VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL AND ELECTRONICS ENGINEERING

—o0o—

HOMEWORK REPORT

Chương 3 - Thiết kế mạch tổ hợp

SUPERVISOR: Nguyễn Trung Hiếu

SUBJECT: Digital System Design and Verification (EE3213)

GROUP: 04

List of Members

STT	MSSV	Họ Và Tên	Lớp
1	2213874	Nguyễn Thanh Tùng	L01
2	2210780	Nguyễn Đại Đồng	L01
3	2213496	Nguyễn Quốc Tín	L01

Ho Chi Minh, ../../20..

Mục lục

Danh sách hình vẽ

Danh sách bảng

List of Listings

Câu 2

Thiết kế mạch tổ hợp tìm vị trí bit 1 đầu tiên (tính từ MSB) của chuỗi 24-bit. Cho các standard cell như sau: cổng not, các cổng logic 2 ngõ vào, mux 2-1, mux 4-1.

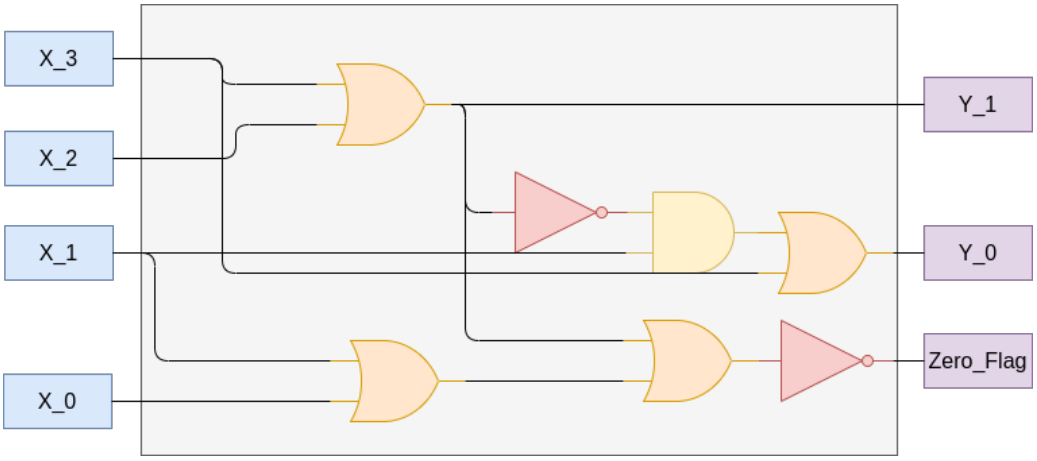
a) Thiết kế mạch chỉ được dùng các standard cell trên.

Đầu tiên nhóm em sẽ thiết kế từ một bộ tìm kiếm vị trí bit 1 đầu tiên (tính từ MSB) cho một chuỗi 4-bit trước.

Input				Output		Zero Flag
X_3	X_2	X_1	X_0	Y_1	Y_0	V
0	0	0	0	0	0	1
0	0	0	1	0	0	0
0	0	1	X	0	1	0
0	1	X	X	1	0	0
1	X	X	X	1	1	0

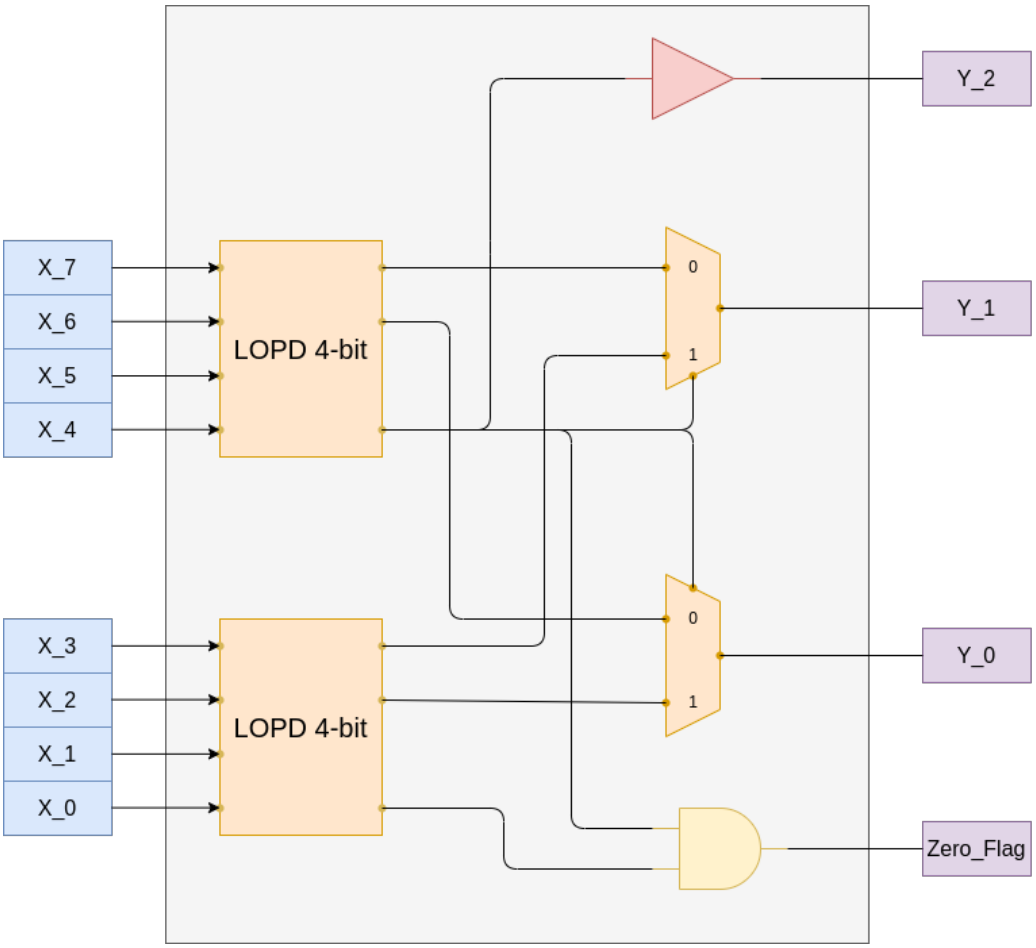
Bảng 1: Bảng sự thật của bộ phát hiện bit 1 (Leading one position) cho 4 bit.

