

Case 設計

```

always@(*)begin
    case(FS)
        4'b0000:    Y = A<<1'b1;           //LSF
        4'b0001:    Y = A<<<1'b1;          //ASF
        4'b0010:    Y = A>>1'b1;          //LSR
        4'b0011:    Y = A>>>1'b1;         //ASR
        4'b0111:    Y = ya2;               //|A - B|
        4'b1000:    Y = Ax*Bx;             //A*B
        4'b1001:    Y = 32'd0;              //zero
        4'b1010:    Y = A;                 //A
        4'b1011:    Y = B;                 //B
        4'b1100:    Y = A & B;              //A&B
        4'b1101:    Y = A | B;              //A|B
        4'b1110:    Y = A^B;               //A^B
        4'b1111:    Y = ~A;                //~A
        4'b0100, 4'b0101,4'b0110:   Y = ya; //and and sub
    endcase

```

1 LSF LSR

採用最基本的平移運算方法，以兩個箭頭代表向左（右）移動位數。

2 ASF ASR

我把傳進來的 A 跟 B 轉為 wire signed，讓他會把第一位當成他的正負號，這樣就可以直接使用三個箭頭來進行位移了。

如果沒有這樣做了話，在左移時，他會在左邊直接補一個 0 而不是補 1。

```

wire signed [n-1:0] A,B;
wire signed [15:0] Ax = A[15:0];
wire signed [15:0] Bx = B[15:0];
wire [n-1:0] ya,ya2;
wire [3:0] FS;
wire Cin,ca,ca2;
reg [n-1:0] Y;
reg N,C,V,Z;

```

而判斷 carry out 和 overflow 的方法：

```

    end
//ASF ASR
4'b0001,4'b0011:begin
    N = Y[n-1];
    C = A[n-1] & (~FS[1]);
    V = A[n-1]^A[n-2];
    Z = ~(|Y);
end
//add

```

C 的判斷方法：為了一次處理 A S F 跟 A S R 所以我把 A[n-1]去跟~FS[1]去 & 起來，目的是

A S R ($\sim FS[1] = 0$)： 這樣了話，不管什麼東西去 & 都會變成 0。

A S F ($\sim FS[1] = 1$)： 這樣去 & 了話就會是他自己本身。

V 的判斷方法： $A[n-1]$ (原來的第一位) $\wedge A[n-2]$ (後來的第一位) 這樣就可以知道是否有 O V E R F L O W 了。

