Lab Session 4
Synthesis & Register Files & Convolution

Advisor: Lih-Yih Chiou

Speaker: Michael

Date: 03.17.2021

Outline

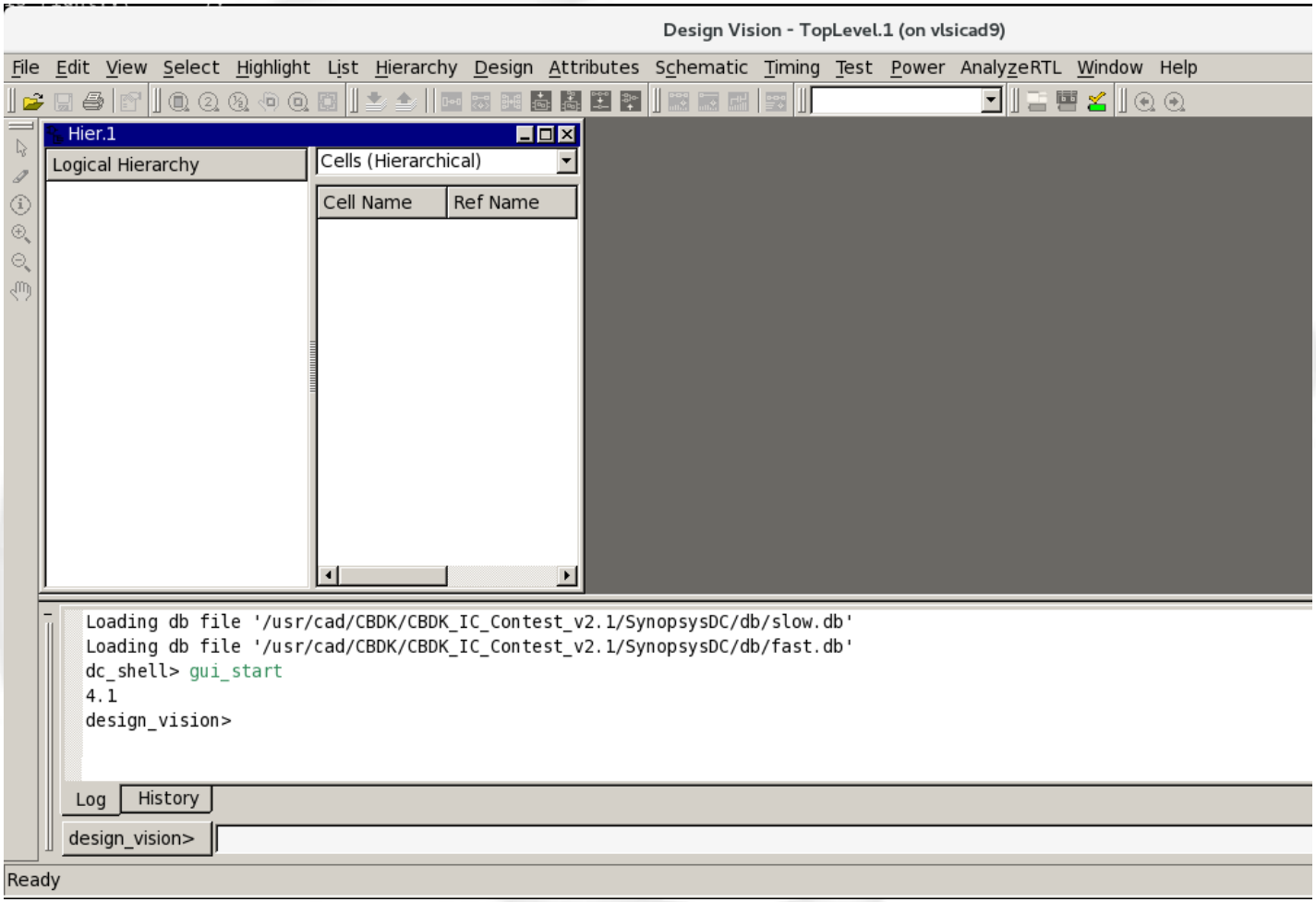
- Synthesis steps
- Synthesis tips
- Lab A : Register File
- Lab B : Convolution and activation function
- Homework

Outline

- Synthesis steps
- Synthesis tips
- Lab A : Register File
- Lab B : Convolution and activation function
- Homework

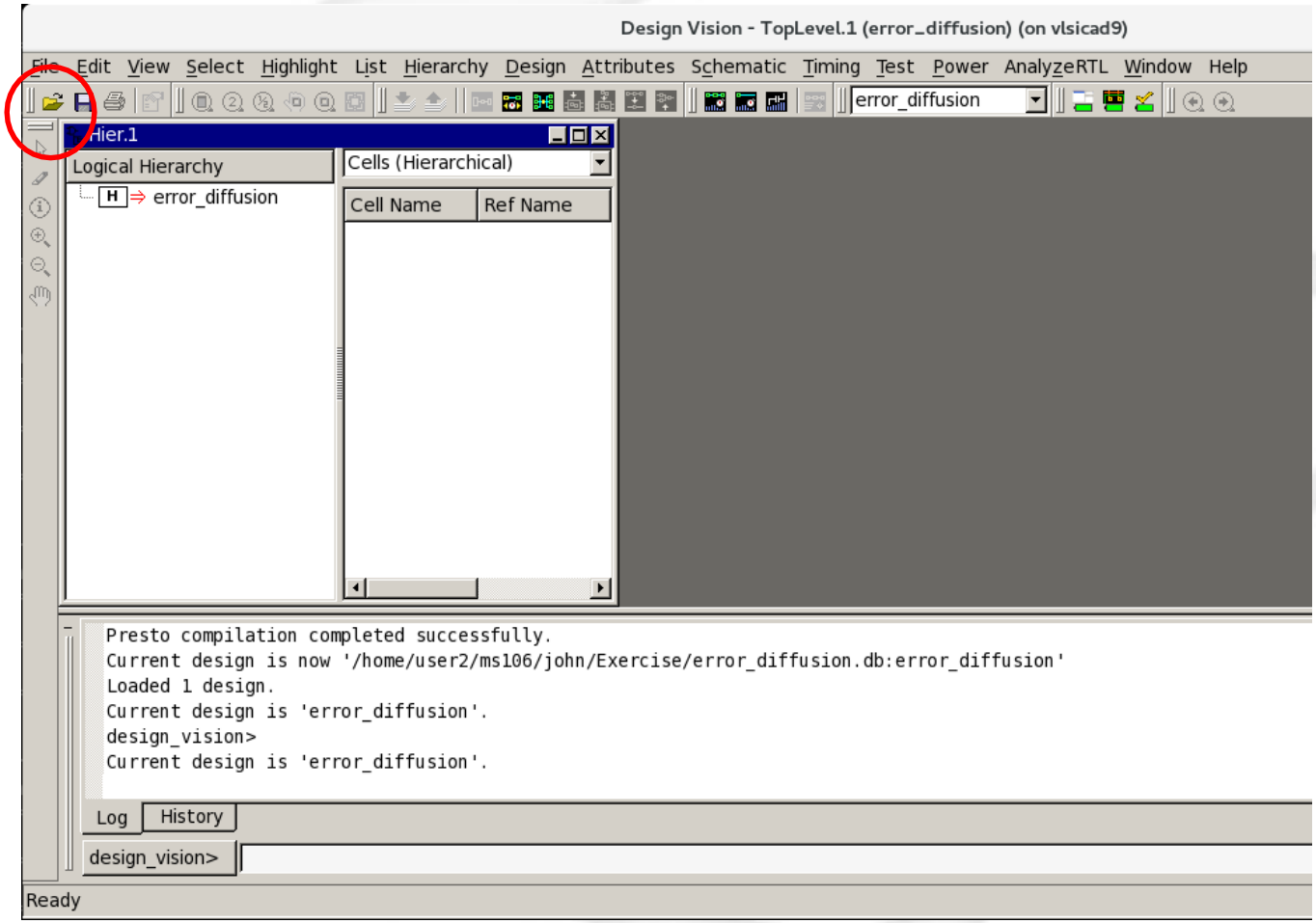
Open Design Vision

- Command: dv &



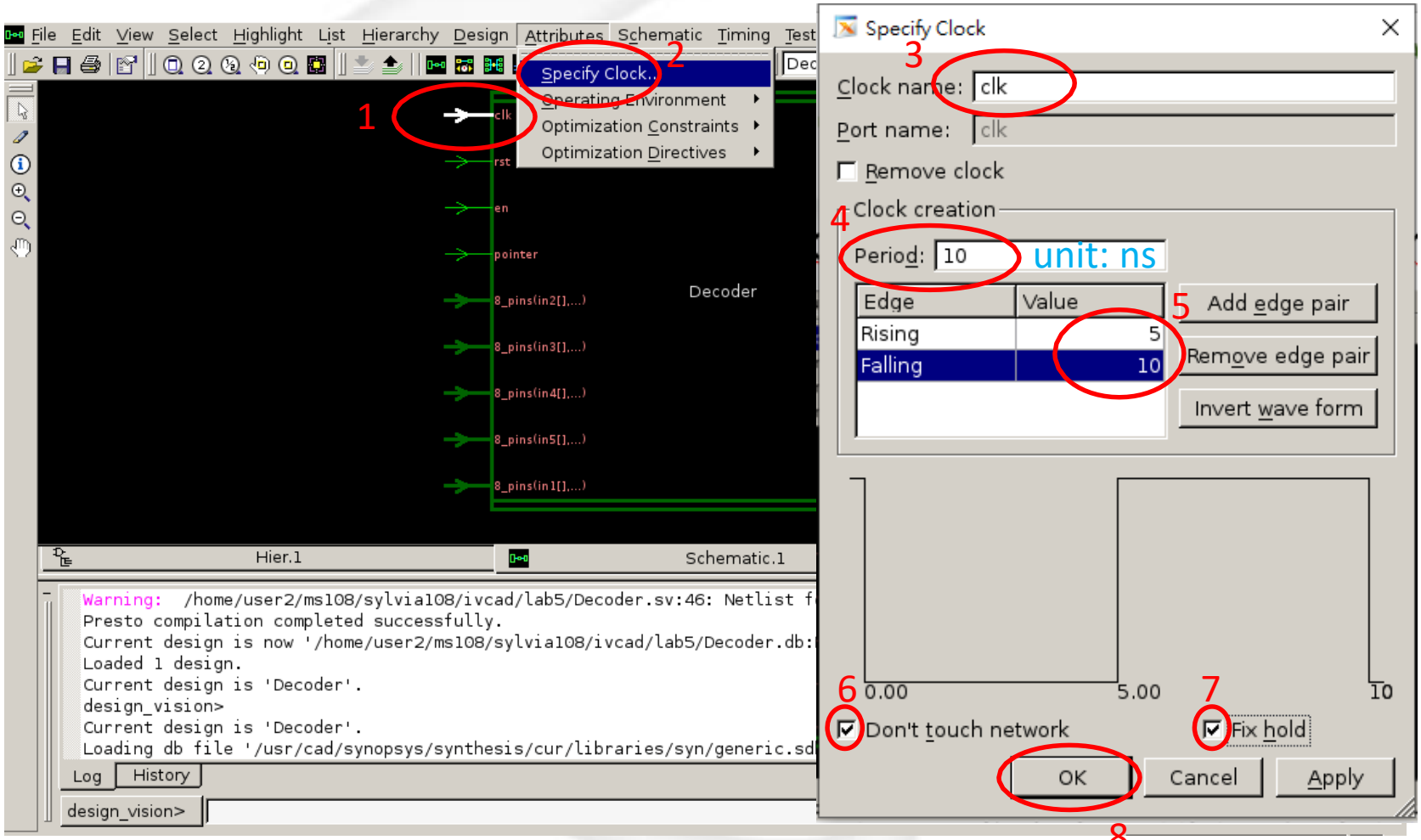
Read File

- Change hierarchy: current_design top

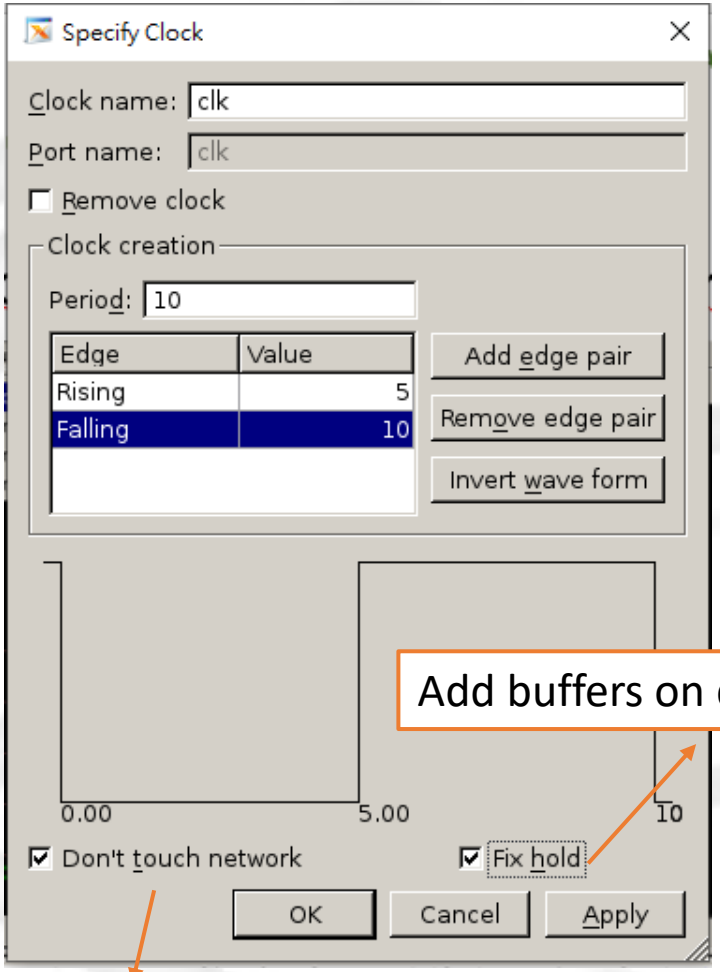


Clock Specification (1/2)

- Command: create_clock –period 10 [get_ports clk]



Clock Specification (2/2)



