National Cheng Kung University

Department of Electrical Engineering

Introduction to VLSI CAD (Spring 2022)

Lab Session 6

Design of Compressing system and Decompressing system

Name	Student ID	
郭家佑	F64096114	
Practical	Points	Marks
Lab 6_1	65	
Lab 6_2	35	
Notes		

Due: 15:00 April 13, 2022@ moodle

Deliverables

1) All Verilog codes including testbenches for each problem should be uploaded.
   NOTE: Please **DO NOT** include source code in the paper report!
2) All homework requirements should be uploaded in this file hierarchy.
3) NOTE: 1. Please **DO NOT** upload waveforms (.fsdb or .vcd)!
4) If you upload a dead body which we can't even compile, you will get NO credit!
5) All Verilog file should get at least 90% SuperLint Coverage.
6) All homework requirements should be uploaded in this file hierarchy or you will not get full credit, if you want to use some sub modules in your design but you do not include them in your tar file, you will get 0 point.

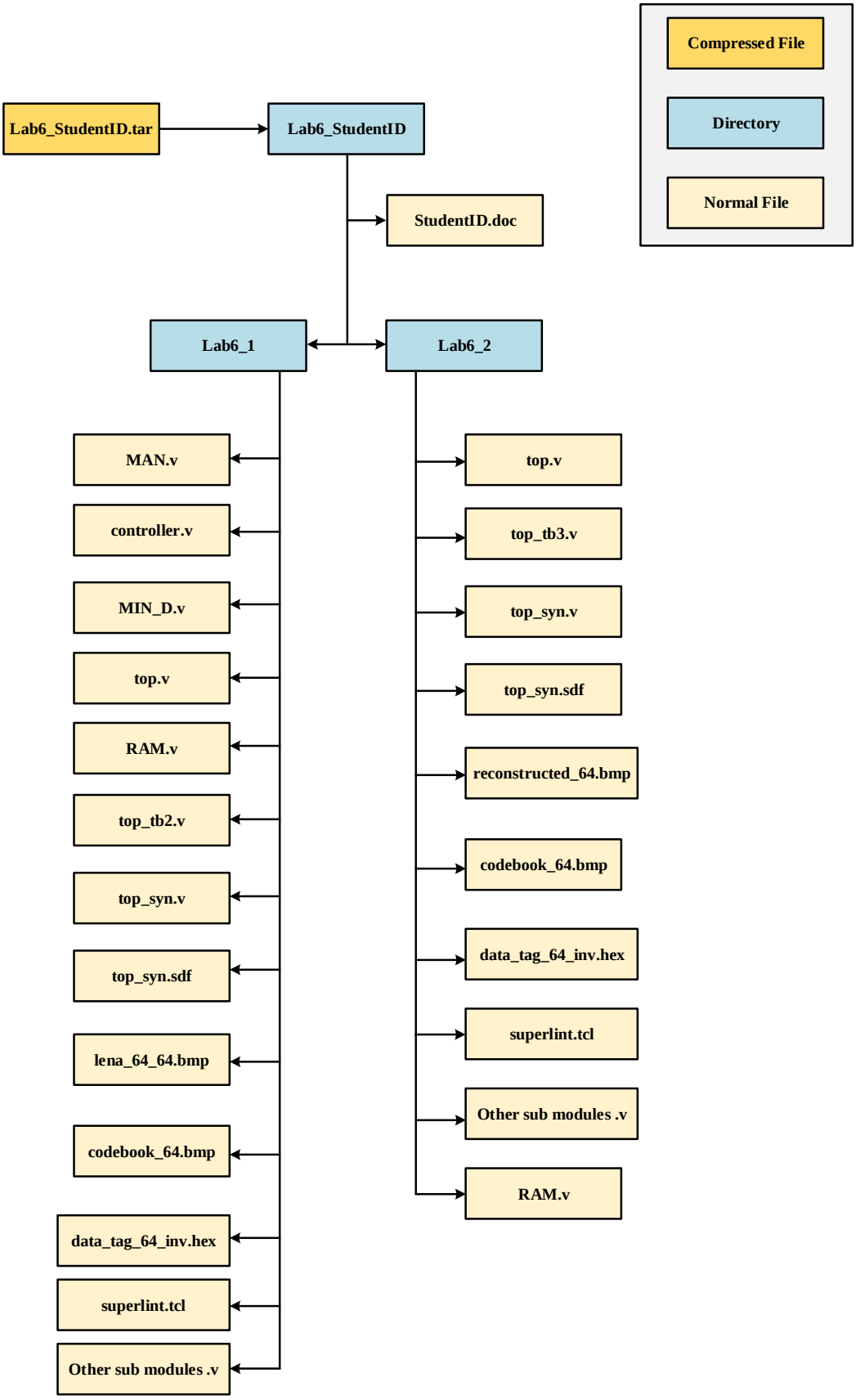
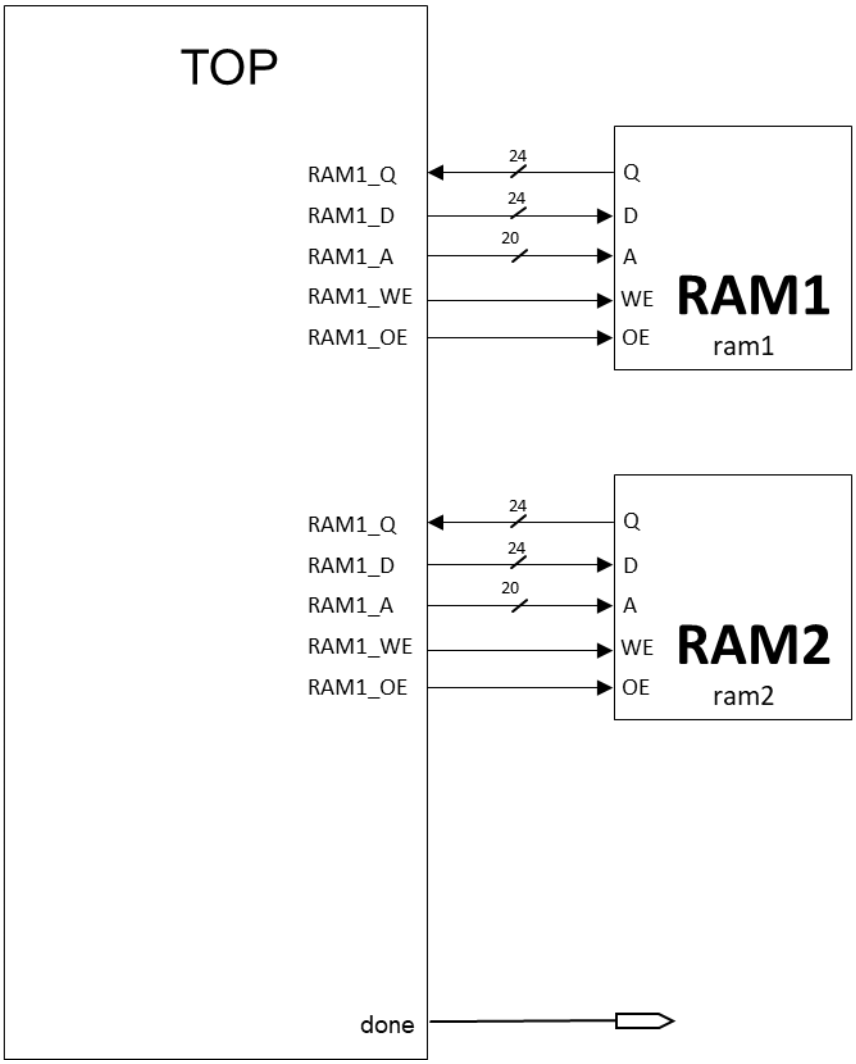


