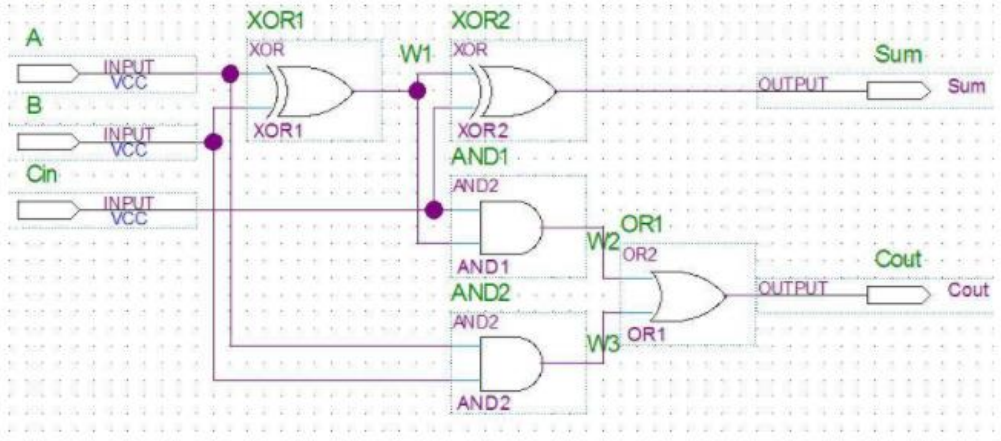
1.模型架構

```
module 模組名稱( In1, In2, Out1, Out2, InOut1 );

    input in1, in2;
    output Out1, Out2;
    inout InOut1;

    wire In1, In2, Out1;
    wire InOut1;
    reg Out2;

    // 以下為三種層級分別描述 In1 與 In2 and 做 and 運算的方法
    // 邏輯閘層次( Gate Level )
    and and1( Out1, In1, In2 );

    // 資料流層次( Dataflow Level )
    assign Out1 = In1 & In2;

    // 行為層次( Behavior Level )
    allways @(*) begin
        Out2 = In1 & In2;
    end

endmodule
```

2.Gate level

一位元全加器



程式碼：

```
module Full_Adder( A, B, Cin, Sum, Cout );

    input A, B, Cin;
    output Sum, Cout;

    wire W1, W2, W3;

    xor xor1( W1, A, B );
    and and1( W2, W1, Cin );
    and and2( W3, A, B );
    xor xor2( Sum, W1, Cin );
    or  or1( Cout, W2, W3 );

endmodule
```

3.Dataflow level

一位元全加器程式碼：

```
module Full_Adder( A, B, Cin, Sum, Cout );

    input A, B, Cin;
    output Sum, Cout;

    wire W1, W2, W3;

    assign W1 = A^B;
    assign W2 = W1&Cin;
    assign W3 = A&B;
    assign Sum = W1^Cin;
    assign Cout = W2|W3;

endmodule
```

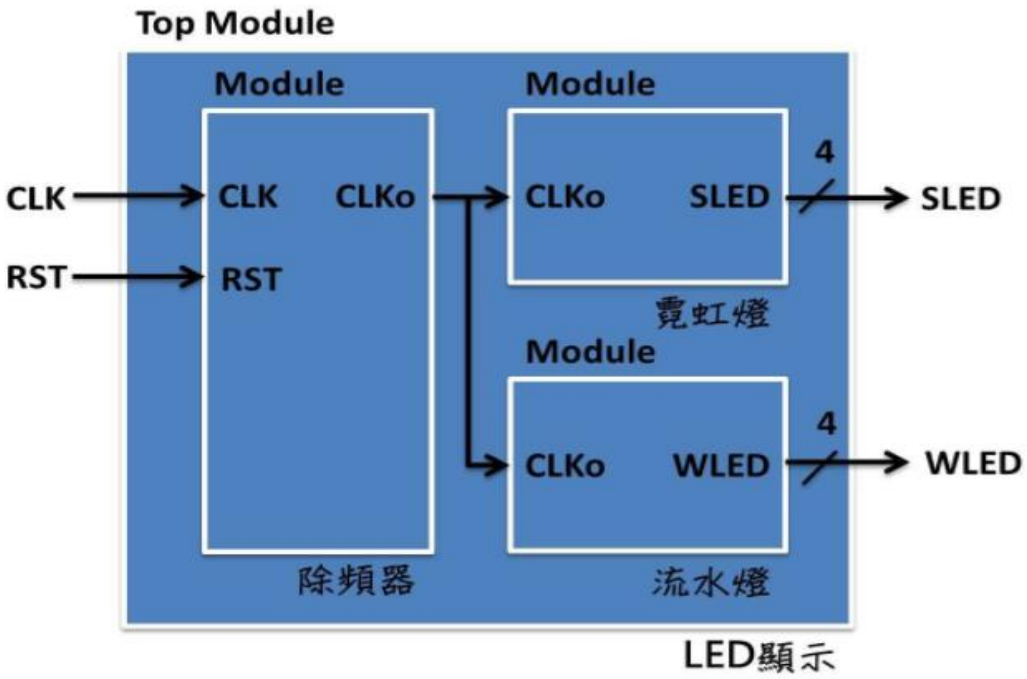
4.Behavior level

一位元全加器程式碼：

```
module Full_Adder( A, B, Cin, Sum, Cout );

    input A, B, Cin;
    output Sum, Cout;

    wire W1, W2, W3;

    always @( A, B, Cin ) begin
        { Cout, Sum } = A + B + Cin;
    end

endmodule
```

5.模組化階層化

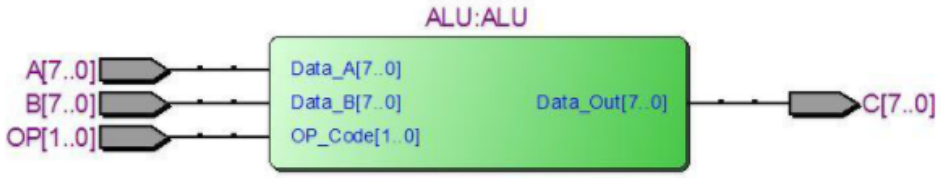
- Module可以有無限多個，但Top Module只能有一個



一位元全加器程式碼：

```
module Top_Module( A, B, Cin, Sum, Cout );

    input A, B, Cin;
    output Sum, Cout;

    Full_Adder FAD(    // 使用 always 的 Full_Adder
        .A(A),
        .B(B),
        .Cin(Cin),
        .Sum(Sum),
        .Cout(Cout)
    );

endmodule
```

6.ALU



程式(ALU) :

```
`define    ADD 2'b00
`define    SUB 2'b01
`define    AND 2'b10
`define    OR  2'b11

module ALU( Data_A, Data_B, OP_Code, Data_Out );

    parameter   Data_Size = 8;
    parameter   OP_Code_Size = 2;

    input   [Data_Size-1:0] Data_A, Data_B;
    input   [OP_Code_Size-1:0] OP_Code;
    output  [Data_Size-1:0] Data_Out;

    reg     [Data_Size-1:0] Data_Out;

    always @( Data_A, Data_B, OP_Code ) begin
        case( OP_Code )
            `ADD:       Data_Out <= Data_A + Data_B;
            `SUB:       Data_Out <= Data_A - Data_B;
            `AND:       Data_Out <= Data_A & Data_B;
            `OR:        Data_Out <= Data_A | Data_B;
            default:    Data_Out <= 0;
        endcase
    end

endmodule
```