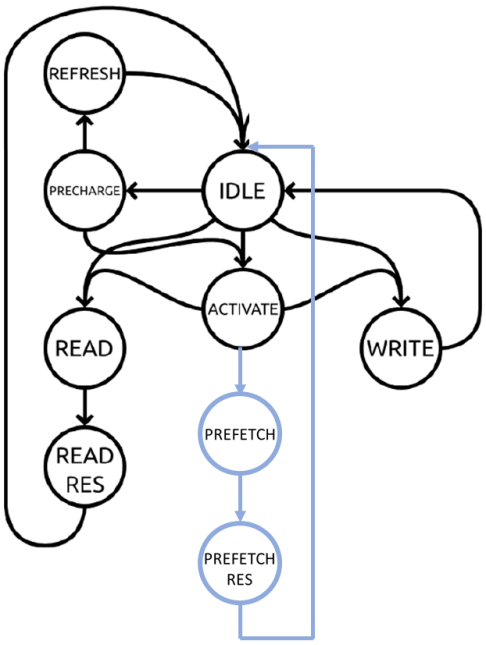
SoC lab-sdram report

312510140 何律明
312410220 張承宗
109651066 林柏佑

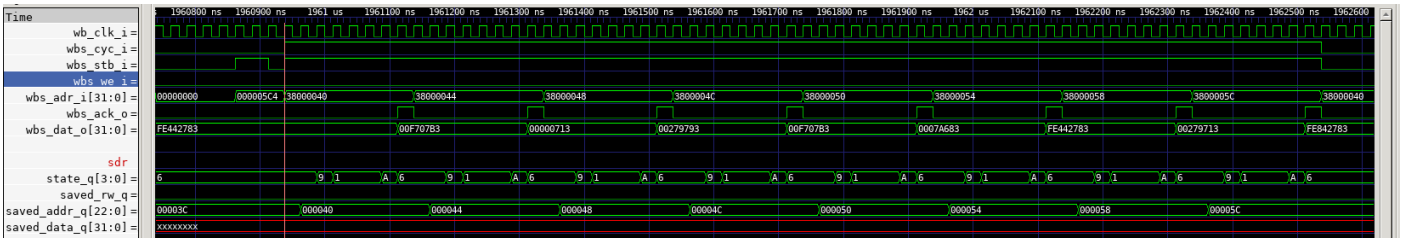
The SDRAM controller design, SDRAM bus protocol：



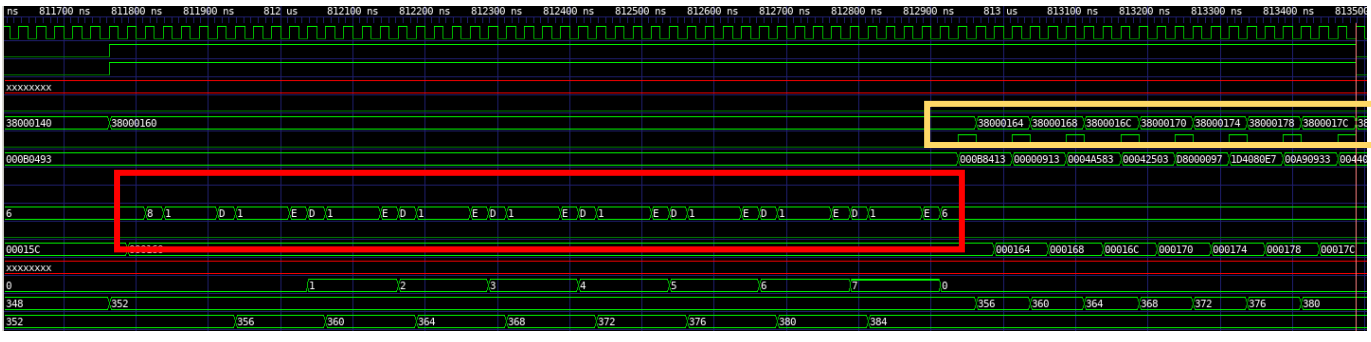
design 和原本的差不多，不過我們額外新增了 2 個 stage，PREFETCH 和 PREFETCH_RES。在 IDLE 或 ACTIVATE 時，若其 Bank 已經是 open 的狀況下，此時即會跳入 PREFETCH，連續抓取 8 次 SDRAM 的 data(連續的 8 個 instruction)，並將其儲存到 cache。而當下次在 IDLE 狀態 cache hit 時，我們就可以直接輸出 cache 裡的值做為 data_out，而不用再去 READ 狀態了。如此一來的 latency 比原本的 exmem 所需要的 10T 來說快上不少。

至於我們的 cacahe 設計的不大，只有 8 個 entry，每一個 entry 擺放 32bit 的 data，這 8 個 entry 的資料是連續的，所以不需要額外 maintain address 放在 cache 當中。

