

National Cheng Kung University
Department of Electrical Engineering

Introduction to VLSI CAD (Spring 2025)

Lab Session 4

**Register File and FIFO Design with Manhattan
and Euclidean Distance Applications**

Name	Student ID	
Practical Sections	Points	Marks
Lab in class	10	
Prob A	40	
Prob B-1	10	
Prob B-2	20	
Report	20	
Notes:		

Due Date: 15:10, March 19, 2025 @ 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 or you will not get the full credit.
NOTE: Please **DO NOT** upload waveforms!
- 3) **Important! TA will use the command in Appendix A to check your design under SoC Lab environment, if your code can not be recompiled by TA successfully using the commands, you will not get the full credit.**
- 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) **File hierarchy should not be changed; it may cause your code can not be recompiled by TA successfully using the autograding commands**

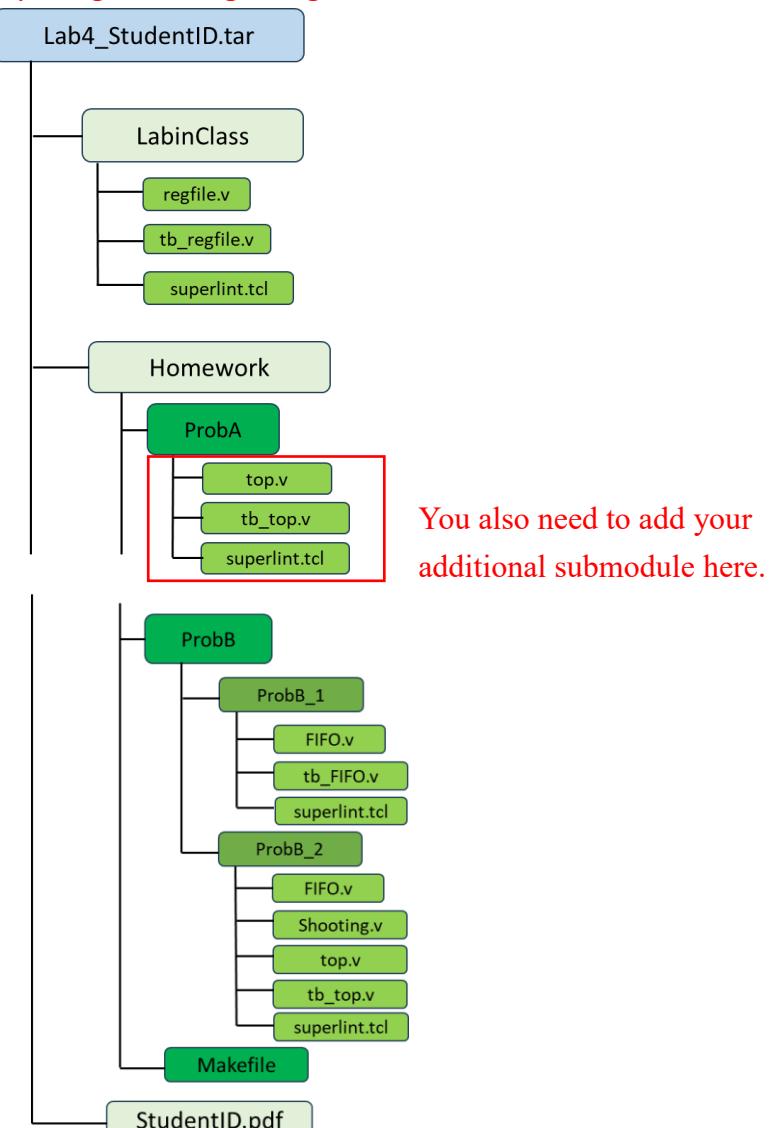
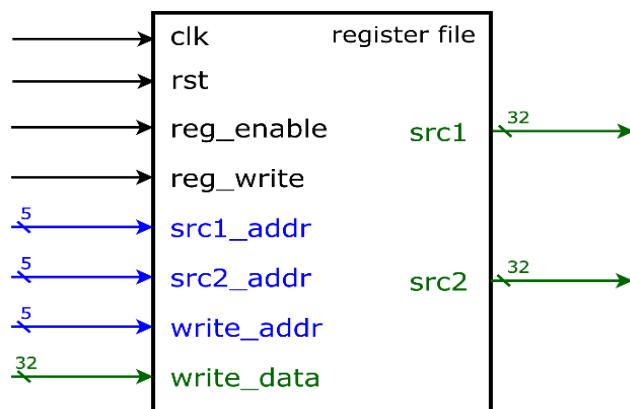


