

Lab D SDRAM

112061603 李柏叡

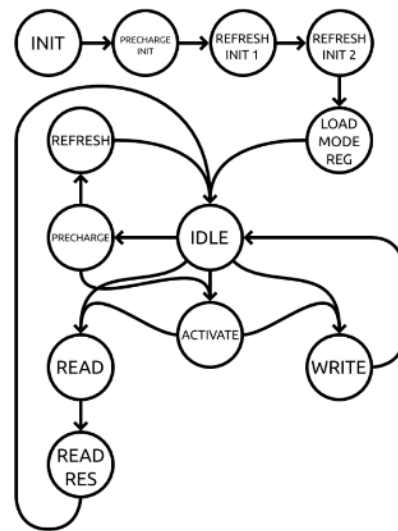
112061620 郭邑哲

112061621 貢曄家

- the SDRAM controller design, SDRAM bus protocol ...
SDRAM controller design:

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

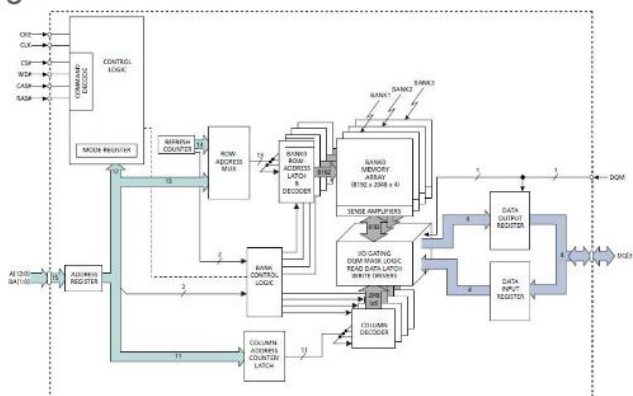


1. INIT:初始化記憶體的參數和設定
2. IDLE:等待接收記憶體的操作命令
3. ACTIVATE:激活特定記憶體的 ROW
4. WRITE: 向已激活的 ROW 寫入資料
5. READ: 向已激活的 ROW 讀取資料
6. READ_RES:Read 完後進行 RESET
7. PRECHARGE:關閉記憶體，準備下一操作
8. REFRESH:刷新記憶體

SDRAM bus protocol:

The behavior model refer to Micron MT48LC64M4A2

- 16 Meg x 4 x 4 banks



1. CLK : Clk 信號
2. CKE : Clk 信號的 enable
3. CS_N : disable 其他 input , except CLK , DQM ,CKE
4. CAS_N : Column address
5. RAS_N : Row address
6. WE_N : Write enable
7. DQM : Data I/O Mask
8. BA : Address of Bank
9. ADDR : Address
10. DQI : Data register input
11. DQO : Data register output

Decode 內部:

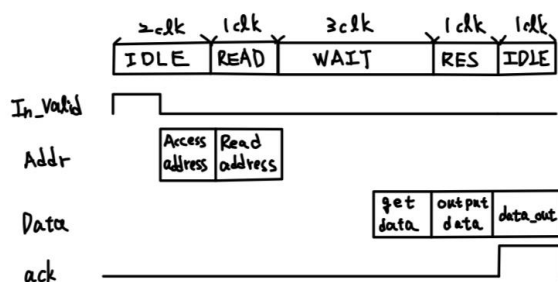
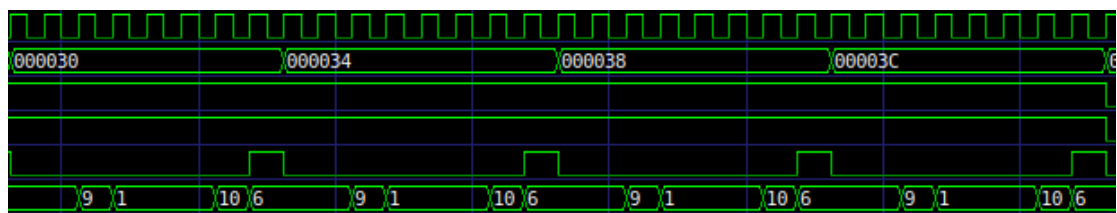
```
// Commands Decode
wire Active_enable = ~Cs_n & ~Ras_n & Cas_n & We_n;
wire Aref_enable   = ~Cs_n & ~Ras_n & ~Cas_n & We_n;
wire Burst_term    = ~Cs_n & Ras_n & Cas_n & ~We_n;
wire Mode_reg_enable = ~Cs_n & ~Ras_n & ~Cas_n & ~We_n;
wire Prech_enable   = ~Cs_n & ~Ras_n & Cas_n & ~We_n;
wire Read_enable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

經 Decode 後，將要執行的命令在 SDRAM 進行。

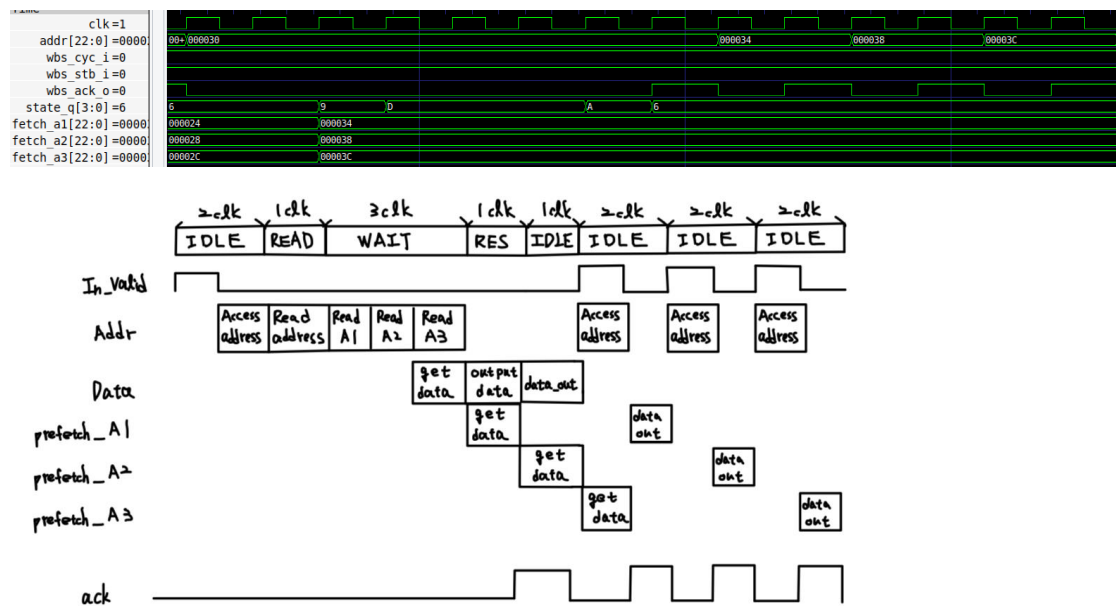
● Introduce the prefetch scheme

為了實作出 prefetch 功能，我們加入了三個 prefetch buffer(為了完全利用 READ state 後的三個 WAIT，因此 prefetch buffer 數量也為 3)來先行預測下一個指令，若成功預測則只需等待兩個 clock cycle 就能讀出，但若預測失敗則一樣需要 8 個 cycle，並會進行下一次的 prefetch。

原:執行四個指令需要 32 個 clk



加入 prefetch 後:執行四個指令需要 14 個 clk



無 Prefetch 的 clock cycles:

```
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
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
-----clock cycle:      64559-----
LA Test 2 passed
```

有 Prefetch 的 clock cycles:

```
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
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
-----clock cycle:      43517-----
LA Test 2 passed
```

- Introduce the bank interleave for code and data
Bank interleave: Using bank interleave to improve access efficiency，將 memory

分成多個獨立 bank，而這些 bank 能同時進行寫入和讀取的功能，提高效率，並減少了等待存取的時間。

透過 section.ids，我們可以修改 code 和 data 的地址，將 Code 和 data 分別放置不同的 Bank，並藉由修改 Origin 的 Bank address，放置在不同 Bank，而達成 Bank interleave 的效果。

```
assign Mapped_RA = user_addr[22:10];
assign Mapped_BA = user_addr[9:8];
assign Mapped_CA = user_addr[7:0];
```

```
mprjram : ORIGIN = 0x38000000, LENGTH = 0x00000200
//code
mprjram_bank2 : ORIGIN = 0x38000200, LENGTH = 0x00000600
//data
```

- Introduce how to modify the linker to load address/data in two different bank

為了使 code execution 和 data fetch 分別在不同 bank，首先我們觀察 code execution 的資料所需記憶體大小

38000130:	380007b7	lui	a5,0x38000
38000134:	28478793	addi	a5,a5,644 # 38000284 <result>
38000138:	00078513	mv	a0,a5
3800013c:	02c12083	lw	ra,44(sp)
38000140:	02812403	lw	s0,40(sp)
38000144:	03010113	addi	sp,sp,48
38000148:	00008067	ret	

發現需要用到第二個 bank

因此配置兩個 bank 給 code execution，其餘為 data fetch

```
.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 = 0x00000200
    mprjram_bank2 : ORIGIN = 0x38000200, LENGTH = 0x00000600
    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 = .;
} > mprjram_bank2 AT > flash

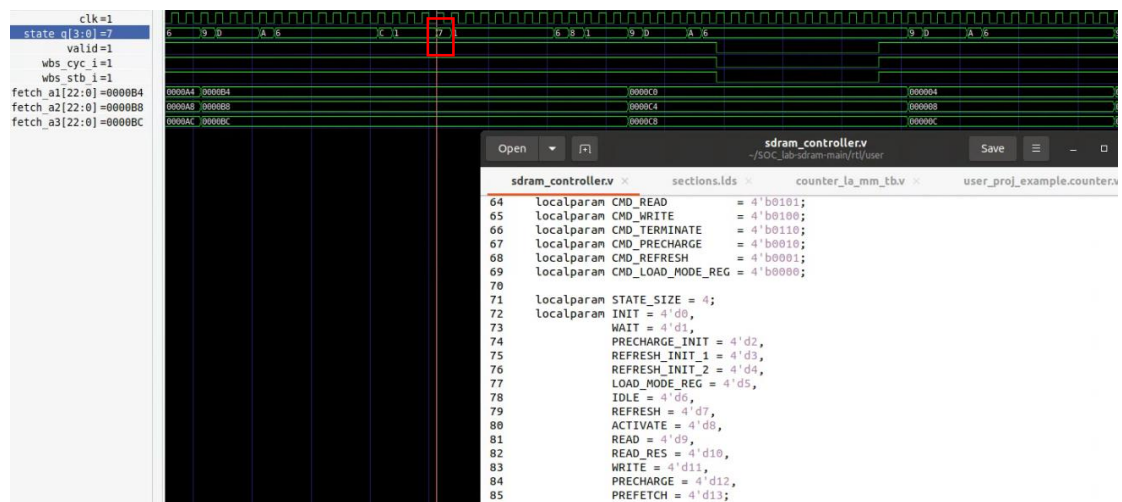
.bss :
{
    . = ALIGN(8);
    _fbss = .;
    *(.dynsbss)
    *(.sbss .sbss.* .gnu.linkonce.sb.*)
    *(.scommon)
    *(.dynbss)
    *(.bss .bss.* .gnu.linkonce.b.*)
    *(COMMON)
    . = ALIGN(8);
    _ebss = .;
    _end = .;
} > mprjram_bank2 AT > flash

