

Computer Architecture 2020

Project 2 Pipelined CPU + L1 Data Cache

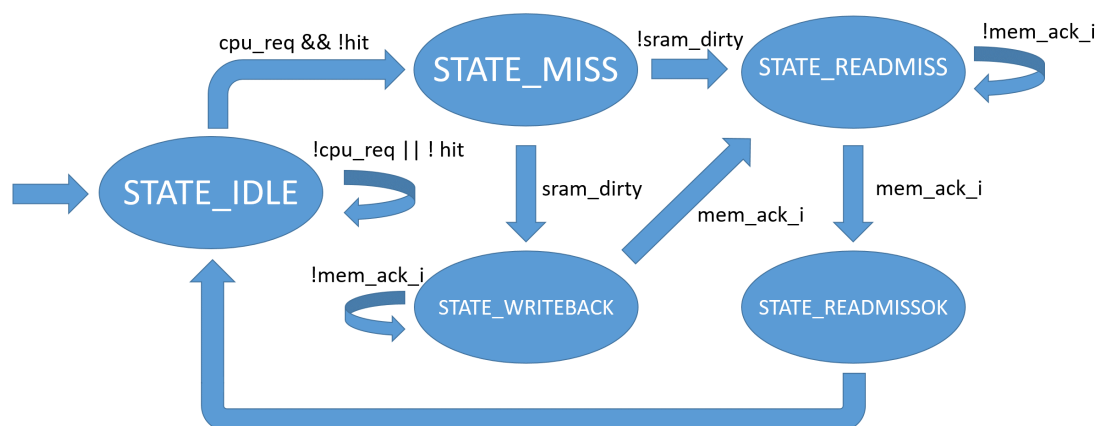
team: RISK-V

Modules Explanation

本次專題的大部分檔案都是直接沿用 project 1 的模組，以下只有列出有更動過和新增的模組。

Dcache Controller

- input :
 - clk_i
 - rst_i
 - mem_data_i
 - mem_ack_i
 - cpu_data_i
 - cpu_addr_i
 - cpu_MemRead_i
 - cpu_MemWrite_i
- output :
 - mem_data_o
 - mem_addr_o
 - mem_enable_o
 - mem_write_o
 - cpu_data_o
 - cpu_stall_o



在這個模組裡面，我們需要做的事情分為三項：

- 接線

大部分模組所需要的接線和轉換，都已經在助教提供的模組裡面有。我們需要多接的只有三個

1. `r_hit_data` :

`r_hit_data` 代表的是當 `hit` 時的資料，因此直接把 `sram_cache_data` assign 給 `r_hit_data`，即可。如果沒有 `hit` 也沒關係，因為不會用到這個資料。

2. `cpu_data` :

放在一個 `always` block 裡面，當 `cpu_offset`、`r_hit_data` 有更動時，把 `cpu_data` 指定成 `r_hit_data` 當中，根據 `cpu_offset` 所指向的內容。由於 `cpu_offset` 是 byte-address，但是我們使用的 bus 是 32-bit，所以需要先轉成 word-address，然後去存取相對應的內容。

3. `w_hit_data` :

放在一個 `always` block 裡面，當 `cpu_offset`、`r_hit_data`、`cpu_data_i` 有更動時，先把 `w_hit_data` 指定成 `r_hit_data` 代表目前的資料，然後根據 `cpu_offset` 對其內容進行修改。addressing 的部分和上面一樣。

● 變數釐清

首先我們需要先釐清每一個控制變數的意義。控制變數有五個：

1. `state`

表示目前的狀態是哪一個，讓後面的 case 可以跳到正確的狀態內。

2. `mem_enable`

當這個設為 `1` 時，Data Memory 會進入讀取資料的 state。

3. `mem_write`

當這個設為 `1` 時，且 Data Memory 的讀取狀態完成 (`mem_ack_i` 為 `1`) 時，Data Memory 會把給他的資料寫入 reg 中。

4. `cache_write`

當這個設為 `1` 時，同時搭配 `cache_sram_enable` 也為 `1` 時，會讓 sram 把給他的資料寫入 cache 中。

5. `write_back`

控制的是給 Data Memory 的 addr 是由誰提供的。若為 `1`，則代表使用 sram 提供的 tag；若為 `0`，則使用 CPU 來的 tag。

● 狀態控制

狀態也有五個，以下分別說明：

1. `STATE_IDLE`

代表待機狀態，當有 `cpu_req` 時，代表 CPU 需要對記憶體進行讀寫，此時，如果是 `hit` 的情況，就代表 sram 有資料，也都會正常的進行讀寫；反之則會進到 `STATE_MISS` 的階段。

2. `STATE_MISS`

不管是讀還是寫，sram 上必須都要先有正確的資料，因此都必須要進到 read miss 的階段，而 Data Memory 也要開始進行讀取。但是如果目前的 sram 的資料是更新過的，代表目前的資料需要被 `write back`。

