系統晶片設計

SOC Design

LAB3

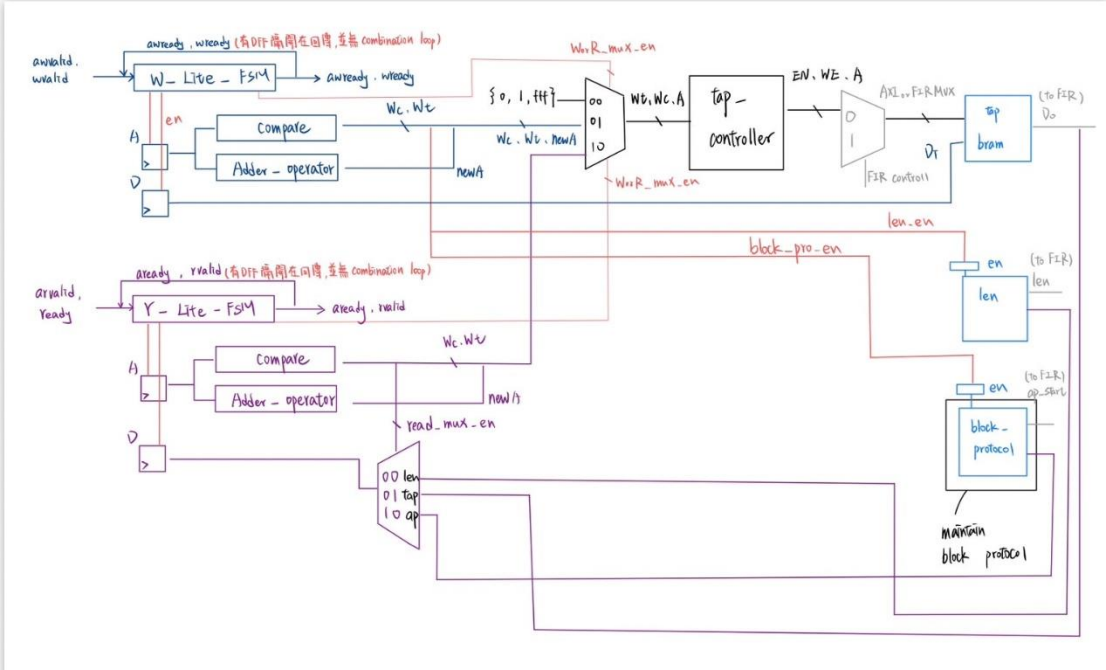
郭銘宸

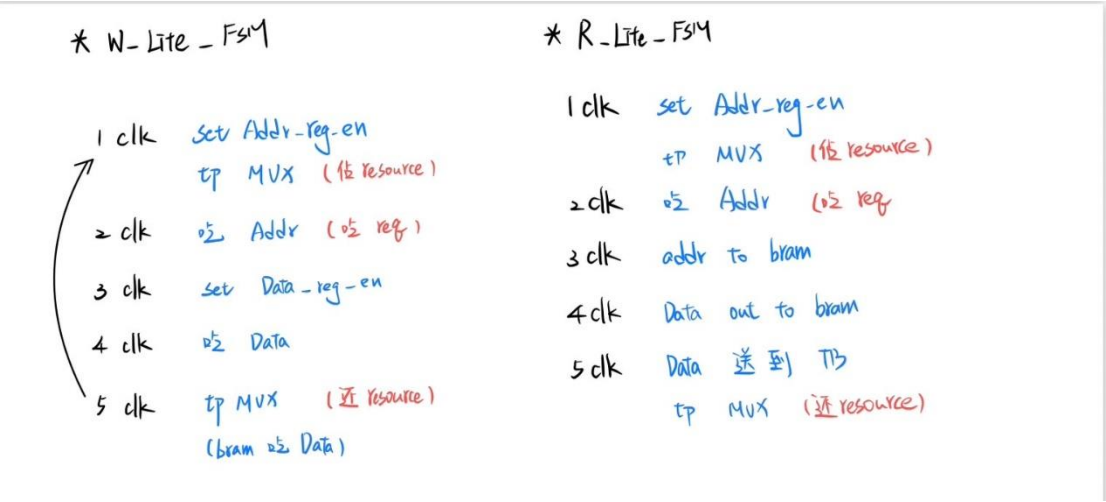
學號:311511082

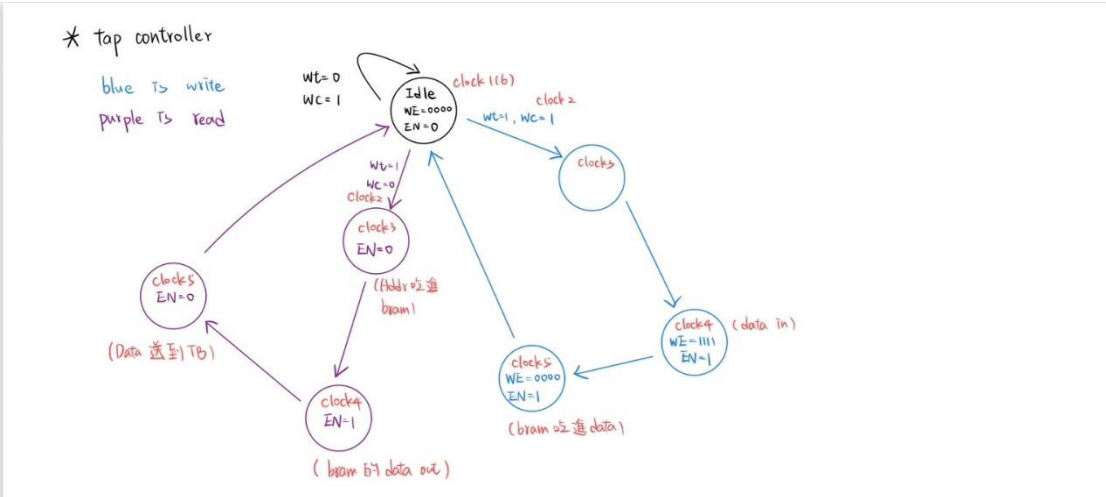
