Lab_D

SDRAM Overview

在LAB_D中，SDRAM主要會用到的port可以分為三大部分：控制訊號、資料傳輸與地址傳輸，其中port的IO命名性質以自身模塊為主，如sdram_dqi代表從SDRAM來的資料(Data)，且對控制器來說為資料輸入端口，故以I結尾(Input)。以下為各端口的實際運作原理與分類。

1. 控制訊號

在本次Lab中，sdram_cle、sdram_cs、sdram_dqm並不會用到，原因是本次並沒有去特別控制System clock，故cle會直接寫死，另外cs用於遮擋command的傳輸，本次實驗沒有deselect的設計，但在控制器中可以看到是有預留state卻並無使用，至於dqm是用於遮擋data的傳輸，本次實驗也並無使用。

(63頁提及deselect: https://www.micron.com/-/media/client/global/documents/products/data-sheet/dram/ddr4/4gb_ddr4_dram_2e0d.pdf
(https://www.micron.com/-/media/client/global/documents/products/data-sheet/dram/ddr4/4gb_ddr4_dram_2e0d.pdf))

sdram_cas、sdram_ras則主要控制SDRAM的內部行為，會將其解碼成如讀取、寫入或刷新等不同狀態，rw則控制寫入與否，1為寫入0為讀取。

在SDRAM的對應端口依序為Cke、Cs_n、Ras_n、Cas_n、We_n，以下為實際SDRAM解碼的行為，可以了解上面有提及的控制訊號是如何運型。

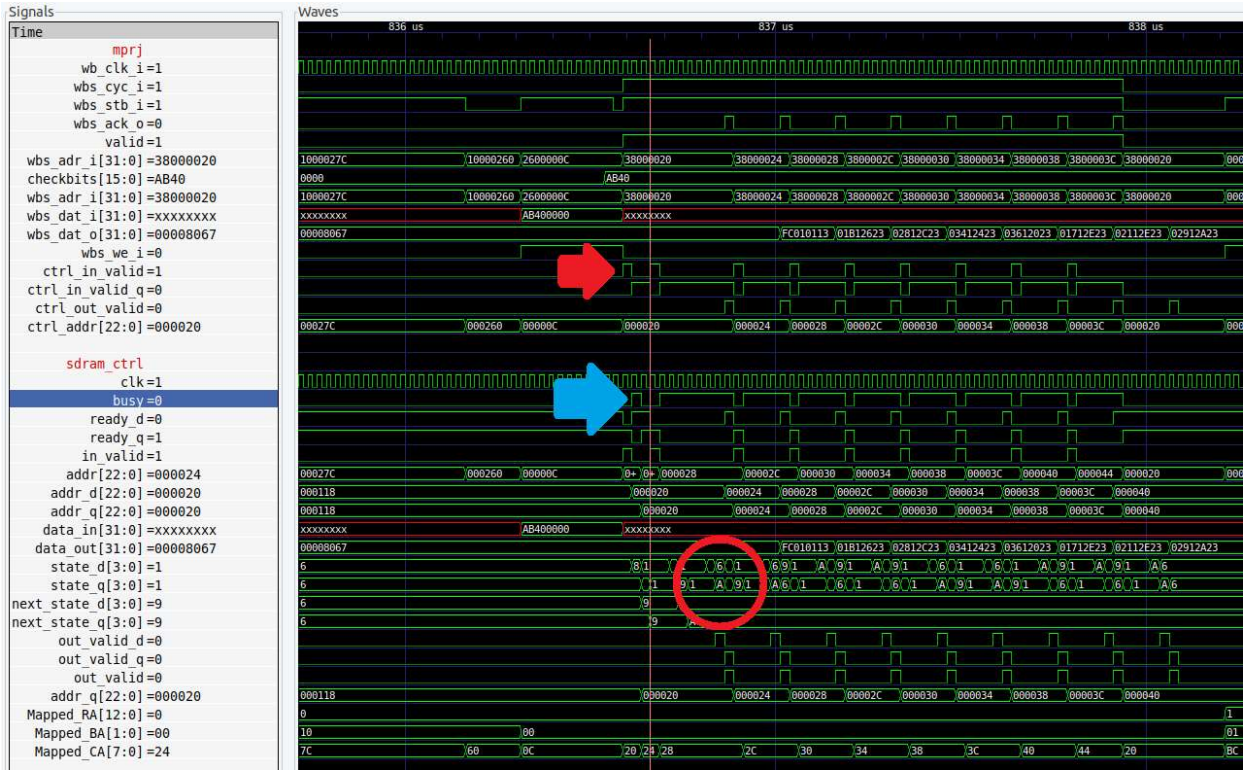
此為控制器的設計

```
localparam CMD_UNSELECTED    = 4'b1000;
localparam CMD_NOP           = 4'b0111;
localparam CMD_ACTIVE        = 4'b0011;
localparam CMD_READ          = 4'b0101;
localparam CMD_WRITE         = 4'b0100;
localparam CMD_TERMINATE     = 4'b0110;
localparam CMD_PRECHARGE     = 4'b0010;
localparam CMD_REFRESH       = 4'b0001;
localparam CMD_LOAD_MODE_REG = 4'b0000;

assign sdram_cs = cmd_q[3];
assign sdram_ras = cmd_q[2];
assign sdram_cas = cmd_q[1];
assign sdram_we = cmd_q[0];
```

