National Cheng Kung University

Department of Electrical Engineering

Introduction to VLSI CAD (Spring 2024)

Lab Session 6

Design of Local Binary Pattern Circuit

Name	Student ID	
鄭喆嚴	E14096724	
Practical	Points	Marks
Lab 6_1	35	
Lab 6_2	65	
Notes		

Due: 15:00 April 17, 2024 @ moodle

Summary

Hardware				
		RTL(∨/Χ)	Synthesis(∨/Χ)	
LBP		∨	∨	
CLBP		∨	∨	
Synthesis result (LBP/CLBP)				
Area		Simulation time (ps)		
160.652165/17595.170367		85588033/292262000		
Superlint (number of inline messages) (LBP/CLBP)				
Total lines	Warning	Error	coverage(%)	
209/867	0/11	0/0	100%/98.73%	

Note: You must complete and fill out this form with your design information!!!

Deliverables

1) All Verilog codes including testbenches, .bmp and .hex for each problem should be uploaded.
2) NOTE: Please **DO NOT** include source code in the paper report!
3) NOTE: 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.

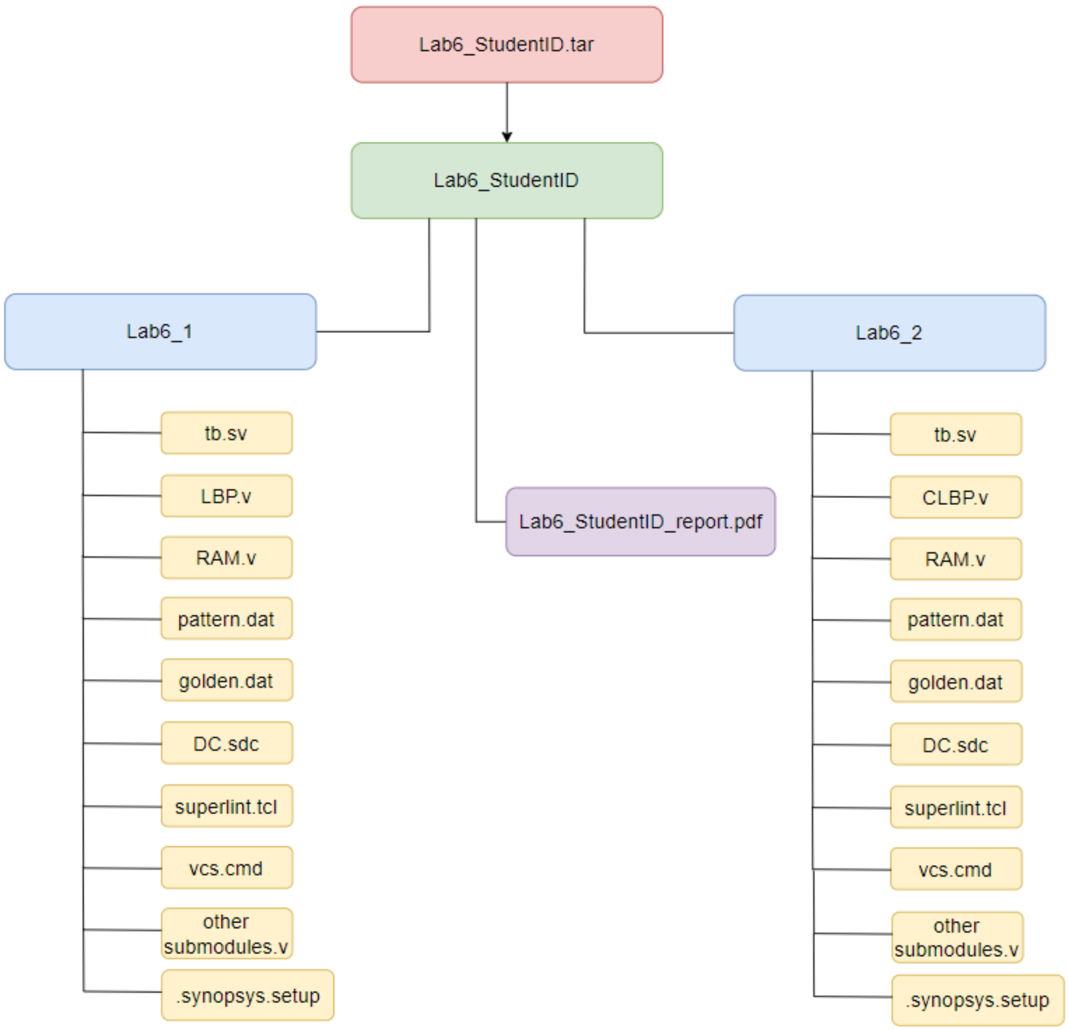


Fig.1 File hierarchy for Homework submission

The design inside the LBP block can be completed by your free will, but do not modify the I/O ports of the LBP block. The block diagram of the testbed-DUT (design under test) system is as shown in **Fig2**.

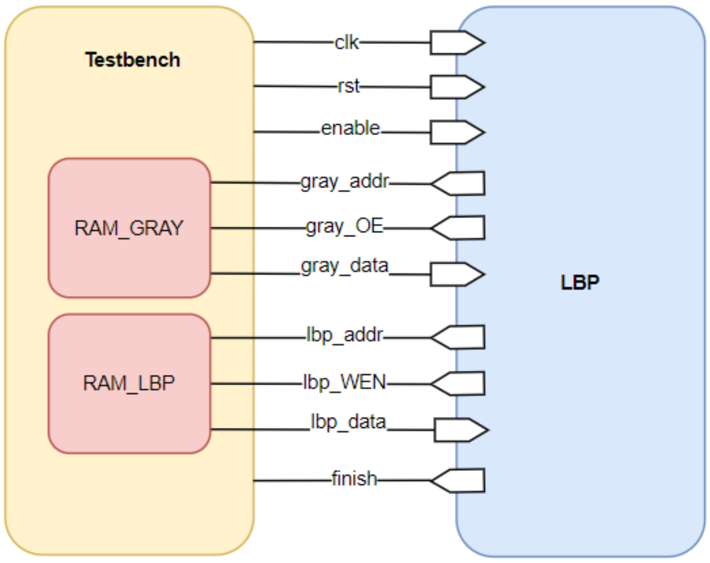


Fig2. The block diagram of local binary pattern circuit

➢ Port list of LBP:

Signal	I/O	Bit-width	Description
clk	I	1	Clock signal
rst	I	1	Reset signal
enable	I	1	Circuit enabling signal
gray_addr	O	12	Address signal connected to RAM_GRAY
gray_OE	O	1	Read enable signal to RAM_GRAY
gray_data	I	8	Read data signal from RAM_GRAY
lbp_addr	O	12	Address signal connected to RAM_LBP
lbp_WEN	O	1	Write enable signal to RAM_LBP
lbp_data	O	8	Write data signal to RAM_LBP
finish	O	1	Indication signal of the circuit is finished

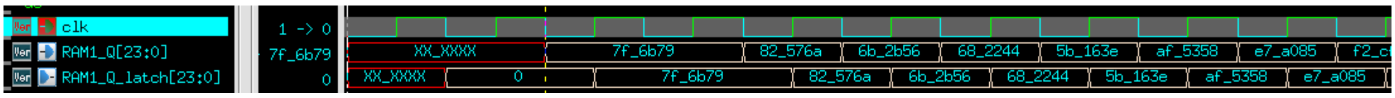


Fig3. example waveform for RAM

- Understanding the function:

  Once system is initialized, it

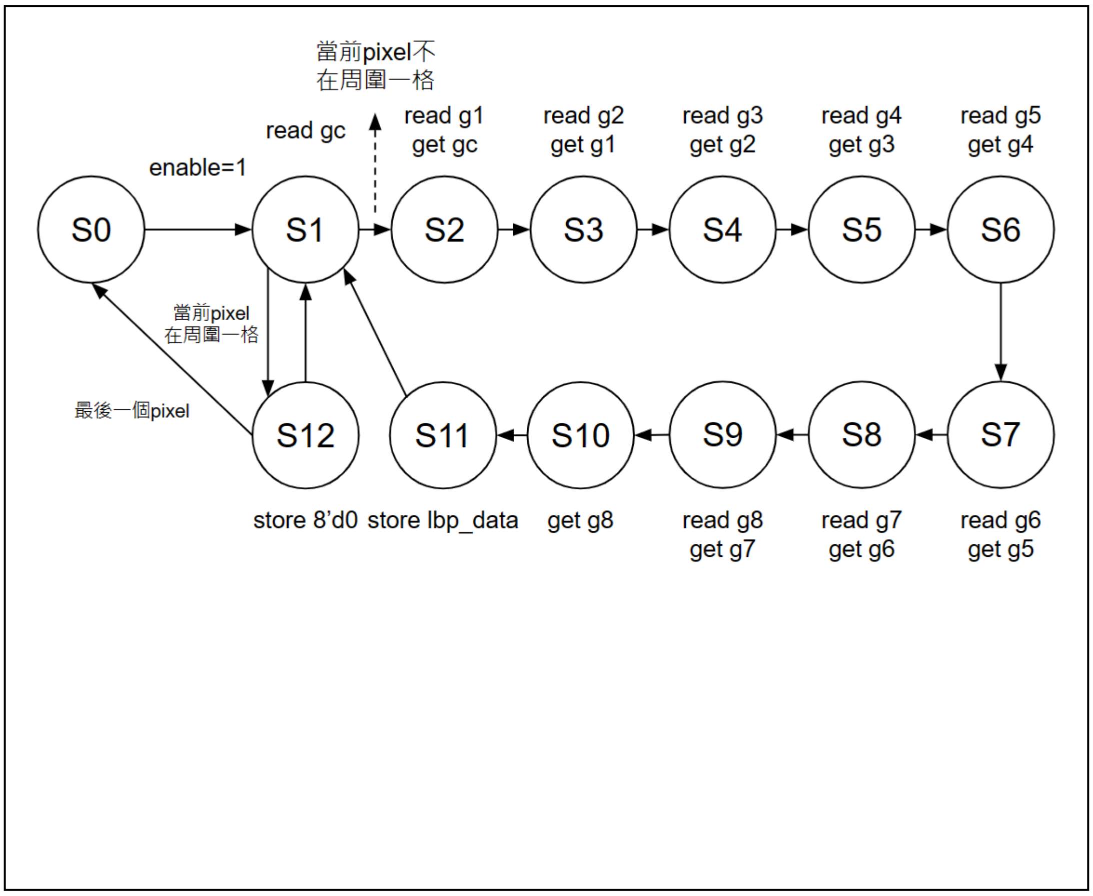
  a) Choose a pixel in the image and select its neighboring pixels.
  b) Construct the mask using threshold function.
  c) Combine the binary values for all neighboring pixels to obtain a binary code for the central pixel and convert it to a decimal value.
  d) Repeat steps a)–c) for each pixel in the image to obtain a binary code for each pixel.

- Know the basic design rules
  - All operations are activated on the positive edge of the clock.
  - Control signals:
    - *RAM_WE*: To store the data into RAM
    - *RAM_OE*: To read data from RAM
    - *finish*: Stop the process

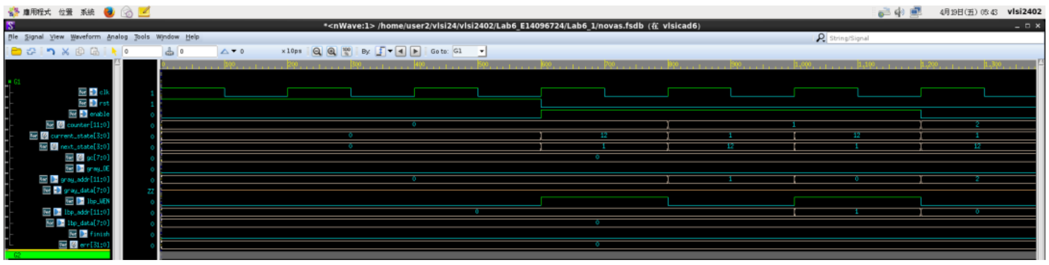
- Describe your design in detail. You can draw internal architecture or block diagram to help elaborate your design, if don't, plain text description is allowed.

