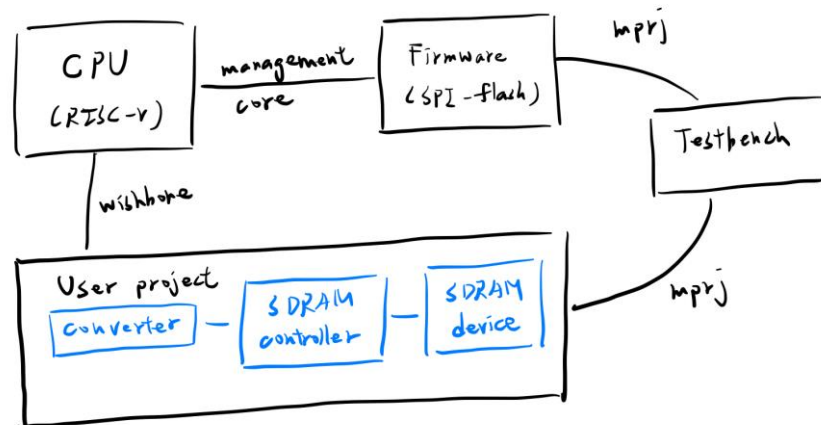
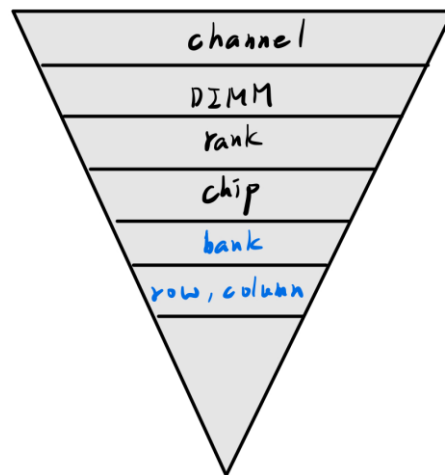


Architecture



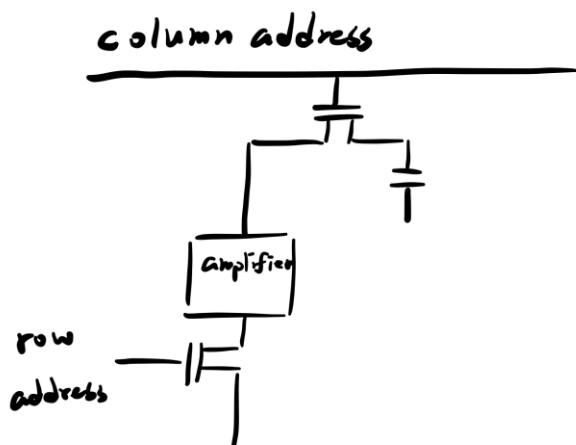
這次的 lab 主要是把之前的 system 中 bram 改成用 sdram 來存，這邊先畫一個簡單的 hierarchy



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 row，來

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 **row** 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

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 $1/\text{burst}$ 的量，就能達到 high speed 的預期，提高讀寫效率

Code

The timing diagram displays various signals over a time period from 1104500 ns to 1105100 ns. The signals include control signals like `clk`, `valid`, `wbs_cyc_i`, `wbs_stb_i`, `out_valid`, `wbs_ack_o`, `wbs_we_i`, `busy`, `cle_q`, `cmd_q[3:0]`, `dq_en_q`, `in_valid`, and data signals like `wbs_addr_i[31:0]`, `wbs_dat_i[31:0]`, `wbs_dat_o[31:0]`, `addr[22:0]`, `a_d[12:0]`, `ba_q[1:0]`, `rw`, `data_d[31:0]`, `data_in[31:0]`, `data_out[31:0]`, `data_q[31:0]`, `delay_ctr_q[15:0]`, `dq_q[31:0]`, and `dq_i_q[31:0]`. The diagram shows the timing relationships between these signals, with a vertical red line indicating a specific point in time.

