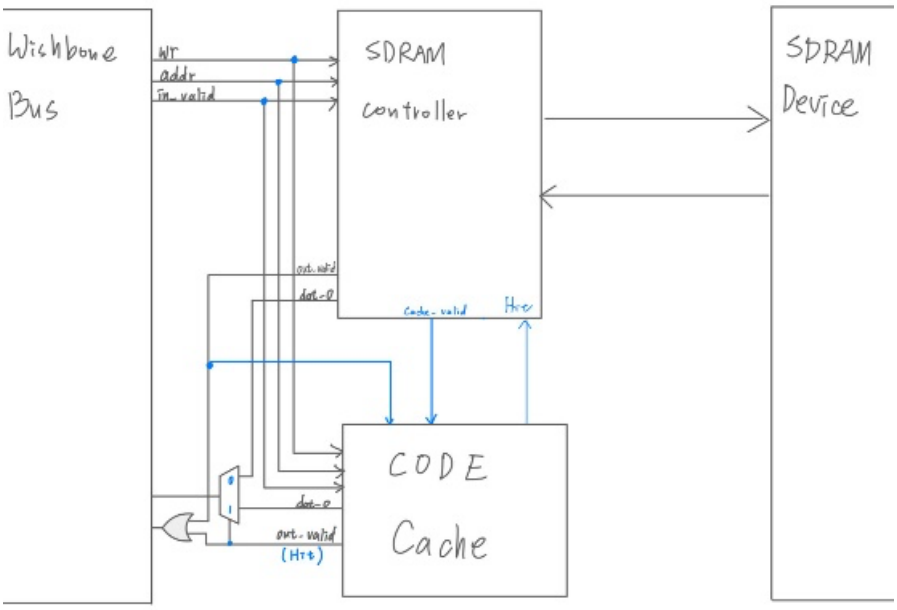
SOC LAB D

312611096 黃鉦淳 / 511506022 何佳玲 / 312605003 王語

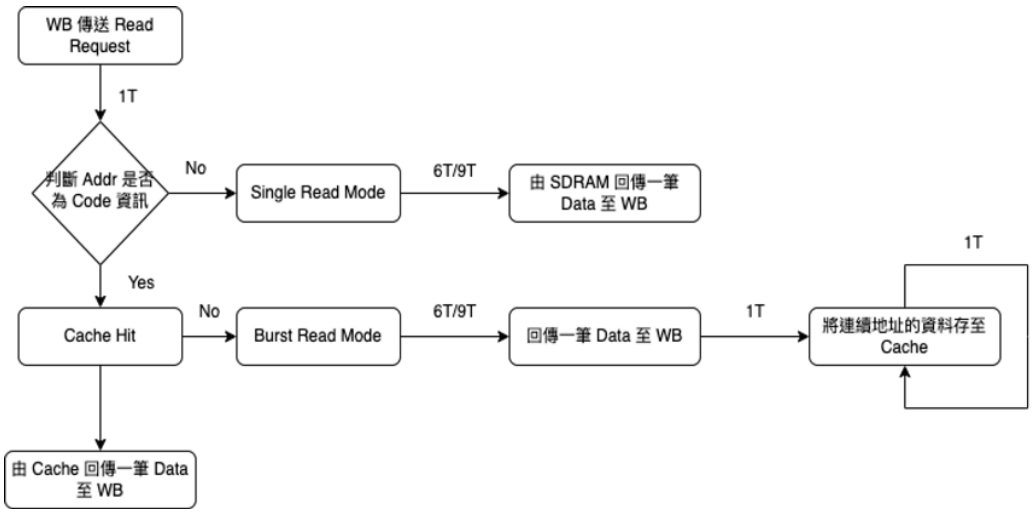
Notion Link : https://irradiated-hellebore-357.notion.site/LAB-D-SDRAM-4128635388f44d099ccf5abd650cbda2?pvs=4

