+5.5

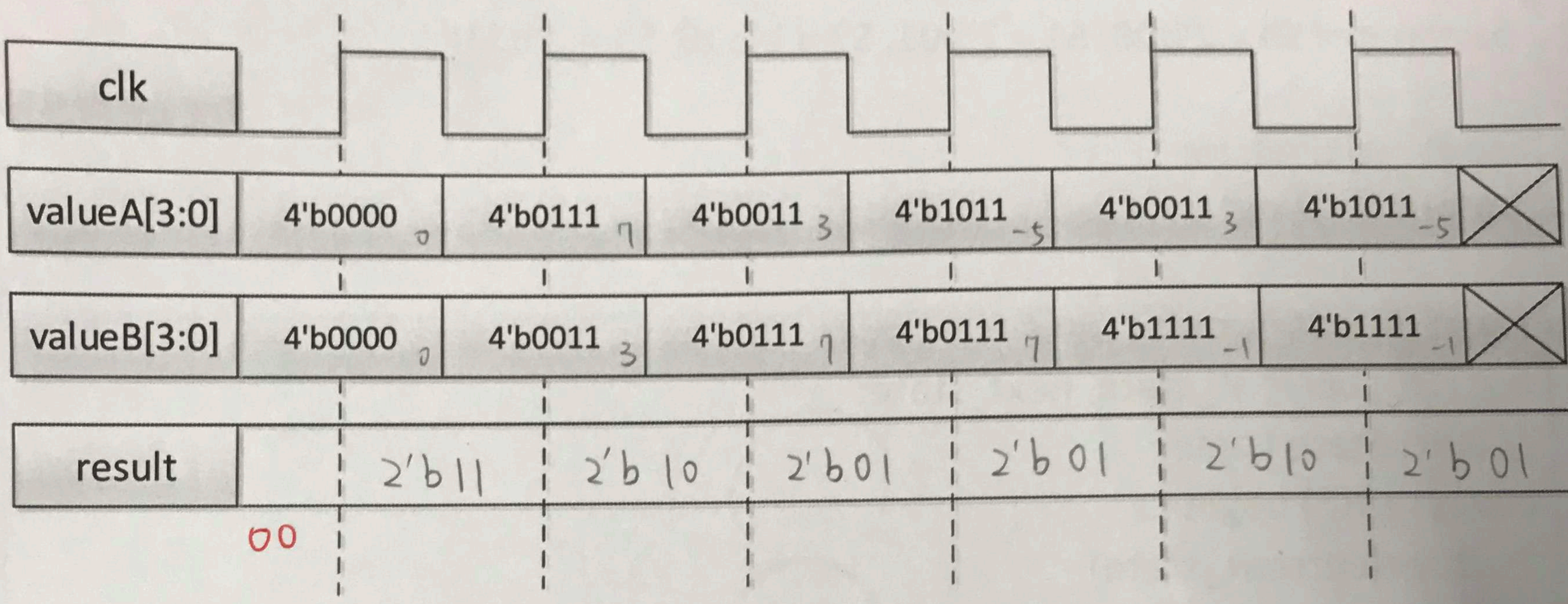
1. Code Debugging and Simulation (10pts)

A. (6pts) Identify syntax and semantic errors. Correct them and put annotations. There are 12 errors and 0.5pts for each.

```
module 2value_comparator (result, clk, rst, valueA[3:0], valueB[3:0]) ;
input clk;
input rst; /* Active High Asynchronous Reset */

signed input [3:0] valueA;
signed input [3:0] valueB;

output [1:0] result;

reg BigA, BigB;


assign BigA = (valueA >= valueB)? 1'b1: 1'b0;
assign BigB = (valueA =< valueB)? 1'b1: 1'b0;

always (posedge clk or rst);
begin
if(rst)
    result <= 2'b0;
else
    result <= {BigA, BigB};
end
endmodules
```

+3.5

B. (4pts) Please draw the waveform for the circuit in part A. Suppose that the signal **result** is reset to zero in the beginning. Use "xx" to indicate values that cannot be determined given the provided information.



2. Finite State Machine and Simulation (10pts)

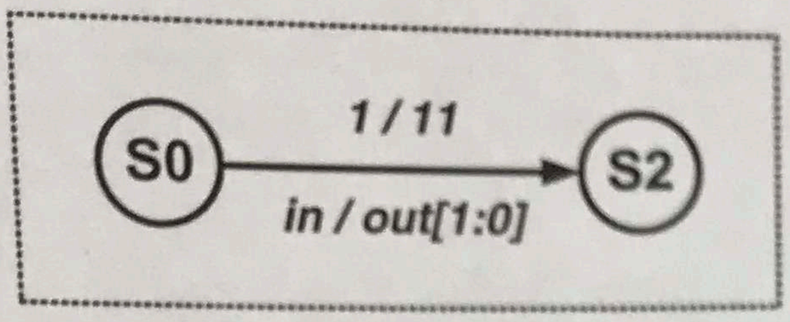
Given a Finite-State-Machine (FSM) as below.

