EECS 2070 02 Digital Design Labs 2019

Lab 1

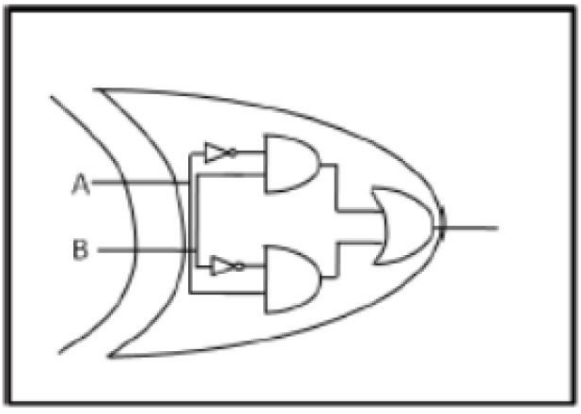
學號：107062107　姓名：王領崧

1.前言

利用實作 1-bit ALU 來熟悉三種 verilog 的描述方法，並且利用 4 個 1-bit 的 module 組出一個 4-bits ALU。Bonus 的部分則是多了一個比大小的函數，一樣需要利用 4 個 1-bit 組出一個 4-bits ALU。

2.實作過程

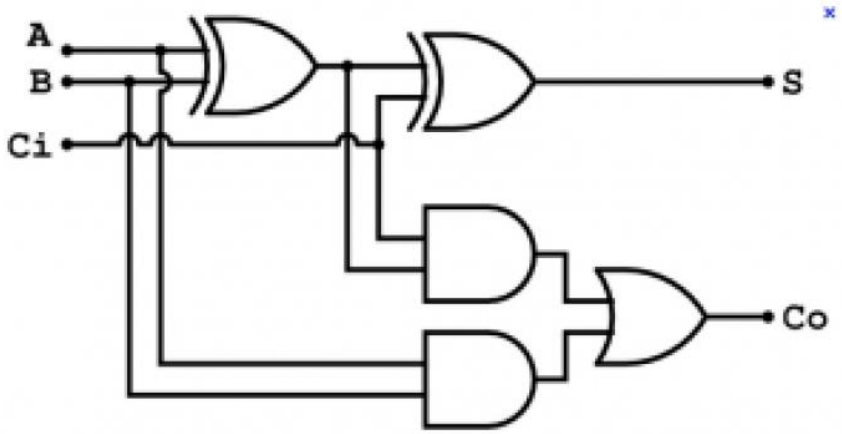
myxor




```
module myxor (out, a, b);
    input a, b;
    output out;
    wire not_a, not_b;
    wire tmp1, tmp2;

    // build not gate
    not not0(not_a, a);
    not not1(not_b, b);

    // use not gate with and gate
    and and0(tmp1, not_a, b);
    and and1(tmp2, not_b, a);

    // combine tmp1 and tmp2 with or gate
    or or0(out, tmp1, tmp2);
endmodule
```

myxor 的部分，我使用了提供的 Design 1，利用 Gate level 的寫法，依序拼出 not and or 三種邏輯閘，進而做出與 XOR 相同的邏輯功能。

lab1_1




```
module lab1_1 (a, b, c, aluctr, d, e);
    input a, b, c;
    input [1:0] aluctr;
    output d, e;

    wire d_a_i, d_tmp0, d_b_i, d_c_i, d_d_i, d_tmp1;
    wire e_a_i, e_tmp0, etmp1, e_tmp2;

    /* output e part */
    // full adder Cout part
    and and2(e_tmp0, a, b);
    and and3(e_tmp1, a, c);
    and and4(e_tmp2, b, c);
    or or1(e_a_i, e_tmp0, e_tmp1, e_tmp2);

    mux4_to_1 mux0(.q_o(e), .a_i(e_a_i), .b_i(0), .c_i(0), .d_i(0), .sel_i(aluctr));
```

```
    /* output d part */
    // full adder S part
    myxor xor0(.out(d_tmp0), .a(a), .b(b));
    myxor xor1(.out(d_a_i), .a(d_tmp0), .b(c));

    // a and b
    and and0(d_b_i, a, b);

    // a nor b
    or nor0(d_tmp1, a, b);
    not nor1(d_c_i, d_tmp1);

    // a xor b
    myxor xor2(.out(d_d_i), .a(a), .b(b));

    mux4_to_1 mux1(.q_o(d), .a_i(d_a_i), .b_i(d_b_i), .c_i(d_c_i), .d_i(d_d_i), .sel_i(aluctr));
endmodule
```

