Lab1_Team32_Report

組別：Team32

組長：蘇勇誠（108062373）

組員：張晏瑄（108062273）

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

Design

題目要求設計 4-bit 的 1-to-4 demultiplexer，我們先設計 1bit 的 1-to-4 demultiplexer，再將 4 個 1bit 的 demultiplexer 整合已達到題目要求。 而 1-to-4 demultiplexer 可由 3 個 1-to-2 demultiplexer 組成。

Detail

1. 先實作一個 1-to-2 demultiplexer
    - Truth Table of 1-to-2 Demultiplexer

in	sel	a	b
0	0	0	0
0	1	0	0
1	0	1	0
1	1	0	1

由 truth table，我們可以得到 $a = \overline{sel} \cdot in, b = sel \cdot in$

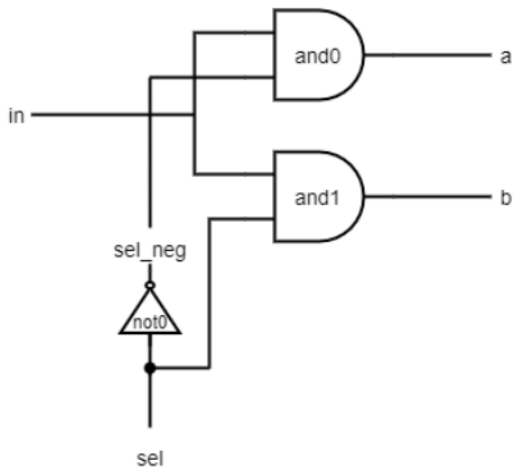
```
wire sel_neg;
```

```
not not0(sel_neg, sel);

and and0(a, sel_neg, in);
and and1(b, sel, in);
```

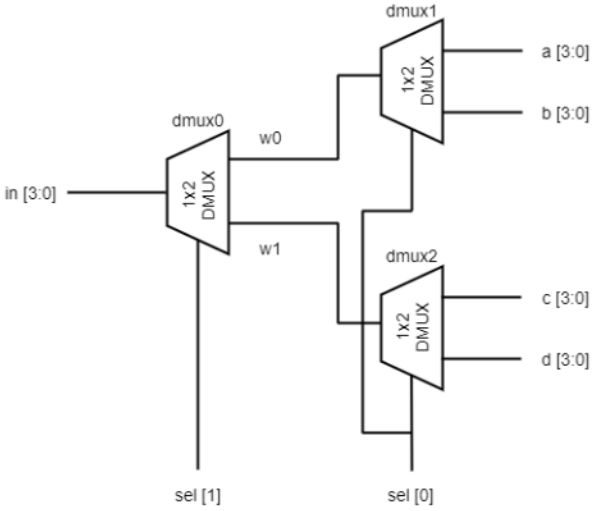
2. 再將3個 1-to-2 demultiplexer 組合，以達到和 1 個 1-to-4 demultiplexer 相通效果

```
wire [3:0] w0, w1;

Dmux_1x2_1bit dmux0[3:0](.in(in), .a(w0), .b(w1), .sel(sel[1])) ;
Dmux_1x2_1bit dmux1[3:0](.in(w0), .a(a), .b(b), .sel(sel[0])) ;
Dmux_1x2_1bit dmux2[3:0](.in(w1), .a(c), .b(d), .sel(sel[0])) ;
```

Gate Level Diagram



1-to-2 DMUX



1-to-4 DMUX

Testbench

因 `sel` 只有 2 bit 所以只需測試 4 筆資料。當 output 被選擇後，該 output 會被設為 in 的值，而其他 output 則是被設為 0。

```
initial begin
    repeat (2 ** 2) begin
        in = in + 4'b1;
        #1 sel = sel + 2'b1;
    end
```

```
    #1 $finish;
end
```

4-bit simple crossbar switch with MUX/DMUX

Design

根據題目要求，設計出 4-bit simple crossbar switch，而 4-bit simple crossbar switch 是由兩個 MUX 和兩個 DMUX 組成。因此，我們先設計出 DMUX 和 MUX 的 module，再利用 DMUX 和 MUX 組合成 4-bit 的 crossbar switch。

