106010006 黃詩瑜 電機 21

Lab 2: FPGA Emulation

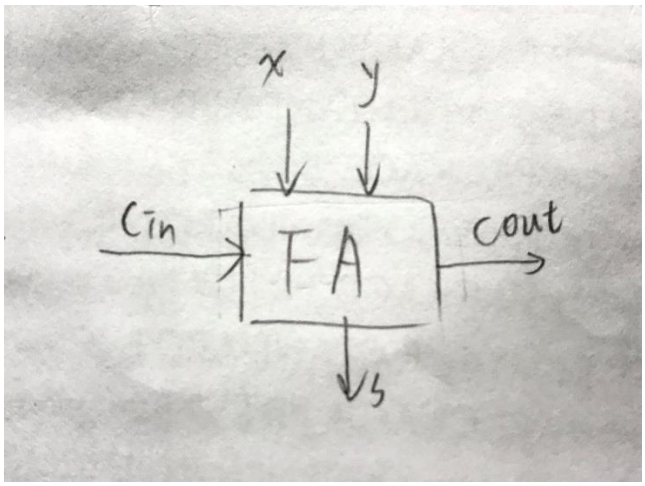
1. **Emulate exp1 in lab1 (a full adder *s+cout=x+y+cin*) with the following parameters.**

I/O	*x*	*y*	*cin*	*s*	*cout*
LOC	V17	V16	W16	U16	E19

Design Specification:

Input: x, y, cin

Output: s, cout



Implementation:

將之前實驗做的 full adder 實現在 FPGA 板上，使用 lab1 第一題的 module code，再將 input 與 output 設定到題目指定的 port 中，就能完成。FPGA 板上的 V17,V16,W16 可以控制 input，往上撥就代表 1，往下則是 0，cin 代表 carry，x,y 與 cin 相加後的結果會顯示於 cout 和 s 對應的 LED 燈上，亮燈則代表 1，不亮就是 0，結果可由 truth table 應證。

討論:

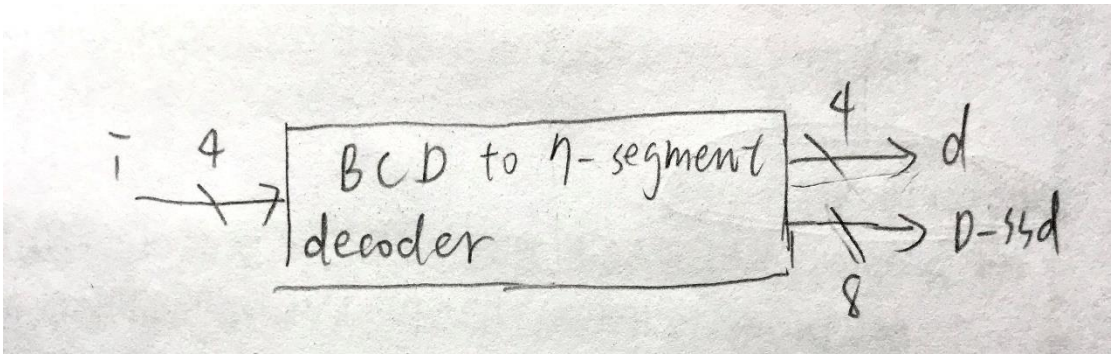
這題很簡單，只要照著講義的步驟就能完成。

2. **Derive a BCD (*i*[3:0]) to 7-segment display decoder (*D_ssd*[7:0]), and also use four LEDs (*d*[3:0]) to monitor the 4-bit BCD number. (Other values of *I* outside the range will show F).**

