R10922171 陳嘉政 Lab2

● Modules explanation

1. 此 project 除保留一些原本的 modules 外，將 data memory 改為 off-chip(在 testbench 與 cpu 溝通)，將原來的 data memory 位置取代成 data cache controller(dcache)，透過 dcache controller 來存取 dcache sram。(CPU 中新增 dcache，dacache 中又包含 dcache_sram)
2. 將 dcache 輸出的 stall 信號線接到 PC 與每個 pipeline register，當 cache miss 時，表示要到 memory 搬資料，所以 cache controller 會輸出 stall 信號，讓整個 pipeline 暫停。
3. Cache controller 的運作流程如下:

(1) 一開始為 idle state，當 CPU request 來時(memread or memwrite)，如果同時資料在 cache sram 中(hit)，則資料直接送給 CPU，不須 stall，controller 繼續 idle;否則，進入(2)。

(2) 將 mem_enable 設 1 表示要使用到 memory，接著檢查 sram 中此 CPU address 對應的 set 中選到的 block 的 dirty bit 是否為 1，如果為 1 表示此 block 有被寫過，要先寫回 memory，所以 mem_write 及 write_back 設為 1。接著進入 miss state(3)。

(3) 表示 write miss 跟 read miss 皆有可能。再檢查一次 sram dirty bit，若為 1 表示此 block 被寫過需寫回 memory，將 mem_enable、mem_write、write_back 設為 1，進入 writeback state(4);否則只需將所要的 block 從 memory 中搬上來覆蓋掉 cache block，所以 mem_enable 設 1，mem_write 及 write_back 設 0，進入 readmiss state(5)。

(4) 表示要 write to cache，且 write miss，要寫入的 block 被修改過，需先寫回 memory。等待 memory ack 信號，如果 memory 為 wait state 且經過了 10 個 clock，將 ack 設為 1。若 ack 為 1，表示 memory 寫入完成，將 mem_write 及 write_back 設為 0，轉成 readmiss state(5)。

(5) 等待 memory 把所要的 block 搬上來。等待 memory ack 信號，如果信號為 1 表示 memory 已經把資料搬上來，mem_enable 設 0，cache_write 設 1(因為接著要寫入 cache)，進入 readmissok state(6);若信號為 0，則繼續等待。

(6) 資料成功寫入 cache，將 cache_write 設回 0，轉回 idle state(1)。

4. Cache read 如果有 hit，讀出來的資料即為 sram 傳出的 data，否則為 mem_data_i，資料讀給 r_hit_data。因為 r_hit_data 有 256 bits，而 CPU 只要 32 bits，將 CPU_offset 做 alignment(除 4 再乘 4)後乘以 8(轉成 bit)，即可拿來表示要取的資料在 r_hit_data 中的位置，用 for loop 將 32 個 bits 全設給 CPU_data。Cache write 也是如此，先把 w_hit_data 設成 r_hit_data，

然後再用 for loop 將 CPU_data 32bits 傳到 w_hit_data 中對應位置。

5. dcache 中包含另一個 dcache_sram module，負責儲存資料。Input port 分別有 clk_i、rst_i，從 controller 傳入的 addr_i(CPU index)、tag_i(CPU tag bits+ valid + dirty)、data_i(CPU data)、enable_i(是否使用 sram)、write_i(是否寫入 sram); output port 有 tag_o(輸出的 tag bits + valid + dirty)、data_o(sram 中的 data)及 hit_o(表示所求的 block 是否在 sram 中)。對照 2-way set associative cache，Sram 的儲存方式為 16*2 的 tag bits 及 data，我在裡面宣告了一樣大小的 count array，每個 block 都會對應到一個 count 以紀錄其參考時間。初始化時，將 tag、data 及 count array 全設 0，在 clock 來時，如果有 CPU request(enable_i 為 1)及 write_i 為 1(寫入 sram)，表示此動作為 write，會先根據 CPU address 切出 index，index 指出哪一個 set，同時比較 CPU tag 及兩個 block 的 tag，如果比中 0 號，再加上 0 號 block 的 valid bit 為 1 則 hit0 為 1，否則為 0，1 號也是相同方式。確定 hit0 或 hit1 後，將 CPU tag 及 data 寫到此 block 中，將此 block 的 count 改 1，另一個 block 改 0(1 表示最近被參考過)。如果兩個 block 都非所求，則啟動 LRU 機制，看哪一個 block 的 count 為 0 就將 memory 的 tag 及 data 覆蓋此 block(0 表示最近沒被使用)，再修改 count 數字。
如果有 CPU request(enable_i 為 1)但 write_i 為 0，表示此動作為 read。因為沒有寫入 sram，只需做參考時間紀錄，把 read 到的 block count 設 0，另一個設 0。

6. 最後，dcache sram 的輸出。只要有其中一個 block 為 hit，hit_o 就是 1。Tag_o 的部分，如果沒有 enable 就全設 0。否則，如果 hit0/1 輸出 0/1 號 block 的 tag，傳回給 CPU;如果沒有 hit，則 Tag_o 用來作為寫回 memory 用，加上 LRU 機制，所以 Tag_o 為要被置換的 block(count 為 0)。Data_o 的方法跟 tag_o 完全相同。

- ## Difficulties encounterd

Cache control 的運作原理雖然還滿清楚的，但是一到了硬體層面的實作，變得複雜許多，wire 跟 reg 一大堆，我花了不少時間去研究出每個零件的作用是甚麼，同時也透過此次的實作了解硬體的作用機制跟軟體是如此不同。

- ## Development Environment

OS: Windows 10

Compiler: iverilog

IDE: visual studio code