

Lab #D – SDRAM  
Controller and SDRAM

Group5

Members:

M11207415 陳謝鎧

M11207002 陳泊佑

M11107426 廖千慧

M11207328 吳奕帆

## ● SDRAM controller design

**INIT:** 初始狀態，初始化 SDRAM，在正常操作 SDRAM 前須先進行初始化。即對內部邏輯控制單元進行初始化。下一步在這次的 lab 中會直接進到 IDLE 這個階段。

*INIT>IDLE*

**IDLE:** 空閒狀態，當 bank 被 precharged 後等一段時間 SDRAM 將會處於 IDLE 階段

**ACTIVATE(row activate):** 當處於 IDLE 狀態時，收到控制訊號後 bank 和 row 被選中後的狀態。

*IDLE>ACTIVATE*

**WRITE:** 在處於 row activate 時，收到 WRITE 後 bank 和 column 被選中後的狀態，則執行資料的寫入。

*IDLE>ACTIVATE>WRITE>IDLE* 同個 bank 中的寫入步驟

*IDLE>PRECHARGE>ACTIVATE>WRITE>IDLE* 不同 bank 的寫入步驟

**READ:** 在 row activate 時，收到 READ 後 bank 和 column 被選中後的狀態，負責執行 SDRAM 讀取的步驟。

*IDLE>ACTIVATE>READ>READ\_RES>IDLE* 同個 bank 中的讀取步驟

*IDLE>PRECHARGE>ACTIVATE>READ>READ\_RES>IDLE* 不同 bank 的讀取步驟

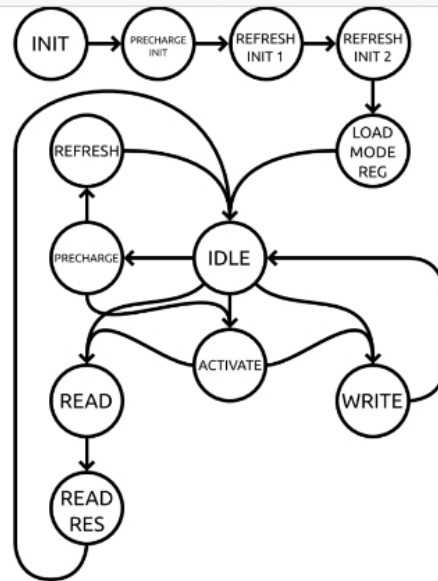
**READ\_RES:** read 完後進行 reset。

**PRECHARGE:** 預充電，當需要將 row 關閉時則需要進行 precharge，即若要更換不同的 bank 進行讀寫動作時即需要先進行 precharge 關閉原本的 bank 和 row。

**REFRESH:** 為了保持 SDRAM 的數據一直都在，需要不停的刷新，主要分為兩種，auto refresh 和 self refresh。在執行 auto refresh 時需要先將所有的 active bank 進行 precharge。

## SDRAM Controller

- INIT→IDLE
- IDLE→ACTIVATE→WRITE→IDLE
- IDLE→ACTIVATE→READ→READ\_RES→IDLE
- IDLE→WRITE→IDLE
- IDLE→READ→READ\_RES→IDLE
- IDLE→PRECHARGE→ACTIVATE→WRITE→IDLE
- IDLE→PRECHARGE→ACTIVATE→READ→READ\_RES→IDLE
- IDLE→PRECHARGE→REFRESH→IDLE



