Lab 1 Gate-Level Modeling

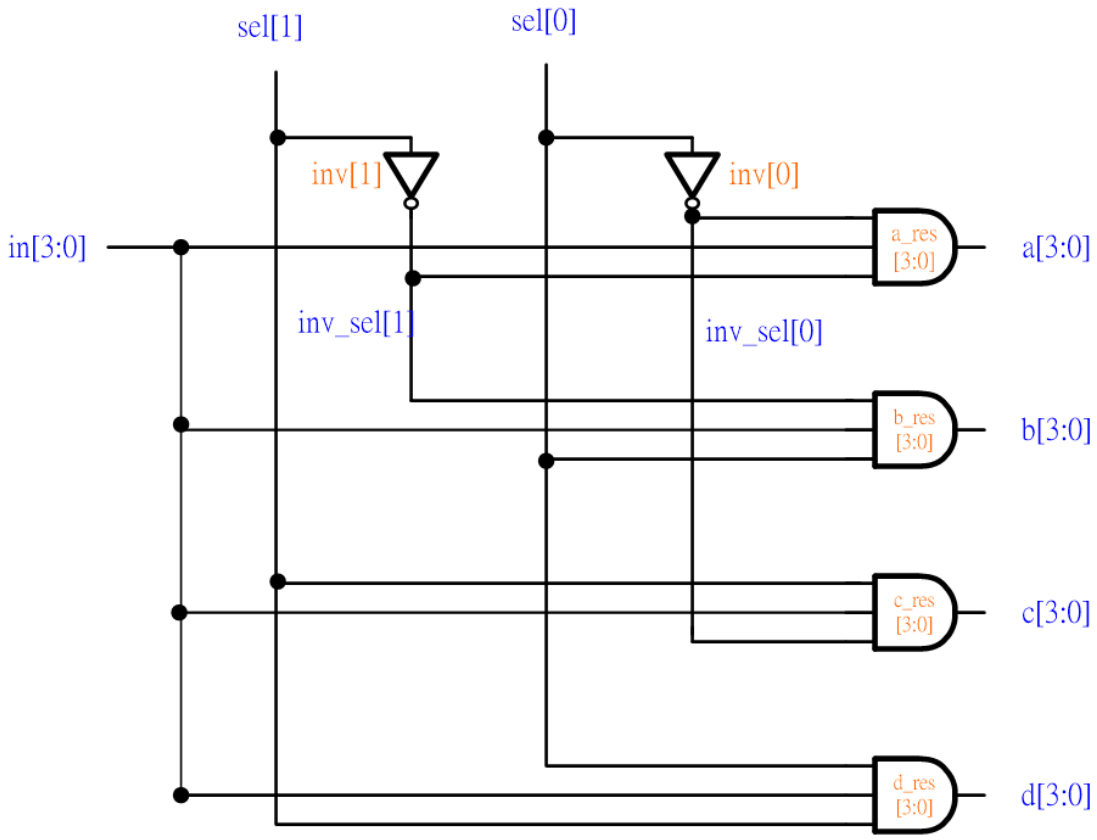
Lab Report

組長:劉奇泓 109033135

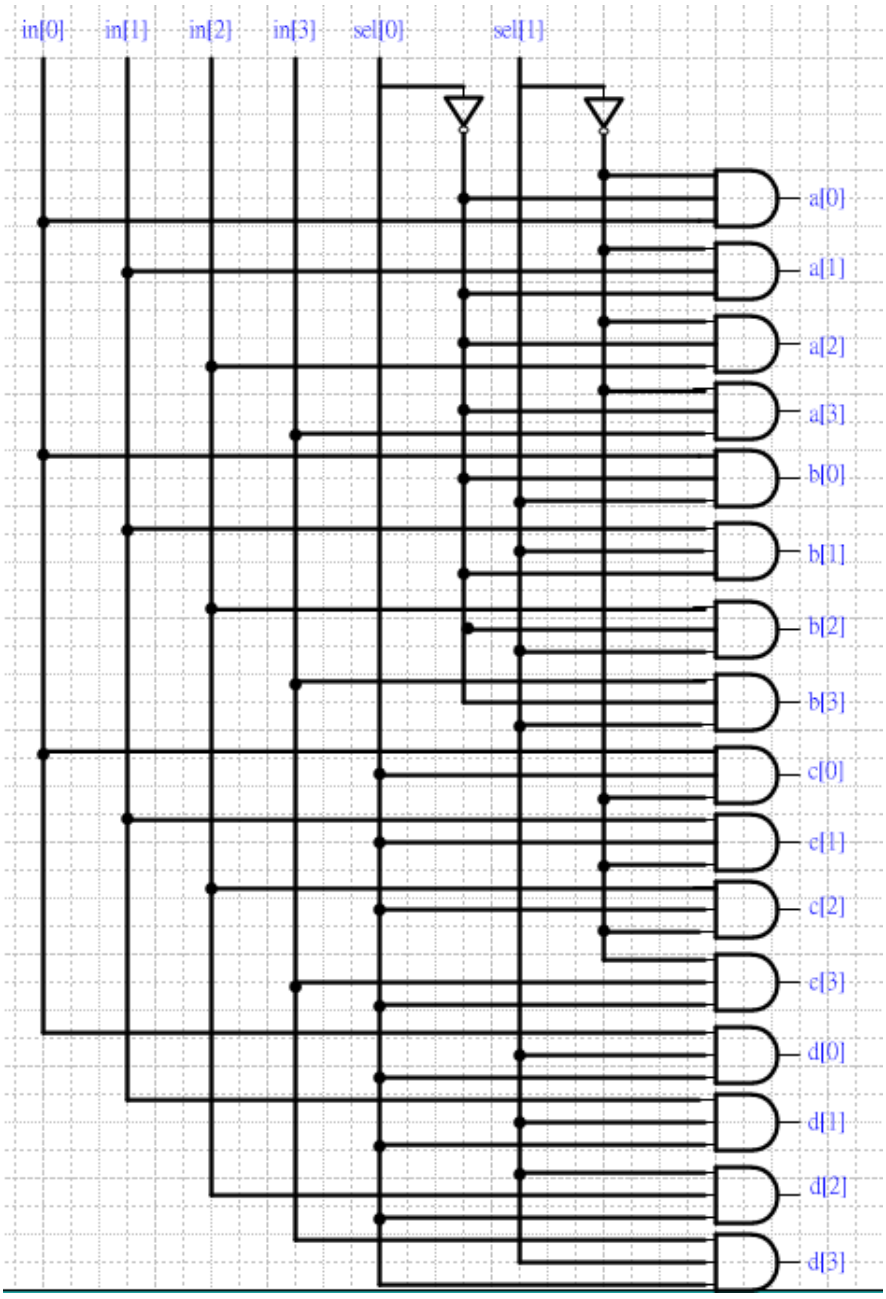
組員:洪聖祥 109062315

1. 4-bit 1-to-4 de-multiplexer (DMUX)