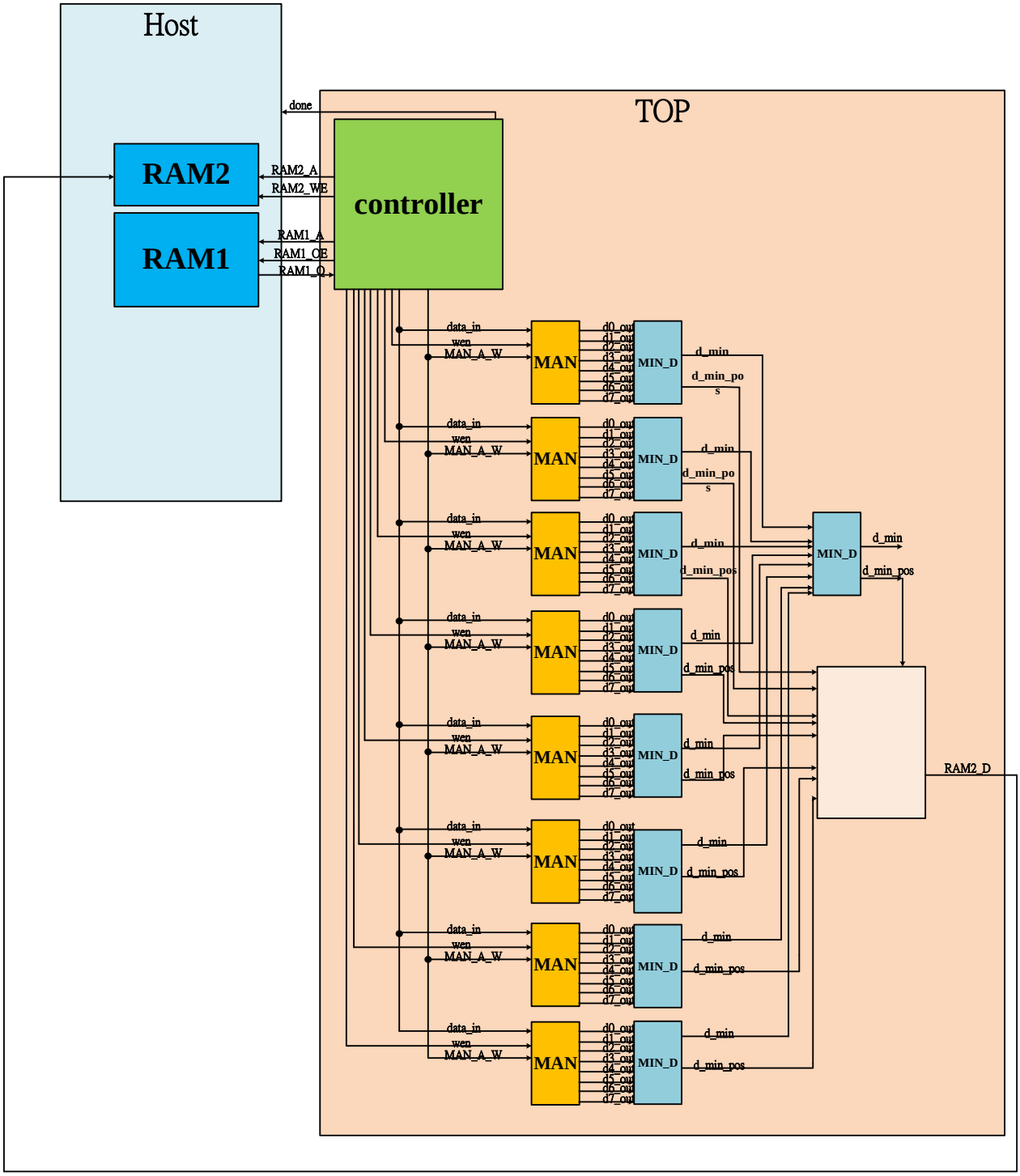
Fig.1 File hierarchy for Homework submission

Lab 6_1: compress the image with a codebook

You are about to integrate all components (MAN, MIN_D, controller…) to form a compressing system (*system*). The system will be the framework for your final design lab. The block diagram of system is as shown in **Fig2** and **Fig3**. (Clock pin and reset pin is ignored in the graph, but you should implement it)



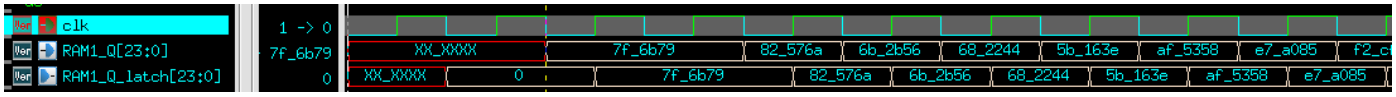
▲Fig2. The block diagram of system (external)



▲Fig3. The block diagram of system (internal)

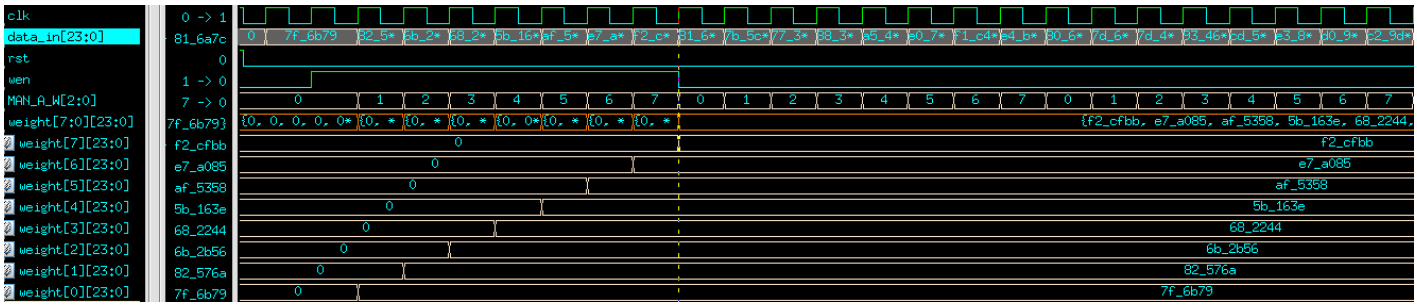
- **Port list of each module:**
- controller

signal	I/O	#bit	Description
clk	input	1	clock
rst	input	1	reset, active high, asynchronous
RAM1_Q	input	24	data output from RAM1
done	output	1	When all data tags are written into RAM2, done is pull to 1, or done is push to 0 at positive edge clk
RAM1_OE	output	1	RAM1 read enable signal
RAM1_A	output	20	RAM1 address signal
RAM2_WE	output	1	RAM2 write enable
RAM2_A	output	20	RAM2 address signal
RAM1_Q_latch	output	24	Store data output from RAM1 for MAN
wen0~wen7	output	1	Write enable signal for MAN
MAN_A_WEIGHT	output	3	Write address for MAN



- **MAN**

signal	I/O	#bit	Description
clk	input	1	clock
rst	input	1	reset, active high, asynchronous
data_in	input	24	data from RAM1
wen	input	1	Write enable signal，at positive edge clk，if wen is 1， write data_in into register file according to MAN_A_W
MAN_A_W	input	3	Write address for MAN
d0_out~d7_out	output	11	Manhattan distances between a pixel and 8 weights in a codebook
weight	reg	8x24	Store 8 weights in a codebook



- **MIN_D**

signal	I/O	#bit	Description
clk	input	1	clock
rst	input	1	reset, active high, asynchronous
d0~d7	input	11	8 Manhattan distances computed from MAN
d_min	output	11	The shortest distance among those 8 Manhattan distances
d_min_pos	output	3	Range from 0~7,for example, if id is 0, then it represent d0 is the smallest distance; If id is 1, then it represent d1 is the smallest distance, and so on.

- top

signal	I/O	#bit	Description
clk	input	1	clock
rst	input	1	reset, active high, asynchronous
RAM1_Q	input	24	data output from RAM1
RAM1_D	output	24	Data written into RAM1
RAM1_A	output	20	RAM1 address signal
RAM1_WE	output	1	RAM1 write enable
RAM1_OE	output	1	RAM1 read enable signal
done	output	1	When all data tags are written into RAM2, done is pull to 1, or done is push to 0 at positive edge clk
RAM2_D	output	24	Data written into RAM2
RAM2_A	output	20	RAM2 address signal
RAM2_WE	output	1	RAM2 write enable
RAM2_OE	output	1	RAM2 read enable signal

- Understanding the function:

  Once system is initialized, it

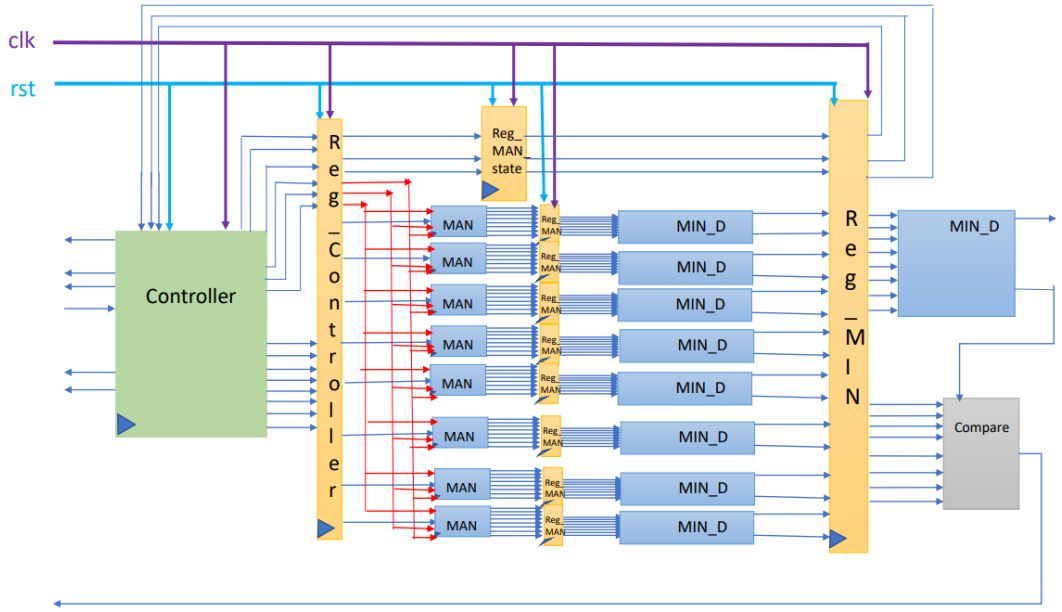
  a) read codebook from the RAM1 to 8 MAN instances
  b) read a pixel from the RAM1 at a time, and compute the Manhattan distances among that pixel and 64 weights in a codebook, after that, choose the id which represent the data tag for the weight having the shortest distance among other weight
  c) writes the data tag back to the RAM2;
  d) repeats the process step (b)-(c) until the last pixel of RAM1 is updated;
  e) flags “done” when step (d) is completed

