

National Cheng Kung University

Department of Electrical Engineering

Introduction to VLSI CAD (Spring 2022)

Lab Session 7

SOM Processing System

Name	Student ID	
郭家佑	F64096114	
Practical	Points	Marks
Lab 7_1	45	
Lab 7_2	45	
Demo	10	
Notes		

Due: 15:00 April 27, 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.

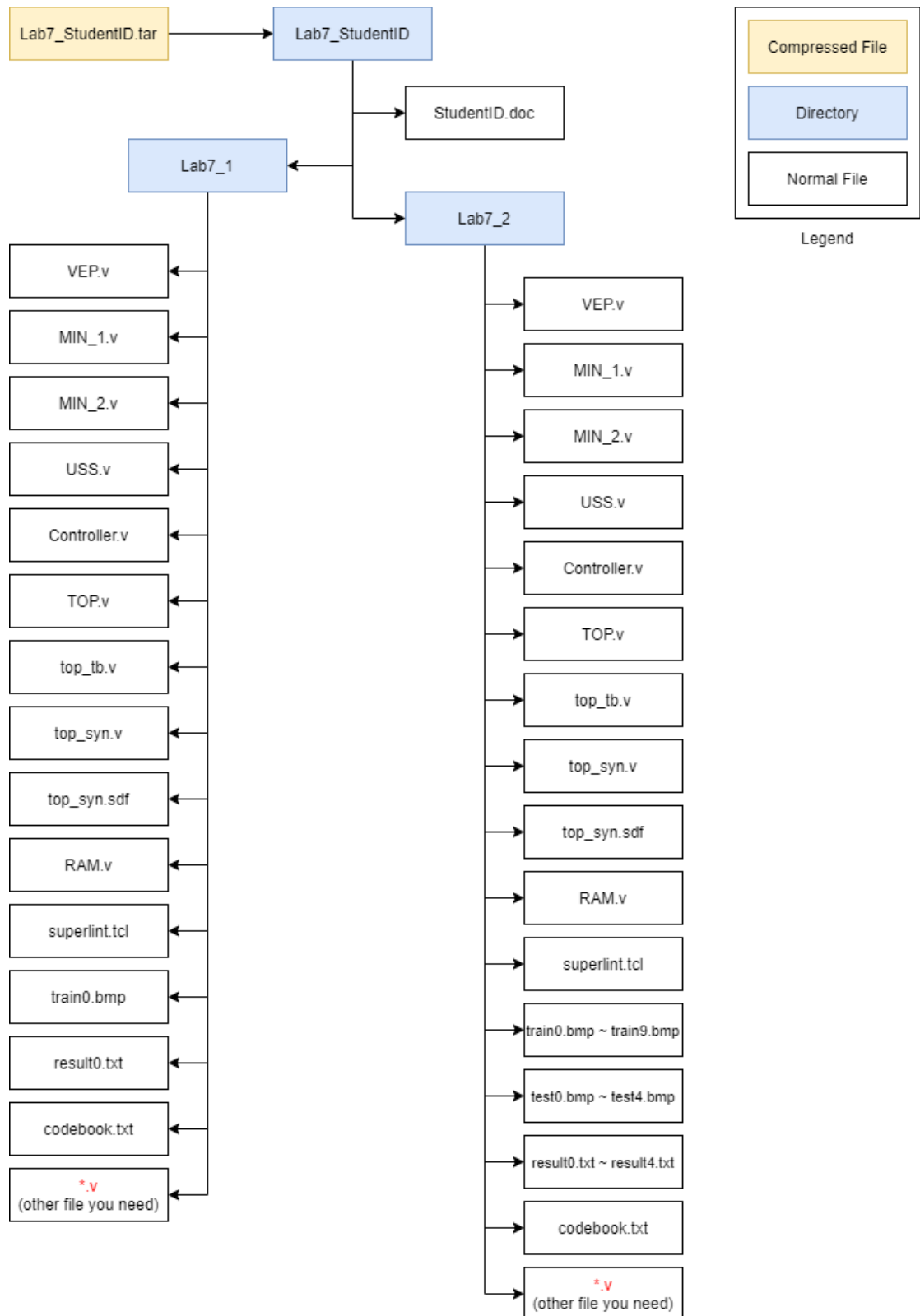
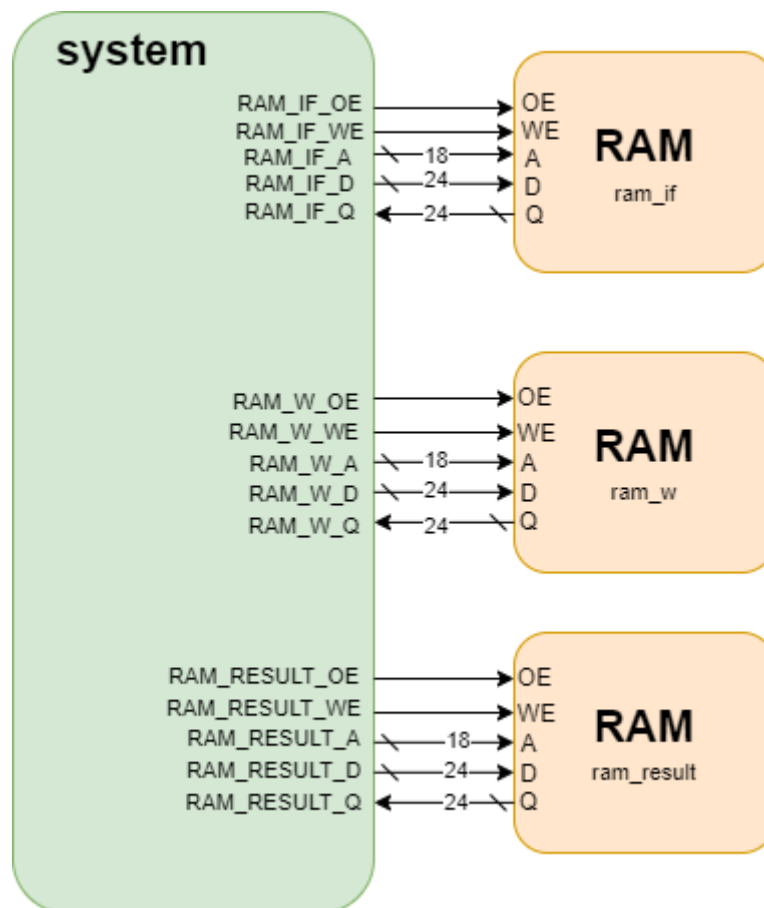
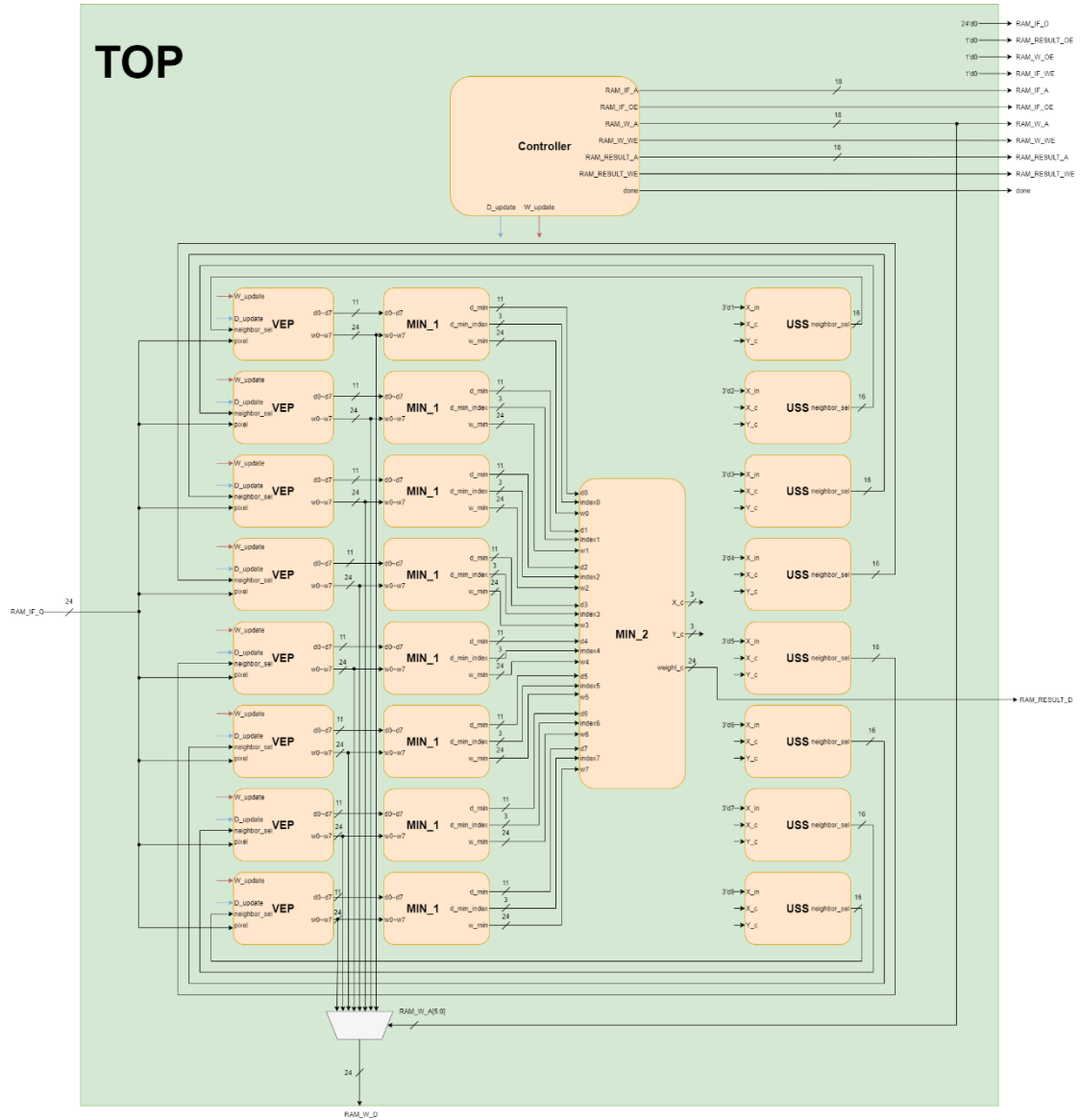