此為SDRAM內部設計

```
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;
```

最後還有一busy訊號較為特殊，其對Wishbone和SDRAM並無端口傳輸，而是單獨用在Wishbone和控制器的valid訊號轉換，也是後面會提到Prefetch主要相關的運作端口，由於SDRAM讀取資料需要時間，因此控制器內部可以暫存一筆地址用於下一次輸出。在workbook內講到的，由於Wishbone只有ack可以判斷資料是否送出，原設計要等待Wishbone傳送下一筆地址進來，如此的話控制器將會停留在IDLE STATE三個cycle導致時間的浪費，而當有Prefetch地址時，控制器會感應到有保存的地址(紅色箭頭在wb沒動作時有兩個in_valid)，就會將Busy訊號拉起並只在IDLE STATE停留一個週期(紅圈處)直接去讀取保存住的地址以減少讀取時間，且當Busy拉起時(藍色箭頭)Wishbone和控制器之間的轉換就會將valid卡住，避免多輸入一筆地址。



2. 資料傳輸

資料傳輸的端口又可以分為兩類，一是對FPGA上Wishbone資料輸入輸出，分別為data_in和data_out，二是對SDRAM本身的資料輸入輸出，分別是sdram_dqi和sdram_dqo，分別對應到wbs_dat_i、wbs_dat_o、Dqi和Dqo。顯而易見其四個端口都只是直接將port直接接上，並無特殊運作方式。

3. 地址傳輸

基本同資料傳輸的方式一樣分為兩類，對Wishbone為user_addr，對應到ctrl_addr，是已經wbs_adr_i做decode後的地址，另外對SDRAM的地址又分為兩個，sdram_ba和sdram_a，前者為區分bank的位址後者則是row和column的地址，對應到Ba和Addr。

Prefetch schme

在看了mm的波型檔後可以發現其實地址在短時間內基本上不會重複，因此其實在這個lab中buffer的用處不大除非數量夠多，但也會造成面積浪費，因此以下將進行不討論buffer的功用。

Prefetch主要是在busy訊號為0時，也就是控制器內部沒有放暫存的地址時就輸入下一筆地址，但實際Wishbone因為還未收到上一筆地址的資料故ack不會拉起，也因此不會有下一筆地址，所以在這個lab中我們將猜測下一筆地址為現在地址的+4(也就是當下記憶體的下一筆資料、紅圈處)，如此猜測的原因是波型檔大部分都是連續讀取，只有少數幾次是跳去其他不規則的地址，因此此猜測的成效相當的好。

其電路設計很簡單，主要內容為：

1. 在ctrl_in_valid時代表可以更新下一筆資料。
2. 猜測下一筆是現在地址的+4，也必須先把現在地址輸入後再輸入下一筆。
3. 紀錄上次輸入的地址為何。

第一點很好理解，就是要更新下一筆地址。

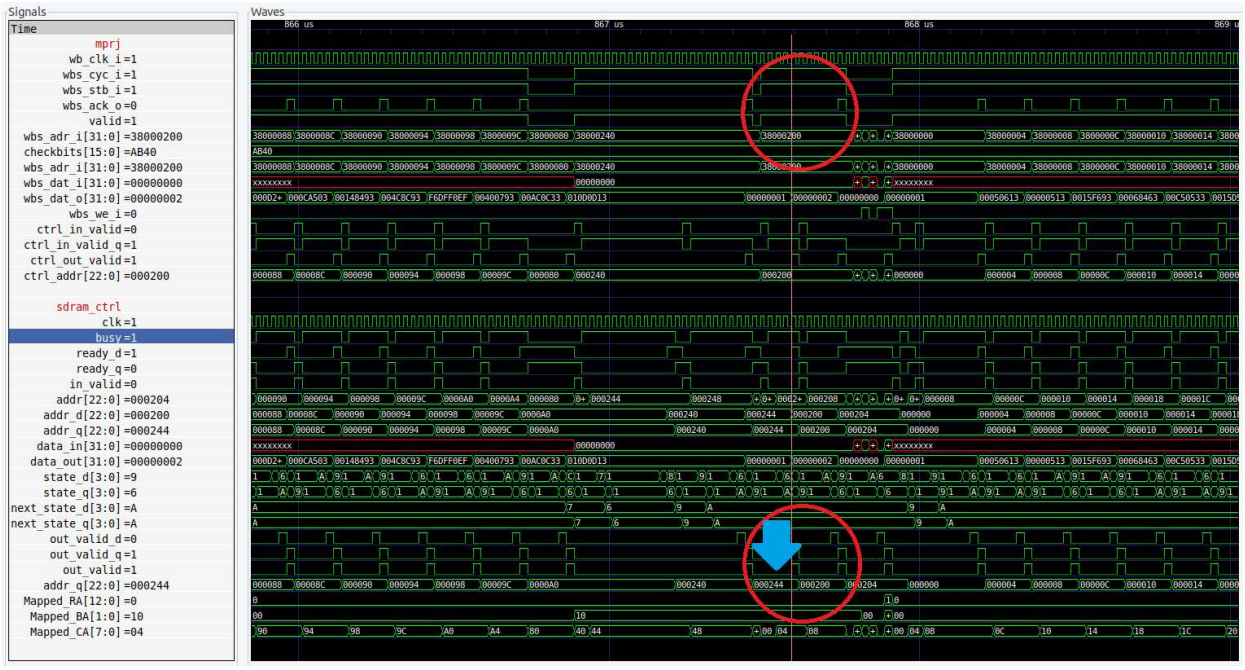
```
always@(posedge clk)begin
    if(rst)
        next_addr <= 0;
    else
        next_addr <= (ctrl_in_valid && !wbs_we_i) ? user_addr + 4 : next_addr;
end
```