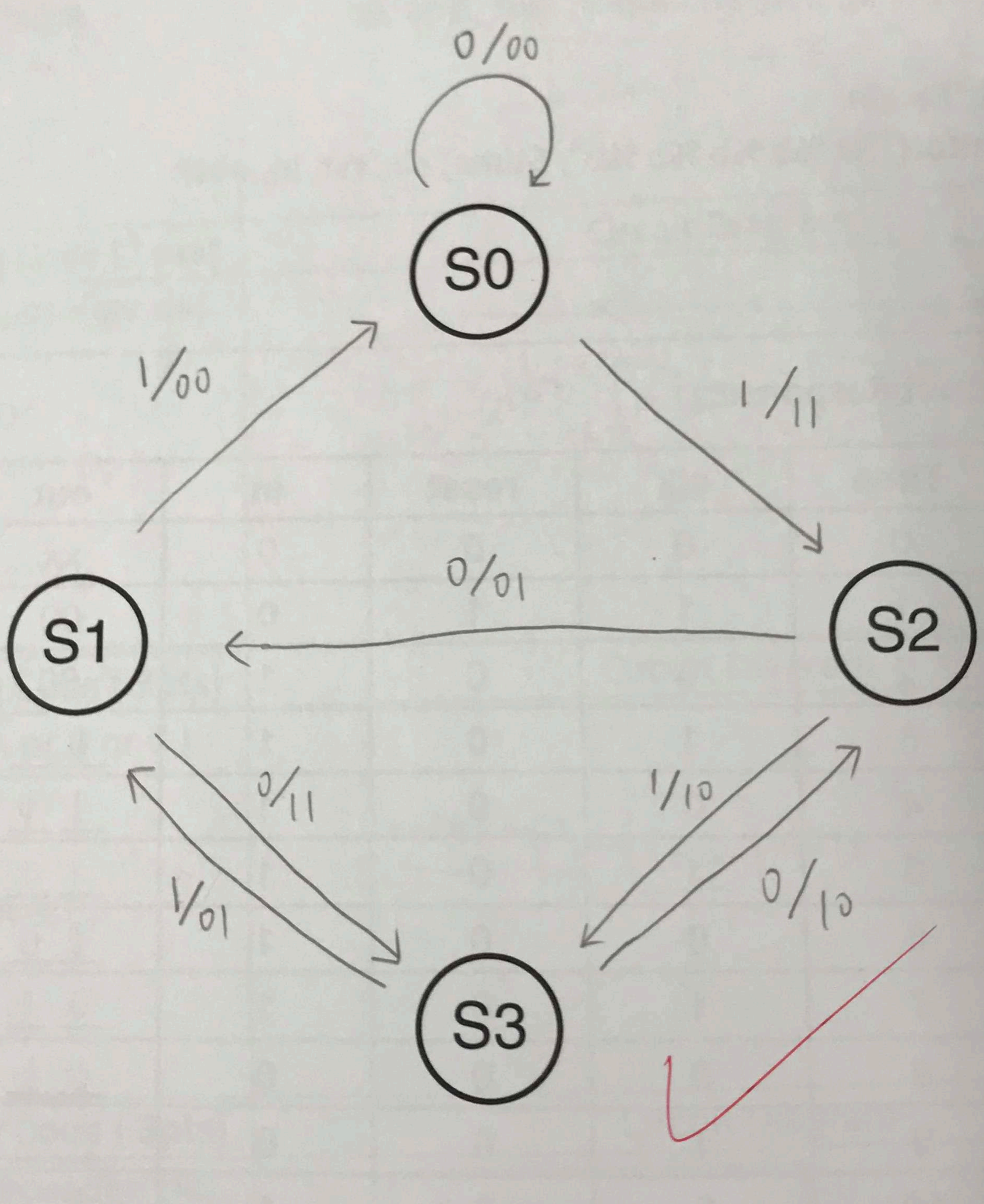
```
module FSM (clk, rst, in, out_r);
parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;

input    clk, rst, in;
output [1:0] out_r;

reg [1:0] out_r, out;
reg [1:0] current_state, next_state;
// Next State Logic
always @(*) begin
      case(current_state)
      S0: next_state = (in == 1'b0)? S0 : S2;
      S1: next_state = (in == 1'b1)? S0 : S3;
      S2: next_state = (in == 1'b0)? S1 : S3;
      S3: next_state = (in == 1'b1)? S1 : S2;
      default: next_state = 2'b01;
      endcase
end
// Current State Memory & Output Register
always@(posedge clk or posedge rst)
begin
     if(rst) begin
          current_state <= 0;
          out_r <= 0;
     end
     else begin
          current_state <= next_state;
          out_r <= out;
     end
end
// Output Logic
always@(*) begin
     out[1] = (~in) ~^ current_state[0];
     out[0] = in ^ current_state[0] ^ current_state[1];
end
endmodule
```

		$\bar{in}=0$		$\bar{in}=1$	
		out[1]	out[0]	out[1]	out[0]
S0	00	0	0	1	1
S1	01	1	1	0	0
S2	10	0	1	1	0
S3	11	1	0	0	1

(a) (5pts) Please draw a state transition graph below for this FSM.

Example



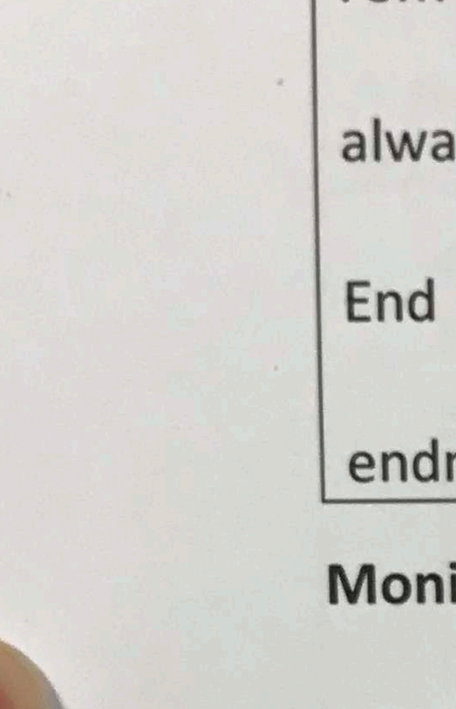


(b) (5pts) The FSM is included as a design under test (DUT) in the testbench. After simulation, the command window shows the outputs. Please finish the outputs based on operation of the FSM.

```
module testbench;
reg    clk, rst;
reg    in;
wire [1:0] out;

FSM DUT(.clk(clk), .rst(rst), .in(in), .out_r(out));

always @(*)begin
      $monitor("%t %b %b %b %b", $time, clk, rst, in, out);
End

endmodule
```

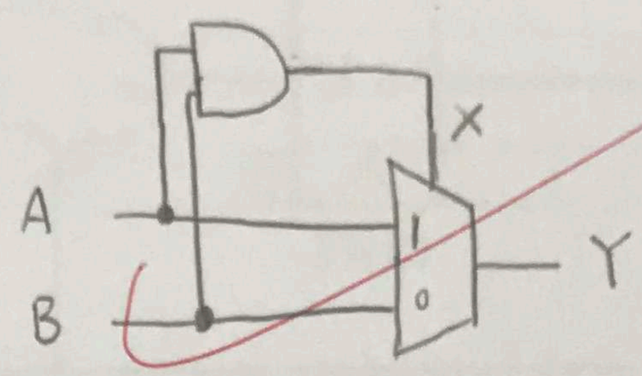
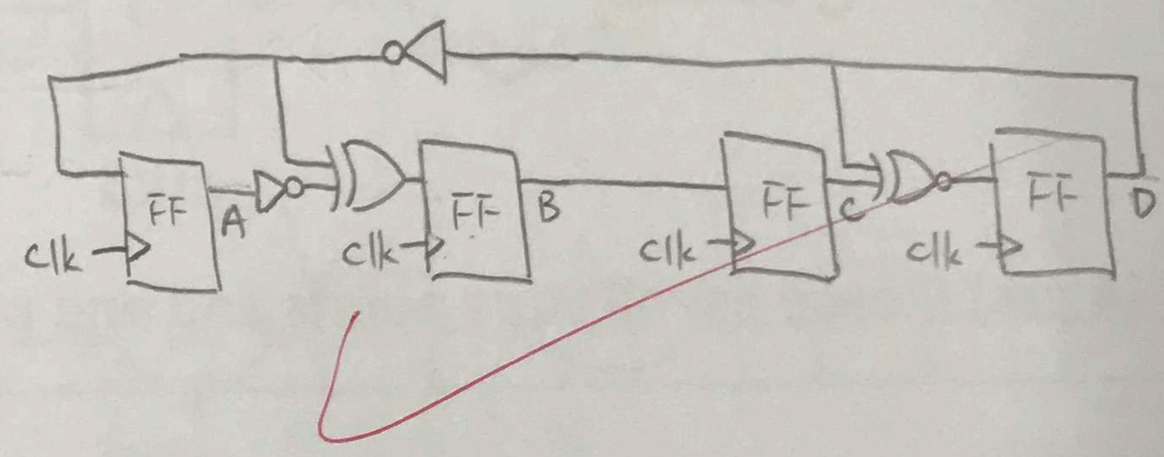
Monitor Output Response:

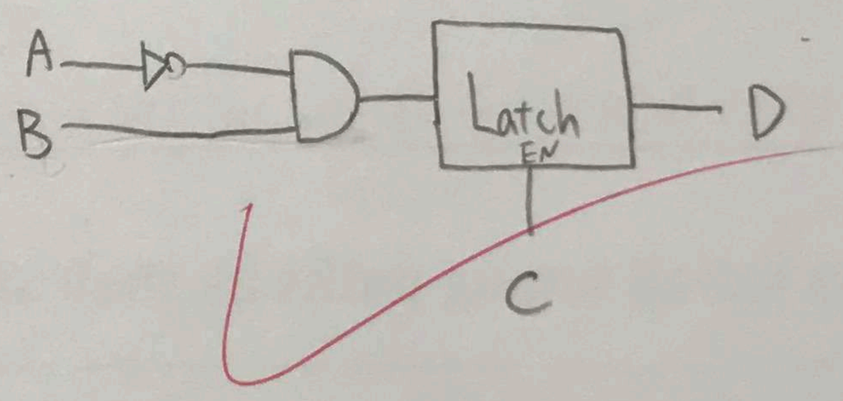
Time	clk	reset	in	out
0	0	0	0	xx
1	1	1	0	00
2	0	0	1	00
3	1	0	1	11
4	0	0	1	11
5	1	0	1	10
6	0	0	1	10
7	1	0	1	01
8	0	0	0	01
9	1	0	0	11
10	0	0	1	11
11	1	0	1	01
12	0	0	0	01
13	1	0	0	11
14	0	0	0	11
15	1	0	0	10
16	0	0	0	10



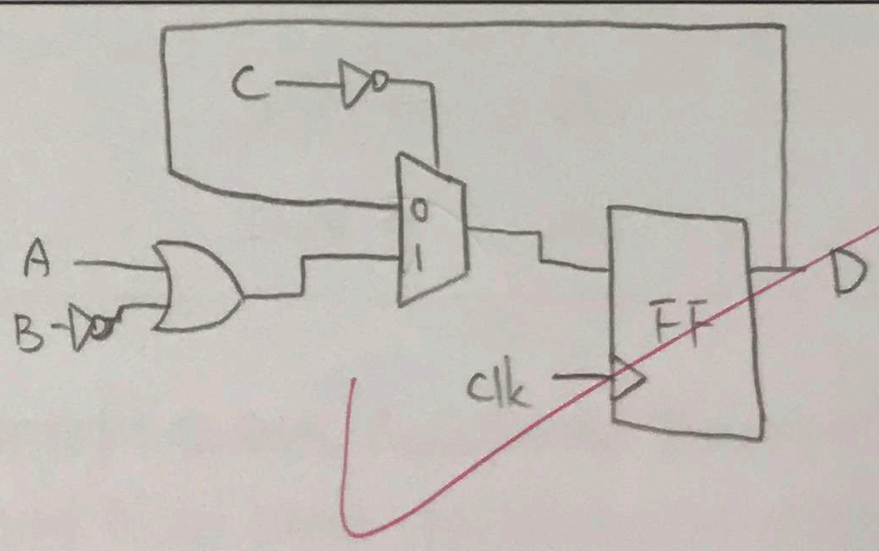
3. Logic Synthesis + Blocking & Non-Blocking (10 pts)

Please draw the corresponding circuits (in the right column) according to the Verilog codes (in the left column). You can use AND, OR, NAND, NOR, XOR, XNOR, NOT, MUX, D Flip-Flop, Latch in the circuit diagram.

(a) Verilog Code **(2 pts)**	Circuit Diagram
always @(*) begin X = A&B; Y = (X)? A: B; end	
(b) Verilog Code **(2 pts)**	Circuit Diagram
always @(posedge clk) begin A <= ~D; B <= ~A ^ ~D; C <= B; D <= C ~^ D; end	
(c) Verilog Code **(3pts)**	Circuit Diagram

always@(A or B or C)
begin
if (C)
D = ~A & B;
end
	

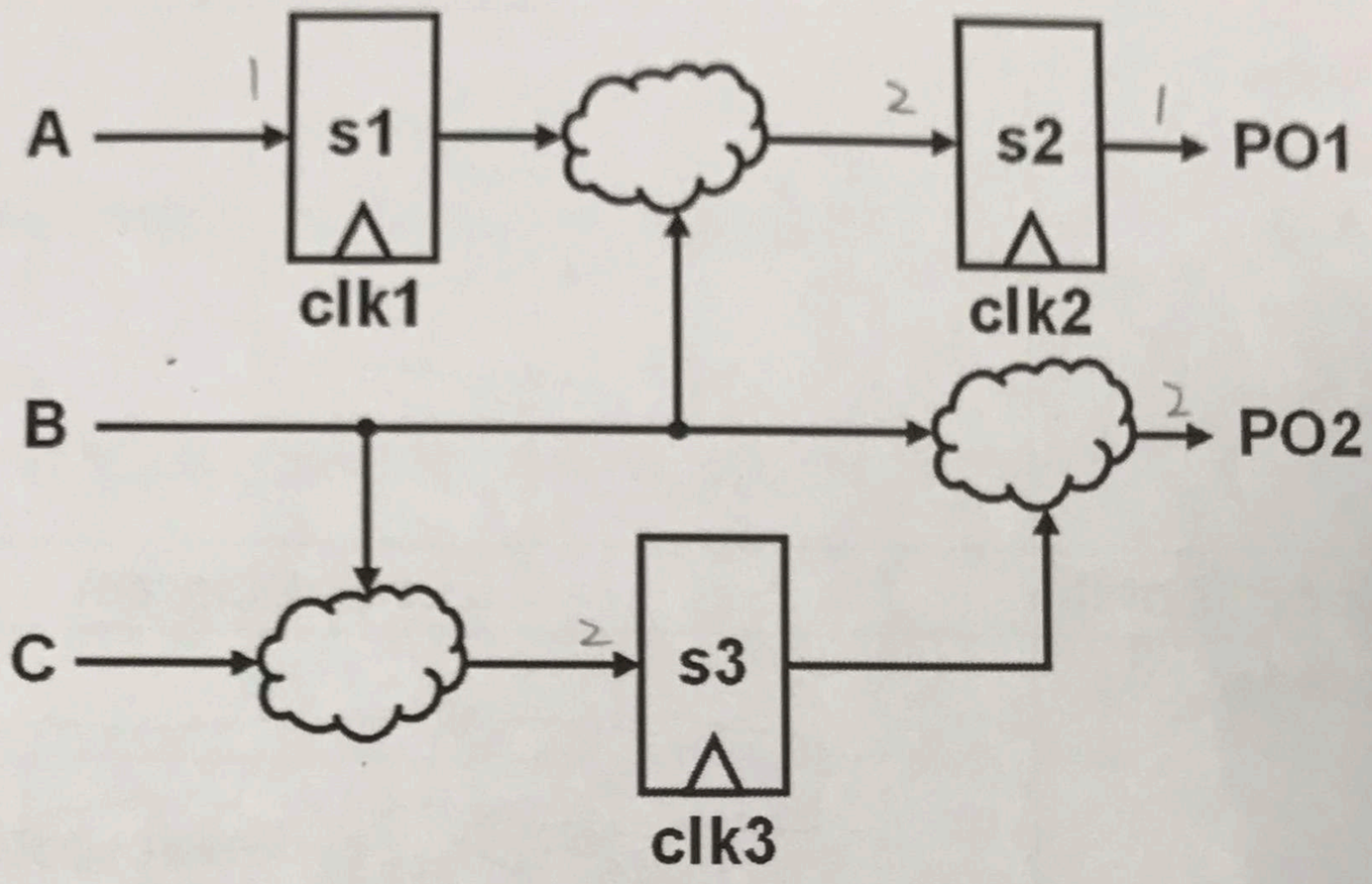
(d) Verilog Code **(3pts)**	Circuit Diagram

