 testbench 需要讀取記憶體（proc_read）且：
   (1) Block 為 invalid
   (2) Block 為 valid 且 read miss 且 clean（dirty bit = 0）
   此時切換到 MEM_READ 狀態，從 memory 讀取並等待 mem_ready。
2. **READ_REPLACE**：
   當 testbench 需要寫值（proc_write）且：
   (1) Block 為 invalid 且 write miss
   (2) Block 為 valid 且 write miss 且 clean
   此時需要從記憶體讀值，且將指定的 word 取代成要寫入的值，並存在 cache。
3. **READ_WRITE**：
   當 testbench 要讀值，read miss 且該 block 為 valid、dirty 時，
   要將原本在 block 中的值寫回 memory，並從記憶體讀值。
   若使用 write back + write buffer 的話，可以先讀值後再寫值，且寫值的同時可以直接切回 IDLE 狀態，而不用等待記憶體完成寫入，提升效能。

4. Performance evaluation (read/write miss rate, execution cycles, stalled cycles, …)

以下為本次實作中最快的版本，詳細比較請參考 bonus。

（Write back + write allocate + write buffer + direct mapped + Mealy machine + read preloader）

- **Read miss rate**: 25.0%
- **Write miss rate**: 25.0%
- **Read miss penalty**: 0.16 cycles
- **Write miss penalty**: 3.98 cycles
- **Execution cycles**: 3072 (73.6% total cycles)
- **Stalled cycles**: 1100 (26.4% total cycles)
- **Total cycles**: 4172

5. Comparing the performance of two kinds of architecture, discuss the results

本次作業使用的 testbench 由於只有非常規則的循序讀寫，故並沒有辦法顯現出 direct mapped 和 2-way 之間的效能差異，以下比較兩種做法的差異，configuration 和上題相同，差別只有將 direct mapped 改為 2-way set associative：

	Read miss rate	Write miss rate	Execution cycles	Stalled cycles	Total cycles
DM	25.0%	25.0%	3072	1104	4172
2-way	25.0%	25.0%	3072	1108	4180

可以發現效能是很接近的，實際上 2-way 要稍微慢一點點，但差異並不明顯，這是由於這次的 testbench 對兩種做法來說 miss rate 基本上是一樣的，且這次兩種實作方式的 miss penalty 差不多，所以相比之下應該不會有很大差別。

6. Simulation results

(1) 通過 testbench 模擬（～4172 cycles）

```
Loading snapshot worklib.tb_cache:v ..................... Done
*Verdi* Loading libsscore_ius152.so
ncsim> source /usr/cad/cadence/INCISIV/cur/tools/inca/files/ncsimrc
ncsim> run
FSDB Dumper for IUS, Release Verdi_N-2017.12, Linux, 11/12/2017
(C) 1996 - 2017 by Synopsys, Inc.
*Verdi* : Create FSDB file 'cache.fsdb'
*Verdi* : Begin traversing the scopes, layer (0).
*Verdi* : End of traversing.
Memory has been initialized.

Processor: Read initial data from memory.
    Done correctly so far! ^_^

Processor: Write new data to memory.
    Finish writing!

Processor: Read new data from memory.
    Done correctly so far! ^_^

==== CONGRATULATIONS! Pass cache read-write-read test. ====

Finished all operations at:                20858 ns
Exit testbench simulation at:              20908 ns


Simulation complete via $finish(1) at time 20907500 PS + 0
./tb_cache.v:170        $finish;
ncsim> exit
b04060@cad39:~/check$
```

