

11010EECS207001Logic Design Lab

LAB1 : Gate-Level Verilog

TEAM5

組長：108062213 顏浩昀

組員：106062304 黃鈺舒

Prof. Chun-Yi Lee

2021.09.30

11010EECS207001

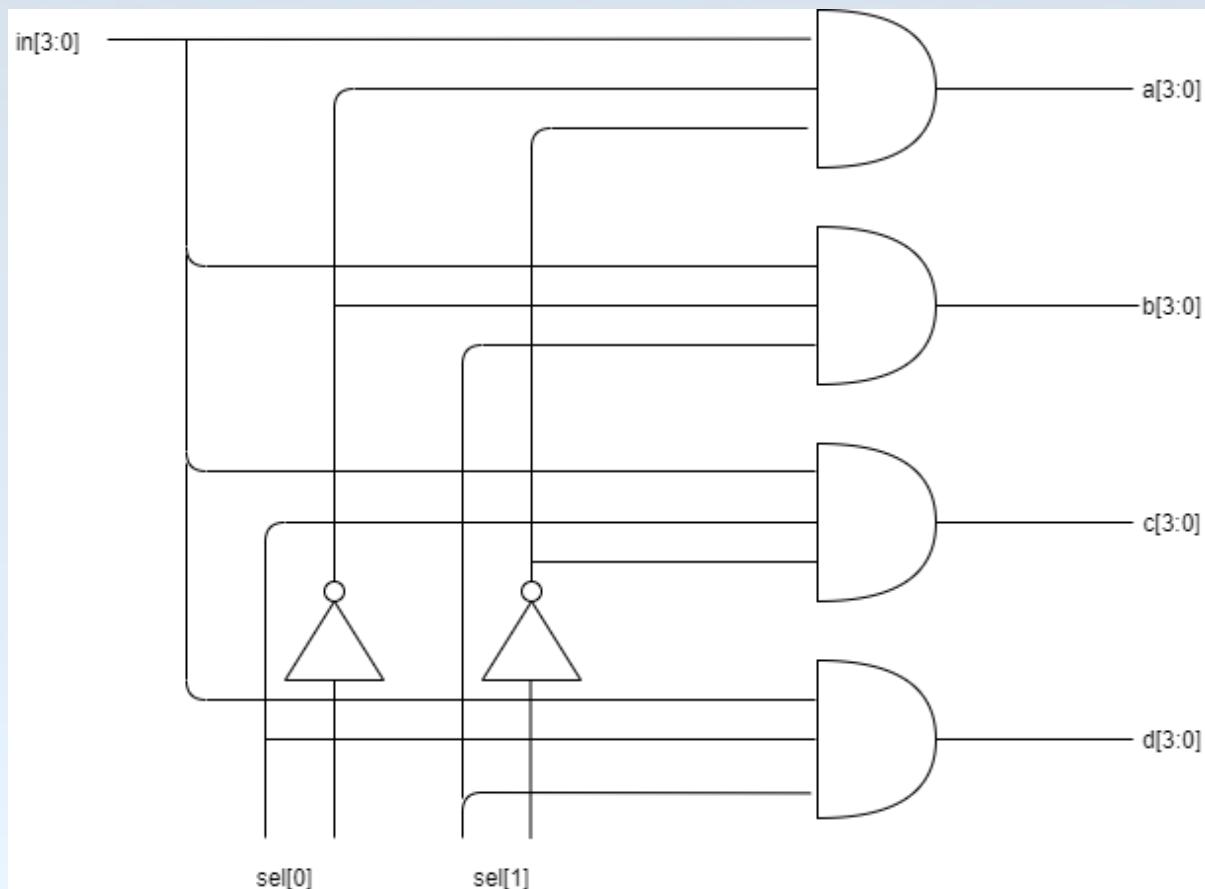
Logic Design Lab

TEAM5 LAB1

2020.09.30

4-bit 1-to-4 De-multiplexer

I. Logic Design Diagram



II. Explanation

sel 的兩個 bit 會跟輸入訊號 in 在 and gate 相遇並選出要讓資料在哪裡 output。

ex:

sel[1:0]=00->運用兩個 not gate 轉為 11 通過 and gate->輸出到 A

sel[1:0]=01->輸出到 B

sel[1:0]=10->輸出到 C

sel[1:0]=11->輸出到 D

(實作方式如下圖)

```
module Dmux_1x4_4bit(in, a, b, c, d, sel);
    input [3:0] in;
    input [1:0] sel;
    output [3:0] a, b, c, d;

    wire nsela, nselb;

    not not1(nsela, sel[0]);
    not not2(nselb, sel[1]);
    and and1[3:0](a, in, nselb, nsela);
    and and2[3:0](b, in, nselb, sel[0]);
    and and3[3:0](c, in, sel[1], nsela);
    and and4[3:0](d, in, sel[1], sel[0]);
endmodule
```

III. Testbench

這題 testbench 使用窮舉法，首先將 in 與 sel 先設為 0，透過 2 層 repeat 分別跑過所有可能性。內層的 repeat 在每一個 sel 值會跑 16 次，使 in 的值可以完整跑過。外層的 repeat 則跑 4 次，代表 sel 從 00 到 11 的過程。

```
reg CLK = 1;
reg [4-1:0] in=4'b0000;
reg [2-1:0] sel=2'b00;
wire [4-1:0] a, b, c, d;
reg pass = 1;

repeat(4) begin
    repeat(2**4) begin
        @(posedge CLK)
        simulate;
        @(negedge CLK)
        in = in + 4'b1;
    end
    #5
    sel = sel + 1'b1;
end
```

另外，為了準確且方便的對照正確性，除了人工對照波形圖的數值外，我們還製作了 task simulate，將透過 if-else 條件控制去判斷 in-out 的配對是否正確，並且用 pass 這個訊號做為辨別，如果有錯誤 pass 會從 1 變為 0。(詳細實作方式請見下圖)