Add buffers on data path to satisfy hold time

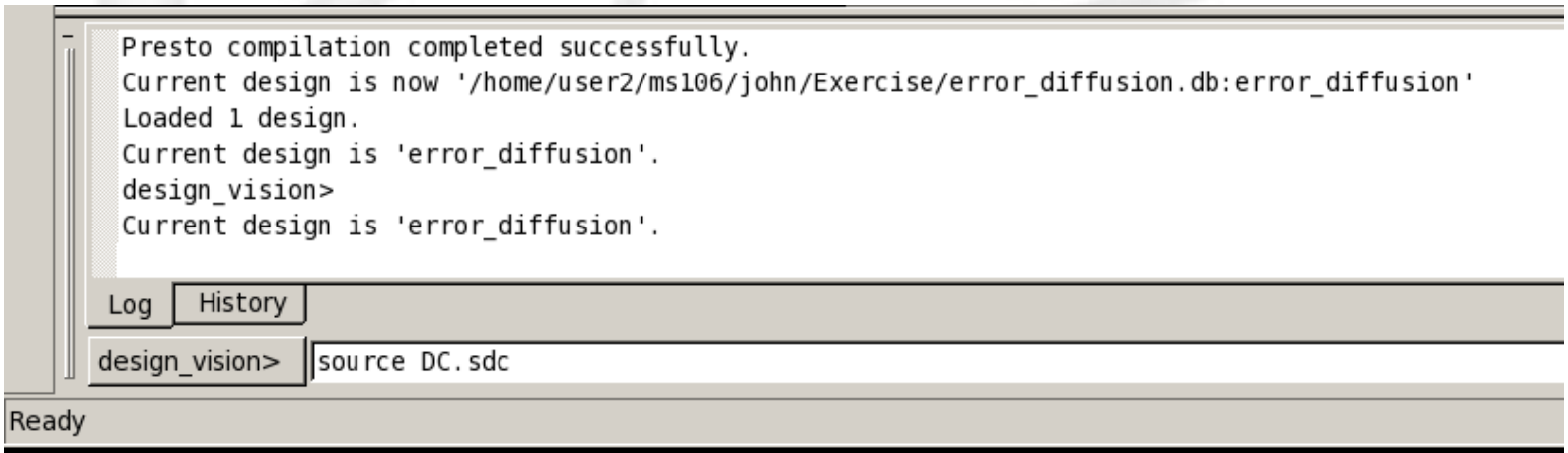
Don’t add buffers on clock path

Read Design Constraints File

- Command: source DC.sdc

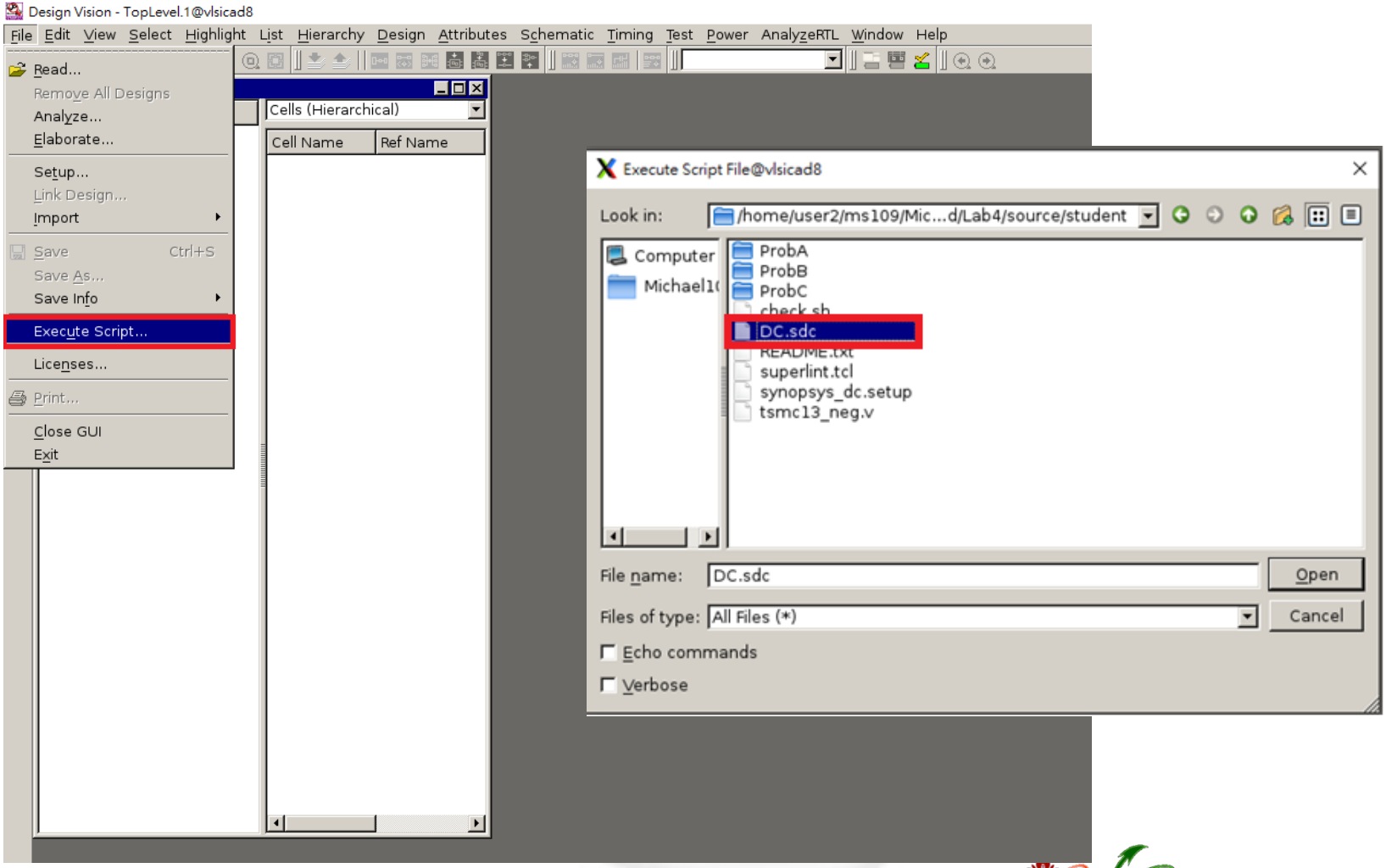
```
# operating conditions and boundary conditions #

set cycle  10.0        ;#clock period defined by designer

create_clock -period $cycle [get_ports  clk]
set_dont_touch_network      [get_clocks clk]
set_clock_uncertainty  0.1  [get_clocks clk]
set_clock_latency      0.5  [get_clocks clk]

set_input_delay  5      -clock clk [remove_from_collection [all_inputs] [get_ports clk]]
set_output_delay 0.5    -clock clk [all_outputs]
set_load         0.1     [all_outputs]
set_drive        1      [all_inputs]

set_operating_conditions  -max slow  -min fast
set_wire_load_model -name tsmc13_wl10 -library slow

set_max_fanout 20 [all_inputs]
```

DC.sdc

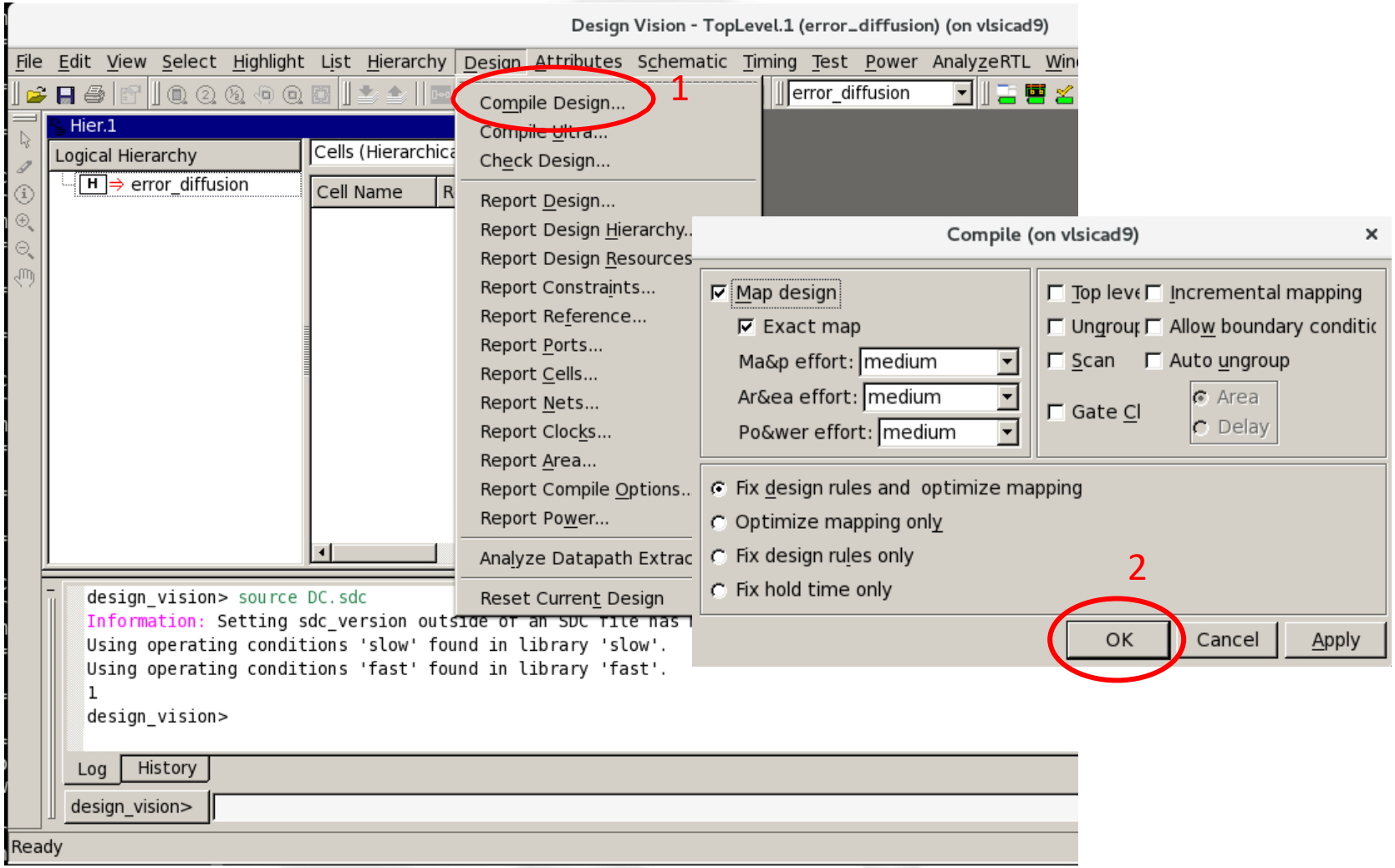


Read Design Constraints File

- File -> Execute Script File

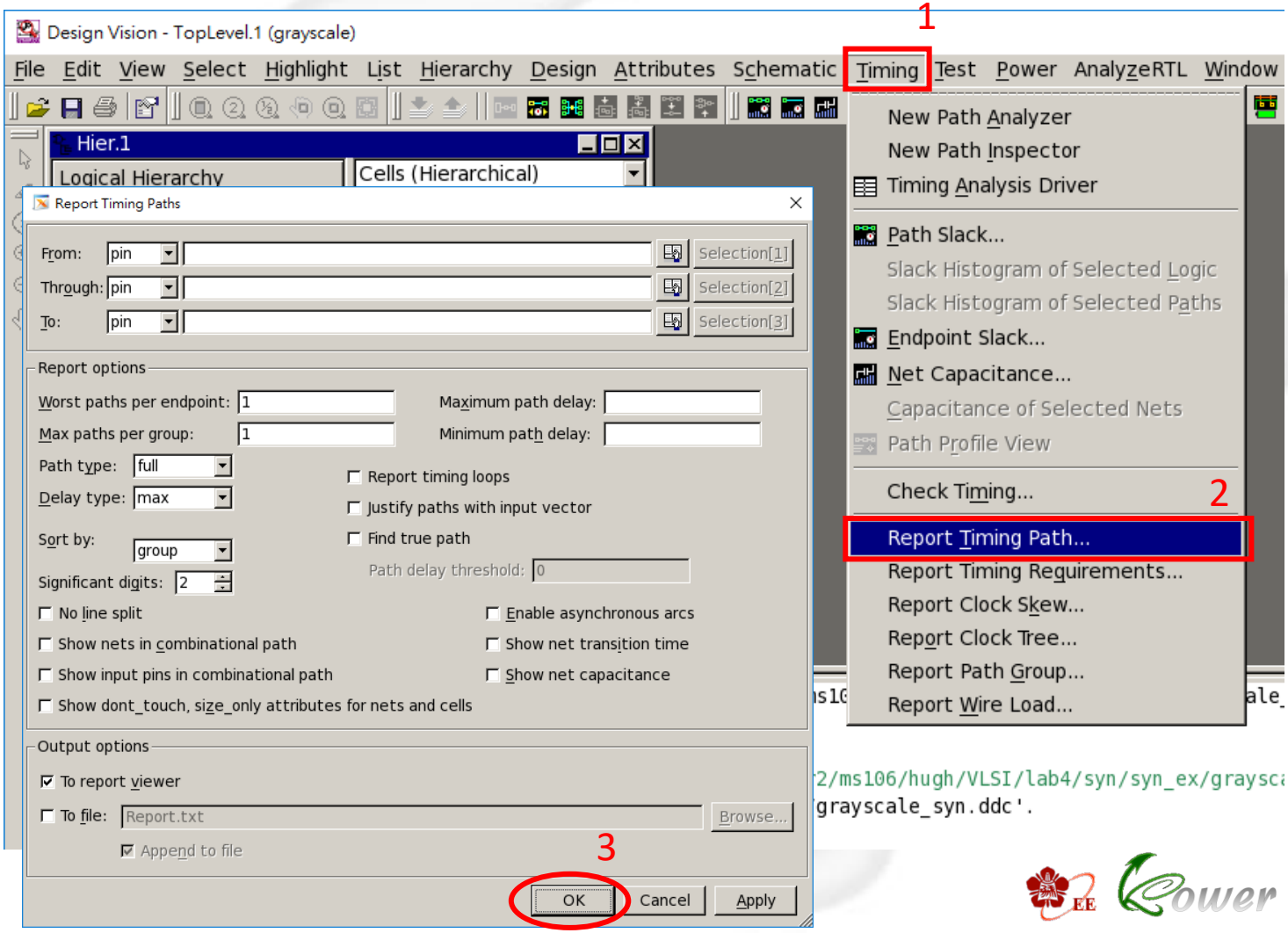


Compile Design



Report Timing (1/2)