➢ Sdram Controller design

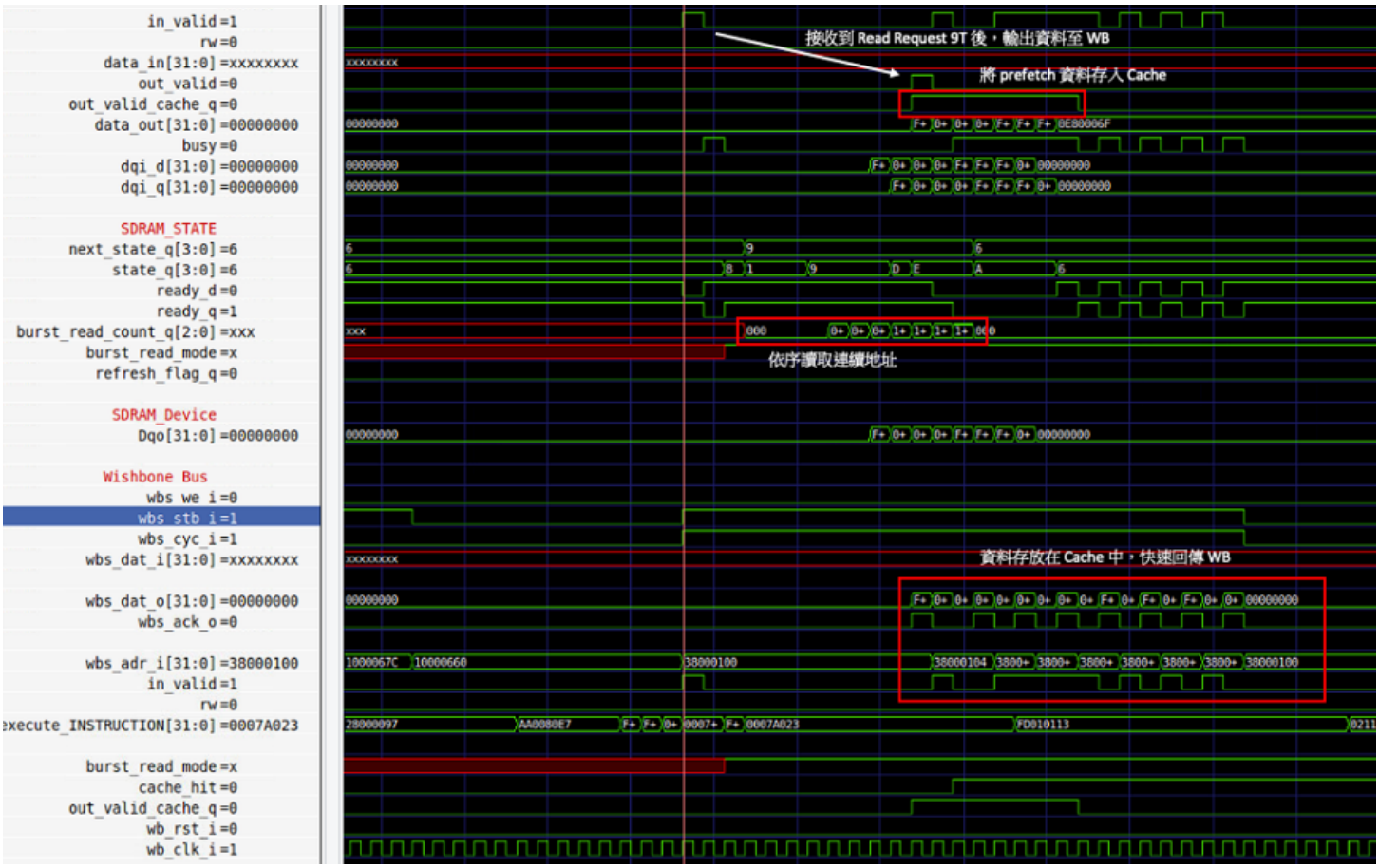
➢ 架構



➢ Flow chart



➢ Waveform

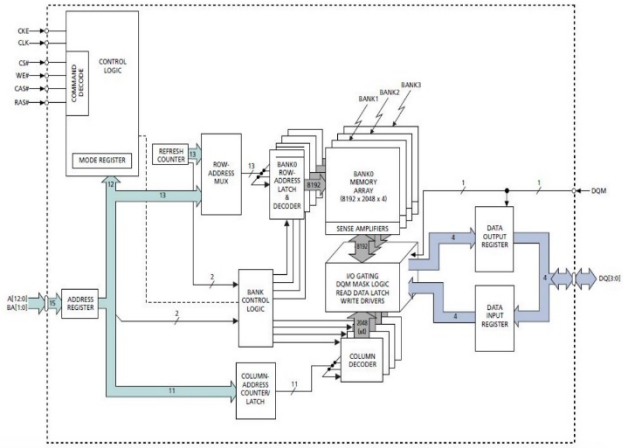

in_valid=1
rw=0
data_in[31:0]=xxxxxxxx
out_valid=0
out_valid_cache_q=0
data_out[31:0]=00000000
busy=0
dqi_d[31:0]=00000000
dqi_q[31:0]=00000000
SDRAM_STATE
next_state_q[3:0]=6
state_q[3:0]=6
ready_d=0
ready_q=1
burst_read_count_q[2:0]=xxx
burst_read_mode=x
refresh_flag_q=0
SDRAM_Device
Dqo[31:0]=00000000
Wishbone Bus
wbs_we_i=0
wbs_stb_i=1
wbs_cyc_i=1
wbs_dat_i[31:0]=xxxxxxxx
wbs_dat_o[31:0]=00000000
wbs_ack_o=0
wbs_adr_i[31:0]=38000100
in_valid=1
rw=0
execute_INSTRUCTION[31:0]=0007A023
burst_read_mode=x
cache_hit=0
out_valid_cache_q=0
wb_rst_i=0
wb_clk_i=1
接收到 Read Request 9T 後，輸出資料至 WB
將 prefetch 資料存入 Cache
依序讀取連續地址
資料存放在 Cache 中，快速回傳 WB

➢ SDRAM bus protocol

➢ SDRAM DEVICE