系所:電機所碩二

一、Block Diagram

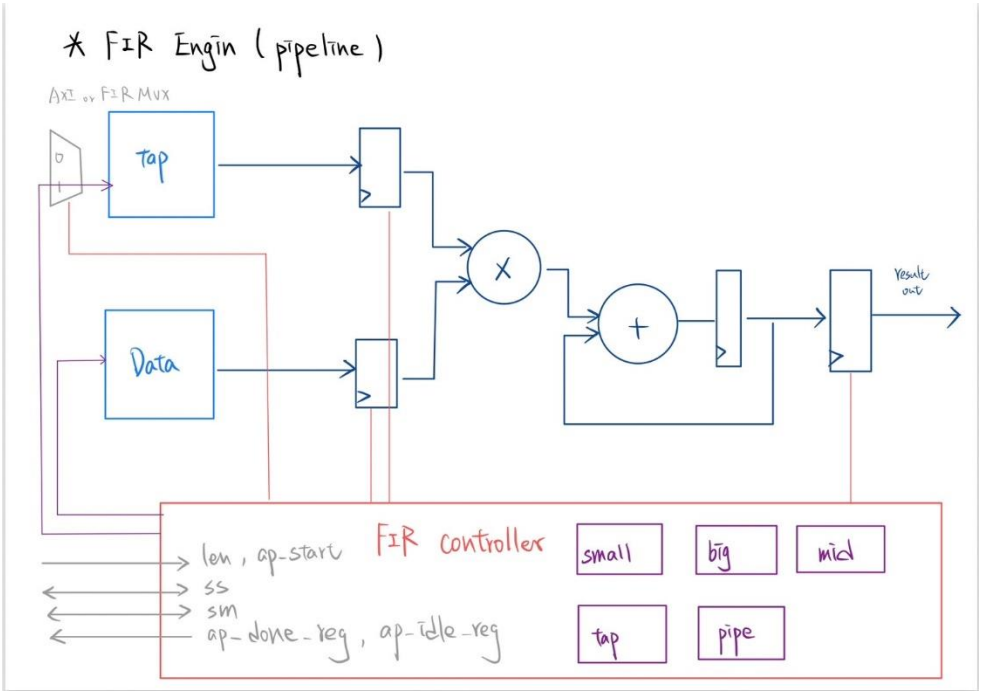
1. AXI-Lite Interface

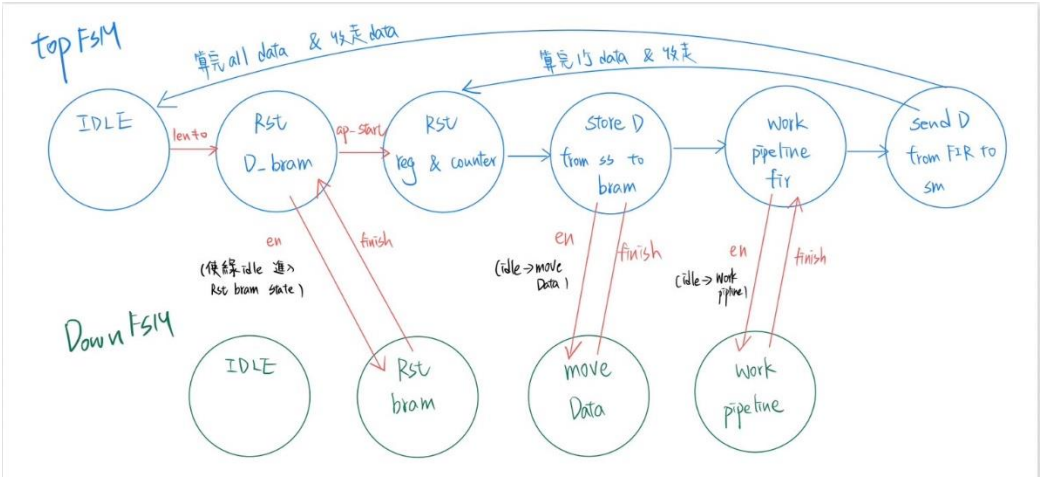


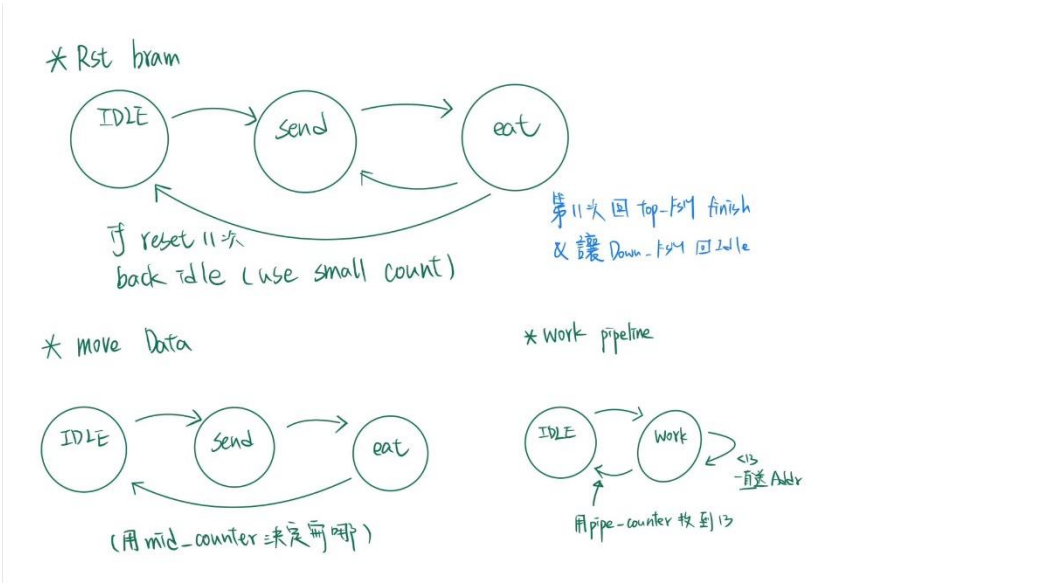




2. FIR Engine


* FIR Engin (pipeline)
AXI or FIR MUX
tap
Data
X
+
Result out
len, ap_start
FIR controller
small
big
mid
ss
sm
ap_done_reg, ap_idle_reg
tap
pipe