Detail

1. Top_module

```
not not0(control_neg, control);

Dmux_1x2_1bit Dmux0[3:0](.in(in1), .a(w1), .b(w2), .sel(control_neg));
Dmux_1x2_1bit Dmux1[3:0](.in(in2), .a(w3), .b(w4), .sel(control));

Mux mux0[3:0](.a(w1), .b(w3), .sel(control_neg), .f(out1));
Mux mux1[3:0](.a(w2), .b(w4), .sel(control), .f(out2));
```

2. MUX (1bit 2-to-1 MUX)

```
not n0(neg_sel, sel);

and and0(w0, a, neg_sel);
and and1(w1, b, sel);
```

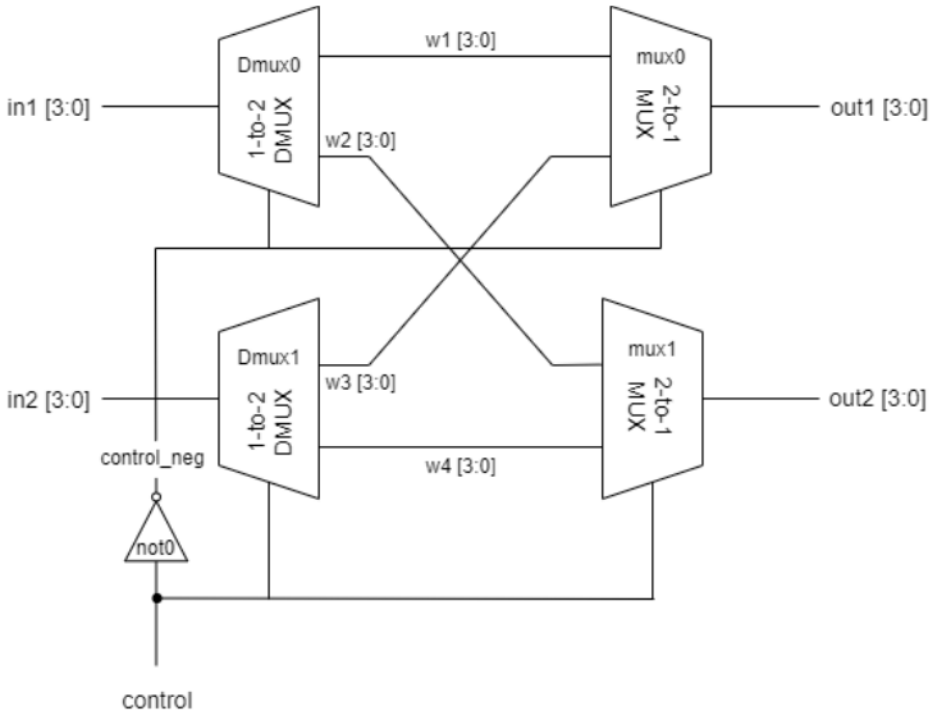
```
or or0(f, w0, w1);
```

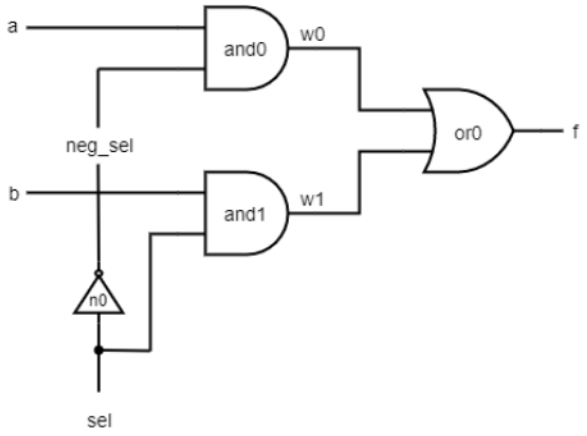
3. DMUX (1bit 1-to-2 DMUX)

```
not not0(sel_neg, sel);

and and0(a, sel_neg, in);
and and1(b, sel, in);
```

