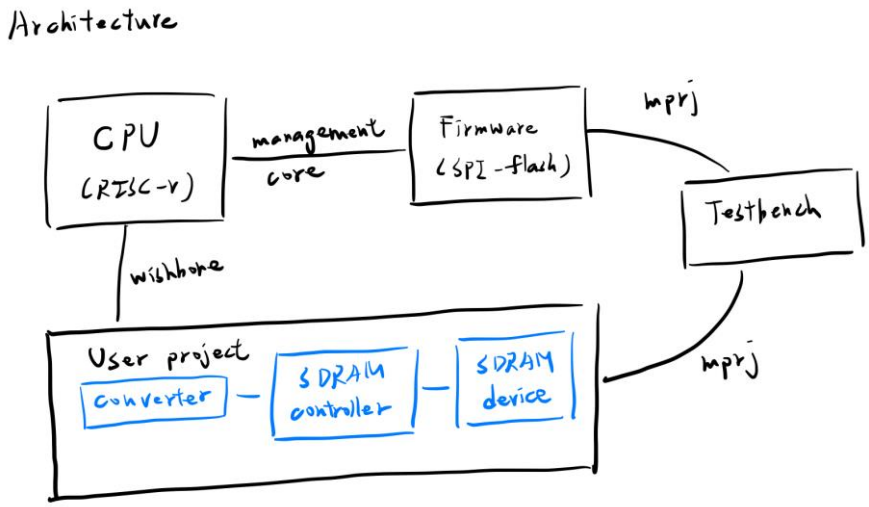
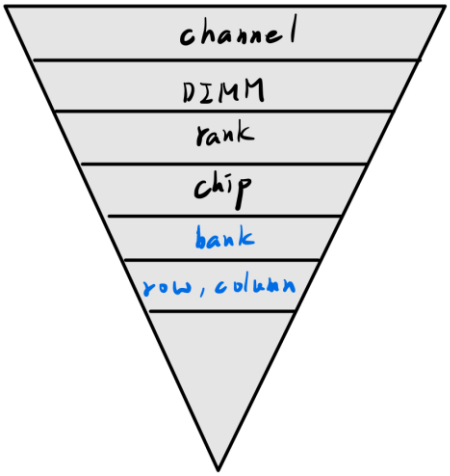
Report

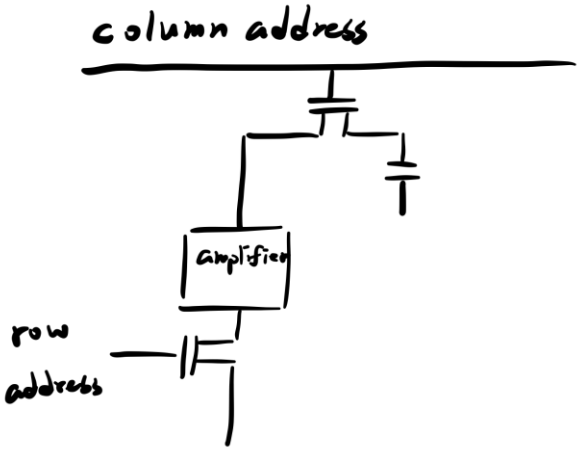
311512042 電控碩 黃俊誠



這次的 lab 主要是把之前的 system 中 bram 改成用 sdram 來存，這邊先畫一個簡單的 hierarchy



藍色就是這次會主要著手的部分，bank 裡其實還有一些 row buffer 來存 read write 的 data, 等 address valid 後做 handshake，並且判斷是 0 (read) 或 1 (write)
而 Sdram controller 是用來對同一個 cs 的 sdram 進行讀寫操作
首先