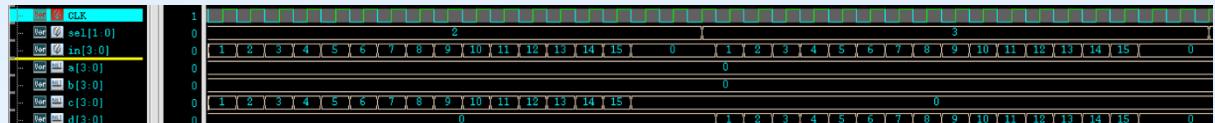
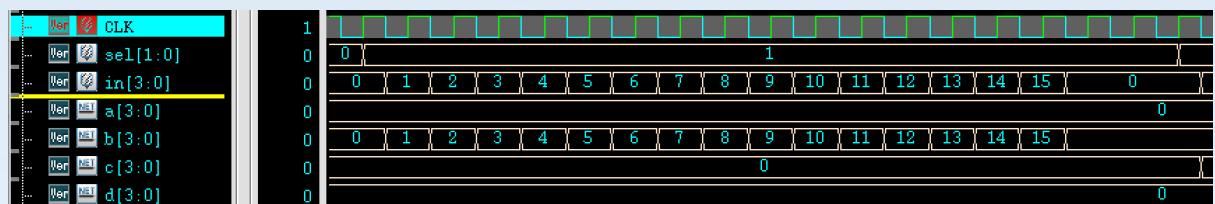
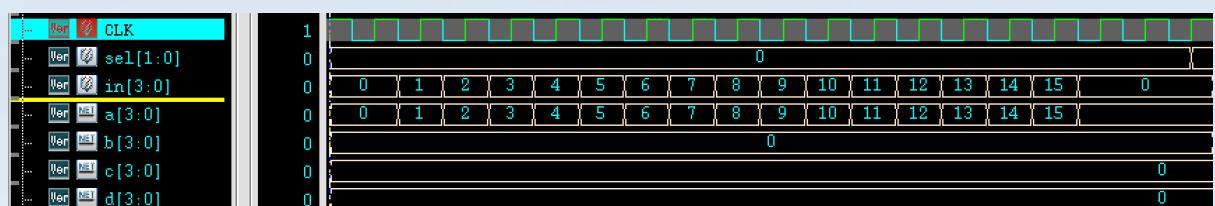
```
task simulate;
    if(sel === 2'b00) begin
        if(a !== in) begin
            pass = 0;
        end
    end
    else if(sel === 2'b01) begin
        if(b !== in) begin
            pass = 0;
        end
    end
    else if(sel === 2'b10) begin
        if(c !== in) begin
            pass = 0;
        end
    end
    else begin
        if(d !== in) begin
            pass = 0;
        end
    end
end
endtask
```

III. Wave

下圖先以完整波形圖的 pass 訊號，表示整個的運作是沒有問題的。



接著分別以 sel 為 00,01,10,11 時的波形圖展示各組的正確性。



11010EECS207001

Logic Design Lab

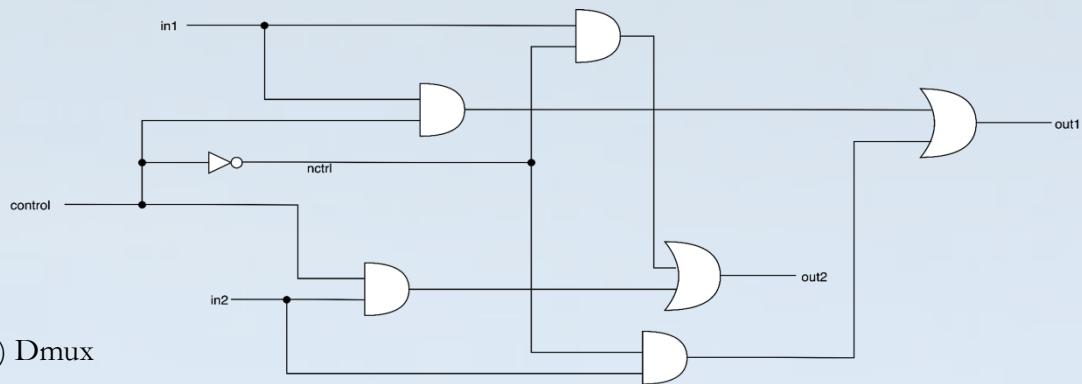
TEAM5 LAB1

2020.09.30

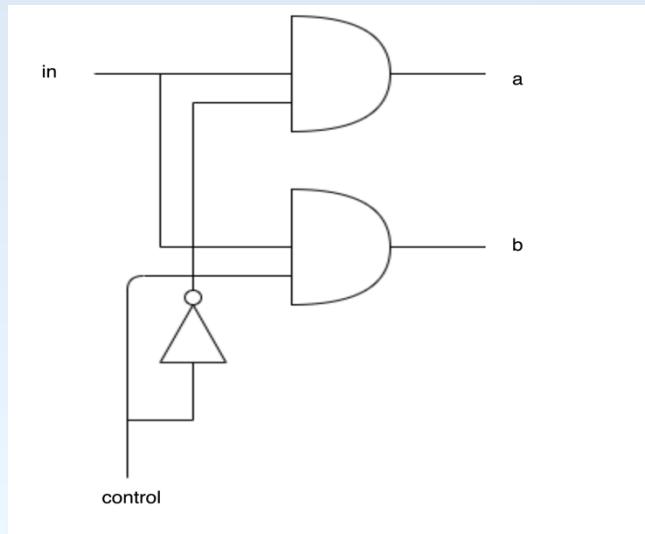
4-bit simple crossbar switch with MUX/DMUX