always@(posedge clk)
begin
if (!C)
D <= A | ~B;
end
	

4. Timing Analysis (25 pts)

A. (7 pts) <Timing Path>

The following figure shows a circuit with three primary inputs (A, B, C) and two primary outputs (PO1, PO2). The three sequential elements have individual data inputs (s1, s2, s3) and clocks (clk1, clk2, clk3). The cloud-shape modules are combinational elements.



(a) (4 pts) Please list all start points and end points.

start

A. B. C

clk1 . clk2 . clk3

共 6 個

end

S1. S2. S3

PO1 . PO2

共 5 個

+4

(b) (3 pts) Please list all timing paths by their start point and end point.

(Ex: clk1 to s2 is a timing path)

A → S1

B → PO2

B → S2

B → S3

C → S3

clk1 → S2

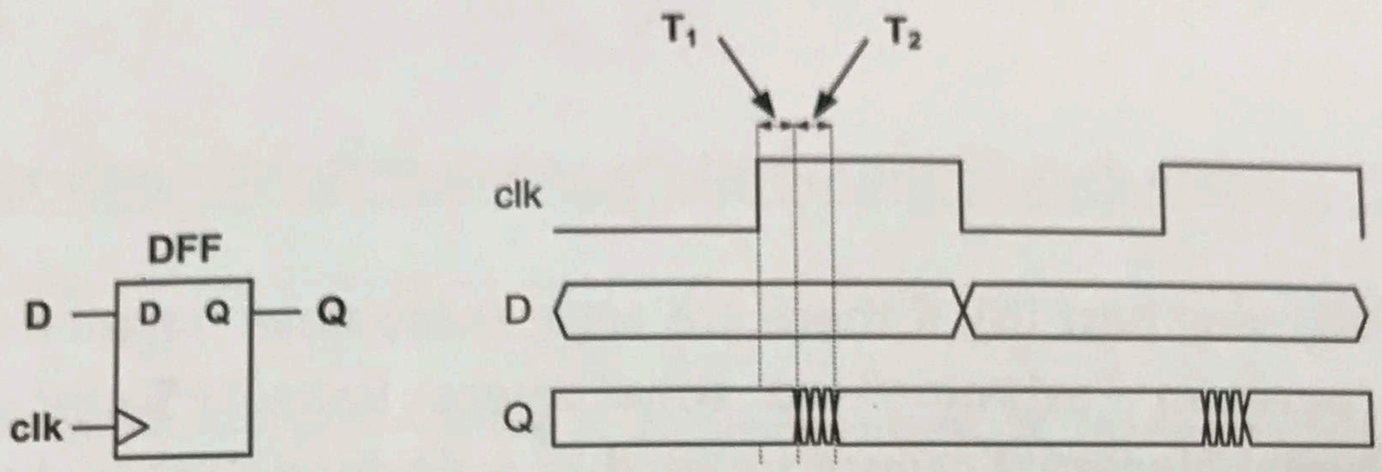
clk2 → PO1

clk3 → PO2

+3

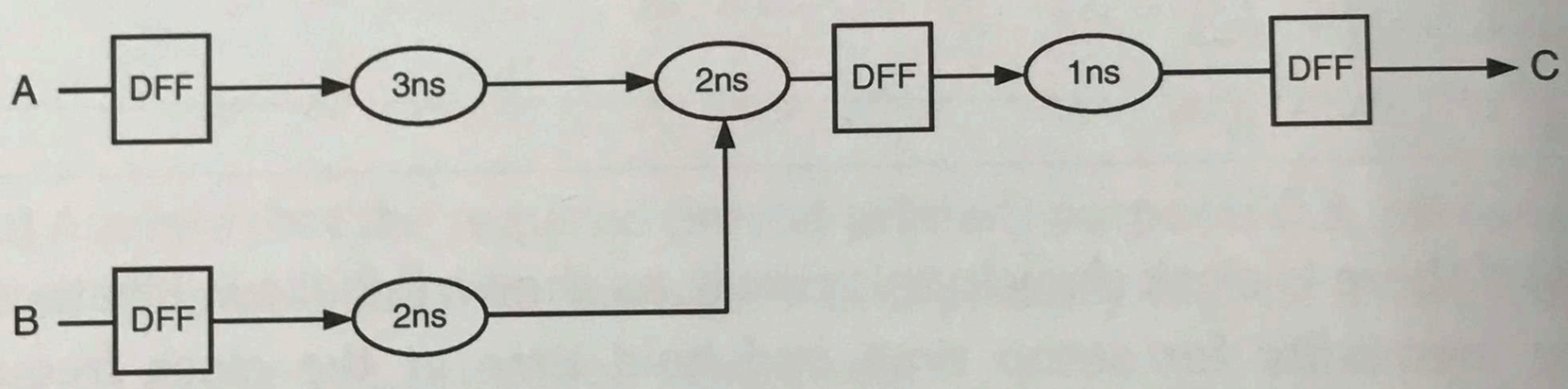
B. (12 pts) <Hold Time & Setup Time>

Suppose that the timing characteristics of the flip-flops in the circuit are the same. Their timing diagrams and parameters can be described as follows:



T_1=0.4ns, T_2= 0.1ns

The circuit below operates at the clock frequency of **250MHz.** Suppose that the rise, fall delays for each combinational element are the same.