3 ADD and SUB

加法的部分，採用上課所教的 bit slice 的方法，參考了老師上課打個 code 並進行修改以符合格式，bit slice 的重點在於每一為自己進行加減，並使用 carry in 和 carry out 的處理，將進位帶到下一位來運算。

```

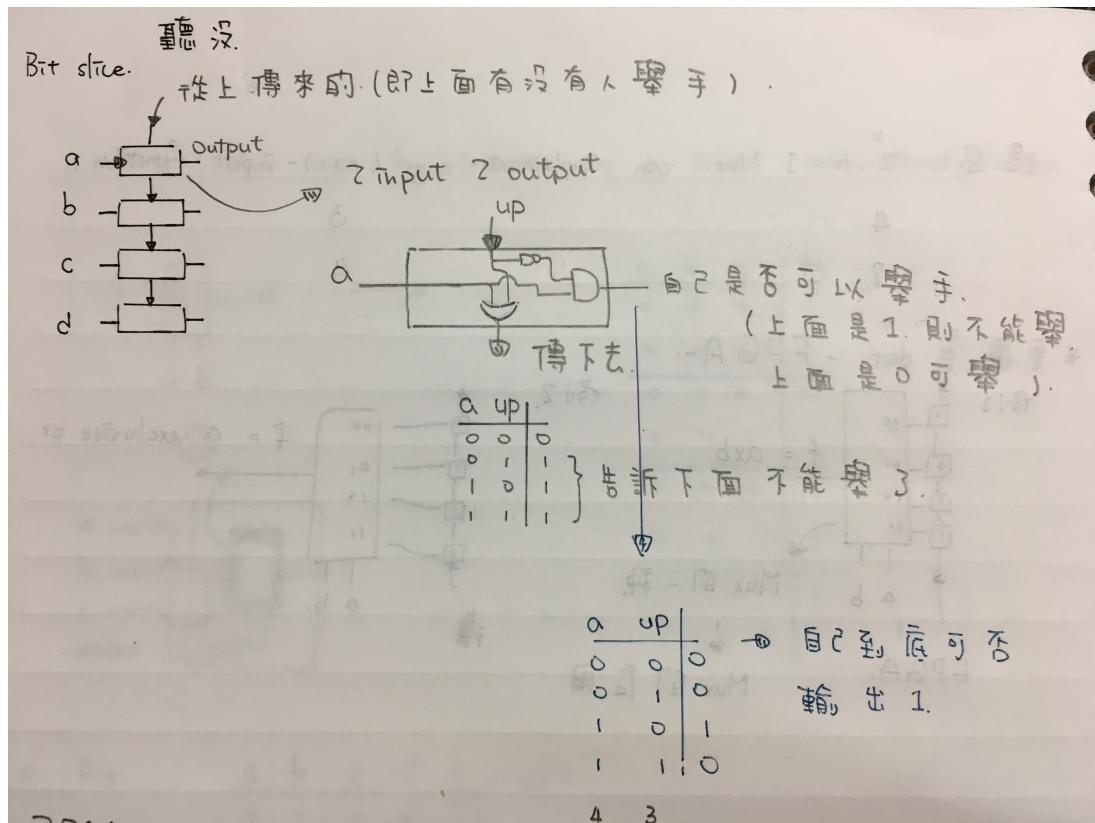
module AdderSubtracter(A,B,Cin,mode,Cout,Y);
parameter n = 32;
input A,B,Cin,mode;
output Y,Cout;

wire mode,Cin,Cout;
wire [n-1:0] A,B;
wire [n-1:0] b2,Y;
wire [n:0] c;

assign c[0] = Cin | mode;
assign b2 = B ^ {n{mode}};
assign Y = A ^ b2 ^ c[n-1:0];
assign c[n:1] = A & b2 |
                A & c[n-1:0] |
                b2 & c[n-1:0];
assign Cout = c[n];
endmodule

```

Bit slice 的基本概念：



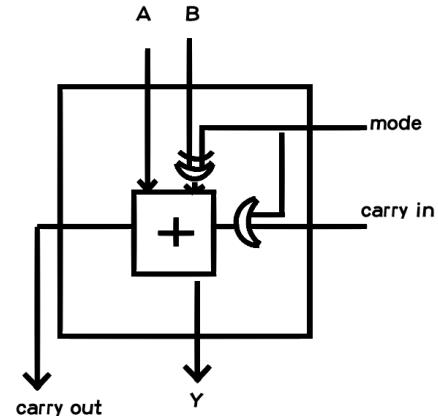
Y 的運算方法，經 K-MAP (下表) 的結論可得與 $X \text{ OR }$ 結果相同 而 CARRY OUT 的結論跟 MAJORITY 相同。

故可用之前所學的內容寫出這個 AdderSubtracter。

A	B	Carry in	Y	Carry out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	10	1	1	1

而處理 sub 的方法是先將 B 換成他的二補數：

即先將他與 mode(= 1 代表 減) X O R 起來，然後在 mode (= 1) 跟 carry in 相加 (O R)。即得下圖



而判斷 carry out 和 overflow 的方法：

```

end
//add
4'b0100,4'b0101:begin
    N = Y[n-1];
    C = ca;
    V = (A[n-1] ^ Y[n-1]) & (B[n-1] ^ Y[n-1]);
    Z = ~(|Y);
end
//sub
4'b0110:begin
    N = Y[n-1];
    C = ca;
    V = (A[n-1] ^ B[n-1]) & (A[n-1] ^ Y[n-1]);
    Z = ~(|Y);
end
    
```

C 的判斷方法：即 AdderSubtracter 的 Cout。

V (加法) 的判斷方法：當 A 、 B 同號，然後 Y 異號時，即 O V E R F L O W 。

所以：

$A[n-1] \wedge Y[n-1]$	$B[n-1] \wedge Y[n-1]$	$(A[n-1] \wedge Y[n-1]) \wedge (B[n-1] \wedge Y[n-1])$
0	0	0
0	1	0
1	0	0
1 (代表 A 跟 Y 異號)	1 (代表 B 跟 Y 異號)	1 (代表 overflow)