.mprjram :
{
    . = ALIGN(8);
    _fsram = .;
    *libgcc.a:*(.text .text.*)
} > mprjram AT > flash

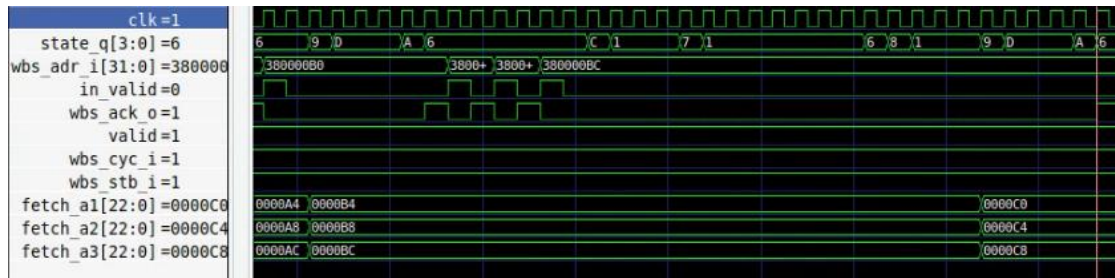
```

- Observe SDRAM access conflicts with SDRAM refresh (reduce the refresh period)

SDRAM 在運作時，需定期 refresh 才能儲存正確的 data，但 refreshing takes plenty of time，因此當 SDRAM 的同一個 section 發生 access 和 refresh 時，便會發生 conflicts，而為了減少 conflicts，我們可以透過 reduce the refresh period，已達成此目的。



下圖情況會有 conflicts，造成 read 出一個 data 耗更多 clk(23clk)。



● Others

為了讓運算速度加快，我們參考 Lab06 做法，發現將 gcclib 乘法運算從原本的 flash 搬進 mprjram 中能讓 CPU 在讀取這個運算時的時間減少不少。

原：

```
.text :
{
    _ftext = .;
    /* Make sure crt0 files come first, and they, and the isr */
    /* don't get disposed of by greedy optimisation */
    *crt0*(.text)
    KEEP(*crt0*(.text))
    KEEP(*(.text.isr))

    *(.text .stub .text.* .gnu.linkonce.t.*)
    _etext = .;
} > flash
```

```
10000718 <__mulsi3>:
10000718:      00050613      mv      a2,a0
1000071c:      00000513      li      a0,0
10000720:      0015f693      andi    a3,a1,1
10000724:      00068463      beqz    a3,1000072c <__mulsi3+0x14>
10000728:      00c50533      add     a0,a0,a2
1000072c:      0015d593      srli    a1,a1,0x1
10000730:      00161613      slli    a2,a2,0x1
10000734:      fe0596e3      bnez    a1,10000720 <__mulsi3+0x8>
10000738:      00008067      ret
```

運算速度：

```
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
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
-----clock cycle: 252375-----
LA Test 2 passed
```

改：

```

.mprjram :
{
    . = ALIGN(8);
    fsram = .;
    *libgcc.a:*(.text .text.*)
} > mprjram AT > flash

38000000 <__mulsi3>:
38000000:      00050613          mv      a2,a0
38000004:      00000513          li       a0,0
38000008:      0015f693          andi      a3,a1,1
3800000c:      00068463          beqz     a3,38000014 <__mulsi3+0x14>
38000010:      00c50533          add      a0,a0,a2
38000014:      0015d593          srli     a1,a1,0x1
38000018:      00161613          slli     a2,a2,0x1
3800001c:      fe0596e3          bnez     a1,38000008 <__mulsi3+0x8>
38000020:      00008067          ret

```

運算速度:

```

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
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
-----clock cycle:      43517-----
LA Test 2 passed

```

$(252375-43517)/252375=0.82757$

較原本加快了約 83%

- Github link : https://github.com/ruei7916/SOC_lab-sdram