要先 active，然後做 pre charge，等 task 結束後，有些設計會判斷接下來是不是用到同一個 ram，來

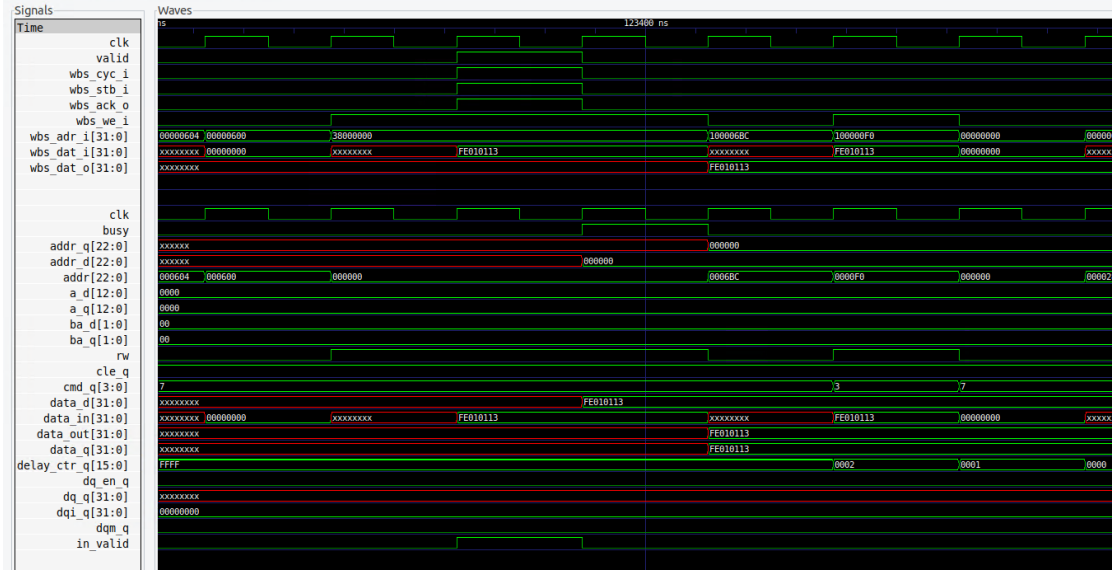
決定要直接 refresh 掉還是保留著繼續做存取，一般 ram 的存取都是一列一列的順序，每個 row 都有 row buffer，常常是讀寫交錯所以會使用兩個
這一個個的 buffer 形成 bank，看要選哪個 bank，或是說選哪個 group 做存取和讀寫，同一個行就不用馬上就 refresh，這樣可以節省重新 precharge 的 time 跟 power，但相對的就是要看設計上是不是常常 bank 裡面跳來跳去，那就不適合了，不過本次 lab 是選用固定 precharge 的方式，以練習相關的 coding 能力

Prefetch

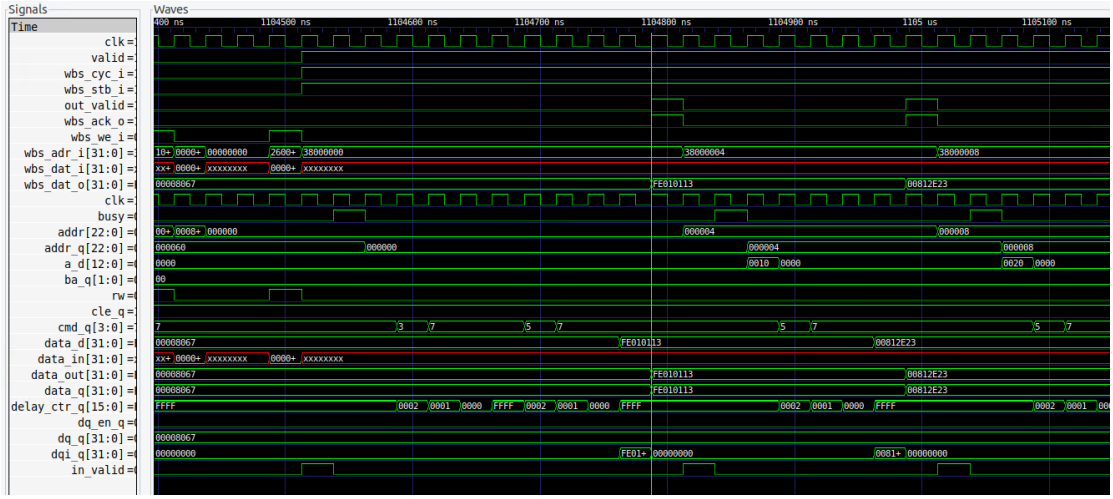
Prefetch 主要可以提升 performance，也就是預取一部分 bit 的資料，輸出就能超過原本的量，如果 burst 到最大的 data 量，那需要的頻率就能縮小 1/burst 的量，就能達到 high speed 的預期，提高讀寫效率
簡單講就是先把資料放到 buffer，request 一來就能直接送過去

Code

```
jimmy@jimmy-VirtualBox:~/Desktop/caravel-soc_fpga-lab/lab-sdram/testbench/counte
r_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.
LA Test 1 started
Call function adder() in User Project BRAM (mprjram, 0x38000000) return value pa
ssed, 0x2233
LA Test 2 passed
```



跑完模擬可以發現從 write enable 拉起到寫入花了三個 T，sdram 則是同時寫入 data 的



Read mode 時，address 送到後，等了 7 個 T，data 才讀出來

Firmware matmul

接下來要試著用 matmul 的 firmware 控制 user project，首先要改一下 include 的 rtl list

```
## User project
-v ../../rtl/user/user_project_wrapper.v
-v ../../rtl/user/user_proj_example.counter.v
-v ../../rtl/user/sdram_controller.v
-v ../../rtl/user/sdr.v
```