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

```
5 ## User project
6 -v ../../rtl/user/user_project_wrapper.v
7 -v ../../rtl/user/user_proj_example.counter.v
8 -v ../../rtl/user/sdram_controller.v
9 -v ../../rtl/user/sdr.v

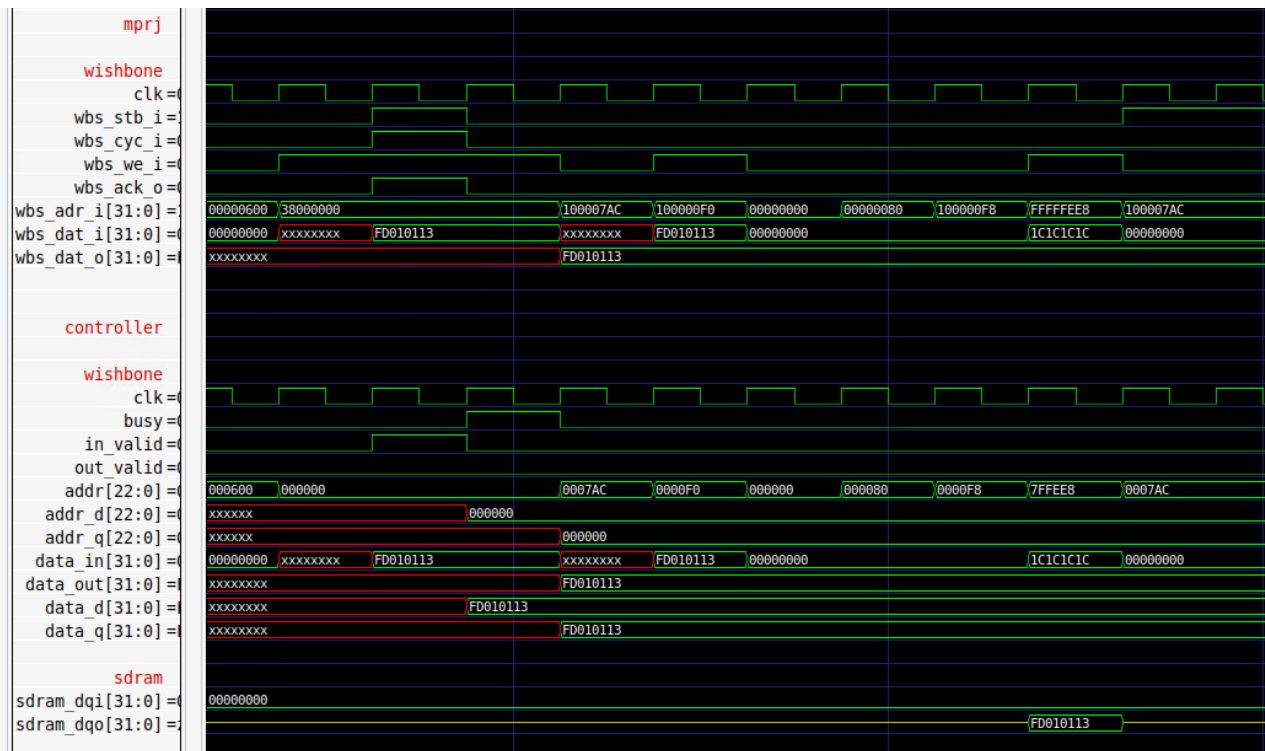
36 ## STD CELLS - they need to be below the defines.v files
37 #-v ../../cvc-pdk/sky130_sram_2kbyte_1rw1r_32x512_8.v
```

接著就是將 multiple 的結果寫進 sdram

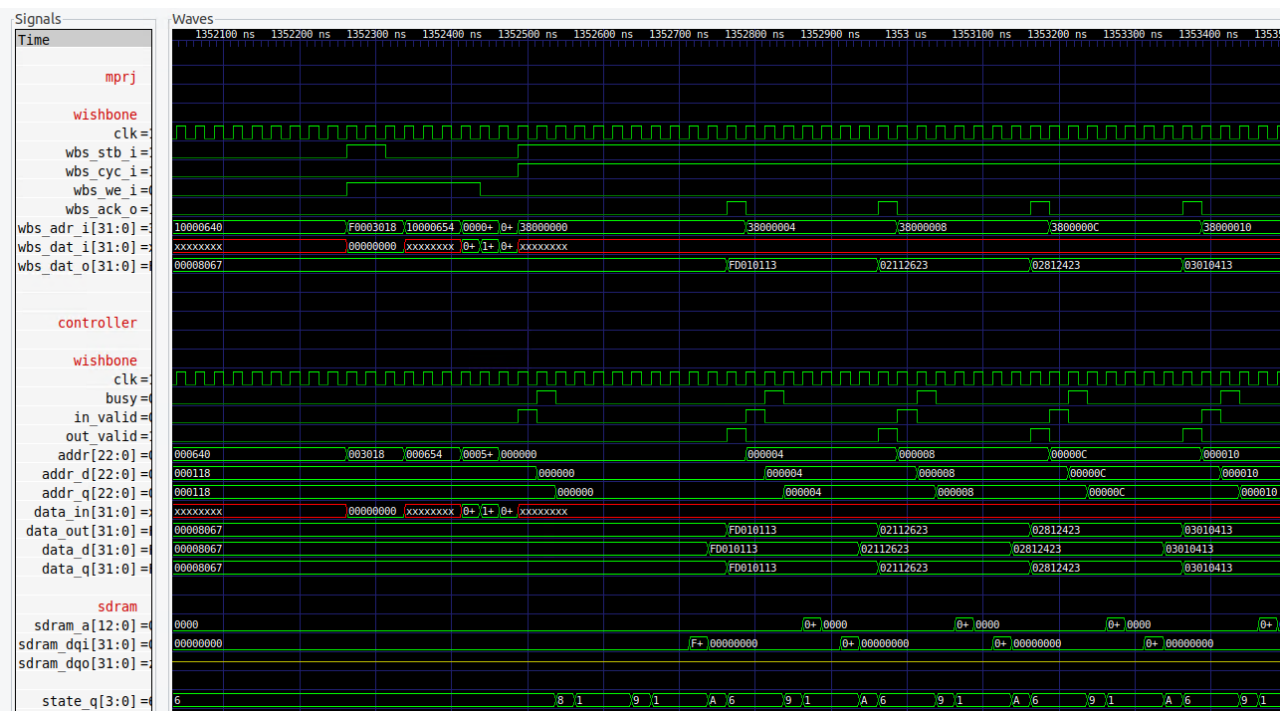
```
jimmy@jimmy-VirtualBox:~/Desktop/caravel-soc_fpga-lab/lab-sdram/testbench/counte
r_la_mm$ source run_sim
Reading counter_la_mm.hex
counter_la_mm.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la_mm.vcd opened for output.
LA Test 1 started