Fig.1 File hierarchy for Homework submission

You are about to integrate all components (VEP, MIN_1, MIN_2, USS, Controller...) to form a SOM processing system. 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
<u>clk</u>	Input	1	Clock
<u>rst</u>	Input	1	Reset signal, active high
<u>D_update</u>	Output	1	Distance update enable, active high
<u>W_update</u>	Output	1	Weight update enable, active high
RAM_IF_A	Output	18	Input feature RAM address
RAM_IF_OE	Output	1	Input feature RAM output enable
RAM_W_A	Output	18	Weight RAM address
RAM_W_WE	Output	1	Weight RAM write enable
RAM_RESULT_A	Output	18	Result RAM address
RAM_RESULT_WE	Output	1	Result RAM write enable
done	Output	1	Pull to 1 if the system is done

➤ **VEP**

Signal	I/O	Bit	Description
<u>clk</u>	Input	1	Clock
<u>rst</u>	Input	1	Reset signal, active high
<u>W_update</u>	Input	1	Weight update enable, active high
<u>D_update</u>	Input	1	Distance update enable, active high
<u>neighbor_sel</u>	Input	16(2x8)	Neighborhood function of 8 VEP weights (00=>1, 01=>0.25, 10=>0.125, 11=>0)
pixel	Input	24	Input pixel from <u>RAM_if</u>
d0~d7	Output	11	Manhattan distance between 8 weights
w0~w7	Output	24	8 weights

➤ **MIN_1**

Signal	I/O	bit	Description
<u>clk</u>	Input	1	Clock
<u>rst</u>	Input	1	Reset signal, active high
<u>d0~d7</u>	Input	11	Manhattan distance between 8 weights
<u>w0~w7</u>	Input	24	8 weights
<u>d_min</u>	Output	11	Minimum distance between d0~d7
<u>d_min_index</u>	Output	3	Index of minimum distance
<u>W_min</u>	output	24	Weight of minimum distance

➤ **MIN_2**

signal	I/O	bit	Description
<u>clk</u>	Input	1	clock
<u>rst</u>	Input	1	Reset signal, active high
<u>d0~d7</u>	Input	11	Minimum distance from MIN_1
<u>w0~w7</u>	Input	24	Weight of minimum distance from MIN_1
<u>index0~index7</u>	Input	3	Index of minimum distance from MIN_1
<u>X_c</u>	Output	3	The X coordinate of center weight
<u>Y_c</u>	Output	3	The Y coordinate of center weight
<u>weight_c</u>	output	24	Center weight

➤ **USS**

signal	I/O	bit	Description
<u>clk</u>	input	1	Clock
<u>rst</u>	Input	1	Reset signal, active high
<u>X_in</u>	Input	3	USS module index
<u>X_c</u>	Input	3	The X coordinate of center weight
<u>Y_c</u>	Input	3	The Y coordinate of center weight
<u>neighbor_sel</u>	Output	16(2x8)	Neighborhood function of 8 VEP weights (00=>1, 01=>0.25, 10=>0.125, 11=>0)

➤ Top

Signal	I/O	bit	Description
<u>clk</u>	Input	1	Clock
<u>rst</u>	Input	1	Reset signal, active high
RAM_IF_Q RAM_W_Q RAM_RESULT_Q	Input	24	Data output from RAM
RAM_IF_OE RAM_W_OE RAM_RESULT_OE	Output	1	RAM output enable signal
RAM_IF_WE RAM_W_WE RAM_RESULT_WE	Output	1	RAM write enable signal
RAM_IF_A RAM_W_A RAM_RESULT_A	Output	18	RAM address
RAM_IF_D RAM_W_D RAM_RESULT_D	output	24	Data written into RAM
done	Output	1	Pull to 1 if the system is done

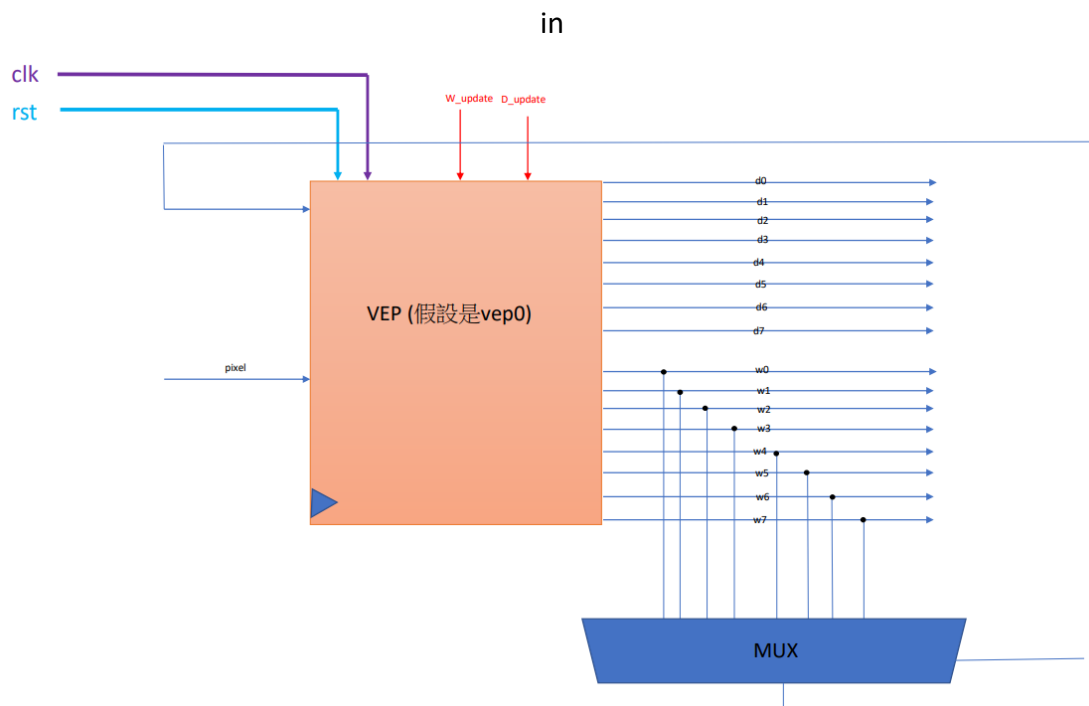
➤ Understanding the function:

Once system is initialized, it

- a) Read input pixel from RAM_if
- b) Calculate Manhattan distance and find the minimum distance
- c) Update the weight memory
- d) Repeats the process step(a)~(c) until the last pixel of RAM_if is read
- e) writes the trained codebook to the RAM_w
- f) read input pixel from RAM_if and inference the picture
- g) writes the lossy compression picture to the RAM_result
- h) repeats the process step (f)~(g) until the last pixel of RAM_result is written;
- i) flags "done" when system is completed

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

■ VEP



1. 在 training weight 時，W_update 會一直為 1(可更新 weight)，VEP 會將 input 進來的 pixel 先經過一個 pixel_latch 存起來，因為 RAM 給的時候已經是在負緣了，剩下的時間已經很少了，若繼續計算會導致 clk period 便很長。接著利用 pixel_latch 算出與 VEP 內 64 個 weight(初始值為)距離，並將這個距離 output，以及將 VEP 內部的 weight 傳下去(這部分其實在 training 時可以忽略)，經過後面 module 運算後，可由 USS module 傳來每一個 weight 的 neighbor_sel，進而依造每個要更新的權重(1. 傳來的 neighbor 00 01 10 11 分別用 continuous assign 來判斷要 shift 量 2. input 與 weight 的減法在之前算中心距離就算過一遍可以直接拿來用) 並在下一個 cycle 更新 weight。

2. 在 w_weight 階段時，W_update 會變為 0(即不可以更動 weight 了)，控制訊號的 RAM_W_A 會 fanout 一條 wire 來給 mux 做選擇訊號，在依照選擇訊號將 weight 一個一個寫入到 RAM_W 內。

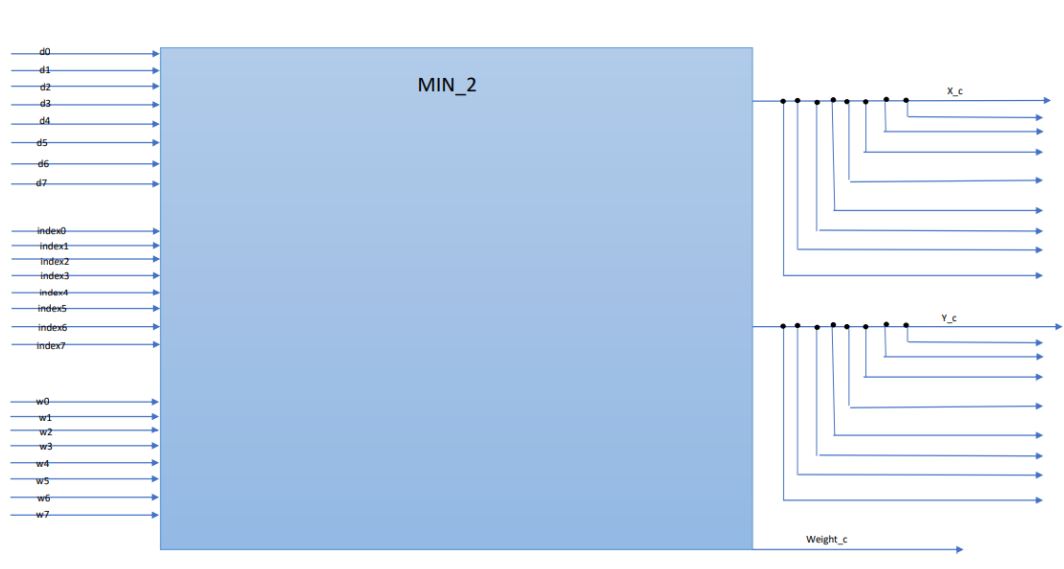
3. 在 w_pic 階段，W_update 會一樣為 0(即不可以更動 weight 了)，這次我就不用 LAB6_2 的設計 pipeline 回來 MAN 在 output 出去(也就是說 weight 都用 F/F 存在 VEP 內部，不會 output 出來)，而我這次的做法是將其 weight output 下去，選擇 8 個最小的就繼續傳下去，直到 resultt，因為一個 cycle 的時間已經被 VEP 拖很久了，所以我不想要在回去 VEP 內輸出，這樣完成就要兩個 cycle 會拖更久。