- Command: report_timing



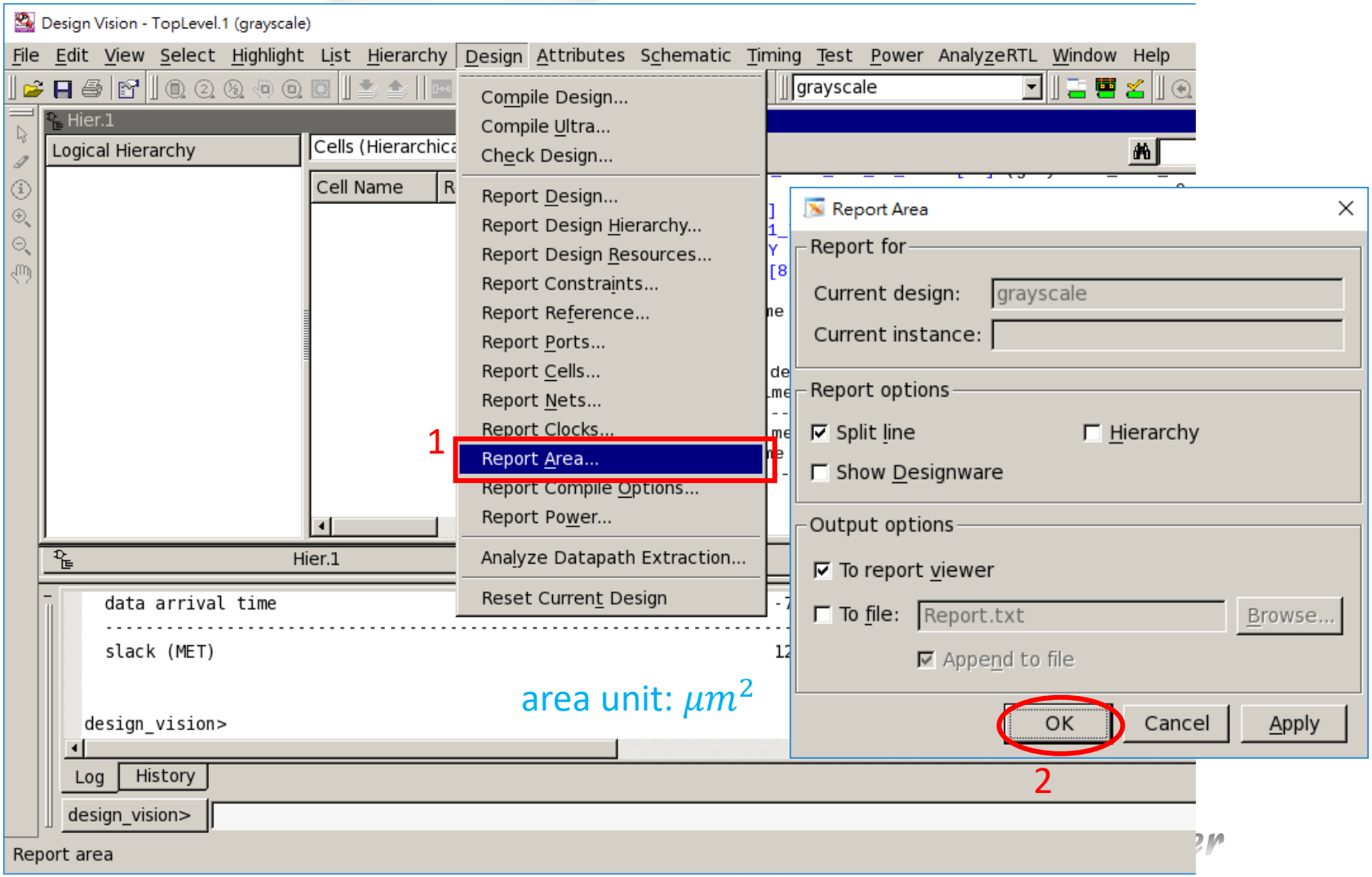
Report Timing (2/2)

- Slack must no less than 0, or it will lead to timing violation

Point	Incr	Path
clock clk (rise edge)	0.00	0.00
clock network delay (ideal)	0.50	0.50
input external delay	5.00	5.50 f
d[1] (in)	0.06	5.56 f
U78/Y (XOR2X1)	0.31	5.87 r
add_0_root_add_0_root_add_31_2/A[2] (grayscale_DW01_add_0)	0.00	5.87 r
add_0_root_add_0_root_add_31_2/U8/Y (OA21X4)	0.22	6.09 r
add_0_root_add_0_root_add_31_2/U9/Y (AO21XL)	0.47	6.56 r
add_0_root_add_0_root_add_31_2/U3/Y (NOR2X1)	0.21	6.77 f
add_0_root_add_0_root_add_31_2/U2/Y (OR2X1)	0.33	7.10 f
add_0_root_add_0_root_add_31_2/U12/Y (OAI2BB1X2)	0.13	7.23 r
add_0_root_add_0_root_add_31_2/U1_4/CO (ADDFHX2)	0.21	7.44 r
add_0_root_add_0_root_add_31_2/U1_5/CO (ADDFHX2)	0.22	7.66 r
add_0_root_add_0_root_add_31_2/U1_6/CO (ADDFHX4)	0.22	7.88 r
add_0_root_add_0_root_add_31_2/U4/Y (XOR2X1)	0.24	8.12 r
add_0_root_add_0_root_add_31_2/U1/Y (XOR2X2)	0.31	8.43 f
add_0_root_add_0_root_add_31_2/SUM[7] (grayscale_DW01_add_0)	0.00	8.43 f
add_39/A[2] (grayscale_DW01_inc_0)	0.00	8.43 f
add_39/U1_1_2/CO (ADDHX1)	0.34	8.77 f
add_39/U3/Y (AND2X2)	0.28	9.06 f
add_39/U6/Y (AND2X4)	0.20	9.26 f
add_39/U2/Y (AND2X2)	0.25	9.50 f
add_39/U13/Y (NAND2X1)	0.20	9.70 r
add_39/U1/Y (XOR2X2)	0.20	9.90 r
add_39/SUM[7] (grayscale_DW01_inc_0)	0.00	9.90 r
U40/Y (AOI22X2)	0.17	10.07 f
U39/Y (OAI21X2)	0.15	10.22 r
q_reg[7]/D (DFFRX2)	0.00	10.22 r
data arrival time		10.22
clock clk (rise edge)	10.00	10.00
clock network delay (ideal)	0.50	10.50
clock uncertainty	-0.10	10.40
q_reg[7]/CK (DFFRX2)	0.00	10.40 r
library setup time	-0.18	10.22
data required time		10.22
data required time		10.22
data arrival time		-10.22
slack (MET)		0.00