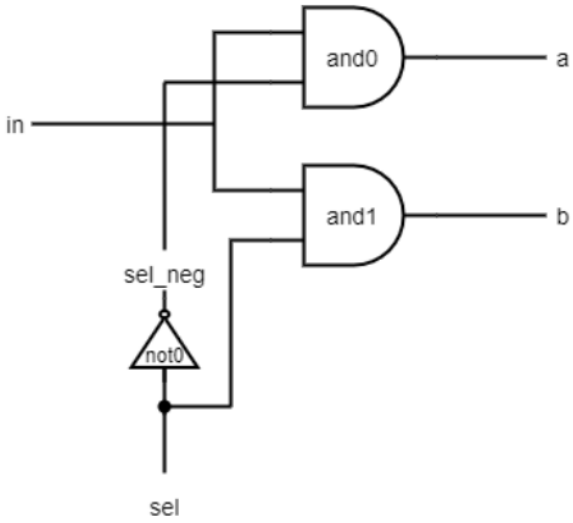
Gate Level Diagram

1. crossbar switch
2. 1-bit MUX
3. 1-bit DMUX





1 bit 2-to-1 MUX



1 bit 1-to-2 DMUX

Testbench

由於control只有1bit，故output只有以下兩種CASE

CASE 1: control = 0 ⇒ out1 = in1, out2 = in2

CASE 2: control = 1 ⇒ out1 = in2, out2 = in1

我們只需要brute force把以上2種CASE列出來，就能驗證crossbar switch的正確性。

```
initial begin
    repeat (2) begin
        #1 control = control + 1'b1;
        in1 = in1 + 4'b1;
        in2 = in2 + 4'b1;
    end
    #1 $finish;
end
```

4-bit 4x4 crossbar with simple crossbar switch

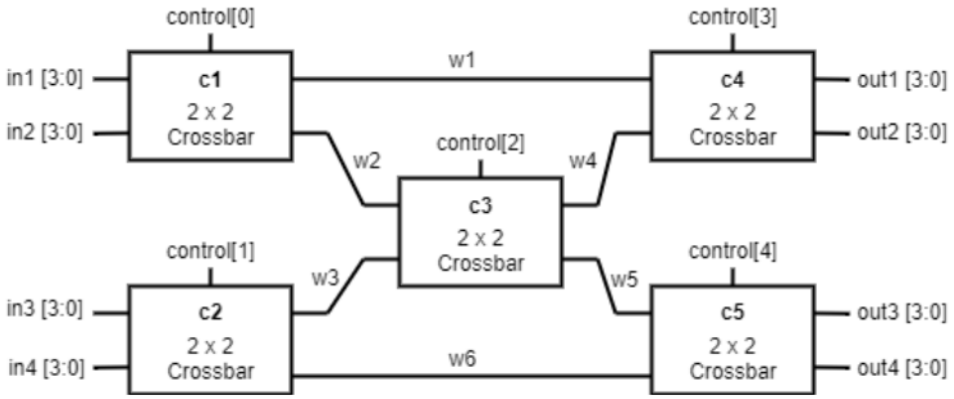
Design

根據題目要求，設計 4-bit 4x4 crossbar with simple crossbar switch，而 simple crossbar switch 已在上一題做出來了，故只需要 reuse 該 module 即可。

Detail

```
Crossbar_2x2_4bit c1(.in1(in1), .in2(in2), .control(control[0]), .out1(w1), .out2(w2));
Crossbar_2x2_4bit c2(.in1(in3), .in2(in4), .control(control[1]), .out1(w3), .out2(w6));
Crossbar_2x2_4bit c3(.in1(w2), .in2(w3), .control(control[2]), .out1(w4), .out2(w5));
Crossbar_2x2_4bit c4(.in1(w1), .in2(w4), .control(control[3]), .out1(out1), .out2(out2));
Crossbar_2x2_4bit c5(.in1(w5), .in2(w6), .control(control[4]), .out1(out3), .out2(out4));
```