第二點是避免連續讀取的時候輸入錯誤的地址，而當與記憶體無關時直接寫為0，user_addr為實際輸入控制器的地址，取代原先的ctrl_addr。

```
assign user_addr = (next_in && valid && !wbs_we_i ) ? next_addr : ctrl_addr;

always@(posedge clk)begin
    if(rst)
        next_in <= 0;
    else begin
        if(valid && !wbs_we_i)
            next_in <= (ctrl_in_valid_q) ? 1 : next_in;
        else
            next_in <= 0;
    end
end
```

第三點紀錄上一次輸入資料是為了避免在兩次access記憶體其實不是連續的但時序上又很相近，使上一筆預測的資料還沒送回來時，下一個地址卻已經發送過來，又剛好out_valid拉起導致ack送出讓地址與資料不符。為避免此情況，我們在ack時多設置了一個限制，也就是上一筆送進去的資料要與現在wishbone在等待的位置相差4，否則將判定為錯誤資料ack就不予拉起(紅圈處與藍色箭頭)。

```
assign wbs_ack_o = (wbs_we_i) ? ~ctrl_busy && valid : ctrl_out_valid && valid && (diff==4);

assign diff = last_in_addr-ctrl_addr;

always@(posedge clk)begin
    if(rst)
        last_in_addr <= 0;
    else begin
        last_in_addr <= (valid && ctrl_in_valid) ? user_addr : last_in_addr;
    end
end
```



另外為了能在busy為0時輸入下一筆addr則必須將原ctrl_in_valid_q修改，使其在沒有busy時也能歸零，否則ctrl_in_valid就只會等到有out_valid的下一個cycle才拉起。

```
assign ctrl_in_valid = wbs_we_i ? valid : ~ctrl_in_valid_q && valid && !ctrl_busy;

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_busy) //只有修改此處
            ctrl_in_valid_q <= 1'b0;
    end
end
```

Bank Interleave

在韌體的資料夾中，有section.ids可以修改code和data的地址，在控制器中可以看到地址的解碼方式，前8個bit為column address，之後接續的是bank address有兩個bit，最後才是row address有13個bit，因此若要分離bank的話只要修改addr[9:8]，所以在本次lab中將code放在0x3800_0000到

0x38000200，也就是放在第1和第2個bank，則放在0x3800_0200到0x38000400，也就是第3和第4個bank，以此達成Bank Interleave的效果。