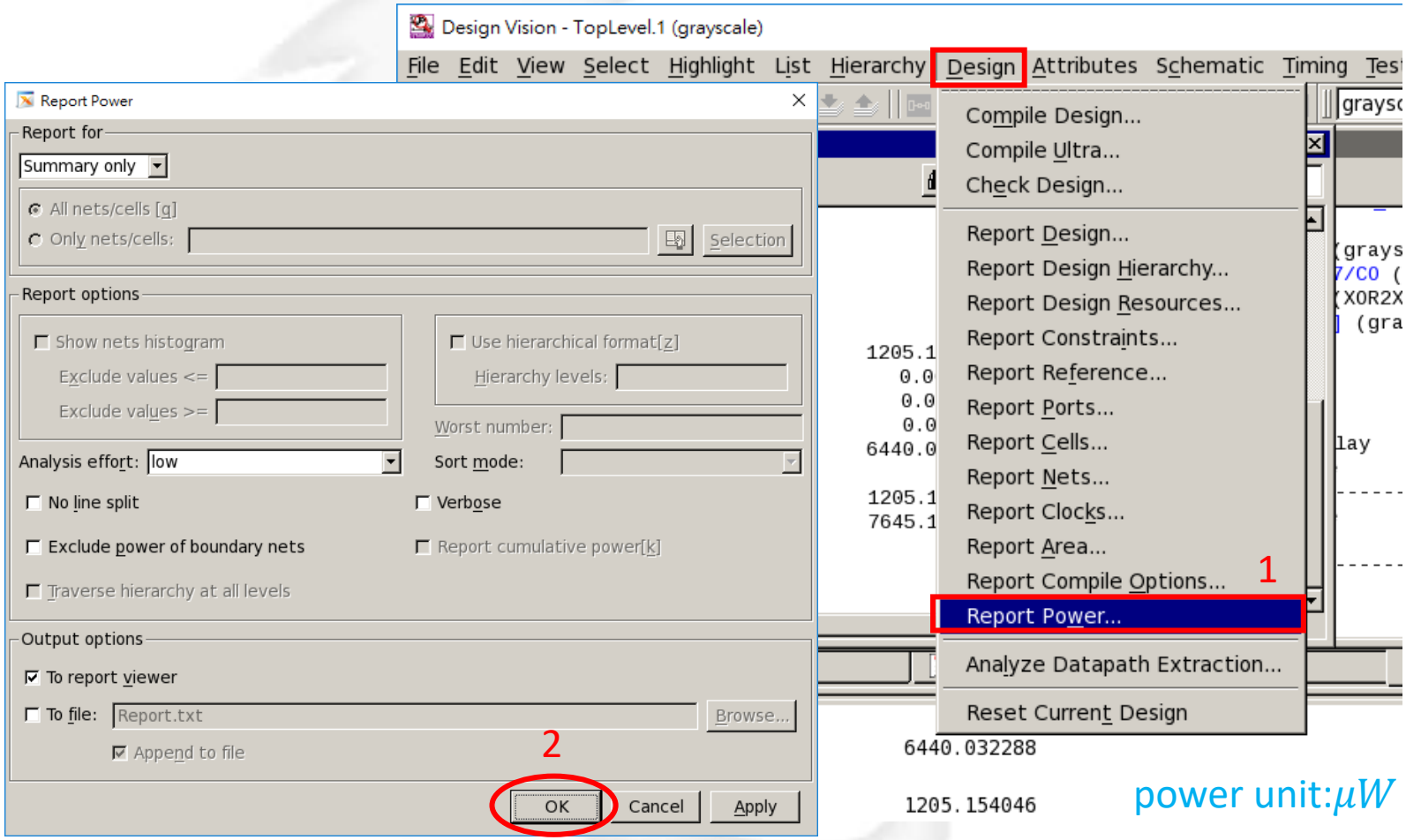
Report Area

- Command: report_area



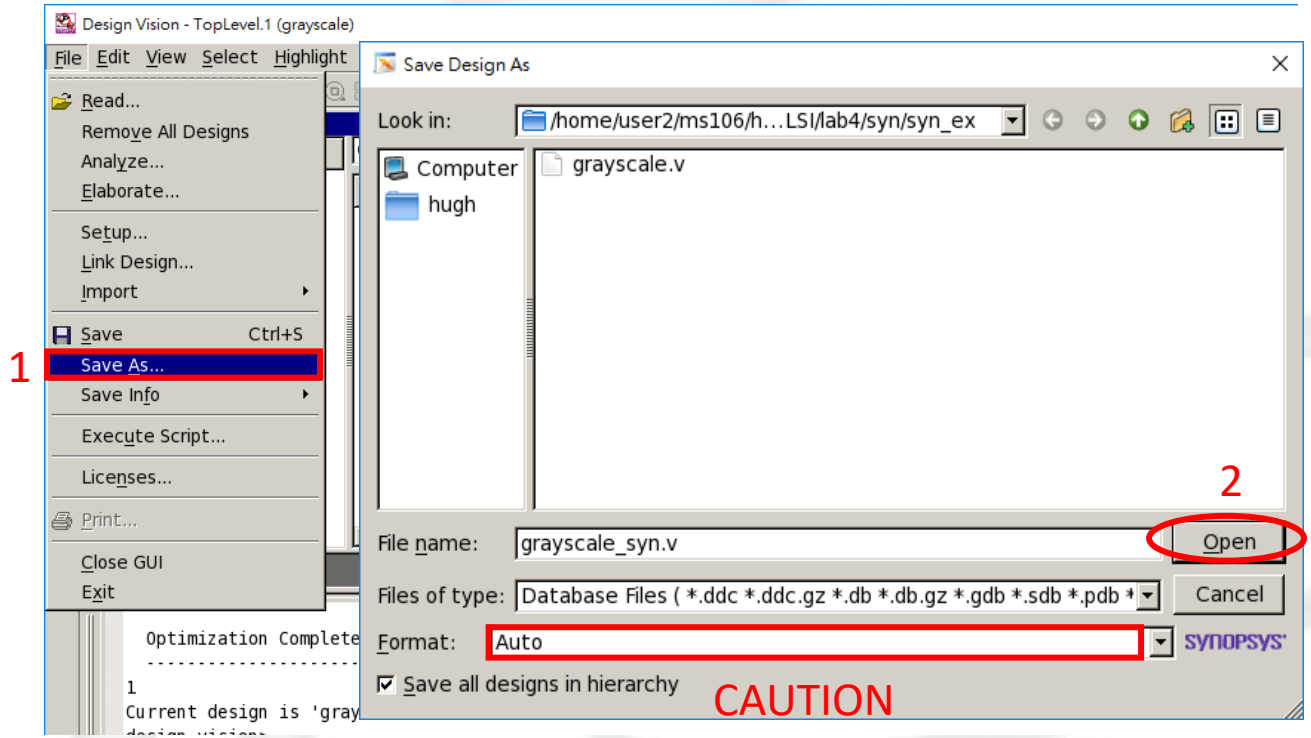
Report Power

- Command: report_power



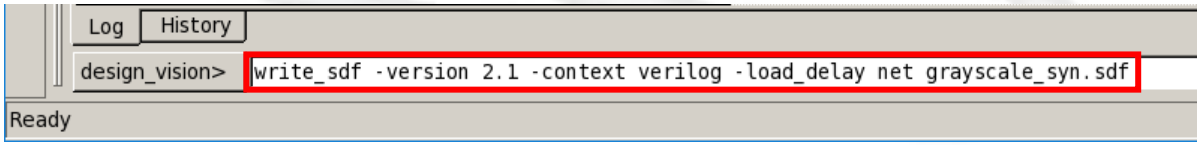
Save File

The Verilog file after synthesis



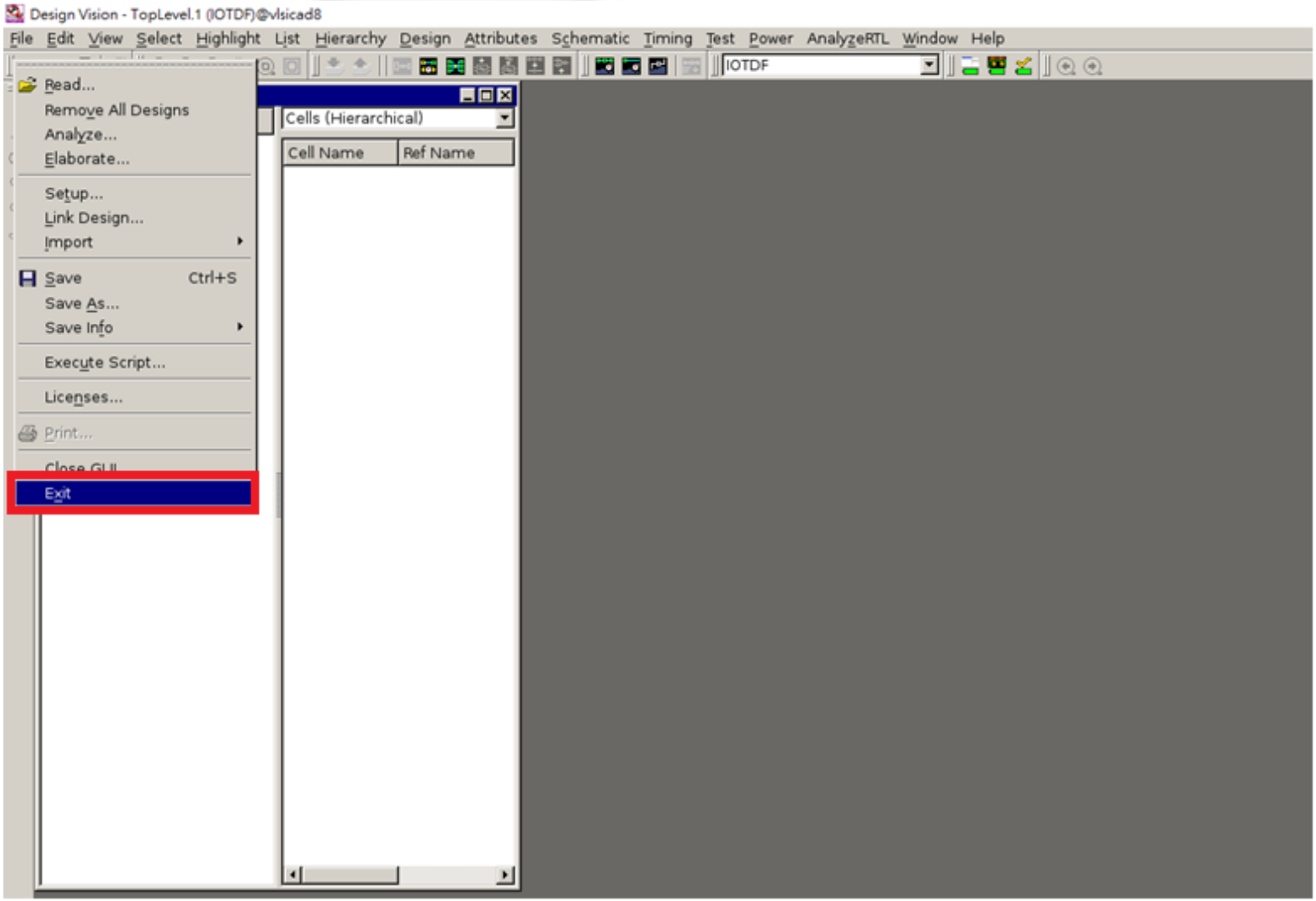
SDF File:

write_sdf -version 2.1 -context verilog -load_delay net grayscale_syn.sdf



Close Design Vision

- File -> Exit



Outline

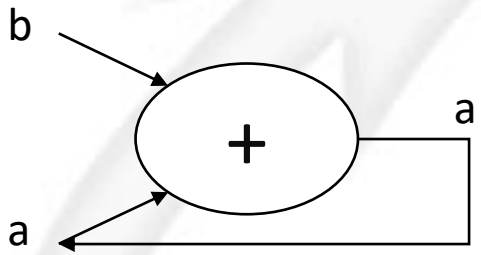
- Synthesis steps
- Synthesis tips
- Lab A : Register File
- Lab B : Convolution and activation function
- Homework

Multiplication and Modulus operator

- $x \times 0.75 = x \times 0.5 + x \times 0.25$
  - Verilog code: $x * 0.75$ (not synthesizable)
  - $(x \gg 1) + (x \gg 2)$
- $x \ mod \ 8$
  - Verilog code: x % 8 (not synthesizable)
  - X[2:0]

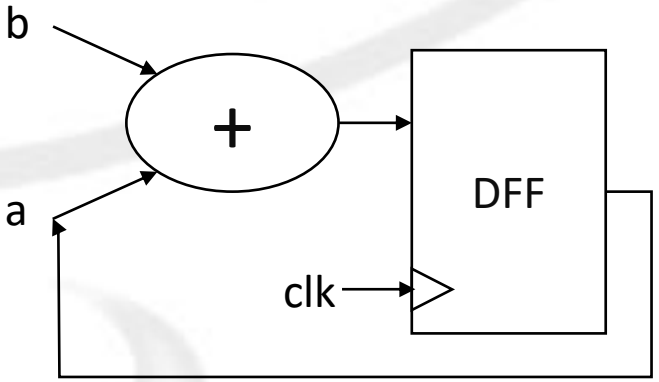
Logic loop

- assign a = b + a;
  - ➔ Static timing analyzer cannot resolve (not synthesizable)



- Change to Sequential circuit

```
reg a;

always@(posedge clk)begin
    a <= b + a;
end
```

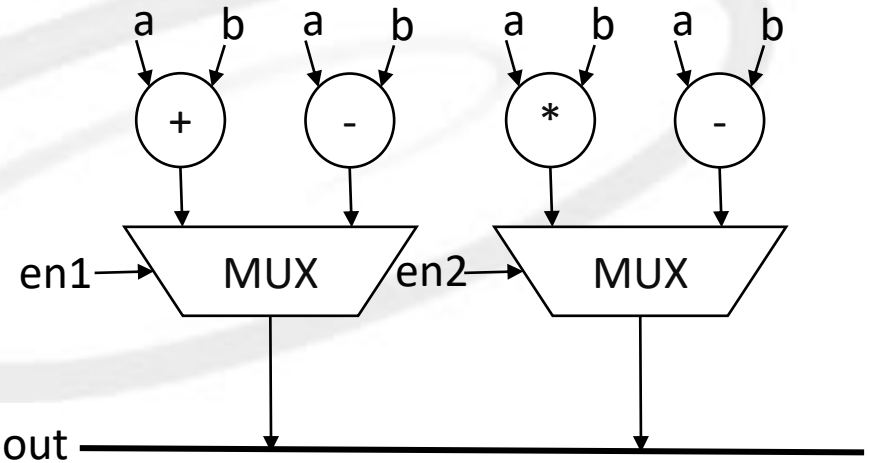


Multi-driven output

- Write same output in the same always block

```
2: Error:
/home/user2/ms108/sylvia108/ivcad/lab5/multi_driven.v:11: Net
'out' or a directly connected net is driven by more than one source,
and not all drivers are three-state. (ELAB-366)
```

```
module top(a,b,en1,en2,out);
input a, b, en1, en2;
output reg out;

always@(*)begin
    if(en1) out = a+b;
    else out = a-b;
end

always@(*)begin
   if(en2) out = a*b;
   else out = a-b;
end

endmodule
```




```
always@(*)begin
    if(en1) out = a+b;
    else if(en2) out = a*b;
    else out = a-b;
end
```

Latch Inference (1/2)

- If inputs do not arrive at the same time, latch will lock the value
- Write full case of combinational circuit

Case1: no else case

```
module top(a,b,en1,out);

input a,b,en1;
output reg out;

always@(*)begin
  if(en1)
    out = a | b;
end

endmodule
```

add ”else”

Case2: no default

```
module top(a,b,en1,out);
input a,b;
input [1:0] en1;
output reg [2:0] out;

always@(*)begin
  case(en1)
      2'd0: out = 3'b001;
      2'd1: out = 3'b010;
      2'd2: out = 3'b100;
  endcase
end
endmodule
```

add ”default”

Register Name	Type	Width	Bus	MB	AR	AS	SR	SS	ST
out_reg	Latch	3	Y	N	N	N	-	-	-

Latch Inference (2/2)

- Give information to all signals which in same block, even though some signals do not change

```
always@(*)begin
  if(state == `GREEN)begin
     green = 1'b1;
     yellow = 1'b0;
     red = 1'b0;
  end
  else if(state == `YELLOW)begin
     green = 1'b0;
     yellow = 1'b1;
     //red = 1'b0;
  end
  else if(state == `RED)begin
     //green = 1'b0;
     yellow = 1'b0;
     red = 1'b1;
  end
  else begin
     green = 1'b0;
     yellow = 1'b0;
     red = 1'b0;
  end
end
```

Register Name	Type	Width
red_reg	Latch	1
green_reg	Latch	1

Coding style

- Sequential circuit and combinational circuit should NOT write in same block!
- Combinational circuit do NOT need reset signal.

```
module seq_comb(clk,rst,a,b,result,addr);

  input clk,rst;
  input [2:0] a,b;
  output reg [3:0] result;
  output reg [2:0] addr;

  always@(posedge clk or posedge rst)begin
    if(rst) begin
         result = 3'd0;
         addr <= 3'd0;
    end
    else begin
        result = a+b;
        addr <= addr + 3'd1;
    end
  end

endmodule
```

```
module seq_comb(clk,rst,a,b,result,addr);

  input clk,rst;
  input [2:0] a,b;
  output reg [3:0] result;
  output reg [2:0] addr;

  always@(*)begin
      result = a+b;
  end

  always@(posedge clk or posedge rst)begin
    if(rst) addr <= 3'd0;
    else    addr <= addr + 3'd1;
  end

endmodule
```

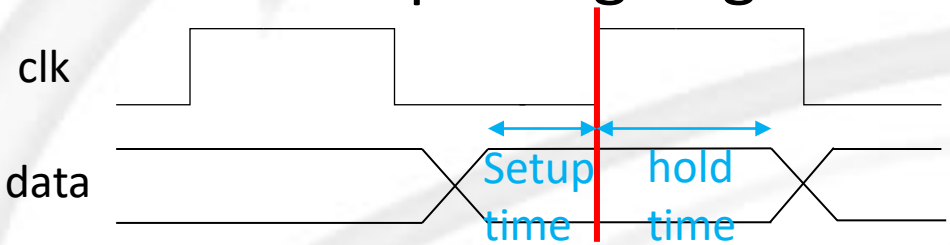
Combinational circuit

Sequential circuit

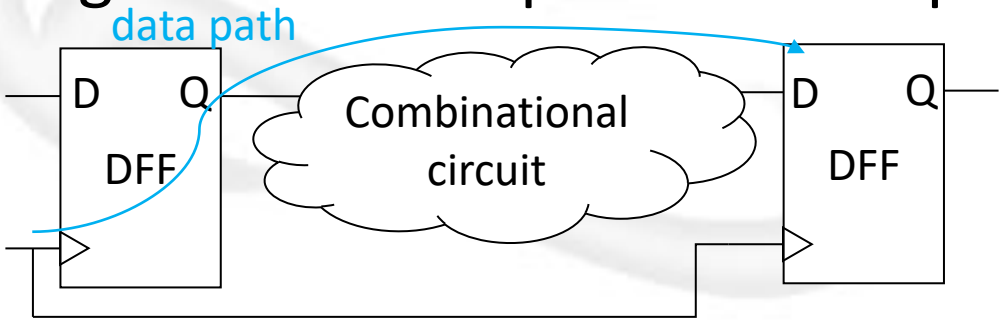
```
=============================================
|    Register Name    |   Type    | Width |
=============================================
|     result_reg      | Flip-flop |   4   |
|      addr_reg       | Flip-flop |   3   |
=============================================
```

Timing (1/2)

- Setup time: the amount of time that data should be stable before the capturing edge of clock
- Hold time: the amount of time that data should be stable after the capturing edge of clock



- Data path delay = setup time + combinational circuit delay + hold time
- Timing slack = clock period - data path delay



Timing (2/2)

- If timing slack is less than 0, it will appear timing violation when post-simulation

```
Warning!  Timing violation
          $setuphold<hold>( negedge G &&& (SandR == 1):19665833306 PS, posedge D:196658
           File: ./tsmc18.v, line = 23811
          Scope: top_tb.top.ctrl.\out_mem_addr_reg[20]
           Time: 19665833338 PS
```

- Two solutions:
  - Revise clock period to fit the design
    - DC.sdc `set cycle 15.0 ;#clock period defined by designer`
    - Testbench
      ```
      `timescale 1ns/1ps
      `define INTERVAL 15
      ```
  - Find the critical path and adjust it
    - critical path: the longest delay path

Outline

- Synthesis steps
- Synthesis tips
- Lab A : Register File
- Lab B : Convolution and activation function
- Homework