我設計的 LBP 一開始 reset 後會先在 idle 的 state，當 enable 為 1 時就會開始進行運算，我使用一個 counter 來記錄現在所在的 pixel，如果 counter 現在所在的位置是在整張圖片的周圍一格，就直接跳到將 8'd0 存入 RAM_LBP 的 state；如果 counter 現在所在的位置在整張圖片的中心(不是在周圍一格)，就先利用 FSM 從 RAM_GRAY 中一個一個 state 拿取所在的 pixel 和周圍的資料，接著再用 gk>=gc 來判斷 lbp_data 的每個 bit 為 1 或 0 後跳至下個 state，最後一個 state 再將 lbp_data 存入 RAM_LBP 中，當跑到後一個 pixel 後就回到 idle 的 state。

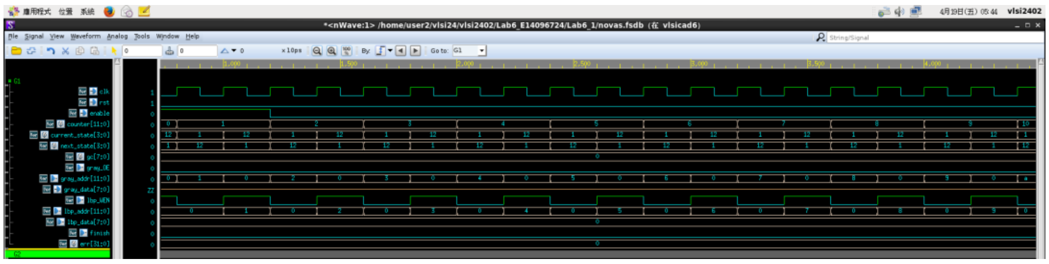
- Controller
  - Draw your state diagram in controller and explain it.



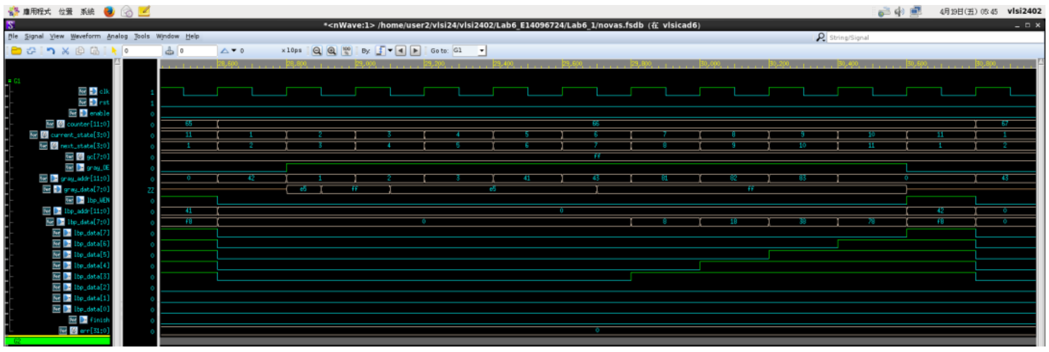
1. Complete the LBP module, in the system.
2. Compile the verilog code to verify the operations of this module works properly.
3. Synthesize your *LBP.v* with following constraint:
   - Clock period: no more than 2.0 ns.
   - Don't touch network: clk.
   - Wire load model: N16ADFP_StdCellss0p72vm40c.
   - Synthesized verilog file: *LBP_syn.v*.
   - Timing constraint file: *LBP_syn.sdf*.
4. Please **attach your waveforms** and **specify your operations** on the waveforms.



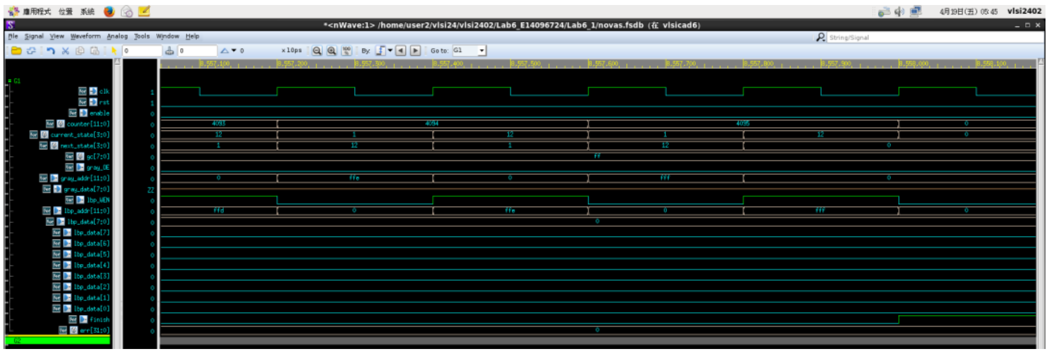
從波形中可以看到，一開始在 reset 後先在 idle 的 state，當 enable 為 1 時就會開始進行運算，counter 也開始計算。



