實驗三

Adder, subtractor, multiplexer

學生: 謝旻錡

學號: 313512078

日期: 2025/03/22

1. 實驗目的

藉由實驗學會使用 FPGA 完成加法、減法及乘法的功能。Part1 及 part 5 也順便複習 lab1 的 full adder。

2. 實驗程式碼

Part6 demo 題，目標為 $S = (A \times B) + (C \times D)$ ，整體架構為兩個 8bit 乘法器的輸出相加得到答案，並將輸入及輸出顯示在 7 段顯示器上。

Table 1. input/ output table

Input		Output	
KEY_0	Reset	$LEDR_9$	Carry out
KEY_1	clk	HEX_{3-2}	A or C
KEY_2 (select1)	1: select A / B; 0: select C/ D	HEX_{1-0}	B or D
KEY_3 (change_dis)	1: display AB/CD 0: display S	HEX_{3-0}	sum
SW_9	Write enable		
SW_8 (select2)	1: display AB 0:display CD		
SW_{7-0}	Value of input		

- 先架構演算法主體 $S = (A \times B) + (C \times D)$ ，用 LPM 宣告兩個乘法器，竟將其輸出相加得到答案。

```
//define multi and add
mult_sub mult_1(
    .dataa(regA),
    .datab(regB),
    .result(multi1)
);
mult_sub mult_2(
    .dataa(regC),
    .datab(regD),
    .result(multi2)
);

assign result = multi1 + multi2;
```

Figure 1

- 規定輸入與 reset、enable、select1 及 select2 的控制關係。

如果 reset 為 TRUE，則輸入歸零。

當 enable 為 TRUE 時，才能寫入輸入值，且依據 select1 及 select2 之值，選擇將輸入值，賦予 A/B/C/D。

```
always @(posedge clk or negedge reset) begin
    if (!reset) begin
        // 當 reset 為 0, 立即清零 (非同步)
        regA <= 0;
        regB <= 0;
        regC <= 0;
        regD <= 0;
    end else begin
        // 若 enable == 1, 則更新 regA, regB, regC, regD
        if (enable) begin
            case ({select1, select2})
                2'b00: regB <= SW[7:0];
                2'b01: regD <= SW[7:0];
                2'b10: regA <= SW[7:0];
                2'b11: regC <= SW[7:0];
            endcase
        end
    end
end
```

Figure 2

- 7 段顯示器模組

2 進制轉 16 進制的顯示器功能。

```
/*============== submodule =============*/
module HEX (
    input [3:0] hex,    // 4-bit 十六進制輸入
    output reg [6:0] seg  // 7 段顯示器輸出
);
    always @(*) begin
        case (hex)
            4'h0: seg = 7'b1000000;  // 0
            4'h1: seg = 7'b1111001;  // 1
            4'h2: seg = 7'b0100100;  // 2
            4'h3: seg = 7'b0110000;  // 3
            4'h4: seg = 7'b0011001;  // 4
            4'h5: seg = 7'b0010010;  // 5
            4'h6: seg = 7'b0000010;  // 6
            4'h7: seg = 7'b1111000;  // 7
            4'h8: seg = 7'b0000000;  // 8
            4'h9: seg = 7'b0010000;  // 9
            4'hA: seg = 7'b0001000;  // A
            4'hB: seg = 7'b0000011;  // B
            4'hC: seg = 7'b1000110;  // C
            4'hD: seg = 7'b0100001;  // D
            4'hE: seg = 7'b0000110;  // E
            4'hF: seg = 7'b0001110;  // F
            default: seg = 7'b1000000;  // 預設關閉
        endcase
    end
endmodule
```

Figure 3

- 設計 7 段顯示器切換顯式的功能(A/ B/ C/ D/ S)

如果 Cchange_dis 為 FAULSE，則顯示 S 算術結果。

如果 Cchange_dis 為 TRUE，則顯示輸入值。且當 select2 為 FAULSE，顯示 AB 值；當 select2 為 TRUE，顯示 CD 值。

```
//display
//HEX3-0
reg [3:0] hex3,hex2, hex1,hex0;

HEX U3(hex3,HEX3);
HEX U2(hex2,HEX2);
HEX U1(hex1,HEX1);
HEX U0(hex0,HEX0);
always @(*)begin//display switch
    reg_result = result;
    if(change_dis == 0)begin //S
        hex3 = reg_result[15:12];
        hex2 = reg_result[11:8];
        hex1 = reg_result[7:4];
        hex0 = reg_result[3:0];
    end
    else
        if(select2 == 0)begin//AB
            hex3 = regA[7:4];
            hex2 = regA[3:0];
            hex1 = regB[7:4];
            hex0 = regB[3:0];
        end
        else begin//CD
            hex3 = regC[7:4];
            hex2 = regC[3:0];
            hex1 = regD[7:4];
            hex0 = regD[3:0];
        end
end
```