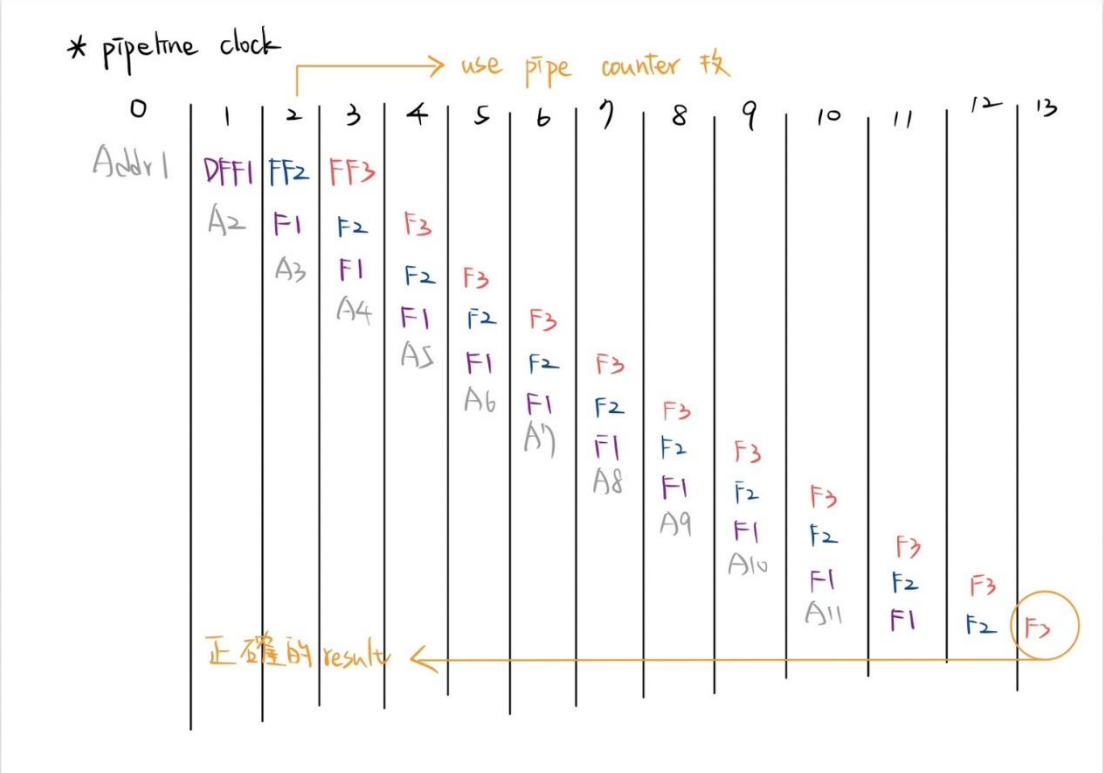

top FSM
算完all data & 收完data
算完1j data & 收完
IDLE
len!=0
Rst D_bram
ap_start
Rst reg & counter
Store D from ss to bram
work pipeline fir
Send D from FIR to sm
en
finish
(使歸idle 進入 Rst bram state)
(idle->move Data)
(idle->work pipline)
Down FSM
IDLE
Rst bram
move Data
work pipeline


* Rst bram
IDLE
send
eat
If reset 11次
back idle (use small count)
第11次回 top-FSM finish
& 讓 Down-FSM 回Idle
* move Data
IDLE
send
eat
(用 mid_counter 決定需哪)
* work pipeline
IDLE
work
<13
一直送Addr
用pipe-counter 收到13