\vee (減法) 的判斷方法：當 A 、 B 異號，然後 Y 又與 A 異號時，即 OVERFLOW 。

所以：

A[n-1] ^ B[n-1]	A[n-1]^Y[n-1]	(A[n-1] ^ B[n-1]) & (A[n-1] ^ Y[n-1])
0	0	0
0	1	0
1	0	0
1 (代表 A 跟 B 異號)	1 (代表 A 跟 Y 異號)	1 (代表 overflow)

4 | A - B |

絕對值的 A 減 B ，我使用了一個比較特別的方法，採用兩次 AdderSubtracter 讓他變成一定 output 正數。

```
module ALU(A,B,Cin,FS,Y,N,C,V,Z);
parameter n = 32;
input A,B,Cin,FS;
output Y,N,C,V,Z;

wire signed [n-1:0] A,B;
wire signed [15:0] Ax = A[15:0];
wire signed [15:0] Bx = B[15:0];
wire [n-1:0] ya,ya2;
wire [3:0] FS;
wire Cin,ca,ca2;
reg [n-1:0] Y;
reg N,C,V,Z;

AdderSubtracter a1(A, B, Cin, FS[1], ca, ya);
AdderSubtracter a2( ya^{n{ya[n-1]}}, 32'b0 | ya[n-1] ,1'b0, 1'b0 ,ca2 ,ya2 );
```

圖中可得

第一次 addersubtracter 後， ya 在去與他的第一位 X OR ，

ya[n-1] = 0 了話： 其值不變

ya[n-1] = 1 了話（表示為負數）：ya 整個數就會變成他的 1'complement 。

然後再進行第二次的加法：

ya[n-1] = 0 了話： 結果及不變

ya[n-1] = 1 了話： 他就會變成他的 2'complement 了。

這樣就能得出來的結果必為正的。

5 A * B

A * B 的部份事先將 A 跟 B 轉換成 wire signed 的形式，這樣在進行乘法運算時，他就會自動判斷他是正數還是負數了。

```
wire signed [n-1:0] A,B;  
wire signed [15:0] Ax = A[15:0];  
wire signed [15:0] Bx = B[15:0];
```

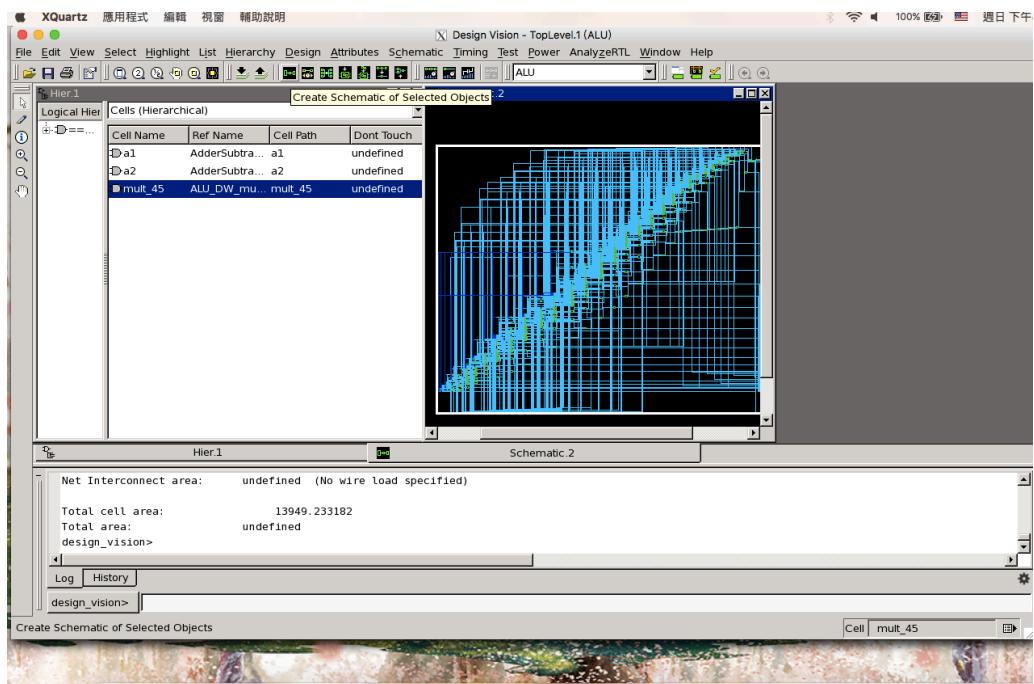
4'b0111:	Y = ya2;	// A - B
4'b1000:	Y = Ax*Bx;	//A*B
4'b1001:	Y = 32'd0;	//zero