Lab A : Register File

- D Latch

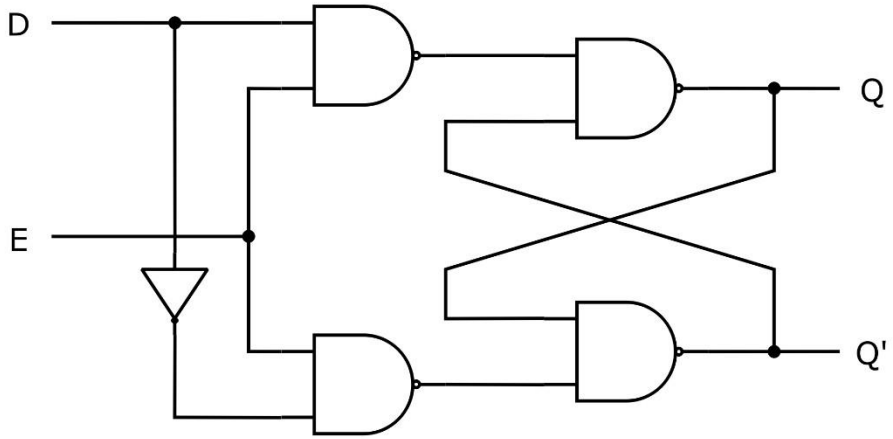
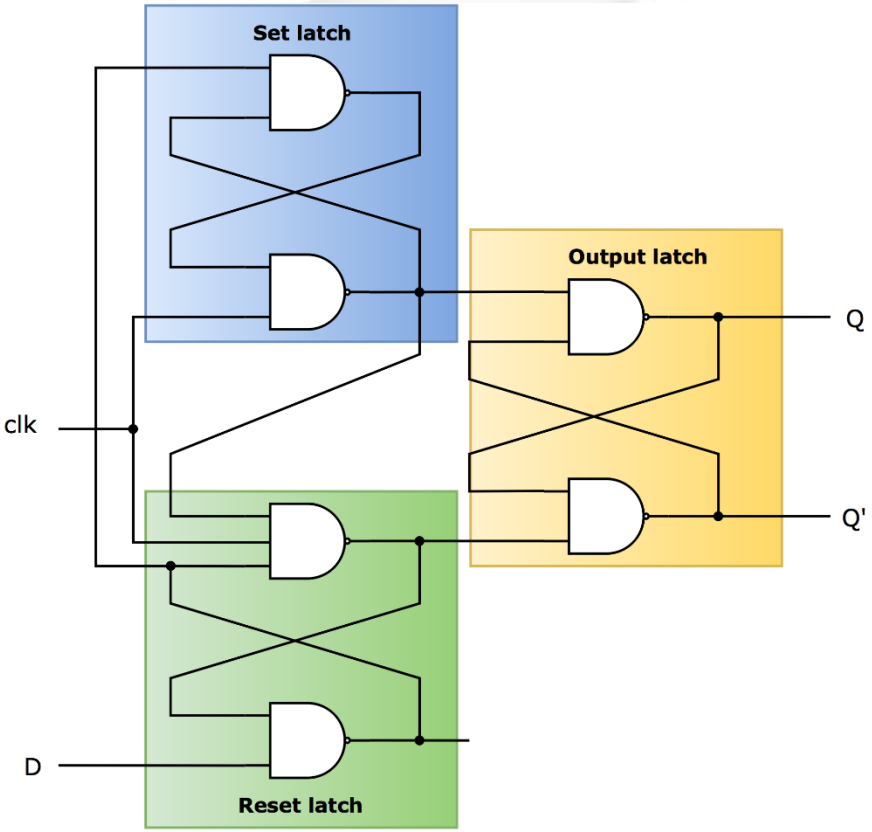


Fig. D latch with enable

E	D	Q	Q_{next}	Q'_{next}
0	x	0	0	1
0	x	1	1	0
1	0	x	0	1
1	1	x	1	0

Lab A : Register File

- D flip-flop

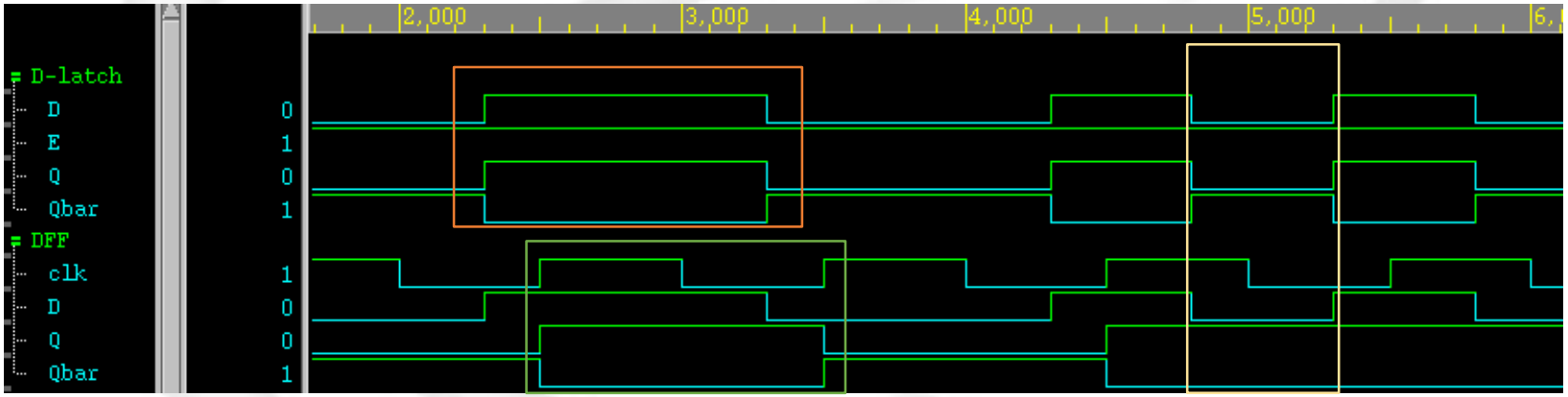


clk	D	Q	Q_{next}	Q'_{next}
0	x	0	0	1
0	x	1	1	0
1	x	0	0	1
1	x	1	1	0
↑	0	x	0	1
↑	1	x	1	0

Fig. Positive-edge-triggered D flip-fliop

Lab A : Register File

- The main difference between latch and flip-flop
  - ➔ Latch
    - Outputs are constantly affected by the inputs as long as the enable signal is asserted.
  - ➔ Flip-Flop
    - Outputs change only at the rising edge of the clock signal.



Lab A : Register File

- Register file is an array of processor registers in a CPU.
  - ➔ A 32 X 32 register file means it has 32 registers and each of them is 32 bits.
- Components
  - ➔ Decoder
  - ➔ Array
  - ➔ Multiplexer
- Multiple output ports allow us to read several data in one cycle.

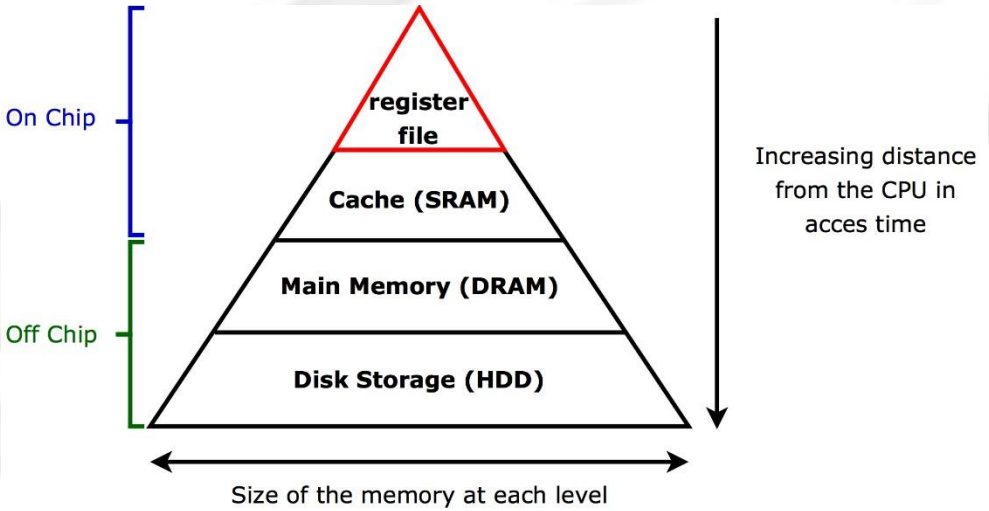


Fig. Memory Hierarchy

Lab A : Register File

- How does it work ?

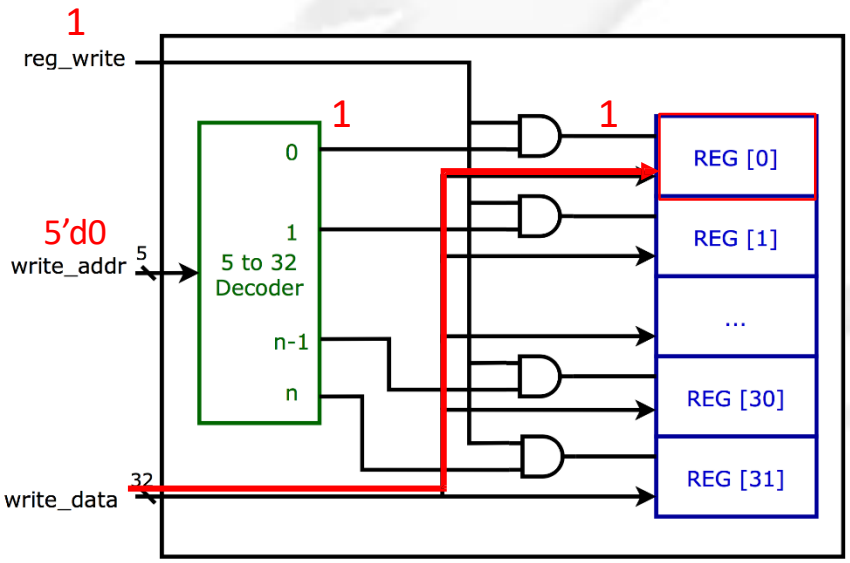


Fig. Write data into register file

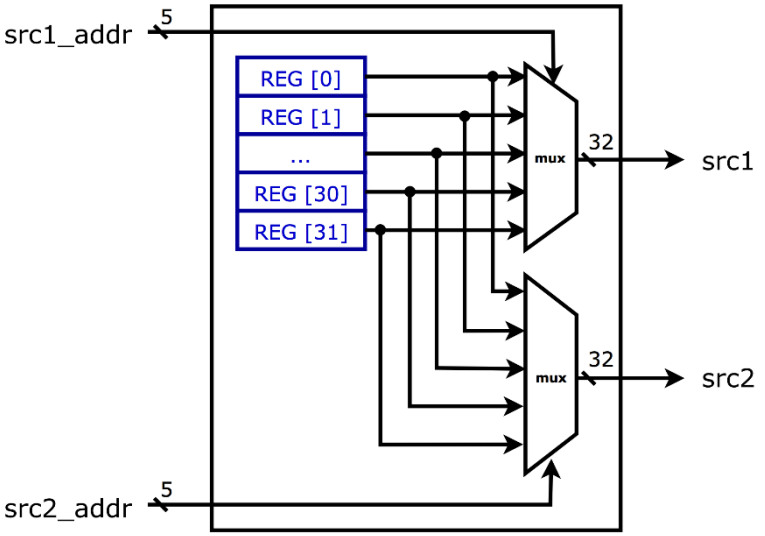
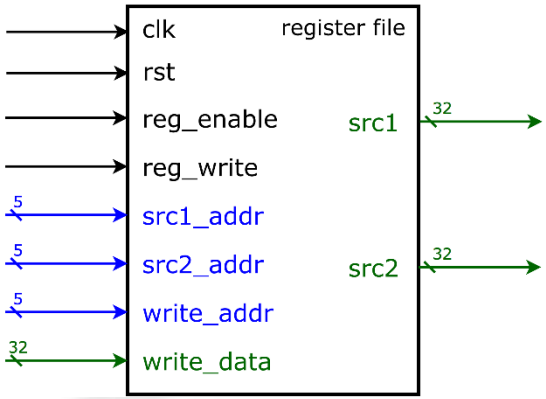


Fig. Read data from register file

Lab A : Register File

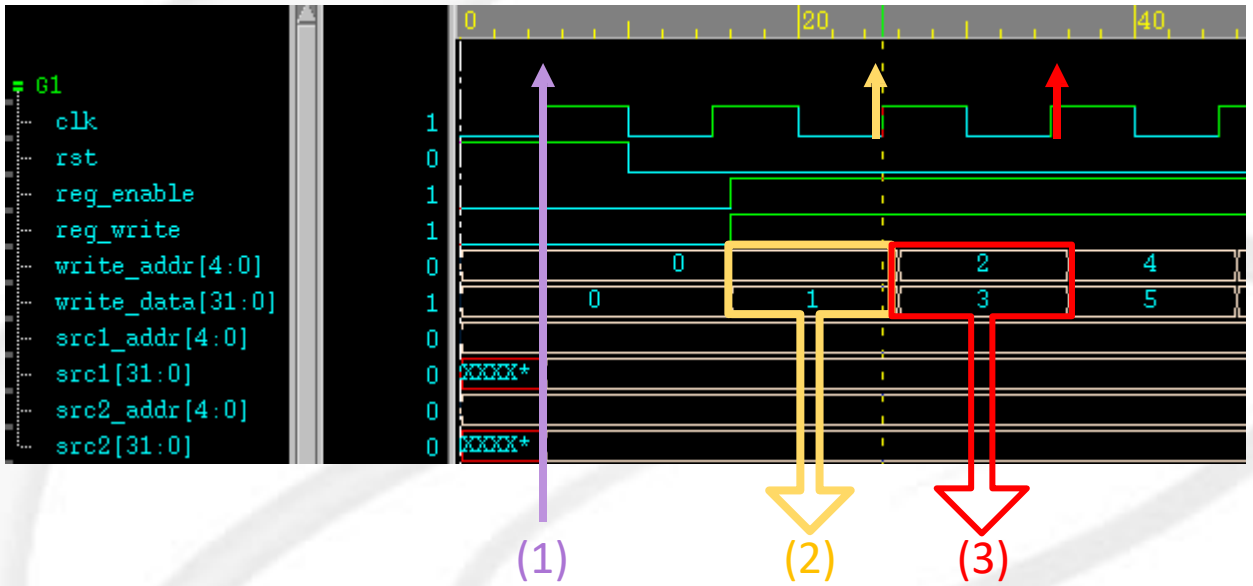
- Port List

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



Lab A : Register File

- How does it write?