I. 邏輯設計圖



Detail:



II. Verilog 程式碼

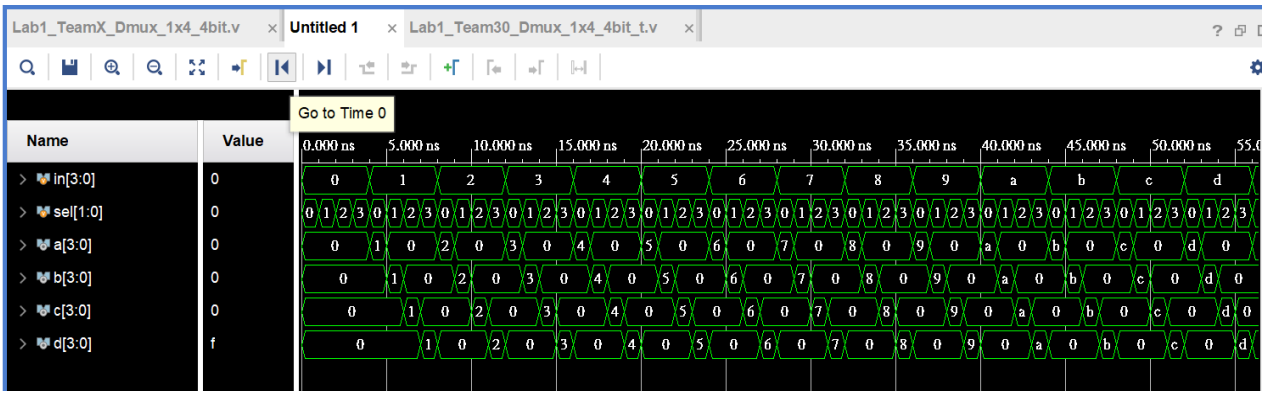
- Lab1_Team30_Dmux_1x4_4bit.v

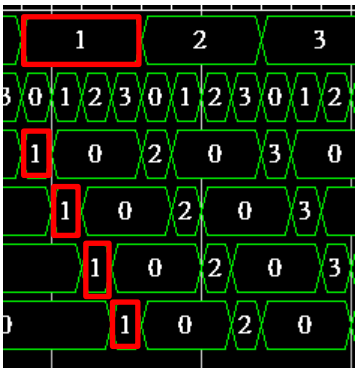
```
`timescale 1ns/1ps

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 [1:0] inv_sel;
not inv[1:0](inv_sel, sel);
// only when the input sel is 2'b00 , output a will be the same signal as input in while other outputs are 0
and a_res[3:0](a, inv_sel[1], inv_sel[0], in);
// only when the input sel is 2'b01 , output b will be the same signal as input in while other outputs are 0
and b_res[3:0](b, inv_sel[1], sel[0], in);
// only when the input sel is 2'b10 , output c will be the same signal as input in while other outputs are 0
and c_res[3:0](c, sel[1], inv_sel[0], in);
// only when the input sel is 2'b11 , output d will be the same signal as input in while other outputs are 0
and d_res[3:0](d, sel[1], sel[0], in);
endmodule
```

- testbench

```
module Dmux_1x4_4bit_t;
reg [3:0] in = 4'b0000;
reg [1:0] sel = 2'b00;
wire [3:0] a, b, c, d;
Dmux_1x4_4bit dm(in, a, b, c, d, sel);
initial begin
// since in is 4bits there are 2**4 probabilities
    repeat (2 ** 4) begin
//since sel is 2bits there are 4 probabilities
        repeat (4)
//I set every 1ns sel accumulates 1    When we are back to this loop again, sel will overflow and it is just 0 again
         #1 sel = sel + 2'b01;
        in = in + 4'b0001;
    end
    $finish;
end
endmodule
```