Fig.1 File hierarchy for Homework submission

Lab in class: 32x32 bit Register File



1. Design a **32 x 32** bit register file with **2** output ports.
 2. Port list

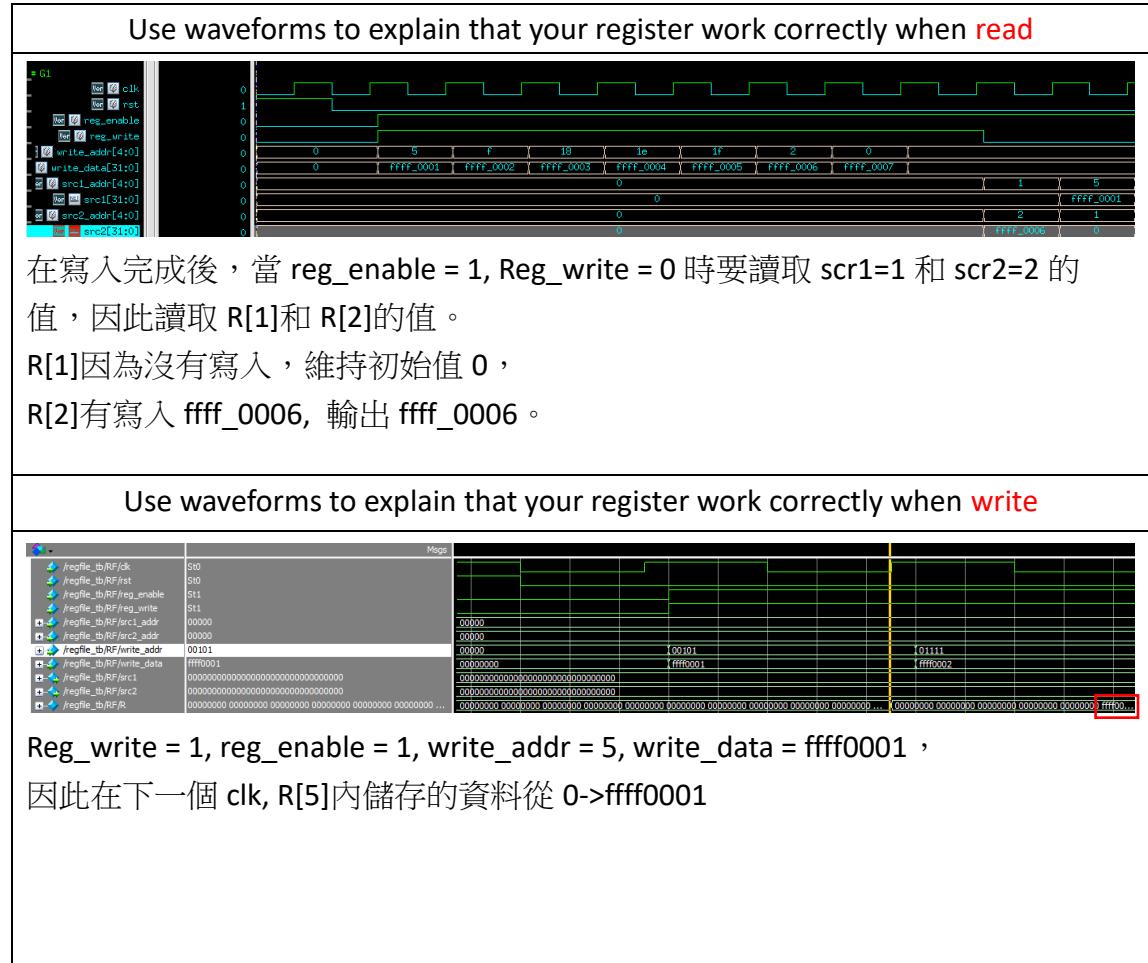
Signal	Type	Bits	Description
clk	input	1	clock
rst	input	1	reset
reg_enable	input	1	0 → off 1 → on
reg_write	input	1	0 → read 1 → write
src1_addr	input	5	source1 address
src2_addr	input	5	source2 address
write_addr	input	5	write address
write_data	input	32	write data
src1	output	32	read data source1
src2	output	32	read data source2