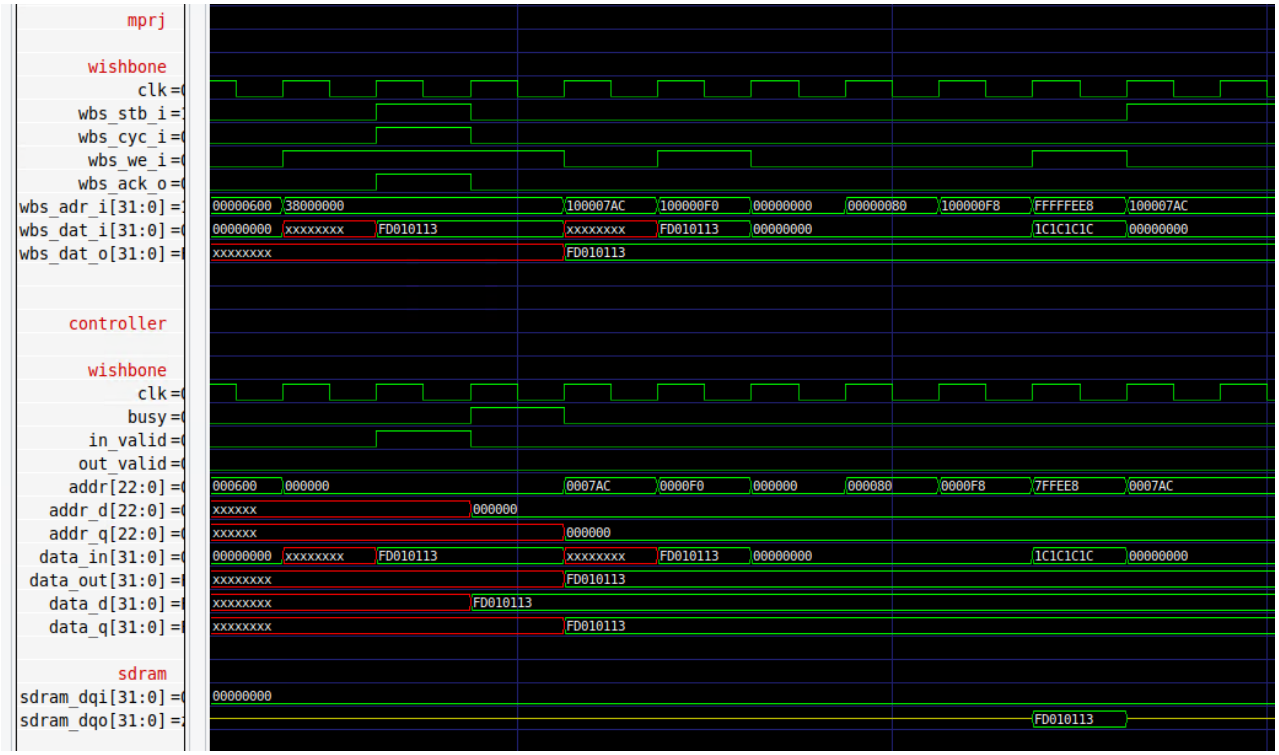
```
## STD CELLS - they need to be below the defines.v files
#-v ../../cvc-pdk/sky130_sram_2kbyte_1rw1r_32x512_8.v
```