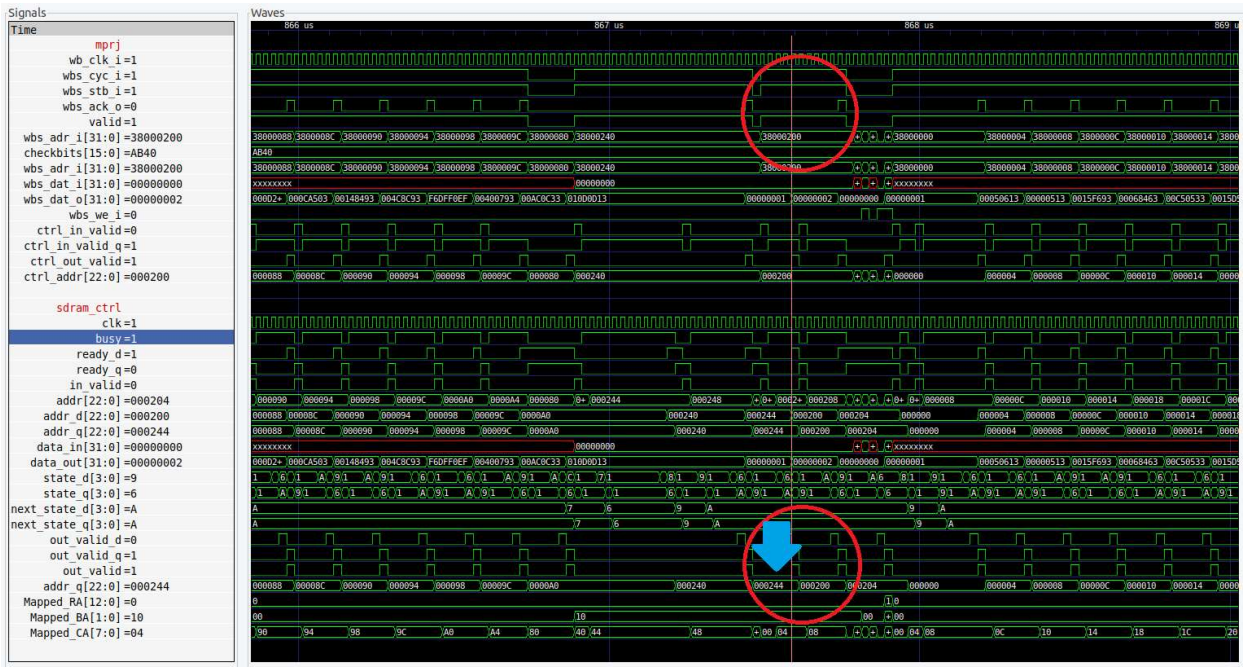
以下圖為例(同上一張圖)，可以發現在紅圈處以及紅圈前一個Wishbone索取資料時地址分別為200和240，再更之前就是在200以下，就是實際Bank Interleave的效果，比較特殊的是倘若沒有Bank Interleave，其執行的時間甚至會快於有Bank InterLeave，在此將估且推斷為是因為沒有分開會需要重新將bank和row做open因此會消耗更多時間，其相關討論將放在下一節。

```
//sdram_controller.v

assign Mapped_RA = user_addr[22:10];
assign Mapped_BA = user_addr[9:8];
assign Mapped_CA = user_addr[7:0];
assign addr = {Mapped_RA, Mapped_BA, Mapped_CA};



//section.ids

//有Bank Interleave
mprjram : ORIGIN = 0x38000000, LENGTH = 0x00000200
all_data : ORIGIN = 0x38000200, LENGTH = 0x00000400
```

```
ubuntu@ubuntu2004:~/sdram_final/linker_pre/testbench/counter_la$ source run_sim
Reading counter_la.hex
counter_la.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la.vcd opened for output.
          830513000, LA matmul started
         1475238000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x003e
         1476313000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x0044
         1503913000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x004a
         1504988000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x0050
         1505263000, LA matmul passed
```

```
//section.ids
//無Bank Interleave
mprjram : ORIGIN = 0x38000000, LENGTH = 0x00000160
all_data : ORIGIN = 0x38000160, LENGTH = 0x00000400
```

```
ubuntu@ubuntu2004:~/sdram_final/linker_pre_nointerleave/testbench/counter_la$ source run_sim
Reading counter_la.hex
counter_la.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la.vcd opened for output.
          830513000, LA matmul started
         1474888000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x003e
         1475963000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x0044
         1503563000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x004a
         1504638000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x0050
         1504913000, LA matmul passed
```