3.

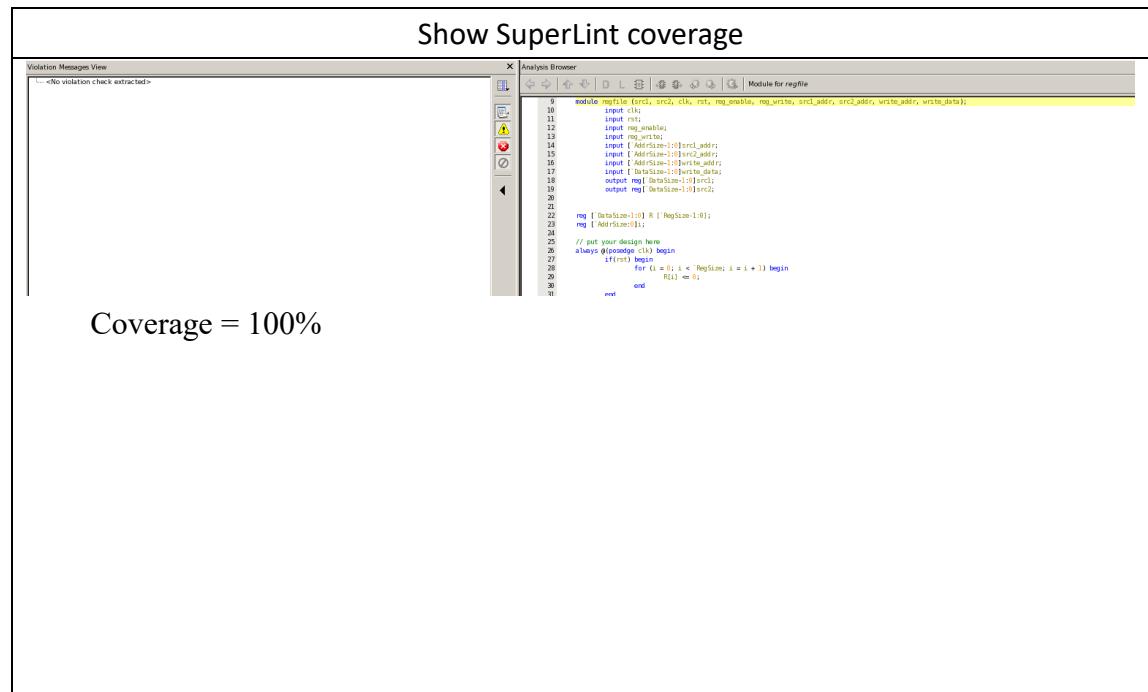
Show the simulation result on the terminal

```
*****  
**          **  
** Congratulations !!    **      / \_||  
**          **  
** Simulation PASS!!    **      / \_||  
**          **      | ^ ^ ^ \|  
*****\m___m_|_||  
  
$finish called from file "tb_Regfile.v", line 208.  
$finish at simulation time      33600  
          V C S   S i m u l a t i o n   R e p o r t  
Time: 336000 ps  
CPU Time:      0.120 seconds;      Data structure size:  0.0Mb  
Fri Mar 14 00:46:06 2025  
CPU time: .122 seconds to compile + .139 seconds to elab + .113 seconds to link + .142 sec
```