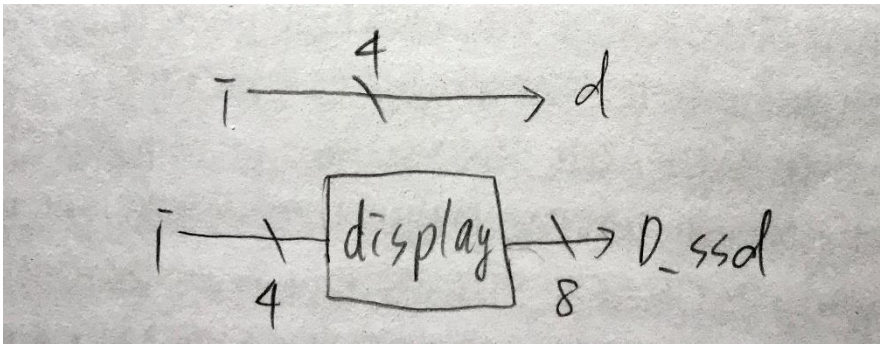
Design Specification:

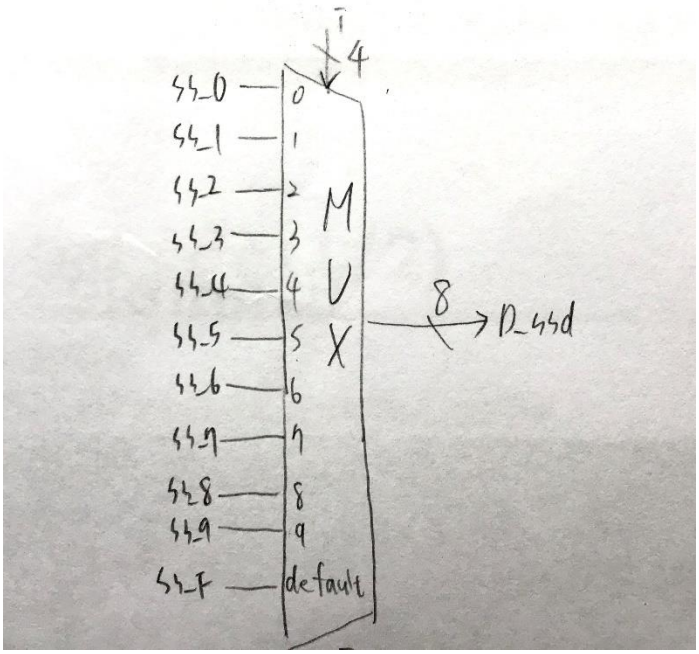
Input: i

Output: d,D_ssd



Implementation:





這題我是利用講義的做法將 module code 拆成兩個.v 檔，一個是將 4-bit input i 利用 MUX 與 decoder 的觀念轉為 7-seg，4-bit input 可以表示 0~15，使用 MUX 選到對應的 0~9，剩下的則是 F，將選到的數用 D_ssd 表示，再接到 7-seg。要將一個數字用 7-seg 顯示需要 8 個 bit 控制，所以先將每個數字對應的 8 個 bit 用”SS_數字”代表，code 會比較清楚。另一個是將 input 直接 assign 給 output d 讓他控制 4 個 LED 燈顯示，再將兩個檔案結合即可。用 switch 往上撥為 1 控制 4-bit input，上排的 4 個 LED 燈會顯示對應的 input，(亮為 1)，再由 7-seg 顯示 0~9 和 F。

Module code1:

```
module display(D_ssd, i);
output [7:0] D_ssd;
input [3:0] i;
reg [7:0] D_ssd;
```

```
/*後面的 8 個 Binary code 依序控制 7-seg 燈的 ABCDEFGP，0 為亮，1 為不亮，
將其定義成 SS_0~SS_F，表示它所顯示的數字*/
`define SS_0 8'b00000011
`define SS_1 8'b10011111
`define SS_2 8'b00100101
`define SS_3 8'b00001101
`define SS_4 8'b10011001
`define SS_5 8'b01001001
`define SS_6 8'b01000001
`define SS_7 8'b00011111
`define SS_8 8'b00000001
`define SS_9 8'b00001001
`define SS_F 8'b01110001

/*由 Input 輸入的 4 個 bits 轉為對應的 7-seg，除了 0~9，其他都是 F*/
always @*
case (i)
4'd0: D_ssd = `SS_0;
4'd1: D_ssd = `SS_1;
4'd2: D_ssd = `SS_2;
4'd3: D_ssd = `SS_3;
4'd4: D_ssd = `SS_4;
4'd5: D_ssd = `SS_5;
4'd6: D_ssd = `SS_6;
4'd7: D_ssd = `SS_7;
4'd8: D_ssd = `SS_8;
4'd9: D_ssd = `SS_9;
default: D_ssd = `SS_F;
endcase
endmodule
```