■ MIN_1



MIN_1 設計就簡單許多，跟 lab6 很像，只是這次多一個 output 要將 d_min_pos 的 weight 傳下去給 w_pic 用。

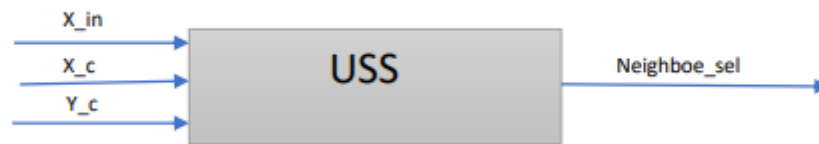
■ MIN_2



而 MIN_2 跟 MIN_1 功能很像，都是在找最短距離的位置，只是就直接將前一級的 Y 座標一起傳入，就不用像我 Lab6 一樣再多寫一個 compare 來比較位置。而輸出的部分在 w_weight 時會將 d0~d7 最小的 XY 座標輸出到 X_c、Y_c

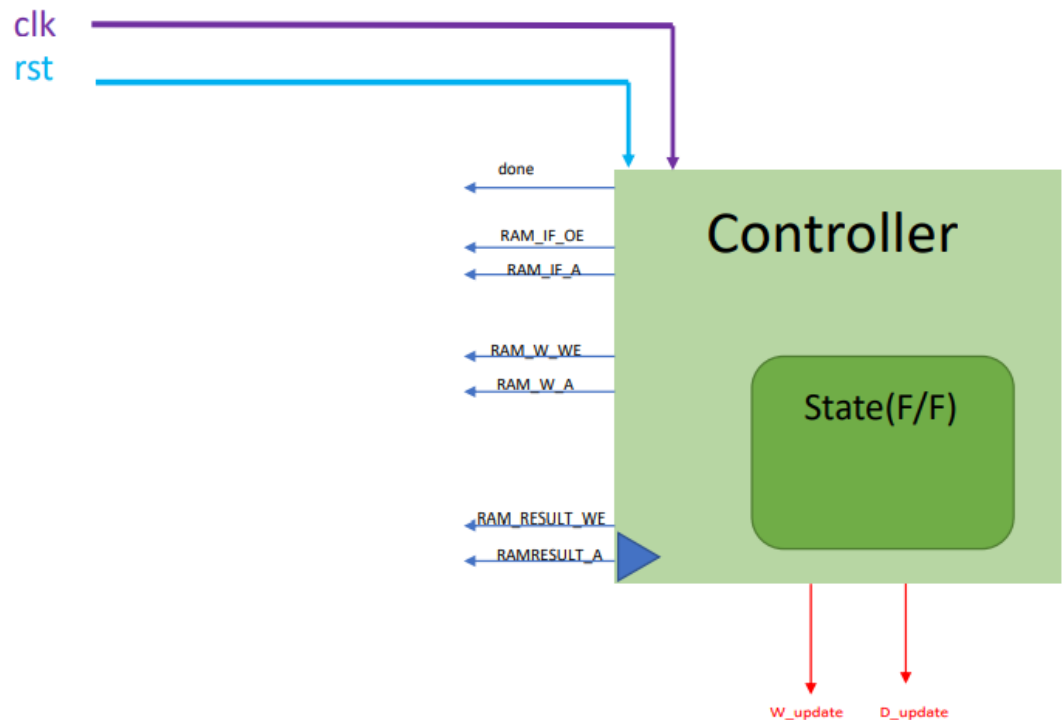
內，供給 USS module 內做使用；而當 w_{pic} 時， $weight_c$ 則會依照算出的 X_c 去找 $w_0 \sim w_7$ 哪一個 $weight$ 是最小的，並輸出到 RAM_RESULT 中。

■ USS



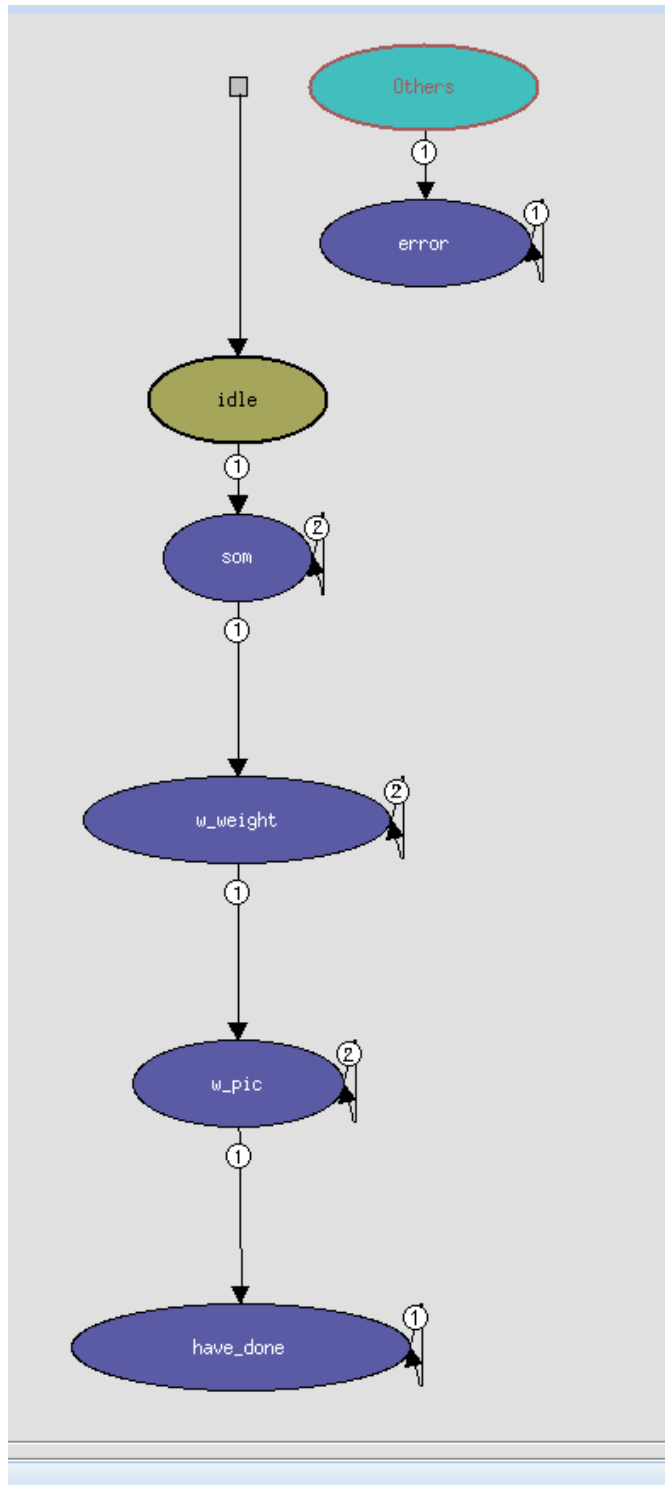
USS module 則時會在 training 時起作用，利用輸入進來的 X_c 、 Y_c 與自身的 X 座標 X_in 來判斷有沒有在 25 宮格內，我是先用自身 X 座標來過濾一層，在依照每一個要輸出的 Y 座標去跟 Y_c 在過濾一層，就可以得到 Neighbor_sel。

■ Controller



Controller module 內部主要有分成三個部分，一個是根據寫入 RAM Address 來更新 state；另一部分是讓用 counter 來更新每次要去 RAM 寫或讀的位置；第三部分是根據 state 來 output W_update 、 D_update 等 enable 訊號。

◆ Draw your state diagram in controller and explain it



當 rst 為 positive trigger 時，state 更新到 idel stae，而下一個 cycle 則直接 shift 到 som state，開始依照 RAM_IF 從-1 開始讀 input pixel 來做 training，當 RAM_IF_A 讀取到最尾端時，則下一個 state 切換到 w_weight，作法也跟前面一樣開始算要寫的位置，直到 63 則跳到下一個 w_pic state，當 w_pic state 的 Address 讀到尾巴時則下一個 cycle shift 到 have_done state，並結束整個 simulation。

1) Complete the Controller, VEP, MIN_1, MIN_2, USS, and TOP module, in the system.

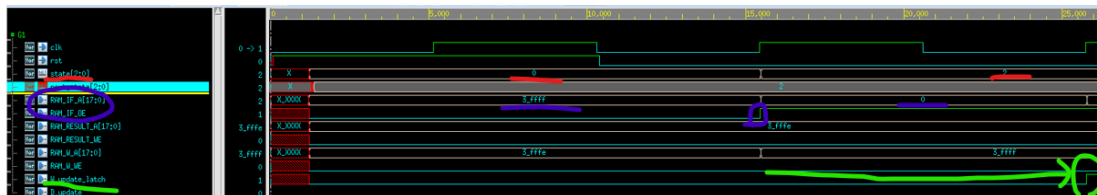
2) Compile the verilog code to verify the operations of this module works properly.

