CA HW5 Report

Q1: General specification of the cache unit

A1:這次作業是設計 cache，那我是用二維的 reg[154:0]CACHE[7:0]來當作 direct-maped cache，其中[154]是 valid bit、[153]是 dirty bit、[152:128]cache tag、[127:0]則是裝四個 word 的 data，並透過 reg [127:0] block 來做運算，等到每個 posedge clk 來的時候再跟新值回去 CACHE，一開始會先判定是 proc_read or proc_write 如果都不是，那就是在最一開始的地方，把所有控制訊號設為 0，proc_stall 設為 1，那如果是 read 的話，就先判定是否為 valid，若不為 valid，先從 memory 把 data 拿來放入 block，接下來再判定是否為 hit，hit 的話就直接讀值，miss 的話就要看 dirty bit 是不是 1，如果是 0，就從 memory 拿 data 放進 CACHE 再做讀，如果 dirty bit 是 1，那就把資料先

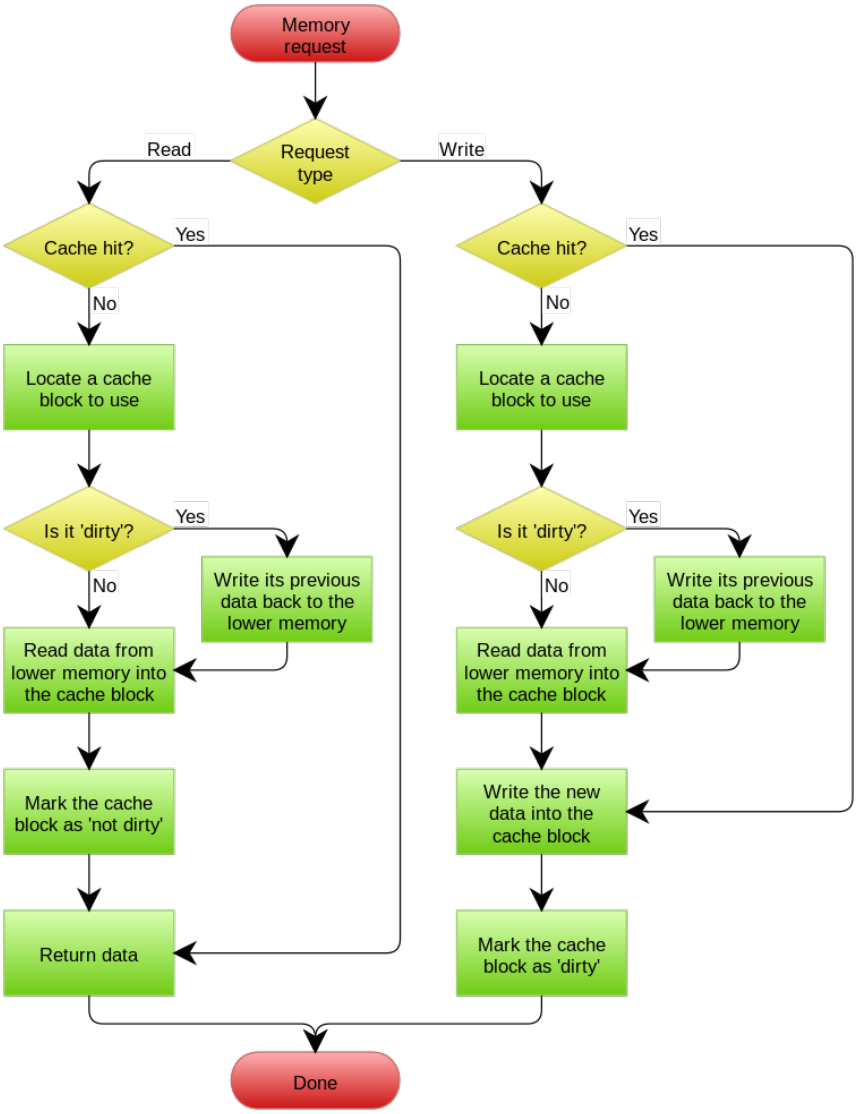
write back 回 memory 再 locate 新的資料回 CACHE，並做讀值，還要把 dirty bit 改回 0，proc_stall 也會改回 0。如果是 write 的話也很類似，但是沒有先判斷是不是 valid，而是直接看是 hit or miss，如果是 hit 就是改值並且把 dirty it 改為 1，如果 miss 的話，要看 dirty bit 是不是 1，如果是 0，就去 memory 拿值（改 tag）並把新的值寫入，dirty bit 改為 1，若 dirty bit 是 1 就要先 write back 回記憶體，再從記憶體拿新的值回來 CACHE 並寫新的值進去 CACHE，然後把 dirty bit 設為 1，用 proc_wdata 寫入新的值後，proc_stall 都要改回 0。