Module code2:

```
/*將上面的 display 檔結合，將 input i assign 給 output d 控制 4 個 led 燈顯示 4-
bit BCD*/
module ssd(D_ssd, i, d);
output [3:0] d;
output [7:0] D_ssd;
```

```
input [3:0] i;

display U0(.D_ssd(D_ssd),.i(i));

assign d[0] = i[0];
assign d[1] = i[1];
assign d[2] = i[2];
assign d[3] = i[3];

endmodule
```

Constraint code:

```
/*4 個  anode control signals*/
set_property PACKAGE_PIN V19 [get_ports {d[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {d[3]}]
set_property PACKAGE_PIN U19 [get_ports {d[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {d[2]}]
set_property PACKAGE_PIN E19 [get_ports {d[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {d[1]}]
set_property PACKAGE_PIN U16 [get_ports {d[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {d[0]}]
/*8-bit common segment controls*/
set_property PACKAGE_PIN W7 [get_ports {D_ssd[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {D_ssd[7]}]
set_property PACKAGE_PIN W6 [get_ports {D_ssd[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {D_ssd[6]}]
set_property PACKAGE_PIN U8 [get_ports {D_ssd[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {D_ssd[5]}]
set_property PACKAGE_PIN V8 [get_ports {D_ssd[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {D_ssd[4]}]
set_property PACKAGE_PIN U5 [get_ports {D_ssd[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {D_ssd[3]}]
set_property PACKAGE_PIN V5 [get_ports {D_ssd[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {D_ssd[2]}]
set_property PACKAGE_PIN U7 [get_ports {D_ssd[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {D_ssd[1]}]
set_property PACKAGE_PIN V7 [get_ports {D_ssd[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {D_ssd[0]}]
```

```
/*4-bit binary input*/
set_property PACKAGE_PIN W17 [get_ports {i[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {i[3]}]
set_property PACKAGE_PIN W16 [get_ports {i[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {i[2]}]
set_property PACKAGE_PIN V16 [get_ports {i[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {i[1]}]
set_property PACKAGE_PIN V17 [get_ports {i[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {i[0]}]
```

討論:

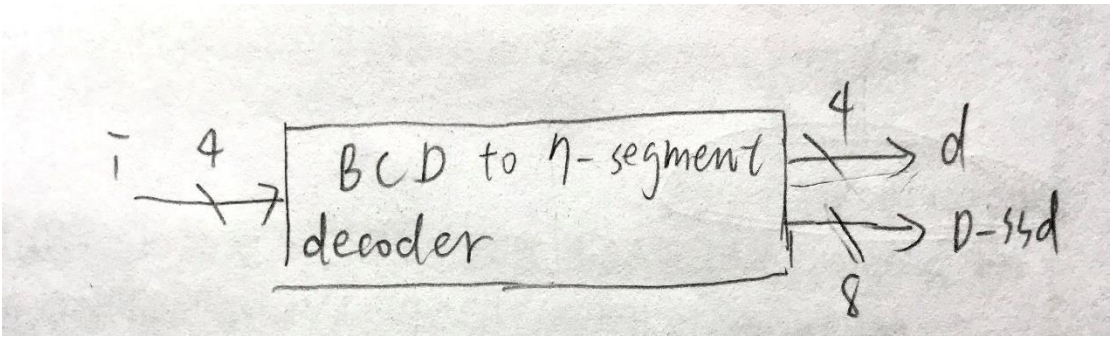
當之後 code 更加複雜後，將檔案區分成小塊，會比較好管理與 debug，是很好的方法。