The behavior model refer to Micron MT48LC64M4A2

- 16 Meg x 4 x 4 banks



[1] Input /output pin:

1. CKE : clk enable
2. CS_N : disable other input, except "CLK" "CKE" "DQM"
3. CAS_N: A[8:0] 列 address
4. RAS_N: A[12:0] 行 address
5. WE_N : write_enable
6. BA[1:0]: 總共 4 個 BLANK
7. A[12:0]: address(不同命令定義不同)
8. DQ[15:0]:data register input/output

[2] decode:

```
// Commands Decode
wire      Active_enable    = ~Cs_n & ~Ras_n &  Cas_n &  We_n;
wire      Aref_enable      = ~Cs_n & ~Ras_n & ~Cas_n &  We_n;
wire      Burst_term       = ~Cs_n &  Ras_n &  Cas_n & ~We_n;
wire      Mode_reg_enable  = ~Cs_n & ~Ras_n & ~Cas_n & ~We_n;
wire      Prech_enable     = ~Cs_n & ~Ras_n &  Cas_n & ~We_n;
wire      Read_enable      = ~Cs_n &  Ras_n & ~Cas_n &  We_n;
wire      Write_enable     = ~Cs_n &  Ras_n & ~Cas_n & ~We_n;
```

經過 decode 會把各項需要執行的描述都在 sdram.v 裡面進行

prech_enable :