所以如果 `dirty bit` 為 `1`，則：

- `mem_enable: 1`
- `mem_write: 1`
- `cache_write: 0`
- `write_back: 1`
- `state: 轉成 STATE_WRITEBACK`

如果 `dirty bit` 為 `0`，則：

- `mem_enable: 1`
- `mem_write: 0`
- `cache_write: 0`
- `write_back: 0`
- `state: 轉成 STATE_READMISS`

3. STATE_READMISS

進來之後，我們必須要等 Data Memory 讀取完成，再把資料帶上去 sram 裡面，因此當 `mem_ack_i` 為 1 時：

- `mem_enable`: 0
- `mem_write`: 0
- `cache_write`: 1
- `write_back`: 0
- `state`: 轉成 `STATE_READMISSOK`

否則保持狀態。

4. STATE_READMISSOK

當 read miss 處理完成，我們要回歸到一開始的狀態，把所有的控制變數都設定成 0，然後狀態就可以回到 `STATE_IDLE`。

5. STATE_WRITEBACK

進到 `STATE_WRITEBACK` 之後，我們必須要等 Data Memory 寫入完成之後，才開始進行 read miss 的處理，因此當 `mem_ack_i` 為 1 時：

- `mem_enable`: 1
- `mem_write`: 0
- `cache_write`: 0
- `write_back`: 0
- `state`: 轉成 `STATE_READMISS`

否則保持狀態。

Dcache Sram

- `input` :
 - `clk_i`
 - `rst_i`
 - `addr_i`
 - `tag_i`
 - `data_i`
 - `enable_i`
 - `write_i`
- `output` :
 - `tag_o`
 - `data_o`
 - `hit_o`

本次專題要求的是：

- 32-bit address
- cache size is 1KB
- 32-byte per cache line
- two-way associative cache
- Replacement policy is LRU
- write back and write allocate

在 verilog 裡面，我們的 `tag` 有 25 bits，最高位是 `valid bit`；次高位是 `dirty bit`，剩下的 23 bits 就是 `tag`。另外，助教提供模組時，已經把 address 的各部分都 parse 好了，所以 cache index 的部分可以直接用 `addr_i`。

我們的 sram 分成四個部分

- 初始化
 - 由助教提供的內容。當 `rst_i` 為 1 時，會把 cache 的所有東西都初始化成 0。
 - 裡面也包含要把 `LRU` 全部預設為 0。
- LRU
 - 我們的實做方式是建 16 個 `reg`，代表下一個要被替換掉的位置。預設是 0，當 `hit0` 為 1，就會換成 1；如果是 `hit1` 為 1，則會換成 0；如果都沒有 hit，就會取 `negation (~)`。
- read
 - 以 `wire hit0 hit1` 代表 `tag_i` 的後 23 bits 和 cache 裡面的 `tag` 比對的結果，同時，`vaild bit` 也要是 1。把 `hit0 || hit1` assign 給 `hit_o` 代表有沒有 hit。
 - `data_o` 的部分則是根據 `hit0`、`hit1` 的結果決定要回傳誰；如果都沒有 hit 的話，會回傳 `LRU` 目前指向的那個 `data`。因為在 `miss` 的情況下還需要用 `data_o` 代表是 `write back`，這時候我們需要把要被替換掉（`LRU` 指向的）的資料讀出來。
 - `tag_o` 的部分，因為裡面其實包含 `vaild bit` 和 `dirty bit`，因此都需要回傳，而回傳的規則同上。
- write
 - 當 `enable_i` 和 `write_i` 同時為 1 時，會根據 `hit0` 還是 `hit1` 來決定要更新誰。在更新的過程中，會把 `vaild bit` 和 `dirty bit` 都設定為 1，同時也會更新 `LRU`。如果都沒有 hit 的話，則是根據 `LRU` 的部分來決定要替換掉誰。

Pipeline latches

和 `proj1` 唯一的不同就是每一個 latch 都多了一個 `mem_stall_i` 的 port。當 `mem_stall_i` 為 1 時，latch 就不會進行更新。

CPU

- `input` :
- `output` :

Testbench

- `input` :
- `output` :

Members and Teamwork

- B07209016 鐘晨瑋：修改 `pipeline latch`、完成 `dcache_sram`。
- B07201022 杜宗穎：主要負責 `CPU` 的接線工、`dcache_sram`。
- B07902063 陳耕宇：合作完成 `dcache_sram`、`dcache_controller`。
- 共同撰寫 `dcache_controller`、Report。

Difficulties Encountered and Solutions

Q1. `or`、`|`、`||` 會搞不清楚。

A1. `or` 是用在 `always` block 裡面，代表有改變的時候會執行迴圈；`|` 指的是 bit-wise or；`||` 指的是 logical or。

Q2. 變數多且複雜，不容易理解。

A2. 慢慢的追蹤變數，邊做筆記。以模擬特定狀況的方式來理解每一個變數的意義和過程的轉變。

Development Environment

- OS: Windows 10 Home ver 1909

- Compiler: Icarus Verilog version 11.0