- Know the basic design rules
  - All operations initiated on the positive edge trigger of the clock
  - Control signals:
    - *RAM_WE*: To store the data to RAM
    - *RAM_OE*: To read data from RAM

- *done*: Stop the process

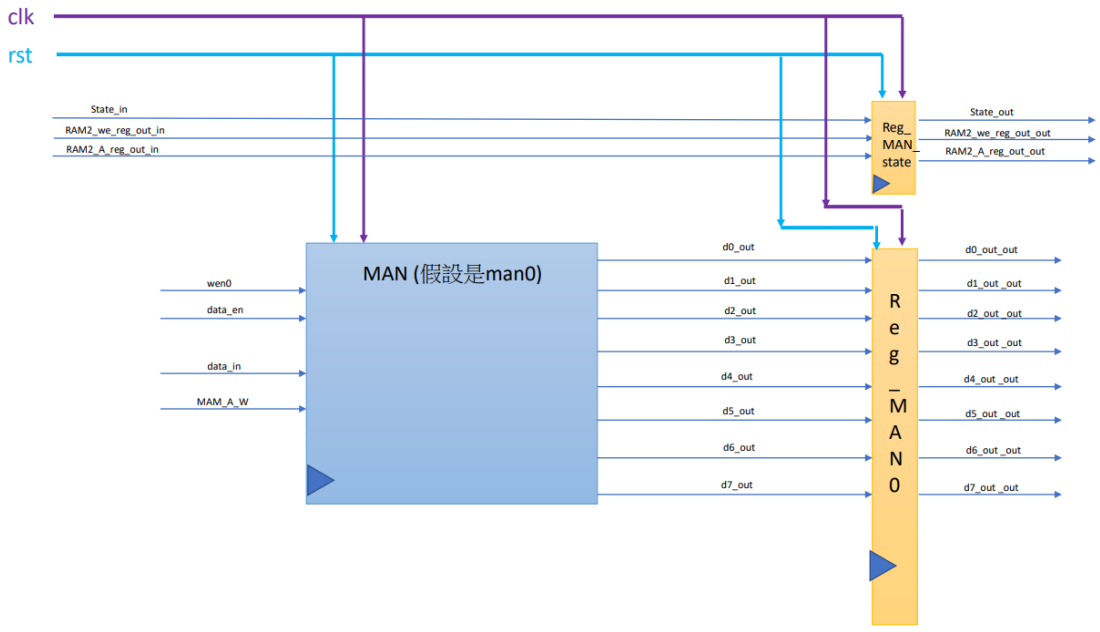
➢ Describe your design in detail. You can draw internal architecture or block diagram to describe your dsign.

■ Top



我的 top module 主要由 pipeline 的形式所組成。Controll 端由 controller 分配控制訊號，而 datapath 端由 11 個 register(4 個 stage 而已)及 8 個 MAN module、8+1 個 MIN_D 及 compare 所組成。

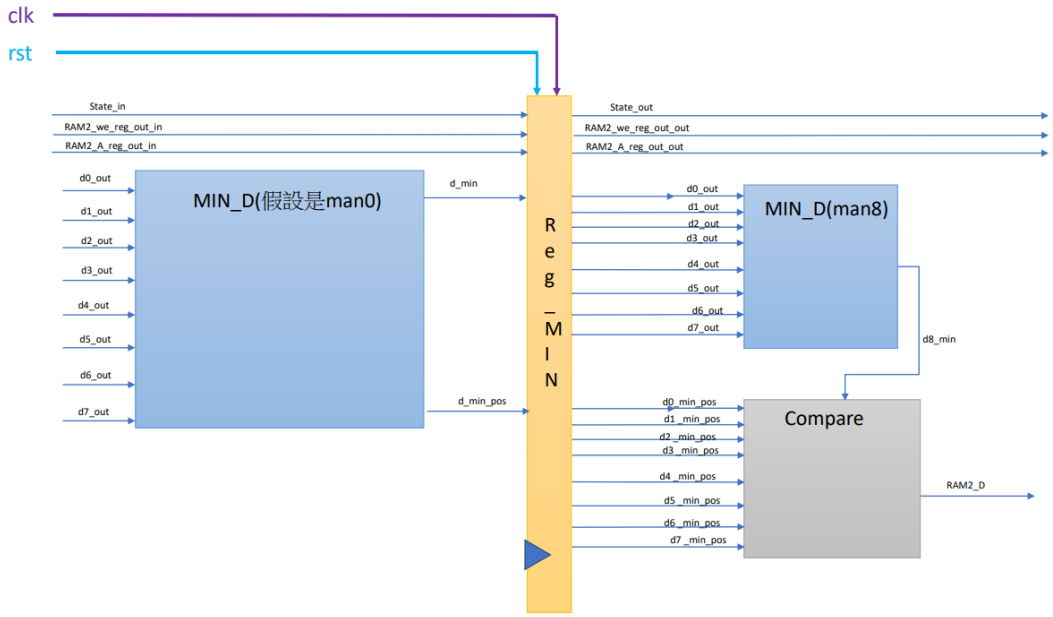
■ MAN



我的 MAN module 如附圖，內部會有 FlipFlop 來存 code book 的 weight，而 input 進來的 input_feature 及 input_color(weight)會共用 data_in 這個 port，所以要有 FSM 來記錄 state 是要寫 weight 還是開始比較了。而 MAN 內部會有 absR、absG、absB reg 來計算每一個的 distance，最後再由 outcome[9:0]作加總並輸出每一個與 weight 的距離(d0~d7)到下一級 register 站存。

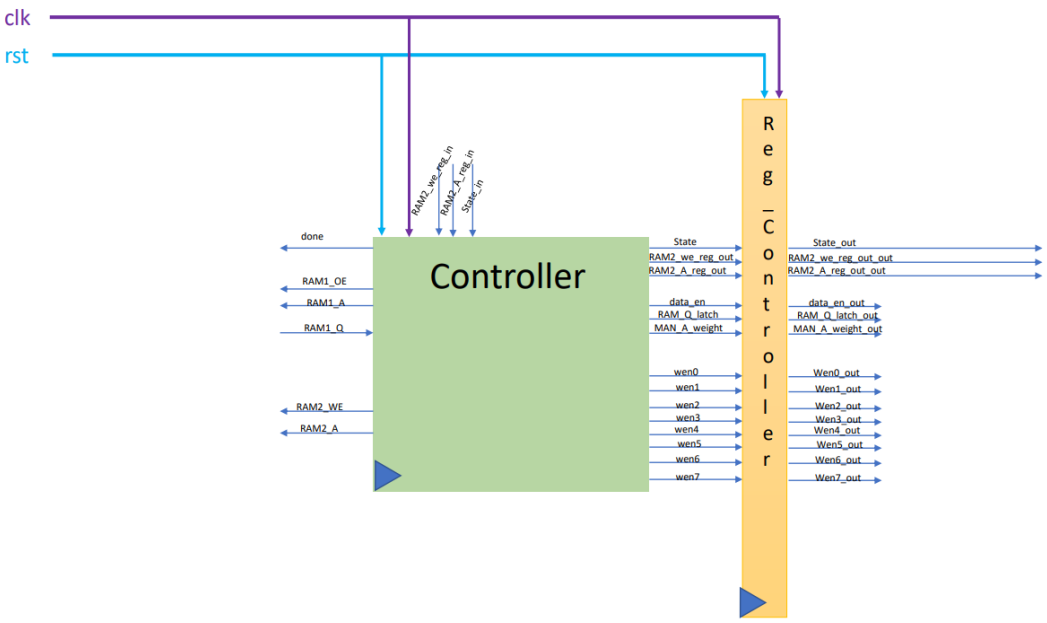
註:我在繳交作業前幾個小時要去修了一下 time violation，結果發現 reg_man 都會出問題，我的理解是我的 MAN 就是用 sequential 寫法了，所以就不用在多插一個 register 了，所以就把 reg_man 砍掉了。