接著就是將 multiple 的結果寫進 sdram

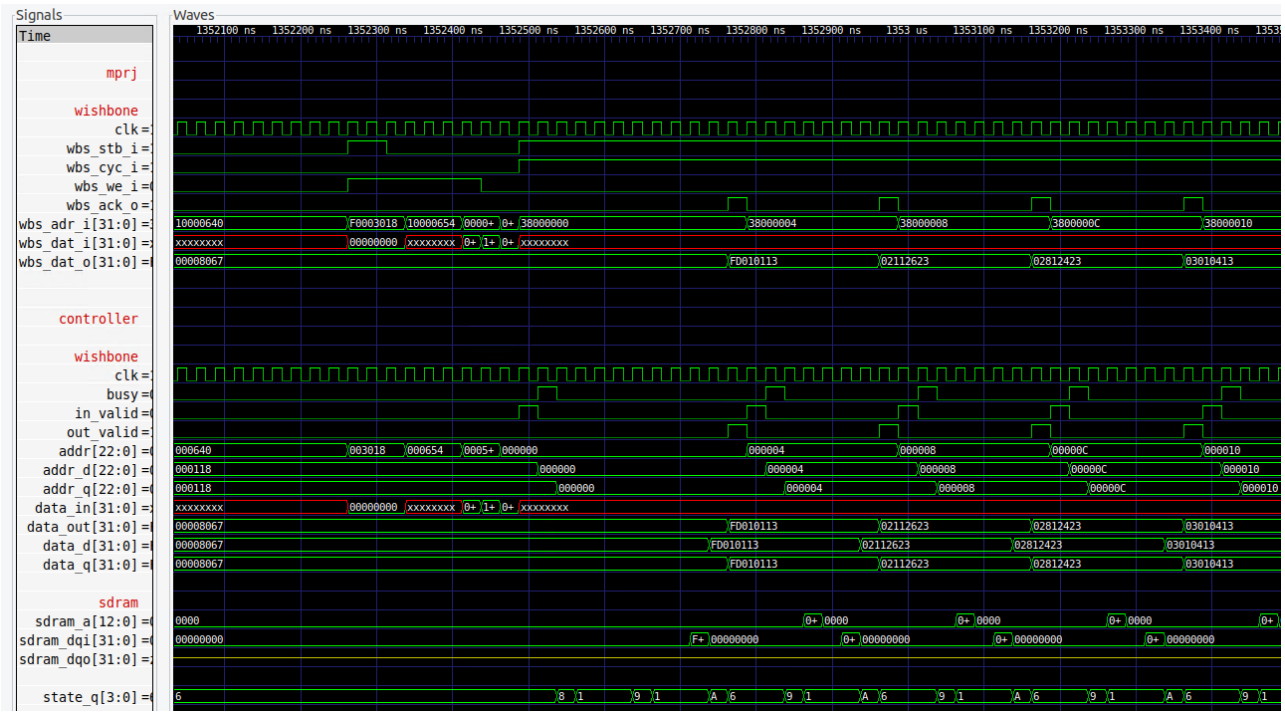
```
jimmy@jimmy-VirtualBox:~/Desktop/caravel-soc_fpga-lab/lab-sdram/testbench/counte
r_la_mm$ source run_sim
Reading counter_la_mm.hex
counter_la_mm.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la_mm.vcd opened for output.
LA Test 1 started
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value p
assed, 0x003e
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value p
assed, 0x0044
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value p
assed, 0x004a
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value p
assed, 0x0050
LA Test 2 passed
```

我這邊改寫了一點，後面也有多送一個 AB5100 來結束 task，可以看到四個值都有正確讀取

Write waveform



Read waveform



發現每次 busy 打上來後 state 的變化模式為 ：

ACTIVATE 1T

READ (9) 1T

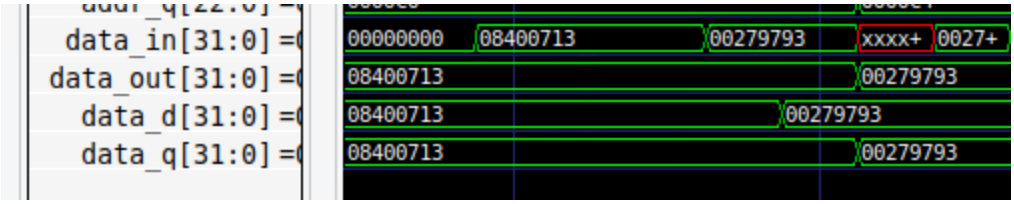
WAIT (1) 3T

READ RESET(A) 1T

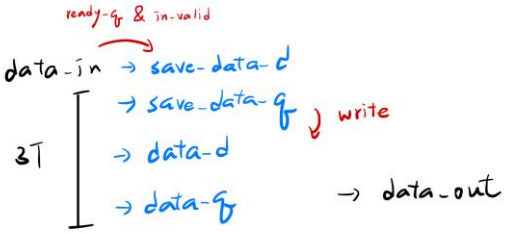
IDLE(6) 3T

也就是讀一筆資料總共花了 9T

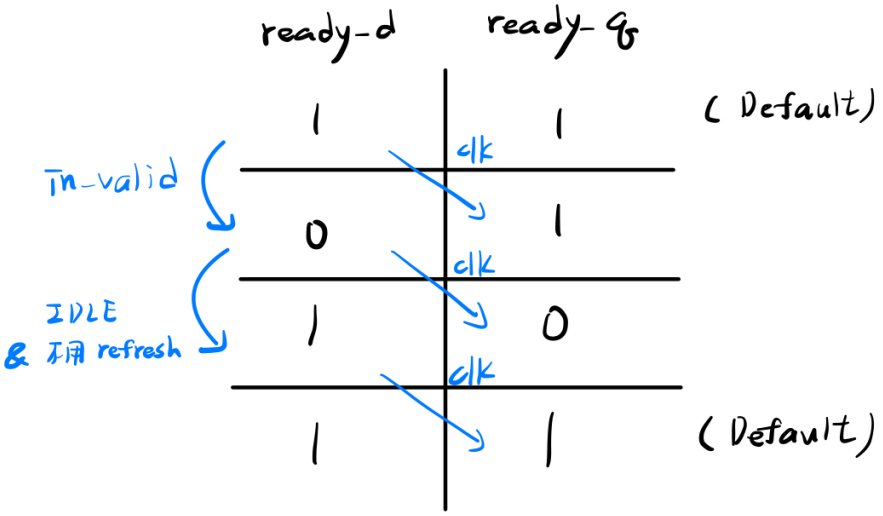
其中 wait 3T 是 code 裡自己寫上的 tCASL=’d2，實際 delay 了 3T