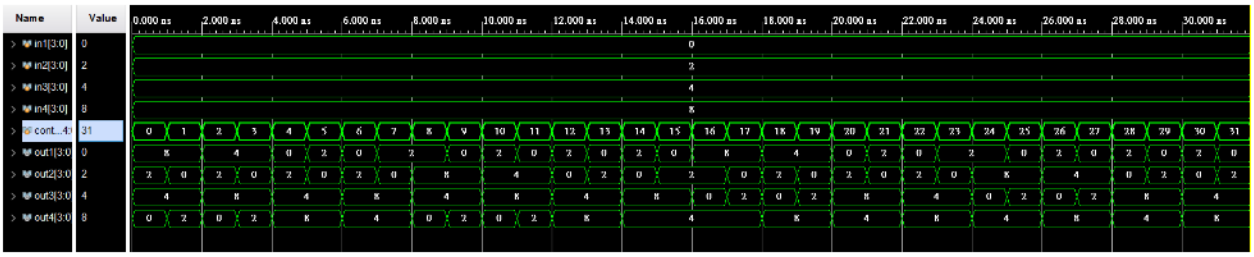
Gate Level Diagram



Testbench

由於 control 有 5 個 bits，所以當 in1~in4 值固定時，output 有 32 種可能，故直接 brute force 列出32 種排列。

```
initial begin
    repeat (31) begin
        #1 control = control + 5'b00001;
    end
    #1 $finish;
end
```



Configurations Which Are Not Routable

假設 in1 = a, in2 = b, in3 = c, in4 = d，故以下 4 種 output 排列永遠不會出現：

out1=d, out2=c, out3=b, out4=a

out1=c, out2=d, out3=b, out4=a

out1=c, out2=d, out3=a, out4=b

out1=d, out2=c, out3=a, out4=b

可以觀察到，out1 及 out2 不會同時有 d 和 c，而 out3 及 out4 也不會同時出現 a 和 b。

1-bit toggle flip flop (TFF)

Design

根據題目要求，設計出 toggle flip flop，而 T flip flop 可 reuse D flip flop 再加入一些 Logic Gate 組成。D flip flop 由兩個 D Latch 和一些 Logic Gate組成。所以我們先設計 D Latch，再利用 D Latch 組合出 D flip flop，再利用 D flip flop 組合成 T flip flop。

Detail

1. T Flip Flop
    - Truth table of T Flip Flop

clk	T	Q(n+1)
0	x	Qn
1	0	Qn
1	1	$\overline{Qn}$

```
not not0(q_neg, q);
not not1(t_neg, t);
```

```
and and0(w1, q, t_neg);
and and1(w2, t, q_neg);
or or0(w3, w1, w2);

and and2(w4, w3, rst_n);

D_Flip_Flop DFF(.clk(clk), .d(w4), .q(q));
```

2. D Flip Flop

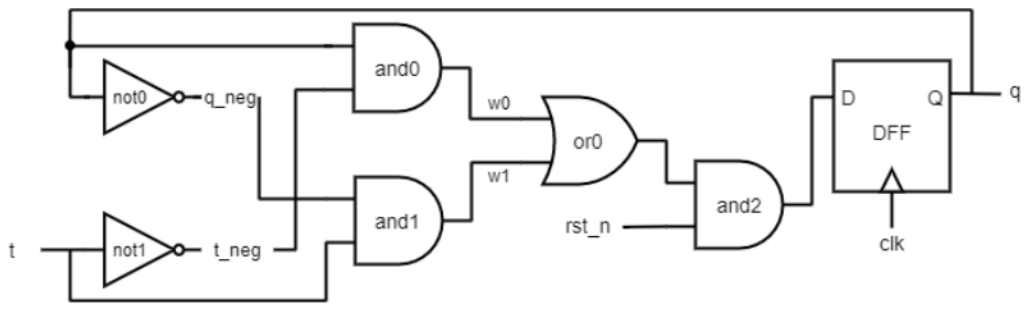
```
not n0(neg_clk, clk);

D_Latch D0(neg_clk, d, w);
D_Latch D1(clk, w, q);
```