若要切換 bank，必須要先將目前正在使用的 bank 關閉後才可以切換新的 bank

➢ Wishbone 和 Sdram controller

因為 Wishbond 只有一個響應訊號 wbs_ack_o 而 Sdram 會由 ctrl_busy&valid 共同決定，因此需要轉換

[1] 轉換 Wishbone 和 SDRAM controller 的訊號

```
assign valid = wbs_stb_i && wbs_cyc_i;
assign ctrl_in_valid = wbs_we_i ? valid : ~ctrl_in_valid_q && valid;
assign wbs_ack_o = (wbs_we_i) ? ~ctrl_busy && valid : ctrl_out_valid;
assign bram_mask = wbs_sel_i & {4{wbs_we_i}};
assign ctrl_addr = wbs_adr_i[22:0];
```

[2] Wishbone address 的末 23 bits 會直接映射到 SDRAM address

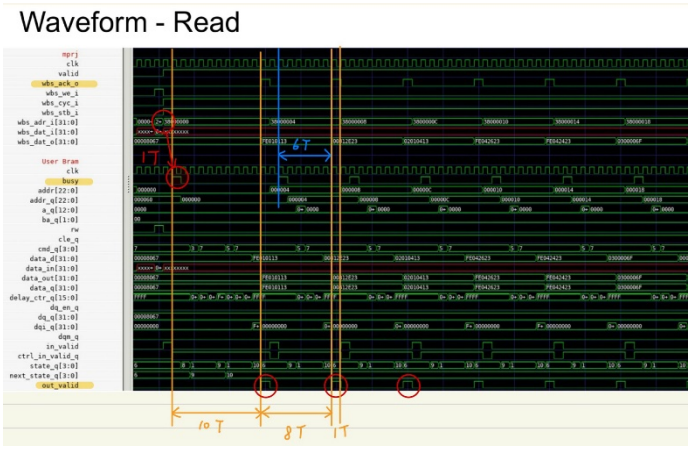
```
always @(posedge clk) begin
        if (rst) begin
            ctrl_in_valid_q <= 1'b0;
        end
        else begin
            if (~wbs_we_i && valid && ~ctrl_busy && ctrl_in_valid_q == 1'b0)
                ctrl_in_valid_q <= 1'b1;
            else if (ctrl_out_valid)
                ctrl_in_valid_q <= 1'b0;
        end
end
```

➢ Introduce the prefectch schme

觀察這三筆訊號之波形，可以看出將「讀取記憶體」任務增加 pipeline 功能，將能使讀取效率提升，可透過前一比地址資訊預測將會使用到的連續記憶體區域，提前讀取至 Cache 中。

➢ Data Prefetch 並存入 Cache

[1] 在原先的 Controller 沒有 Burst mode ，每次讀取資料都需要一定的訪問延遲外加發送 address 的時間 (1T)，若該資料的地址已經被激活 (activate) 則需要 6T ，否則需要 9T 才能得到資料

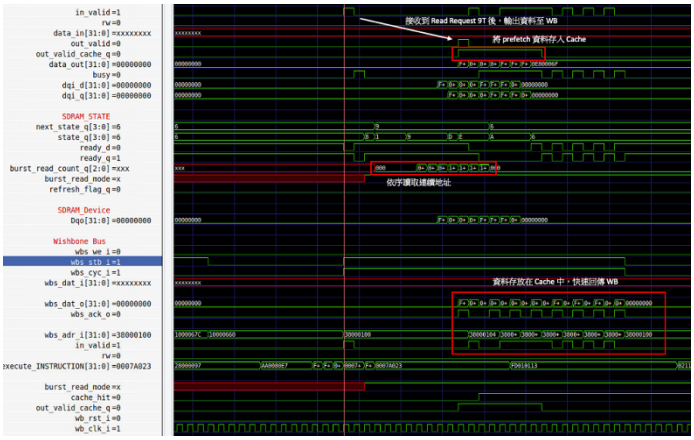


改善: (結果如下圖連續輸出)

[1] 若預先將資料抓取，存在 1T 記憶體中，則訪問只會有 2T 延遲。
sent address (1T) → sent data back & ack (1T)

[2] Wishbone 輸入 address、rw、in_valid 等資訊後，controller 只會經歷 1T Busy，因此可以連續處理多筆訪問

[3] 另外也新增了 cache and burst(會在後面做補充)