lab1_1，我的作法是把所有可能為 d 的答案與 e 的答案先用 wire 儲存起來，接著使用助教所提供的 mux 模組，依據 aluctr 來選擇輸出相對應的答案。函數裡最複雜的 abc 三數相加，d 扮演 Sum 的部份，就是將三數 xor 起來，而 e 扮演 Carry out 的部份，使用 majority function，先將三數兩兩個別 and 起來，再把三個結

果 or 起來。其他的函數就是簡單的 and nor xor，用本身內建的邏輯閘即可完成。

lab1_2

```
module lab1_2 (a, b, c, aluctr, d, e);
    input a, b, c;
    input [1:0] aluctr;
    output d, e;
    wire tmp_d, tmp_e;
    wire and_ab, nor_ab, xor_ab;

    assign {tmp_e, tmp_d} = a + b + c;
    assign and_ab = a & b;
    assign xor_ab = a ^ b;
    assign nor_ab = ~(a | b);

    /* e part */
   assign e = (aluctr == 2'b00) ? tmp_e : 1'b0;

    /* d part */
    assign d = (aluctr == 2'b00) ? tmp_d : (aluctr == 2'b01) ? and_ab : (aluctr == 2'b10) ? nor_ab : xo

endmodule;
```

lab1_2，將 gate level 轉化為 dataflow 的寫法，與 lab1_1 方法一樣，我先把所有的結果算出來並儲存好，而 mux 的部份使用問號冒號，來模擬 if-else 的判斷式，選擇出正確的值指派給最後的 output d、e。

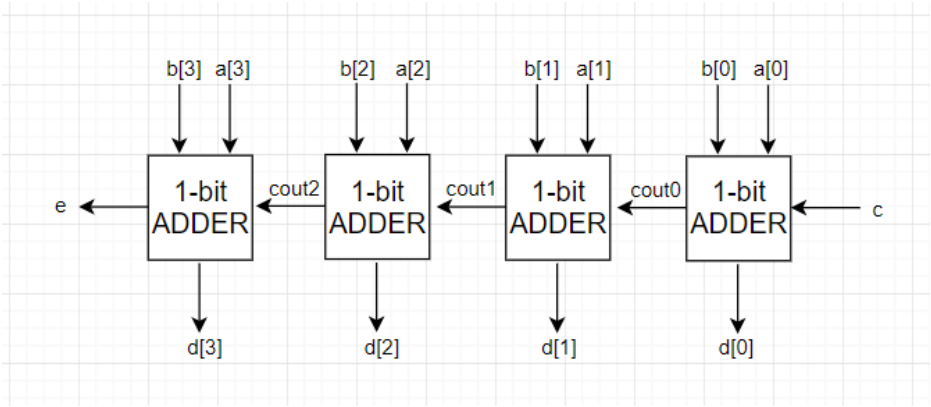
lab1_3

```
module lab1_3 (a, b, c, aluctr, d, e);
    input a, b, c;
    input [1:0] aluctr;
    output d, e;
    reg d, e;

    always @(*) begin
        case (aluctr)
            2'b00 : {e, d} = a + b + c;
            2'b01 : begin
                d = a & b;
                e = 1'b0;
            end
            2'b10 : begin
                d = ~(a | b);
                e = 1'b0;
            end
            2'b11 : begin
                d = a ^ b;
                e = 1'b0;
            end
        endcase
    end
endmodule
```

lab1_3 則是用 Behavioral 的改寫，個人認為是三種中最簡單的，利用在 always block 中加入 case 判斷 aluctr 的不同，給予 d e 相對應的值，比較不一樣的是，我在每一個 case 中，才執行相對應的邏輯式或函數，而不是在 always block 外面事先計算好。