3. D-Latch

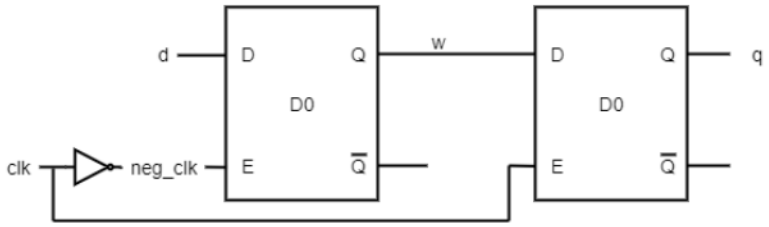
```
not n0(neg_d, d);

nand a0(w1, e, d);
nand a1(w2, e, neg_d);

nand o0(w3, w1, w4);
nand o1(w4, w2, w3);

not n1(w5, w3);
not n2(q, w5);
```

Gate Level Diagram

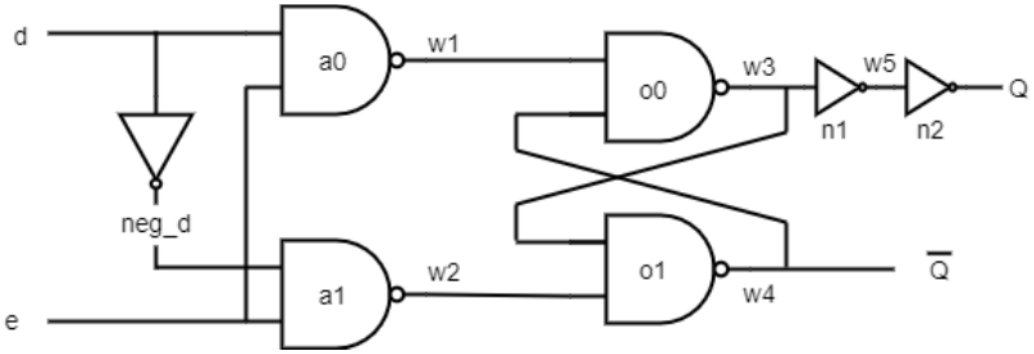
1. T Flip Flop



2. D Flip Flop



3. D latch



Testbench

窮舉所有T Flip Flop可能發生的所有CASE，分別是以下4種CASE

CASE 1: t = 0, q = 0, rst_n = 1 ⇒ q' = 1 (在 3000ns 由 positive edge trigger 使 q 由0變1)

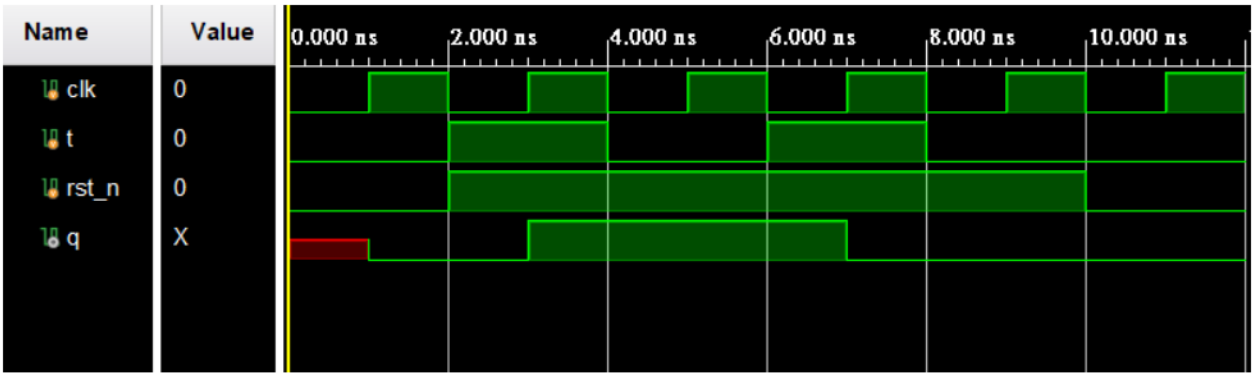
CASE 2: t = 0, q = 1, rst_n = 1 ⇒ q' = 1 (在 5000ns 由 positive edge trigger q 維持為1)

CASE 3: t = 1, q = 1, rst_n = 1 ⇒ q' = 0 (在 7000ns 由 positive edge trigger 使 q 由1變0)

CASE 4: t = 0, q = 0, rst_n = 1 ⇒ q' = 0 (在 9000ns 由 positive edge trigger q 維持為0)