Figure 4

- Overflow LED

```
//overflow
assign LEDR[9] = reg_result[16];
```

3. 實驗結果照片 (optional)

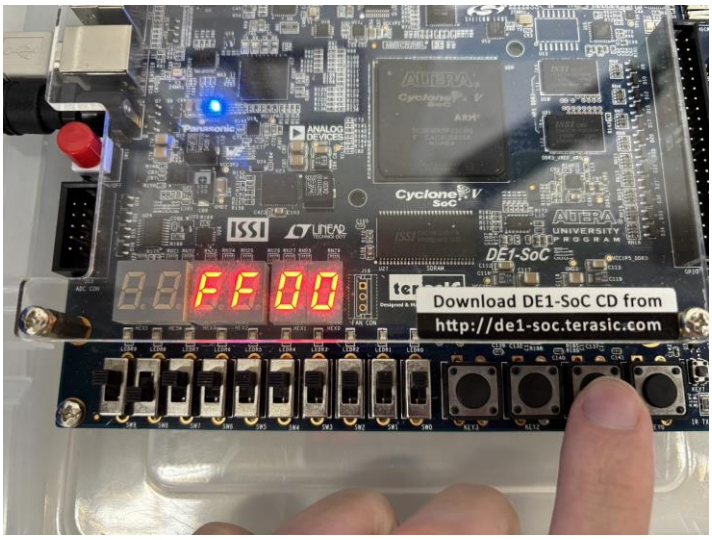


Figure 5. 設定 A 值

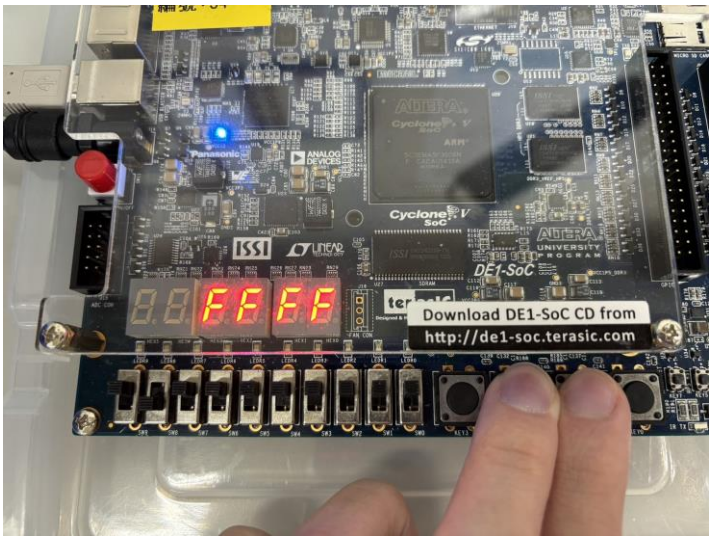


Figure 6. 設定 B 值

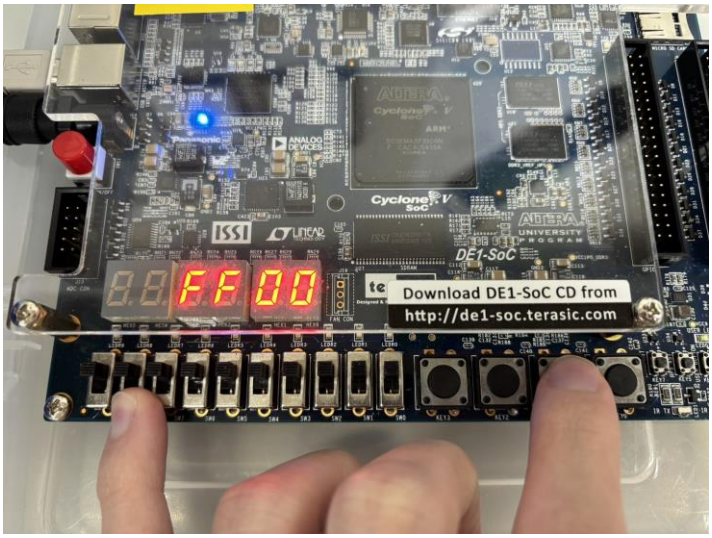


Figure 7. 設定 C 值

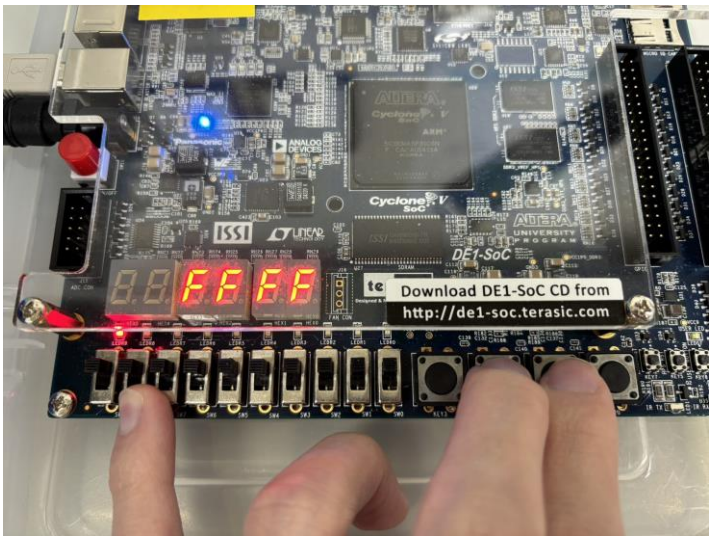


Figure 8. 設定 D 值

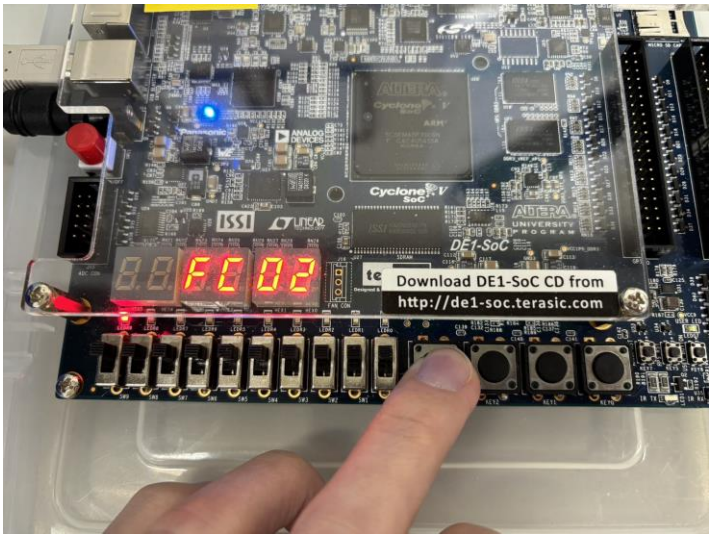


Figure 9. 顯示 S 值(overflow)