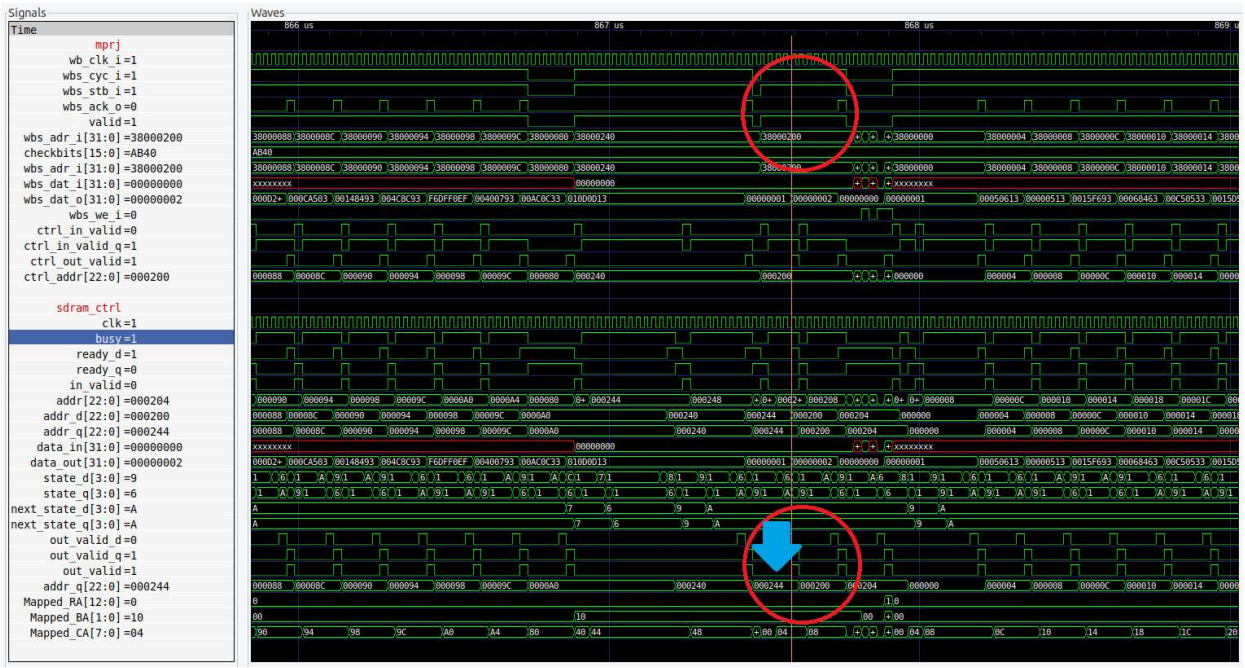


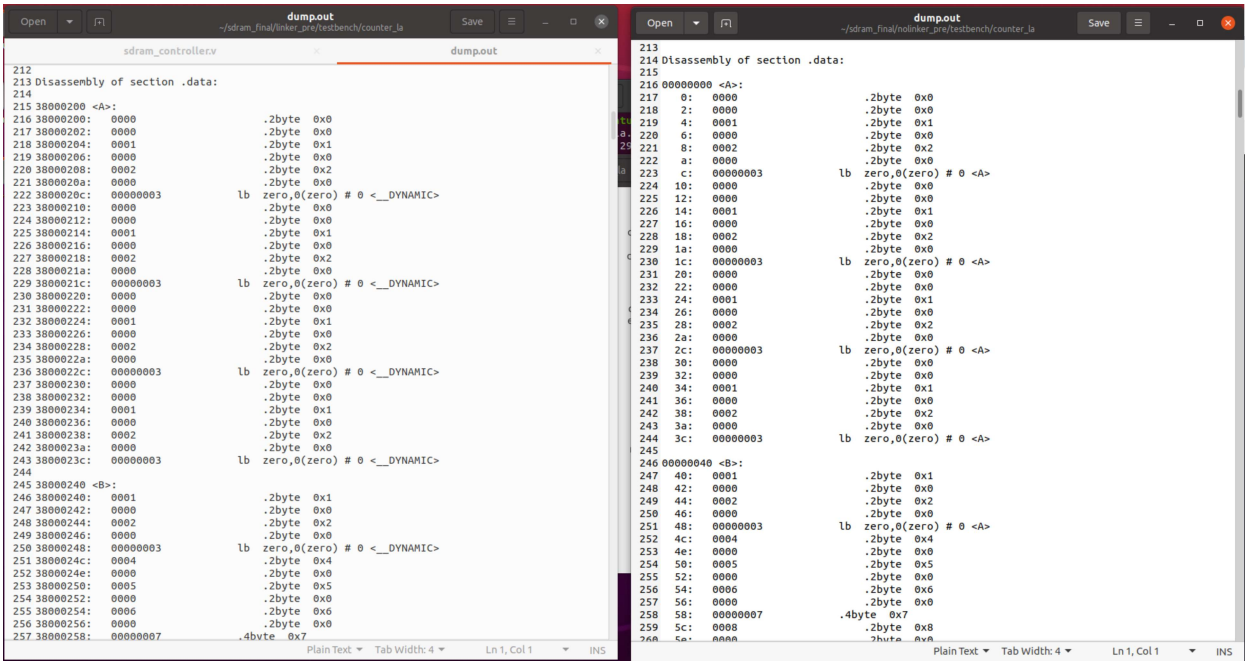
Modify the linker

若是沒有調整linker的話code仍會在sdram中，但data會被存在其他地方，從下圖dump.out中(左為有使用，右為無調整)，可以發現data存的位置並不相同且甚至不在sdram，另外延續上個主題討論沒有將Bank Interleave會較快的原因，是否為在於不用重新open新的bank和row可以透過對調data和code的地址就可以得知，也就是讓data改為0 ~ 160、code改為160 ~ 400，結果也不意外地確實更慢而且比有Bank Interleave還慢，原因顯而易見，從下波型(對又是同一張)可以看到其實大部分時候matmul都是先去access code的部分(0~200)，緊接著就會去access data(200~400)，因此讓code放在data之前，並且無Interleave的話確實應該要加快整體速度，因此可以得知linker的位置應該要跟compile出來的執行順序對應才能讓電路加速，否則會導致反效果。

```
//section.ids
//無Bank Interleave且對調位置
mprjram : ORIGIN = 0x38000160, LENGTH = 0x00000400
all_data : ORIGIN = 0x38000000, LENGTH = 0x00000160
```

```
ubuntu@ubuntu2004:~/sdram_final/linker_pre_nointerleave/testbench/counter_la$ source run_sim
Reading counter_la.hex
counter_la.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la.vcd opened for output.
          830513000, LA matmul started
         1475313000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x003e
         1476388000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x0044
         1503988000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x004a
         1505063000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x0050
         1505338000, LA matmul passed
```