## ● SDRAM bus protocol

外部：

### SDRAM Device

- sdram\_cle, sdram\_cs, sdram\_cas, sdram\_ras, sdram\_we
- sdram\_dqm, sdram\_ba, sdram\_a
- sdram\_dqi, sdram\_dqo

clk:時脈信號

Cke(sdram\_sle):為 clk 信號的 enable

Cs\_n(sdram\_cs):片選信號。

Cas\_n(sdram\_cas):行選信號，輸入資料為行地址的選擇。

Ras\_n(sdram\_ras):列選信號，輸入資料為列地址的選擇。

We\_n(sdram\_we):此為 write 的 enable 信號。

Dqm(sdram\_dqm):數據掩碼(Data I/O Mask)，可以控制 I/O 端口取消那些輸入或輸出的數據。

Ba(sdram\_ba):Bank 的位址，主要用於選擇要用哪一個 bank。

Addr(saram\_a):位址總線，依據當下情況可能為 row 或 column 的 adress。

Dqi(sdram\_dqi):原 in-out pin 的輸入 input。

Dqo(sdram\_dqo): 原 in-out pin 的輸出 output。

內部：

```
// 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;
```

Activate\_enable:用以將選定的 row 的 enable。

Aref\_enable:auto refresh 的 enable。

Burst\_term: SDRAM 處於讀/寫操作過程中可被寫入，突發停止操作被用來截斷固定長度或者整頁長度的突發，執行突發停止命令後，最近執行的數據讀/寫操作被終止，此命令操作並不會通過預充電關閉當前激活列，需通過預充電操作關閉被激活列。

Mode\_reg\_enable:Mode register 的 enable signal。

Prech\_enable: Precharge 的 enable signal。

Read\_enable:讀取的 enable，激活 column。

Write\_enable:寫入的 enable，激活 column。

## ● Introduce the prefetch scheme

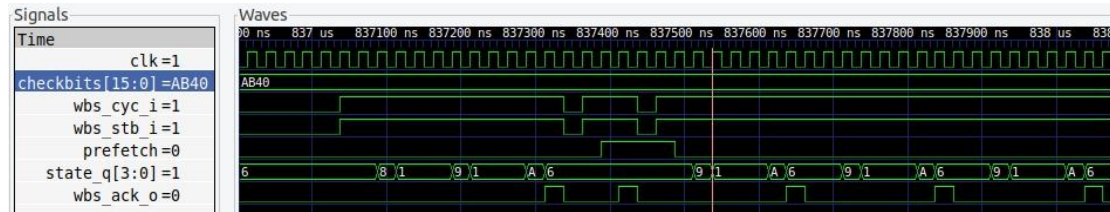
```
//////////set prefetch condition
always@(posedge clk)begin
    if(in_valid)
        prefetch_address <= addr + 22'd4;
    else if(!in_valid && out_valid_q)
        prefetch_address <= 0;

    if((prefetch_address == addr) && in_valid && !rw)
        prefetch <= 1;
    else if((prefetch_address != addr) && in_valid && !rw)
        prefetch <= 0;
    else if(rw)
        prefetch <= 0;
end
```

這是設定是否要 prefetch 的條件，首先當 in\_valid 等於 1，prefetch\_address 會加 4，接著再來判別 prefetch 什麼時候要為 1，並會在 IDLE state 判斷是否要進行 prefetch。

在實作過程中，本來我們是把 prefetch 的 bank access 寫在 READ\_RES state 並在 IDLE 端判別是否要 prefetch，假設有 prefetch 則跳到 READ\_RES state 去 access prefetch addr，雖然這樣有 prefetch，但假如第二次讀取的是同一個 BANK 時，他還是要再重新 ACTIVATE，所以後來我們就在 idle 也加了

prefetch 的 bank access，這樣假設第二次讀取同一個 bank 就不需要在 ACTIVATE，也就是說第一次進行 READ 的時候會需要先進行 ACTIVATE，這個部分需要有 3 個 T 的等待時間，而在第二次如果讀取的是同一個 BANK 時，則只需要 3 個 T 的 READ 來讀取資料就可以了，所以 READ\_RES 所花的時間都只有 1 個 T，這樣也可以省掉 ACTIVATE 3 個 T 的等待時間，如下圖所示。