(1) clock positive edge ↑, reset signal is set to 1, all registers are reset to zero

(2) clock positive edge ↑, register file is enable and in write mode

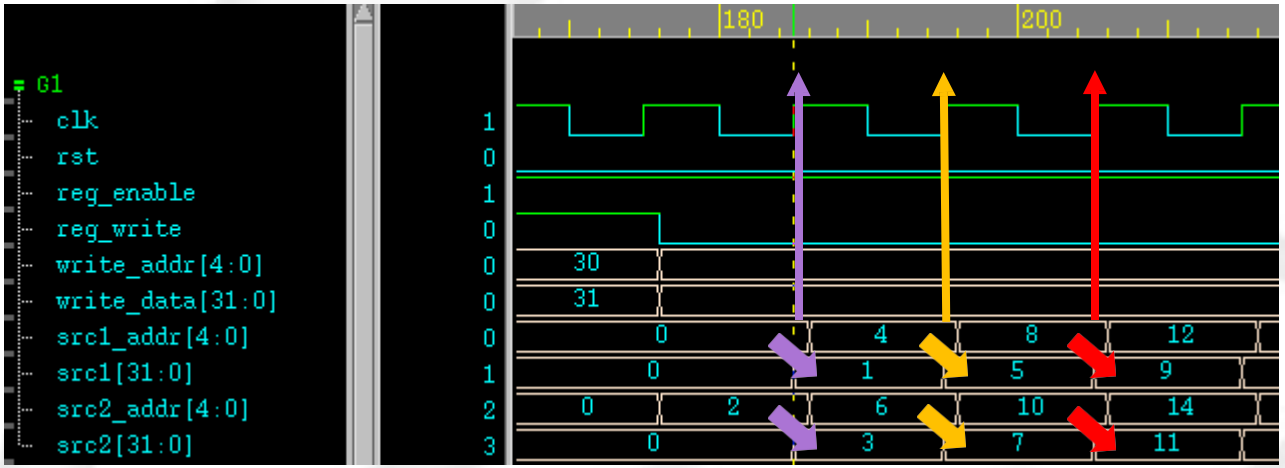
→ 32'd1 is written into REG[0]

(3) clock positive edge ↑, register file is enable and in write mode

→ 32'd3 is written into REG[2]

Lab A : Register File

- How does it read?



(1) clock positive edge ↑, register file is enable and in read mode

↑→ src1 = REG [0] = 32'd1, src2 = REG [2] = 32'd3

(2) clock positive edge ↑, register file is enable and in read mode

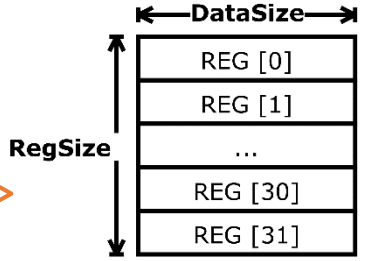
↑→ src1 = REG [4] = 32'd5, src2 = REG [6] = 32'd7

(3) clock positive edge ↑, register file is enable and in read mode

↑→ src1 = REG [8] = 32'd9, src2 = REG [10] = 32'11

Lab A : Reference Code (1/2)

```
1   `timescale 1ns/10ps
2
3   // ---------------------- define ---------------------- //
4   `define DataSize    32
5   `define RegSize     32
6   `define AddrSize    5
7
8   module regfile (clk, rst, reg_enable, reg_write, src1_addr, src2_addr,
9                   write_addr, write_data, src1, src2);
10
11  // ---------------------- input  ---------------------- //
12  input                       clk;
13  input                       rst;
14  input                       reg_enable;
15  input                       reg_write;
16  input   [`AddrSize-1:0]     src1_addr;
17  input   [`AddrSize-1:0]     src2_addr;
18  input   [`AddrSize-1:0]     write_addr;
19  input   [`DataSize-1:0]     write_data;
20
21  // ---------------------- output ---------------------- //
22  output  [`DataSize-1:0]     src1;
23  output  [`DataSize-1:0]     src2;
24
25  // ----------------------  reg   ---------------------- //
26  reg     [`DataSize-1:0]     src1;
27  reg     [`DataSize-1:0]     src2;
28  reg     [`DataSize-1:0]     REG [`RegSize-1:0];
29
30  integer i;
```



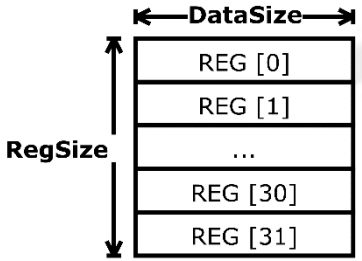
Lab A : Reference Code (2/2)

```
43  always@(posedge clk or posedge rst)
44  begin
45    if(rst)begin
46      for(i=0; i<`RegSize; i=i+1)
47        REG[i] <= 32'b0;
48      src1 <= 32'b0;
49      src2 <= 32'b0;
50    end
51    else begin
52      if(reg_enable)begin
53        if(reg_write)
54          REG[write_addr] <= write_data;
55        else begin
56          src1 <= REG[src1_addr];
57          src2 <= REG[src2_addr];
58        end
59      end
60      else begin
61        src1 <= 32'b0;
62        src2 <= 32'b0;
63      end
64    end
65  end
66  endmodule
```

Using for loop to reset the register array

register write

register read



Outline

- Synthesis steps
- Synthesis tips
- Lab A : Register File
- Lab B : Convolution and activation function
- Homework

Lab B : Convolution and activation function

- Convolution is a mathematical operation that does the integral of the product of two functions, that produces a third function.
- It's an important concept on neural network

$$(f * g)(n) = \int_{-\infty}^{\infty} f(\tau)g(n-\tau)d\tau$$

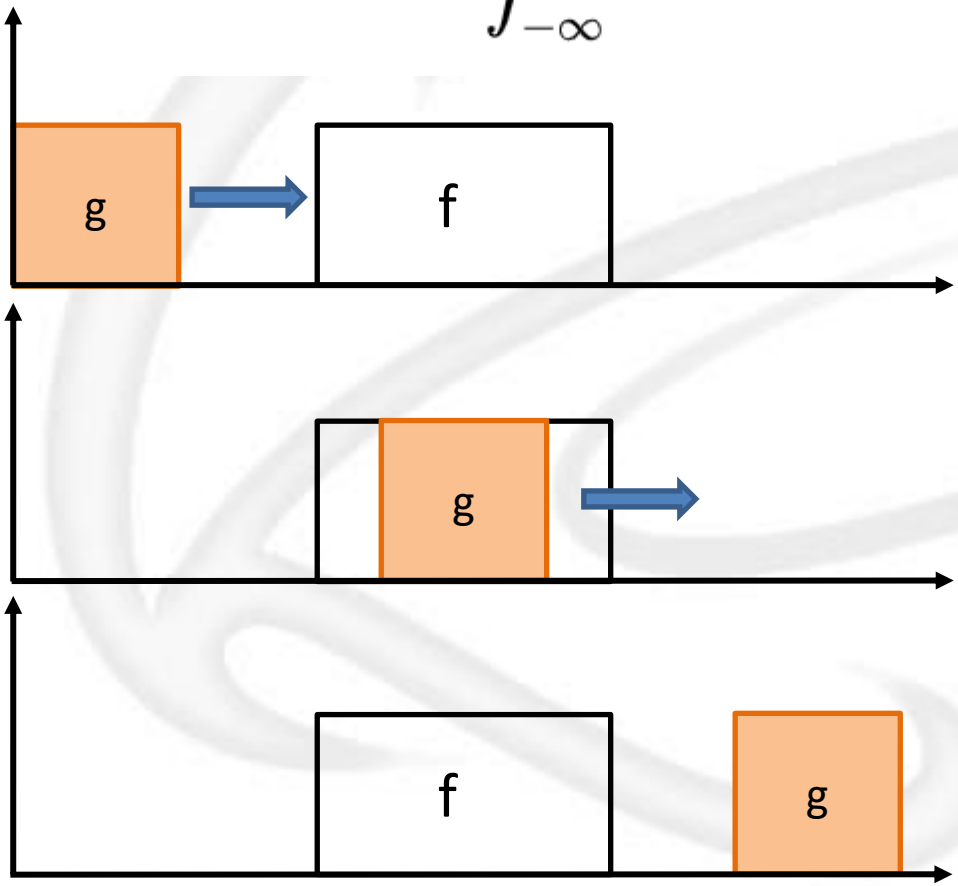
Discrete convolution :

$$(f * g)(n) = \sum_{\tau=-\infty}^{\infty} f(\tau)g(n-\tau)$$

Lab B : Convolution and activation function

- Convolution procedure

$$(f * g)(n) = \int_{-\infty}^{\infty} f(\tau) g(n - \tau) d\tau$$



Lab B : Convolution and activation function

- In CNN application, we use input feature map and weight as $f(x)$ and $g(x)$

Input feature map

p0,0	p0,1	p0,2	p0,3	p0,4	p0,5	p0,6	p0,7
p1,0	p1,1	p1,2	p1,3	p1,4	p1,5	p1,6	p1,7
p2,0	p2,1	p2,2	p2,3	p2,4	p2,5	p2,6	p2,7
p3,0	p3,1	p3,2	p3,3	p3,4	p3,5	p3,6	p3,7
⋮							

weight

w0	w1	w2
w3	w4	w5
w6	w7	w8

Lab B : Convolution and activation function

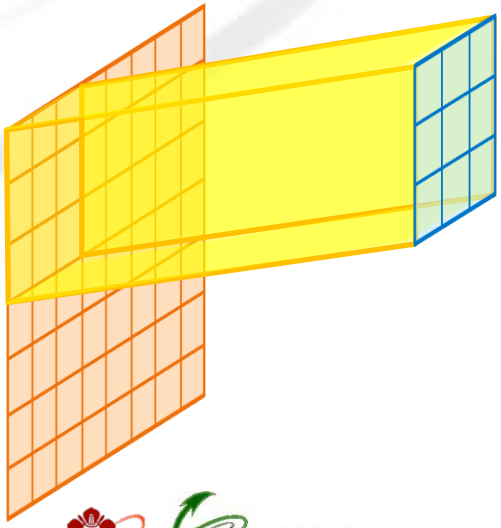
- Each “new pixel” is the sum of “old pixel” multiply with the corresponding weight.
- EX: When k = 0, n=0

weight

w0	w1	w2
w3	w4	w5
w6	w7	w8

Input feature map

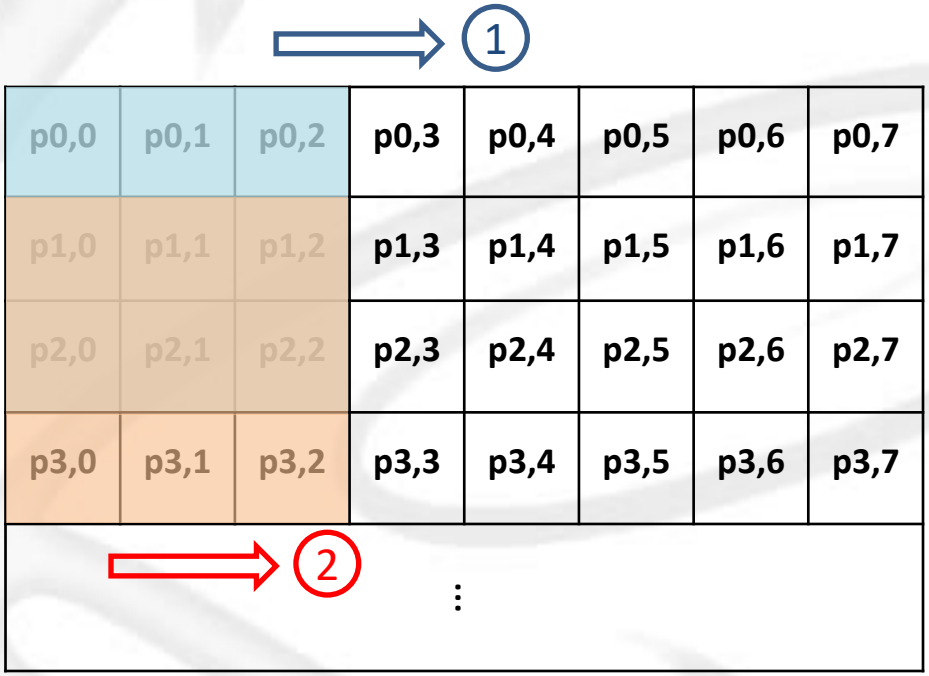
p0,0	p0,1	p0,2	p0,3	p0,4	p0,5	p0,6	p0,7
p1,0	p1,1	p1,2	p1,3	p1,4	p1,5	p1,6	p1,7
p2,0	p2,1	p2,2	p2,3	p2,4	p2,5	p2,6	p2,7
p3,0	p3,1	p3,2	p3,3	p3,4	p3,5	p3,6	p3,7
⋮							



$$\text{new } p_{0,0} = p_{0,0} * w_0 + p_{0,1} * w_1 + p_{0,2} * w_2 + p_{1,0} * w_3 + p_{1,1} * w_4 + p_{1,2} * w_5 + p_{2,0} * w_6 + p_{2,1} * w_7 + p_{2,2} * w_8$$