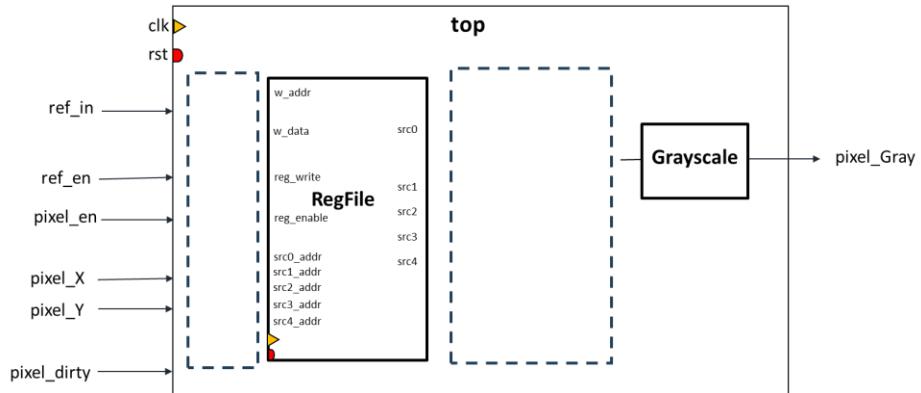
4. Show waveforms to explain that your register work correctly when **read** and **write**.



5.



Prob A: Image Repair



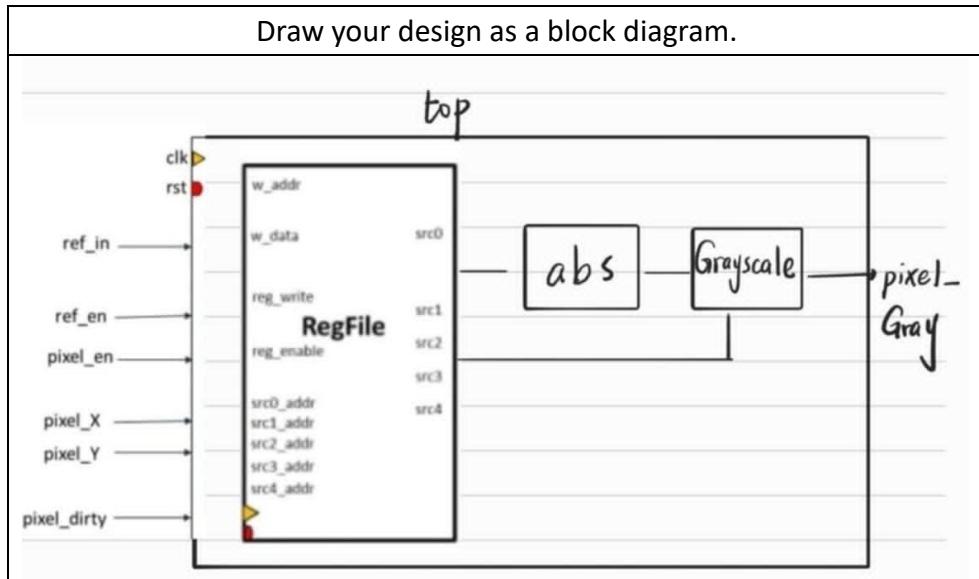
1. Design an image repair circuit in `top.v`, and set the ports of the top module according to the diagram below.