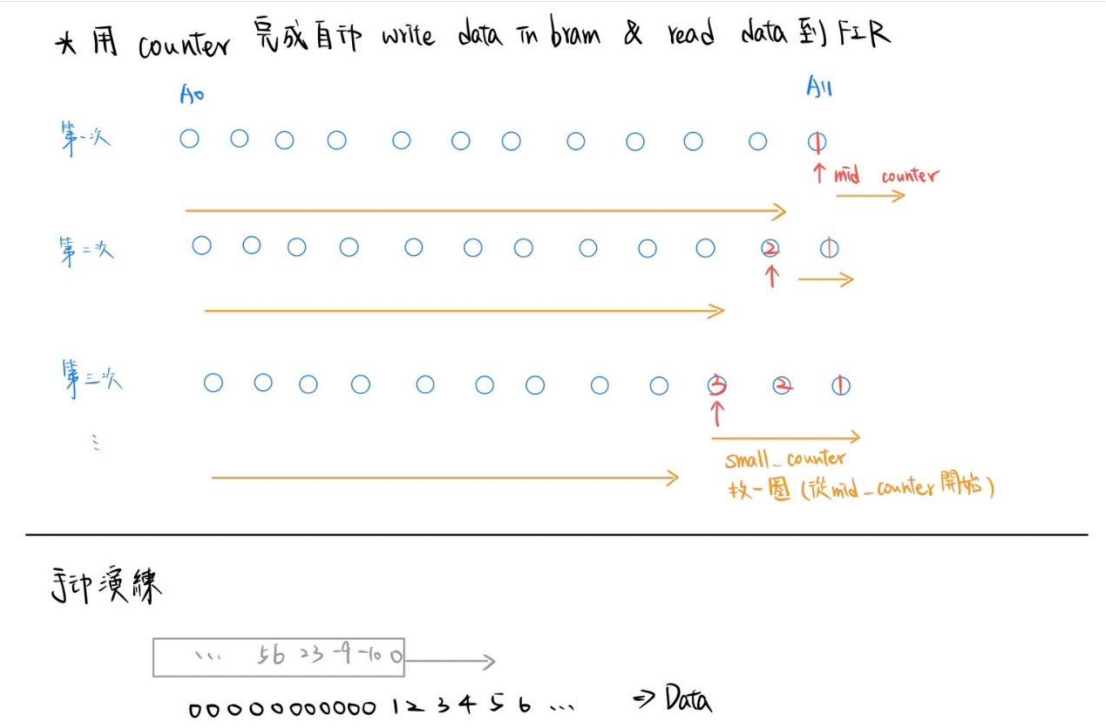
二、Describe operation

1. 描述架構圖

- 會有 AXI_lite_write、AXI_lite_read 與 FIR Engine 三者對 tap bram 送 address、EN、WE，為了避免三者輸入撞在一起產生 unknow 的情況，所以有使用兩個 MUX 將訊號隔開。黑色 MUX 是存在於 AXI_Lite_Interface 中，對於接收 request 而言，這個 MUX 是決定是否能將地址(request)接收至 bram 的資源，所以在 write 與 read 的 FSM，當某一方接收到 request 的 valid 必須先將此資源占用，才能接收地址，當處理完 request 再歸還此 MUX 資源。後面灰色 MUX 則是 FIR controller 控制，當 controller 在接收到 ap_start=1 之前，皆是 0(lite 訊號通過)，當 FIR 開始工作則會切到 1，待 FIR 作完全部工作切回 0，注意此時 lite interface 還是可以 read length 與 block protocol，只是不能對 tap bram access 而已。
- FIR_Controller FSM，在收到長度後可以先進行 data bram 的 reset，FIR 運算須將前面填充 11 個 0 在開始滑動 window，所以對 data bram 的 reset 0 是不可避免的，若與 lite 在傳 tap 時同時先 reset 可以節省時間，歸零完會停留在 rst_d_dram 狀態(藍色)等待 ap_start=1 啟動運算。
- FIR pipeline 設計



- Tap & data pointer 設計



2. How to receive data-in and tap parameters and place into SRAM
   - Tap

     會由 w_Lite_FSM 來管理 lite interface 的資源，當順利占用 MUX 會會將地址平行左兩件事，第一是 compare，這區塊用意為解析地址為何，發訊號給對應儲存 reg 將對的 data 存起來，例如若地址為 length 的地址，則會發訊號給 length register 讓她們將 data 儲存。Read 也相同，只是多一個 mux 決定要讓哪一個 data 出去。
   - Data-in

     由 FIR controller FSM 中的 store_D_from_ss_to_bram 來管理，會這樣做是因為後面 fir 運算資源只有一套，就算切完 pipeline 後也需要 14cycle 來完成一筆運算，如過用 fifo 很快就滿了，因為 data 消耗很慢，所以就用一個 state 來完成接收 data。在這個 state 會用 mid_counter 來決定要存哪一個地址，他會從地址 10 往回數且不同重複，這樣的設定可以讓後面 pipeline 運算的 data 自動化，此 mid_counter 為完成一個正確輸出時才會減 1。
3. How to access shiftRam and tapRam to do computation
   - shiftRam