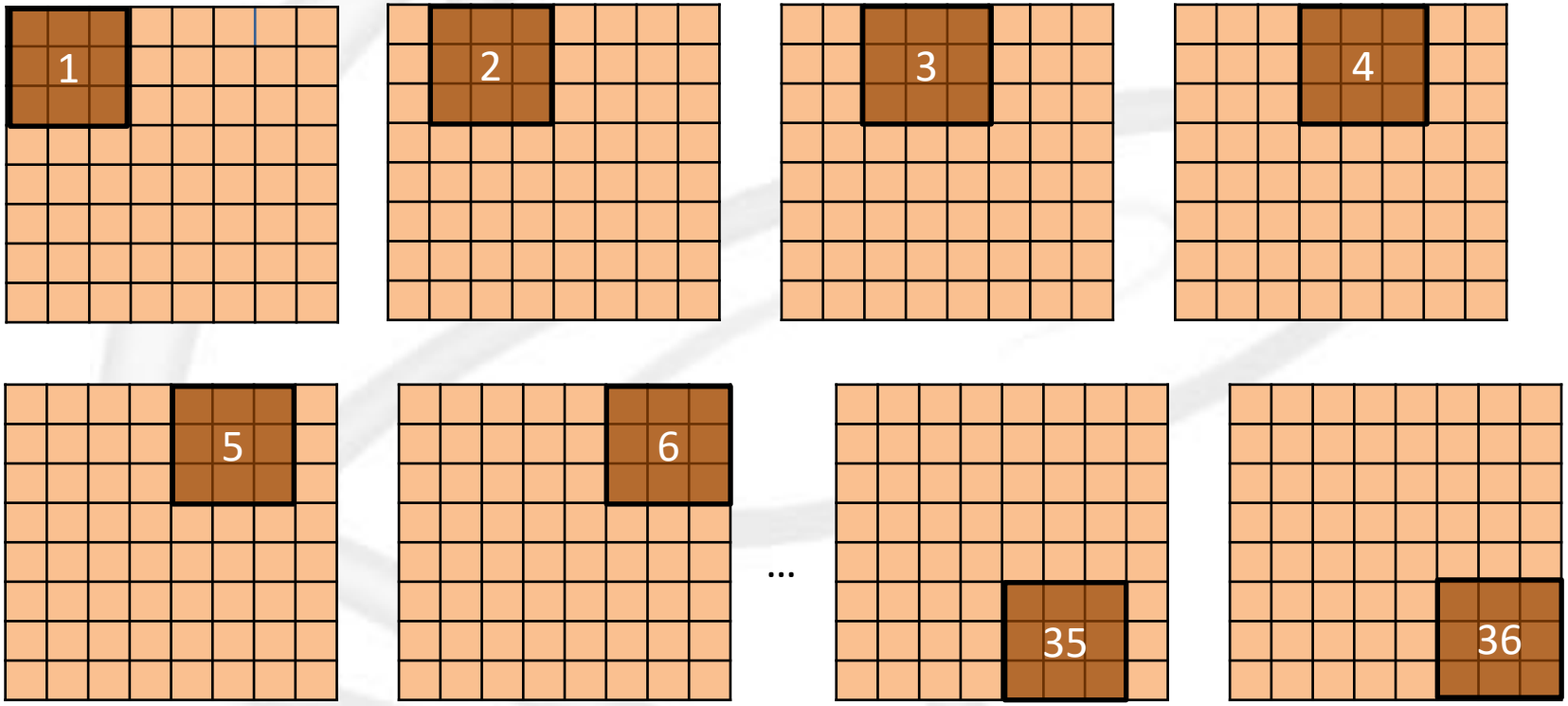
Lab B : Convolution and activation function

- Weight filter moves to the right until the complete width, then jumps to the next row of beginning column, and repeat the process.



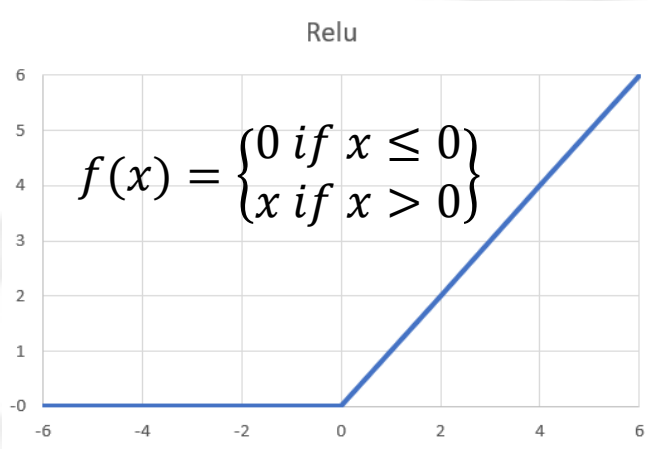
Lab B : Convolution and activation function

- If the input image size is $n * n$, then the output image size will be $(n-2) * (n-2)$
- Ex: input image = 8*8, then output image = 6*6



Lab B : Convolution and activation function

- Activation functions determine the output of each node.
- It can limit the data range to fit user's requirement.
- It can make the operation nonlinear.
- "Relu" is one of the most common activation function.

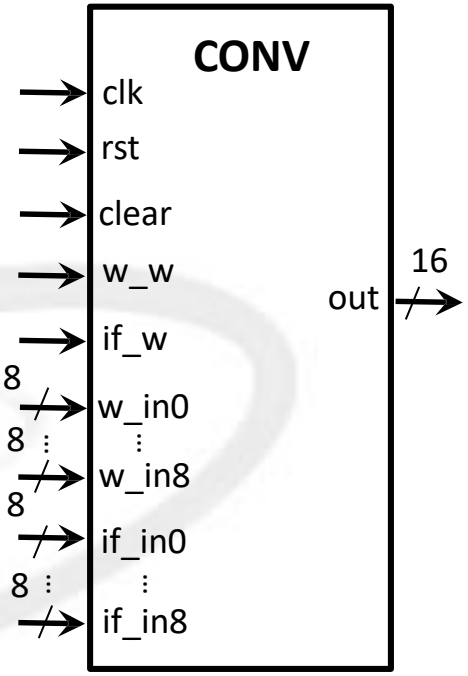


Lab B : Convolution and activation function

- Port list

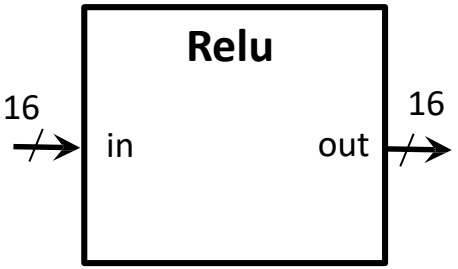
CONV

Signal	Type	Bits	Description
clk	input	1	Clock pin.
rst	input	1	Reset pin.
clear	input	1	Set all registers to 0.
w_w	input	1	Write weight enable. When w_w is high, then w_in0~8 is available.
if_w	input	1	Write input feature map enable. When if_w is high, then if_in0~8 is available.
w_in0~8	input	8 each	Input weight data.
if_in0~8	input	8 each	Input input feature map data.
out	output	16	Output data.



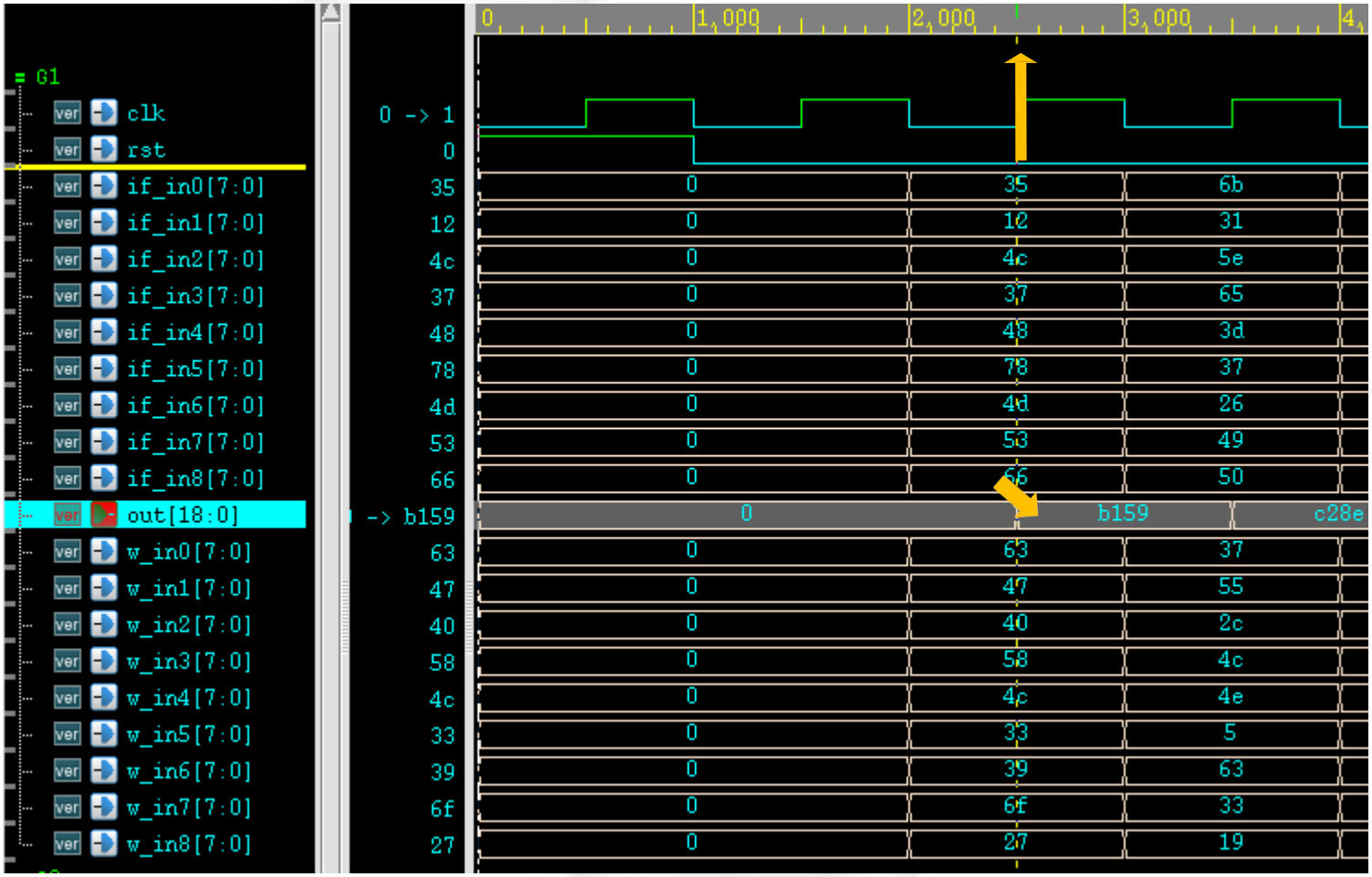
Relu

Signal	Type	Bits	Description
in	input	16	Input data.
out	output	16	Output data.



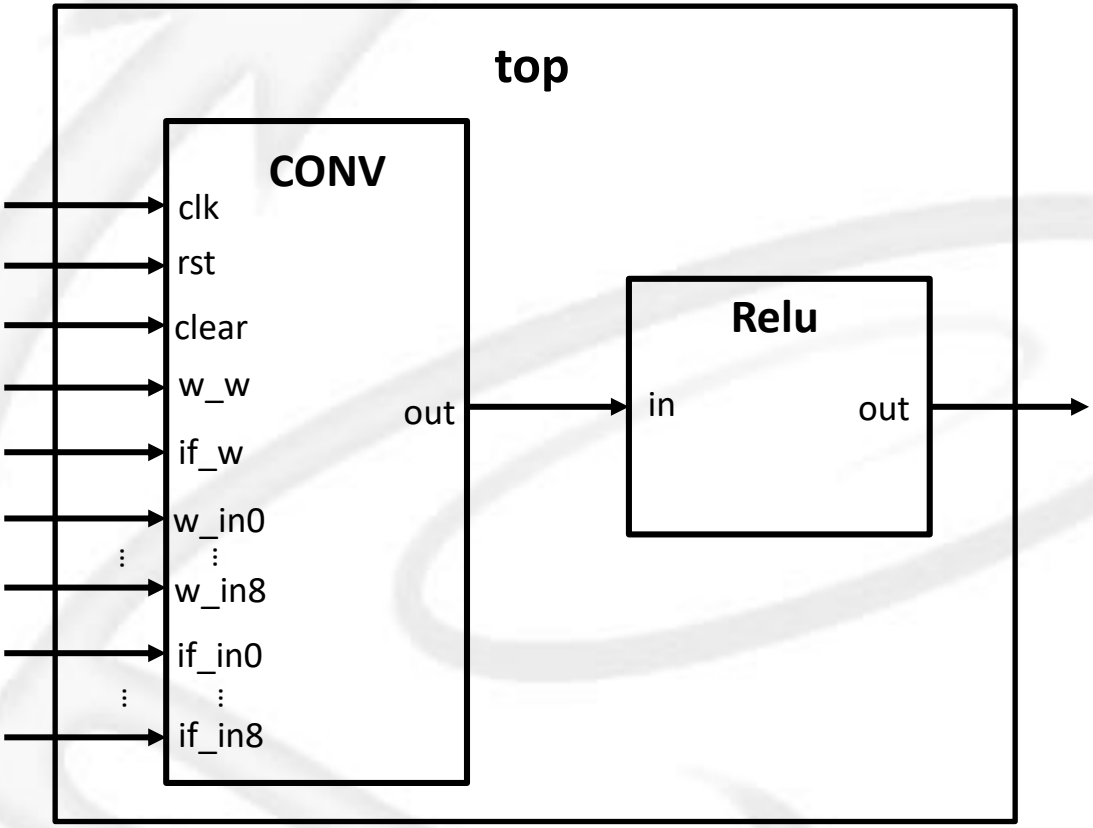
Lab B : Convolution and activation function

- Output data should be update at positive edge clock.



Lab B : Convolution and activation function

- Combine CONV and Relu module.



Outline

- Synthesis steps
- Synthesis tips
- Lab A : Register File
- Lab B : Convolution and activation function
- Homework

Homework

- Due day : **2021-03-31, Wed, 15:00**
- ProbA
  - ➔ Design a 64 X 32 register file based on LabA's structure.
- ProbB
  - ➔ Revise a sequential circuit "simple vending machine"
- ProbC
  - ➔ Design an Convolution and Parametric Relu file based on LabB's structure.
  - ➔ You can use behavior modeling in this problem.