Signal	Type	Bits	Description
<code>clk</code>	input	1	Clock signal
<code>rst</code>	input	1	Reset, active high
<code>ref_en</code>	input	1	Write compared pixel enable. When <code>ref_en</code> is high, then <code>ref_in</code> is available
<code>ref_in</code>	input	28	Reference pixel
<code>pixel_en</code>	input	1	Write input pixel coordinate enable When <code>pixel_en</code> is high, then <code>pixel_X</code> , <code>pixel_Y</code> , <code>pixel_dirty</code> is available
<code>pixel_X</code>	input	2	x-coordinate
<code>pixel_Y</code>	input	2	y-coordinate
<code>pixel_broken</code>	input	1	1 :Pixel is broken 0: Pixel isn't broken
<code>pixel_Gray</code>	output	8	Grayscale output

2. Design a register file to store the RGB values of each pixel.
3. Design a Grayscale module that can convert RGB to grayscale.
4. Connect all the designed modules in `top.v`.

Note: Make sure to include the used modules in `top.v`.

5.

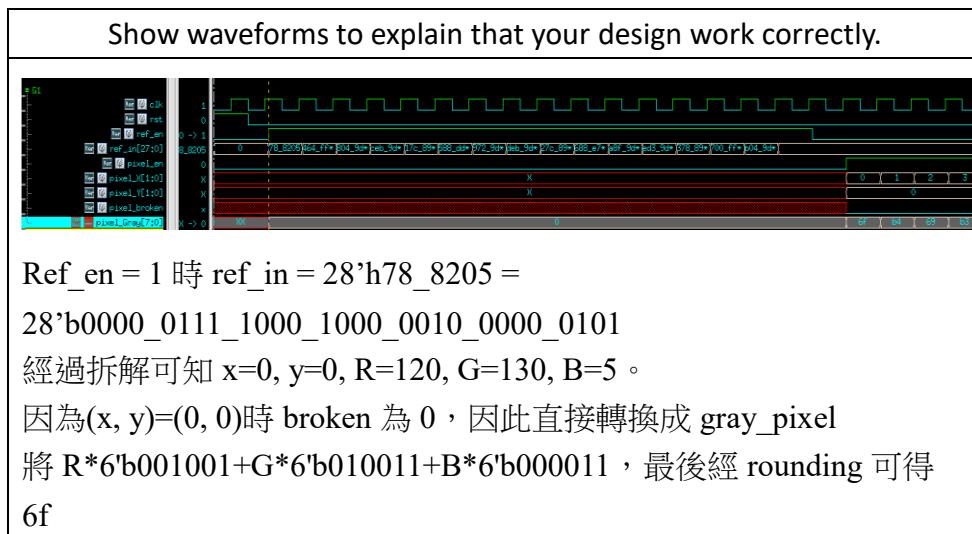


6.

Show the simulation result on the terminal

```
*****  
** Congratulations !! ** / \_ \| |  
** Simulation PASS!! ** / \_ ^ ^ \ |  
** | ^ ^ ^ | w |  
*****  
  
$finish called from file "tb_top.v", line 557.  
$finish at simulation time 119600  
V C S S i m u l a t i o n R e p o r t  
Time: 1196000 ps  
CPU Time: 0.110 seconds; Data structure size: 0.0Mb  
Fri Mar 14 01:52:17 2025  
CPU time: .141 seconds to compile + .144 seconds to elab + .115 seconds to link + .133 seconds
```

7.



8.

Show SuperLint coverage

Description (Order by Category)

Category: CODINGSTYLE (5)