Q2:Read/Write Policy

分別有 write through 和 write back 兩種，write through 的話不用用到 dirty bit，因為他只要一發生改變就會存回 memory，雖然比較慢，但是

可以確保，memory 和 CACHE 裡的值是一樣的，那如果使用 write back 的話，則兩邊的值可能會不一樣，但是不用一改值就送回 memory，只有他要再改的時候才需要存回 memory（By dirty bit）。

Q3:State Graph



Q4:

Read/Write 的 miss rate 應該都和 block 的 size 有關，這次要求的 block size 為 4-word，所以一次 miss 會更新四個 word 的共同 tag，miss rate 應為 25%。

若為 read/write operation 沒有經過 write back 的話，會先 stall 6 個 cycle，在每個 word 又 1 個 cycle，總共 10 個 cycle，如果需要 write back，則會 stall 13 個 cycle，在每個 word 又 1 個 cycle，總共 17 個 cycle。

Q5:

CLOCK CYCLE TIME:5ns

TOTAL Cell Area:74020 um^2

```
                worklib.tb_cache:v <0x386ec312>
                        streams:   7, words: 11702
        Loading native compiled code:     ..................... Done
        Building instance specific data structures.
        Design hierarchy summary:
                                    Instances  Unique
                Modules:                 4662     530
                UDPs:                    1942      12
                Primitives:             13851      10
                Timing outputs:          6034     533
                Registers:               1421     189
                Scalar wires:            7441       -
                Expanded wires:           190       3
                Always blocks:              4       4
                Initial blocks:             6       6
                Cont. assignments:          0       3
                Pseudo assignments:         1       1
                Timing checks:          13129    1372
                Interconnect:           13548       -
                Delayed tcheck signals:  3720    1345
                Simulation timescale:     1ps
        Writing initial simulation snapshot: worklib.tb_cache:v
Loading snapshot worklib.tb_cache:v ..................... Done
ncsim> source /usr/cad/cadence/INCISIV/cur/tools/inca/files/ncsimrc
ncsim> run
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:                50273 ns
Exit testbench simulation at:              50323 ns


Simulation complete via $finish(1) at time 50322500 PS + 0
./tb_cache.v:172         $finish;
```

補充：

這次在寫作業遇到最大的問題是如何在 write back 的時候 stall 兩次，一開始是這樣寫

```
if (CACHE[index][153]) begin//dirty bit=1
    proc_stall=1;
    mem_addr={CACHE[index][152:128],index};
    mem_write=1;
    mem_wdata=CACHE[index][127:0];

    if (mem_ready) begin // mem ready for write to

    proc_stall=0;
    mem_write=0;
    mem_ready_2w=1;

    if (mem_ready_2) begin // mem going to read from

    proc_stall=1;
    mem_read=1;
    mem_addr=proc_addr[29:2];

    if (mem_ready) begin//mem ready for read from

    proc_stall=0;
    mem_read=0;

    CACHE[index][31:0]=mem_rdata[31:0];
    CACHE[index][63:32]=mem_rdata[63:32];
    CACHE[index][95:64]=mem_rdata[95:64];
    CACHE[index][127:96]=mem_rdata[127:96];
    CACHE[index][152:128]=proc_addr[29:5];

    set_1=CACHE[index][31:0];
    set_2=CACHE[index][63:32];
    set_3=CACHE[index][95:64];
    set_4=CACHE[index][127:96];

    CACHE[index][153]=0;

    case(block_offset)
        2'b00:proc_rdata=CACHE[index][31:0];
        2'b01:proc_rdata=CACHE[index][63:32];
        2'b10:proc_rdata=CACHE[index][95:64];
        2'b11:proc_rdata=CACHE[index][127:96];
         endcase

    end //mem ready for read from
```

但是 mem_write 不會上升，所以也不會 mem_ready 兩次，後來嘗試在這裡寫 state graph：

```
always @(*) begin
    case(state)

        idle:if (mem_write) begin
                    next_state=writebackto;
                    //mem_write=1;
                      end
                    else begin
                    next_state=idle;
                    end

    writebackto:if (mem_ready) begin
                    next_state=getfromMem;
                    proc_stall_w=0;
                    mem_write_w=0;
                    end
                else begin
                    next_state=writebackto;
                end

    getfromMem:if (mem_ready) begin
                    next_state=idle;
                    proc_stall_w=0;
                    mem_read_w=0;
    end
                else begin
                    next_state=getfromMem;
                end


    default:next_state=idle;

    endcase

end
```

proc_stall 也還是不受控ＱＱＱＱ，最後是乾脆在一開始就把 proc_stall 設為 1，到要 output 才把 proc_stall 調回 0，才解決問題，這次 nWave 真是看到眼睛都快脫窗了ＱＱ