250MHz = 4ns

$T_{cq} = T_1 + T_2 = 0.5\,ns$

$T_{cq,cd} = T_1 = 0.4\,ns$

(a) (3 pts) Write the timing inequality for setup time and hold time.

setup time

$T_{cq} + T_{logic} + T_{su} < T_{clk}$

$\Rightarrow T_{su} < T_{clk} - T_{cq} - T_{logic}$

$T_{su} < 4\,ns - 0.5\,ns - (3ns + 2ns)$ ← longest path

$\Rightarrow T_{su} < -1.5\,ns$

hold time

$T_{cq,cd} + T_{logic,cd} > T_h$

$0.4\,ns + 1\,ns > T_h$ ← short path

$T_h < 1.4\,ns$

(b) (2 pts) If T_{setup} **(Setup Time) = 0.4ns,** T_{hold} **(Hold Time) = 1.5ns**

Are there setup time and hold time violations in this circuit? Use the timing inequalities in (a) for setup/hold time at the clock frequency to check them.

+2

$T_{su} < -1.5\ ns$, $0.4\ ns > -1.5ns \Rightarrow$ setup time violation

$T_h < 1.4\ ns$, $1.5\ ns > 1.4\ ns \Rightarrow$ hold time violation

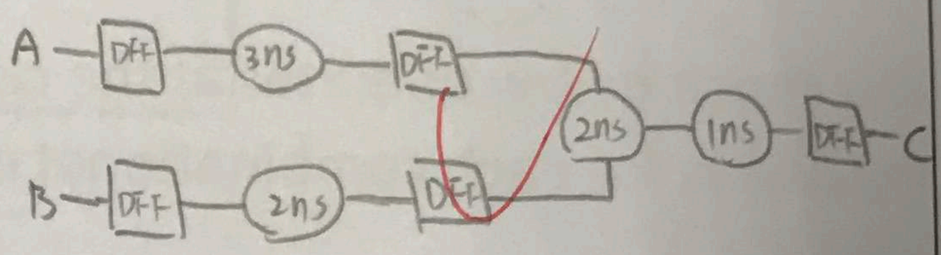
(c) (3 pts) Following part (b), if there are setup/hold time violations in this circuit, how to perform "retiming" to solve these issues? Suppose that all of combinational elements cannot be further separated. Please draw your circuit and write the timing inequalities of the modified circuits after retiming.

+3

retiming：調整 DFF 的位置

for setup：使 longest path (3+2ns) 變短

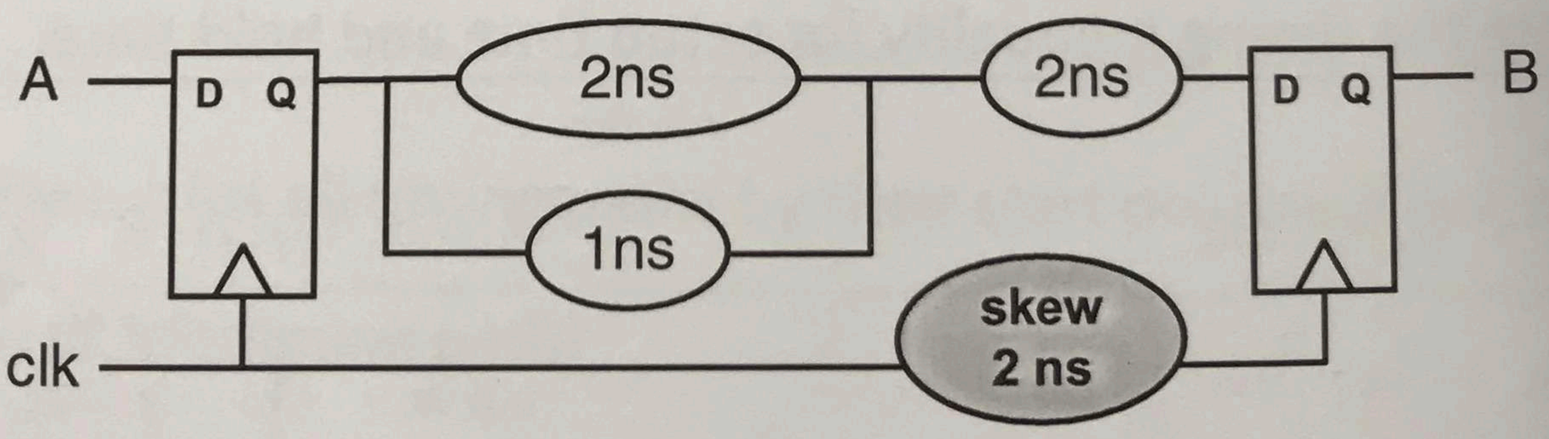
for hold：使 shortest path (1ns) 變長)



$\Rightarrow T_{su} < 4 - 0.5 - 3 = 0.5 \Rightarrow T_{su} < 0.5\ ns$, $\therefore T_{su} = 0.4$ ok

$\Rightarrow T_h < 0.4\ ns + 2\ ns = 2.5 \Rightarrow T_h < 2.5\ ns$, $\therefore T_h = 1.5$ ok

(d) (4 pts) If there is clock skew in this circuit, as shown in bellow. Please write the timing inequality for setup time and hold time at the clock frequency of **200MHz without** any timing violation.