     對於 FIR 運算資料來源，也就是 access data bram 的地址我用 small_counter 來當我的 pointer，small_counter 會從 mid_counter 的地址開始往上數 11 個，所以會先將 mid_counter 的地址存入，

small_counter。

- tapRam

  tap 的地址由 tap_counter 來當 pointer，從 0 數到 10。

4. How ap_done is generated

   當 FIR_Controller FSM 在 send_D_from_fir_to_sm 狀態時，裡面會有小型 FSM，當運算次數與輸入長度相同時(big_counter==599)，則會產生 ap_done，然後當收到 sm 的 ready 也就是資料被收走，則會產生 ap_done。可以由下圖看到當 FIR 完成全部運算且資料收走 idle 與 done 都會是 1，當 ap_done 被讀取，則 ap_done 為 0，ap_idle 為 1。

```
OK: exp =           0, rdata =           0
[PASS] [Pattern        598] Golden answer:       -1098, Your answer:       -1098
------End the data input(AXI-Stream)------
[PASS] [Pattern        599] Golden answer:        -915, Your answer:        -915
OK: exp =           2, rdata =           6
OK: exp =           4, rdata =           4
----------Congratulations! Pass------------
---------------Round 3 End-----------------
$finish called at time : 418015 ns : File "/home/ubuntu/course/lab3/fir/tb/fir_tb.v" Line 299
exit
INFO: [Common 17-206] Exiting xsim at Sat Oct 21 06:29:14 2023...
```

三、Resource usage : including FF, LUT, Bram

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	329	0	0	53200	0.62
LUT as Logic	329	0	0	53200	0.62
LUT as Memory	0	0	0	17400	0.00
Slice Registers	471	0	0	106400	0.44
Register as Flip Flop	300	0	0	106400	0.28
Register as Latch	171	0	0	106400	0.16
F7 Muxes	0	0	0	26600	0.00
F8 Muxes	0	0	0	13300	0.00

1.1 Summary of Registers by Type

Total	Clock Enable	Synchronous	Asynchronous
0	_	-	-
0	_	-	Set
0	_	-	Reset
0	_	Set	-
0	_	Reset	-
0	Yes	-	-
27	Yes	-	Set
444	Yes	-	Reset
0	Yes	Set	-
0	Yes	Reset	-

2. Memory

Site Type	Used	Fixed	Prohibited	Available	Util%
Block RAM Tile	0	0	0	140	0.00
RAMB36/FIFO*	0	0	0	140	0.00
RAMB18	0	0	0	280	0.00

* Note: Each Block RAM Tile only has one FIFO logic available and therefore (

3. DSP

Site Type	Used	Fixed	Prohibited	Available	Util%
DSPs	3	0	0	220	1.36
DSP48E1 only	3				

四、Time report

1. Try to synthesize the design with max frequency
   可以看到我最長 path 為中間 fir 的乘加運算(也就是 data bram 出來的第一個 reg 到加完存起來的 reg)，若在乘法完結果後再加一級 pipeline 可以有效降低 clock。從最長 path 看到我需要 10.672ns 完成乘加，所以最大頻率需大於這個，在 constraints 我設 12 是有 met 的，如下圖示。
2. Report timing on longest path, slack

```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack (MET) :             1.223ns  (required time - arrival time)
  Source:                 fir_data_reg_out_reg[16]/C
                            (rising edge-triggered cell FDCE clocked by axis_clk  {rise@0.000ns fall@6.000ns period=12.000ns})
  Destination:            fir_result_out_reg_out_reg[29]/D
                            (rising edge-triggered cell FDCE clocked by axis_clk  {rise@0.000ns fall@6.000ns period=12.000ns})
  Path Group:             axis_clk
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            12.000ns  (axis_clk rise@12.000ns - axis_clk rise@0.000ns)
  Data Path Delay:        10.672ns  (logic 8.380ns (78.520%)  route 2.292ns (21.480%))
  Logic Levels:           9  (CARRY4=5 DSP48E1=2 LUT2=2)
  Clock Path Skew:        -0.145ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    2.128ns = ( 14.128 - 12.000 )
    Source Clock Delay      (SCD):    2.456ns
    Clock Pessimism Removal (CPR):    0.184ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
```

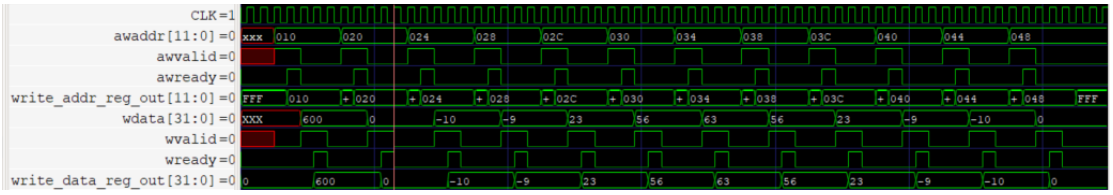
```
--------------------------------------------------------------------------------------
| Design Timing Summary
| ---------------------
--------------------------------------------------------------------------------------------------

    WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints      WHS(ns)      THS(ns)  THS Failing Endpoints  THS Total Endpoints     WPWS(ns)     TPWS(ns)
 TPWS Failing Endpoints  TPWS Total Endpoints
    -------      -------  ---------------------  -------------------      -------      -------  ---------------------  -------------------     --------     --------  ---
-------------------------------------
      1.223        0.000                      0                  319        0.137        0.000                      0                  319        5.500        0.000
                   0                  301
```