(2) read_verilog 沒有 latch

```
Compiling source file /home/raid7_2/userb04/b04060/check/cache_dm_mealy_wb_rp.v

Statistics for case statements in always block at line 116 in file
        '/home/raid7_2/userb04/b04060/check/cache_dm_mealy_wb_rp.v'
=================================================
|          Line          |  full/ parallel  |
=================================================
|          130           |    auto/auto     |
|          136           |    auto/auto     |
|          142           |    auto/auto     |
|          148           |    auto/auto     |
=================================================

Inferred memory devices in process
        in routine cache line 340 in file
                '/home/raid7_2/userb04/b04060/check/cache_dm_mealy_wb_rp.v'.
=============================================================================================
|      Register Name      |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
=============================================================================================
|     buf_wdata_r_reg     | Flip-flop |  128  |  Y  | N  | Y  | N  | N  | N  | N  |
| buf_write_request_r_reg | Flip-flop |   1   |  N  | N  | Y  | N  | N  | N  | N  |
|    buf_request_r_reg    | Flip-flop |   1   |  N  | N  | Y  | N  | N  | N  | N  |
|        state_reg        | Flip-flop |   3   |  Y  | N  | Y  | N  | N  | N  | N  |
|        block_reg        | Flip-flop | 1240  |  Y  | N  | Y  | N  | N  | N  | N  |
|    proc_stall_r_reg     | Flip-flop |   1   |  N  | N  | Y  | N  | N  | N  | N  |
|    proc_rdata_r_reg     | Flip-flop |  32   |  Y  | N  | Y  | N  | N  | N  | N  |
|     buf_addr_r_reg      | Flip-flop |  28   |  Y  | N  | Y  | N  | N  | N  | N  |
=============================================================================================
Statistics for MUX_OPs
=================================================
| block name/line  | Inputs | Outputs | # sel inputs |
=================================================
|    cache/123     |   8    |   155   |      3       |
=================================================

Inferred memory devices in process
        in routine buffer line 535 in file
                '/home/raid7_2/userb04/b04060/check/cache_dm_mealy_wb_rp.v'.
=============================================================================================
|      Register Name      |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
=============================================================================================
|     prev_addr_r_reg     | Flip-flop |  28   |  Y  | N  | Y  | N  | N  | N  | N  |
|        state_reg        | Flip-flop |   3   |  Y  | N  | N  | Y  | N  | N  | N  |
|    buf_ravail_r_reg     | Flip-flop |   1   |  N  | N  | Y  | N  | N  | N  | N  |
|     buf_raddr_r_reg     | Flip-flop |  28   |  Y  | N  | Y  | N  | N  | N  | N  |
|     buf_rdata_r_reg     | Flip-flop |  128  |  Y  | N  | Y  | N  | N  | N  | N  |
|     buf_stall_r_reg     | Flip-flop |   1   |  N  | N  | N  | Y  | N  | N  | N  |
|     buf_rdone_r_reg     | Flip-flop |   1   |  N  | N  | Y  | N  | N  | N  | N  |
|     mem_addr_r_reg      | Flip-flop |  28   |  Y  | N  | Y  | N  | N  | N  | N  |
|     mem_read_r_reg      | Flip-flop |   1   |  N  | N  | N  | Y  | N  | N  | N  |
|     mem_write_r_reg     | Flip-flop |   1   |  N  | N  | Y  | N  | N  | N  | N  |
|     mem_wdata_r_reg     | Flip-flop |  128  |  Y  | N  | Y  | N  | N  | N  | N  |
| cont_read_counter_r_reg | Flip-flop |   3   |  Y  | N  | Y  | N  | N  | N  | N  |
=============================================================================================
Presto compilation completed successfully.
Current design is now '/home/raid7_2/userb04/b04060/check/cache.db:cache'
Loaded 2 designs.
Current design is 'cache'.
cache buffer
```

7. Bonus: other techniques to improve the cache performance

以下簡介三個本次使用的技巧，最後對三種方法的結果進行比較。

(1) Mealy machine

Mealy machine 由於 output 和 input 間有 combinational 部分，故可以不用等到下一個 cycle，相對於 Moore machine 來說可能可以有更好的效能，另外 Mealy machine 一般來說狀態也較少，缺點是 output 可能會有 glitch 產生，所以也要在輸出的地方加上 register 來避免 glitch。本次作業中可用最少 cycle 數完成 testbench 的實作方式基於 Mealy machine。

結果：Total cycles: **Moore machine**: 9275 → **Mealy machine**: 6164

(2) Write back + write buffer

雖然 write back 可以不用 write buffer 實作，但是額外加上一個 write buffer 可以提高效能，由於這次 testbench 可以得知 4 次 write 才有一次 write miss，且 memory 可在 4 個 cycle 完成寫入，由於寫入時可以同時進行其他操作，故可以將部分 latency 隱藏起來。

這次實作的 write buffer 會輸出 stall 訊號，當 cache 要寫入記憶體時會經過 write buffer，但和直接寫入 memory 不同的是可以不用等待 mem_ready，而直接切到下一個 state，此時在 buffer 還沒寫完時 stall 為 1，若接下來的操作中 cache 需要讀寫 buffer，則會等到 buffer 的 stall 為 0 時才進行動作。以這次的 testbench 來說不會有快速連續寫入的操作，故 buffer 可以將許多 memory writing 的 latency 隱藏，write buffer 大小為 128bit。

結果：Testbench 的 write 階段共 1024 次 write 操作，所花費的 cycle 數為：**2044**

(3) Read preloader

觀察 memory.v 可以發現本次作業使用的 memory 其 clock cycle 為 cache 的 1/4，不過又可以想到 cache 每個 block 為 4 個 word，且 testbench 為循序讀取，故有機會實現 read 操作的 latency hiding。