III. 波形圖





當 sel = 2’b00 時，a 的值必須等於 in 其餘為 0

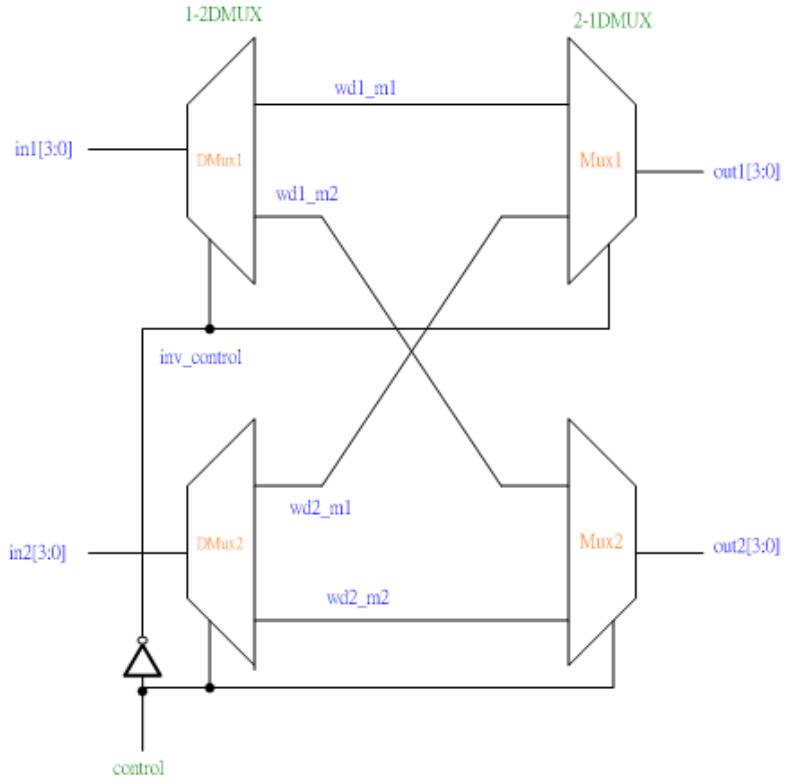
當 sel = 2’b01 時，b 的值必須等於 in 其餘為 0

當 sel = 2’b10 時，c 的值必須等於 in 其餘為 0

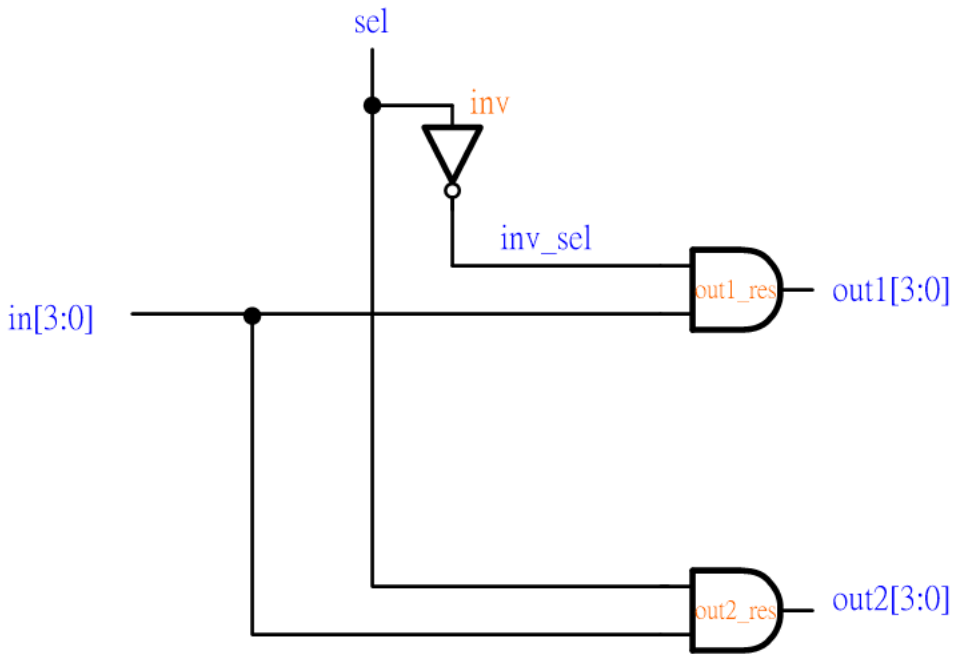
當 sel = 2’b11 時，d 的值必須等於 in 其餘為 0

2. 4-bit simple crossbar switch with MUX/DMUX

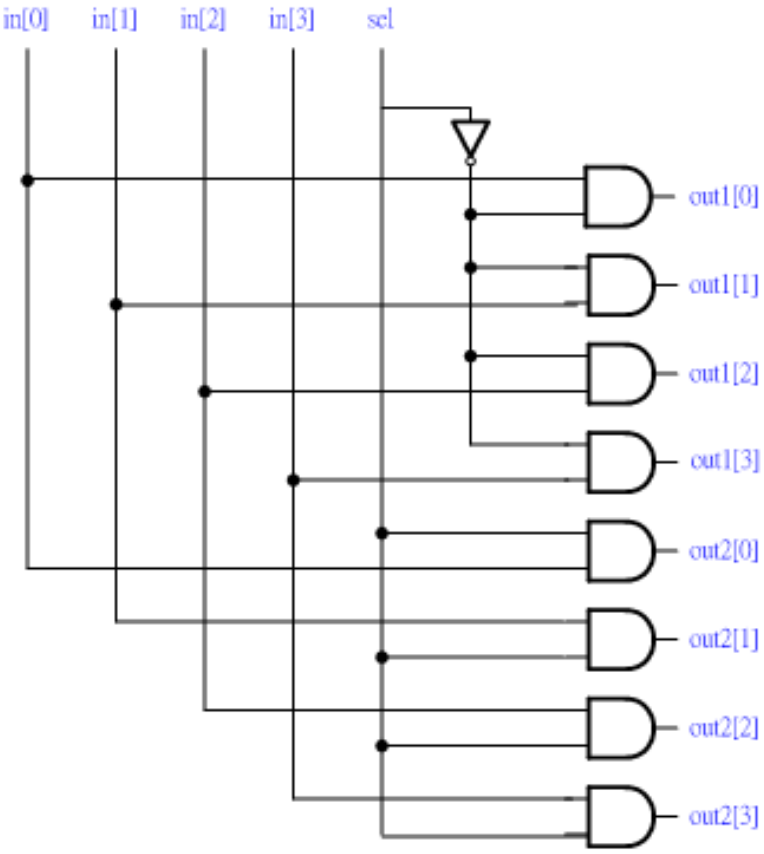
I. 邏輯設計圖



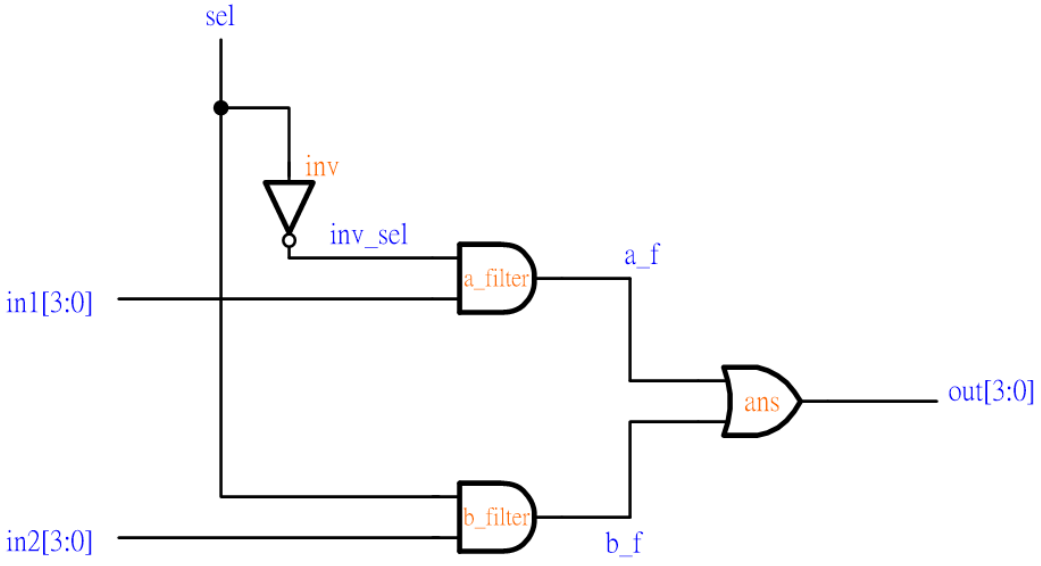
- 1-to-2 DMUX:



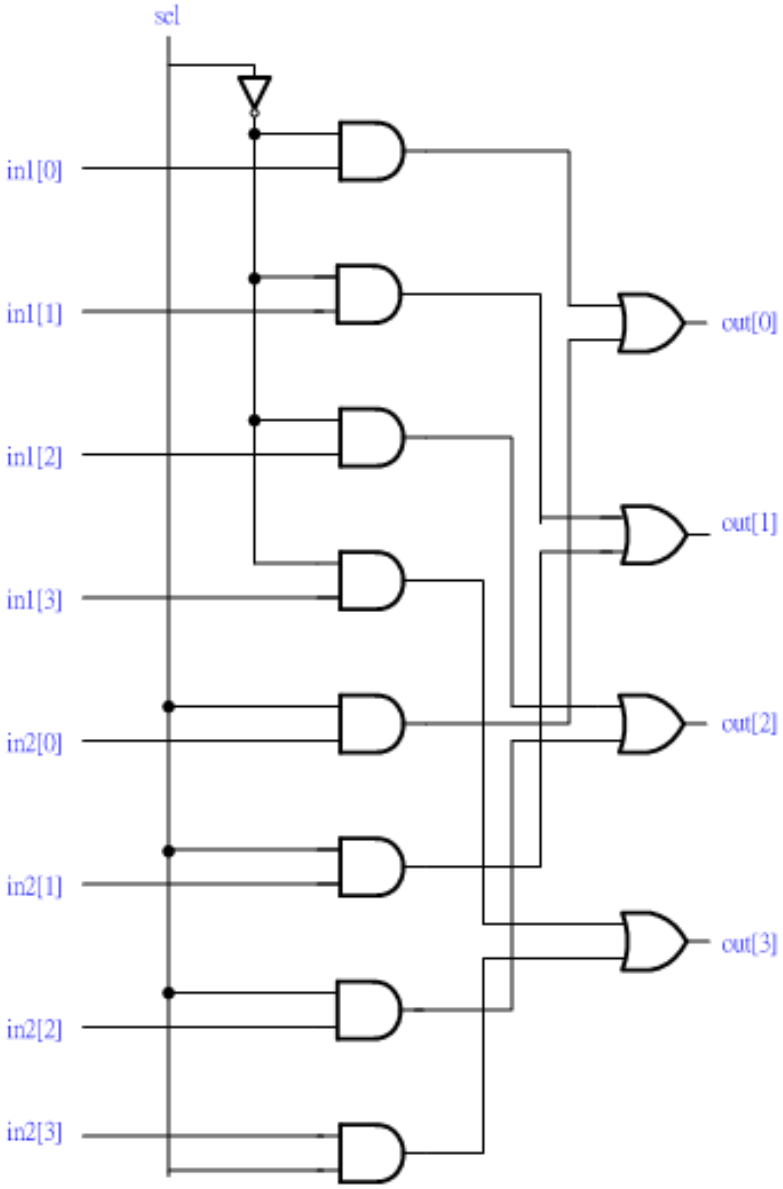
Detail:



- 2-to-1 MUX:



Detail:



II.Verilog 程式碼

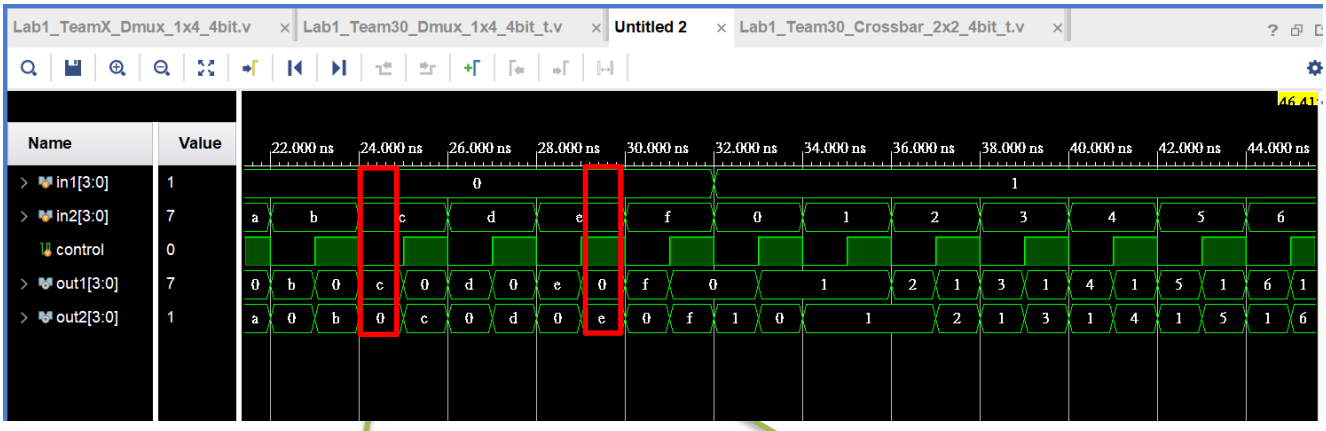
- Lab1_Team30_Crossbar_2x2_4bit.v

```
`timescale 1ns/1ps

module Crossbar_2x2_4bit(in1, in2, control, out1, out2);
input [3:0] in1, in2;
input control;
output [3:0] out1, out2;
wire inv_control;
not inv(inv_control, control);
wire [3:0] wd1_m1, wd1_m2, wd2_m1, wd2_m2;
// reuses the module written below
DMux_1x2_4bit DMux1(in1, wd1_m1, wd1_m2, inv_control);
DMux_1x2_4bit DMux2(in2, wd2_m1, wd2_m2, control);
Mux_2x1_4bit Mux1(wd1_m1, wd2_m1, out1, inv_control);
Mux_2x1_4bit Mux2(wd1_m2, wd2_m2, out2, control);
endmodule

module Mux_2x1_4bit(in_1, in_2, out, sel);
input [3:0] in_1, in_2;
input sel;
output [3:0] out;
wire inv_sel;

wire [3:0] a_f, b_f;
not inv(inv_sel, sel);
// only when sel is 0 output out will be the same as input in_1
and a_filter[3:0](a_f, in_1, inv_sel);
// only when sel is 1 output out will be the same as input in_2
and b_filter[3:0](b_f, in_2, sel);
// since sel is either 0 or 1 , the signal of output out depends on input sel which chooses the input in_1 or in_2
or ans[3:0](out, a_f, b_f);
endmodule

module DMux_1x2_4bit(in, out1, out2, sel);
input [3:0] in;
input sel;
output [3:0] out1, out2;
wire inv_sel;
not inv(inv_sel, sel);
// only when the input sel is 0, output out1 will be the same signal as input in while the other output is 0
and out1_res[3:0](out1, in, inv_sel);
// only when the input sel is 1, output out2 will be the same signal as input in while the other output is 0
and out2_res[3:0](out2, in, sel);
endmodule
```