CASE 5: 當 rst_n = 0 時，相當於input d = 0至D Flip Flop，故不論q和t的值為何，q' 均為 0 (在 10000~12000ns)

```
always#(1) clk = ~clk;

initial begin
    @(negedge clk) t = 1'b1; rst_n = 1'b1; //CASE 1 2000~400ns
    @(negedge clk) t = 1'b0; //CASE 2 4000~6000ns
    @(negedge clk) t = 1'b1; //CASE 3 6000~8000ns
    @(negedge clk) t = 1'b0; //CASE 4 8000~10000ns
    @(negedge clk) rst_n = 1'b0; //CASE 5 10000~12000NS
    @(negedge clk) $finish;
end
```



Learning

1. Gate Level Design

   此次 Lab 老師要求要全用 gate level 實作，雖然上學期有修邏輯設計，但要求並沒那麼嚴格，故還需花點時間熟悉。

2. Testbench

   這是我們第一次寫 testbench，我們先看簡報上的概念後，再理解 basic question 的 testbench 後，就大致可以試著寫出來一些 testbench。

3. Vivado elborate 錯誤

   在寫 verilog 的時候有跑出一些 elborate 錯誤訊息，比較常犯的錯就是匿名的元件呼叫以及引數名字不符，下次在寫的時候必須細心一點。

   - 匿名的元件呼叫

   ```
   // it should be
   // Dmux_1x4_4bit d1 (...);
   Dmux_1x4_4bit (
       .in (in),
       .sel (sel),
       .a (a),
       .b (b),
       .c (c),
       .d (d)
   );
   ```

   ```
   [USF-XSim-62] 'elaborate' step failed with error(s).
   ```

4. Gate Level Diagram

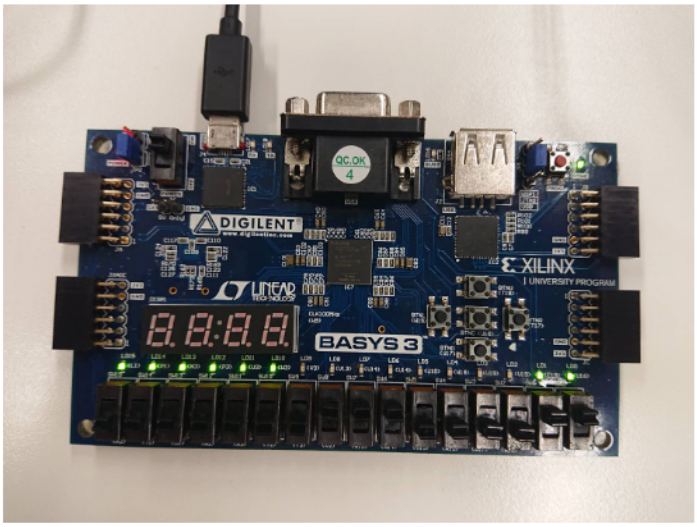
使用 draw.io 作圖，在作圖過程中常會不小心忘記標示邏輯閘、wire 名稱，以及畫線的時候要怎麼畫才不會看起來很亂等，這些小細節在往後的 Lab 都會特別注意。

5. module 分割的重要性

   如 4x4 crossbar switch 可由 5 個 2x2 crossbar switch 實作，而 1-to-4 的 DMUX 也可由 3 個 1-to-2 的 DMUX 實作。透過 reuse module，不僅可以省下很多時間，也可以較好 trace code，以及更為簡潔。

FPGA 的連接

先撰寫 XDC 檔，將 IO map 到 pin，再 generate bitstream，接著將FPGA接上電腦，再將 generate bitstream program 到 FPGA 上，即可完成。而遇到較大的問題則為組員的電腦都無法偵測到 FPGA，無法完成燒錄的動作，而我們採取的解決方式為使用資電館的電腦燒錄。



分工

雙方都各自先寫題目，最後 report 再以共筆形式一起寫。

- 蘇勇誠（108062373）

  DMUX: Detail, Design

  Crossbar switch 2x2 and 4x4, TFF: Detail, Design, Testbench

- 張晏瑄（108062273）

  Gate Level Diagram

  Crossbar switch 4x4: Design, DMUX: Testbench