I. Logic Design Diagram

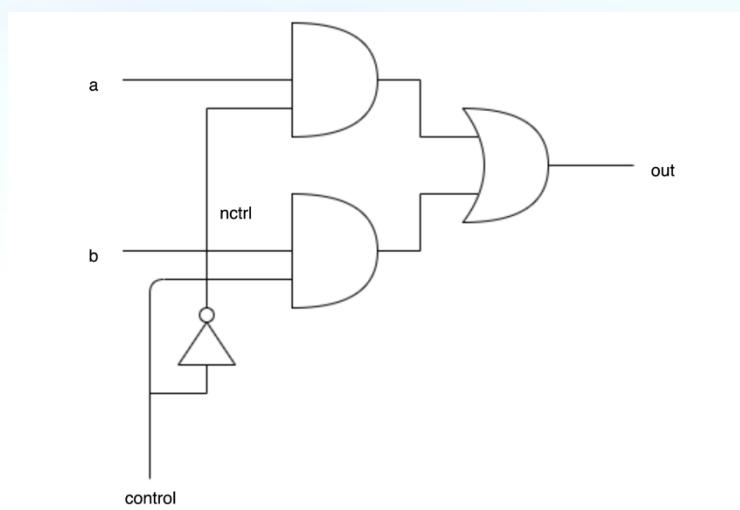
(i) 完整電路圖



(ii) Dmux



(iii) Mux



(iv) gate 省略方式說明：在 Dmux 和 mux 相接時，因為 Dmux 的 and gate 與 Mux 的 and gate 會連續且接相同的東西（Dmux 的 and gate 前會是 in 與 control，輸出的 b 會再與 control 相接於 Mux 的 and gate，因此可以省略一個 and gate，同理適用於 a 上），因此每一組可以省略兩個 and gate。Dmux 與 mux 的 control 訊號與 not gate 也可一起使用，可以再節省 not gate 數量，也因此合併時會是上面完整電路的樣子。

以完整電路圖說明：Dmux_in1 包含 not gate、與 in1 相關的兩個 and gate, Dmux_in2 包含 not gate 與兩個 in2 相關的 and gate。Mux_out1 包含 not gate, out1 前的 or gate 及 or gate 前連接的兩個 and gate；Mux_out2 則是包含 not gate, out2 前的 or gate 及更之前的兩個 and gate。

II. Explanation

依照上述已經簡化過的電路，直接做出 crossbar，這樣不用另外撰寫 Mux 和 Dmux 的 module，實際上也使用更少的 gate。

```

input [3:0] in1, in2;
input control;
output [3:0] out1, out2;

wire nctrl;
wire [4-1:0] in1_ctrl, in2_ctrl, in1_nctrl, in2_nctrl;

not not1(nctrl, control);
and and1[3:0](in1_ctrl, in1, control);
and and2[3:0](in2_ctrl, in2, control);
and and3[3:0](in1_nctrl, in1, nctrl);
and and4[3:0](in2_nctrl, in2, nctrl);
or or1[3:0](out1, in1_ctrl, in2_nctrl);
or or2[3:0](out2, in2_ctrl, in1_nctrl);

```

III. Testbench

本題的 TestBench 使用窮舉法。首先先將 control 值設為 0，in1,in2 分別設為 0000、0001，接著在每個 negedge clk 時將 in1,in2 分別加 1。在重複 16 次後所有 in1,in2 的值都已經測試過，因此將 control 改設為 1，並且在重複一次前面的動作，讓 in1,in2 再次從初始值開始跑過 16 次。

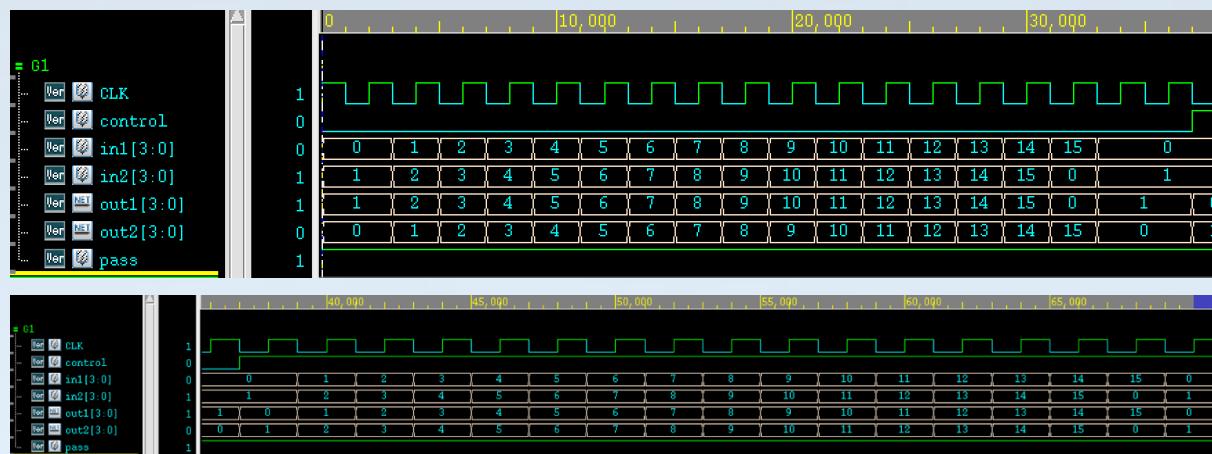
```
reg control = 1'b0;
reg [4-1:0] in1 = 4'b0000;
reg [4-1:0] in2 = 4'b0001;
repeat (2**4) begin
    @(posedge CLK)
        simulate;
    @(negedge CLK)
        in1 = in1 + 4'b1;
        in2 = in2 + 4'b1;
end
#4
control = 1'b1;
repeat (2**4) begin
    @(posedge CLK)
        simulate;
    @(negedge CLK)
        in1 = in1 + 4'b1;
        in2 = in2 + 4'b1;
end
```

另外，與前一小題相同，在 posedge clk 的地方使用 task simulate 去判斷執行結果的正確性，並搭配 pass 訊號，可以一目了然知道整個的結果。(詳細實作請見下圖)

```
task simulate; |
    if(control === 1'b0) begin
        if(out1 !== in2 || out2 !== in1) begin
            pass = 0;
        end
    end
    else begin
        if(out1 !== in1 || out2 !== in2) begin
            pass = 0;
        end
    end
endtask
```