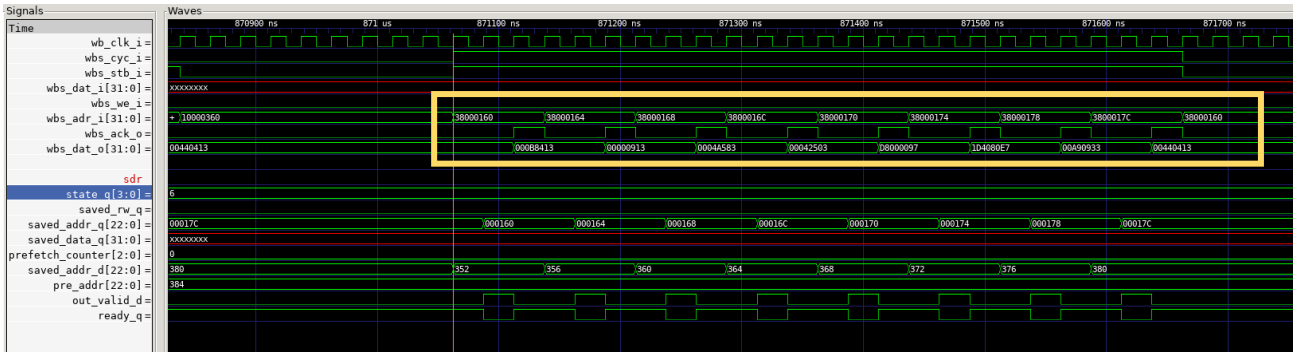
正常沒有做 PREFETCH 的讀寫波形：



做了 PREFETCH（紅色框框處）後的波形：



當 Cache hit（和 PREFETCH 時的 addr 相同）（黃色框框處）時的波形：



Introduce the prefetch schme：

Prefetch Scheme 是一種可以優化系統性能的策略。主要利用處理器的空閒時間，提前加載可能需要的數據或指令，以減少數據訪問延遲並提高整體執行速度。這種預取操作可以通過硬件實現，也可以通過編譯器等軟件工具進行優化。

Introduce the bank interleave for code and data：

```
blkRam#(.SIZE(mem_sizes), .BIT_WIDTH(DQ_BITS))
Bank0(
    .clk(Sys_clk),
    .we(bwen[0]),
    .re(bren[0]),
    .waddr(Col_brst[9:0]),
    .raddr(Col_brst[9:0]),
    .d(bdi[0]),
    .q(bdq[0])
);
blkRam#(.SIZE(mem_sizes), .BIT_WIDTH(DQ_BITS))
Bank1(
    .clk(Sys_clk),
    .we(bwen[1]),
    .re(bren[1]),
    .waddr(Col_brst[9:0]),
    .raddr(Col_brst[9:0]),
    .d(bdi[1]),
    .q(bdq[1])
);
blkRam#(.SIZE(mem_sizes), .BIT_WIDTH(DQ_BITS))
Bank2(
    .clk(Sys_clk),
    .we(bwen[2]),
    .re(bren[2]),
    .waddr(Col_brst[9:0]),
    .raddr(Col_brst[9:0]),
    .d(bdi[2]),
    .q(bdq[2])
);
blkRam#(.SIZE(mem_sizes), .BIT_WIDTH(DQ_BITS))
Bank3(
    .clk(Sys_clk),
    .we(bwen[3]),
    .re(bren[3]),
    .waddr(Col_brst[9:0]),
    .raddr(Col_brst[9:0]),
    .d(bdi[3]),
    .q(bdq[3])
);
```

```
assign      Dqo = Dq_temp;
wire    [DQ_BITS - 1 : 0]  Dq_temp;
assign      Dq_temp = (Data_out_enable_buf) ? bdq[Bank_temp_buf] : 0;
```

可以看出記憶體被分成 4 個 bank，每個 bank 的也都能夠被獨立存取。而 SDRAM 的 output Dqo 會依據不同情況，抓取不同 bank 的 output 值，進而實現了 bank interleave。

在原本的 sdram controller 當中，bank address 是 user_address(wishbone addr)的[9:8]這兩的 bit，所以說要上 firmware code & data 放在不同的 bank 上面，這兩 bit 勢必不能相同，所以說每個 bank 只有 0X100 的大小而已。在下一頁的 linker 對應圖片當中可以看到 dff 的[9:8] = 2'b00, mprjram 的[9:8] = 2'b01，如此一來就可以做到 firmware code & data 放在不同的 bank 當中。

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

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

```
rm -f counter_la.hex

riscv32-unknown-elf-gcc -O1 -Wl,--no-warn-rwx-segments -g \
    --save-temps \
    -Xlinker -Map=output.map \
```