Từ bảng 1, ta rút gọn và có được mạch như sau:

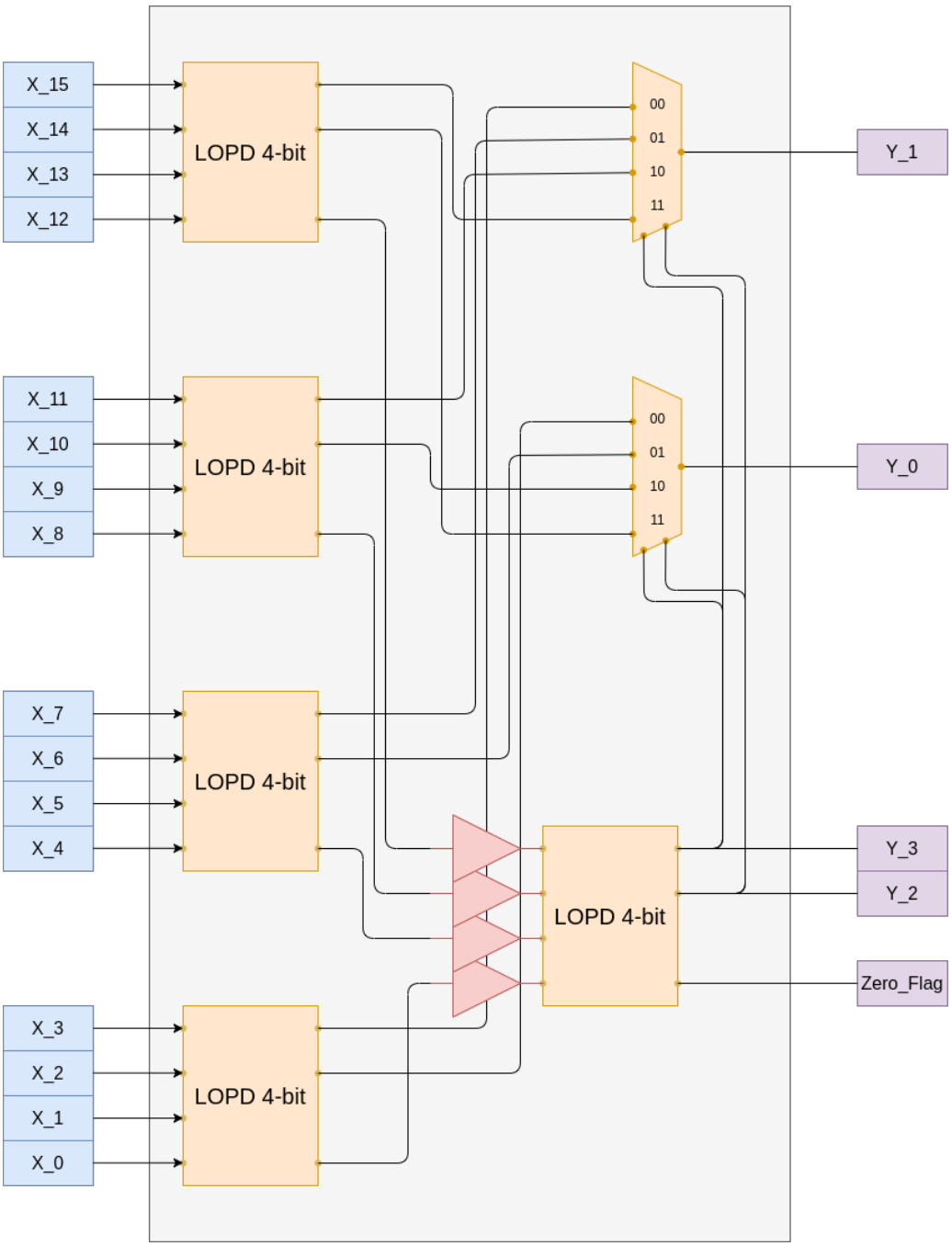


Hình 1: Sơ đồ logic của bộ LOPD 4bit.

Từ bộ LOPD 4-bit trên, ta triển khai bộ LOPD 8-bit và bộ LOPD 16-bit như sau:

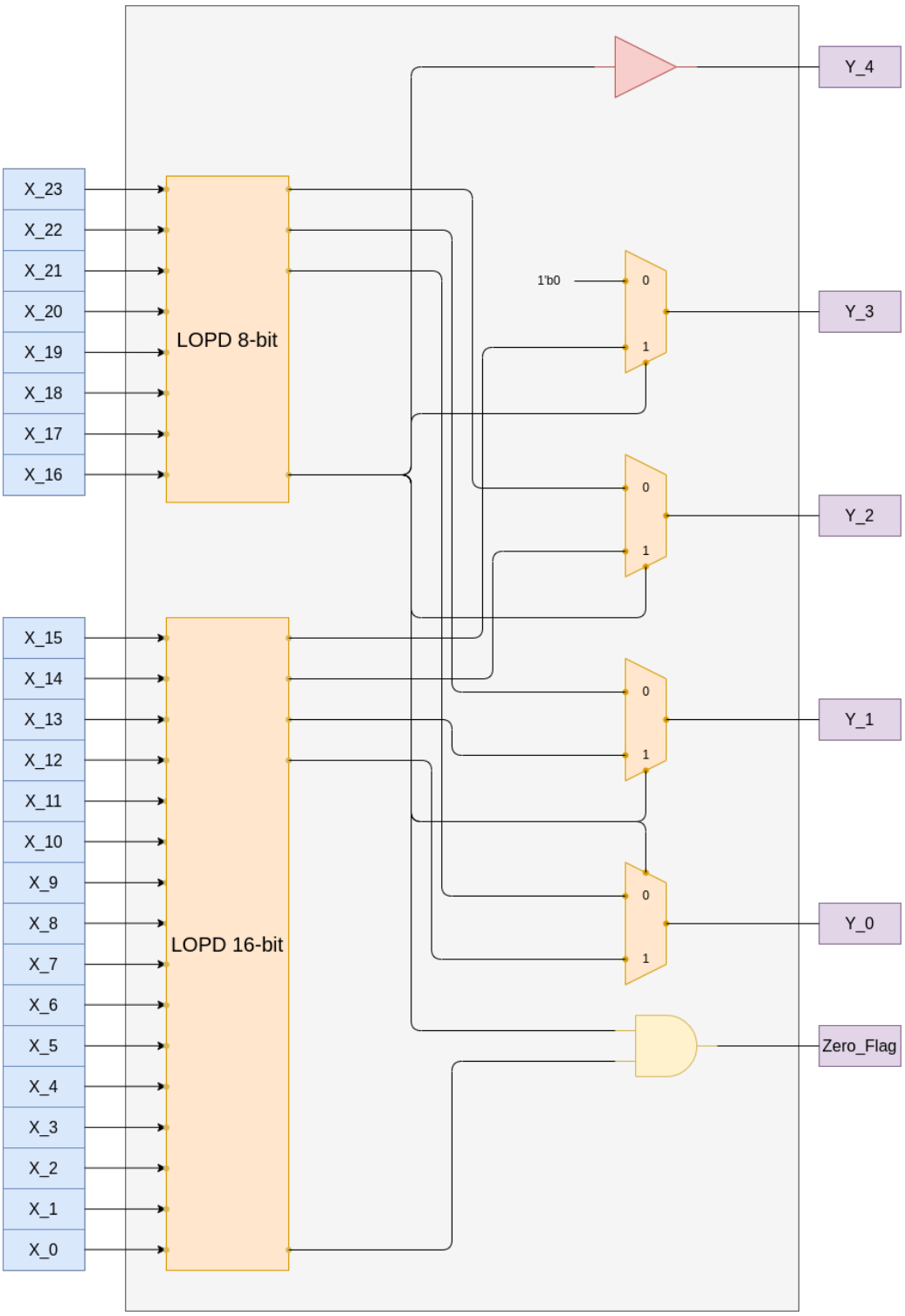


Hình 2: Sơ đồ logic của bộ LOPD 8bit.



Hình 3: Sơ đồ logic của bộ LOPD 16bit.

Từ bộ LOPD 8-bit và LOPD 16-bit trên, ta ghép lại thành 24-bit với LOPD 8-bit vào vị trí 8-bit cao (từ 23 → 16) và bộ LOPD 16-bit vào 16-bit thấp (từ 15 → 0).



Hình 4: Sơ đồ logic của bộ LOPD 24-bit.

b) Viết chương trình HDL mô tả mạch đã cho.

```
module LOPD_4bit(
    input logic [3:0]       i_data  ,
    output logic [1:0]      o_pos_one,
    output logic            o_zero_flag
);

////////////////// LDO //////////////////
// D[3] | D[2] | D[1] | D[0] |PO[3] | PO[2] | PO[1] | PO[0] | ZERO_FLAG |
//   0  |   0  |   0  |   0  |  0   |   0   |   0   |   0   |     1     |
//   0  |   0  |   0  |   1  |  0   |   0   |   0   |   1   |     0     |
//   0  |   0  |   1  |   X  |  0   |   0   |   1   |   0   |     0     |
//   0  |   1  |   X  |   X  |  0   |   1   |   0   |   0   |     0     |
//   1  |   X  |   X  |   X  |  1   |   0   |   0   |   0   |     0     |
////////////////// LDOD //////////////////
// D[3] | D[2] | D[1] | D[0] | PO[1] | PO[0] | ZERO_FLAG |
//   0  |   0  |   0  |   0  |   0   |   0   |     1     |
//   0  |   0  |   0  |   1  |   0   |   0   |     0     |
//   0  |   0  |   1  |   X  |   0   |   1   |     0     |
//   0  |   1  |   X  |   X  |   1   |   0   |     0     |
//   1  |   X  |   X  |   X  |   1   |   1   |     0     |

assign o_zero_flag = ~((o_pos_one[1])|(i_data[1]|i_data[0])) ;
assign o_pos_one[1] = i_data[3] | i_data[2];
assign o_pos_one[0] = ((~(i_data[3] | i_data[2])) & (i_data[1])) |  (i_data[3]);

endmodule
```

Listing 1: Chương trình mô tả LOPD 4-bit.

```
module LOPD_8bit(
    input logic [7:0]       i_data  ,
    output logic [2:0]      o_pos_one,
    output logic            o_zero_flag
);

//////////////////////////////////////////////////////////
// LOPD_4bit_unit_0
//////////////////////////////////////////////////////////
logic w_zero_flag_0;
logic [1:0] w_pos_one_0;
LOPD_4bit LOPD_4bit_unit_0 (
    .i_data         (i_data[3:0]),
    .o_pos_one      (w_pos_one_0),
    .o_zero_flag    (w_zero_flag_0)
);

//////////////////////////////////////////////////////////
// LOPD_4bit_unit_1
//////////////////////////////////////////////////////////
logic w_zero_flag_1;
logic [1:0] w_pos_one_1;
LOPD_4bit LOPD_4bit_unit_1 (
    .i_data         (i_data[7:4]),
    .o_pos_one      (w_pos_one_1),
    .o_zero_flag    (w_zero_flag_1)
);

//////////////////////////////////////////////////////////
// LOD_8bit_unit
//////////////////////////////////////////////////////////
assign o_zero_flag = w_zero_flag_0 & w_zero_flag_1;
assign o_pos_one[2] = ~(w_zero_flag_1);
```

```
assign o_pos_one[1] = (w_zero_flag_1) ? w_pos_one_0[1] : w_pos_one_1[1];
assign o_pos_one[0] = (w_zero_flag_1) ? w_pos_one_0[0] : w_pos_one_1[0];

endmodule
```