- testbench

```
module Crossbar_2x2_4bit_t;
reg [3:0] in1 = 4'b0000;
reg [3:0] in2 = 4'b0000;
reg control = 1'b0;
wire [3:0] out1, out2;
Crossbar_2x2_4bit crsbar(in1, in2, control, out1, out2);
// try all the probabilities , there are 2^4 * 2^4 *2 probabilites
initial begin
// since in1 is 4bits there are 2**4 probabilities
    repeat (2 ** 4) begin
// since in2 is 4bits there are 2**4 probabilities
        repeat (2 ** 4) begin
// since control is 1bit there are only 2 probabilites
            repeat (2)
// I set every 1ns the signal of control plus 1
                #1 control = control + 1'b1;
            in2 = in2 + 4'b0001;
        end
        in1 = in1 + 4'b0001;
    end
    $finish;

end
endmodule
```

III. 波形圖

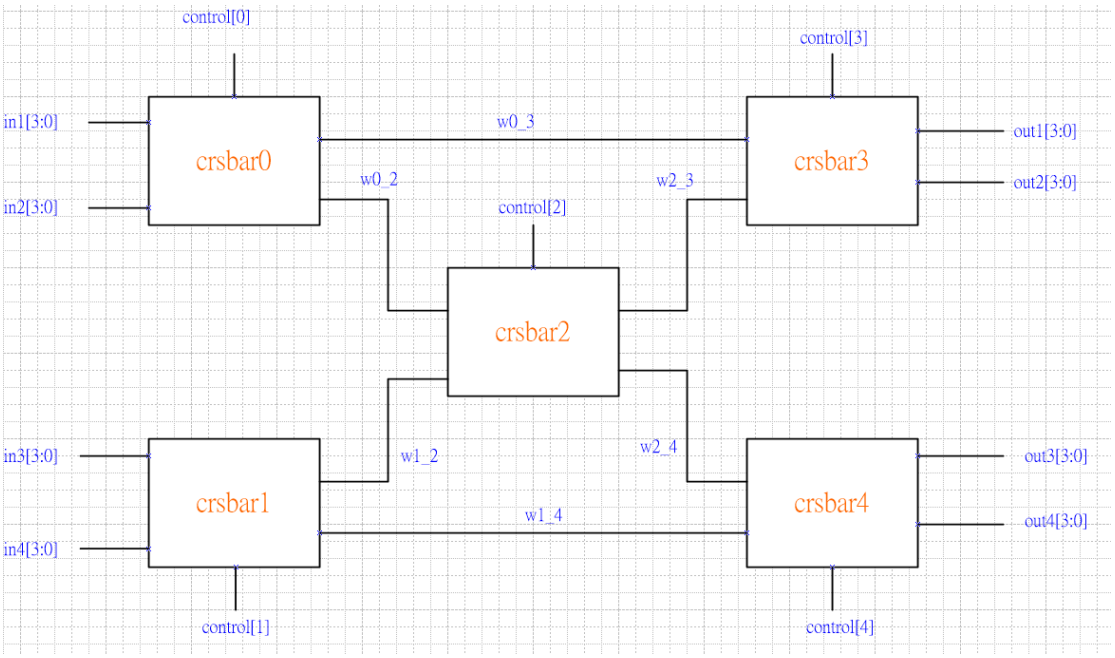


當 control = 0， out2 的值等於 in1 的值； out1 的值等於 in2 的值

當 control = 1， out2 的值等於 in2 的值； out1 的值等於 in1 的值

3. 4-bit 4x4crossbar with simple crossbar switch

I. 邏輯設計圖



- crsbar ：Question 2 製作的 4-bit simple crossbar

II.Verilog 程式碼

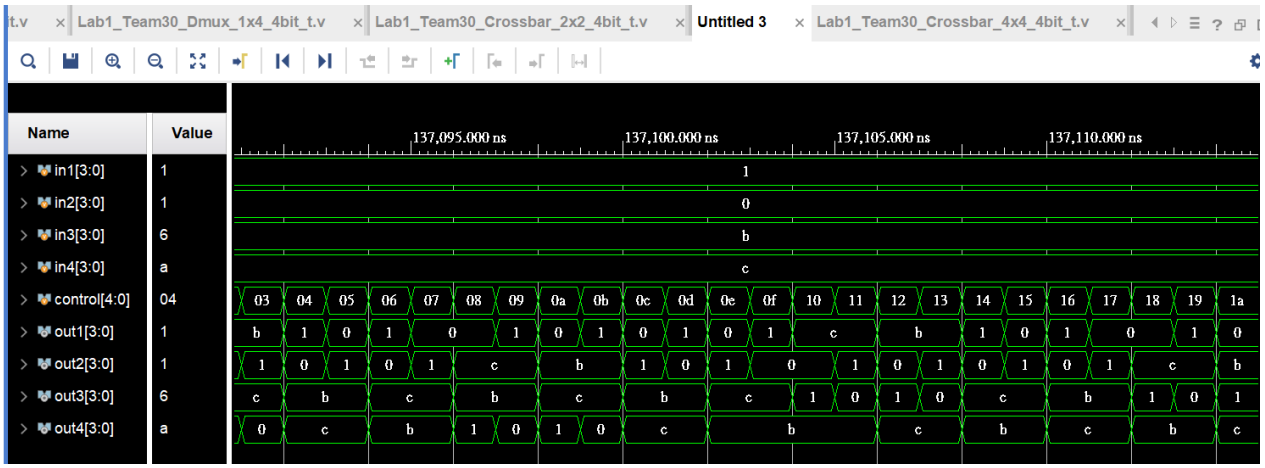
- Lab1_Team30_Crossbar_4x4_4bit.v

```
`timescale 1ns/1ps

module Crossbar_4x4_4bit(in1, in2, in3, in4, out1, out2, out3, out4, control);
input [3:0] in1, in2, in3, in4;
input [4:0] control;
output [3:0] out1, out2, out3, out4;
wire [3:0] w0_3, w0_2, w1_2, w1_4, w2_3, w2_4;
//reuses the module Crossbar_2x2_4bit written below
Crossbar_2x2_4bit crsbar0 (in1, in2, control[0], w0_3, w0_2);
Crossbar_2x2_4bit crsbar1 (in3, in4, control[1], w1_2, w1_4);
Crossbar_2x2_4bit crsbar2 (w0_2, w1_2, control[2], w2_3, w2_4);
Crossbar_2x2_4bit crsbar3 (w0_3, w2_3, control[3], out1, out2);
Crossbar_2x2_4bit crsbar4 (w2_4, w1_4, control[4], out3, out4);
endmodule

// reuse the modules in the previous question
module Crossbar_2x2_4bit(in1, in2, control, out1, out2);
input [3:0] in1, in2;
input control;
output [3:0] out1, out2;
not inv(inv_control, control);