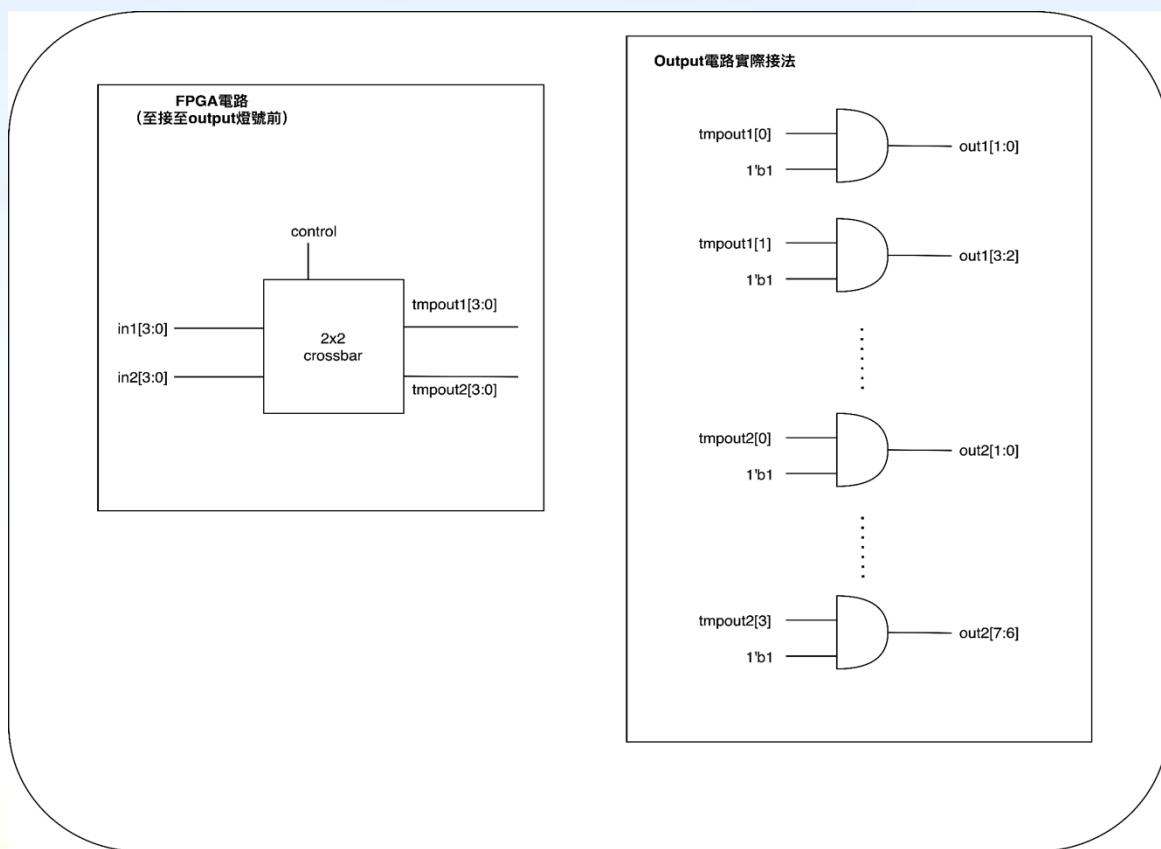
III. Wave

下圖為完整波形圖（分割為兩張），以 control 為 0 和 1 做為分界。也特別看出當 control 從 0 變成 1 時，out 的對應輸出馬上有跟著變動。另外 pass 也從頭到尾都是 1，代表運作皆正常。



V.FPGA 實作

(i) FPGA diagram



(ii) Explanation

首先透過 Crossbar_2x2_4bit module 取得應該要輸出的 out 值，這邊以 tmpout1, tmpout2 分別代表 out1, out2。接著因為題目要求，每一個 tmpout 接到 FPGA 上要以兩個燈號顯示，因此實際的 output out1, out2 大小為[7:0]而非[3:0]，然後透過 and gate，將 tmpout 與 1'b1 接至 out (詳細接法見下圖程式碼)。

```
wire [4-1] tmpout1, tmpout2; and and1[1:0](out1[1:0], tmpout1[0], 1'b1);
and and2[1:0](out1[3:2], tmpout1[1], 1'b1);
and and3[1:0](out1[5:4], tmpout1[2], 1'b1);
and and4[1:0](out1[7:6], tmpout1[3], 1'b1);
and and5[1:0](out2[1:0], tmpout2[0], 1'b1);
and and6[1:0](out2[3:2], tmpout2[1], 1'b1);
and and7[1:0](out2[5:4], tmpout2[2], 1'b1);
and and8[1:0](out2[7:6], tmpout2[3], 1'b1);

Crossbar_2x2_4bit cross(
    .in1(in1),
    .in2(in2),
    .control(control),
    .out1(tmpout1),
    .out2(tmpout2)
);
```