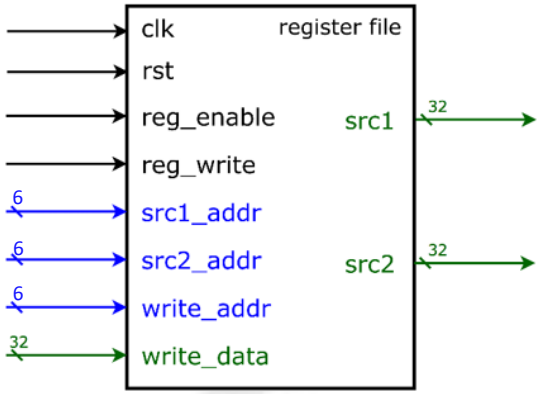
Attention :

1. Make sure all your verilog code can be compiled in SOC lab.
2. Behavioral descriptions are acceptable.

Prob A : 64 X 32 register file

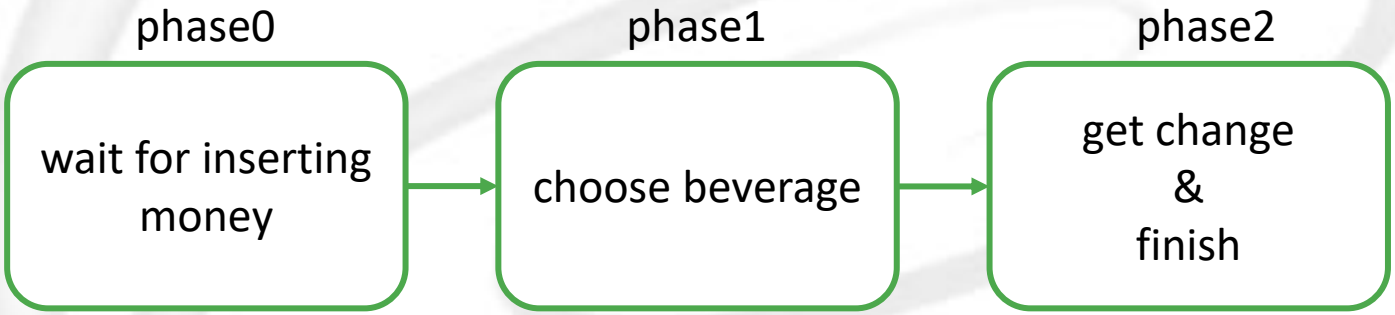
Port list

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	6	source1 address
src2_addr	input	6	source2 address
write_addr	input	6	write address
write_data	input	32	write data
src1	output	32	read data source1
src2	output	32	read data source2



Prob B : a simple vending machine

- Revise a sequential circuit “simple vending machine”
- Assume the amount of inserted money is always no less than the price of beverage which you want to buy
- Steps:

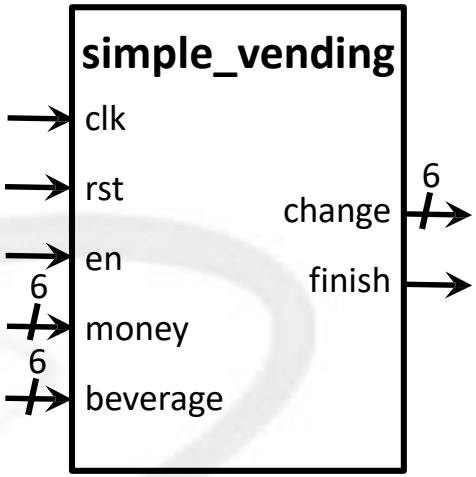


Prob B : a simple vending machine

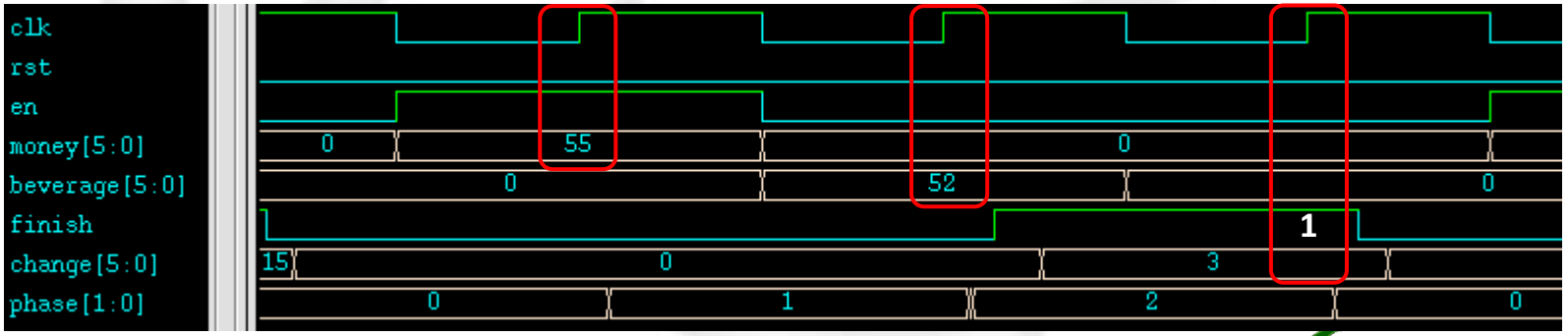
Port list

simple_vending

Signal	Type	Bits	Description
clk	input	1	clock
rst	input	1	reset
en	input	1	machine enable (When inserting money)
money	input	6	inserted money (assume money > beverage)
beverage	input	6	the price of beverage
change	output	6	the rest of money
finish	output	1	When the machine gives change, finish pulls to 1. Otherwise, finish is 0.



Waveform (post-simulation)

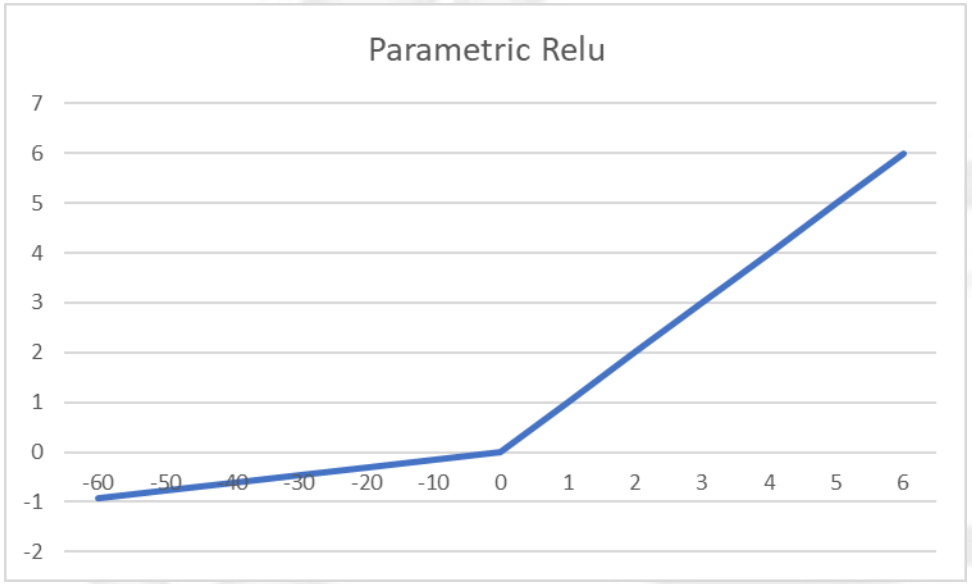


Prob B reference code with some bugs

```
module mini_vending(clk,rst,en,money,beverage,change,finish);
  // ------- I/O port --------- //
  input              clk,rst,en;
  input       [5:0] money,beverage;
  output reg  [5:0] change;
  output reg        finish;
  // -------------------------- //
  reg [1:0] phase;
  reg [5:0] money_temp;
 // -------------------------- //

  // phase0 : insert money
   always@(posedge rst or posedge clk)begin
       if(rst) begin
          phase <= 2'd0;
          money_temp <= 6'd0;
       end
       else if(phase == 2'd0 && en == 1'b1) begin
         phase <= 2'd1;
         money_temp <= money;
       end
   end

  // phase1 : choose beverage
  always@(posedge rst or posedge clk)begin
       if(rst) begin
          phase <= 2'd0;
          money_temp <= 6'd0;
       end
       else if(phase == 2'd1) begin
          phase <= 2'd2;
          money_temp <= money_temp - beverage;
       end
  end

  // phase2 : give change
  always@(posedge rst or posedge clk)begin
       if(rst) begin
          phase <= 2'd0;
          money_temp <= 6'd0;
       end
       else if(phase == 2'd2) begin
          phase <= 2'd0;
          money_temp <= 6'd0;
       end
  end

  // change and finish
  always@(*)begin
     if(phase == 2'd2) begin
        change = money_temp;
        finish = 1'b1;
     end
     else begin
        finish = 1'b0;
     end
  end

endmodule
```

Prob C : Convolution and Parametric Relu

- **Use Parametric Relu as activate function**
  (do not use multiplier)



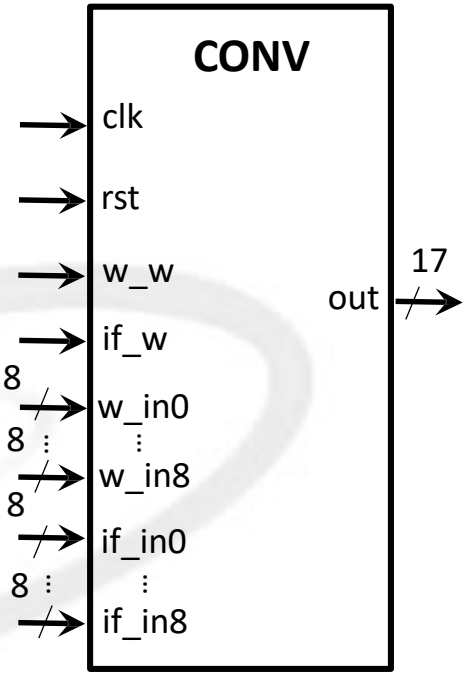
$$f(x) = \begin{cases} 0.015625x, if\ x \leq 0 \\ x, if\ x > 0 \end{cases}$$

Prob C : Convolution and Parametric Relu

- Port list

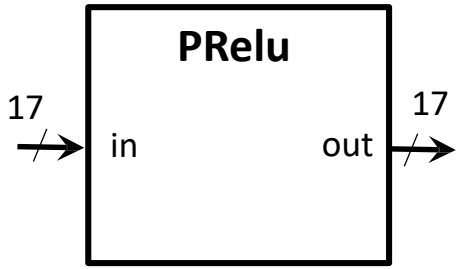
CONV

Signal	Type	Bits	Description
clk	input	1	Clock pin.
rst	input	1	Reset pin.
w_w	input	1	Write weight enable. When w_w is high, then w_in0~8 is available.
if_w	input	1	Write input feature map enable. When if_w is high, then if_in0~8 is available.
w_in0~8	input	8 each	Input weight data.
if_in0~8	input	8 each	Input input feature map data.
out	output	17	Output data.



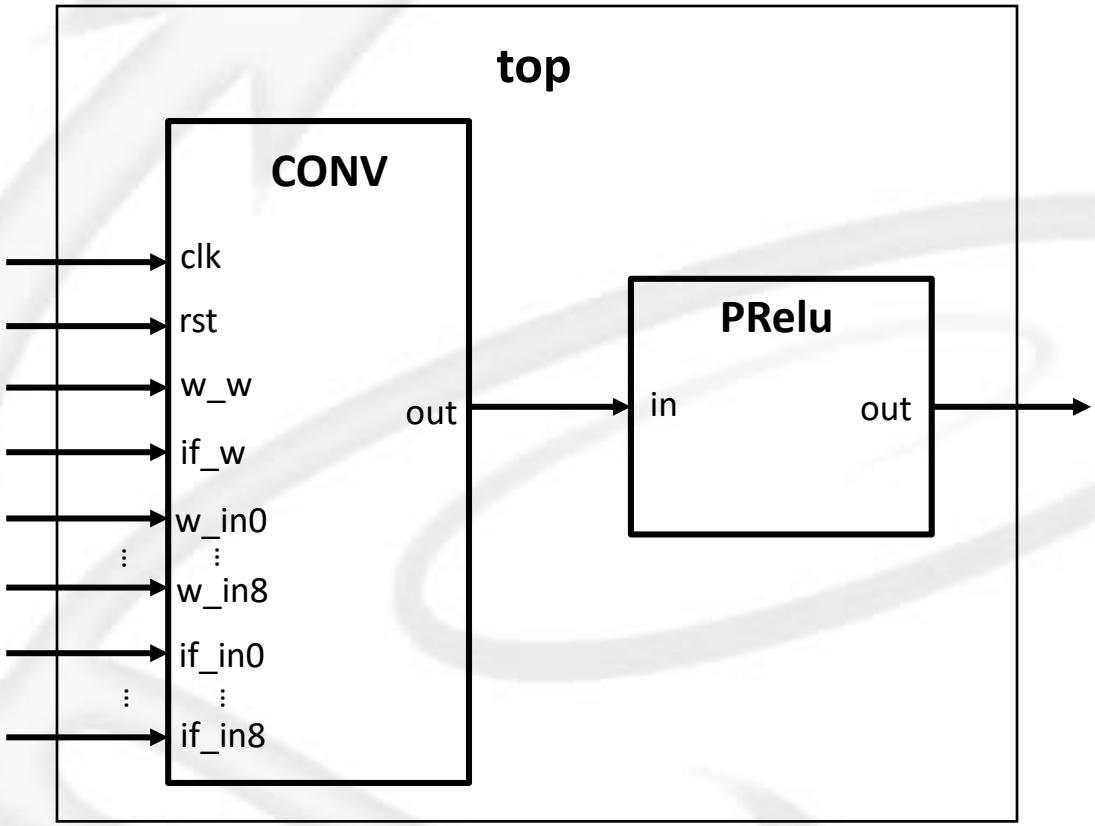
Parametric Relu

Signal	Type	Bits	Description
in	input	17	Input data.
out	output	17	Output data.



Prob C : Convolution and Parametric Relu

- Combine CONV and PRelu module.



Simulation Commands

- Please use the command below to verify your design

Prob	Syn	command
A	pre	ncverilog regfile_tb.v +access+r +define+FSDB
B	pre	ncverilog mini_vending_tb.v +access+r +define+FSDB
	post	ncverilog mini_vending_tb.v +access+r +define+FSDB+syn
C	pre	ncverilog top_tb.v +access+r +define+tb1+FSDB ncverilog top_tb.v +access+r +define+tb2+FSDB
	post	ncverilog top_tb.v +access+r +define+tb1+FSDB+syn ncverilog top_tb.v +access+r +define+tb2+FSDB+syn

Remind

1. DO NOT paste code in report file!

3. ONLY.*tar* formats are accepted!

※ Violating these rules will lead to credit *deduction*

Thank you
For your attention!!