Listing 2: Chương trình mô tả LOPD 8-bit.

```
module LOPD_16bit(
    input logic [15:0]       i_data  ,
    output logic [3:0]      o_pos_one,
    output logic            o_zero_flag
);


// ////////////////////////////////////////////////////////////
// // LOPD_4bit_unit
// ////////////////////////////////////////////////////////////
logic [1:0] w_one_position_0_0;
logic       w_zero_flag_0_0;
LOPD_4bit LOPD4BIT_0_0 (
    .i_data         (i_data[3:0]),
    .o_pos_one      (w_one_position_0_0),
    .o_zero_flag    (w_zero_flag_0_0)
);
logic [1:0] w_one_position_0_1;
logic       w_zero_flag_0_1;
LOPD_4bit LOPD4BIT_0_1 (
    .i_data         (i_data[7:4]),
    .o_pos_one      (w_one_position_0_1),
    .o_zero_flag    (w_zero_flag_0_1)
);
logic [1:0] w_one_position_0_2;
logic       w_zero_flag_0_2;
LOPD_4bit LOPD4BIT_0_2 (
    .i_data         (i_data[11:8]),
    .o_pos_one      (w_one_position_0_2),
    .o_zero_flag    (w_zero_flag_0_2)
);
logic [1:0] w_one_position_0_3;
logic       w_zero_flag_0_3;
LOPD_4bit LOPD4BIT_0_3 (
    .i_data         (i_data[15:12]),
    .o_pos_one      (w_one_position_0_3),
    .o_zero_flag    (w_zero_flag_0_3)
);

logic [1:0] w_one_position_1_0;
LOPD_4bit LOPD4BIT_1_0 (
    .i_data         ({~w_zero_flag_0_3, ~w_zero_flag_0_2, ~w_zero_flag_0_1, ~w_zero_flag_0_2}),
    .o_pos_one      (w_one_position_1_0),
    .o_zero_flag    (o_zero_flag)
);

// ////////////////////////////////////////////////////////////
// // LOD_16bit_unit
// ////////////////////////////////////////////////////////////
MUX_4_1 #(
    .SIZE_DATA  (1)
) MUX_4_1_0(
    .i_data_0       (w_one_position_0_0[0]),
    .i_data_1       (w_one_position_0_1[0]),
    .i_data_2       (w_one_position_0_2[0]),
    .i_data_3       (w_one_position_0_3[0]),
```

```
    .i_select       ({o_pos_one[3], o_pos_one[2]}),
    .o_data         (o_pos_one[0])
);
MUX_4_1 #(
    .SIZE_DATA  (1)
) MUX_4_1_1(
    .i_data_0       (w_one_position_0_0[1]),
    .i_data_1       (w_one_position_0_1[1]),
    .i_data_2       (w_one_position_0_2[1]),
    .i_data_3       (w_one_position_0_3[1]),
    .i_select       ({o_pos_one[3], o_pos_one[2]}),
    .o_data         (o_pos_one[1])
);

assign o_pos_one[2] = w_one_position_1_0[0];
assign o_pos_one[3] = w_one_position_1_0[1];

endmodule

module MUX_4_1 #(
    parameter SIZE_DATA = 1
)(
    input logic [SIZE_DATA-1:0]     i_data_0    ,
    input logic [SIZE_DATA-1:0]     i_data_1    ,
    input logic [SIZE_DATA-1:0]     i_data_2    ,
    input logic [SIZE_DATA-1:0]     i_data_3    ,
    input logic [1:0]               i_select    ,
    output logic [SIZE_DATA-1:0]    o_data
);
reg [SIZE_DATA-1:0] w_o_mux;
always_comb begin : MUX_4_1_1
    case (i_select)
        2'b00:
            w_o_mux = i_data_0;
        2'b01:
            w_o_mux = i_data_1;
        2'b10:
            w_o_mux = i_data_2;
        2'b11:
            w_o_mux = i_data_3;
        default:
            w_o_mux = '0;
    endcase
end
assign o_data = w_o_mux;

endmodule
```

Listing 3: Chương trình mô tả LOPD 16-bit.

```
module Question2 #(
    parameter SIZE_DATA     = 24    ,
    parameter SIZE_LOPD     = 5
)(
    input logic [SIZE_DATA-1:0]     i_data          ,
    output logic [SIZE_LOPD-1:0]    o_one_position  ,
    output logic                    o_zero_flag
);

logic [15:0]     LOPD16_i_data;
assign LOPD16_i_data = i_data[15:0];
logic [3:0]      LOPD16_o_pos_one;
logic            LOPD16_o_zero_flag;
```

```
14 logic [7:0]      LOPD8_i_data;
15 assign LOPD8_i_data = i_data[23:16];
16 logic [2:0]      LOPD8_o_pos_one;
17 logic            LOPD8_o_zero_flag;
18 LOPD_16bit LOPD_16bit_UNIT_LSB (
19     .i_data             (LOPD16_i_data),
20     .o_pos_one          (LOPD16_o_pos_one),
21     .o_zero_flag        (LOPD16_o_zero_flag)
22 );
23
24 LOPD_8bit LOPD_8bit_UNIT_MSB (
25     .i_data             (LOPD8_i_data),
26     .o_pos_one          (LOPD8_o_pos_one),
27     .o_zero_flag        (LOPD8_o_zero_flag)
28 );
29
30 assign o_zero_flag = LOPD16_o_zero_flag & LOPD8_o_zero_flag;
31 assign o_one_position[0] = LOPD8_o_zero_flag ? LOPD16_o_pos_one[0] : LOPD8_o_pos_one[0];
32 assign o_one_position[1] = LOPD8_o_zero_flag ? LOPD16_o_pos_one[1] : LOPD8_o_pos_one[1];
33 assign o_one_position[2] = LOPD8_o_zero_flag ? LOPD16_o_pos_one[2] : LOPD8_o_pos_one[2];
34 assign o_one_position[3] = LOPD8_o_zero_flag ? LOPD16_o_pos_one[3] : 1'b0;
35 assign o_one_position[4] = ~LOPD8_o_zero_flag;
36
37 endmodule
```