附上 I/O Port 圖檔表示 FPGA 上 input 與 output 實作。

▼ in1 (4)	IN				<input checked="" type="checkbox"/>	14	LVC MOS33*	▼	3.300				NONE	▼	NONE	▼	
in1[3]	IN				<input checked="" type="checkbox"/>	14	LVC MOS33*	▼	3.300				NONE	▼	NONE	▼	
in1[2]	IN				<input checked="" type="checkbox"/>	14	LVC MOS33*	▼	3.300				NONE	▼	NONE	▼	
in1[1]	IN				<input checked="" type="checkbox"/>	14	LVC MOS33*	▼	3.300				NONE	▼	NONE	▼	
in1[0]	IN				<input checked="" type="checkbox"/>	14	LVC MOS33*	▼	3.300				NONE	▼	NONE	▼	
▼ in2 (4)	IN				<input checked="" type="checkbox"/>	(Multiple)	LVC MOS33*	▼	3.300				NONE	▼	NONE	▼	
in2[3]	IN				<input checked="" type="checkbox"/>	34	LVC MOS33*	▼	3.300				NONE	▼	NONE	▼	
in2[2]	IN				<input checked="" type="checkbox"/>	14	LVC MOS33*	▼	3.300				NONE	▼	NONE	▼	
in2[1]	IN				<input checked="" type="checkbox"/>	W14	LVC MOS33*	▼	3.300				NONE	▼	NONE	▼	
in2[0]	IN				<input checked="" type="checkbox"/>	14	LVC MOS33*	▼	3.300				NONE	▼	NONE	▼	
▼ out1 (8)	OUT				<input checked="" type="checkbox"/>	U11	LVC MOS33*	▼	3.300	12	▼	SLUVV	▼	NONE	▼	FP_VTT_50	▼
out1[7]	OUT				<input checked="" type="checkbox"/>	V14	LVC MOS33*	▼	3.300	12	▼	SLOW	▼	NONE	▼	FP_VTT_50	▼
out1[6]	OUT				<input checked="" type="checkbox"/>	U14	LVC MOS33*	▼	3.300	12	▼	SLOW	▼	NONE	▼	FP_VTT_50	▼
out1[5]	OUT				<input checked="" type="checkbox"/>	U15	LVC MOS33*	▼	3.300	12	▼	SLOW	▼	NONE	▼	FP_VTT_50	▼
out1[4]	OUT				<input checked="" type="checkbox"/>	W18	LVC MOS33*	▼	3.300	12	▼	SLOW	▼	NONE	▼	FP_VTT_50	▼
out1[3]	OUT				<input checked="" type="checkbox"/>	V19	LVC MOS33*	▼	3.300	12	▼	SLOW	▼	NONE	▼	FP_VTT_50	▼
out1[2]	OUT				<input checked="" type="checkbox"/>	U19	LVC MOS33*	▼	3.300	12	▼	SLOW	▼	NONE	▼	FP_VTT_50	▼
out1[1]	OUT				<input checked="" type="checkbox"/>	E19	LVC MOS33*	▼	3.300	12	▼	SLOW	▼	NONE	▼	FP_VTT_50	▼
out1[0]	OUT				<input checked="" type="checkbox"/>	U16	LVC MOS33*	▼	3.300	12	▼	SLOW	▼	NONE	▼	FP_VTT_50	▼
▼ out2 (8)	OUT				<input checked="" type="checkbox"/>	(Multiple)	LVC MOS33*	▼	3.300	12	▼	SLOW	▼	NONE	▼	FP_VTT_50	▼
out2[7]	OUT				<input checked="" type="checkbox"/>	L1	LVC MOS33*	▼	3.300	12	▼	SLOW	▼	NONE	▼	FP_VTT_50	▼
out2[6]	OUT				<input checked="" type="checkbox"/>	P1	LVC MOS33*	▼	3.300	12	▼	SLOW	▼	NONE	▼	FP_VTT_50	▼
out2[5]	OUT				<input checked="" type="checkbox"/>	N3	LVC MOS33*	▼	3.300	12	▼	SLOW	▼	NONE	▼	FP_VTT_50	▼
out2[4]	OUT				<input checked="" type="checkbox"/>	P3	LVC MOS33*	▼	3.300	12	▼	SLOW	▼	NONE	▼	FP_VTT_50	▼
out2[3]	OUT				<input checked="" type="checkbox"/>	U3	LVC MOS33*	▼	3.300	12	▼	SLOW	▼	NONE	▼	FP_VTT_50	▼
out2[2]	OUT				<input checked="" type="checkbox"/>	W3	LVC MOS33*	▼	3.300	12	▼	SLOW	▼	NONE	▼	FP_VTT_50	▼
out2[1]	OUT				<input checked="" type="checkbox"/>	V3	LVC MOS33*	▼	3.300	12	▼	SLOW	▼	NONE	▼	FP_VTT_50	▼
out2[0]	OUT				<input checked="" type="checkbox"/>	V13	LVC MOS33*	▼	3.300	12	▼	SLOW	▼	NONE	▼	FP_VTT_50	▼