從波形中可以看到，當前的 pixel 在圖片的周圍一格時會直接跳到另一個 state 並儲存 0 到 RAM_LBP 中。

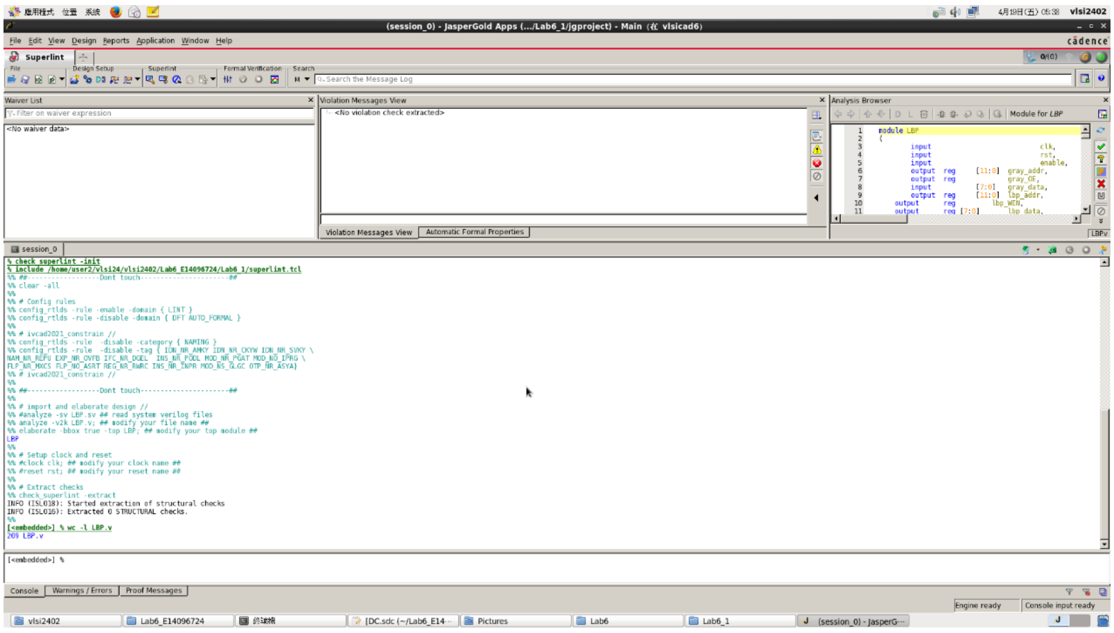


從波形中可以看到，在每個 state 從 RAM_GRAY 中拿取資料後就馬上進行運算 lbp_data 後跳到下一個 k，並且在最後一個 state 將運算完的資料儲存回 RAM_LBP 中，接著再回到開始運算的 state。



從波形中可以看到，最後在計算完 counter 為 4095 時就讓 finish 變成 1。

5. Show SuperLint coverage (including all files)



```
module LBP
(
        input                   clk,
        input                   rst,
        input                   enable,
        output  reg     [11:0]  gray_addr,
        output  reg             gray_OE,
        input           [7:0]   gray_data,
        output  reg     [11:0]  lbp_addr,
    output      reg         lbp_WEN,
    output      reg [7:0]       lbp_data,
```

```
% check superlint -init
% include /home/user2/vlsi24/vlsi2402/Lab6_E14096724/Lab6_1/superlint.tcl
%% ##-----------------Dont touch--------------------##
%% clear -all
%%
%% # Config rules
%% config_rtlds -rule -enable -domain { LINT }
%% config_rtlds -rule -disable -domain { DFT AUTO_FORMAL }
%%
%% # ivcad2021_constrain //
%% config_rtlds -rule  -disable -category { NAMING }
%% config_rtlds -rule  -disable -tag { IDN_NR_AMKY IDN_NR_CKYW IDN_NR_SVKY \
NAM_NR_REPU EXP_NR_OVFB IFC_NR_DGEL  INS_NR_PODL MOD_NR_PGAT MOD_NO_IPRG \
FLP_NR_MXCS FLP_NO_ASRT REG_NR_RWRC INS_NR_INPR MOD_NS_GLGC OTP_NR_ASYA}
%% # ivcad2021_constrain //
%%
%% ##-----------------Dont touch--------------------##
%%
%% # import and elaborate design //
%% #analyze -sv LBP.sv ## read system verilog files
%% analyze -v2k LBP.v; ## modify your file name ##
%% elaborate -bbox true -top LBP; ## modify your top module ##
LBP
%%
%% # Setup clock and reset
%% #clock clk; ## modify your clock name ##
%% #reset rst; ## modify your reset name ##
%%
%% # Extract checks
%% check_superlint -extract
INFO (ISL018): Started extraction of structural checks
INFO (ISL016): Extracted 0 STRUCTURAL checks.
%%
[<embedded>] % wc -l LBP.v
209 LBP.v
```

LBP.v : (1 – 0/209) * 100% = 100%

6. Your clock period, total cell area, post simulation time with screenshot.

Clock period: 1.0

Total cell area: 160.652165

Post simulation time: 85588033 ps

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

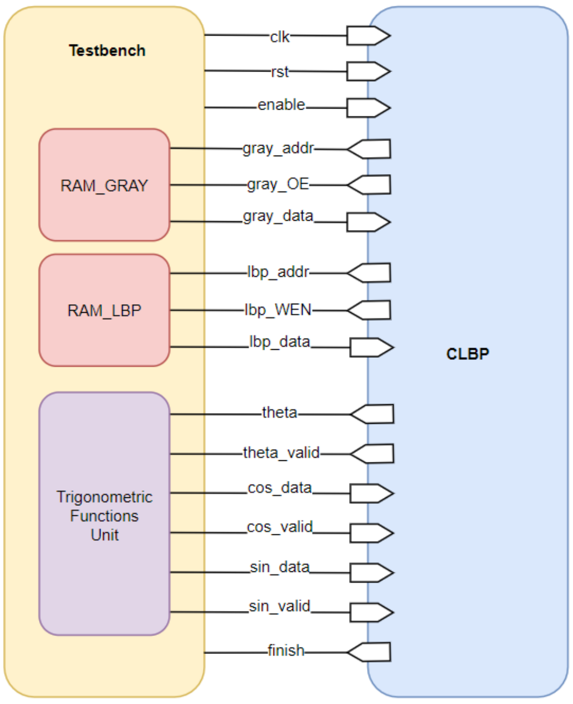
我在每個 state 讀取後計算都使用同一個 register 來讓 cell area 盡量減少，當每一筆資料從 RAM_GRAY 中拿取後就馬上進行運算，這樣就可以用同一個 register 來處存資料，並且在拿取到最後一筆資料實就馬上進行運算並且處存到 RAM_LBP 中，這樣盡可能的加快執行速度。

8. Lessons learned from this lab

這次的 lab 讓我學到了有關於 LBP 的運作方式以及如何在 Verilog 中實現 LBP，我也學到了如何使用 RAM 來正確地獲取資料，需要在下一個 clock 才能讀取到所需的資料，這個讓我需要另外思考如何設計 FSM，我設計了多個 state 來獲取 gc 和周圍的 gk，我還另外設計一個 state 來處理如果當前 pixel 在圖片的周圍時直接儲存 0。這次實驗的過程中我也學到了很多在實作上的經驗。

Lab 6_2: Circular Local Binary Pattern

Extend the original LBP algorithm to circular one.