-4

$200\ MHz = 5ns$

setup time

$T_{cq} + T_{logic} + T_{su} \leq T_{clk} + T_{skew}$

$\Rightarrow 0.5 + (2+2) + T_{su} < 5 + 2$

$T_{su} < 2.5ns$

hold time

$T_{cq,cd} + T_{logic,cd} > T_h + T_{skew}$

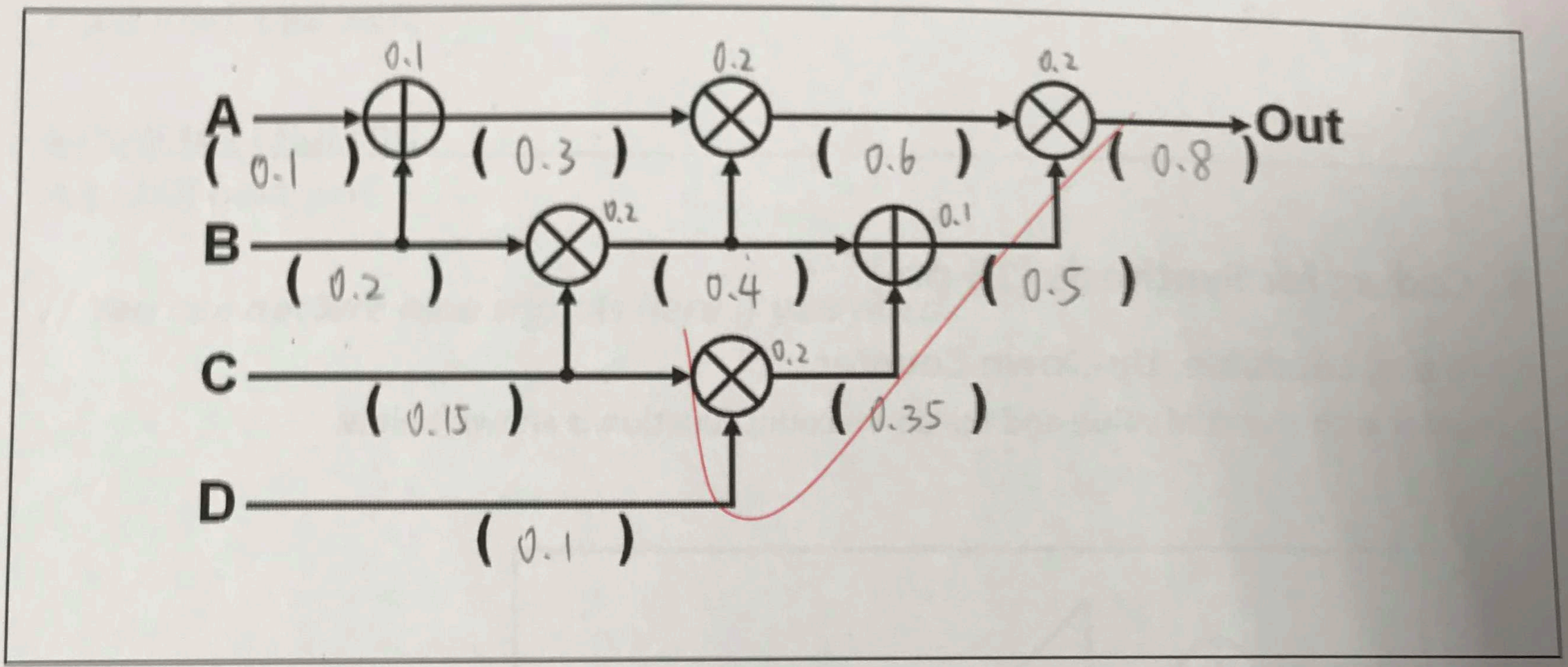
$0.4 + (1+2) > T_h + 2$

$T_h < 1.4\ ns$

C. **(6 pts) <Required Time & Slack>**

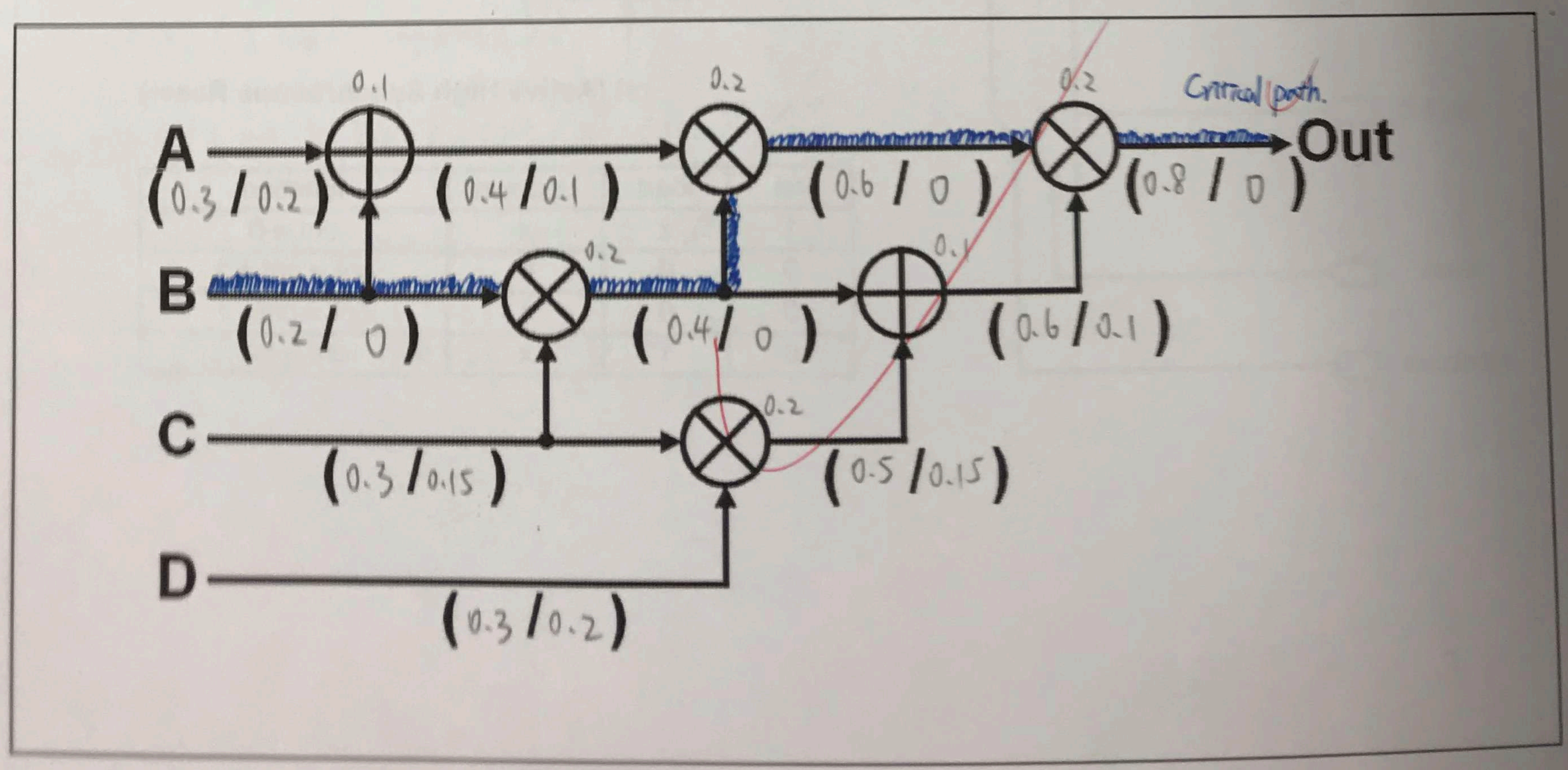
The following figure shows a combinational circuit with primary inputs A, B, C, and D. Arrival time of these primary inputs are 0.1, 0.2, 0.15, and 0.1 respectively. A multiplier has a delay of 0.2 and a adder has a delay of 0.1. Assume delay values of wires are zero.

(a) (2 pts) Write down the arrival time of the output of each edge.