➢ Introduce the bank interleave for code and data and modify the linker to load address/data in two different bank

將 Code and data 兩種資料分開至不同的 Bank ，即可方便 prefetch 的設計，可以清楚知道哪些 address 存放的是將要 load 到 PE 做運算的資料。

Bank 判斷:

[1] WB address 的末 22 bits 會直接斷應到 SDRAM address 。

[2] 本實驗沒有調整 mapping 方式，各 bits 代表資訊如下表所示

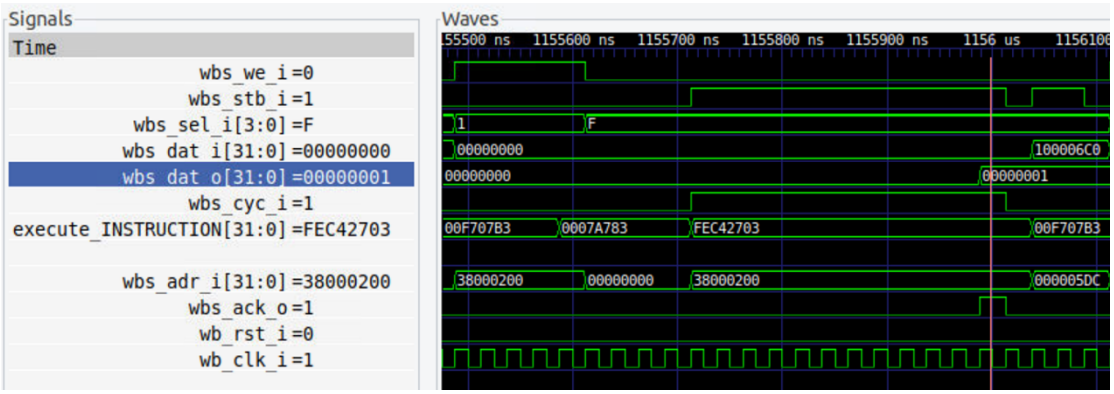
Bank 0	Bank 1	Bank 2	Bank 3
3800_X 0 XX	3800_X 1 XX	3800_X 2 XX	3800_X 3 XX
3800_X 4 XX	3800_X 5 XX	3800_X 6 XX	3800_X 7 XX
3800_X 8 XX	3800_X 9 XX	3800_X A XX	3800_X B XX
3800_X C XX	3800_X D XX	3800_X E XX	3800_X F XX

sections.lds

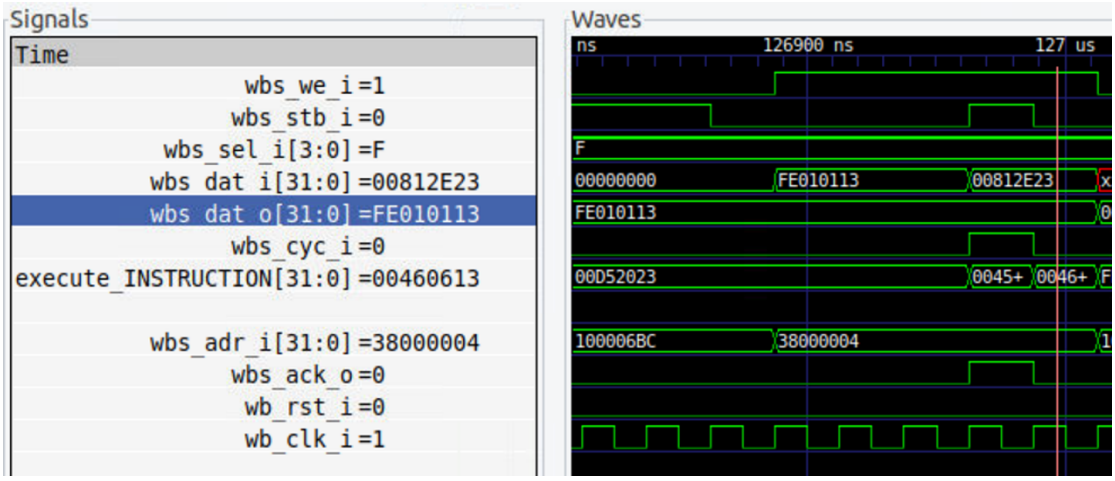
分離前:

```
MEMORY {
  vexriscv_debug : ORIGIN = 0xf00f0000, LENGTH = 0x00000100
  dff : ORIGIN = 0x00000000, LENGTH = 0x00000400
  dff2 : ORIGIN = 0x00000400, LENGTH = 0x00000200
  flash : ORIGIN = 0x10000000, LENGTH = 0x01000000
  mprj : ORIGIN = 0x30000000, LENGTH = 0x00100000
  mprjram : ORIGIN = 0x38000000, LENGTH = 0x00400000
  hk : ORIGIN = 0x26000000, LENGTH = 0x00100000
  csr : ORIGIN = 0xf0000000, LENGTH = 0x00010000
}
```