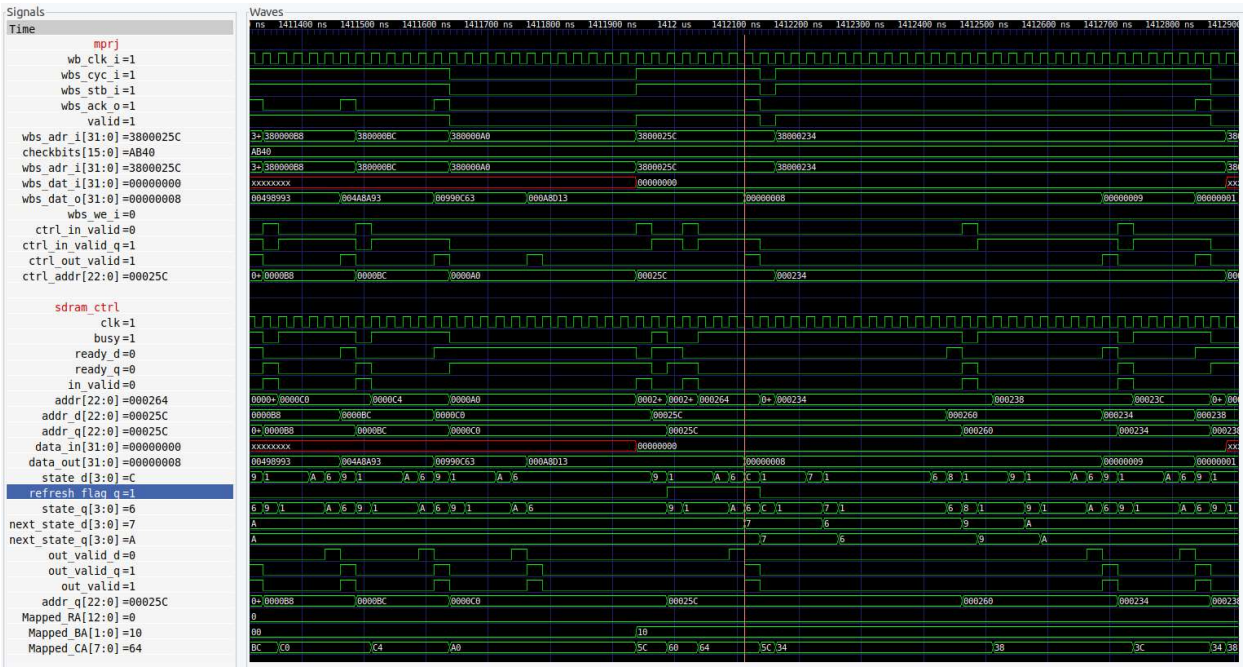




Reduced the period

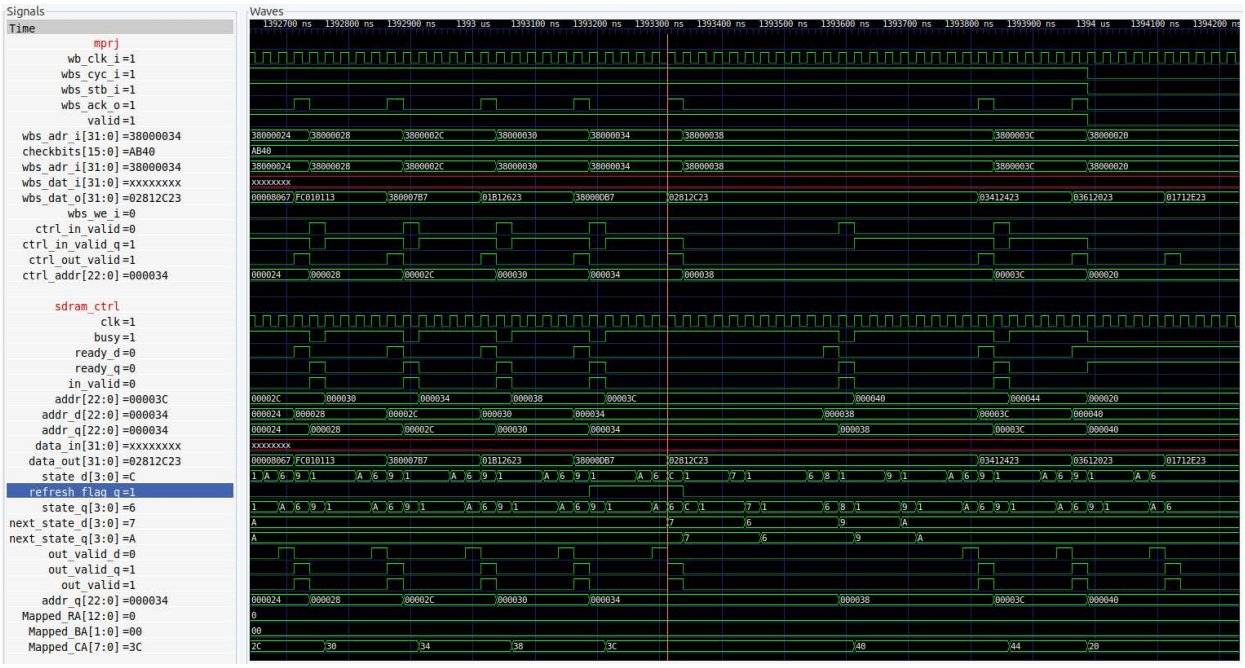
根據課程中提供的控制器的檔案中tREF為6個cycle，以此去做模擬可以發現當sdram要做refresh時若遇Wishbone正在索取data，Wishbone仍只會等待wbs_ack_o輸出，因此若refresh時間可以縮短，或是避開Wishbone要資料時做refresh就可以加快速度。

```
ubuntu@ubuntu2004:~/sdram_final/linker_pre/testbench/counter_la$ source run_sim
Reading counter_la.hex
counter_la.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la.vcd opened for output.
          830513000, LA matmul started
         1475238000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x003e
         1476313000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x0044
         1503913000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x004a
         1504988000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x0050
         1505263000, LA matmul passed
```



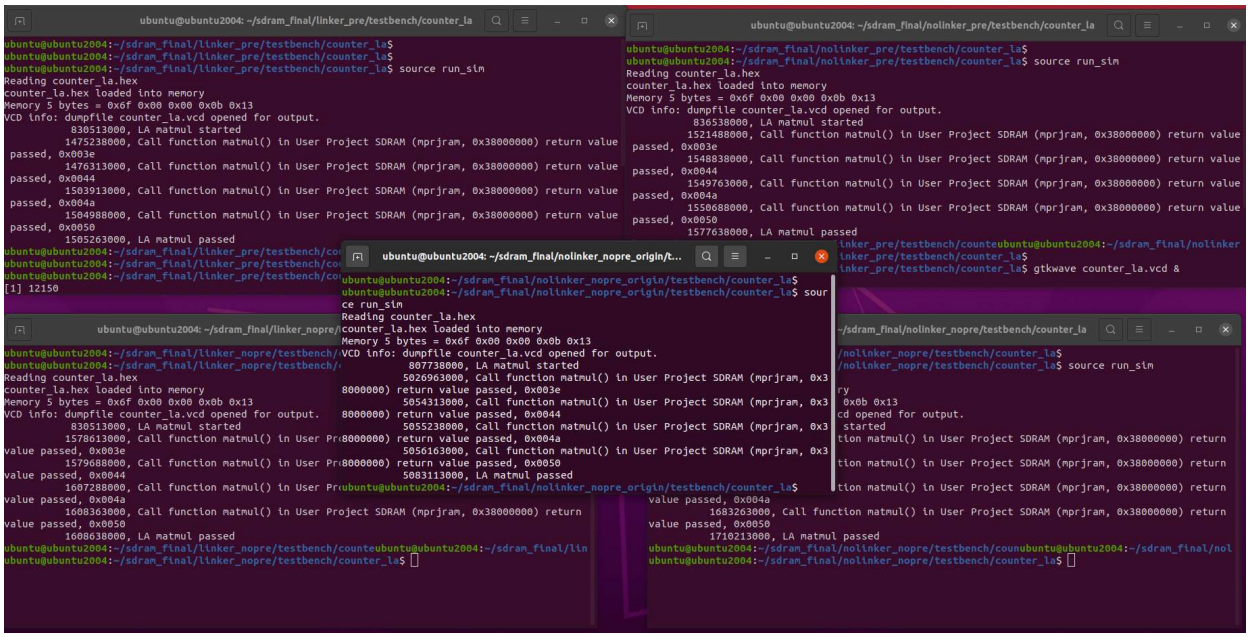
若將tREF改為3個cycle去做模擬，電路輸出時間上確實會縮短但卻沒有特別突出的效果，原因是refresh並非每次都在Wishbone索取時發生，並且發生次數也並不太頻繁，因此若要透過縮減tREF來加速電路的話，應去利用其他方式讓其在Wishbone沒索取data時在refresh會有較佳的效果，此外從波型上也可以發現refresh後prefetch的地址將會無效，等於prefetch並無發生要額外浪費cycle再次讀取，當然也可以透過buffer的方式將資料暫存，但從根本上解決應該要是盡量避免refresh在wishbone索取data時發生。

```
ubuntu@ubuntu2004:~/sdram_final/linker_pre/testbench/counter_la$ source run_sim
Reading counter_la.hex
counter_la.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la.vcd opened for output.
          830513000, LA matmul started
         1472038000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x003e
         1473113000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x0044
         1500713000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x004a
         1501788000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x0050
         1502063000, LA matmul passed
```