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

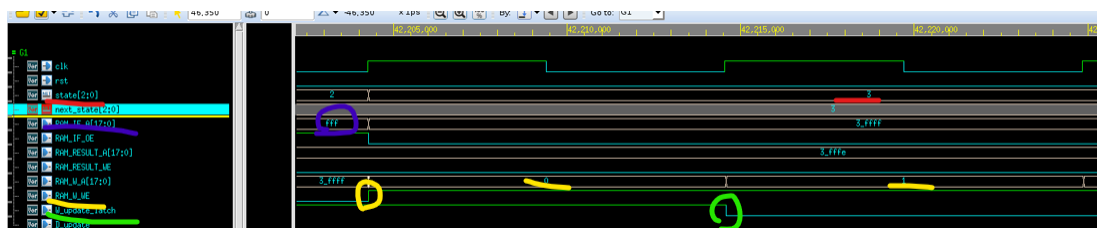
- Clock period: no more than **20 ns**.
- Don't touch network: clk.
- Wire load model: saed14rvt_ss0p72v125c.
- Synthesized verilog file: *top_syn.v*.
- Timing constraint file: *top_syn.sdf*.

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

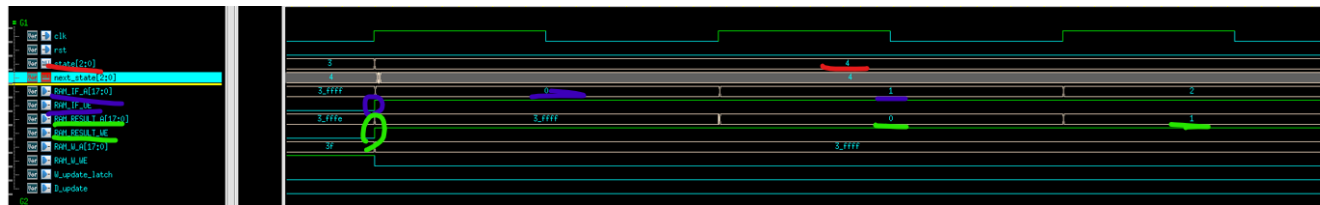
Controller:



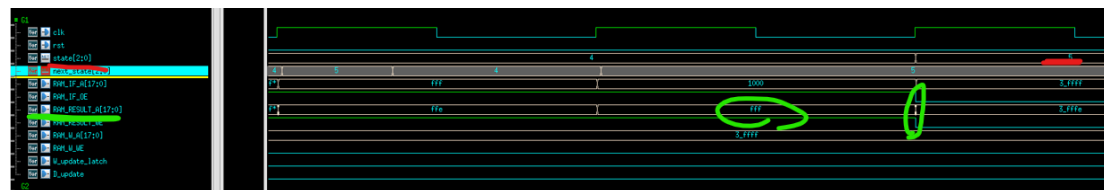
可看到當 rst trigger 時，state 會初始到 idle state，而下一個 cycle 則會 shift 到 som state 算 weight(橘色部分)。而當 state 為 som 時，RAM_IF_OE 為 1 則可從記憶體內讀取資料(紫筆所畫)，並將 pixel 送入八個 VEP 中進行 training。而綠色的 W_update_latch 是因為我有將 pixel 用 pixel_latch 存下來，方便下一個 cycle 有更久的時間做運算，所以 W_update 也要順延一個週期在 output 出去。



當 RAM_IF_A 讀到 4095 時(紫筆所畫)，RAM_IF_OE 的訊號會變成 Low，且會將 next_state 設為 w_weight，並在下一個 cycle 將 state 更新成 w_weight(紅筆所畫)。在這個 state，VEP 內部 F/F 存的 weight 就不能更動了，所以 W_update 為 Low(綠筆所畫)。而要將 weight 輸出到 RAM_W 去，WE 必須為 high(黃筆所畫)，而寫入的位置會依入從 0~64。

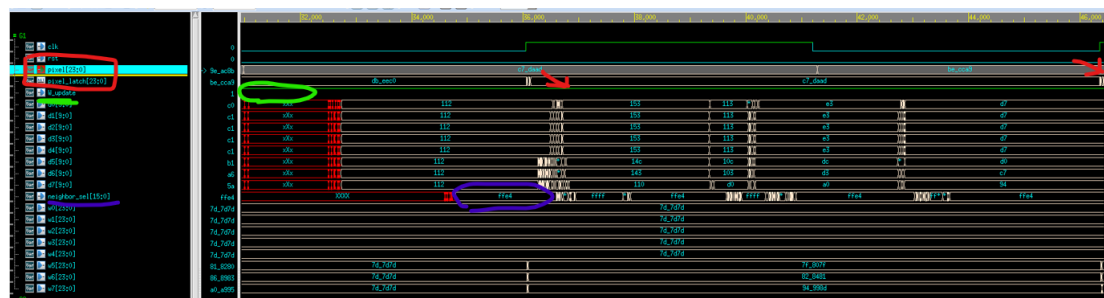


將 64 個 weight 寫入 RAM_W 後，state 就會切換到 w_pic 了(紅筆所畫)。而從 RAM_IF 讀取 pixel 這部分跟前面 som state 的做法是一樣的(紫筆所畫)，但是要將算出來的 tag(inference)寫回 RAM_RESULT 就要比 pixel_latch 延遲一個 cycle 了，因為 RAM 必須在下一個正緣才能寫入(綠筆所畫)。

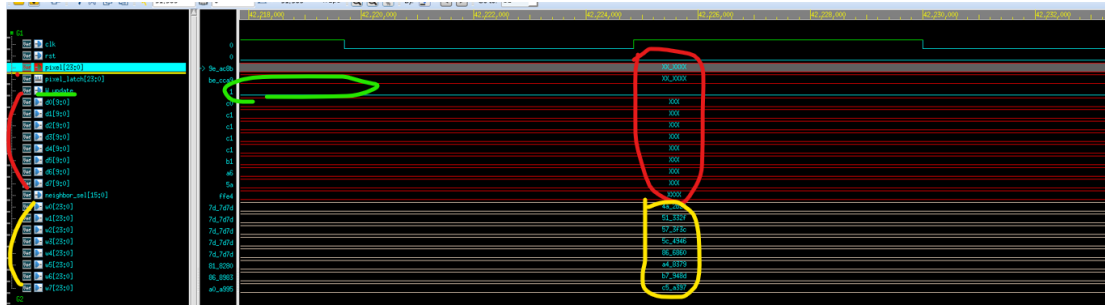


而當 RAM_RESULT_A 寫到最後一個位置時，next_state 必須設為 have_done(紅筆所畫)，並將所有有關 RAM 的 enable 訊號設為 low(綠筆所畫)。

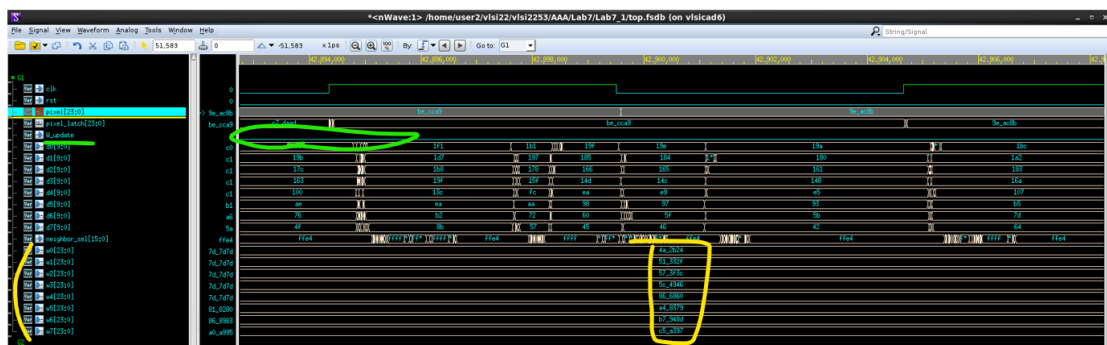
VEP:



可看到在 SOM state 時，紅色的 pixel_latch 會 keep 住 pixel 的值一個 cycle，而由 controller 來綠色的 W_update 在 SOM state 時都為 high。而紫色的 neighbor_sel 因為會經過中經其他 module 的 delay 而比較晚送達。



進入 w_weight 階段時，因為 RAM_IF_OE 的訊號都為 low，即代表資料無法從記憶體讀進來了，所以會造成算短最短距離的紅色部分都為 XX。而 W_update 為 low 的關係(綠筆所畫)，VEP 內存的 weight 不能再更動，所以會看到黃色的 weight 都會保持值。



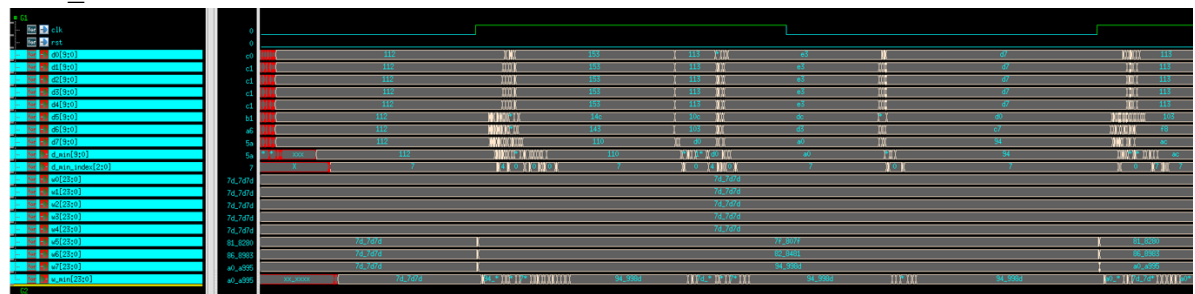
進入 w_pic state 時，VEP 又開始運作了，只是這次 w_update 都為 low，所以 weight 都不能更動，只能依造 input 進來的 pixel 來與已經收斂完的 weight 算曼哈頓距離，直到 input 進來的 4096 個 pixel 算完就結束。