(iii) 實作成果展示：以 PDF 上的例子為例，並改變 control 值，確認燈號會轉變，也同時測試到所有燈號皆能正確亮起與熄滅。

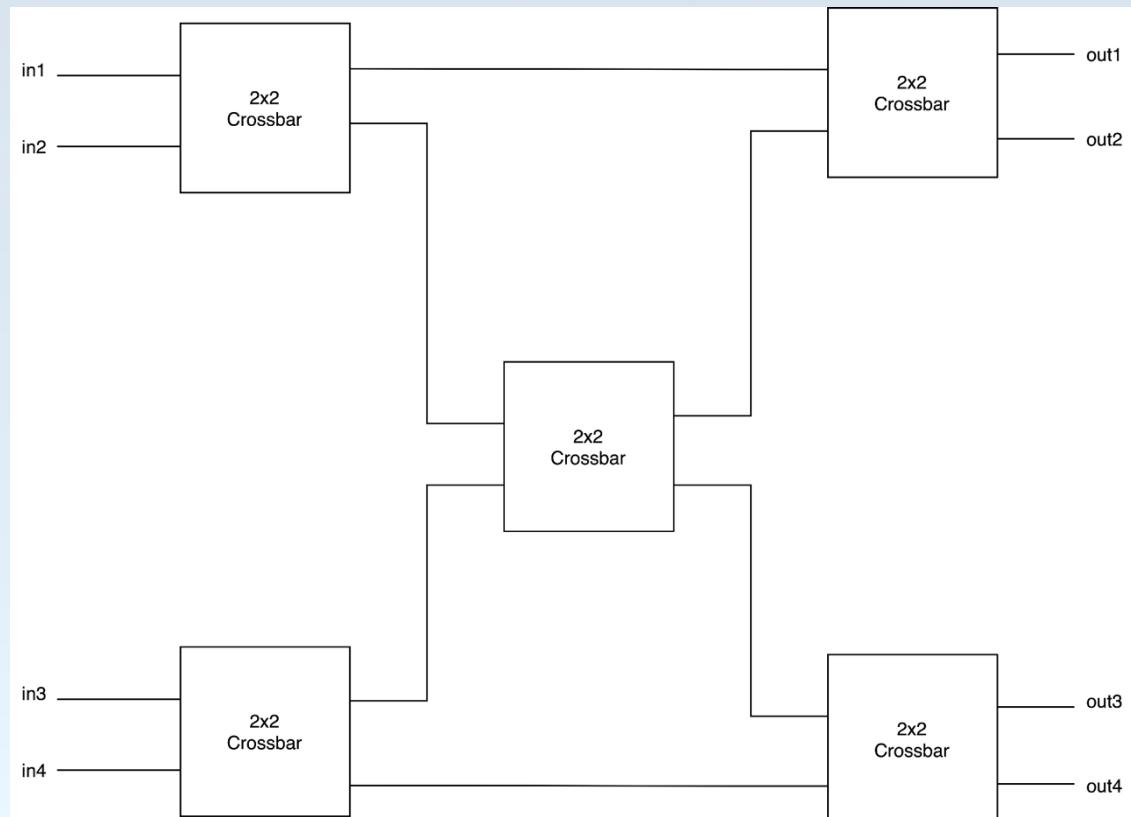


4-bit 4x4_crossbar with simple crossbar switch

I. Logic Design Diagram

(i) 2x2_crossbar 見前一小題 diagram 圖

(ii) 4x4_crossbar diagram



II. Explanation

本題有 $\text{in1}, \text{in2}, \text{in3}, \text{in4}$ ，依照排列組合 output 理應有 $4!=24$ 種排列方式，但仔細計算上表會發現僅有 20 種排列方式，代表有 4 種是 not routable 的，分別是 $\{\text{out1}, \text{out2}, \text{out3}, \text{out4}\} = \{\text{in4}, \text{in3}, \text{in2}, \text{in1}\}, \{\text{in4}, \text{in3}, \text{in1}, \text{in2}\}, \{\text{in3}, \text{in4}, \text{in2}, \text{in1}\}, \{\text{in3}, \text{in4}, \text{in1}, \text{in2}\}$
(詳細對應列表如下表)

control	out1	out2	out3	out4
0	4	2	3	1
1	4	1	3	2
10	3	2	4	1
11	3	1	4	2
100	1	2	3	4
101	2	1	3	4
110	1	2	4	3
111	2	1	4	3
1000	2	4	3	1
1001	1	4	3	2
1010	2	3	4	1
1011	1	3	4	2
1100	2	1	3	4
1101	1	2	3	4

1110	2	1	4	3
1111	1	2	4	3
10000	4	2	1	3
10001	4	1	2	3
10010	3	2	4	4
10011	3	1	4	4
10100	1	2	3	3
10101	2	1	3	3
10110	1	2	1	4
10111	2	1	2	4
11000	2	4	1	3
11001	1	4	2	3
11010	2	3	1	4
11011	1	3	2	4
11100	2	1	4	3
11101	1	2	4	3
11110	2	1	3	4
11111	1	2	3	4

(out 底下數字代表 in，1 表示 in1, 2 表示 in2，依此類推)

III. Testbench

本題 testbench 使用窮舉法，將 control 先設為 00000，四個 in 分別設為 0000、0001、0010、0011 四個不同數值，方便確認 output 是否正確。這個 testbench 分為兩個 repeat，內層的 repeat 會在每個 negedge clk 將 in 的值加 1，每一個 control 值跑 16 次。外層的 repeat 則是用來改變 control 值，總共會跑 32 次，每次會結束會將 control 值加 1，將 control 值由 00000 一路改變到 11111，確保每一種 cross 組合都有被測試過。

```

reg CLK = 1;
reg [5-1:0] control = 5'b00000;
reg [4-1:0] in1 = 4'b0000;
reg [4-1:0] in2 = 4'b0001;
reg [4-1:0] in3 = 4'b0010;
reg [4-1:0] in4 = 4'b0011;
wire [4-1:0] out1, out2, out3, out4;
reg pass = 1'b1;

repeat (2**5) begin
    repeat (2**4) begin
        @(posedge CLK)
        simulate;
        @(negedge CLK)
        in1 = in1 + 1'b1;
        in2 = in2 + 1'b1;
        in3 = in3 + 1'b1;
        in4 = in4 + 1'b1;
    end
    #4
    control = control + 1'b1;
end

```

另外，在 posedge 時，與前面小題相同，使用 task simulate 去判斷是否有正確的將 in 值轉移至 out 值。尤其這題組和眾多，用肉眼並不容易確認，更凸顯使用這種方式搭配 pass 訊號的好處。(由於組數眾多，下圖僅以 task 頭尾為例)

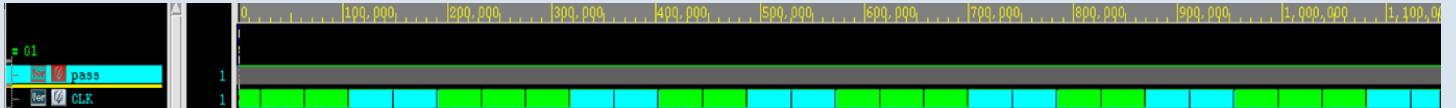
```

task simulate;
    if(control === 5'b00000) begin
        if(out1!==in4 || out2!==in2 || out3!==in3 || out4!==in1) begin
            pass = 0;
        end
        else pass = 1;
    end
    else if(control === 5'b00001) begin
        if(out1!==in4 || out2!==in1 || out3!==in3 || out4!==in2) begin
            pass = 0;
        end
        else pass = 1;
    end
else if(control === 5'b11111) begin
    if(out1==in1 || out2==in2 || out3==in3 || out4==in4) begin
        pass = 0;
    end
    else pass = 1;
end
else begin
    pass = 0;
end
endtask

```

III. Wave

因總數眾多，下圖先以整體的 pass 為輔助，確定各組都正確。



接著舉出數組為例（有 00000->00001, 01100->01101->01110, 11101->11110->11111），可以看看出當 control 值一轉變時 out 值有馬上跟著轉變，若慢慢比對也可以看出 out 對 in 的關係皆正確。