wire [3:0] wd1_m1, wd1_m2, wd2_m1, wd2_m2;
DMux_1x2_4bit DMux1(in1, wd1_m1, wd1_m2, inv_control);
DMux_1x2_4bit DMux2(in2, wd2_m1, wd2_m2, control);
Mux_2x1_4bit Mux1(wd1_m1, wd2_m1, out1, inv_control);
Mux_2x1_4bit Mux2(wd1_m2, wd2_m2, out2, control);
endmodule

module Mux_2x1_4bit(in_1, in_2, out, sel);
input [3:0] in_1, in_2;
input sel;
output [3:0] out;
wire inv_sel;
wire [3:0] a_f, b_f;
not inv(inv_sel, sel);
and a_filter[3:0](a_f, in_1, inv_sel);
and b_filter[3:0](b_f, in_2, sel);
or ans[3:0](out, a_f, b_f);
endmodule



module DMux_1x2_4bit(in, out1, out2, sel);
input [3:0] in;
input sel;
output [3:0] out1, out2;
wire inv_sel;
not inv(inv_sel, sel);
and out1_res[3:0](out1, in, inv_sel);
and out2_res[3:0](out2, in, sel);
endmodule
```

- testbench

```
module Crossbar_4x4_4bit_t();
reg [3:0] in1 = 4'b0000;
reg [3:0] in2 = 4'b0000;
reg [3:0] in3 = 4'b0000;
reg [3:0] in4 = 4'b0000;
reg [4:0] control = 5'b00000;
wire [3:0] out1, out2, out3, out4;
Crossbar_4x4_4bit crsbar(in1, in2, in3, in4, out1, out2, out3, out4, control);
// try all the probabilities , there are 2^4 * 2^4 * 2 ^4 * 2^4 * 2^5probabilites
initial begin
// since in1 is 4bits there are 2**4 probabilities
    repeat (2 ** 4) begin
// since in2 is 4bits there are 2**4 probabilities
        repeat (2 ** 4) begin
// since in3 is 4bits there are 2**4 probabilities
            repeat (2 ** 4) begin
// since in4 is 4bits there are 2**4 probabilities
                repeat (2 ** 4) begin
// since control is 5bits there are 2**5 probabilities
                    repeat (2 ** 5)

// I set it every 1ns the signal of control pluses 1
                    #1 control = control + 5'b00001;
                    in4 = in4 + 4'b0001;
                end
                in3 = in3 + 4'b0001;
            end
            in2 = in2 + 4'b0001;
        end
        in1 = in1 + 4'b0001;
    end
    $finish;
end
endmodule
```