MUX:



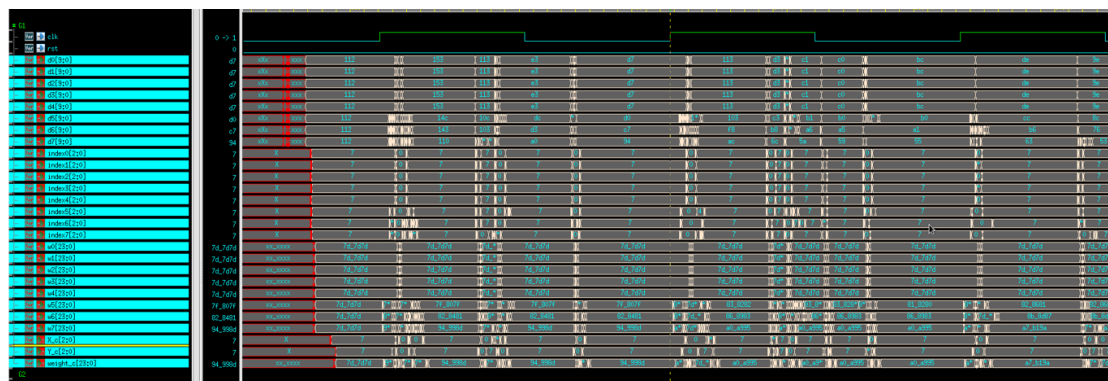
可看到 mux 會根據要寫入 RAM_W_A 的訊號的不同的 sel 訊號來挑選從 VEP 過來的 input weight，並由 o 將其輸出。

MIN_1:

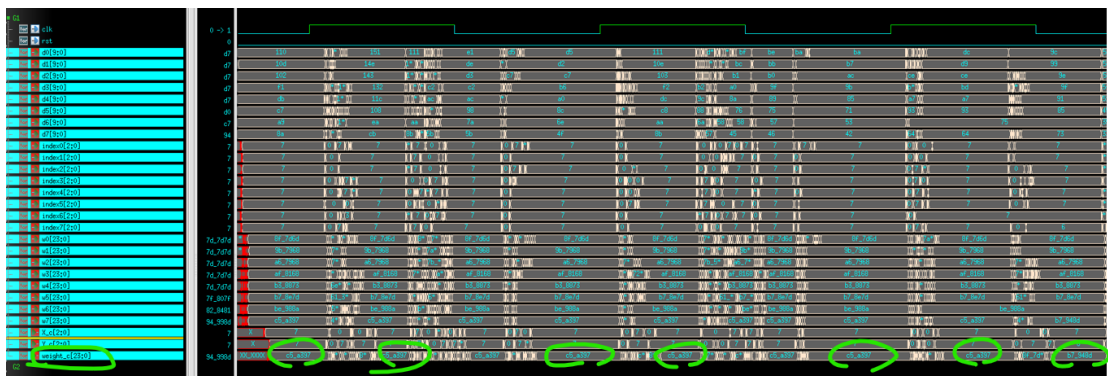


Min_1 的功能很簡單，就是從八個曼哈頓距離選出最小的，並將其 index、距離和所對應的 weight 一起輸出。

MIN_2:

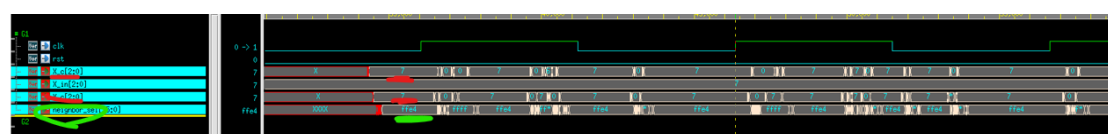


在 som state 中，min_2 的功能比 min_1 複雜一點，要將它是第幾個 vep(X 座標)，和該 vep 最小的曼哈頓距離(Y 座標)透過 X_c、Y_c 輸出。



而在 w_pic state，min_2 又多了一項輸出 inference 的功能，將算出的中心點座標轉換成該對應的 weight，並輸出到 RAM_RESULT 裡。

USS:



Uss module 則會對前面傳來的中心座標做出周圍訊號要更新的權重比例。像是第一筆資料的中心座標為(7,7)，則(7,7) 更新的 neighbor 權重比例為 1，周圍的(6,7)(7,6)(6,6)更新的 neighbor 權重比例為 0.25，再更周圍的(5,7)(5,6)(5,5)(6,5)(7,5) 則會更新比例為 0.125。

所以就以 vep7 的角度來看(7,0)(7,1)(7,2)(7,3)(7,4)(7,5)(7,6)(7,7)，所輸出的 neighbor_sel 會為(11,11,11,11,11,10,01,00)，即為波型上綠筆所畫的 ffe4。

5) Show simulation result

Presim:

```
RAM_RESULT[4086] = 92675b, pass
RAM_RESULT[4086] = 92675b, pass
RAM_RESULT[4087] = 92675b, pass
RAM_RESULT[4088] = 906155, pass
RAM_RESULT[4089] = 8c624a, pass
RAM_RESULT[4090] = 8c624a, pass
RAM_RESULT[4091] = 976d55, pass
RAM_RESULT[4092] = 976d55, pass
RAM_RESULT[4093] = 8c624a, pass
RAM_RESULT[4094] = 875f4c, pass
RAM_RESULT[4095] = 8d6958, pass
```

```
*****
**                                     **
** Congratulations !!               **
**                                     **
** Simulation PASS!!                **
**                                     **
*****
```

Simulation complete via \$finish(1) at time 85072850 PS + 2

./top_tb.v:236 \$finish;

xcelium> exit

TOOL: xmvverilog 20.09-s007: Exiting on Apr 22, 2022 at 23:43:48 CST (total: 00:05:36)

vlsicad6:/home/user2/vlsi22/vlsi2253/AAA/Lab7/Lab7_1 %

Postsim:

```
RAM_RESULT[4079] = 9c462b, pass
RAM_RESULT[4079] = 6e462b, pass
RAM_RESULT[4080] = 6e462b, pass
RAM_RESULT[4081] = 6e462b, pass
RAM_RESULT[4082] = 724933, pass
RAM_RESULT[4083] = 825846, pass
RAM_RESULT[4084] = 906155, pass
RAM_RESULT[4085] = 92675b, pass
RAM_RESULT[4086] = 92675b, pass
RAM_RESULT[4087] = 92675b, pass
RAM_RESULT[4088] = 906155, pass
RAM_RESULT[4089] = 8c624a, pass
RAM_RESULT[4090] = 8c624a, pass
RAM_RESULT[4091] = 976d55, pass
RAM_RESULT[4092] = 976d55, pass
RAM_RESULT[4093] = 8c624a, pass
RAM_RESULT[4094] = 875f4c, pass
RAM_RESULT[4095] = 8d6958, pass
```

```
*****
**                                     **
** Congratulations !!               **
**                                     **
** Simulation PASS!!                **
**                                     **
*****
```

Simulation complete via \$finish(1) at time 85072850 PS + 2

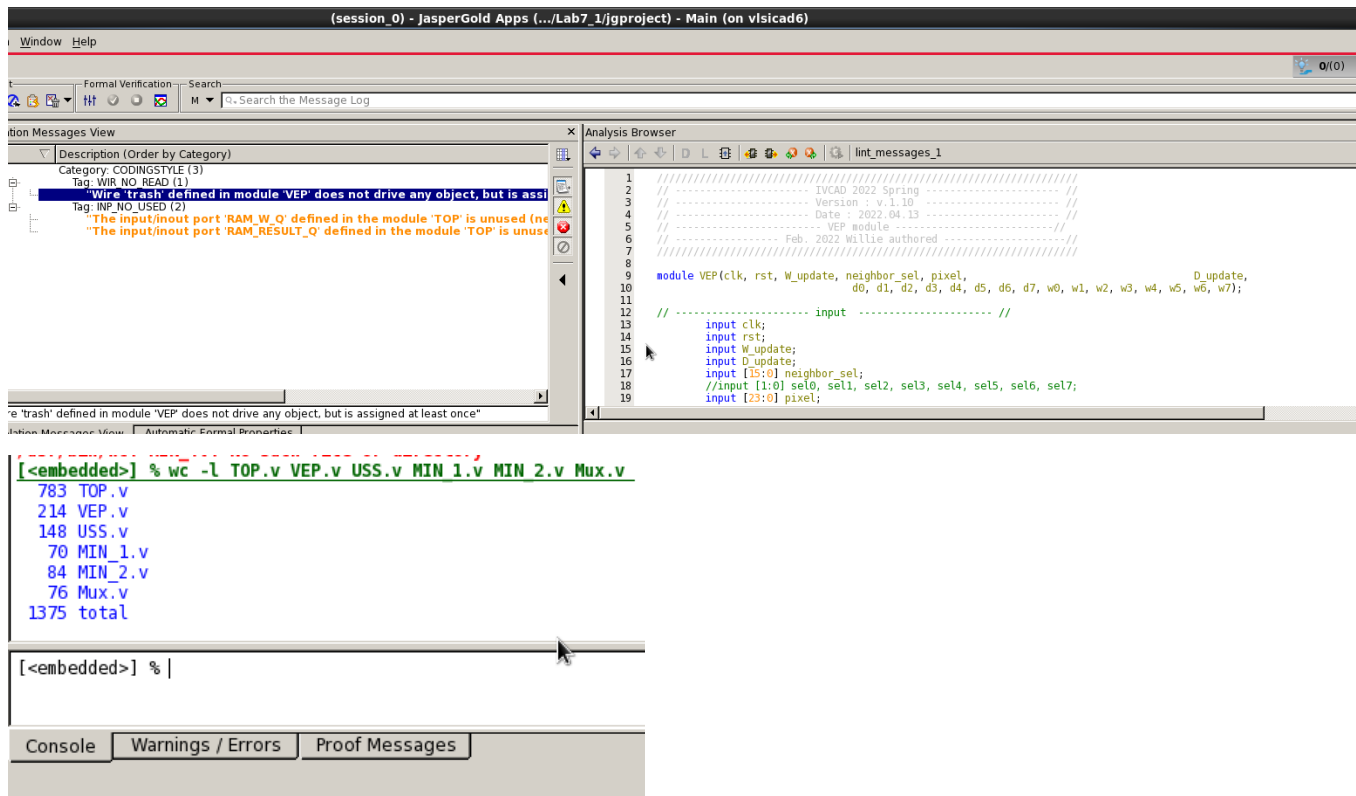
./top_tb.v:237 \$finish;

xcelium> exit

TOOL: xmvverilog 20.09-s007: Exiting on Apr 27, 2022 at 11:18:08 CST (total: 00:25:28)

vlsicad6:/home/user2/vlsi22/vlsi2253/AAA/Lab7/Lab7_1 %

6) Show SuperLint coverage (TOP.v)