V 的判斷方法：

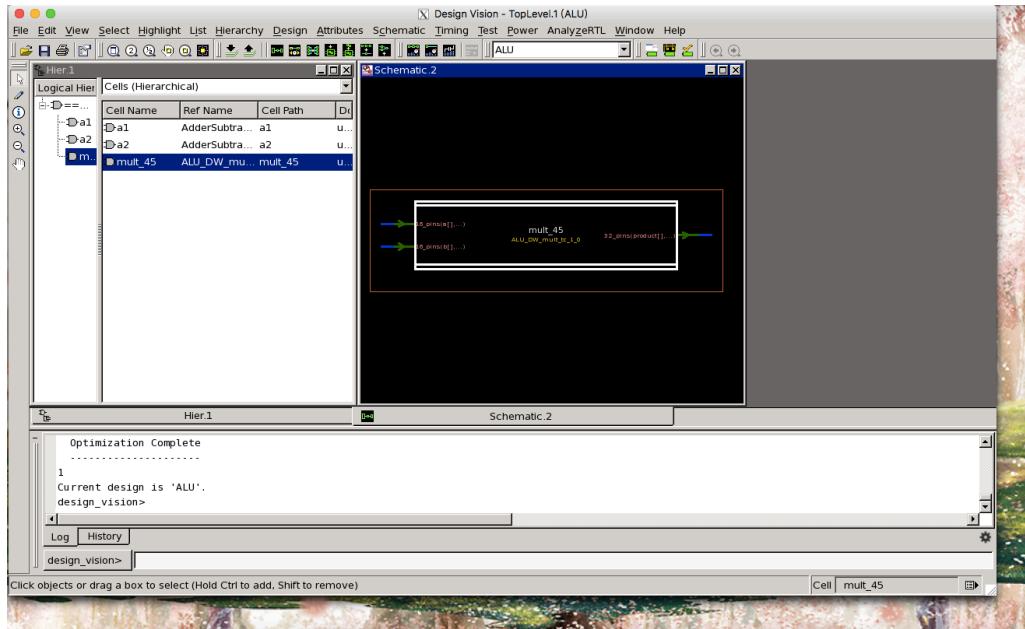
由於是 15 位數乘以 15 位數，所以他的結果必不會超過 32 位數，所以絕對不會 OVERFLOW。所以直接 output 0 就好。

```
...  
//A * B  
4'b1000:begin  
    N = Y[n-1];  
    C = 1'b0;  
    V = 1'b0;  
    Z = ~(|Y);  
end
```

6 design vision



可見有 a1 跟 a2 兩個 AdderSubtracter



7 area

一開始，我的 area 是 14000 多，經過幾次修改後，與同學討論的結果，我將一些 N、C、V、Z 相同的提出來寫在另一個 case 裡，這樣成功地減少了 1000 左右的 area，但總體而言仍然過大，個人猜測可能是因為 A*B 的關係導致而成的，還有一些可能是邏輯閘過多的關係，目前還找不到可以精簡的地方，仍然在研究中。

```
slow (File: /theda21_2/CBDK_IC_Contest/cur/SynopsysDC/db/slow.db)

Number of ports:                      367
Number of nets:                        1625
Number of cells:                      1171
Number of combinational cells:        1167
Number of sequential cells:           1
Number of macros/black boxes:         0
Number of buf/inv:                    128
Number of references:                 34

Combinational area:                  13949.233182
Buf/Inv area:                       624.643190
Noncombinational area:               0.000000
Macro/Black Box area:                0.000000
Net Interconnect area:              undefined (No wire load specified)

Total cell area:                    13949.233182
Total area:                         undefined
```