(1) full adder:

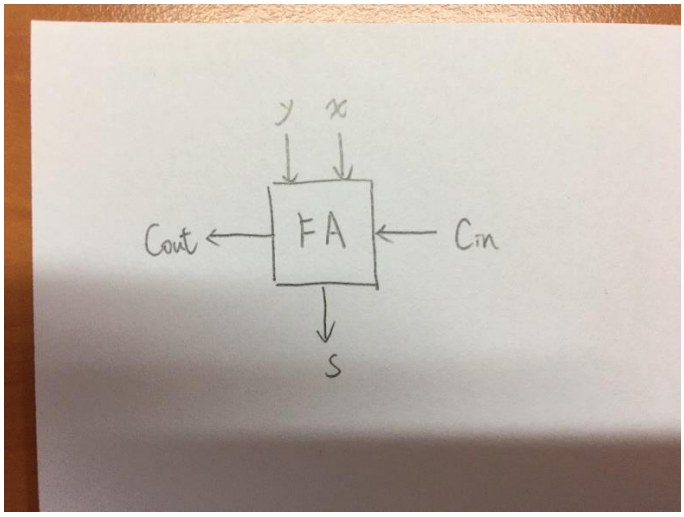
IO:

For a one bit full adder:

Input: x, y, cin

Output: s, cout

Block diagram:

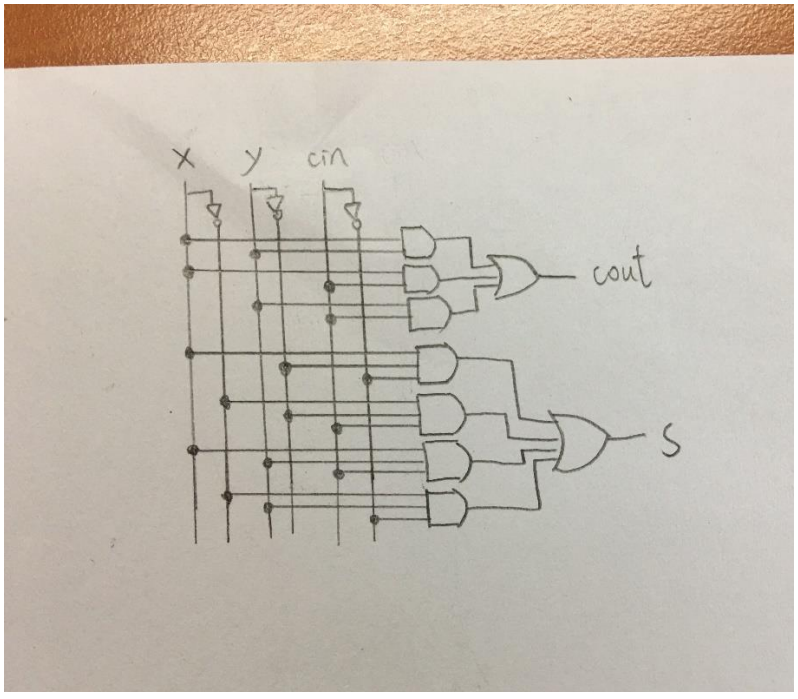



Logic function:

cout = x y + x cin + y cin

s = x y’ cin’ + x’ y’ cin + x y cin + x’ y cin’

Logic diagram:



Verilog:

```
module fulladder(
    input x,
    input y,
    input cin,
    output s,
    output cout
    );

assign s = x&~y&~cin | ~x&~y&cin | x&y&cin | ~x&y&~cin;
assign cout = x&y | y&cin | x&cin;

endmodule
```