Coverage = 99.8%

(有兩個是 RAM 跟 TOP 之間的 port 沒被用到，分別為 RAM_W_Q、RAM_RESULT_Q。另一個是因為我把 Design 改成 single cycle 後 D_update 就沒用到了，所以就多一個 unused)

7) Your clock period, total cell area, post simulation time (TOP.v)

Clock period: 10.3ns

Total cell area: 17958

Post simulation time: 85072ns

```

Number of ports:                31533
Number of nets:                 71228
Number of cells:                37563
Number of combinational cells:  34848
Number of sequential cells:     1789
Number of macros/black boxes:   0
Number of buf/inv:              11878
Number of references:            30

Combinational area:             15995.588078
Buf/Inv area:                   2367.718782
Noncombinational area:          1963.012840
Macro/Black Box area:           0.000000
Net Interconnect area:          10504.222427

Total cell area:                17958.600918
Total area:                     28462.823345
design_vision>
Log History
design_vision>
Ready

```

```


RAM_RESULT[4090] = 8c624a, pass
RAM_RESULT[4091] = 976d55, pass
RAM_RESULT[4092] = 976d55, pass
RAM_RESULT[4093] = 8c624a, pass
RAM_RESULT[4094] = 875f4c, pass
RAM_RESULT[4095] = 8d6958, pass

```

```

*****
**               **
** Congratulations !! **
**               **
** Simulation PASS!!  **
**               **
*****

```



```

Simulation complete via $finish(1) at time 85072850 PS + 2
./top_tb.v:237          $finish;
xcelium> exit
TOOL:  xmverilog      20.09-s007: Exiting on Apr 27, 2022 at 11:18:0
vlsicad6:/home/user2/vlsi22/vlsi2253/AAA/Lab7/Lab7_1 %

```

8) 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.

原本我也是切成好幾個 stage 來設計 LAB7，但是後來寫到 VEP 要拉 neighbor_sel 的線回來的時候想到不太對，因為若要用 pipeline 加速的話，下一筆要 training 的 weight 必須就要先更新過的，不能等 4~5 個 cycle 才更新，這要會有 data hazard。原本也想改成用 forwarding 和 flush 來解決這個 data hazard，但後來覺得可以 pipeline 的 pixel 少之又少，因為 input pixel 的圖片通常都是彼此間為類似的 pixel，所以要找中心點必會與上一次更新的有關，而且要 forwarding+flush 面積一定會大很多不符合效益，所以後來就沒常識了。

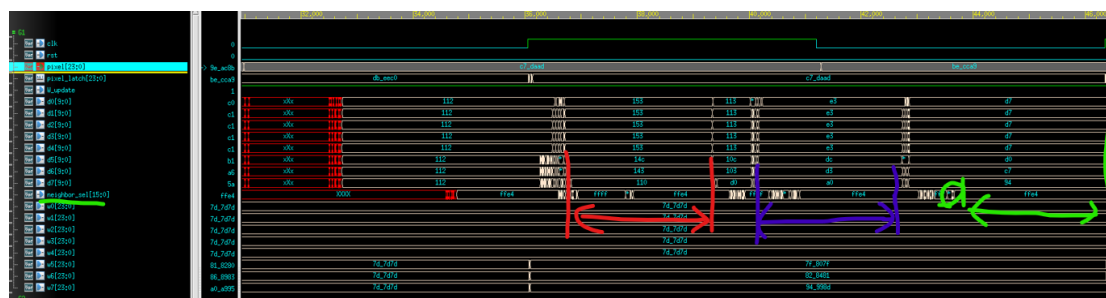
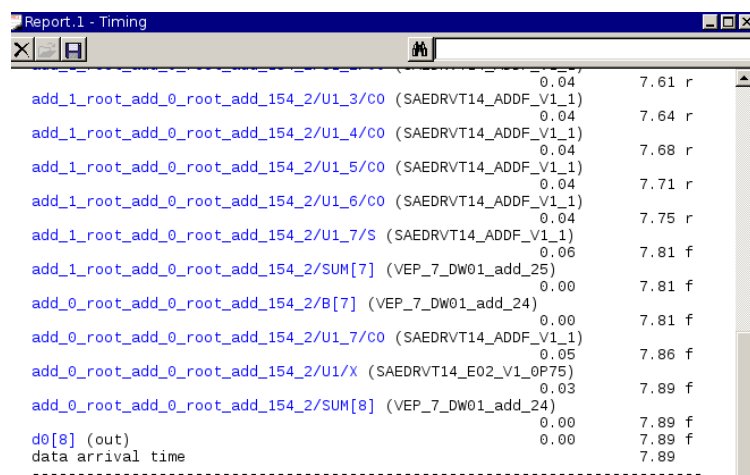
後來遇到前面 SOM 沒辦法 pipeline 後，我就在想如果後面算 inference 能夠用 pipeline 是能夠快多少?原因為我大致就抓了 VEP 一定會 dominate 整個 cycle 的 period，而 pipeline 若沒辦法壓低 period 則跟 single cycle 是同等的事情，而且

面積一定還會因為一對 register 而大幅增加。此外，我看到題目兩題給的 inference 數量都沒有比要 traing 的還多，所以就大膽猜測用 single cycle 來寫會比較快，頂多若發現 period 超過 20ns 則換成要插 register 的寫法。後來 postsim 出來果然時間比別組快將近一倍，而且面積也小好幾千，就大概知道後面用 pipeline 其實沒辦法快多少，因為 inference 的數量不夠多。

我優化面積的方法無疑就是將不會用到的 bit 數能砍就砍，能刪的 reg 就刪或是用 wire 代替，而我發現我原本用 for 寫的電路跟我把 for 裡的電路全部展開的寫法相比，合成 tool 會將用 for 的合的比我自己寫還小一點，蠻訝異的。此外若原本用 continuous assign 反而面積會因為合成的原件固定了，會比用 reg +always(*)的 combinational 還大。

底下為各個 module 所需花的最長時間：

VEP time :



VEP area:

```
saed14rvt_ss0p72v125c (File: /usr/cad/CBDK/SAED14_EDK_rvt/db_cc

Number of ports:                3323
Number of nets:                 8101
Number of cells:                5450
Number of combinational cells:  5122
Number of sequential cells:     216
Number of macros/black boxes:   0
Number of buf/inv:              1216
Number of references:           133

Combinational area:             1921.054778
Buf/Inv area:                   229.015197
Noncombinational area:          236.563205
Macro/Black Box area:           0.000000
Net Interconnect area:          2992.401393

Total cell area:                2157.617982
Total area:                     5150.019375
design_vision>
```

可看到 VEP 內部有大量 F/F，主導了整個 TOP module 的面積。

MIN_1 time:

```
U576/X (SAEDRVT14_OAI21_OP5)    0.01    0.56 r
U6/X (SAEDRVT14_OAI22_1)        0.03    0.59 f
U4/X (SAEDRVT14_INV_S_1)        0.04    0.63 r
U388/X (SAEDRVT14_OA2BB2_V1_1)  0.05    0.69 r
U295/X (SAEDRVT14_EO2_V1_OP75)  0.04    0.73 r
U294/X (SAEDRVT14_ND2_CDC_OP5)  0.02    0.75 f
U332/X (SAEDRVT14_OR4_1)        0.03    0.78 f
U176/X (SAEDRVT14_OR4_1)        0.04    0.81 f
U92/X (SAEDRVT14_AN4_1)         0.03    0.84 f
U100/X (SAEDRVT14_AN3_OP75)     0.03    0.88 f
U37/X (SAEDRVT14_BUF_ECO_1)     0.04    0.92 f
U434/X (SAEDRVT14_A022_1)       0.04    0.96 f
U433/X (SAEDRVT14_A0221_OP5)    0.02    0.99 f
U348/X (SAEDRVT14_OR2_1)        0.01    1.00 f
w_min[11] (out)                 0.00    1.00 f
data arrival time                1.00
-----
(Path is unconstrained)

design_vision>
```

MIN_2 time:

```
U614/X (SAEDRVT14_OAI21_OP5)    0.01    0.57 r
U44/X (SAEDRVT14_OAI22_1)        0.03    0.60 f
U8/X (SAEDRVT14_INV_S_1)         0.04    0.64 r
U441/X (SAEDRVT14_OA2BB2_V1_1)  0.06    0.69 r
U197/X (SAEDRVT14_EO2_V1_OP75)  0.05    0.74 r
U196/X (SAEDRVT14_ND2_CDC_OP5)  0.02    0.76 f
U259/X (SAEDRVT14_OR4_1)        0.03    0.79 f
U112/X (SAEDRVT14_OR4_1)        0.04    0.82 f
U3/X (SAEDRVT14_INV_S_1)         0.03    0.85 r
U25/X (SAEDRVT14_BUF_ECO_1)     0.04    0.88 r
U76/X (SAEDRVT14_NR2_MM_1)       0.03    0.92 f
U29/X (SAEDRVT14_BUF_ECO_1)     0.02    0.94 f
U425/X (SAEDRVT14_A0221_OP5)    0.04    0.98 f
U424/X (SAEDRVT14_OR2_1)        0.02    1.00 f
weight_c[19] (out)              0.00    1.00 f
data arrival time                1.00
-----
(Path is unconstrained)

design_vision>
```