▲The block diagram of circular local binary pattern circuit

➢ Port list of top:

Signal	I/O	Bit-width	Description
clk	I	1	Clock signal
rst	I	1	Reset signal
enable	I	1	Circuit enabling signal
gray_addr	O	12	Address signal connected to RAM_GRAY
gray_OE	O	1	Read enable signal to RAM_GRAY
gray_data	I	8	Read data signal from RAM_GRAY
lbp_addr	O	12	Address signal connected to RAM_LBP

lbp_WEN	**O**	**1**	**Write enable signal to RAM_LBP**
lbp_data	**O**	**8**	**Write data signal to RAM_LBP**
theta	**O**	**25(fixed-point)**	**Current neighbor's angle signal(unit is in radian)**
theta_valid	**O**	**1**	**Indication signal of current neighbor's angle is valid**
cos_data	**I**	**25(fixed-point)**	**Cosine value of the theta(from testbench)**
cos_valid	**I**	**1**	**Indication signal of cosine value is valid**
sin_data	**I**	**25(fixed-point)**	**Sine value of the theta(from testbench)**
sin_valid	**I**	**1**	**Indication signal of sine value is valid**
finish	**O**	**1**	**Indication signal of the circuit is finished**

➢ Understanding the function:

Once system is initialized, it

a) Choose a pixel(center) in the image and select its neighboring pixels.

b) Calculate bilinear interpolation:

   1) Determine r_x & r_y.
   2) Determine x_1, x_2, y_1, y_2.
   3) Determine t_x, t_y.
   4) Determine w1,w2, w3,w4.
   5) Determine $f(0,0)$, $f(0,1)$, $f(1,0)$, $f(1,1)$.
   6) Determine neighbor.

c) Repeat b) to calculate all neighbors' values.

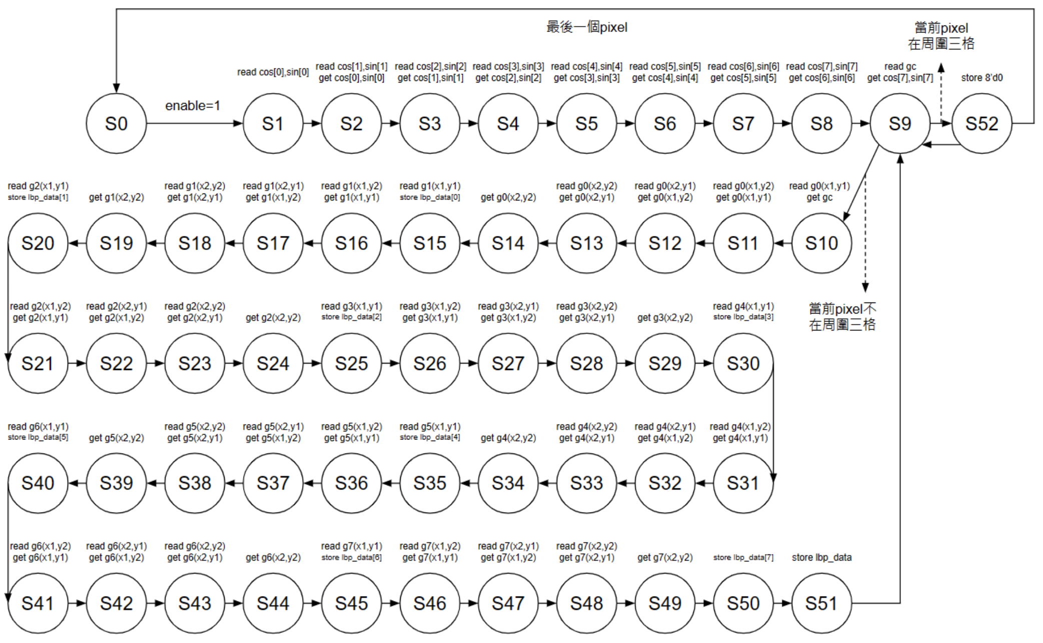
d) Construct the mask using threshold function.

e) Combine the binary values for all neighboring pixels to obtain a binary code for the central pixel and convert it to a decimal value.

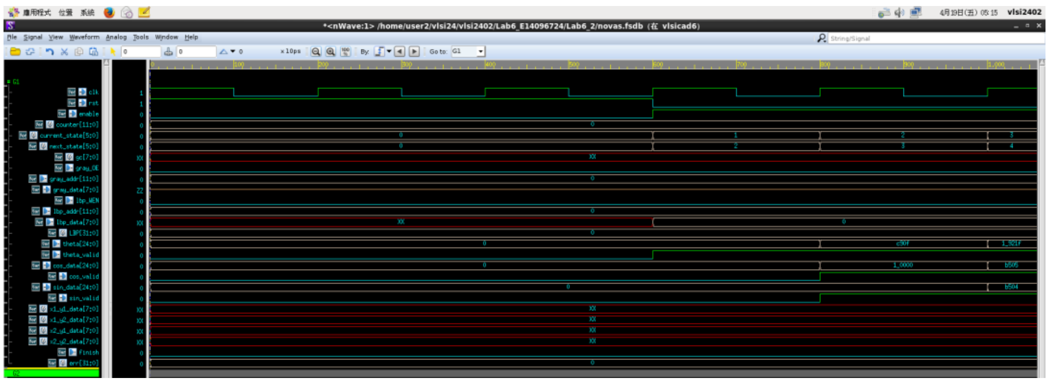
f) Repeat steps a)–e) for each pixel in the image to obtain a binary code for each pixel.

