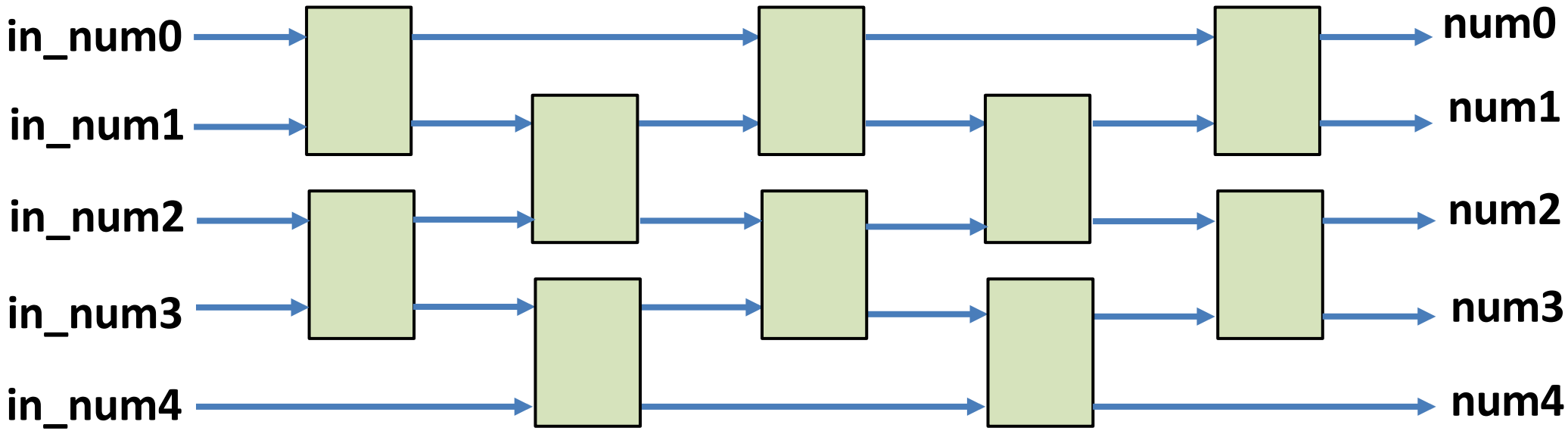
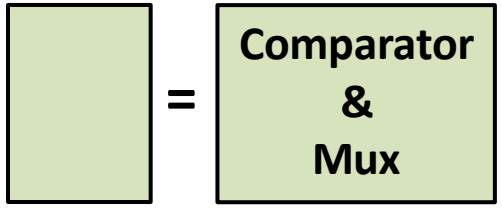
Lab02 Code Review

參考架構 (Merge Sort)





Reference code

Main module - Sort

```
//------------------------------------------------------------------------
//   LOGIC DECLARATION
//------------------------------------------------------------------------
logic [5:0] lv0_n0, lv0_n1, lv0_n2, lv0_n3, lv0_n4;
logic [5:0] lv1_n0, lv1_n1, lv1_n2, lv1_n3, lv1_n4;
logic [5:0] lv2_n0, lv2_n1, lv2_n2, lv2_n3, lv2_n4;
logic [5:0] lv3_n0, lv3_n1, lv3_n2, lv3_n3, lv3_n4;
logic [5:0] lv4_n0, lv4_n1, lv4_n2, lv4_n3, lv4_n4;
```

```
//------------------------------------------------------------------------
//   Your design
//------------------------------------------------------------------------
comparator  comp_lv0_0(.in_0(in_num0), .in_1(in_num1), .out_0(lv0_n0), .out_1(lv0_n1));
comparator  comp_lv0_1(.in_0(in_num2), .in_1(in_num3), .out_0(lv0_n2), .out_1(lv0_n3));
assign lv0_n4 = in_num4;

comparator  comp_lv1_0(.in_0(lv0_n1),  .in_1(lv0_n2),  .out_0(lv1_n1), .out_1(lv1_n2));
comparator  comp_lv1_1(.in_0(lv0_n3),  .in_1(lv0_n4),  .out_0(lv1_n3), .out_1(lv1_n4));
assign lv1_n0 = lv0_n0;

comparator  comp_lv2_0(.in_0(lv1_n0),  .in_1(lv1_n1),  .out_0(lv2_n0), .out_1(lv2_n1));
comparator  comp_lv2_1(.in_0(lv1_n2),  .in_1(lv1_n3),  .out_0(lv2_n2), .out_1(lv2_n3));
assign lv2_n4 = lv1_n4;

comparator  comp_lv3_0(.in_0(lv2_n1),  .in_1(lv2_n2),  .out_0(lv3_n1), .out_1(lv3_n2));
comparator  comp_lv3_1(.in_0(lv2_n3),  .in_1(lv2_n4),  .out_0(lv3_n3), .out_1(lv3_n4));
assign lv3_n0 = lv2_n0;

comparator  comp_lv4_0(.in_0(lv3_n0),  .in_1(lv3_n1),  .out_0(lv4_n0), .out_1(lv4_n1));
comparator  comp_lv4_1(.in_0(lv3_n2),  .in_1(lv3_n3),  .out_0(lv4_n2), .out_1(lv4_n3));
assign lv4_n4 = lv3_n4;
assign out_num = lv4_n2;


endmodule
```