USS time :

input external delay	0.00	0.00	r
X_c[2] (in)	0.00	0.00	r
U3/X (SAEDRVT14_INV_S_1)	0.02	0.02	f
U75/X (SAEDRVT14_AN2_MM_1)	0.03	0.05	f
U83/X (SAEDRVT14_OA21B_1)	0.02	0.07	r
U62/X (SAEDRVT14_AOI21_OP5)	0.03	0.10	f
U56/X (SAEDRVT14_INV_S_1)	0.02	0.13	r
U59/X (SAEDRVT14_OAI22_1)	0.03	0.16	f
U55/X (SAEDRVT14_INV_S_1)	0.02	0.18	r
U82/X (SAEDRVT14_OA21_1)	0.03	0.21	r
U52/X (SAEDRVT14_OAI21_OP5)	0.03	0.24	f
U40/X (SAEDRVT14_INV_S_1)	0.02	0.26	r
U80/X (SAEDRVT14_OA21_1)	0.03	0.29	r
U30/X (SAEDRVT14_ND2_CDC_OP5)	0.02	0.31	f
sel2_out[0] (out)	0.00	0.31	f
data arrival time		0.31	

(Path is unconstrained)			

MUX time:

Point	Incr	Path

input external delay	0.00	0.00 f
sel[2] (in)	0.00	0.00 f
U385/X (SAEDRVT14_INV_S_1)	0.01	0.01 r
U1/X (SAEDRVT14_NF2_MM_1)	0.03	0.04 f
U195/X (SAEDRVT14_AN2_MM_1)	0.05	0.09 f
U18/X (SAEDRVT14_BUF_ECO_1)	0.04	0.13 f
U512/X (SAEDRVT14_A022_1)	0.04	0.17 f
U511/X (SAEDRVT14_A0221_OP5)	0.03	0.20 f
U1140/X (SAEDRVT14_OR4_1)	0.03	0.23 f
U342/X (SAEDRVT14_OR4_1)	0.03	0.25 f
o[19] (out)	0.00	0.26 f
data arrival time		0.26

(Path is unconstrained)		

Total period time: $7.89+1+1+0.31=10.2\text{ns}$ (mux 的 0.26 不算在 critical path) (必須大於 10.2 才不會有 time violation，所以我合到 10.3 就已經是極限了，試 10.2 的時候就有四五個 violation 導致 weight 全錯了)

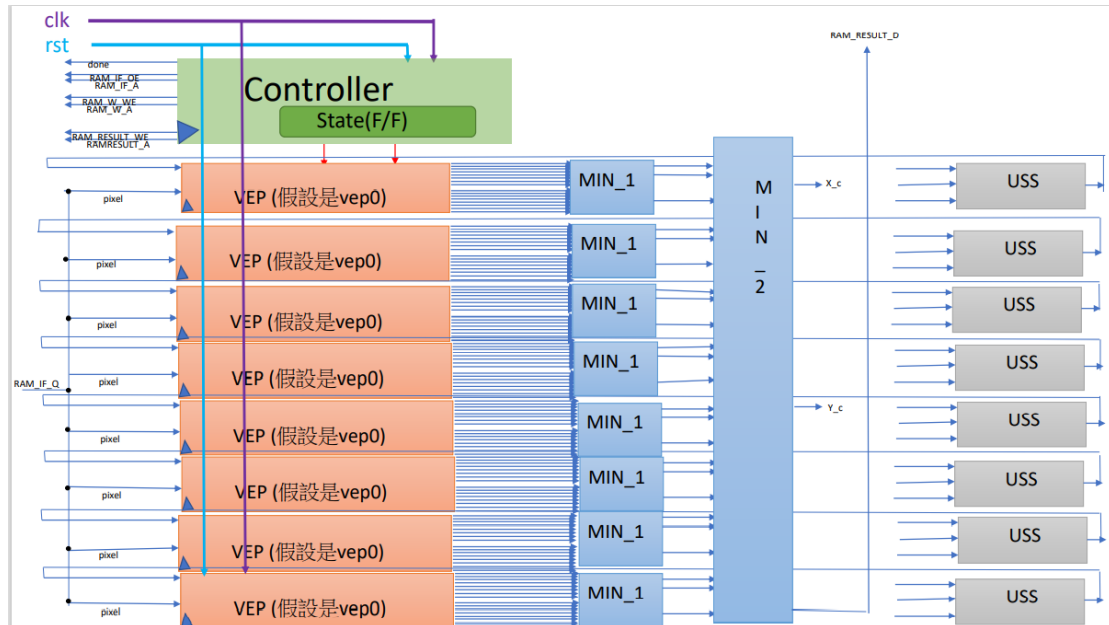
➤ Lessons learned from this lab

這個 lab 首先讓我學到了什麼是 NN，光是讀懂題目就比我寫完 RTL 還久了，利用 traing+更新比例的方式來收斂 input pixel 的特徵值，還蠻有意思的，而且這個壓縮技術聽 VLSI 組的教授有說過實驗室和業界都有在做，覺得受益良多阿。

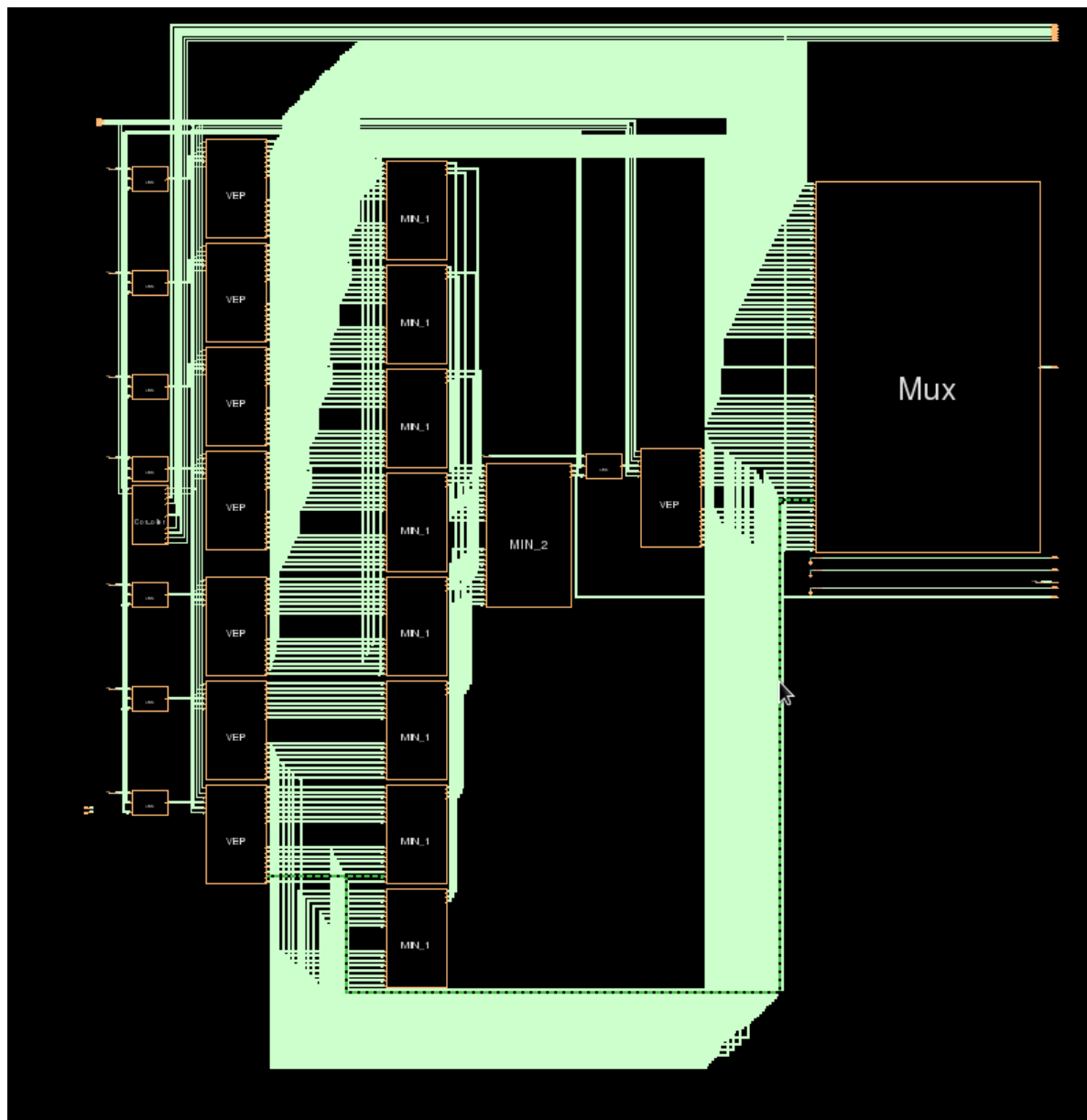
雖然這次 Lab 都是我自己完成的，缺少了團隊合作及溝通 module 間運作的體驗，但我覺得我對於計算所需的 period 更加了解了，盡然可以利用各個小 module 的最長時間來加總 critical time。