經過嘗試，我們將 mprjram 位址空間改成如上，並利用-O1 來讓 compiler 幫我們優化 assembly code，就能很好的將兩者分開到不同的 bank。我們也發現用-O1 可以很完美的將 firmware matmul()的 code 壓縮在 0x100 的空間下面。

```
257   Disassembly of section .mprjram:
258
259   38000100 <matmul>:
260   38000100: fc010113          addi  sp,sp,-64
261   38000104: 02112e23          sw  ra,60(sp)
262   38000108: 02812c23          sw  s0,56(sp)
263   3800010c: 02912a23          sw  s1,52(sp)
264   38000110: 03212823          sw  s2,48(sp)
265   38000114: 03312623          sw  s3,44(sp)
266   38000118: 03412423          sw  s4,40(sp)
```

D

```
314   380001d8: 01c12b83          lw  s7,28(sp)
315   380001dc: 01812c03          lw  s8,24(sp)
316   380001e0: 01412c83          lw  s9,20(sp)
317   380001e4: 01012d03          lw  s10,16(sp)
318   380001e8: 00c12d83          lw  s11,12(sp)
319   380001ec: 04010113          addi  sp,sp,64
320   380001f0: 00008067          ret
```

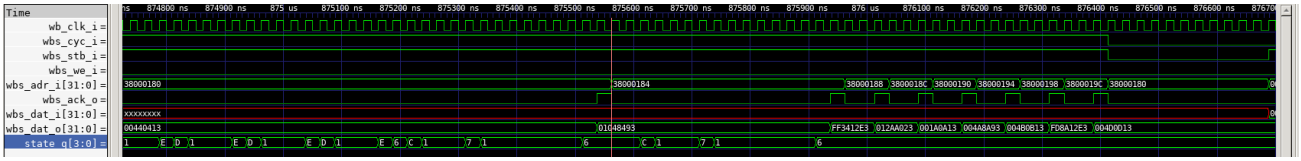
可以看到很完美的 from 0x38000100 ~ 0x380001f0，都是在同一個 bank 當中。

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

將 refresh 的頻率提高以便觀察（原為 750T）：

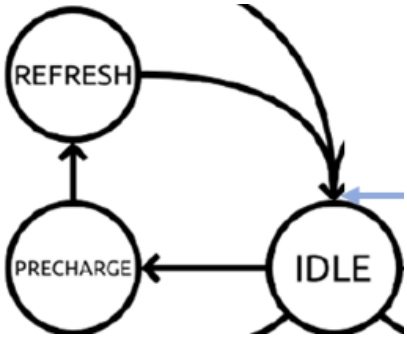
```
33      // Jiin: SDRAM Timing  3-3-3, i.e. CASL=3, PRE=3, ACT=3
34      localparam tCASL            = 13'd2;       // 3T actually
35      localparam tPRE             = 13'd2;       // 3T
36      localparam tACT             = 13'd2;       // 3T
37      localparam tREF             = 13'd6;       // 7T
38      localparam tRef_Counter     = 10'd50;     //
```

可以觀察到以下波型：



FSM 的 state 如下：

```
localparam INIT = 4'd0,
           WAIT = 4'd1,
           PRECHARGE_INIT = 4'd2,
           REFRESH_INIT_1 = 4'd3,
           REFRESH_INIT_2 = 4'd4,
           LOAD_MODE_REG = 4'd5,
           IDLE = 4'd6,
           REFRESH = 4'd7,
           ACTIVATE = 4'd8,
           READ = 4'd9,
           READ_RES = 4'd10,
           WRITE = 4'd11,
           PRECHARGE = 4'd12,
           PREFETCH  = 4'd13,
           PREFETCH_RES  = 4'd14;
```



這個波型的情況就是系統想要 read 位於 3800184 的 data，但是 controller 因為 refresh 的時間到了，state 表現為 6(IDLE) => C(Precharge) + 1(Wait) => 7(Refresh) + 1(Wait)，符合 Refresh 結束後才開始處理系統的 request，因為前面有 pre-fetch 過，後面就是在 IDLE state 進行一連串的 page hit 輸出。

```
localparam tPRE             = 13'd2;       // 3T
localparam tACT             = 13'd2;       // 3T
localparam tREF             = 13'd6;       // 7T
```

這個 access conflict 至少需要讓系統額外等 precharge 3T+1T 和 Refresh 的 1T+7T 共 12T。

Others:

在原本的 user address 當中，bank address 被宣告在[9:8]，如此設計的方式讓一個 bank 只能放 0x100 這麼多資料。

但是藉由 address remapping 的方式將 bank address 放到更高的 bit 去，可以達到讓 firmware code 有更大的擺放空間。

藉由此想法，在我們之後的 final project 當中，應該會去更動 BA 的位置，如此以來才能 fully utilize 每一個 bank bram address 應有的完整擺放空間。