- MIN_D



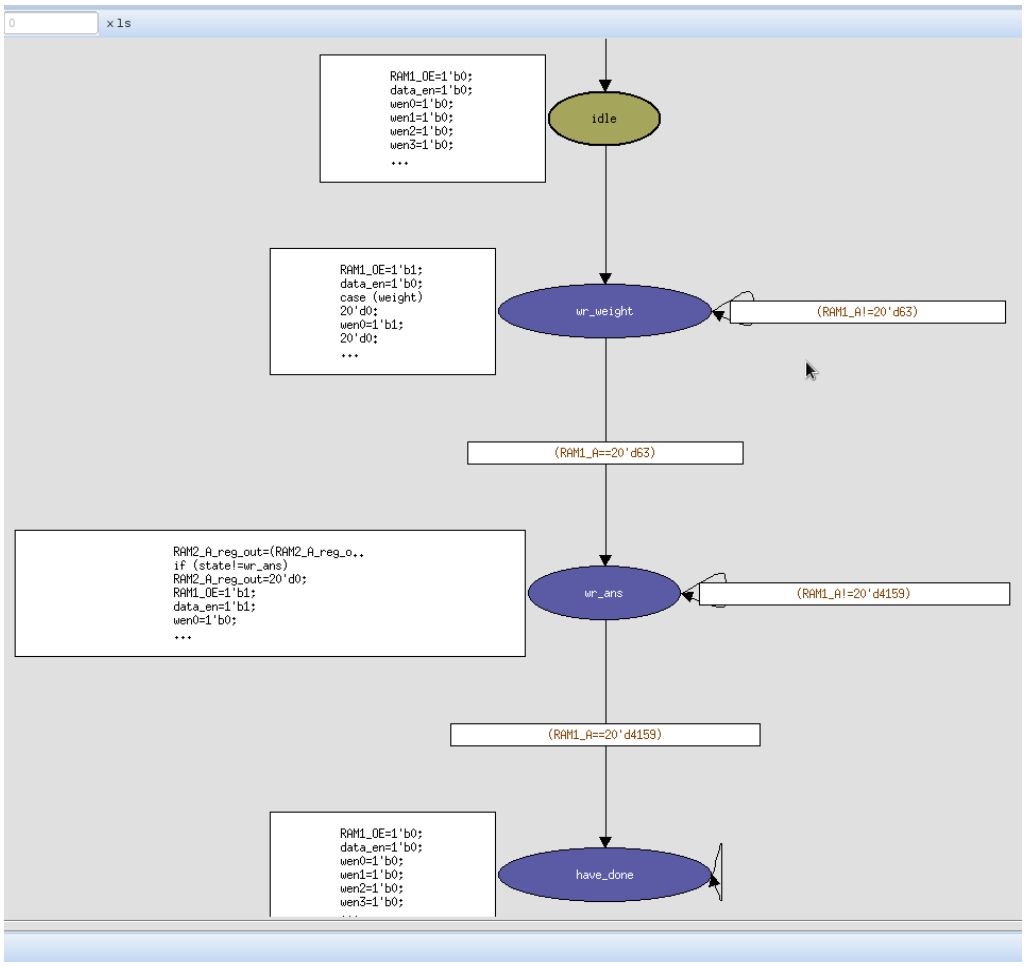
MIN_D 會收到前一級 MAN module 送來的 d0~d7，並利用內部共 7 個 2to1 mux 來篩選出最小的距離及他的 index 是幾號，並輸出到下一級 register 站存。而 register 中會存有 8 個 MIN_D 所計算出的 8 個 min 及 pos，他會將 8 個 min 再送去 MIN_D 再比一次得到最小的，而 pos 則送入 Compare 裡對照找出誰才是最小的 index，並將這個 index(tag)輸出。

- Controller



我個 controller 分成三端，左端是跟 top module 輸出做為緩衝用的；右端是控制各個 module 的訊號，如 wen、MAN_A_weight 則是由 RAM_A 決定；而上端三條則是經過 pipeline 傳回來要寫回去 RAM3 的判斷訊號。

- Draw your state diagram in controller and explain it



我的 controller 只有四個 state，分別為 idle,wr_weight,wr_ans,have_done。Idle 是當 rst 為 high 時用來跟 RAM 讀資料緩衝用的，而當資料有讀入 controller 時要準備開始寫的時候(下一個 cycle)，就會把 next_state=wr_weight；而 wr_weight 就是將讀進來 code book 的資料存入 MAN 的 F/F，這時的控制訊號則是要依據要寫到的格 weight 給對應的 wen 和 MAN_A_weight，而 data_en 則為 low(不能寫進 input_feature)，此外若是寫到最後一個 weight 則 next_state 為 wr_ans，其餘都保持原 state；而當 state 為 wr_ans 時則不可以再動 weight 了(wen=0)，一邊讀 data_in 進 input_feature(data_en)，一邊將答案寫入到對應的 RAM2 裡，也是一樣寫到最後一個的時候 next_state 要設為 have_done；have_done 的 state 會用 pipeline 一直傳下去，直到傳回來 controller(state_in)時，用 continuous assign 來輸出真正的 done 為 high。

1) Complete the controller ,MAN, MIN_D, and top module, in the system.

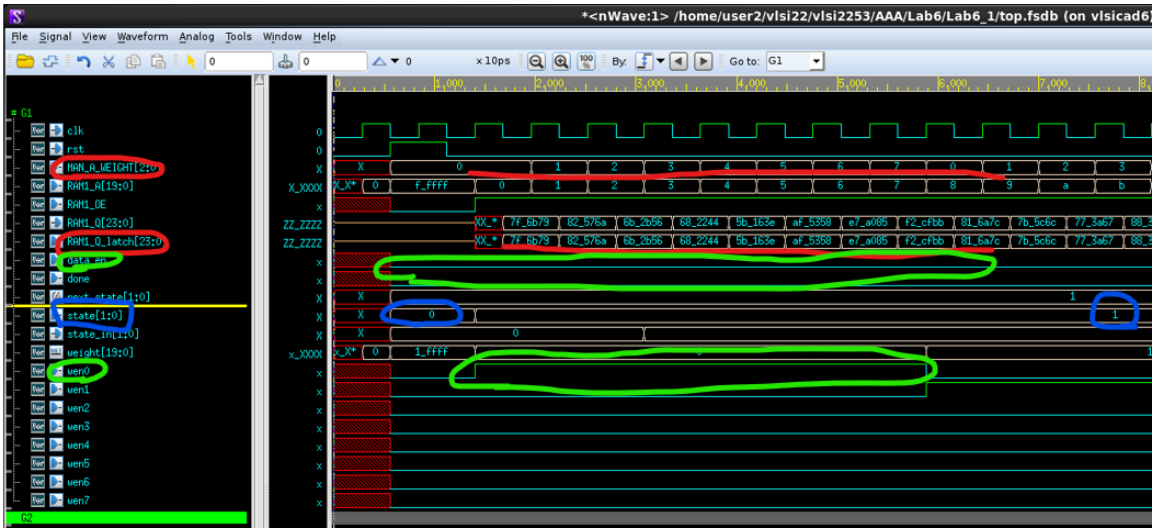
2) Compile the verilog code to verify the operations of this module works properly.

3) Synthesize your *top.v* with following constraint:

- Clock period: no more than 20 ns.
- Don't touch network: clk.
- Wire load model: saed14rvt_ss0p72v125c.
- Synthesized verilog file: *top_syn.v*.
- Timing constraint file: *top_syn.sdf*.

4) Please **attach your waveforms** and **specify your operations** on the waveforms.

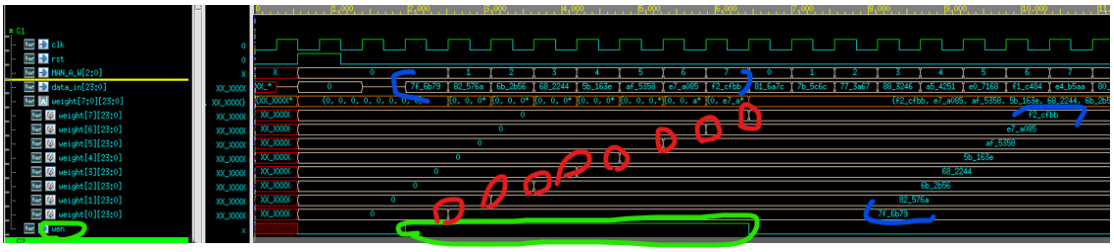
Controller:



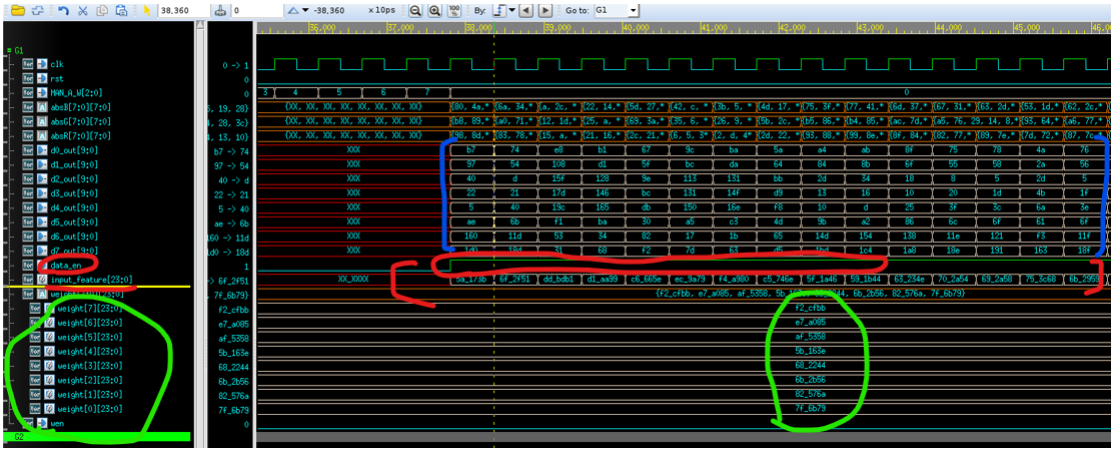
可看到 controller 內的 state(藍筆所畫)在 rst 為 high 時為 idle 狀態(0)，而下一個 state 即可進入 wr_weight(1)，並將要寫入 weight 的資訊(紅筆所畫)如 RAM1_Q_latch 即 MAN_A_WEIGHT 傳下去，而控制可不可以寫入 MAN 的 enable 訊號(綠筆所畫)為 data_en 和 wen，因為還沒要傳 input_feature 所以 data_en 為 low，而 wen0 要寫 0~7 的 weight 所以為 high。

而當寫完 64 個 weight 時，state (藍筆所畫)會由 1➔2 wr_ans。這時的 enable 控制訊號(綠筆所畫)wen 必須不能再寫了所以為 low，而 data_en 因為要開始寫入 input_feature(紅筆所畫)所以要為 high。而當算出 tag 要寫回去 RAM2 時的地址必須透或從 controller 一路傳下去的 address(如紫筆所畫)，不能直接由當前 controller 所發送的位置，因為為 piperline 架構，所以會慢 3 個 cycle 才輸出。

MAN:

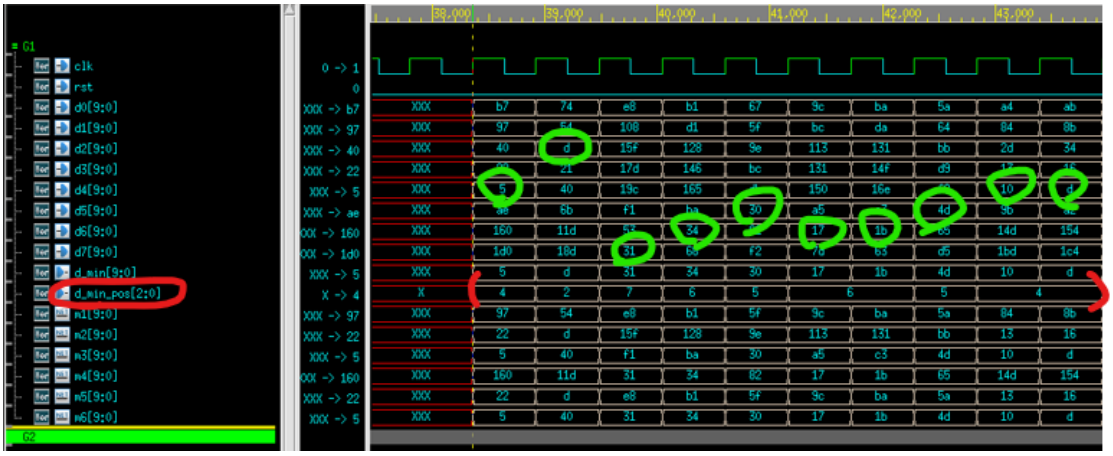


可看到當 controller 送入 man0 綠色的 wen 為 high 時，可將藍色的 weight 寫入到 man0 的 F/F 裡，並依造 MAN_A_W 的位置寫入該對應的位置。



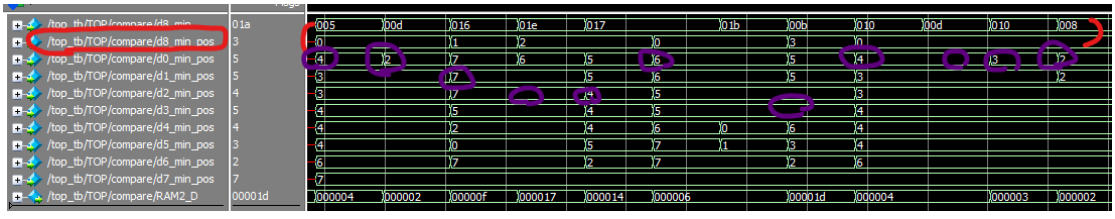
而當 controller 輸入的綠色 wen 為 low 時，存在 MAN 裡的 weight F/F 就不能再變更了。而紅色的 data_en 為 high 時就可以將 data_in 寫到 input_feature，並送入 combinational 計算出藍色的 d0~d7。

MIN:



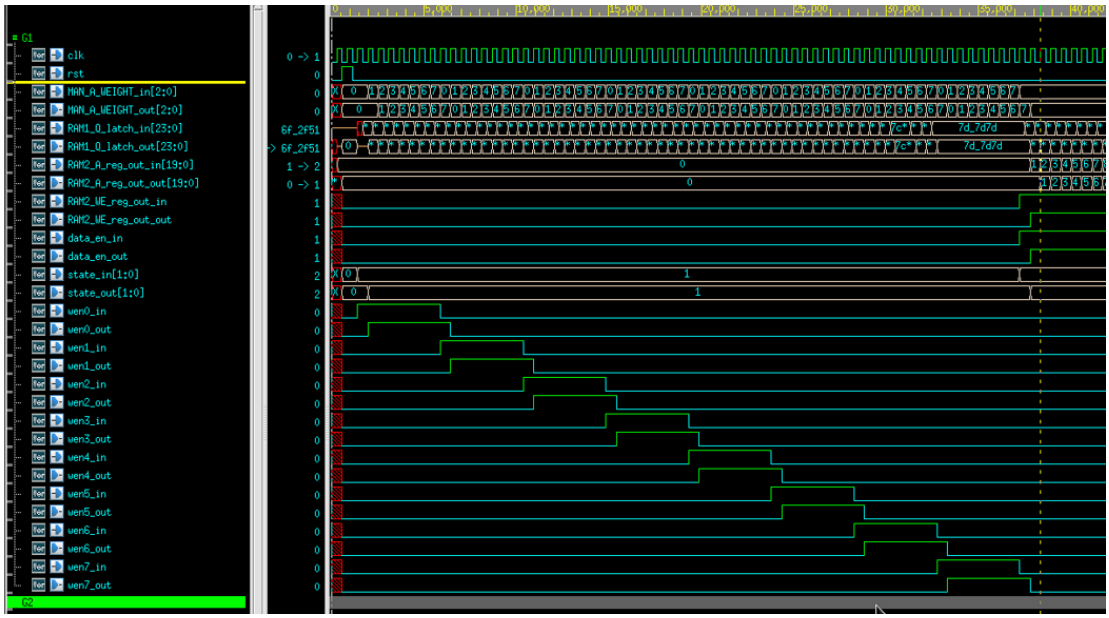
MIN 的功能就是找出 8 個裡面最小的，並輸出他的 distance 及 pos。

Compare:



經由 d8_min_pos 算出八個 MIN_D 送過來的最小值哪一個才是最最小值，並記錄下他是哪一個 MIN，再經由從八個 MIN 傳入到 compare 內的原本位置計算出是 64 個中哪一個 tag。如紫筆所畫，前兩筆因為 d8_min_pos 都是 0，所以 RAM_D 正為 4、2，而第三筆因為 d8_min_pos 位置為 1，所以 7 還要在加上 8 得到 f。

Register:



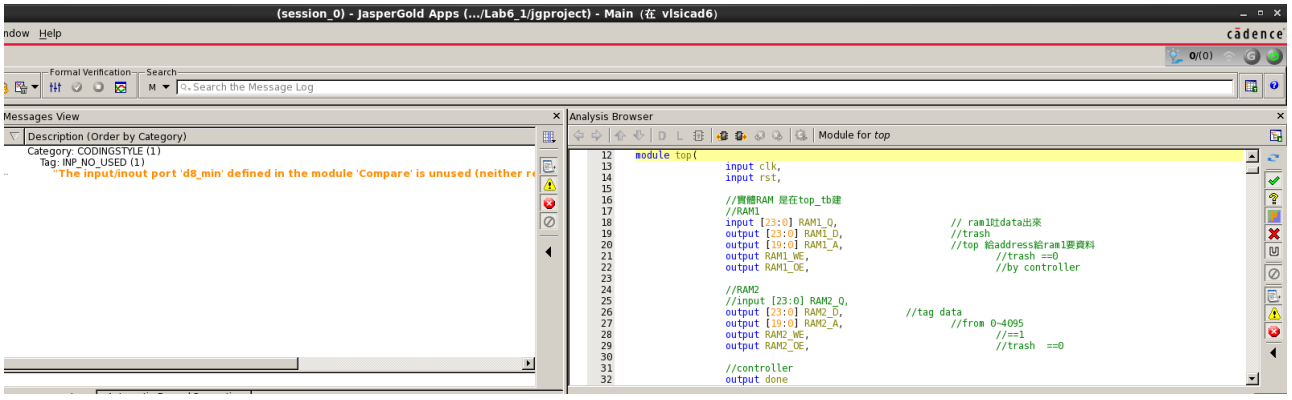
4 個 stage 內 register 的功用都是將前一級算出來的 ouput 站存起來，好讓下一個 cycle 再輸出出去，就可以降低最長路徑所需花費的時間。

5) Show SuperLint coverage (top.v)

```
[<embedded>] % wc -l top.v MAN.v controller.v MIN_D.v Compare.v
  621 top.v
  342 MAN.v
  316 controller.v
   36 MIN_D.v
   60 Compare.v
 1375 total

[<embedded>] %
```

Console | Warnings / Errors | Proof Messages



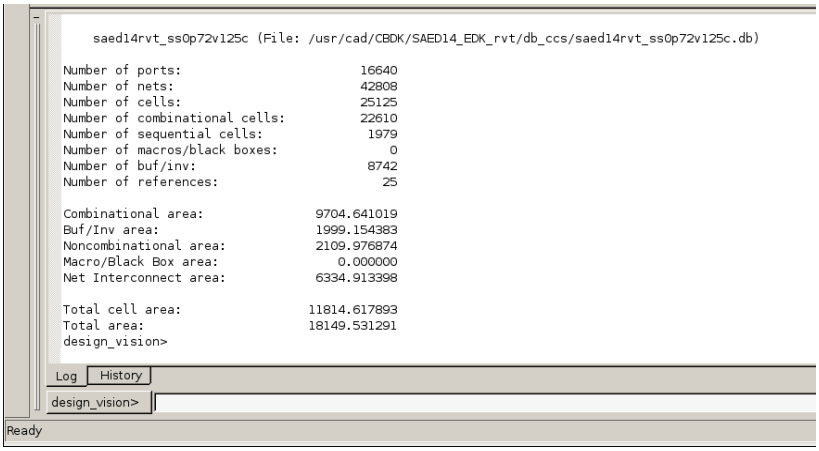
Coverage=99.93%(ram ,reg_controller , reg_man, reg_min 行數沒算到)

6) Your clock period, total cell area, post simulation time (top.v)

Clock period: 2.6ns

Total cell area: 11814

Post simulation time: 10832ns



```
    saed14rvt_ss0p72v125c (File: /usr/cad/CBDK/SAED14_EDK_rvt/db_ccs/saed14rvt_ss0p72v125c.db)

Number of ports:                        16640
Number of nets:                         42808
Number of cells:                        25125
Number of combinational cells:          22610
Number of sequential cells:              1979
Number of macros/black boxes:               0
Number of buf/inv:                       8742
Number of references:                      25

Combinational area:               9704.641019
Buf/Inv area:                     1999.154383
Noncombinational area:            2109.976874
Macro/Black Box area:                0.000000
Net Interconnect area:            6334.913398

Total cell area:                 11814.617893
Total area:                      18149.531291
design_vision>
```

Log History

design_vision>

Ready

```
RAM2[4083] = 07, pass
RAM2[4084] = 06, pass
RAM2[4085] = 0c, pass
RAM2[4086] = 05, pass
RAM2[4087] = 14, pass
RAM2[4088] = 1c, pass
RAM2[4089] = 14, pass
RAM2[4090] = 14, pass
RAM2[4091] = 14, pass
RAM2[4092] = 14, pass
RAM2[4093] = 14, pass
RAM2[4094] = 14, pass
RAM2[4095] = 1d, pass
```

```
        ****************************
        **                        **       |\__||
        **  Congratulations !!    **      / O.O  |
        **                        **    /_____   |
        **  Simulation PASS!!     **   /^ ^ ^ \  |
        **                        **  |^ ^ ^ ^ |w|
        ****************************   \m___m__|_|
```

```
Simulation complete via $finish(1) at time 10832900 PS + 3
./top_tb2.v:177                 $finish;
xcelium> exit
TOOL:    xmverilog       20.09-s007: Exiting on Apr 15, 2022 at 21:25:09 CST  (total: 00:01:19)
vlsicad6:/home/user2/vlsi22/vlsi2253/AAA/Lab6/Lab6_1 %
```

7) Please describe how you optimize your design when you run into problems in synthesis .ex: plug in some registers between two instances to shorten your datapath, resource sharing for some registers to reduce your cell area.

Timing:

為了避免用 single cycle 做完所有的事情而導致 data path 過長，我採用的是 pipeline 的做法，平均分配每一個 cyle 內每個 stage 所要做的事，就不會發生執行一個 module 時期他 module 都在閒置的情況，指犧牲了一開始的 3 個 cycle，而可以使 period 押的很低，大幅提升效率。

Area:

我在面積的部分只採用了能省則省的策略，像是有些位置其實根本不用到 20bits，用 8bits 就可以表示所有的位置了。此外也會共用一些 wire 或 reg，因為 state 的更換後某些線就不會用到了，就可以拿來傳這個 state 會用到的

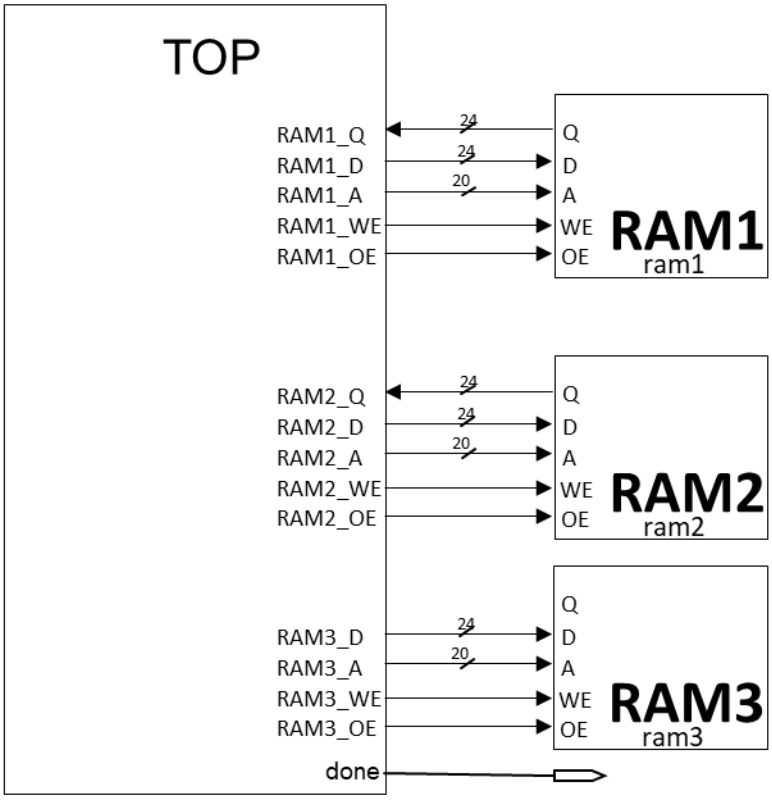
資料了。

- Lessons learned from this lab

  這次的實驗才讓我知道原來之前寫的 code 都是垃圾，速度慢面積又大，而藉由每個人都在互相比較誰的面積多小，時間多小後，就有了優化自己 design 的競賽，彼此就會炫耀我少了幾個 reg，我插了幾個 stage，我 period 變多小之類的。因此，日後再寫 verilog 時不能再以方便來寫，要一邊想合成後的電路長怎樣，並且盡量使面積縮小。