## ● Introduce the bank interleave for code and data

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

.bss :
{
    . = ALIGN(8);
    _fbss = .;
    *(&.dynsbss)
    *(&.sbss .sbss.* .gnu.linkonce.sb.*)
    *(&.scommon)
    *(&.dynbss)
    *(&.bss .bss.* .gnu.linkonce.b.*)
    *(&.COMMON)
    . = ALIGN(8);
    _ebss = .;
    _end = .;
} > mprj AT > flash
```

```
580 Disassembly of section .bss:
581
582 30000000 <flag>:
583 30000000: 0000 .2byte 0x0
584 ...
585
586 30000004 <result>:
587 ...
588
589 Disassembly of section .mprjram:
590
591 38000000 <matmul>:
592 38000000: fd010113 addi sp,sp,-48
593 38000004: 02112623 sw ra,44(sp)
594 38000008: 02812423 sw s0,40(sp)
595 3800000c: 03010413 addi s0,sp,48
596 38000010: fe042623 sw zero,-20(s0)
597 38000014: fc042e23 sw zero,-36(s0)
598 38000018: fe042623 sw zero,-20(s0)
599 3800001c: 0e80006f j 38000104 <matmul+0x104>
600 38000020: fe042423 sw zero,-24(s0)
601 38000024: 0c80006f j 380000ec <matmul+0xec>
602 38000028: fe042023 sw zero,-32(s0)
603 3800002c: fe042223 sw zero,-28(s0)
604 38000030: 07c0006f j 380000ac <matmul+0xac>
605 38000034: fec42783 lw a5,-20(s0)
```

Disassembly of section .data:

```
30000000 <A>:
30000000: 0000 .2byte 0x0
30000002: 0000 .2byte 0x0
30000004: 0001 .2byte 0x1
30000006: 0000 .2byte 0x0
30000008: 0002 .2byte 0x2
3000000a: 0000 .2byte 0x0
3000000c: 00000003 lb zero,0(zero) # 0 <__DYNAMIC>
30000010: 0000 .2byte 0x0
30000012: 0000 .2byte 0x0
30000014: 0001 .2byte 0x1
30000016: 0000 .2byte 0x0
30000018: 0002 .2byte 0x2
3000001a: 0000 .2byte 0x0
3000001c: 00000003 lb zero,0(zero) # 0 <__DYNAMIC>
```

透過 section 來指定放 code 跟 data 的位置。