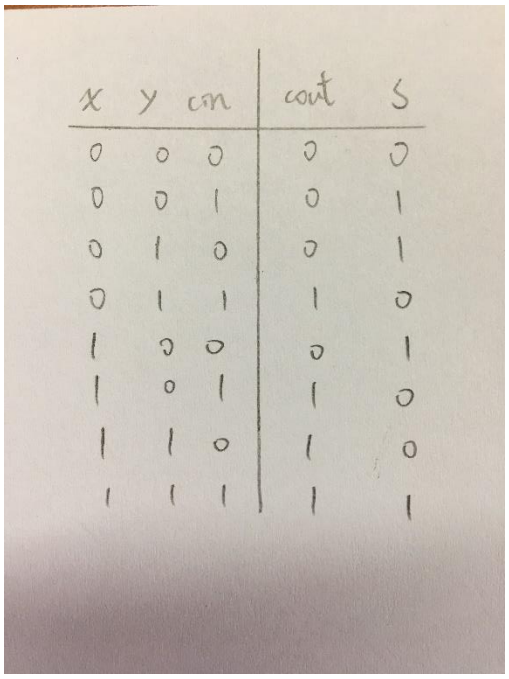
Testbench:

```
module test_fulladder;
wire COUT,S;
reg X,Y,CIN;

fulladder U0(.cout(COUT), .s(S), .x(X), .y(Y), .cin(CIN));

initial
begin
    X=0;Y=0;CIN=0;
    #10 X=0;Y=0;CIN=1;
    #10 X=0;Y=1;CIN=0;
    #10 X=0;Y=1;CIN=1;
    #10 X=1;Y=0;CIN=0;
    #10 X=1;Y=0;CIN=1;
    #10 X=1;Y=1;CIN=0;
    #10 X=1;Y=1;CIN=1;
end

endmodule
```

Simulation:


Discussion:

這個實驗是要建構一個包含進位(carry in、carry out)的加法器。



| x | y | cin | cout | S |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

透過 truth table 可以得知輸入(x、y、cin)與輸出(cout、s)之間的關係；也就是說，當輸入為奇數個 1 時會產生 s，而當輸入超過兩個 1 時，會產生 cout。

在模擬圖中，將輸入值的所有可能性跑過一次，可以驗證加法器產生的結果，最高為 3，最低為 0。

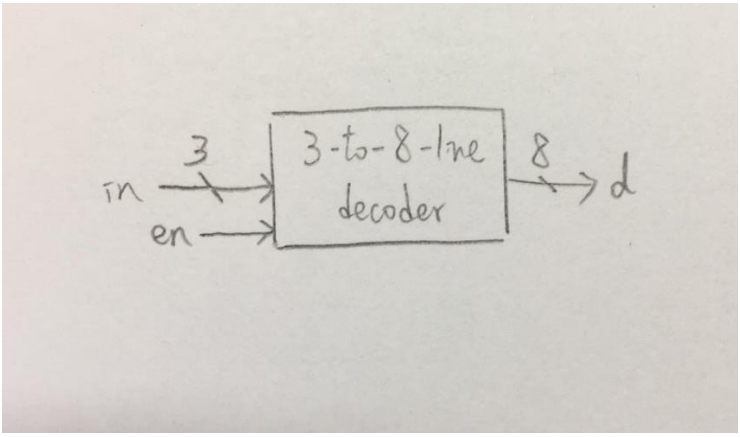
(2) 3-to-8-line decoder with enable:

IO:

Input: [2:0]in, en

Output: [7:0]d

Block diagram:




Logic function:

d7 = in2 in1 in0 en

d6 = in2 in1 in0’ en

d5 = in2 in1’ in0 en

d4 = in2 in1’ in0’ en

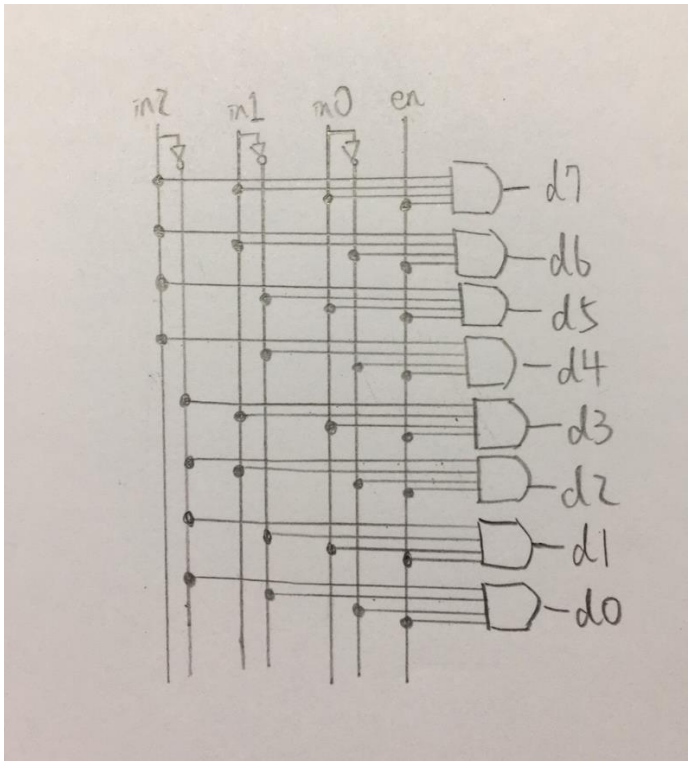
d3 = in2’ in1 in0 en

d2 = in2’ in1 in0’ en

d1 = in2’ in1’ in0 en

d0 = in2’ in1’ in0’ en

Logic diagram:



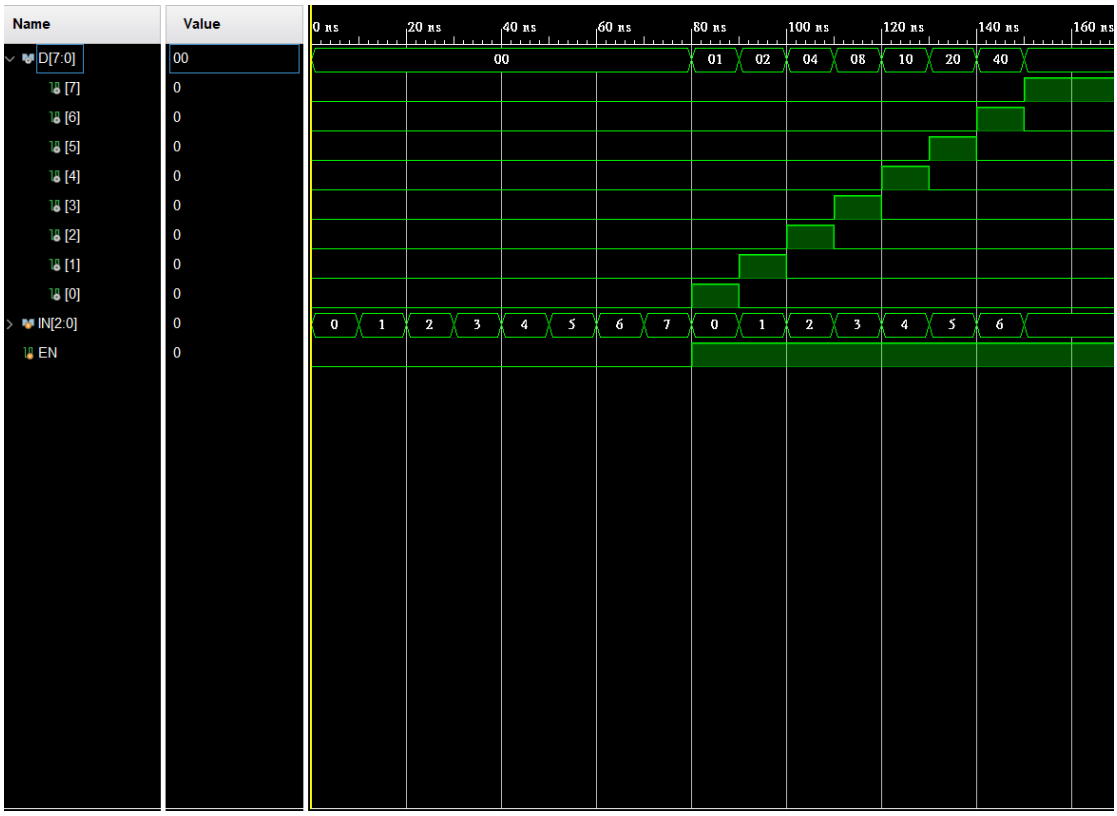

Verilog:

```
module decoder(
    input [2:0]in,
    input en,
    output [7:0]d
    );
assign d[7] = in[2]&in[1]&in[0]&en;
assign d[6] = in[2]&in[1]&~in[0]&en;
assign d[5] = in[2]&~in[1]&in[0]&en;
assign d[4] = in[2]&~in[1]&~in[0]&en;
assign d[3] = ~in[2]&in[1]&in[0]&en;
assign d[2] = ~in[2]&in[1]&~in[0]&en;
assign d[1] = ~in[2]&~in[1]&in[0]&en;
assign d[0] = ~in[2]&~in[1]&~in[0]&en;

endmodule
```

Testbench:

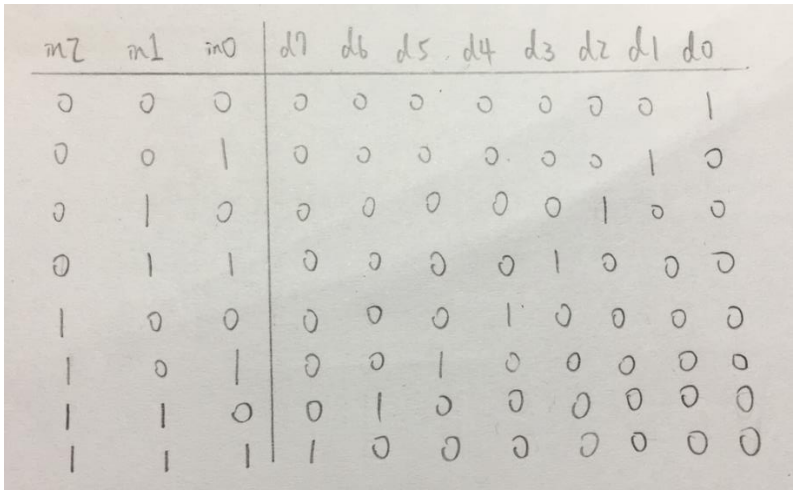
```
module test_3to8decoder;
wire [7:0]D;
reg [2:0]IN;
reg EN;
decoder U0(.d(D), .in(IN), .en(EN));
initial
begin
    EN = 0;IN[2] = 0;IN[1] = 0;IN[0] = 0;
    #10 EN = 0;IN[2] = 0;IN[1] = 0;IN[0] = 1;
    #10 EN = 0;IN[2] = 0;IN[1] = 1;IN[0] = 0;
    #10 EN = 0;IN[2] = 0;IN[1] = 1;IN[0] = 1;
    #10 EN = 0;IN[2] = 1;IN[1] = 0;IN[0] = 0;
    #10 EN = 0;IN[2] = 1;IN[1] = 0;IN[0] = 1;
    #10 EN = 0;IN[2] = 1;IN[1] = 1;IN[0] = 0;
    #10 EN = 0;IN[2] = 1;IN[1] = 1;IN[0] = 1;
    #10 EN = 1;IN[2] = 0;IN[1] = 0;IN[0] = 0;
    #10 EN = 1;IN[2] = 0;IN[1] = 0;IN[0] = 1;
    #10 EN = 1;IN[2] = 0;IN[1] = 1;IN[0] = 0;
    #10 EN = 1;IN[2] = 0;IN[1] = 1;IN[0] = 1;
    #10 EN = 1;IN[2] = 1;IN[1] = 0;IN[0] = 0;
    #10 EN = 1;IN[2] = 1;IN[1] = 0;IN[0] = 1;
    #10 EN = 1;IN[2] = 1;IN[1] = 1;IN[0] = 0;
    #10 EN = 1;IN[2] = 1;IN[1] = 1;IN[0] = 1;
end

endmodule
```