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

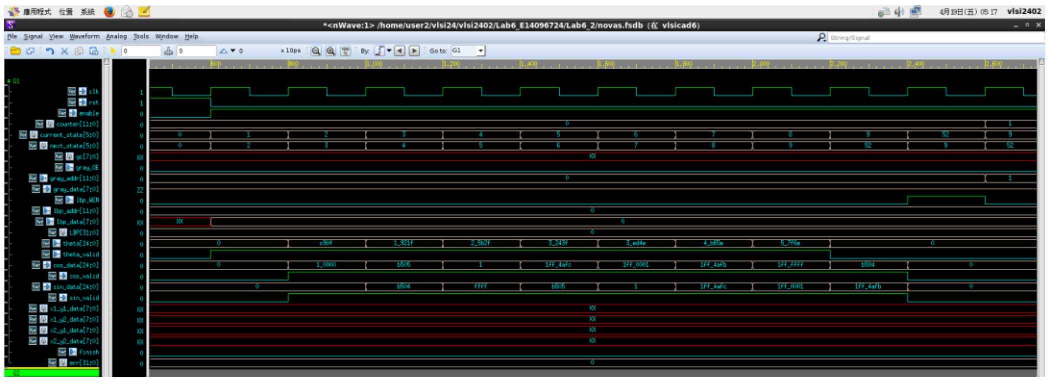
我設計的 CLBP 一開始 reset 後會先在 idle 的 state，當 enable 為 1 時就會開始進行運算，先從 testbench 中一個一個 state 拿取每個角度的 sin 值和 cos 值，並且接著計算出每個角度的 rx、ry、x1、x2、y1、y2、tx、ty、w1、w2、w3、w4，我使用一個 counter 來記錄現在所在的 pixel，如果 counter 現在所在的位置是在整張圖片的周圍三格，就直接跳到將 8'd0 存入 RAM_LBP 的 state；如果 counter 現在所在的位置在整張圖片的中心(不是在周圍三格)，就先利用 FSM 從 RAM_GRAY 中一個一個 state 拿取所在的 pixel 每個角度上的四個資料，接著在每五個 state 後用內差法計算出 LBP 的數值，並且用 LBP>gc 來判斷 lbp_data 的每個 bit 為 1 或 0 後跳至下個 state，最後一個 state 再將 lbp_data 存入 RAM_LBP 中，當跑到後一個 pixel 後就回到 idle 的 state。


最後一個pixel
當前pixel 在周圍三格
enable=1
read cos[0],sin[0]
read cos[1],sin[1] get cos[0],sin[0]
read cos[2],sin[2] get cos[1],sin[1]
read cos[3],sin[3] get cos[2],sin[2]
read cos[4],sin[4] get cos[3],sin[3]
read cos[5],sin[5] get cos[4],sin[4]
read cos[6],sin[6] get cos[5],sin[5]
read cos[7],sin[7] get cos[6],sin[6]
read gc get cos[7],sin[7]
store 8'd0
S0
S1
S2
S3
S4
S5
S6
S7
S8
S9
S52
read g2(x1,y1) store lbp_data[1]
get g1(x2,y2)
read g1(x2,y2) get g1(x2,y1)
read g1(x2,y1) get g1(x1,y2)
read g1(x1,y2) get g1(x1,y1)
read g1(x1,y1) store lbp_data[0]
get g0(x2,y2)
read g0(x2,y2) get g0(x2,y1)
read g0(x2,y1) get g0(x1,y2)
read g0(x1,y2) get g0(x1,y1)
read g0(x1,y1) get gc
S20
S19
S18
S17
S16
S15
S14
S13
S12
S11
S10
read g2(x1,y2) get g2(x1,y1)
read g2(x2,y1) get g2(x1,y2)
read g2(x2,y2) get g2(x2,y1)
get g2(x2,y2)
read g3(x1,y1) store lbp_data[2]
read g3(x1,y2) get g3(x1,y1)
read g3(x2,y1) get g3(x1,y2)
read g3(x2,y2) get g3(x2,y1)
get g3(x2,y2)
read g4(x1,y1) store lbp_data[3]
當前pixel不 在周圍三格
S21
S22
S23
S24
S25
S26
S27
S28
S29
S30
read g6(x1,y1) store lbp_data[5]
get g5(x2,y2)
read g5(x2,y2) get g5(x2,y1)
read g5(x2,y1) get g5(x1,y2)
read g5(x1,y2) get g5(x1,y1)
read g5(x1,y1) store lbp_data[4]
get g4(x2,y2)
read g4(x2,y2) get g4(x2,y1)
read g4(x2,y1) get g4(x1,y2)
read g4(x1,y2) get g4(x1,y1)
S40
S39
S38
S37
S36
S35
S34
S33
S32
S31
read g6(x1,y2) get g6(x1,y1)
read g6(x2,y1) get g6(x1,y2)
read g6(x2,y2) get g6(x2,y1)
get g6(x2,y2)
read g7(x1,y1) store lbp_data[6]
read g7(x1,y2) get g7(x1,y1)
read g7(x2,y1) get g7(x1,y2)
read g7(x2,y2) get g7(x2,y1)
get g7(x2,y2)
store lbp_data[7]
store lbp_data
S41
S42
S43
S44
S45
S46
S47
S48
S49
S50
S51