Submodule – Comparator

```
module comparator(
    in_0,
    in_1,
    out_0,
    out_1
);

input  [5:0] in_0, in_1;
output logic [5:0] out_0, out_1;

assign out_0 = (in_0 <= in_1) ? in_0 : in_1;
assign out_1 = (in_0 <= in_1) ? in_1 : in_0;

endmodule
```

Bad coding example

- Logic declaration should be separated with design region
- Enhance readability

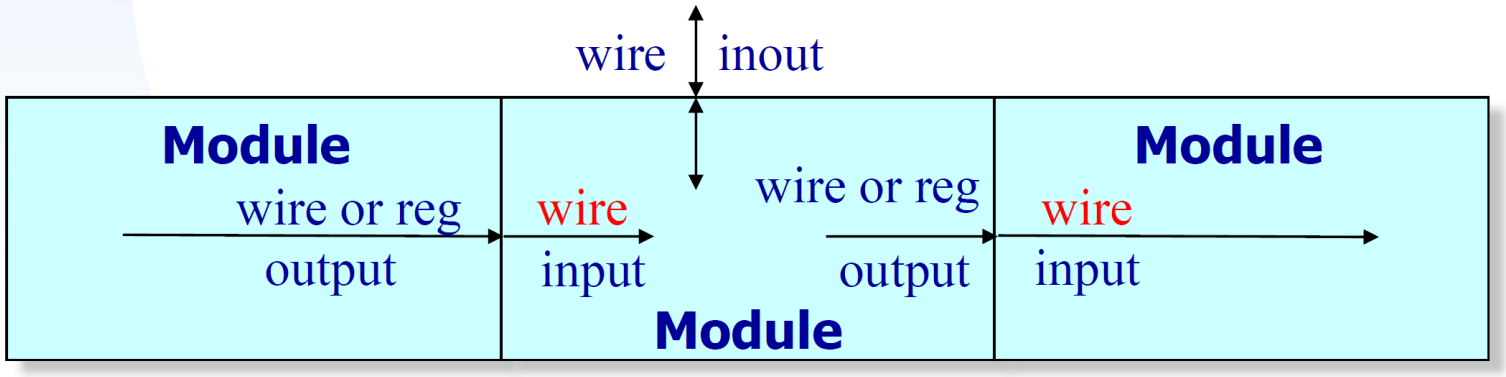
```
//-------------------------------------------------------------------------
//   LOGIC DECLARATION
//-------------------------------------------------------------------------


//-------------------------------------------------------------------------
//   Your design
//-------------------------------------------------------------------------
  wire [5:0] No1_lesser,No1_greater;
  wire [5:0] No2_lesser,No2_greater;
  wire [5:0] No3_lesser,No3_greater;
  wire [5:0] No4_lesser,No4_greater;
  wire [5:0] No5_lesser,No5_greater;
  wire [5:0] No6_lesser,No6_greater;
  wire [5:0] No7_lesser,No7_greater;
  wire [5:0] No8_lesser,No8_greater;
  wire [5:0] No9_lesser,No9_greater;
  wire [5:0] No10_lesser,N010_greater;
```

```
always @ * begin
    logic [5:0] a1;
    logic [5:0] a2;
    logic [5:0] a3;
    logic [5:0] a4;

    logic [5:0] b1;
    logic [5:0] b2;
    logic [5:0] b3;
    logic [5:0] b4;

    logic [5:0] c1;
    logic [5:0] c2;
    logic [5:0] c3;
    logic [5:0] c4;

    //1st layer
    if(in_num0 < in_num1) begin
        a1 = in_num0;
        a2 = in_num1;
    end
    else begin
        a2 = in_num0;
        a1 = in_num1;
    end
```