這之間 data_in 到 data_out 經過了 save_data_q、data_d 和 data_q

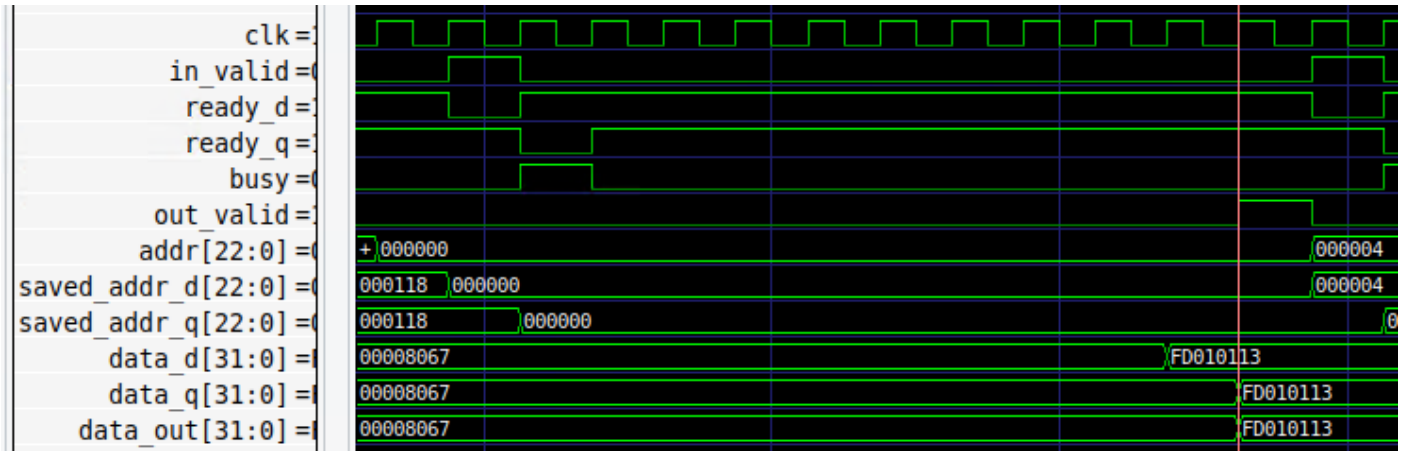


Queue 的 state 變化則是如下所示：



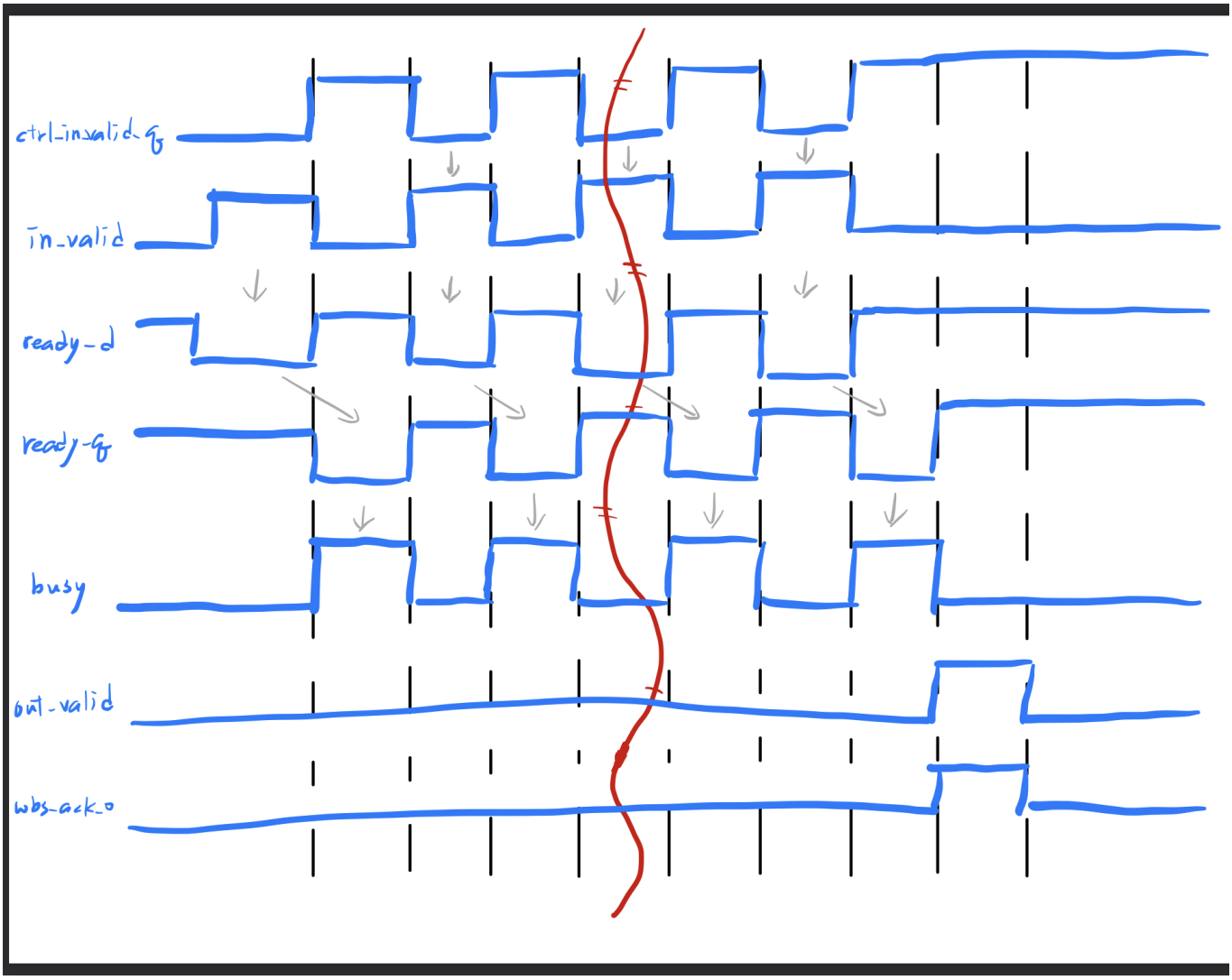