```
// request signal
wire req_data;
wire req_code;
wire[1:0] data_bank; //data
wire [22:0] data_addr;
wire[1:0] code_bank; //code
wire [22:0] code_addr;
assign req_data = (wbs_stb_i & wbs_cyc_i & (wbs_adr_i[31:24] == 8'h30)) ? 1 : 0;
assign req_code = (wbs_stb_i & wbs_cyc_i & (wbs_adr_i[31:24] == 8'h38)) ? 1 : 0;
assign data_bank = (wbs_adr_i[9:8] > 2'b01) ? (wbs_adr_i[9:8] - 2'b10) : wbs_adr_i[9:8];
assign data_addr = (req_data) ? {wbs_adr_i[22:10], data_bank, wbs_adr_i[7:0]} : 0;
assign code_bank = (wbs_adr_i[9:8] < 2'b10) ? (wbs_adr_i[9:8] + 2'b10) : wbs_adr_i[9:8];
assign code_addr = (req_code) ? {wbs_adr_i[22:10], code_bank, wbs_adr_i[7:0]} : 0;
```

接著在利用 wbs\_adr\_i[31:24]、wbs\_stb\_i、wbs\_cyc\_i 來區分 req\_code(3800)還是 req\_data (3000)，若 request\_code 為 1，表示需要看 code\_adr 並依據其數據選擇要哪個 bank，bank 的選擇為 2(10)或 3(11)。  
code\_addr={wbs\_adr\_i[22:10], code\_bank, wbs\_adr\_i[7:0]}  
若 req\_data 為 1，表示需要看 data\_addr 的位址，選定要哪一個 bank，bank 的選擇為 0(00)或 1(01)。  
data\_addr={wbs\_adr\_i[22:10], data\_bank, wbs\_adr\_i[7:0]}

## ● Introduce how to modify the linker to load

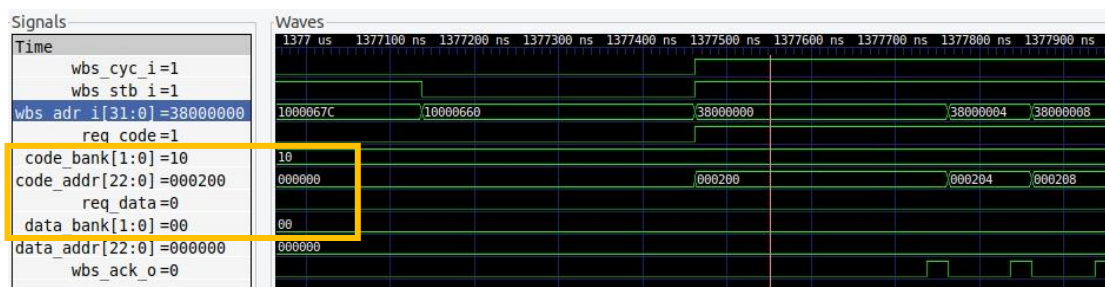
### address/data in two different bank

要在 user 裡面來設定要使用哪一個 bank，下圖是程式碼，我們是判斷 wbs\_adr\_i[9:8] 的值來決定要放哪一個 bank，若 wbs\_adr\_i[9:8] > 01 的話，則 wbs\_adr\_i[9:8] - 2'b10 來決定 data\_bank(00 或 01)，若 wbs\_adr\_i[9:8] < 10 的話，則 wbs\_adr\_i[9:8] + 2'b10 來決定 code\_bank(10 或 11)。

```
wire[1:0] data_bank;
assign data_bank = (wbs_adr_i[9:8] > 2'b01) ? (wbs_adr_i[9:8] - 2'b10) : wbs_adr_i[9:8];
wire [22:0] data_addr;
assign data_addr = (req_data) ? {wbs_adr_i[22:10], data_bank, wbs_adr_i[7:0]} : 0;

wire[1:0] code_bank;
assign code_bank = (wbs_adr_i[9:8] < 2'b10) ? (wbs_adr_i[9:8] + 2'b10) : wbs_adr_i[9:8];
wire [22:0] code_addr;
assign code_addr = (req_code) ? {wbs_adr_i[22:10], code_bank, wbs_adr_i[7:0]} : 0;
```

我們可以從下圖的波型看出 code\_bank 選到 10 的位置，而 data\_bank 選到 00 的位置。





- Observe SDRAM access conflicts with SDRAM

## refresh (reduce the refresh period)

在 SDRAM 中只有通過 refresh 這項技術通過刷新操作才得以保持數據的可靠信。但是當對於同一個 section 若同時發生系統訪問和 refresh 時便會發生衝突。為避免此種狀況發生可以降低 refresh 的週期使發生衝突的機會降低。

- Result

這邊我們是使用 02 來優化 compiler，並透過 checkbit 檢查是否有完成矩陣功能。

```
ubuntu@ubuntu2004:~/Desktop/soclab-sdram/testbench/counter_la_mm$ source run_clean
ubuntu@ubuntu2004:~/Desktop/soclab-sdram/testbench/counter_la_mm$ source run_sim
Reading counter_la_mm.hex
counter_la_mm.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la_mm.vcd opened for output.
LA Test 1 started
counter_la_mm_tb.uut.mprj.mprj.user_bram : at time 4896263000 ERROR: Bank is not Activated for Read
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x003e
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0044
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x004a
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0050
LA Test 2 passed
```

啟用  
移至 1