Call function 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：

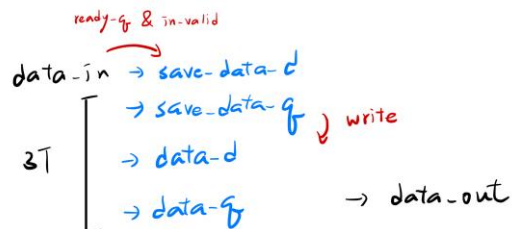
ACTIVATE 1T
 READ (9) 1T
 WAIT (1) 3T
 READ RESET(A) 1T
 IDLE(6) 3T

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

data_in[31:0] =	00000000	08400713	00279793	xxxx+	0027+
data_out[31:0] =	08400713		00279793		
data_d[31:0] =	08400713		00279793		
data_q[31:0] =	08400713		00279793		

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：

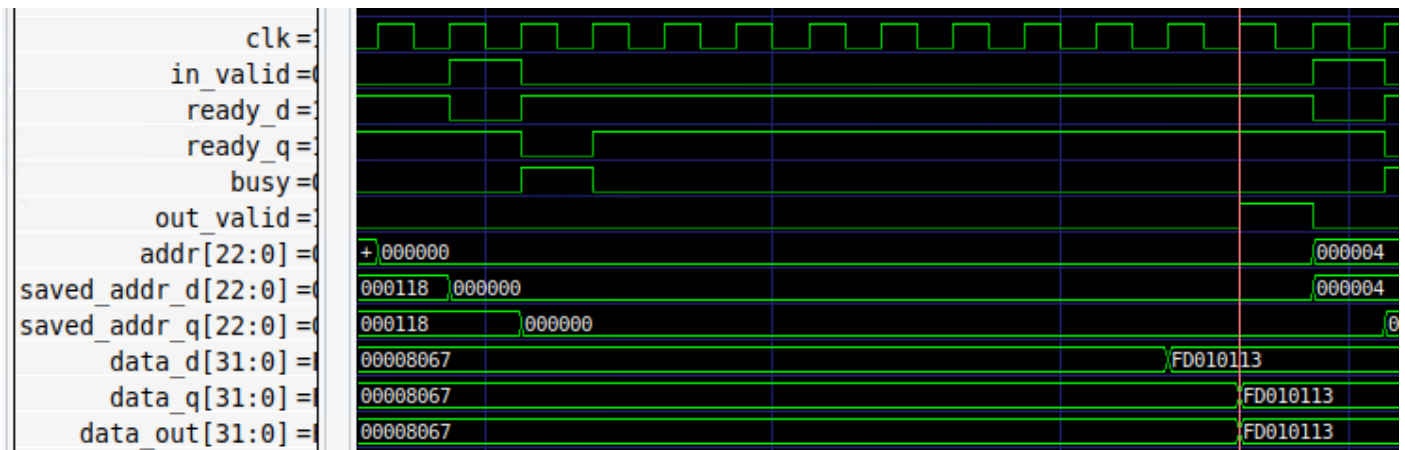
ready_d =	
ready_q =	
state_q[3:0] =	6 8 1

	ready-d	ready-q	
	1	1	(Default)
in-valid	0	1	clk
	1	0	clk
IDLE & 不用 refresh	1	1	clk
	1	1	(Default)

▷ ready-d @ *

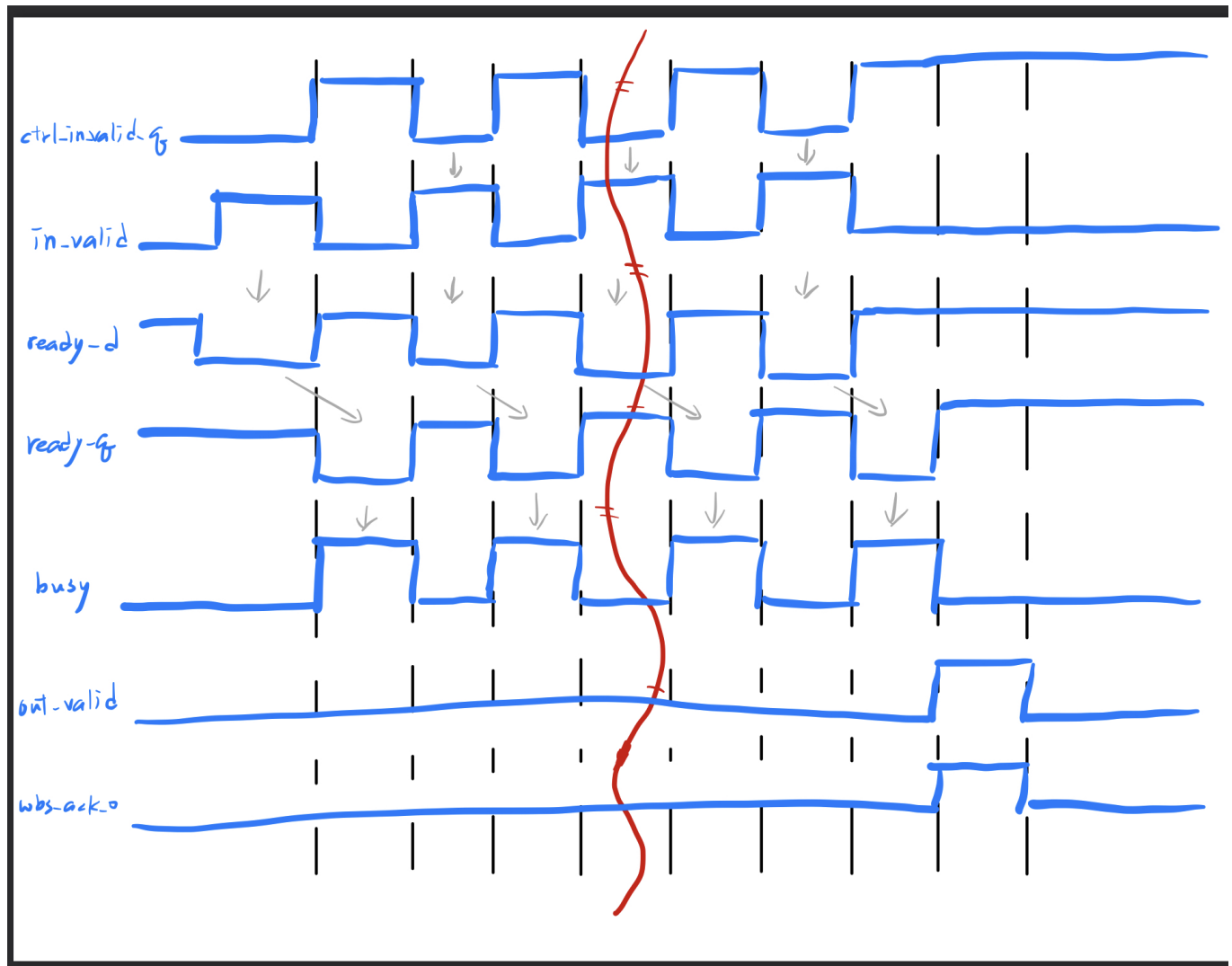
▷ ready-q @ clk