Bad coding example

- In SystemVerilog, logic can represent wire or reg
- However, you should note that the default input/output declaration is wire!
- Wire cannot be assigned in always block!

```
module Comp_and_Mux(
  input [5:0] in1, in2,
  output [5:0] lesser,greater
);
  always@* begin
    if (in1 >= in2) begin
      greater = in1;
      lesser = in2;
    end
    else begin
      greater = in2;
      lesser = in1;
    end
  end
endmodule
```



Bad coding example

- Note that initial block cannot be synthesized!

```
//------------------------------------------------------------------------
//   LOGIC DECLARATION
//------------------------------------------------------------------------
logic [5:0] num [0:4];
initial
begin
num[0]=in_num0;
num[1]=in_num1;
num[2]=in_num2;
num[3]=in_num3;
num[4]=in_num4;
end
```

Bad coding example

- Assign statement cannot use non-blocking assignment!

```
assign {temp[0],temp[1]} <= (in_num0 < in_num1)?{in_num0,in_num1}:{in_num1,in_num0};
assign {temp[2],temp[3]} <= (in_num2 < in_num3)?{in_num2,in_num3}:{in_num3,in_num2};
assign {temp[4],temp[5]} <= (temp[1] < temp[2])?{temp[1],temp[2]}:{temp[2],temp[1]};
assign {temp[6],temp[7]} <= (in_num4 < temp[3])?{in_num4,temp[3]}:{temp[3],in_num4};
assign {temp[8],temp[9]} <= (temp[0] < temp[4])?{temp[0],temp[4]}:{temp[4],temp[0]};
assign {temp[10],temp[11]} <= (temp[5] < temp[6])?{temp[5],temp[6]}:{temp[6],temp[5]};
assign {temp[12],temp[13]} <= (temp[9] < temp[10])?{temp[9],temp[10]}:{temp[10],temp[9]};
assign {temp[14],temp[15]} <= (temp[11] < temp[7])?{temp[11],temp[7]}:{temp[7],temp[11]};
assign {temp[16],temp[17]} <= (temp[8] < temp[12])?{temp[8],temp[12]}:{temp[12],temp[8]};
assign {temp[18],temp[19]} <= (temp[13] < temp[14])?{temp[13],temp[14]}:{temp[14],temp[13]};

assign out_num <= temp[18];
```

Bad coding example

- Don’t use software thinking to write hardware!
- Before writing the code, you should know how the hardware work & generate!

```
//---------------------------------------------------------------------
//   LOGIC DECLARATION
//---------------------------------------------------------------------
    logic [5:0] nums [4:0]; // Array to store the inputs for sorting
    logic [5:0] temp; // Temporary variable used for swapping


//---------------------------------------------------------------------
//   Your design
//---------------------------------------------------------------------

    always_comb begin
        // Load inputs into the array
        nums[0] = in_num0;
        nums[1] = in_num1;
        nums[2] = in_num2;
        nums[3] = in_num3;
        nums[4] = in_num4;

        // bubble sort
        for (int i = 0; i < 4; i++) begin
            for (int j = 0; j < 4 - i; j++) begin
                if (nums[j] > nums[j + 1]) begin
                    // Swap nums[j] and nums[j+1]
                    temp = nums[j];
                    nums[j] = nums[j + 1];
                    nums[j + 1] = temp;
                end
            end
        end

        // the median will be at nums[2]
        out_num = nums[2];
    end
```