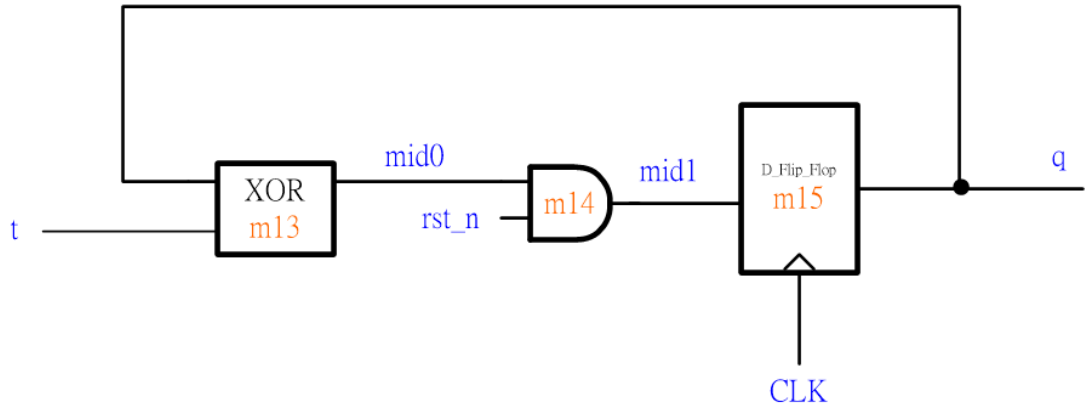
III. 波形圖



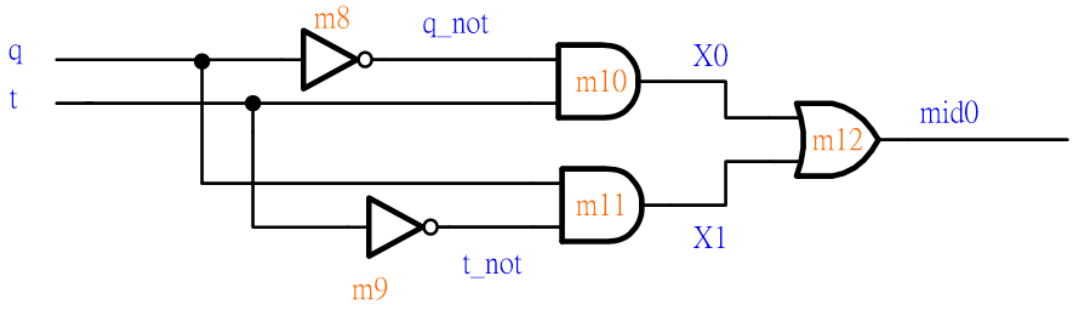
input 有 4 個，理應當 output 會有 24 種組合。但從波形圖比對得知，有 4 種可能不會出現。當 in1 和 in2 的訊號同時傳到 out3 和 out4 以及當 in3 和 in4 的訊號同時傳到 out1 和 out2 在 4*4crossbar 中是不可能出現的(共 4 種可能不會出現)。從邏輯設計圖得知，in1 或 in2 經過 crsbar0 的 output 一定是 w0_3 和 w0_2，但 w0_3 是接到 crsbar3，因此不可能兩個訊號都經過 crsbar4 而得到 out3 和 out4。同理 in3 和 in4。

4. 1-bit toggle flip flop (TFF)

I. 邏輯設計圖



- D_Flip_Flop:使用 basic question 2 做出的 DFF
- XOR:



II.Verilog 程式碼

- Lab1_Team30_Toggle_Flip_Flop.v

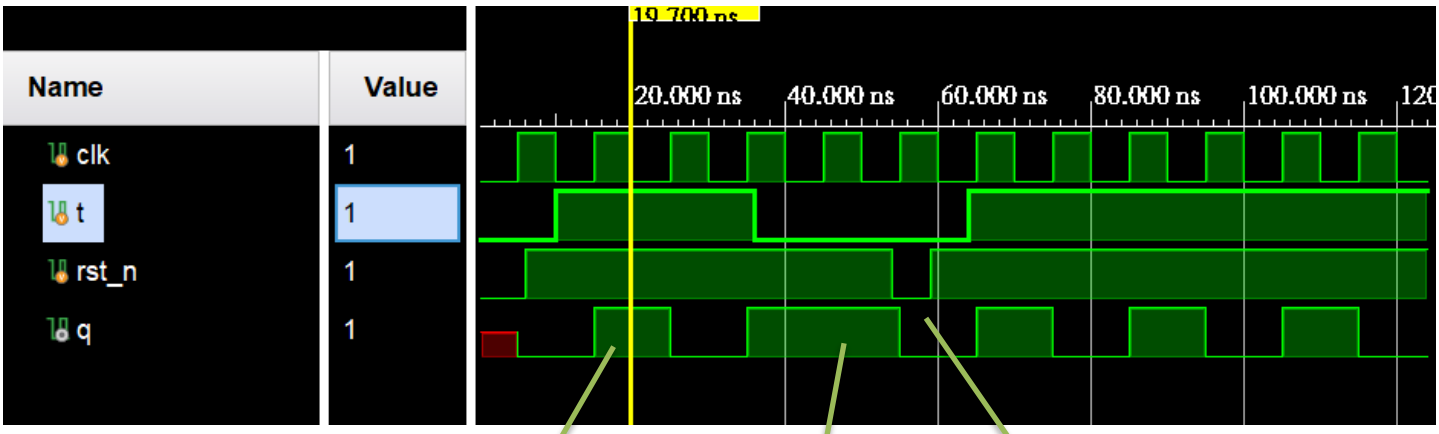
```
module T_Flip_Flop(clk, q, t, rst_n);
  input clk;
  input t;
  input rst_n;
  output q;

  wire mid0, mid1;

  XOR m13(q, t, mid0); // mid0 = q't + qt'
                       // when t = 0, output q , t = 1, output q'
  and m14(mid1, mid0, rst_n); // when rst_n = 0, output 0
  D_Flip_Flop m15(clk, mid1, q);

endmodule

// reuse the design of DFF from basic question
module D_Flip_Flop(clk, d, q);
input clk;
input d;
output q;

wire f1, f2;
not m5(f1, clk);
 D_Latch m6(f1, d, f2);
 D_Latch m7(clk, f2, q);

endmodule


module D_Latch(e, d, q);
input e;
input d;
output q;

wire w0, w1, w2, w3;

not m0(w0, d);
nand m1(w1, d, e);
nand m2(w2, w0, e);
nand m3(w3, q, w2);
nand m4(q, w3, w1);

endmodule
```

```
module XOR(q, t, mid0);
input q;
input t;
output mid0;
wire q_not, t_not, x0, x1;

// q XOR t = q't + qt'
not m8(q_not, q);
not m9(t_not, t);
and m10(x0, q_not, t);
and m11(x1, q, t_not);
or m12(mid0, x0, x1);

endmodule
```

- testbench

```
`timescale 1ns/1ps

module T_Flip_Flop_t;
// input and output signals
reg clk;
reg t;
reg rst_n;
wire q;

// test instance instantiation
T_Flip_Flop TFF(
    .clk(clk),
    .q(q),
    .t(t),
    .rst_n(rst_n)
);

//generate clk
always #5 clk = ~clk;

// generate t and rst_n signal
initial begin
    clk = 1'b0;
    t = 1'b0;
    rst_n = 1'b0;
#6 rst_n = 1'b1;
#4 t = 1'b1;
#26 t = 1'b0;
#18 rst_n = 1'b0;
#5 rst_n = 1'b1;
#5 t = 1'b1;
#60 $finish;
end

endmodule
```