- Tag: ARI_MS_DBLK (1)
 - "Constant '15'@0' will be left-padded by 8 '0' bits"
- Tag: INS_NR_PIECE (1)
 - "A piece of code is defined in a port expression"
- Tag: ASG_MS_RMD (3)
 - "Unequal length operands in assignment in module/design-unit 'top'. Length of RHS is less than LHS. LHS 'product' is 16-bit, RHS is 8-bit."
 - "Unequal length operands in assignment in module/design-unit 'top'. Length of RHS is less than LHS. LHS 'product' is 16-bit, RHS is 8-bit."
 - "The latches 'neighbor_src1' in the process/always block are mixed with combinational logic"
- Category: FILEFORMAT (2)
- Tag: ARI_MS_DRNG (2)
- Category: LOGIC (1)
- Tag: LAT_NR_BLOCK (1)
 - "In module/design-unit regfile, latch is assigned by blocking assignments"
- Tag: AUV_IC_SENL (1)
 - "Sensitivity list incomplete in module top, missing signals: pixel_en"

module top (pixel_Gray, clk, rst, ref_en, ref_in, pixel_en, pixel_X, pixel_Y, pixel_broken);
input [15:0] pixel_Gray;
input [1:0] clk;
input [1:0] rst;
input [1:0] ref_en;
input [15:0] ref_in;
input [1:0] pixel_en;
input [1:0] pixel_X;
input [1:0] pixel_Y;
output [1:0] pixel_broken;
output [1:0] pixel_Gray;
// get your design here
wire [15:0] pixel_R0;
wire [15:0] in_addr, out_addr;
wire [20:0] neighbor_addr[0:3];
assign out_addr = (in_addr << 2) + (ref_in[2:0] << 2);
assign out_addr = pixel_X * (out_addr[3] << 2);
regfile regfil (.srcl(pixel_R0), .clk(clk), .rstr(rst), .ref_enable(ref_en), .ref_write(ref_in), .in(pixel_en));
reg [7:0] R_distan, G_distan, B_distan;
wire [15:0] result_U_distan[0:3];

Coverage = 142/152 = 93%

Prob B-1: FIFO



1. Design a synchronous FIFO with a depth of 8 (able to store 8 data) and a data bit-width of 32 bits.
2. Port list

Signal	I/O	Bits	Description
clk	Input	1	Clock signal
rst	Input	1	Reset signal(active high)
write_en	Input	1	Write enable
write_data	Input	32	Data to be written into FIFO
read_en	Input	1	Read enable
read_data	Output	32	Data being read out of FIFO
full_flag	Output	1	Indicates FIFO is full or not
empty_flag	Output	1	Indicates FIFO is empty or not

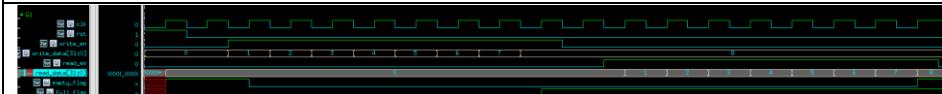
3.

Show the simulation result on the terminal

```
--FIFO test--  
100 The FIFO is full. full_flag is 1.  
110 The FIFO is full. full_flag is 1.  
120 read_data 00000001 is correct  
130 read_data 00000002 is correct  
140 read_data 00000003 is correct  
150 read_data 00000004 is correct  
160 read_data 00000005 is correct  
170 read_data 00000006 is correct  
180 read_data 00000007 is correct  
190 read_data 00000008 is correct.  
  
*****  
**      **      \|_|||  
** Congratulations !!  **      / 0.0 |  
**      **      / \^ ^\ |  
** Simulation PASS!!  **      | ^ ^ ^ | w  
*****  
  
$finish called from file "tb_FIFO.v", line 178.  
$finish at simulation time 19000  
V C S S i m u l a t i o n R e p o r t  
time: 190000 ps  
CPU Time: 0.130 seconds; Data structure size: 0.0Mb  
Fri Mar 14 02:06:51 2025  
CPU time: .125 seconds to compile + .148 seconds to elab + .116 seconds to link + .139 seconds in simulation  
74101034@CSH: >
```

4.

Show waveforms to explain that your design work correctly



一開始先將 input 1, 2, ..., 7, 8 依序讀入 queue, 結束後 full 訊號升起，之後開始做 pop，依序為 1,2,...7,8，此時 empty_flag 升起。

5.