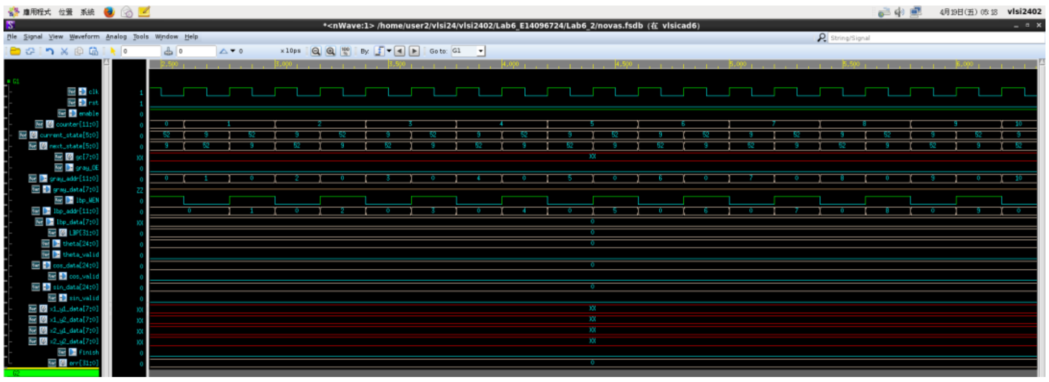
1. Complete the CLBP module.
2. Compile the verilog code to verify the operations of this module works properly.
3. Synthesize your *CLBP.v* with following constraint:
    - Clock period: no more than 2.0 ns.
    - Don't touch network: clk.
    - Wire load model: N16ADFP_StdCellss0p72vm40c.
    - Synthesized verilog file: *CLBP_syn.v*.
    - Timing constraint file: *CLBP_syn.sdf*.
4. Please **attach your waveforms** and **specify your operations** on the waveforms.



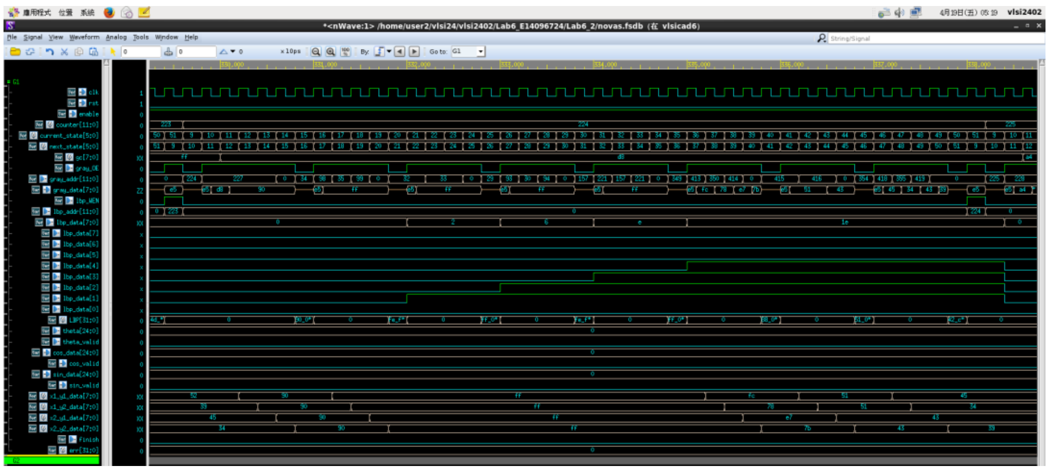
從波形中可以看到，一開始在 reset 後先在 idle 的 state，當 enable 為 1 時就會開始進行運算，counter 也開始計算。



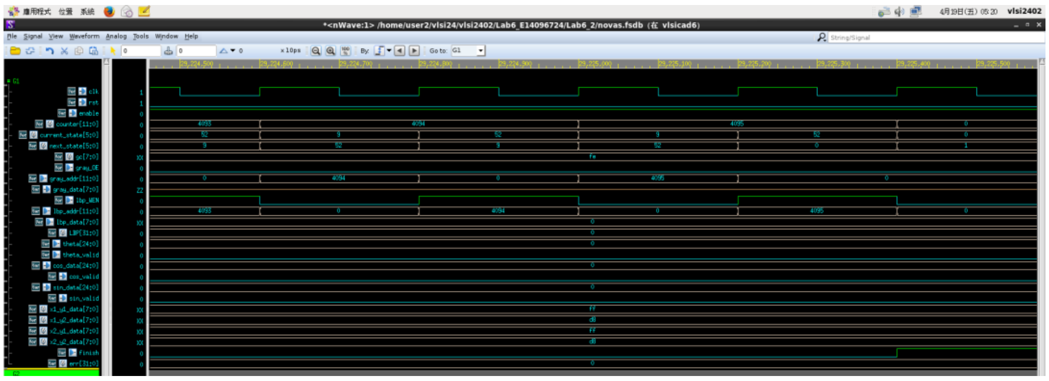
從波形中可以看到，在一開始運算的時候會先從 testbench 中一個一個 state 拿取每個角度的 sin 值和 cos 值，並且接著計算出每個角度的 rx、ry、x1、x2、y1、y2、tx、ty、w1、w2、w3、w4。