Listing 4: Chương trình mô tả LOPD 24-bit.

c) Viết testbench cho mạch, thực hiện testbench với 100 mẫu và tính scoreboard của 100 mẫu đó.

Đầu tiên nhóm em thực hiện triển khai chứng minh kết quả đúng bằng giải thuật sau:

```
1    function automatic logic [SIZE_LOP-1:0] Test_LOPD(
2       input logic [SIZE_DATA-1:0]     f_i_data
3    );
4       logic [SIZE_DATA-1:0] t_temp;
5       int cnt_position_1;
6       begin
7          t_temp = f_i_data;
8          cnt_position_1 = 0;
9
10         if(t_temp == 0) begin
11            Test_LOPD = 0;
12         end else begin
13            while (t_temp[SIZE_DATA-1] == 0) begin
14               t_temp = t_temp << 1;
15               cnt_position_1 ++;
16            end
17            Test_LOPD = SIZE_DATA - cnt_position_1 - 1;
18         end
19      end
20   endfunction
```

Listing 5: Giải thuật chứng minh kết quả của bộ LOPD 24-bit.

- TestCase1: Thực hiện test tất cả các giá trị bit 1 với dùng phương pháp shift left để test tất cả các vị trí bit 1 trong 24-bit.

```
      bit_pos = 1;
      repeat (24) begin
         @(posedge i_clk);
         #1;
         i_addr = i_addr + 1;
         i_data  = bit_pos;
         @(negedge i_clk);
         #1;
         $display("[TIME: %5t] [%s] i_data = %b (%d) \t| o_one_position = %b (%d) \t| o_zero_flag = %b", $time, "Direcly", i_data, i_data, o_one_position, o_one_position, o_zero_flag);
         $display("=> %4s: Expect: %8h, DUT: %8h ", (Test_LOPD(i_data) == o_one_position) ? "PASS" : "FAIL", o_one_position, Test_LOPD(i_data));
         test_count = test_count + 1;
         test_pass  = (Test_LOPD(i_data) == o_one_position) ? test_pass + 1 : test_pass;
         bit_pos = bit_pos << 1'b1;
      end

```

Listing 6: Test 24 trường hợp vị trí bit 1 cho bộ LOPD 24-bit.

Kết quả

```
[TIME: 41000] [Direcly] i_data = 000000000000000000000001 (       1)  | o_one_position = 00000 (0)   | o_zero_flag = 0
=> PASS: Expect: 00000000, DUT: 00000000
[TIME: 51000] [Direcly] i_data = 000000000000000000000010 (       2)  | o_one_position = 00001 (1)   | o_zero_flag = 0
=> PASS: Expect: 00000001, DUT: 00000001
[TIME: 61000] [Direcly] i_data = 000000000000000000000100 (       4)  | o_one_position = 00010 (2)   | o_zero_flag = 0
=> PASS: Expect: 00000002, DUT: 00000002
[TIME: 71000] [Direcly] i_data = 000000000000000000001000 (       8)  | o_one_position = 00011 (3)   | o_zero_flag = 0
=> PASS: Expect: 00000003, DUT: 00000003
...
[TIME: 261000] [Direcly] i_data = 010000000000000000000000 ( 4194304)   | o_one_position = 10110 (22)   | o_zero_flag = 0
=> PASS: Expect: 00000016, DUT: 00000016
[TIME: 271000] [Direcly] i_data = 100000000000000000000000 ( 8388608)   | o_one_position = 10111 (23)   | o_zero_flag = 0
=> PASS: Expect: 00000017, DUT: 00000017
```

Listing 7: Kết quả của TestCase1.

- TestCase2: Thực hiện test ngẫu nhiên giá trị đầu vào.

```
      repeat (100) begin
         @(posedge i_clk);
         #1;
         bit_pos = $urandom_range(0, SIZE_DATA-1);
         i_data = 24'b1 << bit_pos;
         if ($urandom_range(0, 1)) begin
         i_data |= $urandom_range(0, (1 << SIZE_DATA) - 1);
         end
         #5;
         $display("[TIME: %5t] [%s] i_data = %b (%d) \t| o_one_position = %b (%d) \t| o_zero_flag = %b", $time, "Random", i_data, i_data, o_one_position, o_one_position, o_zero_flag);
```

```
11          $display("=> %4s: Expect: %8h, DUT: %8h ", (Test_LOPD(i_data) == o_one_position) ? "PASS" :
    "FAIL", o_one_position, Test_LOPD(i_data));
12          test_count = test_count + 1;
13          test_pass  = (Test_LOPD(i_data) == o_one_position) ? test_pass + 1 : test_pass;
14          i_addr = i_addr + 1;
15        end
16
```

Listing 8: Test 100 trường hợp đầu vào ngẫunhiên cho bộ LOPD 24-bit.