+2

(b) (2 pts) Assume that the required time at primary output is 0.8, please compute required time and slack for each gate and primary inputs. Write down your answer on each edge. (Required time/slack)



+2

(c) (2 pts) Please indicate the critical path and show the critical path by required time values.

critical path : slack 最小的 path

+2

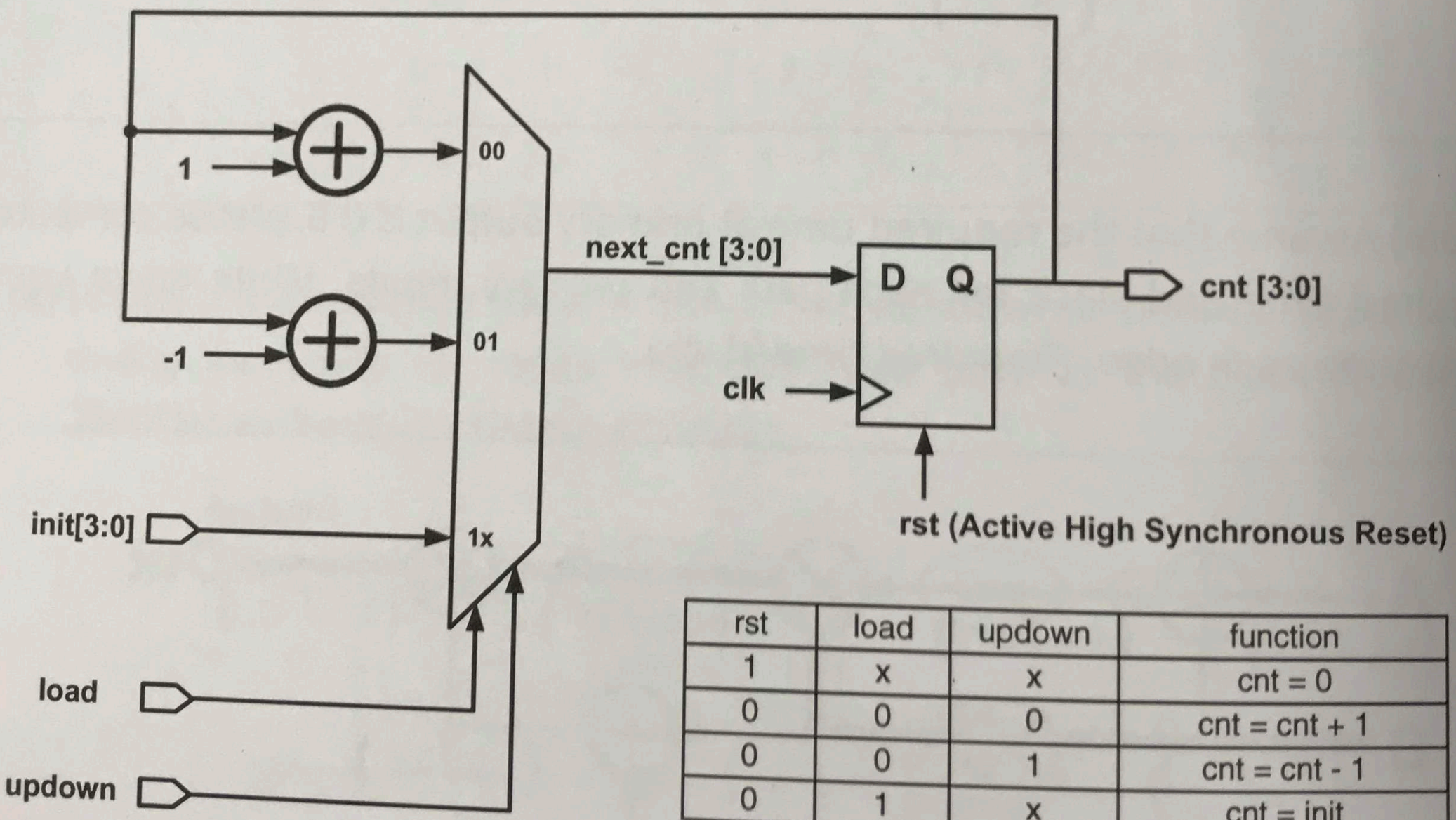
B (0.2 /0) ⊗→ (0.4 /0) ⊗→ (0.6 /0) ⊗→ Out (0.8 /0)

藍色線

5. Coding for Synthesis (15 pts)

A. (8 pts) Loadable, Up-Down Counter

A counter with an initial value and up/down count function is shown below.



rst	load	updown	function
1	x	x	cnt = 0
0	0	0	cnt = cnt + 1
0	0	1	cnt = cnt - 1
0	1	x	cnt = init

Please complete the Verilog code of this counter:

```
module counter (clk, rst, init, load, updown, cnt);

                    // I/O & Reg/Wire Declaration

input clk, rst;
input [3:0] init;
input load, updown;

output reg [3:0] cnt;
reg [3:0] next_cnt;

// You can declare new signals here if you need




                    // Combinational Logic for the Counter (4 pts)

always@(*) begin
    case ({load, updown})
        2'b00 : next_cnt = cnt + 1;
        2'b01 : next_cnt = cnt - 1;
        default : next_cnt = init;
    end case
end
```

4

// Sequential Logic for the Counter (4 pts)

always@(posedge clk) begin
if (rst)
cnt <= 0 ;
else
cnt <= next_cnt ;

end
endmodule

+4

B. (7 pts) Gray Counter

Create a three-bit gray counter with positive edge reset. The preset is initialized to a value of 000. Recall that a Gray Counter changes only one-bit at a time. For example, a 2-bit Gray counter has a count sequence 00, 10, 11, 01 corresponding to a decimal count values of 0, 1, 2, 3 respectively. In the Gray count sequence, only one bit changes between adjacent count values, and the left-most bit is changed as long as it does not result in a code word that has been visited earlier.

000
100
110
010
011
111
101
001

clk: 1-bit clock input, all actions on positive edge
rst: 1-bit reset, causes reset on positive edge (asynchronous)
gray_out: 3-bit result

```
module gray_counter (clk, rst, gray_out);

input clk, rst;
output [2:0] gray_out;

reg [2:0] gray_out;
reg [2:0] gray_tmp;

reg [2:0] count;
```

```
always @ (posedge clk or posedge rst)
begin
    if (rst)
        count <= 0 ;
    else
        count <= count + 1 ;
end

always @ (posedge clk or posedge rst)
begin
    if (rst)
        gray_out <= 0 ;
    else
        gray_out <= gray_tmp ;
end

always @ (*)
begin
    case (count)
        3'd0 : gray_tmp = 3'b100 ;
        3'd1 : gray_tmp = 3'b110 ;
        3'd2 : gray_tmp = 3'b010 ;
        3'd3 : gray_tmp = 3'b011 ;
        3'd4 : gray_tmp = 3'b111 ;
        3'd5 : gray_tmp = 3'b101 ;
        3'd6 : gray_tmp = 3'b001 ;
        3'd7 : gray_tmp = 3'b000 ;
        default : gray_tmp = 3'b000 ;
    end case
end

endmodule
```

6. Synthesis Issues (28 pts)

+6

A. (6 pts) < Important files related to Design Compiler >
Please explain the meaning of the following terminologies and where to use them:

(a) (2 pts) DesignWare library (e.g: dw_fundation.sldb)
(b) (2 pts) Standard Delay File (e.g: CHIP_syn.sdf)
(c) (2 pts) .synopsys_dc.setup

(a) 合成用時那個 gate 的 library，也就是合成時那些 gate 的來源和資訊
(b) 合成後用來記錄時間資訊的檔案，用來模擬用
(c) 初始 design compiler 的設定檔，包含 library 的路徑等資訊

B. (4 pts) < Synopsys Design Constraints File(SDC) >
Please explain the **meaning** of the following command and why we use them in Design Compiler:

-4

(a) (2 pts) *set_dont_touch_network [get_clocks_clk]*
set_ideal_network [get_ports clk]
(b) (2 pts) *set_clock_uncertainty 0.1 [get_clocks clk]*