Simulation:




Discussion:

本次實驗目的為將一個 3bit 的 binary 輸入值解碼成 8bit 的 one hot 然後輸出，而 en 為控制系統的開關，藉由 truth table:




可得知輸入與輸出的對應關係；在波形圖中，將輸入值 in 與 en 的所有可能性跑過一次，即得到最後的解碼結果。

3. comparator:

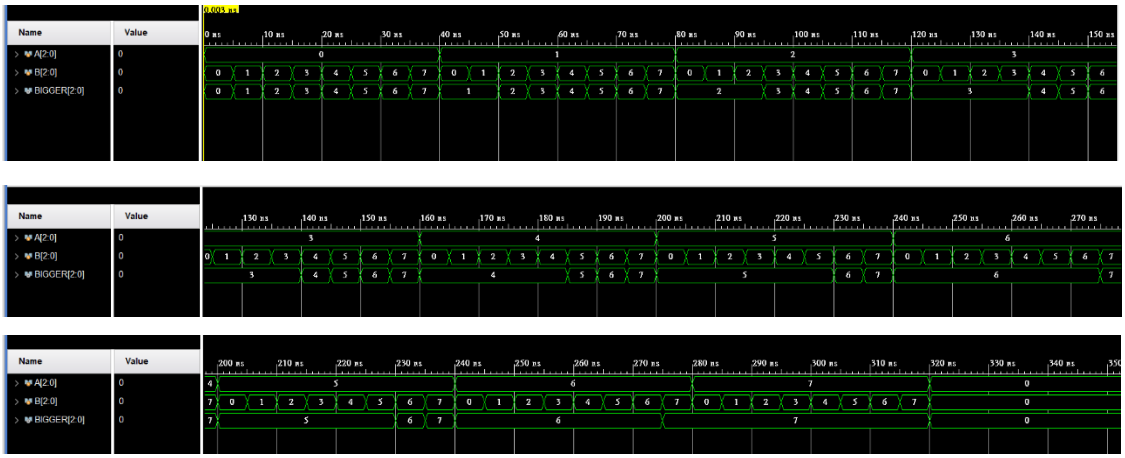
Verilog:

```
module comparator(
    input [2:0]a,
    input [2:0]b,
    output reg [2:0]bigger
    );

always@*
    if (a > b)
        bigger = a;
    else
        bigger = b;
endmodule
```

Testbench:

```
module test_comparator;
reg [2:0]A,B;
wire [2:0]BIGGER;

comparator U0(.a(A), .b(B), .bigger(BIGGER));

initial
begin
    A = 0;B = 0;
    repeat(8)
    begin
        repeat(8)
        #5 B = B + 1;
    A = A + 1;
    end
end

endmodule
```

Simulation:




Discussion:

本實驗比較兩個 3-bit 的 unsigned number，可以利用 verilog 程式裡原本就具備的比較功能就能夠輕易地比較兩數大小，並選擇較大的數輸出。

4. single digit decimal adder:

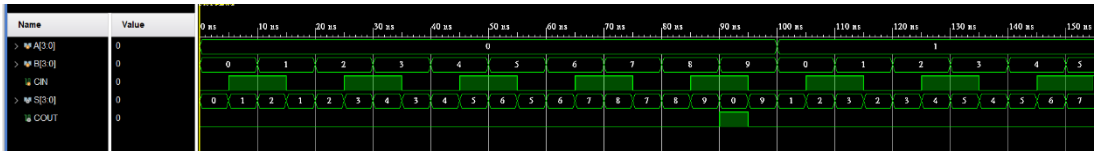
Verilog:

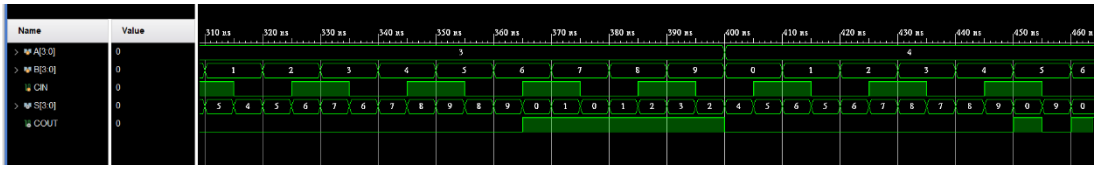
```
module decimalAdder(
    input [3:0]a,
    input [3:0]b,
    input cin,
    output reg [3:0]s,
    output reg cout
    );

always@*
    if (a + b + cin > 9)
        begin
        cout = 1;
        s = a + b + cin + 6;
        end
    else
        begin
        cout = 0;
        s = a + b + cin;
        end
endmodule
```

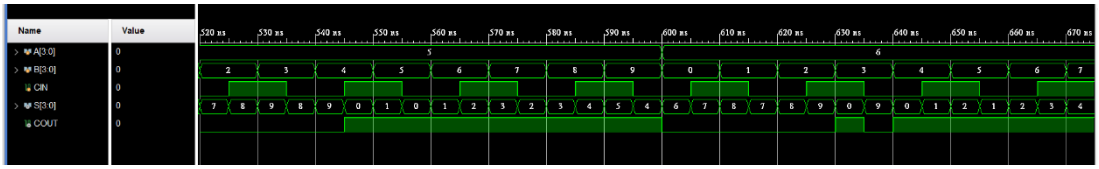
Testbench:

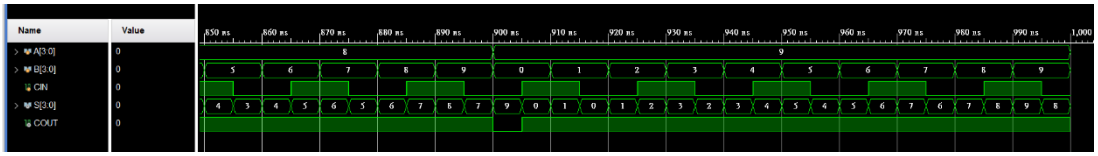
```
module test_decimalAdder;
reg [3:0]A,B;
reg CIN;
wire [3:0]S;
wire COUT;
decimalAdder U0(.a(A), .b(B), .cin(CIN), .s(S), .cout(COUT));

initial
begin
    A = 0;B = 0;CIN = 0;
    repeat(10)
    begin
        repeat(10)
        begin
        #5 CIN = CIN + 1;
        #5 B = B + 1;
        end
    B = 0;
    A = A + 1;
    end
end
endmodule
```

Simulation:






Discussion:

本實驗為建構一個 BCD adder，由於總和大於 9 會產生溢位，因此必須額外加上 6 才能正確的表示十進位的個位數。

Conclusion:

在第一個實驗中，我在進入實驗室前就已經先打好 verilog 了，但是跑出的模擬結果都是 X 跟 Z，因此我便一直想找出哪裡有寫錯，但就是無法找出 bug，直到上課時教授說有時會因為 vivado 自己出了問題而會導致出現 bug，我才終於知道問題所在，最後重開一次 vivado 程式後也順利解決問題。

這次的 lab 雖然不難，但是我已經接近兩個月沒碰 verilog 了，因此在語法等方面都不是那麼的熟練，尤其是 always 等號左邊的變數要宣告成 reg 就是我常常沒注意到的地方，也因此這次實驗還是花了不少時間 debug 和熟悉 verilog，下次實驗就要開始接上電路板了，對我來說又是一個新的挑戰，這學期的邏設實驗讓我終於能將上學期所學的理論應用在實體上，也期望以後的實驗能夠更有效率地完成。