Optimization

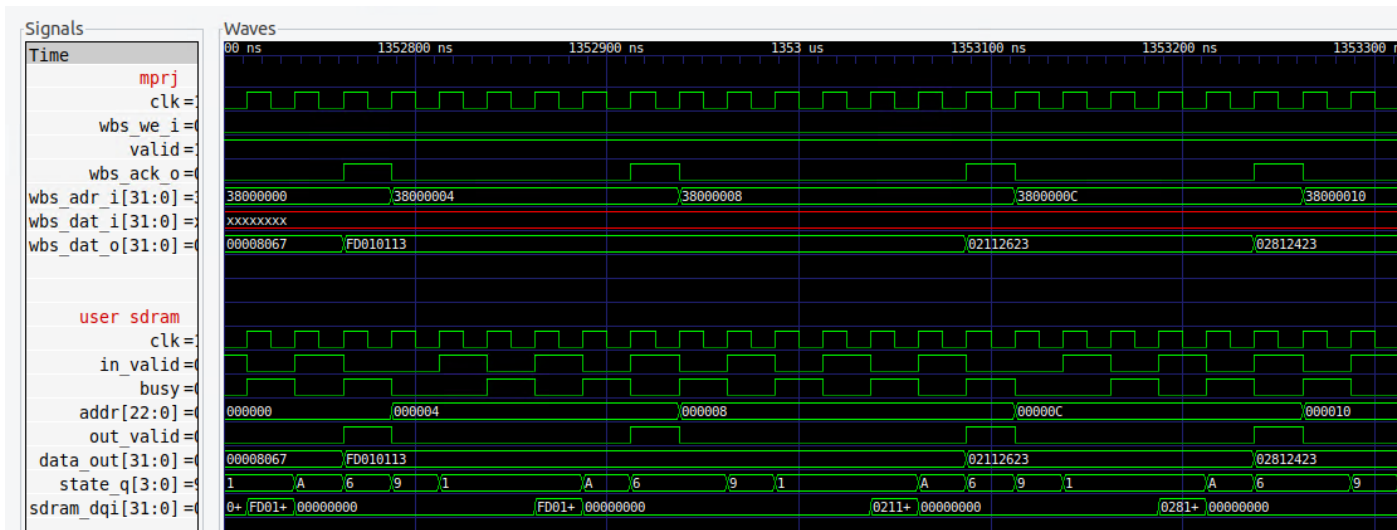


從這張 waveform 可以發現 request 之後 data 很晚才收到，所以 out_valid 一直在 wait，下一個 request 也跟著 wait

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：

$$9T + 1T + (7T + 1T) * 7 = 66T$$

$$9T + 1T + (5T + 1T) * 7 = 52T$$

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

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。

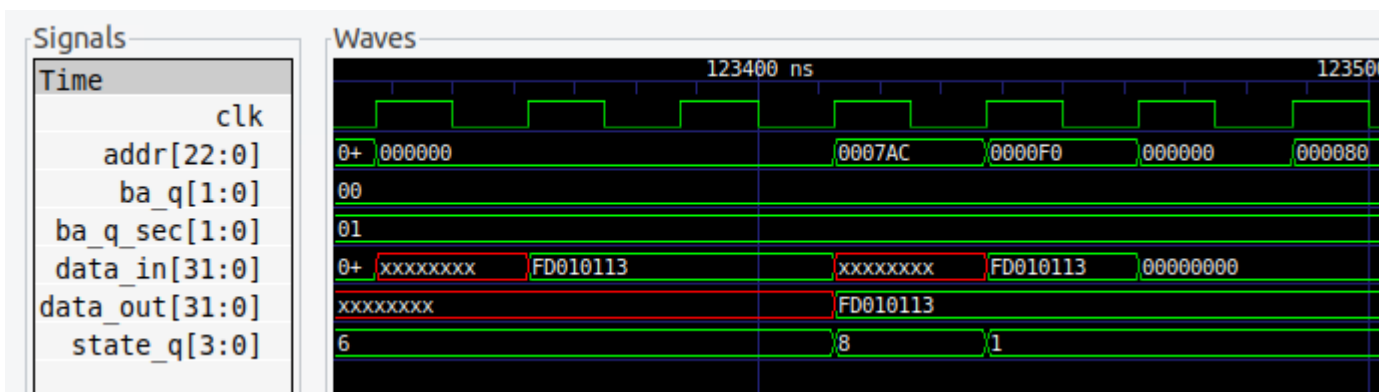
Bank interleave

這邊我的想法是將奇數 address 和偶數 address 分開

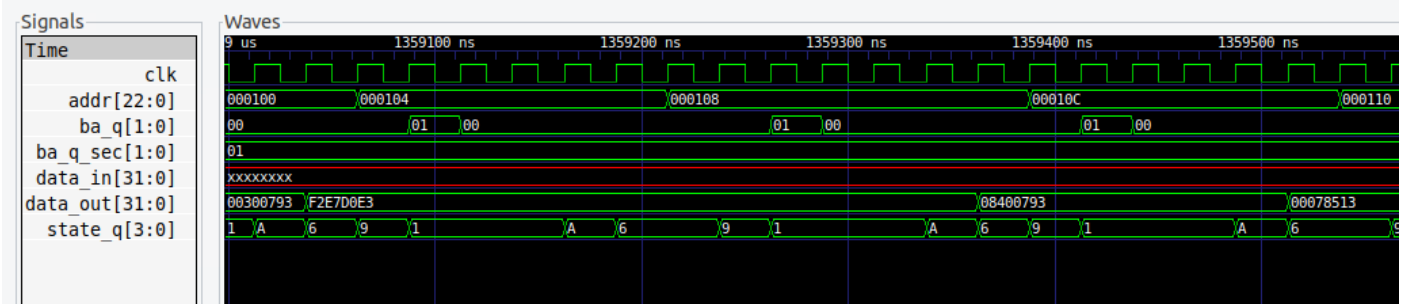
Pipeline 起來的感覺就是 13579...和 246810...的 bit 可以分開存取

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 - 不需要的数据不寫入
- 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 go out-valid 併成 wbs_ack

Converter