從波形中可以看到，當前的 pixel 在圖片的周圍三格時會直接跳到另一個 state 並儲存 0 到 RAM_LBP 中。

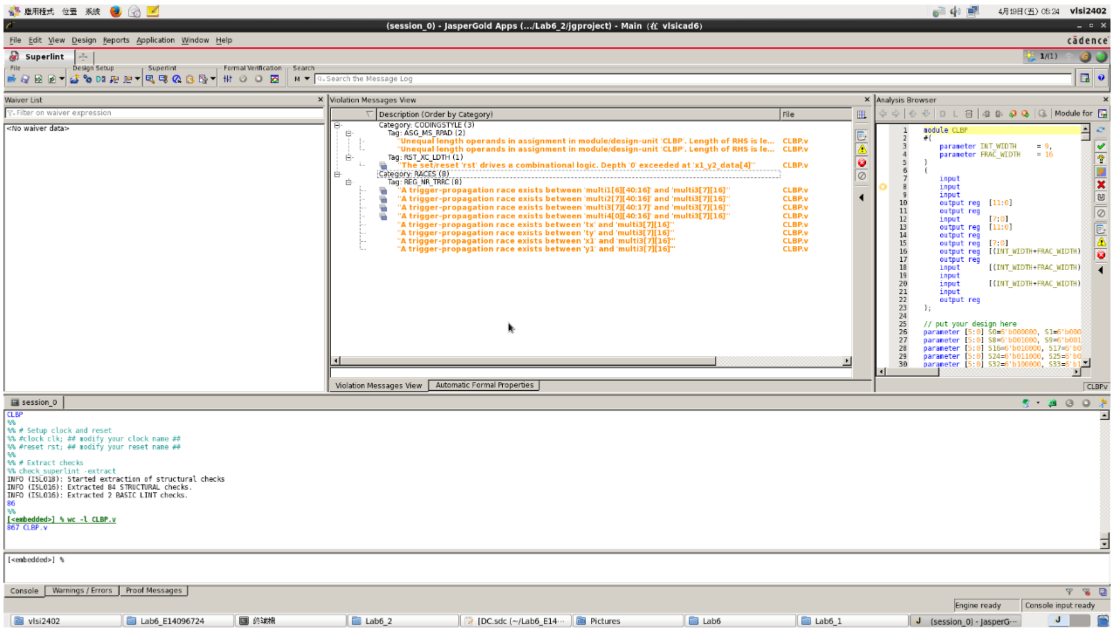


從波形中可以看到，在每個 state 從 RAM_GRAY 中拿取 x1y1, x1y2, x2y1, x2y2 的資料後就馬上進行運算 lbp_data 後跳到下一個 k，並且在最後一個 state 將運算完的資料儲存回 RAM_LBP 中，接著再回到開始運算的 state。



從波形中可以看到，最後在計算完 counter 為 4095 時就讓 finish 變成 1。

5. Show SuperLint coverage (include all files)



```
Category: CODINGSTYLE (3)
  Tag: ASG_MS_RPAD (2)
    "Unequal length operands in assignment in module/design-unit 'CLBP'. Length of RHS is le...   CLBP.v
    "Unequal length operands in assignment in module/design-unit 'CLBP'. Length of RHS is le...   CLBP.v
  Tag: RST_XC_LDTH (1)
    "The set/reset 'rst' drives a combinational logic. Depth '0' exceeded at 'x1_y2_data[4]'"   CLBP.v
Category: RACES (8)
  Tag: REG_NR_TRRC (8)
    "A trigger-propagation race exists between 'multi1[6][40:16]' and 'multi3[7][16]'"   CLBP.v
    "A trigger-propagation race exists between 'multi2[7][40:16]' and 'multi3[7][16]'"   CLBP.v
    "A trigger-propagation race exists between 'multi3[7][40:17]' and 'multi3[7][16]'"   CLBP.v
    "A trigger-propagation race exists between 'multi4[0][40:16]' and 'multi3[7][16]'"   CLBP.v
    "A trigger-propagation race exists between 'tx' and 'multi3[7][16]'"   CLBP.v
    "A trigger-propagation race exists between 'ty' and 'multi3[7][16]'"   CLBP.v
    "A trigger-propagation race exists between 'x1' and 'multi3[7][16]'"   CLBP.v
    "A trigger-propagation race exists between 'y1' and 'multi3[7][16]'"   CLBP.v
```

```
module CLBP
#(
    parameter INT_WIDTH      = 9,
    parameter FRAC_WIDTH     = 16
)
(
    input
    input
    input
    output reg  [11:0]
    output reg
    input       [7:0]
    output reg  [11:0]
    output reg
    output reg  [7:0]
    output reg  [(INT_WIDTH+FRAC_WIDTH)
    output reg
    input       [(INT_WIDTH+FRAC_WIDTH)
    input
    input       [(INT_WIDTH+FRAC_WIDTH)
    input
    output reg
);

// put your design here
parameter [5:0] S0=6'b000000, S1=6'b000
parameter [5:0] S8=6'b001000, S9=6'b001
parameter [5:0] S16=6'b010000, S17=6'b0
parameter [5:0] S24=6'b011000, S25=6'b0
parameter [5:0] S32=6'b100000, S33=6'b1
```

```
CLBP
%%
%% # Setup clock and reset
%% #clock clk; ## modify your clock name ##
%% #reset rst; ## modify your reset name ##
%%
%% # Extract checks
%% check_superlint -extract
INFO (ISL018): Started extraction of structural checks
INFO (ISL016): Extracted 84 STRUCTURAL checks.
INFO (ISL016): Extracted 2 BASIC LINT checks.
86
%%
[<embedded>] % wc -l CLBP.v
867 CLBP.v
```

```
[<embedded>] %
```

CLBP.v : (1 - 11/867) * 100% = 98.73%

6. Your clock period, total cell area, post simulation time with screenshot
   Clock period: 2.0
   Total cell area: 17595.170367
   Post simulation time: 292262000 ps

7. Lessons learned from this lab
   這次的 lab 讓我學到了有關於 CLBP 的運作方式以及如何在 Verilog 中實現 CLBP，經過上一次 LBP 的後這次在做 CLBP 時也覺得更加容易一些，我利用相似的設計來完成 CLBP，只是 CLBP 需要另外拿到 sin 和 cos 的資料，並且計算出內插計算的權重，我另外設計了幾個 state 來提前先運算完所有所需的數據後，再真的在圖片上做運算，來有這次拿取的資料不只有一個了，需要一次拿取四個資料才能運算出當前 pixel 的 lbp_data，這次的 lab 讓我學習到如何設計 FSM 來讓想要的設計更容易完成。

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)	vcs -R -full64 -sverilog tb.sv +access+r +vcs+fsdbon
Lab6_1(post)	vcs -R -full64 -sverilog tb.sv +access+r +vcs+fsdbon +define+SDF+SYN
Lab6_2(pre)	vcs -R -full64 -sverilog tb.sv +access+r +vcs+fsdbon
Lab6_2(post)	vcs -R -full64 -sverilog tb.sv +access+r +vcs+fsdbon +define+SDF+SYN