Kết quả

```
[TIME: 281000] [Random] i_data = 000100000000000000000000 ( 1048576)  | o_one_position = 10100
(20)   | o_zero_flag = 0
=> PASS: Expect: 00000014, DUT: 00000014
[TIME: 291000] [Random] i_data = 000000000000000000000100 (       4)  | o_one_position = 00010 (
2)   | o_zero_flag = 0
=> PASS: Expect: 00000002, DUT: 00000002
[TIME: 301000] [Random] i_data = 000000000000000000000001 (       1)  | o_one_position = 00000 (
0)   | o_zero_flag = 0
=> PASS: Expect: 00000000, DUT: 00000000
[TIME: 311000] [Random] i_data = 010000000000000000000000 ( 4194304)  | o_one_position = 10110
(22)   | o_zero_flag = 0
=> PASS: Expect: 00000016, DUT: 00000016
...
[TIME: 1241000] [Random] i_data = 010000000000000000000000 ( 4194304)   | o_one_position = 10110
(22)   | o_zero_flag = 0
=> PASS: Expect: 00000016, DUT: 00000016
[TIME: 1251000] [Random] i_data = 000000000100000000000000 (   16384)   | o_one_position = 01110
(14)   | o_zero_flag = 0
=> PASS: Expect: 0000000e, DUT: 0000000e
[TIME: 1261000] [Random] i_data = 000000100000000000000000 (  131072)   | o_one_position = 10001
(17)   | o_zero_flag = 0
=> PASS: Expect: 00000011, DUT: 00000011
[TIME: 1271000] [Random] i_data = 000000000010000000000000 (    8192)   | o_one_position = 01101
(13)   | o_zero_flag = 0
=> PASS: Expect: 0000000d, DUT: 0000000d
```

Listing 9: Kết quả của TestCase2.

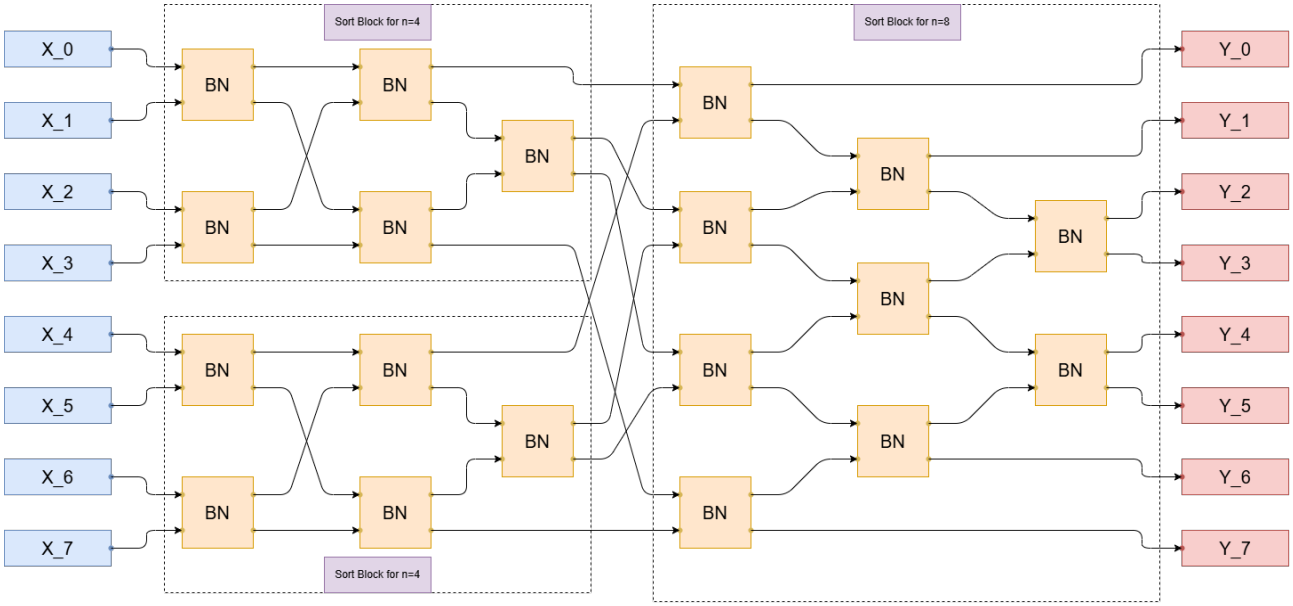
- Kết quả tổng kết.

```
================================
==========TEST SUMMARY==========
Total test cases:    124
Passed          :    124
Failed          :      0
Pass rate       : 100.00%
================================
```

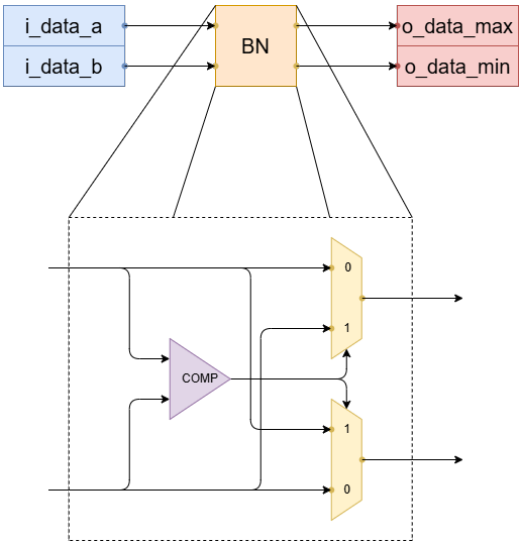
Listing 10: Kết quả của tổng kết của bài test.

Câu 6

Thiết kế một bộ sắp xếp song song như hình dưới, mô tả giải thuật sắp xếp 8 mẫu ngõ vào x_1, x_2,..., x_8, và cho ngõ ra là y_1, y_2, ..., y_8 theo thứ tự giảm dần (hoặc tăng dần).