Other

有無linker以及有無prefetch的速度比較，左半為有做linker右半則無，上半為有prefetch下半則無，中間則為，甚麼都沒做也沒有對compile做優化(無-Os、gcclib)


ubuntu@ubuntu2004: ~/sdram_final/linker_pre/testbench/counter_la
ubuntu@ubuntu2004:~/sdram_final/linker_pre/testbench/counter_la$
ubuntu@ubuntu2004:~/sdram_final/linker_pre/testbench/counter_la$
ubuntu@ubuntu2004:~/sdram_final/linker_pre/testbench/counter_la$ source run_sim
Reading counter_la.hex
counter_la.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la.vcd opened for output.
830513000, LA matmul started
1475238000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x003e
1476313000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x0044
1503913000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x004a
1504988000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x0050
1505263000, LA matmul passed
ubuntu@ubuntu2004:~/sdram_final/linker_pre/testbench/co
ubuntu@ubuntu2004:~/sdram_final/linker_pre/testbench/co
ubuntu@ubuntu2004:~/sdram_final/linker_pre/testbench/co
[1] 12150
ubuntu@ubuntu2004: ~/sdram_final/nolinker_pre/testbench/counter_la
ubuntu@ubuntu2004:~/sdram_final/nolinker_pre/testbench/counter_la$
ubuntu@ubuntu2004:~/sdram_final/nolinker_pre/testbench/counter_la$ source run_sim
Reading counter_la.hex
counter_la.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la.vcd opened for output.
836538000, LA matmul started
1521488000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x003e
1548838000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x0044
1549763000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x004a
1550688000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x0050
1577638000, LA matmul passed
ubuntu@ubuntu2004:~/sdram_final/nolinker_pre/testbench/counte
ubuntu@ubuntu2004:~/sdram_final/nolinker_pre/testbench/counter_la$
ubuntu@ubuntu2004:~/sdram_final/nolinker_pre/testbench/counter_la$ gtkwave counter_la.vcd &
ubuntu@ubuntu2004: ~/sdram_final/nolinker_nopre_origin/t...
ubuntu@ubuntu2004:~/sdram_final/nolinker_nopre_origin/testbench/counter_la$
ubuntu@ubuntu2004:~/sdram_final/nolinker_nopre_origin/testbench/counter_la$ sour
ce run_sim
Reading counter_la.hex
counter_la.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la.vcd opened for output.
807738000, LA matmul started
5026963000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x003e
5054313000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x0044
5055238000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x004a
5056163000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x0050
5083113000, LA matmul passed
ubuntu@ubuntu2004:~/sdram_final/nolinker_nopre_origin/testbench/counter_la$
ubuntu@ubuntu2004: ~/sdram_final/linker_nopre/
ubuntu@ubuntu2004:~/sdram_final/linker_nopre/testbench/
ubuntu@ubuntu2004:~/sdram_final/linker_nopre/testbench/counter_la$
Reading counter_la.hex
counter_la.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la.vcd opened for output.
830513000, LA matmul started
1578613000, Call function matmul() in User Pr
value passed, 0x003e
1579688000, Call function matmul() in User Pr
value passed, 0x0044
1607288000, Call function matmul() in User Pr
value passed, 0x004a
1608363000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x0050
1608638000, LA matmul passed
ubuntu@ubuntu2004:~/sdram_final/linker_nopre/testbench/counte
ubuntu@ubuntu2004:~/sdram_final/linker_nopre/testbench/counter_la$
~/sdram_final/nolinker_nopre/testbench/counter_la
/nolinker_nopre/testbench/counter_la$
/nolinker_nopre/testbench/counter_la$ source run_sim
ry
0x0b 0x13
cd opened for output.
started
tion matmul() in User Project SDRAM (mprjram, 0x38000000) return
tion matmul() in User Project SDRAM (mprjram, 0x38000000) return
tion matmul() in User Project SDRAM (mprjram, 0x38000000) return
value passed, 0x004a
1683263000, Call function matmul() in User Project SDRAM (mprjram, 0x38000000) return value passed, 0x0050
1710213000, LA matmul passed
ubuntu@ubuntu2004:~/sdram_final/nolinker_nopre/testbench/counubuntu@ubuntu2004:~/sdram_final/nol
ubuntu@ubuntu2004:~/sdram_final/nolinker_nopre/testbench/counter_la$