Data Fetch:(In Bank 2)



Code Execution(In Bank 0 ~ 3)

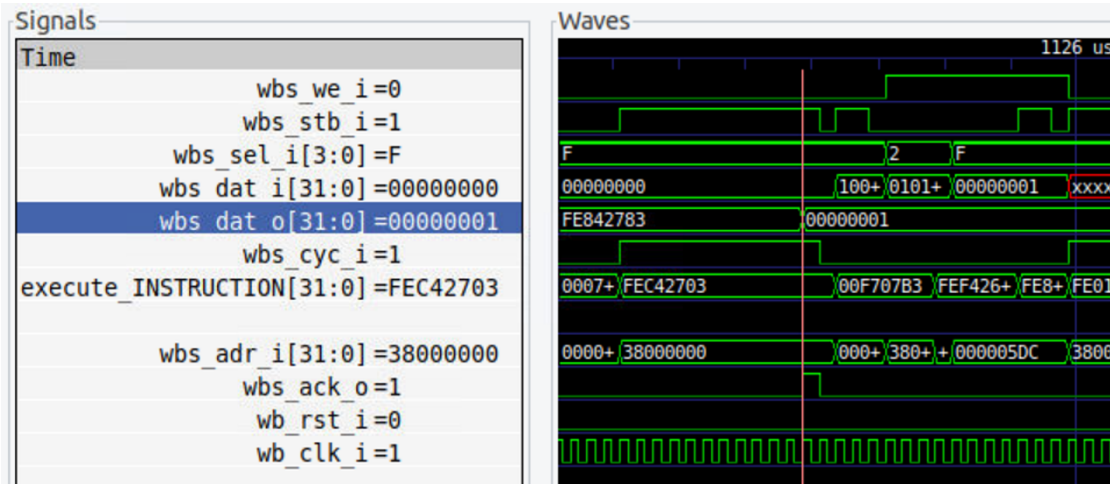


分離後:

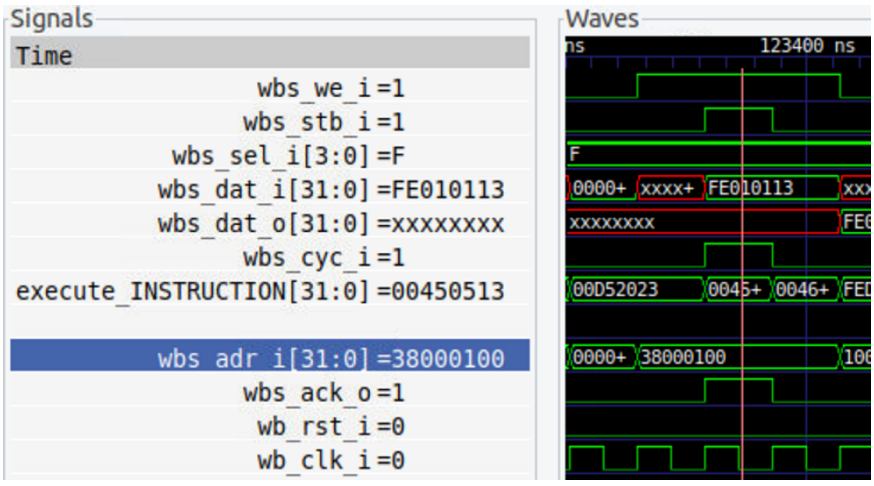
```
MEMORY {
  vexriscv_debug : ORIGIN = 0xf00f0000, LENGTH = 0x00000100
  dff : ORIGIN = 0x00000000, LENGTH = 0x00000400
  dff2 : ORIGIN = 0x00000400, LENGTH = 0x00000200
  flash : ORIGIN = 0x10000000, LENGTH = 0x01000000
  mprj : ORIGIN = 0x30000000, LENGTH = 0x00100000
  /* mprjram : ORIGIN = 0x38000000, LENGTH = 0x00400000 */
  mprjram : ORIGIN = 0x38000100, LENGTH = 0x00000300
  add_data : ORIGIN = 0x38000000, LENGTH = 0x00000100
  /* code_and_others : ORIGIN = 0x38000100, LENGTH = 0x00000300 */
  hk : ORIGIN = 0x26000000, LENGTH = 0x00100000
  csr : ORIGIN = 0xf0000000, LENGTH = 0x00010000
}
```

```
.data :
{
  . = ALIGN(8);
  _fdata = .;
  *(.data .data.* .gnu.linkonce.d.*)
  *(.data1)
  _gp = ALIGN(16);
  *(.sdata .sdata.* .gnu.linkonce.s.*)
  . = ALIGN(8);
  _edata = .;
} > add_data AT > flash
```

Data Fetch (In Bank 0)



Code Execution(In Bank 1~3)



➢ Ohter SDRAM access conflicts with SDRAM refresh (refresh period)

我們新增了 Prefetch 的新功能，收到地址後會預先判斷該地址之資料是否為 Code 或是 Data ，若為 Code 資訊，則會用 Burst Mode 連續讀取 8 個連續記憶體資料 (已經為本實驗任務特化，省略 program burst length 步驟 ，但是 Prefetch 會衍生出 Refresh issue ：

- Refresh cycle：750T
- CAS：3T
- Prefetch data num：8 addresses

若要連續 prefetch 多筆資料，且不想要被中斷，可能讓 Refresh 時間超出 750 T，並且不是只超出一點點。

Ex : Time=749 T 時 Prefetch ，則實際 Refresh 時間為 749+24 = 773 T

solution：將 Refresh cycle 縮短至 750- 3*8 T = 726 T

當遇上 refresh 時，sdram 的 access 請求會被延後處理。

➢ OTHER

FSM:

將原先的 read mode 調整成 Burst read mode .

原先的 read mode

訪問地址是已經激活的 Row

IDLE >ACTIVATE>WAIT>READ>WAIT>ReadRes>IDLE

訪問地址是尚未經激活的 Row

IDLE >READ>WAIT>ReadRes>IDLE

需要等待 SDRAM Device 處理，所以會經歷 Wait ，但是 burst 模式就能省略這些時間。

Burst read mode

IDLE >(ACTIVAT)>READ>ReadAndOutput>ReadRes>IDLE

READ：只進行資料讀取，不輸出資料。

ReadAndOutput：進行資料讀取與輸出資料。

ReadRes：不進行資料讀取，只輸出資料。

➢ 進階優化

此優化是針對 Add 任務進行特化，無法直接套用於 Matrix mult task
觀察波形圖可以發現，同樣地址的 Code 會被重複訪問多次，若增加 Cache size 就可以加以保存，下次需要訪問時就不需要等待 9T 時間，只要 Cache Hit 僅需 2T 就能獲得資料。