Lab 7_2

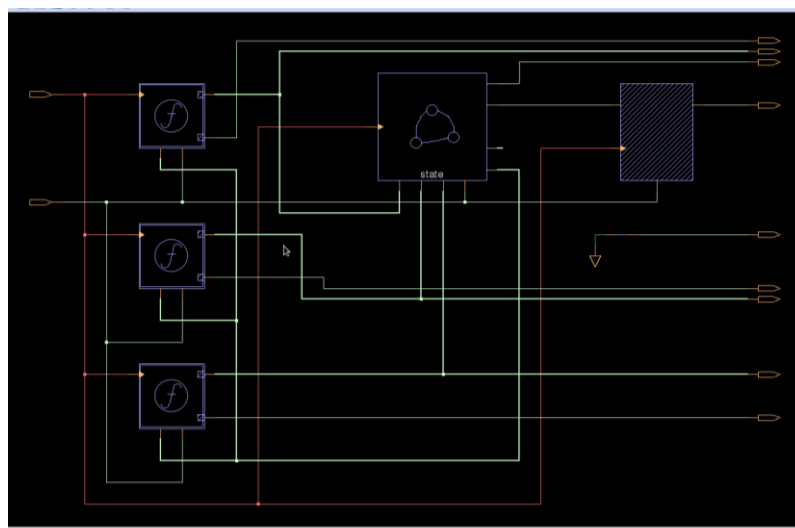
- Draw your state diagram and explain your design. You can draw internal architecture to describe your design

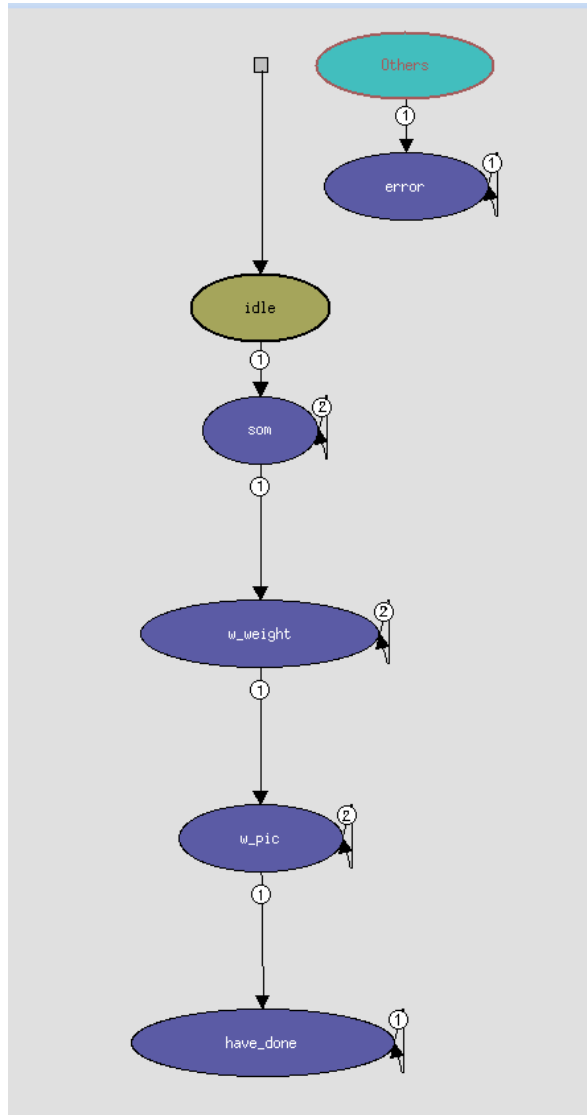


TOP module



State diagram (跟前一題幾乎一模一樣)





- 9) Complete the Controller, VEP ,MIN_1, MIN_2, USS, and TOP module, in the system.
- 10) Compile the verilog code to verify the operations of this module works properly.
- 11) 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*.

Simulation complete via \$finish(1) at time 639678 NS + 2

Postsim:

```
RAM_RESULT[20466] = c0a090, pass
RAM_RESULT[20467] = d9b8a8, pass
RAM_RESULT[20468] = f0d6c4, pass
RAM_RESULT[20469] = fbd6ca, pass
RAM_RESULT[20470] = fbd6ca, pass
RAM_RESULT[20471] = f0d6c4, pass
RAM_RESULT[20472] = fbd6ca, pass
RAM_RESULT[20473] = f0d6c4, pass
RAM_RESULT[20474] = fbd6ca, pass
RAM_RESULT[20475] = f0d6c4, pass
RAM_RESULT[20476] = d9cabe, pass
RAM_RESULT[20477] = ccb8a9, pass
RAM_RESULT[20478] = a48a79, pass
RAM_RESULT[20479] = 634734, pass
```

```
*****
**                                     **
** Congratulations !!               **
**                                     **
** Simulation PASS!!               **
**                                     **
*****
```

```
Simulation complete via $finish(1) at time 639678 NS + 2
./top_tb.v:503      $finish;
xcellium> exit
TOOL:  xmverilog      20.09-s007: Exiting on Apr 27, 2022 at 12:33:11 CST (total: 00:43:25)
vlsicad6:/home/user2/vlsi22/vlsi2253/AAA/Lab7/Lab7_2 %
```

14) Show SuperLint coverage (TOP.v)

The screenshot shows the Cadence SuperLint interface. The 'Analysis Messages View' pane on the left displays a warning: "Wire 'trash' defined in module 'VEP' does not drive any object, but is assigned at least once". The 'Analysis Browser' pane on the right shows the code for the 'VEP' module. Below the interface, a terminal window displays the command `% wc -l TOP.v VEP.v USS.v MIN 1.v MIN 2.v Mux.v` and its output:

```
783 TOP.v
214 VEP.v
148 USS.v
70 MIN_1.v
84 MIN_2.v
76 Mux.v
1375 total
```

The terminal also shows the command `%` and the 'Console' tab is selected at the bottom.

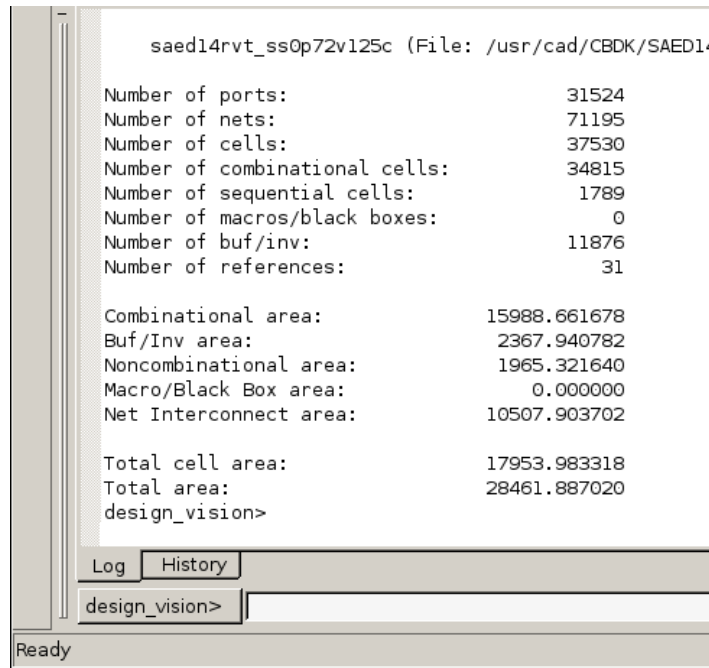
Coverage = 99.8%

15) Your **clock period**, **total cell area**, **post simulation time** (TOP.v)

Clock period: 10.4ns (10.3ns 有 time violation 了)

Total cell area: 17953

Post simulation time: 639678ns



```

RAM_RESULT[20476] = 05c406, pass
RAM_RESULT[20477] = ccb8a9, pass
RAM_RESULT[20478] = a48a79, pass
RAM_RESULT[20479] = 634734, pass
  
```

```

*****
**                                     **
** Congratulations !!               **
**                                     **
** Simulation PASS!!                **
**                                     **
*****
  
```

```

Simulation complete via $finish(1) at time 639678 NS + 2
./top_tb.v:503      $finish;
xcelium> exit
TOOL:  xmverilog      20.09-s007: Exiting on Apr 27, 2022 at
vlsicad6:/home/user2/vlsi22/vlsi2253/AAA/Lab7/Lab_2 %
  
```

➤ Lessons learned from this lab

我這題跑後模擬跑得夠久，跑了四次每次都要 40 分鐘左右，而且經過快 20 分鐘以後才跳出一個 time violation 真的很想撞牆，要重合+重跑 simulation QQ。

這次的 Lab 跟上次一樣，我都犯了檔案管理的不好示範，一直在筆電和 soc 電腦做更改 code，導致最後有快 10 幾個版本，而且修改的部分是每一份都有參雜一點，這部分助教有叫我要做更正，不要把新舊檔混在一起寫。

總之，我覺得不知不覺就在助教身上學到很多知識，像是最基本的 `always(*)` 內部用 `<=` 合出來也是個 latch，或是一些好用的寫 code 技巧，都讓我對數位越來越有興趣了，明年都想去比賽了。感謝助教用心教導!!

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
Lab7_1(pre-sim)	<code>ncverilog top_tb.v +define+X (WEIGHT, RESULT, FULL)</code>
Lab7_1 (pre-sim with waveform)	<code>ncverilog top_tb.v +access+r +define+FSDB+X</code>
Lab7_1(post-sim)	<code>ncverilog top_tb.v +define+syn+X</code>
Lab7_1 (post-sim with waveform)	<code>ncverilog top_tb.v +access+r +define+FSDB+syn+X</code>
Lab7_2(pre-sim)	<code>ncverilog top_tb.v +define+X (WEIGHT, RESULT0, RESULT1, RESULT2, RESULT3, RESULT4, FULL)</code>
Lab7_2 (pre-sim with waveform)	<code>ncverilog top_tb.v +access+r +define+FSDB+X</code>
Lab7_2(post-sim)	<code>ncverilog top_tb.v +define+FSDB+syn+X</code>
Lab7_2 (post-sim with waveform)	<code>ncverilog top_tb.v +access+r +define+FSDB+syn+X</code>