3. **Derive a binary (*i*[3:0], 0-9, a, b, c, d, e, f) to 7-segment display decoder (*D*[7:0]), and also use four LEDs (*d*[3:0]) to monitor the 4-bit binary number.**

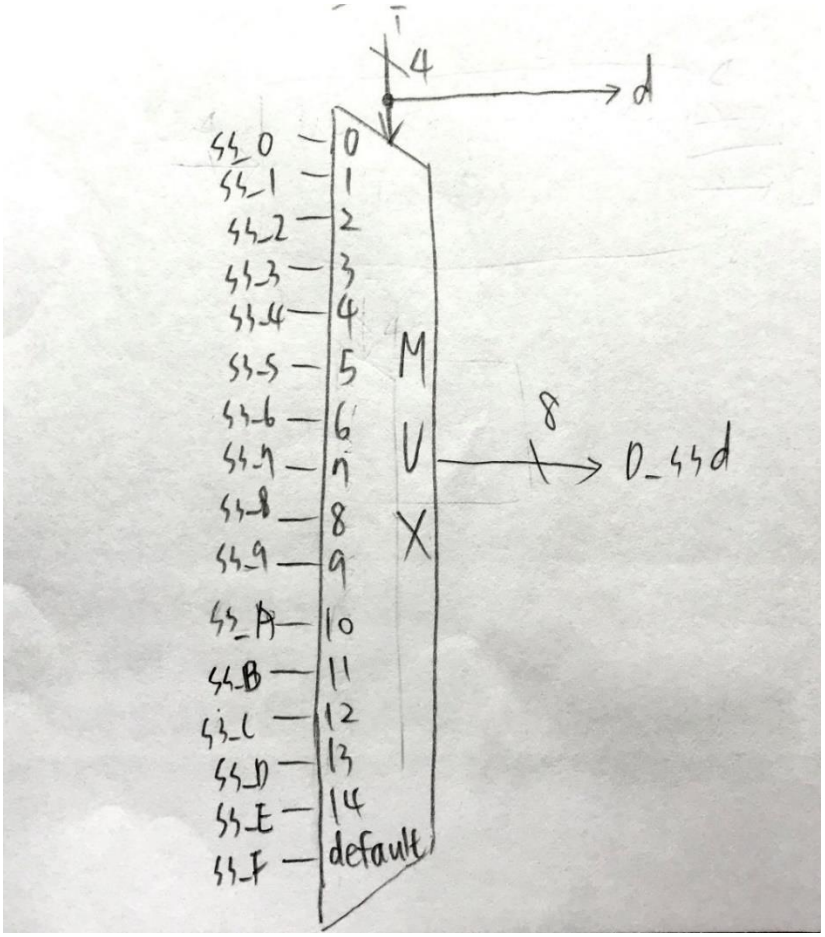
Design Specification:

Input: i

Output: d,D_ssd



Implementation:



這題我只用了一個 source 檔，只要修改上一題的 code 即可，將 MUX 的選項改成 10~14 會對應到 SS_A~E，最後一個是 F，其他不變。

Module code:

```
module lab2_2(
input [3:0]i,
output [3:0]d,
output reg [7:0]D_ssd
    );
/*與上一題一樣，定義 7-seg，只是增加了 A~E*/
    `define SS_0 8'b00000011
    `define SS_1 8'b10011111
    `define SS_2 8'b00100101
    `define SS_3 8'b00001101
```

```
`define SS_4 8'b10011001
`define SS_5 8'b01001001
`define SS_6 8'b01000001
`define SS_7 8'b00011111
`define SS_8 8'b00000001
`define SS_9 8'b00001001
`define SS_A 8'b00010001
`define SS_B 8'b00000001
`define SS_C 8'b01100011
`define SS_D 8'b00000011
`define SS_E 8'b01100001
`define SS_F 8'b01110001

/*多工器增加了 10~14 的選項定義為 A~E*/
always @*
  case (i)
  4'd0: D_ssd = `SS_0;
  4'd1: D_ssd = `SS_1;
  4'd2: D_ssd = `SS_2;
  4'd3: D_ssd = `SS_3;
  4'd4: D_ssd = `SS_4;
  4'd5: D_ssd = `SS_5;
  4'd6: D_ssd = `SS_6;
  4'd7: D_ssd = `SS_7;
  4'd8: D_ssd = `SS_8;
  4'd9: D_ssd = `SS_9;
  4'd10: D_ssd = `SS_A;
  4'd11: D_ssd = `SS_B;
  4'd12: D_ssd = `SS_C;
  4'd13: D_ssd = `SS_D;
  4'd14: D_ssd = `SS_E;
  default: D_ssd = `SS_F;
  endcase
/*將 input i 設定給 output d 控制 4 個 led 燈來顯示 input*/
assign d[0] = i[0];
assign d[1] = i[1];
assign d[2] = i[2];
assign d[3] = i[3];
```

endmodule

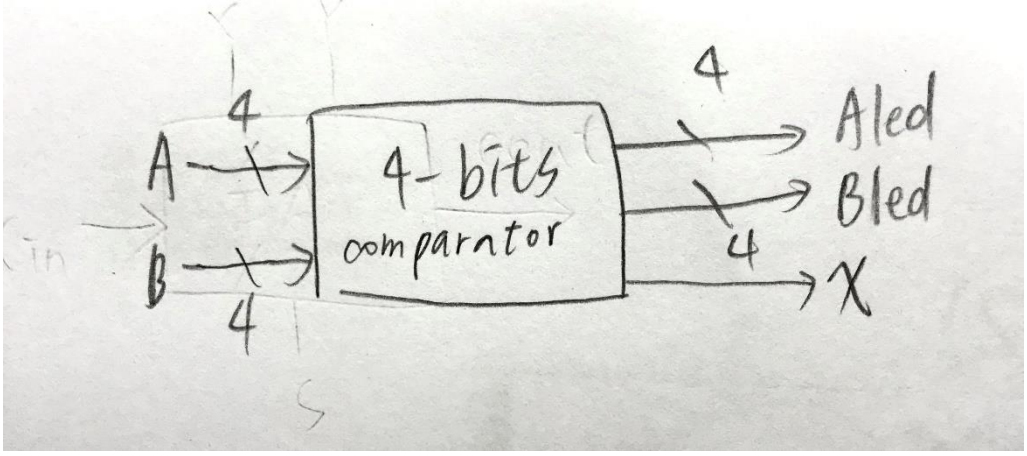
討論:

老師上課有提到 case 最後一個選項要使用 default，幫同學 debug 的時候發現這是一個蠻需要注意的地方。

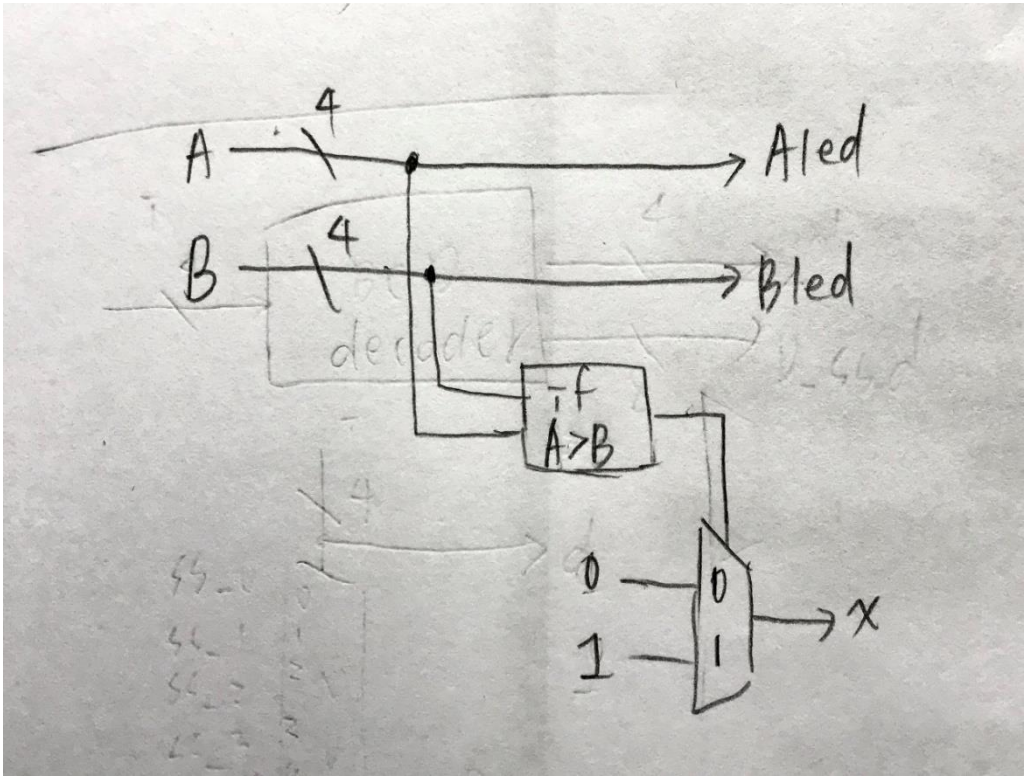
4. **(Bonus) Design a combinational circuit that compares two 4-bit unsigned numbers A and B to see whether A is greater than B. The circuit has one output X such that X = 0 if A ≤ B and X = 1 if A > B. (let A[3:0], B[3:0] be controlled by 8 DIP switches, the binary numbers are displayed on 8 LEDs. The result X is on another LED.)**

Design Specification:

Input: A,B
Output: Aled,Bled,X



Implementation:



這題是要比較 A 與 B 的大小，我將 A,B 用 if 比較再接到 MUX 裡，若 A>B 是 true，MUX 會選到 1 再給 x，若 A>B 是 false，則會把 0 給 x，再將 x 接到隨意的 LED 燈，亮就代表 A>B。A,B 接到 switch 就能控制 input，再把 A,B assign 給 output Aled,Bled，將他們各別接到對應的 switch 上排，就能顯示 input 給的值。

Module code:

```
module lab2_2(
input [3:0]A,B,
output [3:0]Aled,Bled,
output reg X
);
/*將 4-bit input A,B 分別設定給 Aled,Bled，使 input 顯示再 led 燈上*/
      assign Aled[0]=A[0];
      assign Aled[1]=A[1];
      assign Aled[2]=A[2];
```

```
    assign Aled[3]=A[3];

    assign Bled[0]=B[0];
    assign Bled[1]=B[1];
    assign Bled[2]=B[2];
    assign Bled[3]=B[3];

/*比較 A 與 B 的大小，若 A > B 將 X 設為 1，若 A ≤ B 則將 X 設為 0*/
always@*
if(A>B)
X=1;
else
X=0;

endmodule
```

討論:

這題需要的觀念不多，相對於其他題比較容易，debug 的時候發現還是會忘記將 always 的數定義成 reg，這是我之後要更加注意的地方。

結論:

這次實驗很需要細心與耐心，在 vivado 的操作上多了很多步驟，如果 port 設定到錯的位置或是 I/O Std 沒有設定到 LVCMOS33 都會導致實驗失敗，我覺得比較困難的地方是每個 led 燈或是 switch 開關是不同的 high or low active control，這是我比較容易搞混的地方，但是只要多檢查就能完成實驗。