- ready_d @ *
- ready_q @ clk

Optimization

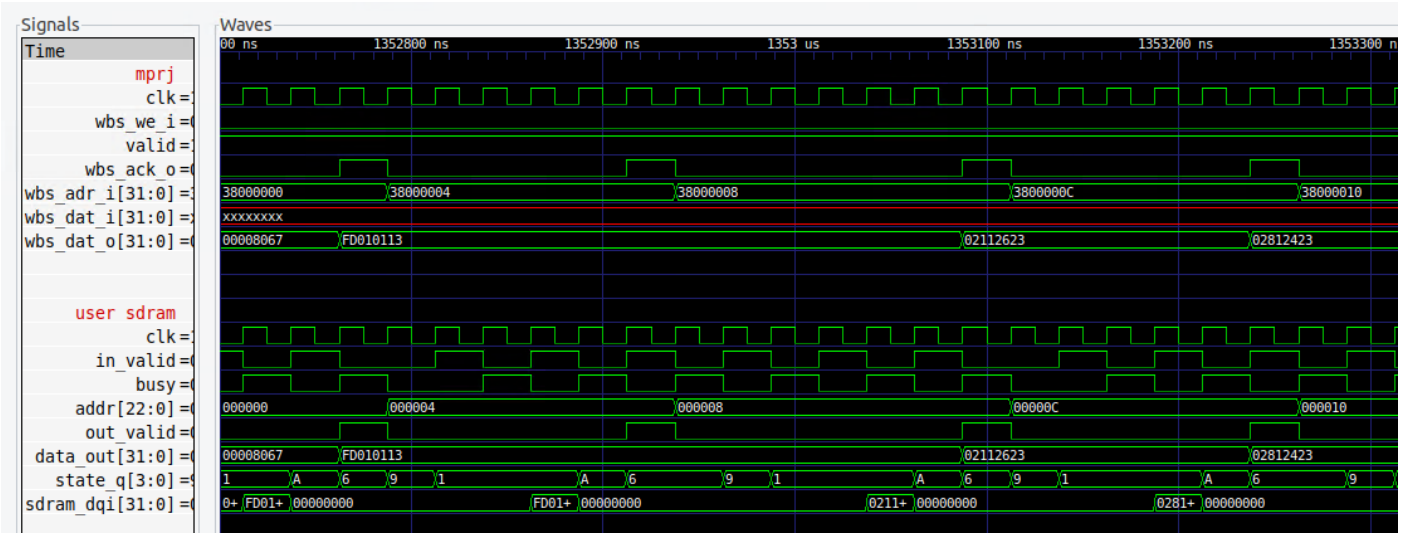


從這張 waveform 可以發現 request 之後 data 很晚才收到，所以 out_valid 一直在 wait，下一個 request 也跟著 wait