11010EECS207001

Logic Design Lab

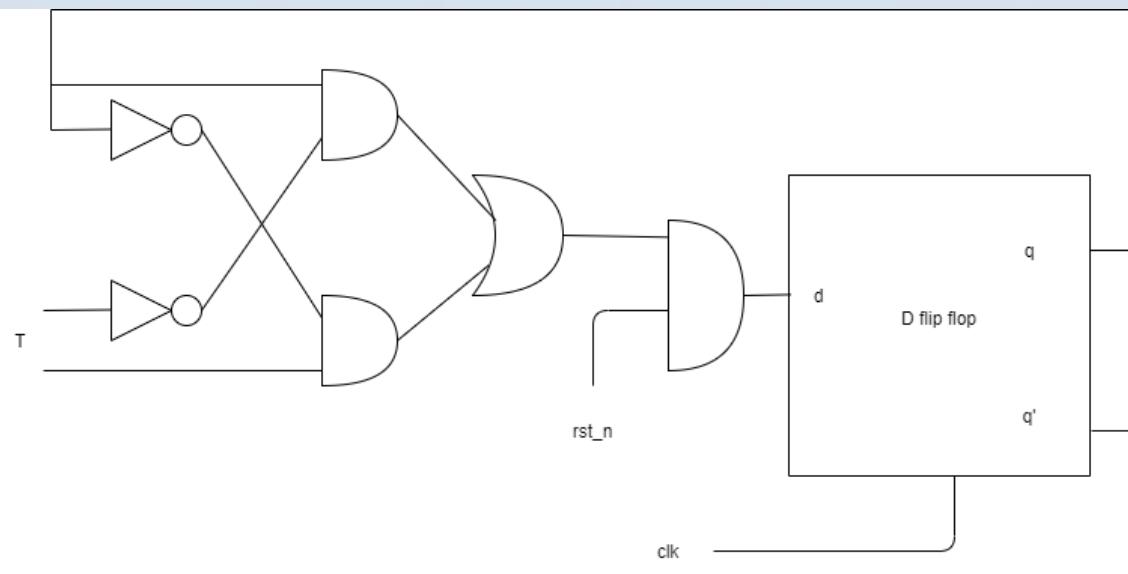
TEAM5 LAB1

2020.09.30

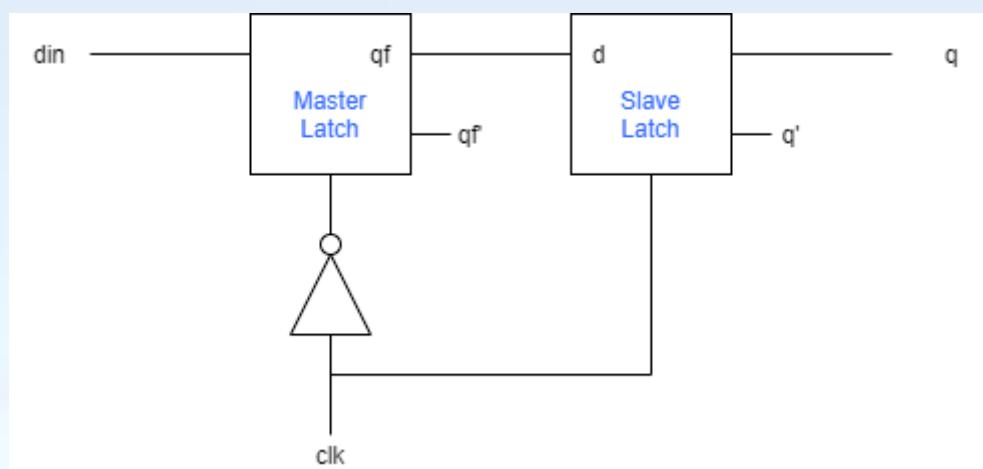
1-bit toggle flip flop (TFF)

I. Logic Design Diagram

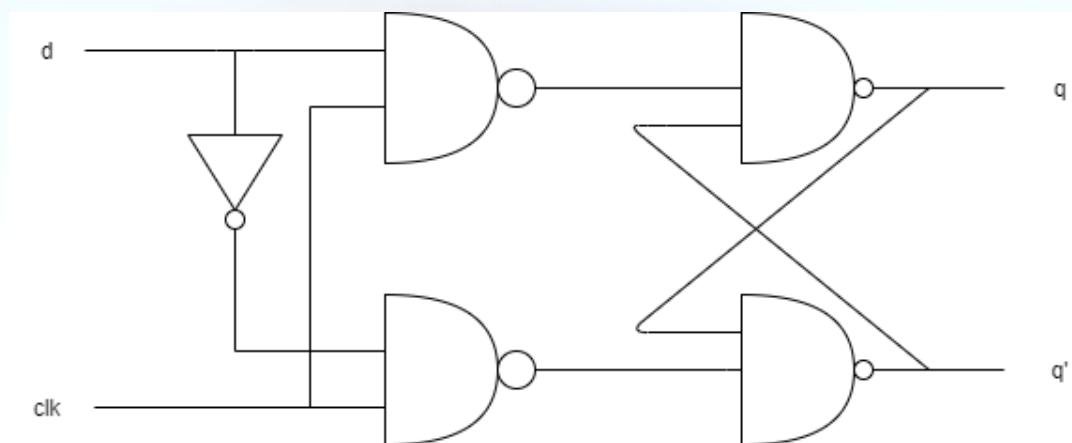
(i) 完整電路圖



(ii) D flip flop:



(iii) D Latch:



II. Explanation

將 q 跟輸入 t 做 XOR，這裡使用 $xor(q,t) = q't + t'q$ 輸出，再跟 rst_n 做 and，檢查是否需要 reset，最後作為輸入訊號 din 進入 d flip flop 的 module，最後輸出的 q ，會作為 input 與 t 進入下一輪運算，輸出時脈+1 後的 $q+1$ 。

din 進入 d flip flop 的 Master Latch module，作為 latch 的 d 輸入訊號，配合反向 clk 輸出 qf 作為 q 再進入 Slave Latch module 作為 d ，最後輸出 output q ，再經由以上循環與 t 結合得到下一輪的 output $q+1$ 。(可見下方 code 實作)

```
module Toggle_Flip_Flop(clk, q, t, rst_n);  
    input clk;  
    input t;  
    input rst_n;  
    output q;  
    wire a,b,tqxor;  
    wire tnot,qnot;  
    wire din;  
    not not1(tnot,t);  
    not not2(qnot,q);  
    and and1(a,t,qnot);  
    and and2(b,q,tnot);  
    or or1(tqxor,a,b);  
    and and3(din,rst_n,tqxor);  
    DFF dff(  
        .clk(clk),  
        .d(din),  
        .q(q)  
    );  
endmodule  
  
module DFF(clk, d, q);  
    input clk;  
    input d;  
    output q;  
    wire nclk, q1;  
    not (nclk, clk);  
    D_Latch master(  
        .e(nclk),  
        .d(d),  
        .q(q1)  
    );  
    D_Latch slave(  
        .e(clk),  
        .d(q1),  
        .q(q)  
    );  
endmodule  
  
module D_Latch(e, d, q);  
    input e;  
    input d;  
    output q;  
    wire dnot,qnot,up,down;  
    not not3(dnot,d);  
    nand nand1(up,d,e);  
    nand nand2(down,dnot,e);  
    nand nand3(q,up,qnot);  
    nand nand4(qnot,down,q);  
endmodule
```

III. Test bench

```
always #5 clk=~clk;  
always #13 t = ~t;  
initial begin  
{rst_n,clk,t}<=0;  
repeat(2) @(posedge clk);  
rst_n<=1;  
#50  
rst_n=0;  
#50  
rst_n=1;  
end  
endmodule
```

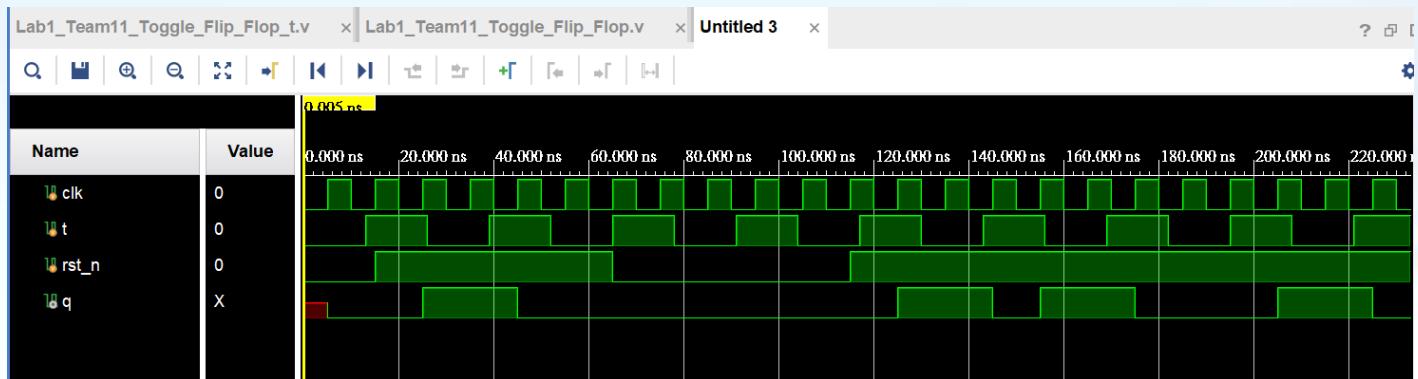
因為這題會用到上個狀態的 q 配合 clk 時脈得到

$q+1$ ，不適合使用窮舉法，所以 t 以時脈的方式輸入，rst_n 一開始為 0，之後設為 1 後，便可以正常運作。中間 65ns~115ns 之間再將 rst_n 歸零一段時間，目的就是要確認它有沒有正常運作。

最後去觀察 clk 的 posedge 可發現，當 clk 為正緣觸發的時間點，且在 $rst_n=1$ 之情況下，當 $t=0$ ，q 不變，維持上個狀態，當 $t=1$ ，q 轉變為 q' 。另外，在配合觀察波形圖可發現，中間

65ns~115ns 之間 $rst_n=0$ ，中間本應在 65ns 轉變為 1 的 q 值並沒有出現，持續到 $rst_n=1$ 後才正常輸出 q 值的波型，因此藉由以上的檢查式來撰寫 testbench。

III. Wave



What do we learn in this Lab

第一次的 lab 最有趣的部分應該是 Crossbar 的部分。一開始並沒有想到用 mux 和 Dmux 的方法，因此發現了上面經過簡化的方式。但看到題目要求使用 mux 和 Dmux 時，一直很困惑明明會使用更多的 gate，為什麼要用這樣的方式呢？再經過細細比較之後，我們終於發現其實那個電路也是用 mux 跟 Dmux 組成，只不過因為相接後會有可以省略的 gate，因此從圖上就很難直接看出是 mux 與 Dmux 的變形。從這個 Lab 看見了電路設計時的廣大奧秘，這些電路的化簡更是我們邏輯設計與邏輯設計實驗這兩堂課學完後應該有所了解的技能。

另外，這次的 testbench 在 4x4_crossbar 遇到些許的困難。雖然我們已經使用 pass 訊號去判斷正確性，但在撰寫 task simulate 時，仍然必須透過人工的方式確認每一組 if-else 的條件。設計過程中也多次遇到其實 module 設計是正確的，但因為 if-else 的條件不小心設錯，導致誤以為設計有誤。因此想要透過這個 report 向助教及教授詢問，以這題來說有沒有更好的 testbench 寫法呢？或是我們在思考 in-out 關係時是否有更好的方式呢（這次是一個一個人工判斷）？

Cooperation

(依學號排列)

106062304 黃鈺舒：

Dmux 電路圖繪製、Dmux module 設計及撰寫、simple_crossbar 電路圖繪製、
TFF 電路圖繪製、TFF module 與 testbench 設計及撰寫。

108062213 顏浩昀：

Dmux testbench 設計及撰寫、simple_crossbar 電路圖繪製、simple_crossbar
module 與 testbench 設計及撰寫、4x4_crossbar 電路圖繪製、4x4_crossbar module
與 testbench 設計及撰寫、FPGA 實作。