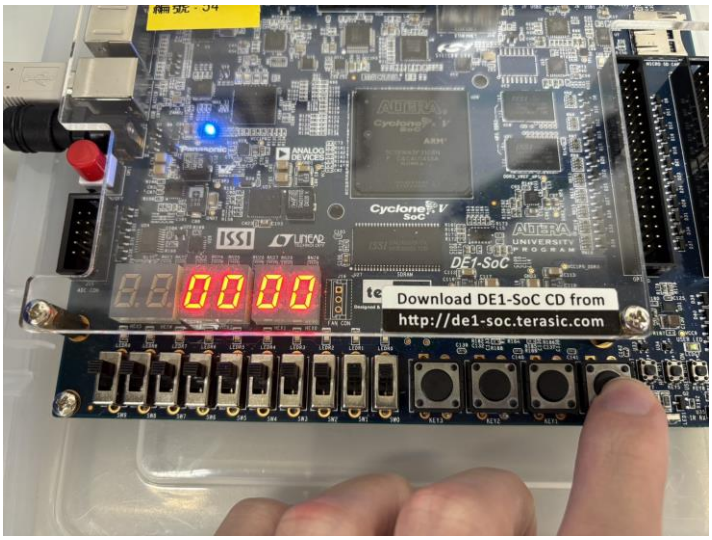
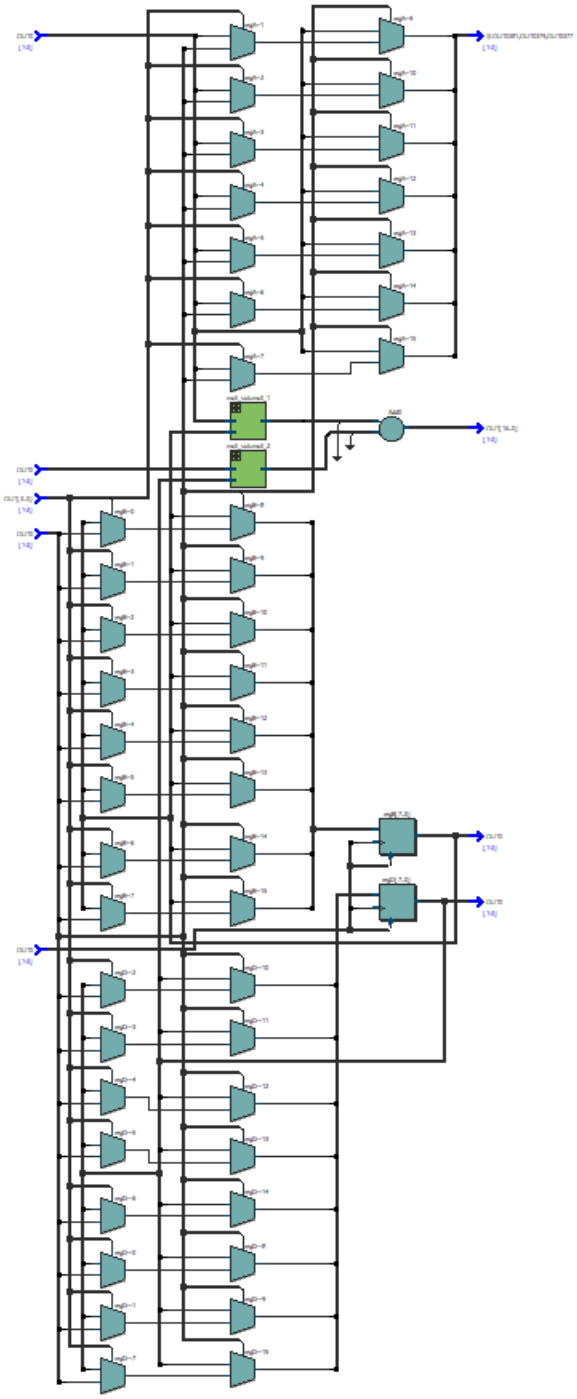
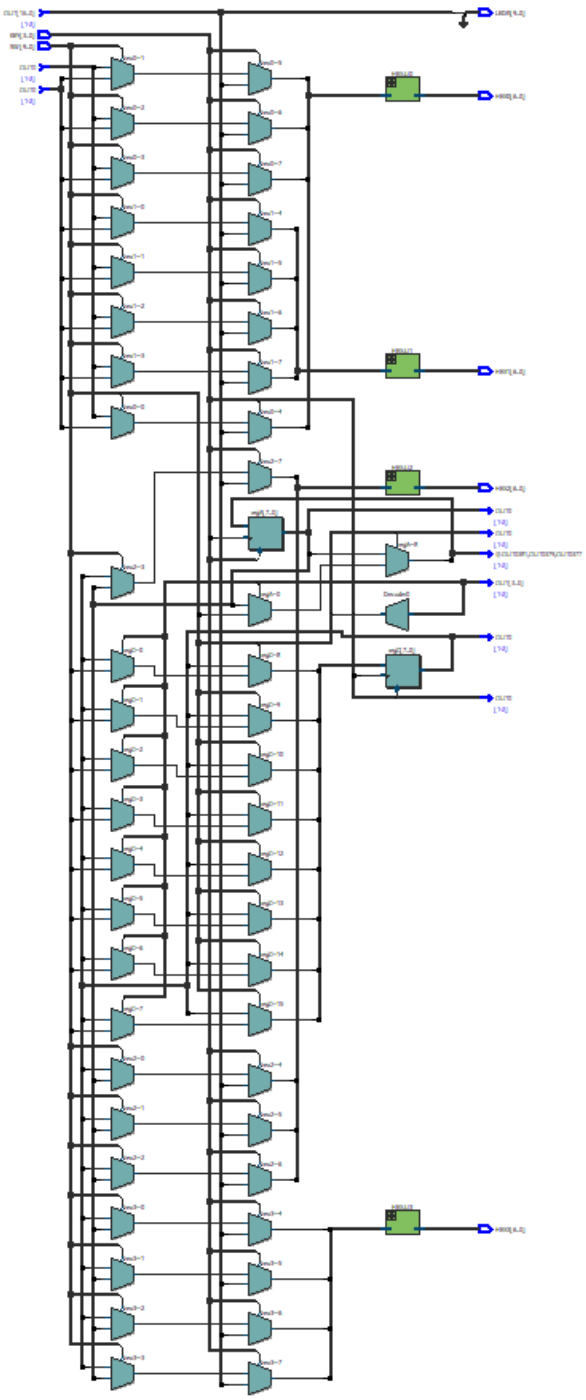


Figure 10. reset

4. RTL 布局(optional)





5. 問題與討論

1. 須注意 assign 或 設定一個變數，其不能同時出現在多個 always 中。
2. 模組如果較為複雜，我覺得使用 LPM 可以節省時間及程式複雜度。
3. Overflow 有兩種方法達成，1 為使用 LPM 的加法器有包含這個 port，2 直接將輸出的 regist 或 wire 設大一點，在 overflow 的位數直接做判斷。此次 demo 我是使用第二種方法。