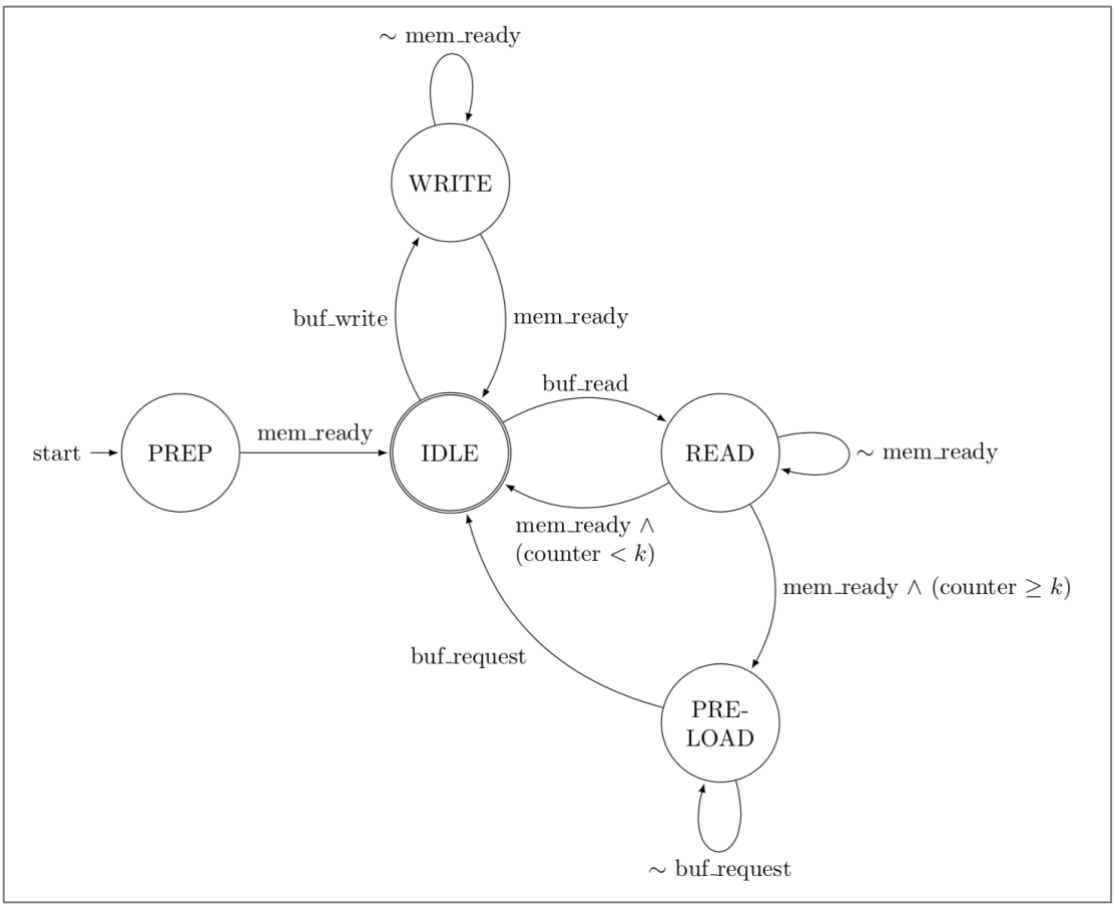
實作方法為：將 write buffer 改為 read/write buffer，此 buffer 會用一個 counter 紀錄當前的「連續地址、連續讀取」操作有多少次，當連續 k 次皆為此類操作，例如：

Read (addr=0x01) → Read (addr=0x02) → Read (addr=0x03) → Read (addr=0x04)

此時 buffer 會預測接下來的操作也是連續地址讀取，故會先從記憶體繼續讀取位於 0x05 的資料，若和 cache 接下來要讀取的地址相同，則會繼續讀取 0x06、0x07…。如此當 cache 要讀值時就不用等待 memory read latency。若 cache 接下來要讀取的地址不是 buffer 已經讀好的地址，或者 cache 的下一個操作是寫入而不是讀取，則 counter 歸零，buffer 回到 IDLE 狀態。FSM 請參考下頁。本實驗中 k=4。

結果：Testbench 的第一輪 read 共 1024 次 read 操作，所花費的 cycle 數為：**1052**
Testbench 的第二輪 read 共 1024 次 read 操作，所花費的 cycle 數為：**1076**

Buffer (write buffer + read preloader) 的 FSM 如下：



(4) Comparison

以下為 ablation study，從最基本的 Moore machine 實作開始不斷加入其他改良。
WB: write buffer, RP: read preloader, DM: direct mapped, 2-way: 2-way set associative

		Read miss rate	Write miss rate	Execution cycles	Stalled cycles	Total cycles
Moore	DM	25.00%	25.00%	3072	7195	10267
	2-way	25.00%	25.00%	3072	7223	10295
Moore+WB	DM	25.00%	25.00%	3072	6203	9275
	2-way	25.00%	25.00%	3072	6231	9303
Mealy+WB	DM	25.00%	25.00%	3072	3092	6164
	2-way	25.00%	25.00%	3072	3124	6196
Mealy+WB+RP	DM	25.00%	25.00%	3072	1100	4172
	2-way	25.00%	25.00%	3072	1108	4180

在各種實作中 DM 和 2-way 的效能都很接近，注意到 Moore machine 由於 input delay 的關係至少要 stall 一個 cycle，另外由於 memory 的 cycle 是 cache 的 1/4，所以如果錯過一個 clock edge 就要再多等 4 個 clock，所以 Moore machine 相對於 Mealy 要慢得多，另外由於 read 完全是照著 memory address 讀取，故 RP 的效果很明顯，可以發現 stalled cycles 降低許多，幾乎只剩下 write 時的 stall。