lab1_4

```
module lab1_4 (a, b, c, aluctr, d, e);
    input [3:0] a,b;
    input [1:0] aluctr;
    input c;
    output [3:0] d;
    output e;
    wire cout0, cout1, cout2;

    lab1_3 alu0(a[0], b[0], c, aluctr, d[0], cout0);
    lab1_3 alu1(a[1], b[1], cout0, aluctr, d[1], cout1);
    lab1_3 alu2(a[2], b[2], cout1, aluctr, d[2], cout2);
    lab1_3 alu3(a[3], b[3], cout2, aluctr, d[3], e);
endmodule
```



lab1_4 為 4-bit 的 ALU，必須用 4 個 1-bit ALU 去組裝而成，abc 三數相加的部份，與 Full Adder 的概念大致相同，從 LSB 開始計算，將 input c 的部份理解為 carry out 的數值，利用 4 個 1-bit ALU module，分別與 a b 和前一個 carry out 進行相加，最後 MSB 的 carry out 就是 e，而每個 module 中相加出來的 Sum，則是存進 d 相同的 bit 中，而其他函數都是 d 為 a b 的邏輯式，因為各個 module 中 abd 所對應 bit 的位置本來就相同，所以相加與邏輯閘的函數剛好可以相符合，感謝助教提供如此優質的題目。

lab1_b1

```
module lab1_b1 (a, b, c, aluctr, d, e);
    input a, b, c;
    input [1:0] aluctr;
    output d, e;
    reg d, e;

    always @(*) begin
        case (aluctr)
            2'b00 : {e, d} = a + b + c;
            2'b01 : begin
                d = a & b;
                e = 1'b0;
            end
            2'b10 : begin
                d = 1'b0;
                e = (a < b || (a == b && c == 0)) ? 1'b0 : 1'b1;
            end
            2'b11 : begin
                d = a ^ b;
                e = 1'b0;
            end
        endcase
    end
endmodule
```

lab1_b1，是將 aluctr==10 的部份，多增加一個比大小的函數，為了不要這麼多行，直接利用問號冒號的 if-else 寫法，照著提供的 pseudo code 硬爆出來。

lab1_b2

```
module lab1_b2 (a, b, c, aluctr, d, e);
    input [3:0] a,b;
    input [1:0] aluctr;
    input c;
    output [3:0] d;
    output e;

    wire cout0, cout1, cout2;

    lab1_b1 alu0(a[0], b[0], c, aluctr, d[0], cout0);
    lab1_b1 alu1(a[1], b[1], cout0, aluctr, d[1], cout1);
    lab1_b1 alu2(a[2], b[2], cout1, aluctr, d[2], cout2);
    lab1_b1 alu3(a[3], b[3], cout2, aluctr, d[3], e);
endmodule
```

lab1_b2，將 b1 所做成的 1-bit ALU 組裝成一個 4-bit

ALU，仔細研究後發現，lab1_4 中連接所使用的參數，完全符合比大小函數，因為 input c 的部份，不只可以當成加法中 carry out 的數值，也可以當作紀錄前面 bit 比大小的結果，因此 code 與 lab1_4 一模一樣。

3.學習到的東西與困難

經由 lab1，有慢慢找回上學期打 verilog 的感覺，也複習了各種描述方式的打法，不過還是覺得 behavioral 使用起來最舒適也最自然，寫作業的過程中，沒有遇到什麼太大的困難，一方面是題目簡單，沒有什麼複雜的功能或是 sequential logic 的出現，另一方面是教授都有提供相關的電路圖，只要照著打，基本上會非常順利，希望往後的 lab 也能順利且如期地完成。

4.想對老師和助教說的話

經過 lab1 的 demo，不知道是因為很多小檔案需要 sim 的關係，整體 demo 的進度好像蠻慢的，我同學就等到 5:10 幾分才完成 demo，因此想請問是因為這次檔案多的緣故，還是未來都差不多是這個速度，感謝~

另外，提供兩則笑話給老師

1. 為什麼連假期間不能出去工作?
   .....因為會變成廉價勞工
2. 問：亞洲舞王是誰？答：羅志祥
   問：那亞洲舞后呢？
   .....雷陣雨