Lab 6_2: decompress the image

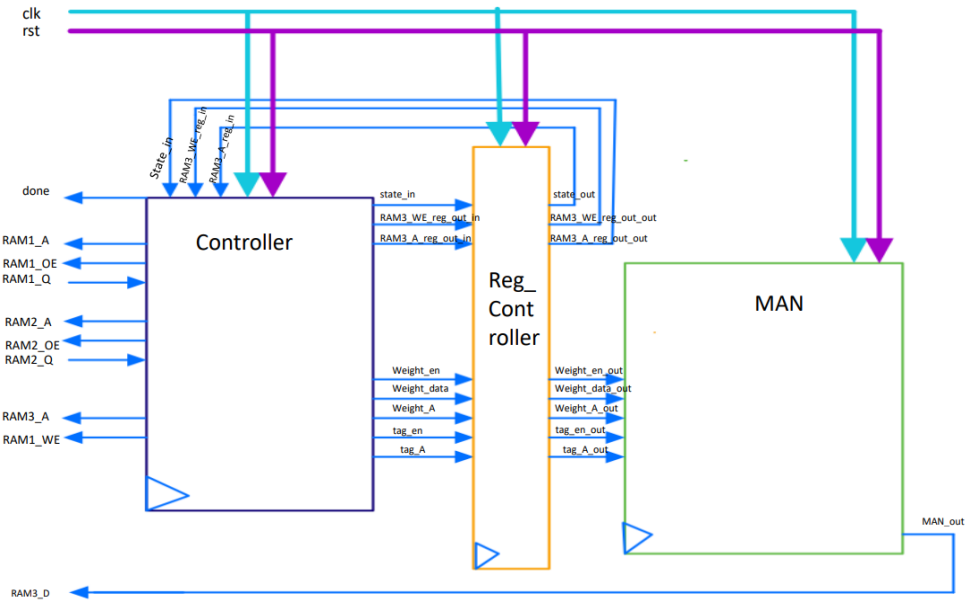
We can use code book and data tags to decompress the image



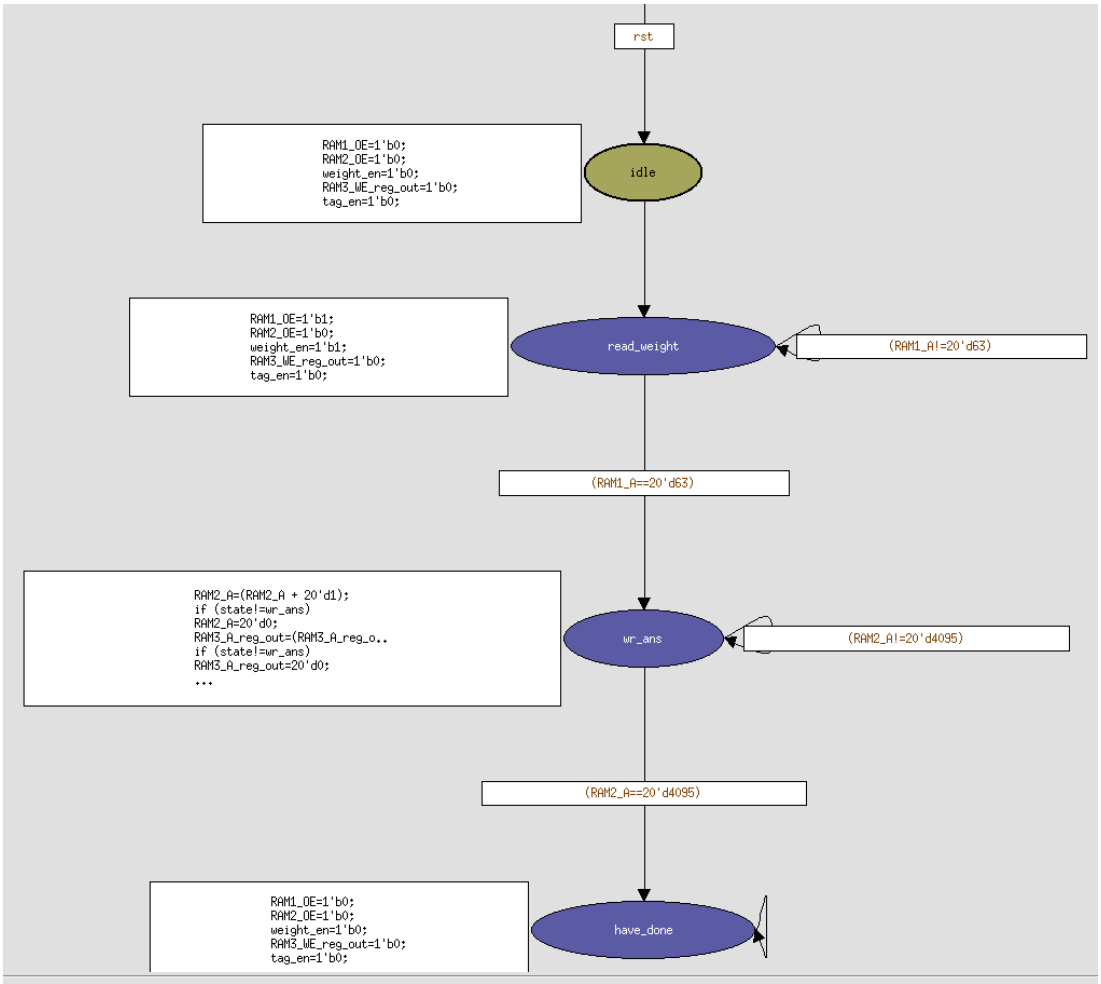
▲Block diagram for architecture(external)

- Control signal
  - done: Stop the process if you decompress all pixels in a figure

- Draw your state diagram and explain your design. You can draw internal architecture to describe your design



Top module 如上圖所示，一樣是用 pipeline 來完成，controller 左端一樣是 top 的緩衝區；上端則是輸入到 RAM3 的控制訊號及 done 的判斷訊號；右端則是控制 MAN 可不可以寫入 weight 或 tag 的控制訊號及 data。而 MAN 裡面會有 FlipFlop 存放 64 筆 weight 資料，並將輸入進來的 tag 當作 index 去裡面取資料並輸出。



第二題的 FSM 我也是設計 4 個 state，跟上一題很像。Idle 一樣是為了 rst 訊號會停滯一個 cycle 的 state；而 read_weight 也是將 codebook 的 weight 寫入 MAN 裡；而 wr_ans 則是依據 RAM2 裡 tag 的位置去 MAN 取出該 weight 並輸出到 RAM3；而 have_done 一樣經由 pipeline 傳回 controller 來判斷，當 state_in==have_done 則輸出 done。

8) Complete the top module, in the system.

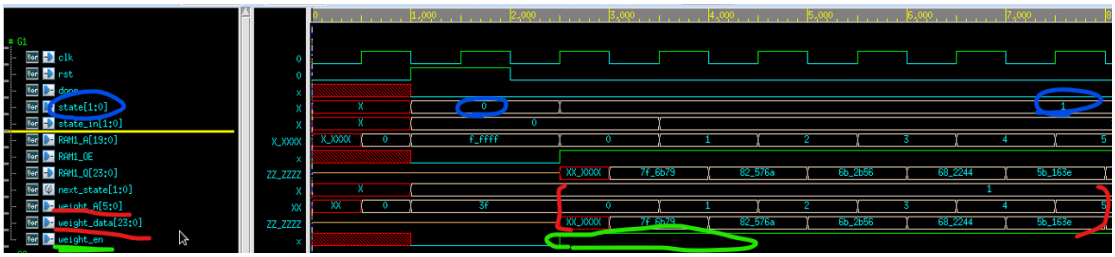
9) Compile the verilog code to verify the operations of this module works properly.

10) Synthesize your *top.v* with following constraint:

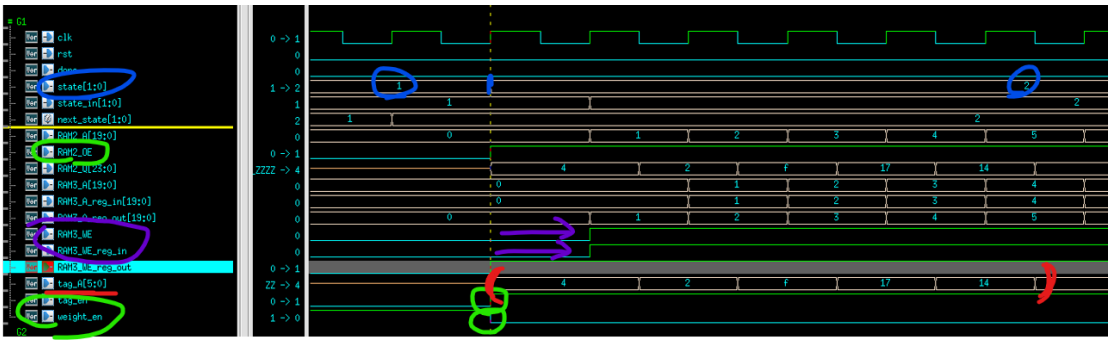
- Clock period: no more than 10 ns.
- Don't touch network: clk.
- Wire load model: saed14rvt_ss0p72v125c.
- Synthesized verilog file: *top_syn.v*.
- Timing constraint file: *top_syn.sdf*.

11) Please **attach your waveforms** and **specify your operations** on the waveforms.