(a) 在合成時不要去優化 clk，把他當成 ideal network，因為合成只先考慮合成 cell，那些 network 等 APR 再去處理
(b) clk tree 有些地方會比較長，因此並不一定會同時抵達 register，(skew) 因此加上 uncertainty 確保在 worst case 也能達到正的 slack

C. (2 pts) < Post-sim >
Explain what does Standard Delay Format (SDF) file do in gate-level simulation?

-2

SDF 檔紀錄了合成後每個 gate 的時間資訊 ex delay.
如果在 simulation 時沒有 SDF 檔，那會有 time violation，
有了 delay 的資訊，才能在 gate-level 做 simulation

D. (10 pts) < Area Report >

Below is the area report after synthesis.

```
****************************************
Report : area
Design : ALU
****************************************
...

Number of cells:                90
Number of references:           10

Combinational area:       1939.291626
Noncombinational area:    2049.062256
Net Interconnect area:      undefined

Total cell area:          3988.353760
Total area:                undefined
```

(a) (2 pts) It sometimes shows "undefined" in total area. Please explain why and describe how to fix it.

(b) Following part(a),

I. **(2 pts)** In cell-based IC design flow, we will focus on total cell area instead of total area, please explain why.

II. **(3 pts)** The total cell area will be underestimated in this situation. Please briefly explain why.

(c) (3 pts) With the same RTL code, if we reduce the clock cycle, what part in the report will increase? Please briefly explain why.

(a) 沒有 wire-load model 的資訊，因此在合成時，會不知道 Net 的 area

fix: 在 design compiler 中要 set_wire_load_model ····

(b) 指面

+10

(c) 前面

(b) i: 因為合成時我們只會做出 gate-level 的 cell，而 cell 放在哪、以及 cell 之前怎麼連線並沒有做優化，到 APR 才會做，因此只需考慮 Total cell area 就好

ii: 因為沒有 set wire load model，沒有線的資訊，自然沒有線的 delay，導致可以運算的時間更多，可以用面積小、速度慢的 cell 來做，因此 area 會較小

(c) Combinarial area 會上升，因為 cycle time 變短，導致要用面積大、速度快的 cell 來算，因此 area 會變大

+6

E. (6 pts) < External IP issue >

If there's a memory module in DUT, and we generate several files from a Memory Compiler. For example, the generated files are **rom_1024x4_t13_slow_syn.db**, **rom_1024x4_t13.v**.

(a) (2 pts) Please explain what these files are for and what will happen if we do not include these files.

.db 檔類似 library，記錄了 delay、溫度等資訊

.v 檔是合成之後的 gate-level 檔.

如果沒有 .db，那就會缺少 timing 的資訊，會有 violation

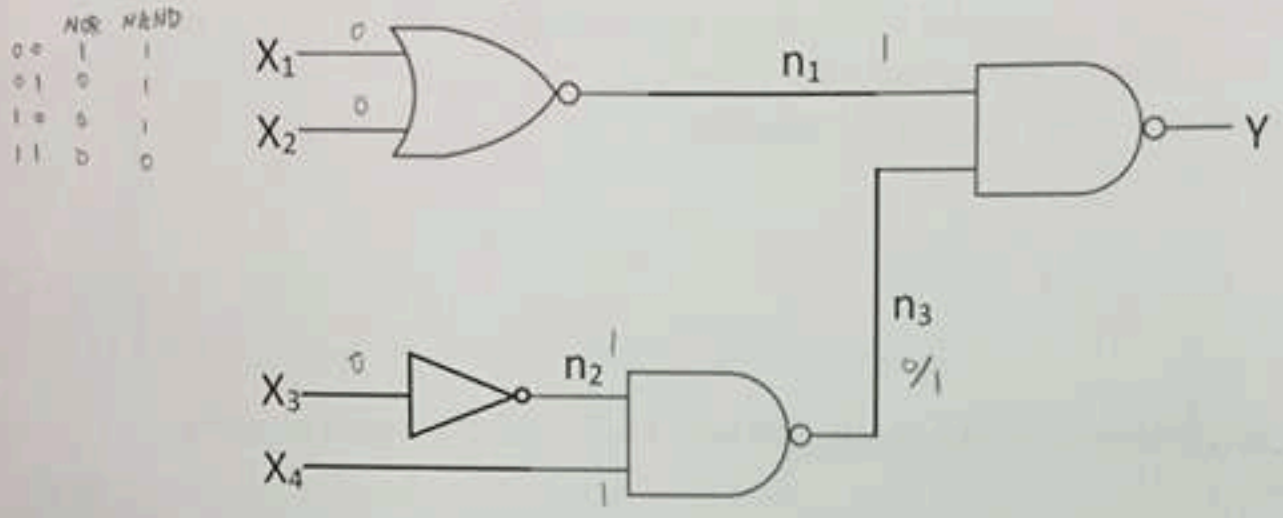
如果沒有 .v，那就沒有 gate-level 的接線及功能，就無法做 simulation

(b) (2 pts) Please modify .synopsys_dc.setup as shown below in order to make design compiler access the memory file. **(JUST NEED TO POINT OUT WHERE TO MODIFY)**

+9

7. Design for Testability (10 pts)

Given the circuit below, please answer the following questions.



(a) How many single stuck-at fault (SSF) are there in the circuit? (2pts)
(b) Generate all possible input test patterns for fault $n_1/0$ (4pts)
(c) Generate all possible input test patterns for fault $n_3/1$ (4pts)

(a) $8 \times 2 = 16$ #

(b) $n_1 = 1 \rightarrow X_1 = 0$ & $X_2 = 0$

To pass n_1 to y, $n_3 = 1 \rightarrow (n_2, X_4) = (0,0), (0,1), (1,0)$

$\Rightarrow (X_3, X_4) = (1,0), (1,1), (0,1)$

$\Rightarrow (X_1, X_2, X_3, X_4) : (0,0,1,0), (0,0,1,1), (0,0,0,1)$ (0,0,0,0)

(c) $n_3 = 0 \rightarrow n_2 = 1$ & $X_4 = 1 \Rightarrow X_3 = 0$

To pass n_3 to Y, $n_1 = 1 \Rightarrow X_1 = 0$ & $X_2 = 0$

$\Rightarrow (X_1, X_2, X_3, X_4) = (0,0,0,1)$ #

```
set search_path "Your_path/CBDK_TSMC013_Arm/CIC/SynopsysDC
$search_path "
set target_library "slow.db fast.db"
set link_library    " * $target_library dw_foundation.db"
set symbol library "tsmc13.sdb generic.sdb"
set synthetic_library "dw_foundation.sldb"
...
```

加入 rom-1024x4_t13-slow-syn.db

(c) (2 pts) Should we synthesis **rom_1024x4_t13.v** with DUT.v ? Please explain why.

不用，因 rom_1024x4_t13.v 已經為 gate-level 且有考慮 delay 的問題，可直接用，僅發合成 DUT