III. 波形圖



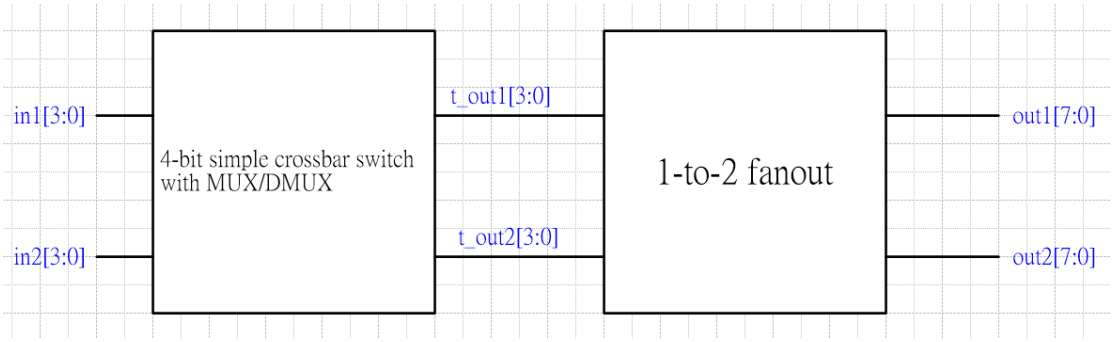
t = 1, rst_n = 1 時，q 在每個 clk posedge complement

t = 0, rst_n = 1 時，q 不產生變化

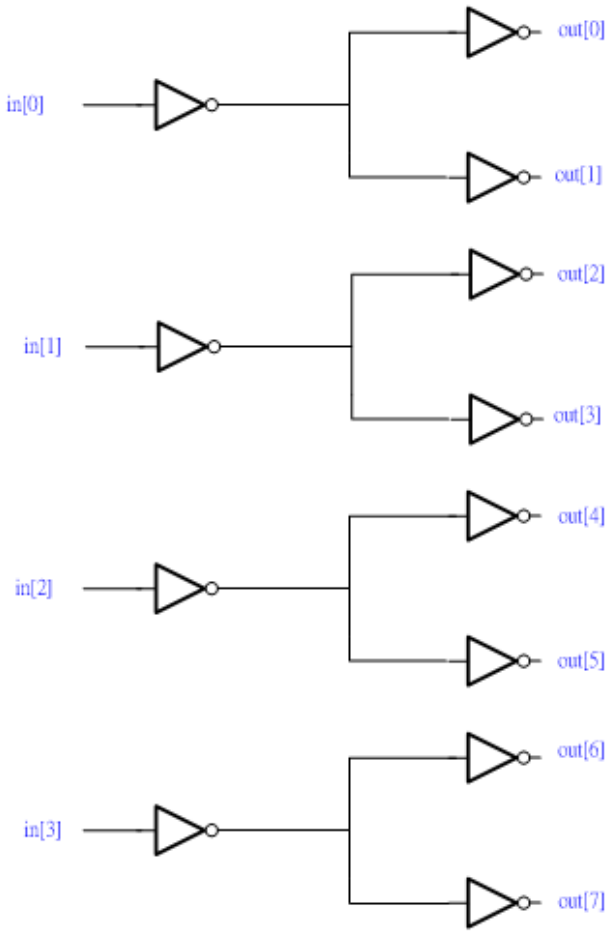
rst_n = 0 且 clk 處於 posedge 狀態時，q reset 變成 0

5. FPGA – 4-bit simple crossbar switch with MUX/DMUX

I. 邏輯設計圖



- 4-bit simple crossbar switch with MUX/DMUX:使用 Question 2 的 module
- 1-to-2 fanout:



II. Verilog 程式碼

```
module Crossbar_2x2_4bit_fpga(in1, in2, control, out1, out2);
input [3:0] in1, in2;
input control;
output [7:0] out1, out2;
wire inv_control;
not inv(inv_control, control);
wire [3:0] wd1_m1, wd1_m2, wd2_m1, wd2_m2, t_out1, t_out2;
// reuses the module written below
DMux_1x2_4bit DMux1(in1, wd1_m1, wd1_m2, inv_control);
DMux_1x2_4bit DMux2(in2, wd2_m1, wd2_m2, control);
Mux_2x1_4bit Mux1(wd1_m1, wd2_m1, t_out1, inv_control);
Mux_2x1_4bit Mux2(wd1_m2, wd2_m2, t_out2, control);
// use fanout to replace 1 signal to 2 signals
Fanout_2 F_1(t_out1, out1);
Fanout_2 F_2(t_out2, out2);
endmodule

module Mux_2x1_4bit(in_1, in_2, out, sel);
input [3:0] in_1, in_2;
input sel;
output [3:0] out;
wire inv_sel;
wire [3:0] a_f, b_f;
not inv(inv_sel, sel);
// only when sel is 0 output out will be the same as input in_1
and a_filter[3:0](a_f, in_1, inv_sel);
// only when sel is 1 output out will be the same as input in_2
and b_filter[3:0](b_f, in_2, sel);
// since sel is either 0 or 1 , the signal of output out depends on input sel which chooses the input in_1 or in_2
or ans[3:0](out, a_f, b_f);
endmodule

module DMux_1x2_4bit(in, out1, out2, sel);
input [3:0] in;
input sel;
output [3:0] out1, out2;
wire inv_sel;
not inv(inv_sel, sel);
// only when the input sel is 0, output out1 will be the same signal as input in while the other output is 0
and out1_res[3:0](out1, in, inv_sel);
// only when the input sel is 1, output out2 will be the same signal as input in while the other output is 0
and out2_res[3:0](out2, in, sel);
endmodule

module Fanout_2(in, out);
//input 4bits
input [3:0]in;
//output 8 bits
output [7:0] out;
wire [3:0]w;
// use 4 not gates to invert signals
not g1[3:0](w, in);
//use 8 not gates to invert signals again , 2 not gates for 2 w[0] so as to w[1] to w[3]
not g2[7:0](out, {w[3], w[3], w[2], w[2], w[1], w[1], w[0], w[0]});
endmodule
```

6. 工作分配

劉奇泓:Question 4 ，FPGA 題的 xdc.檔及連接，實驗報告製作

洪聖祥:Question1~3，FPGA 題的程式碼，實驗報告中說明 1~3 題的波形圖

7. 心得感想

劉奇泓:這是邏設實驗課程中的第一個需要與組員合作的 Lab，我在這一次實驗中學會了自己寫出完善的 testbench，也學會了把 verilog 程式碼轉換成 bitstream、寫入 FPGA 板的過程，希望之後的實驗我能對 verilog 越來越熟練，並更有效率地完成每個實驗!

洪聖祥:這次的 lab 需要用到大量的 verilog 寫各種 module，在寫問題的過程中，我從本來只會粗淺的語法慢慢摸索並釐清各個 module 的邏輯，也第一次自己學會寫 testbench。雖然在寫 testbench 以及燒錄 fpga 時遇到瓶頸，但在跟組員討論以及自己查資料後我逐漸找到解決問題的方法。從這次的 lab 我學到團隊分工、溝通的重要。相信在經過這次 lab 的經驗下次 lab 能更迅速完成實驗。