Controller:

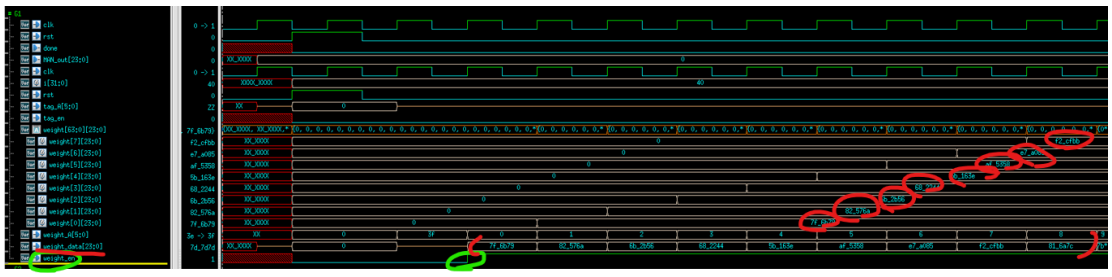


一樣當 rst 為 high 時 state(藍筆)設為 idle(=0)，而 idle 完下一個 state 則是 read_weight，開始傳要寫的 weight 及要寫道的位置下去(如紅筆所畫)，而 enable 訊號(綠筆所畫)要設為 high。

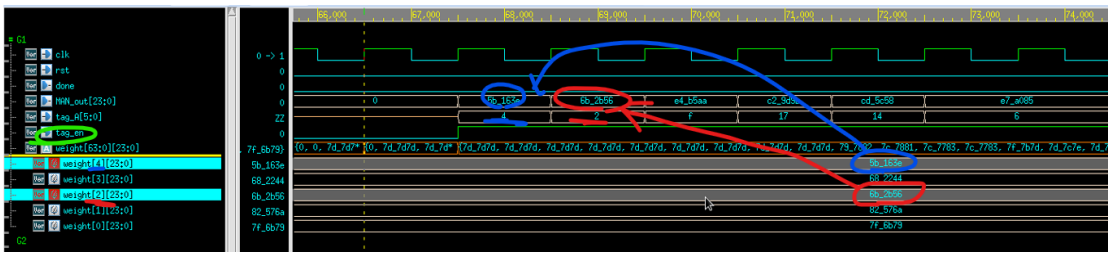


當寫完 64 個 weight 時，state(藍筆所畫)會變成 we_ans(2)。而綠色的 enable 訊號會將要讀 tag 的打開，而要寫 weight 都關上。並將從 RAM2 讀進來紅色的 tag_A 傳下去給 MAN。並且因為是 pipeline 架構，所以要寫回 RAM3 的 WE 跟 A 都要延遲一個 cycle 才能傳入 controller(如紫筆所畫)。

MAN:

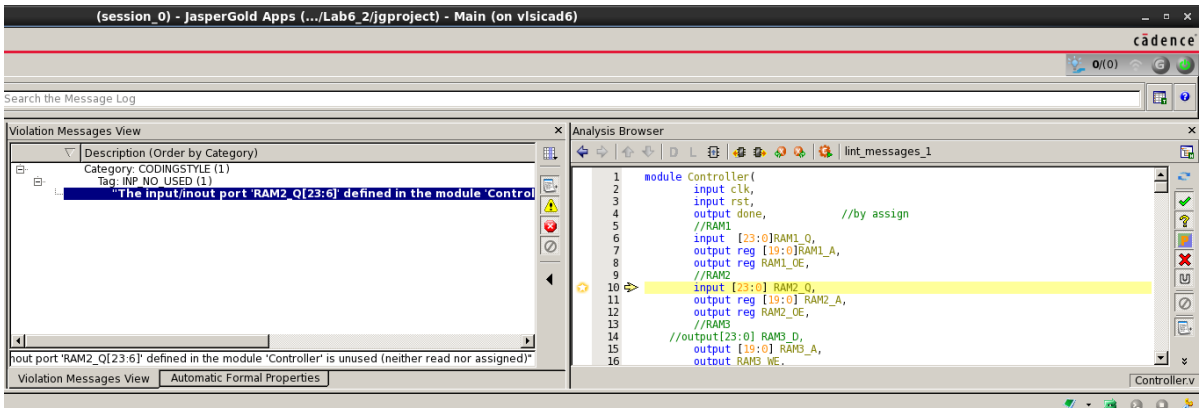


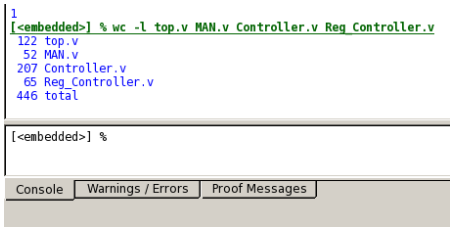
可看到當綠色的 weight_en 為 high 時可以將送入的 weight_data 依照 weight_A 寫入 MAN 的 F/F 裡。



可看到 MAN 會依照收到的 tag_A 去 F/F 裡拿出該位置的 weight 並送出到 MAN_out。

12) Show SuperLint coverage (top.v)





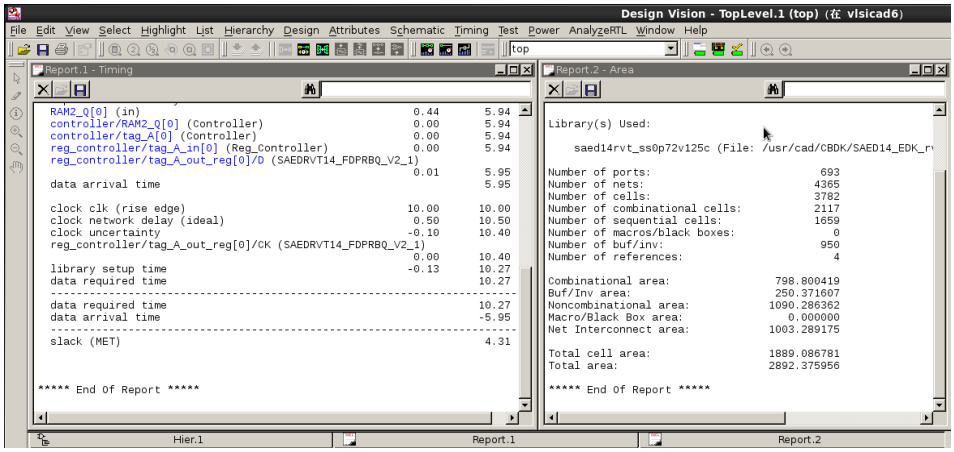
Coverage=99.9%(ram reg_controller 行數沒算到)

13) Your clock period, total cell area, post simulation time (top.v)

clock period：10ns

total cell area：1889

post simulation time: 41645ns




```
RAM3[4073] = e38474, pass
RAM3[4074] = e38474, pass
RAM3[4075] = e07168, pass
RAM3[4076] = e38474, pass
RAM3[4077] = e38474, pass
RAM3[4078] = e07168, pass
RAM3[4079] = e07168, pass
RAM3[4080] = e38474, pass
RAM3[4081] = e38474, pass
RAM3[4082] = e7a085, pass
RAM3[4083] = f2cfbb, pass
RAM3[4084] = e7a085, pass
RAM3[4085] = a54251, pass
RAM3[4086] = af5358, pass
RAM3[4087] = cd5c58, pass
RAM3[4088] = ab6266, pass
RAM3[4089] = cd5c58, pass
RAM3[4090] = cd5c58, pass
RAM3[4091] = cd5c58, pass
RAM3[4092] = cd5c58, pass
RAM3[4093] = cd5c58, pass
RAM3[4094] = cd5c58, pass
RAM3[4095] = c87c6e, pass


        ***************************
        **                       **       |__||
        **  Congratulations !!   **      / O.O  |
        **                       **    /_____   |
        **  Simulation PASS!!    **   /^ ^ ^ \  |
        **                       **  |^ ^ ^ ^ |w|
        ***************************   \m___m__|_|

Simulation complete via $finish(1) at time 41645 NS + 3
./top_tb3.v:138                 $finish;
xcelium> exit
TOOL:   xmverilog       20.09-s007: Exiting on Apr 12, 2022
vlsicad6:/home/user2/vlsi22/vlsi2253/AAA/Lab6/Lab6_2 %
```

- Lessons learned from this lab

  我在這次 lab 合成後會出現每一個 weight 都會順延一個 weight address，經過助教的講解後，我才知道我前幾個 lab 解決 enable 的 else 的 latch 的解法是錯的，我原本用將 always@(*){}內的 =改成<= superlint 就過了，而這樣子的行為其實是 combinational，而非 F/F。所以沒辦法存前一個 cycle 的值，所以改成 posedge clk 就可以變成 F/F 了。

Please compress all the following files into one compressed file (".tar " format) and submit through Moodle website:

※ NOTE:

1. If there are other files used in your design, please attach the files too and make sure they're properly included.
2. Simulation command

Problem	Command
Lab6_1(pre)	ncverilog top_tb2.v +access+r +define+FSDB
Lab6_1(post)	ncverilog top_tb2.v +access+r +define+FSDB+syn
Lab6_2(pre)	ncverilog top_tb3.v +access+r +define+FSDB
Lab6_2(post)	ncverilog top_tb3.v +access+r +define+FSDB+syn