五、Simulation Waveform

1. Coefficient program, and read back

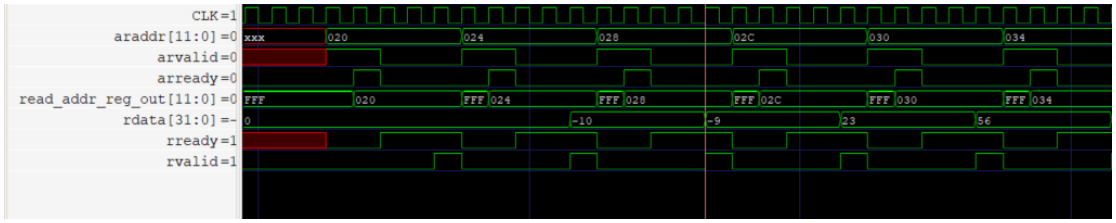
   ➢ Write

   Write_addr_reg_out 與 write_data_reg_out 為將地址與 data 吃進來的 reg，可以看到在 valid 與 ready 同時為 1 時回會將地址與 data 吃進來。



   ➢ Read

   對地址處理如 write 所示。可以看到當 rdata 為正確時會將 valid 升起，然後待 TB ready=1 收下(TB 這邊一直送 1)。



2. Data-in stream-in

   可以看到一開始當 ready 沒升起時不會收 data，當 ready 與 valid 同時為 1 才會收。

3. Data-out stream-out

   可以看出當資料有效時才會送 valid，最後一筆 last 升起 1cycle。

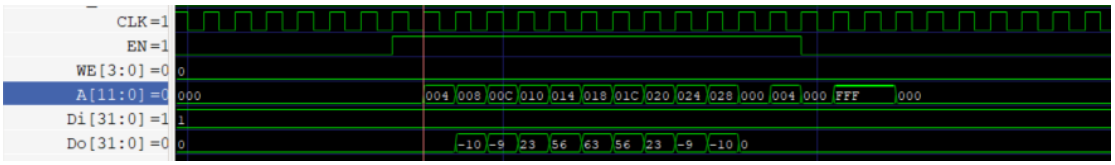
4. RAM access control

   ➢ Write

   可以看到 WE=4'b1111 && EN=1 時，下一個 clock 會將 Di 存到此地址。

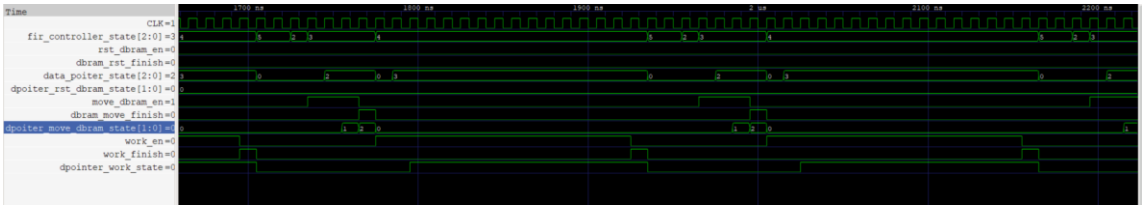
- Read

  可以看到 EN=1 時，下一個 clock data 讀出 bram。注意的是，EN 要在第二個 clock 時是 1，不然 data 會變 0。後面重複讀沒關係，因為那時候 valid 不會升起，不為有效資料。



5. FSM
   - FIR

   可以搭配上面 FIR_contoller_FSM 看，由下圖可以知道當 top_FSM(fir_controller_state)為某固定狀態時，會起動 down_FSM(data_pointer_state)，下面那三個為 down_FSM 切到某一狀態時會做的事情，當不屬於他的狀態時會回到 idle。



   - Tap bram control

   - Axi Lite FSM

     如上圖 protocol 圖一樣我是用此當作輸入。

   - Block protocol