改善目標是讓指令解碼跟執行可以與指令讀取獨立，下面的是理想的結果



希望能連續 8 個 T 收 request，存進 buffer 裡(size=8)，再一起 output 出來



不過實際得到的結果 request 目前只能優化到連送 6 個，output 則是提升到每 5 個 T 一次
提升的程度計算如下:

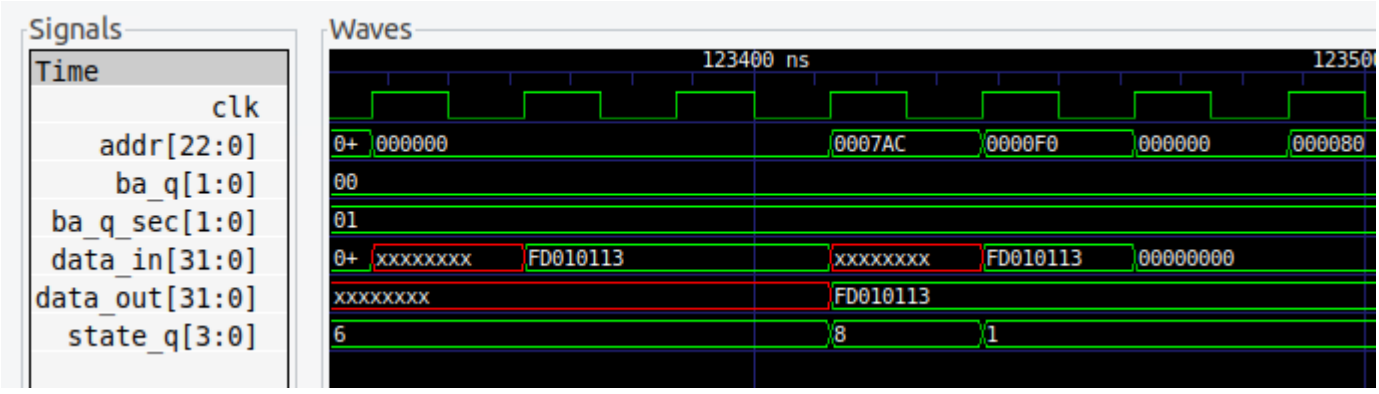
$$9\mathrm{T} + 1\mathrm{T} + (7\mathrm{T} + 1\mathrm{T}) * 7 = 66\mathrm{T}$$

$$9\mathrm{T} + 1\mathrm{T} + (5\mathrm{T} + 1\mathrm{T}) * 7 = 52\mathrm{T}$$

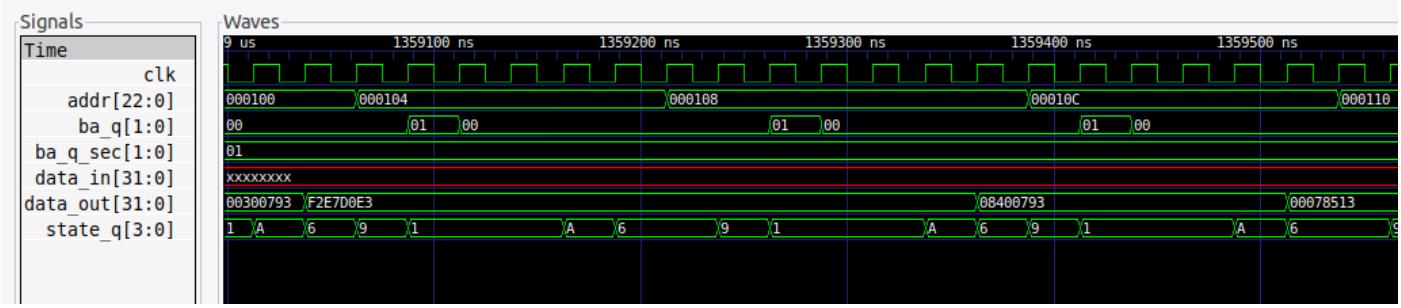
$$\frac{\frac{1}{52\mathrm{T}}}{\frac{1}{66\mathrm{T}}} = \frac{66T}{52T} = 1.269$$

大約提升了 23%的 throughput !!
接下的部分，我想看這幾天能不能繼續試出更好的結果，final project 再呈現出來。

Bank interleave
這邊我的想法是將奇數 address 和偶數 address 分開
Pipeline 起來的感覺就是 13579…和 246810…的 bit 可以分開存取
但實際操作起來有點困難，我試著把兩個 data 分別存到 bank00 和 bank01
Read waveform