SPEC:

controller busy
controller out-valid

\bigvee $wbs_ack = 0$ (wishbone interface)

Code:

$wbs_we_i = 1$
 $wbs_ack = 0$
 $wbs_we_i = 0$

$\left\{ \begin{array}{l} \neg ctrl_busy \ \&\& \ valid \\ (ctrl_busy \ \&\& \ stb_cyc = 1) \\ ctrl_out_valid \\ (ctrl_out_valid) \end{array} \right.$

$\left(\begin{array}{l} out_valid = out_valid_q \\ = out_valid_d \\ = (state == READ_RES) \end{array} \right.$

$\text{data_d} \begin{cases} \text{Default} & = \text{data_q} \\ \text{IDLE} & = \text{save_data_q when write} \\ \text{READ_RES} & = \text{dq_i_q} \end{cases}$

$\text{output} = \text{data_q} @ \text{clk} \leq \text{data_d}$

$\text{Sdram controller} \begin{cases} \text{data_in} \\ \text{data_out} \end{cases} \text{ wishbone}$
 $\begin{cases} \text{sdram_dq_i} \\ \text{sdram_dq_o} \end{cases} \text{ sdram}$

$\text{state_q} : \text{WAIT} = 1 \rightarrow 3T$
 $\text{IDLE} = 6 \rightarrow 3T$
 $\text{REFRESH} = 7$
 $\text{ACTIVE} = 8 \rightarrow 1T$
 $\text{READ} = 9 \rightarrow 1T$
 $\text{READ_RES} = 10 (A) \rightarrow 1T$
 $\text{WRITE} = 11 (B)$
 $\text{PRECHARGE} = 12 (C)$

$6 \xrightarrow{3T} 9 \xrightarrow{1T} 1 \xrightarrow{3T} A \xrightarrow{1T}$

	ready-d	ready-q	
	1	1	(Default)
In-valid	0	1	clk
	1	0	clk
IDLE & 不用 refresh	1	1	clk
			(Default)

▷ ready-d @ *

▷ ready-q @ clk

request \xrightarrow{IT} busy \xrightarrow{ST} ack \xrightarrow{IT} request

▷ 先一直送 request, 存存 prefetch buffer, 再一次送出,

burst = 8 (bit)