(宣告一個 2D Cache 以及對應 tag)

reg [31:0] cache [0:8] [0:7] ;
reg hit_array [0:8] [0:7] ;

Cache size 是為了 Add task 特化，此任務會涉及 0100~021C 等地址，共需 9 column 與 8 row。

因為 Code 資訊在編譯後即可確定，不會有覆寫問題，Cache size 也足夠存放所有 Code 資訊，因此僅需要紀錄該地址是否為空，就能知道該地址是否存放在 Cache 中。

FSM - saving_state (從 SDRAM 存取資料)

IDLE :

若 miss ，進入 SAVING 狀態，等待 SDRAM 將該八組連續地址資訊傳送至 Cache ，並將 hit array 更新為 1’ b1。若 Hit ， 維持 IDLE，不做任何動作。

SAVING :

接收八筆資料後會到 IDLE 狀態。

FSM - output_state (發送資料至 WB)

IDLE :

當發生有效讀取訪問，且地址為 Code 資訊，紀錄該地址。
若 saving_state 處於 IDLE 狀態 :
若 Hit ，進入 OUTPUT 狀態。
若 ~Hit ，維持 IDLE 狀態。
若 saving_state 處於 Saving 狀態 :
持續比對先前紀錄之地址是否 Hit 。
若 Hit ，進入 OUTPUT 狀態。
若 ~Hit ，維持 IDLE 狀態。
(有可能一開始 Miss ，但是 prefetch 過程發生 Hit)

OUTPUT :

Output valid = 1 後回到 IDLE。

並將 Output_flag 設置為 1’ b1。

備注：Output_flag 是為了避免所紀錄地址一再使 Hit 發生而輸出無效資料造成錯誤，Output_flag 會在每次收到有效讀取請求時重置。

➢ SDRAM address 與 cache 地址之映射

SDRAM address 與 Cache 地址之映射

origin (column)

DC:Dont care DC

user_addr [09|08|07|06|05]

0x3800_0100 --> 100 --> 0001_0000_0000 --> 0 1 0 0 0 --> 0

0x3800_0120 --> 120 --> 0001_0010_0000 --> 0 1 0 0 1 --> 1

0x3800_0140 --> 140 --> 0001_0100_0000 --> 0 1 0 1 0 --> 2

0x3800_0160 --> 160 --> 0001_0110_0000 --> 0 1 0 1 1 --> 3

0x3800_0180 --> 180 --> 0001_1000_0000 --> 0 1 1 0 0 --> 4

0x3800_01A0 --> 1A0 --> 0001_1010_0000 --> 0 1 1 0 1 --> 5

0x3800_01C0 --> 1C0 --> 0001_1100_0000 --> 0 1 1 1 0 --> 6

0x3800_01E0 --> 1E0 --> 0001_1110_0000 --> 0 1 1 1 1 --> 7

0x3800_0200 --> 200 --> 0010_0000_0000 --> 1 0 0 0 0 --> 8

offset (row)

[4|3|2]

0x00 --> 0 0 0

0x04 --> 0 0 1

0x08 --> 0 1 0

0x0C --> 0 1 1

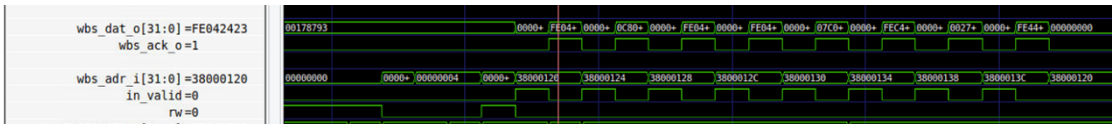
0x10 --> 1 0 0

0x14 --> 1 0 1

0x18 --> 1 1 0

0x1C --> 1 1 1

Waveform:



因為先前已訪問過 0x3800_0120，並將該地址資料存在 Cache 中，因此僅需要 2T 就能回傳資料。