Show SuperLint coverage

Violations Messages View

Description (Order by Category)

- Category: CCR-NR-UCMP(1)
 - Top.CCR_NR_UCMP(1)
 - *Unequal length operands in equality operator encountered (padding produces incorrect result) in module/design-unit
- Category: IDX-NR-DRNG(2)
 - Top.IDX_NR_DRNG(2)
 - *Variable index/range selection of 'write_ptr' is potentially outside the defined range*
 - *Variable index/range selection of 'read_ptr' is potentially outside the defined range*

Analysis Browser

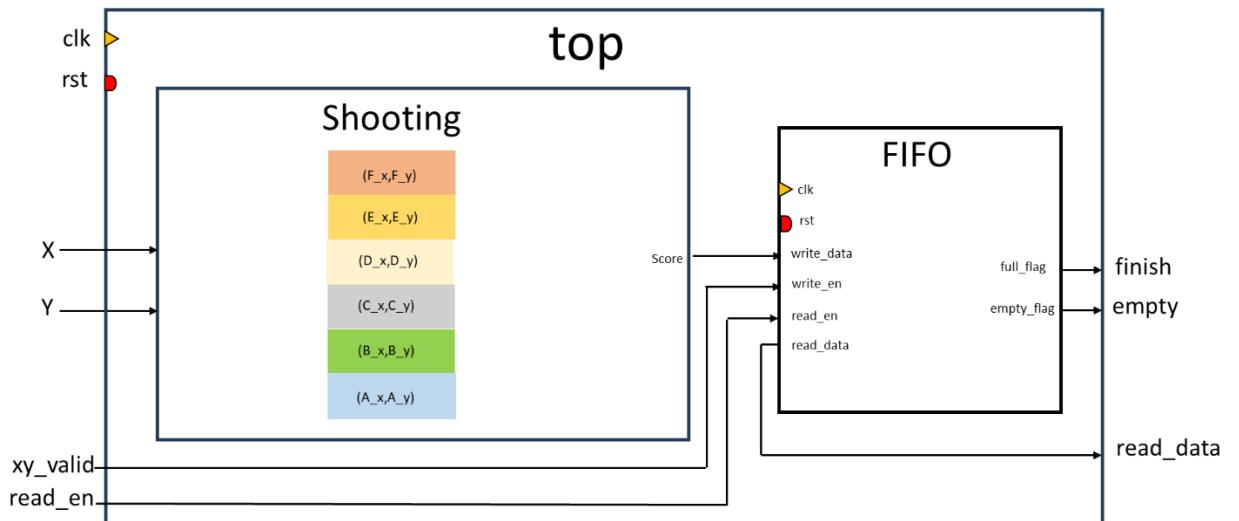
```

1  //create a fifo
2  define Depth 8
3  define DataSize 32
4
5  module FIFO (full_flag, empty_flag, read_data, clk, rst, write_data, write_en, read_en);
6    input [DataSize-1:0] write_data;
7    input                write_en;
8    input                clk;
9    input                rst;
10   output [DataSize-1:0] read_data;
11   output               full_flag;
12   output               empty_flag;
13   output reg [DataSize-1:0] read_data;
14
15
16  // put your design here
17  reg [DataSize-1:0] queue[Depth-1:0];
18  reg [1:0] read_ptr, write_ptr;
19
20  assign full_flag = (write_ptr == read_ptr);
21  assign empty_flag = (write_ptr == read_ptr);
22  integer i;
23
24  always @(posedge clk) begin
25    if(rst) begin
26      read_ptr <= 4'd0;
27      read_ptr <= 4'd0;
28    end

```

Coverage = 47/50 = 94%

Prob B-2: Shooting



1. Design a **Shooting module** based on the port list in the table.

Signal	IO	Bits	Description
X	Input	4	The x-coordinate of the shooting point
Y	Input	4	The y-coordinate of the shooting point
score	Output	32	Shooting point score

2. Connect the completed **Shooting module** and the **FIFO** from ProbB-1 in the **top module**.

- 3.