Write waveform



可以看到這邊 bank 有重疊了，同時存在 01，不過我應該沒有正確的讀寫到不同 bank 的 data 這邊時間考量，沒辦法優化的更好，可能也是等這禮拜再趕工看看能不能在 final project 有所突破了。

這邊做了一些 sdram ctrl 和 signal converter 的 protocol 定義的筆記，一併附上

SDRAM Controller

pin :

cle : keep high in this project (command latch enable)

cs : chip select

cas : column address strobe

ras : row address strobe

we : write enable

dqm : data I/O mask 掩碼控制位 - 不需要的 data 不寫入

ba : bank address

a : 行列共用 address

dqi : 數據輸入

dqo : 數據輸出

---

user

user_addr : 決定 "read" 或 "write"

0 - 7 : column address

8 - 9 : bank address

10 - 22 : row address

rw : write = 1, read = 0

data_in, data_out : 輸入輸出

busy : idle or not

in_valid, out_valid : handshake signals

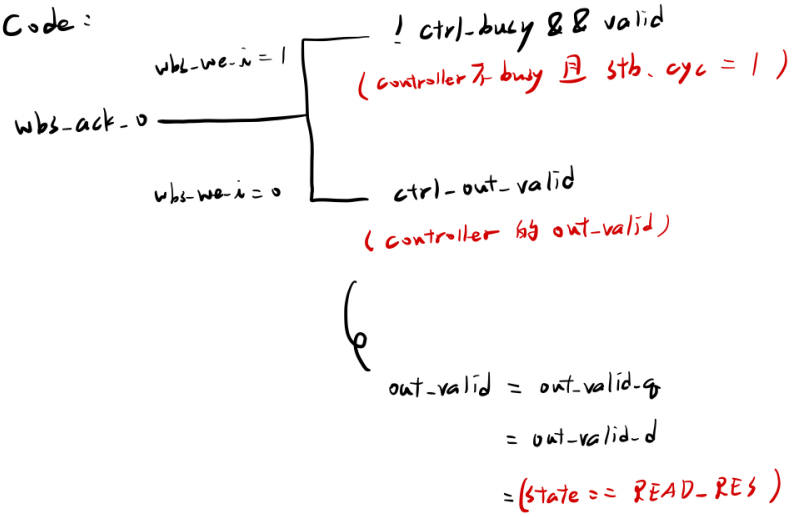
busy 和 out_valid 併成 wbs_ack_o

Converter

SPEC:

controller busy
controller out_valid
> wbs_ack_o (wishbone interface)

Code:




```
{
    out_valid = out_valid_q
              = out_valid_d
              = (state == READ_RES)
```

data_d
- Default : data_q
- IDLE : save_data_q when write
- READ_RES : dqi_q

output = data_q @ clk <= data_d

---

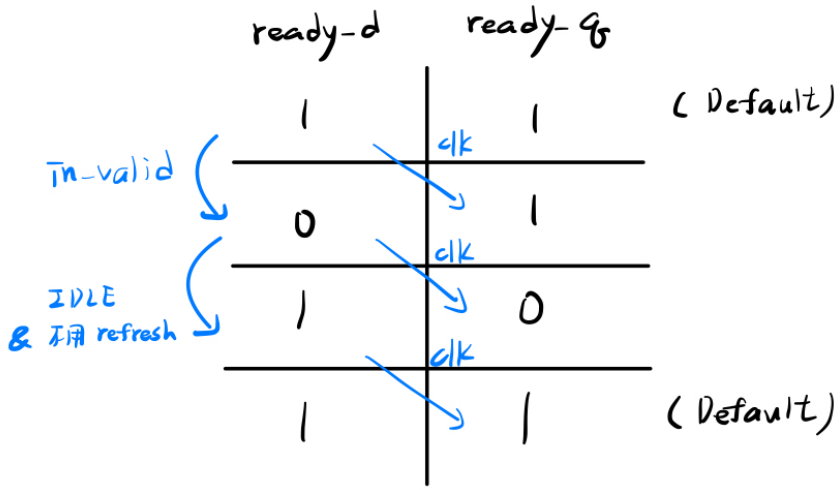
sdram controller
- data_in, data_out) wishbone
- sdram_dqi, sdram_dqo) sdram

---

state_q :
- WAIT = 1 → 3T
- IDLE = 6 → 3T
- REFRESH = 7
- ACTIVE = 8 → 1T
- READ = 9 → 1T
- READ_RES = 10 (A) → 1T
- WRITE = 11 (B)
- PRECHARGE = 12 (C)

6 → 9 → 1 → A

3T 1T 3T 1T



- ready-d @ ✳
- ready-qf @ clk

request —1T→ busy —5T→ ack —1T→ request

- 先一直送 request，存在 prefetch buffer，再一次送出，

  burst = 8 (bit)