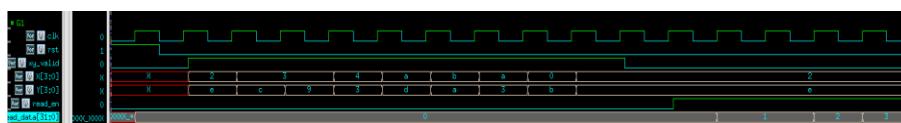
Show the simulation result on the terminal

```
-- Second shot --
FIFO[      0] is correct
FIFO[      1] is correct
FIFO[      2] is correct
FIFO[      3] is correct
FIFO[      4] is correct
FIFO[      5] is correct
FIFO[      6] is correct
FIFO[      7] is correct

*****
**   / \_||_
**   / \_/\_\
** Simulation PASS!!  ** |^__^| |w|
***** \m__m_|_|
***** \m__m_|_|
$finish called from file "tb_top.v", line 142.
$finish at simulation time          48100
          V C S   S i m u l a t i o n   R e p o r t
Time: 481000 ps
CPU Time:      0.120 seconds;      Data structure size:  0.0Mb
Fri Mar 14 02:12:25 2025
CPU time: .124 seconds to compile + .143 seconds to elab + .116 seconds to link + .142 sec
```

4.

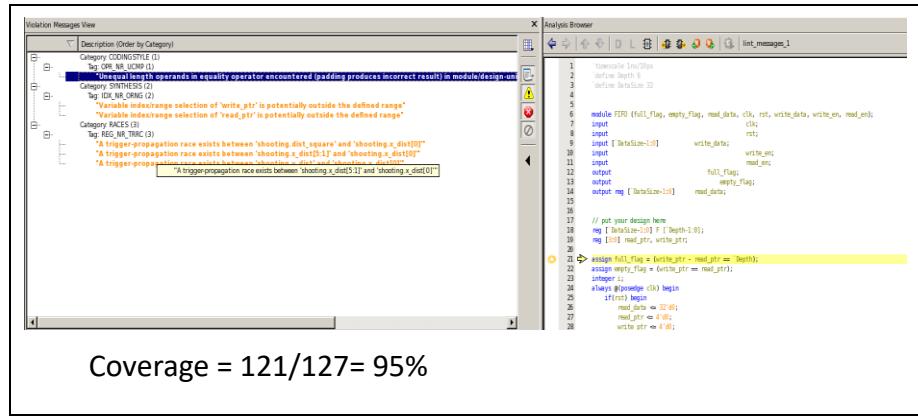
Show waveforms to explain that your design work correctly



Input1 x=2, y=14 因在 A 圈內得 1 分並寫入 queue,
Input2 x=3, y=12 因在 A 圈內得 1 分並寫入 queue,
Input2 x=3, y=9 因在 B 圈內得 2 分並寫入 queue,
依此類推，最後再將值 pop 出來。

5.

Show SuperLint coverage



At last, please write the lesson you learned from Lab4

之前練習 IC contest 時有寫到與圓距離相關寫法，這次練習到讓我更確切知道如何判斷點與圓的相對關係。另外，之前在寫 verilog 就有聽說過如果資料比較大，就會需要寫一個 regfile，如果不可以直接存在陣列，原本不知道是甚麼意思，經過這次親自實作 regfile，讓我更了解這個觀念。

這次 lab 我覺得最複雜的地方是 always@(*)和 always@(posedge clk)使用，我有時候不確定某個訊號要放在哪一種 block 。

Problem	Command
Lab in class	% vcs -R tb_Regfile.v -debug_access+all -full64 +define+FSDB
ProbA	make probA
ProbB-1	make probB_1
ProbB-1	make probB_2