Trong đó, mỗi bộ BN (Bitonic Sort) có cấu trúc như hình:



Cho các standard cell là: Cổng NOT, các cổng logic 2 ngõ vào, Mux 2-1, Mux 4-1.

a) Giả sử các mẫu ngõ vào có độ rộng là 8bit. Thiết kế mạch trên theo phương pháp đã cho và chỉ được sử dụng các standard cell trên.

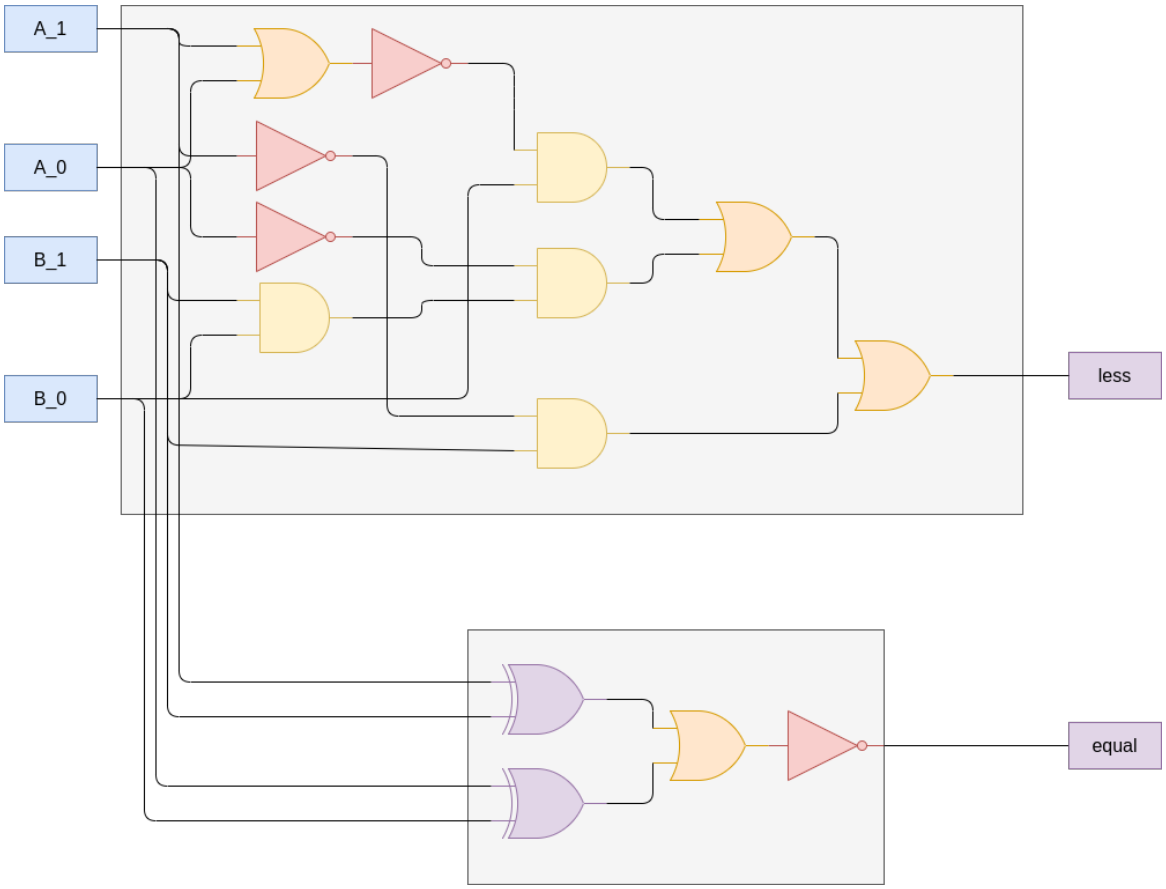
- **Bộ so sánh 8bit**

  + Triển khai bộ so sánh 2-bit

Input				Output	
a_1	a_0	b_1	b_0	less	equal
0	0	0	0	0	1
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	1	0
0	1	0	0	0	0
0	1	0	1	0	1
0	1	1	0	1	0
0	1	1	1	1	0
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	0	1
1	0	1	1	0	0
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	0	0
1	1	1	1	0	1

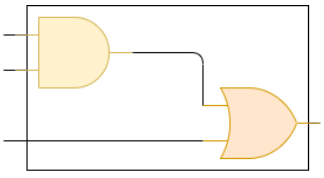
Bảng 2: Bảng sự thật cho bộ so sánh 2-bit $A < B$.

Từ bộ bảng sự thật 2, ta rút gọn kìa K về dạng như sau:

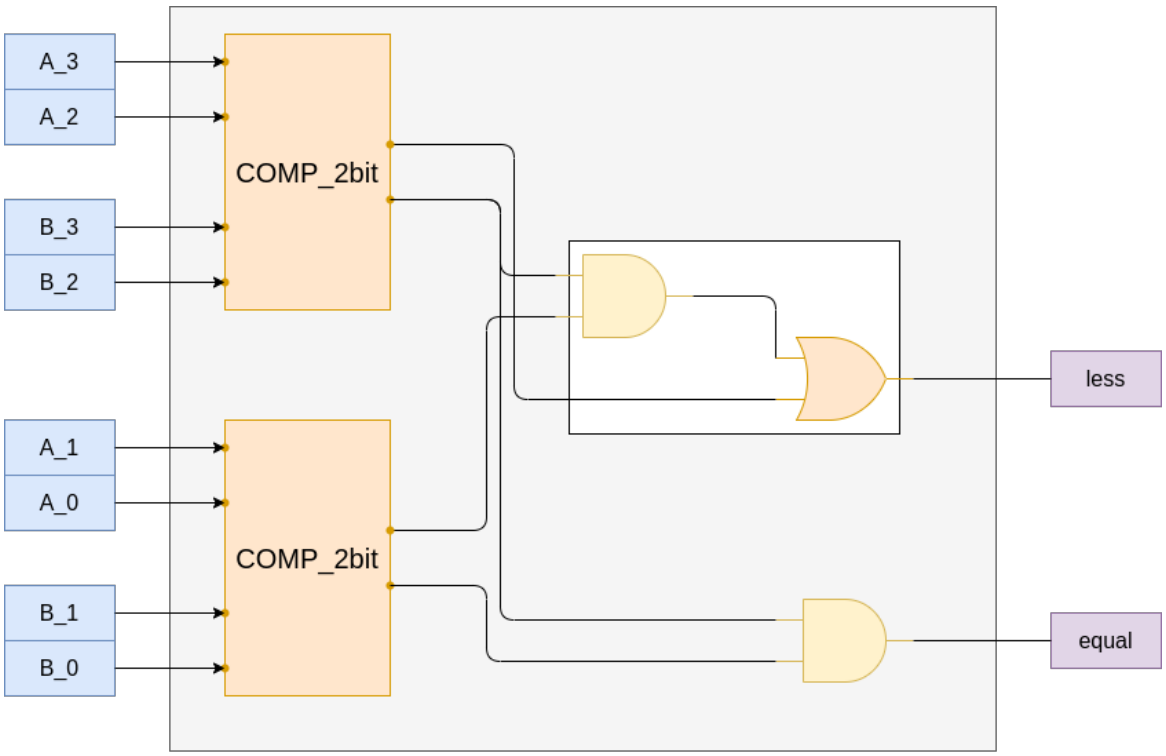


Hình 5: Sơ đồ logic của bộ Comparator 2-bit.

+ Triển khai bộ so sánh 4-bit



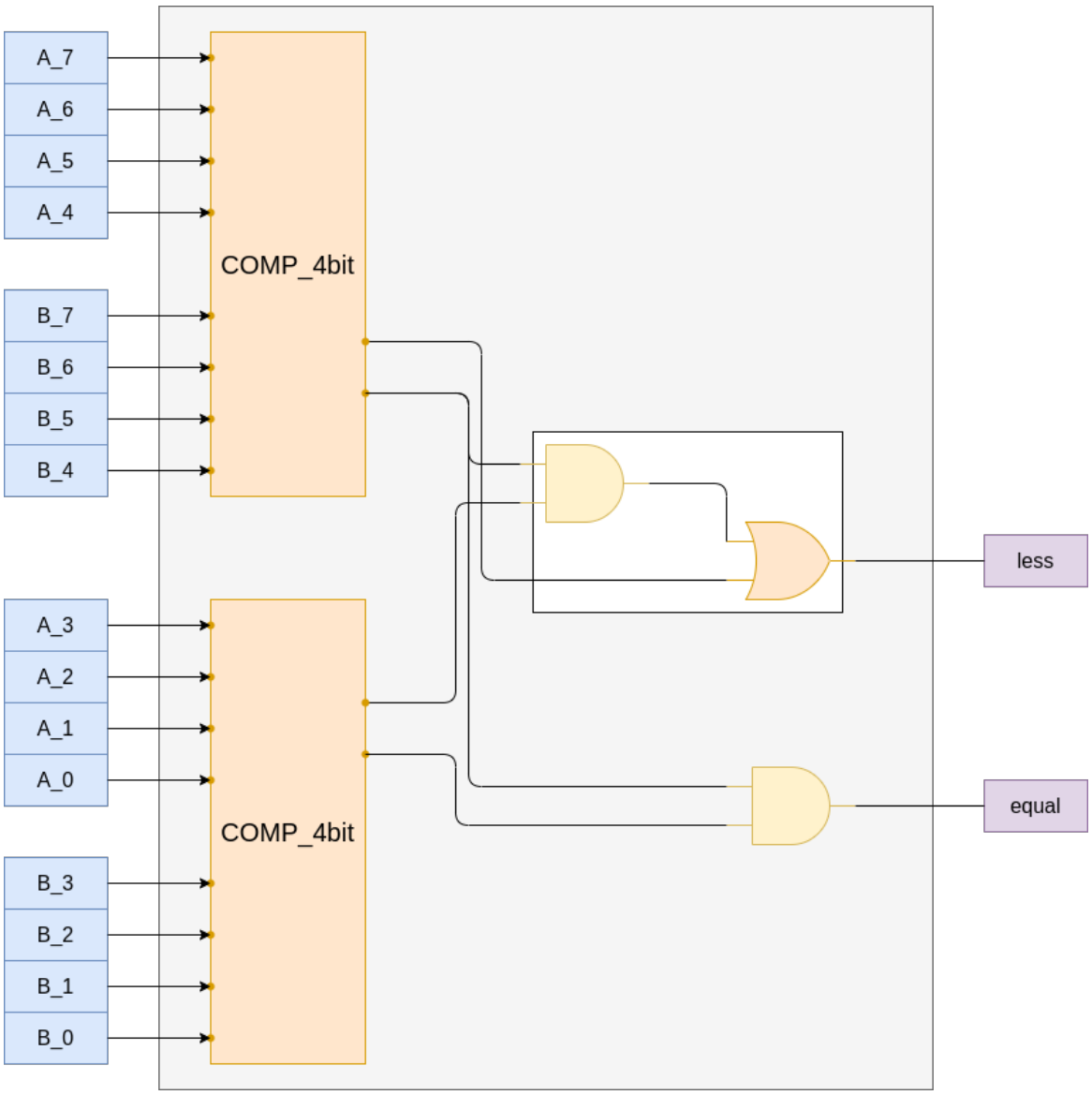
Sử dụng bộ lan truyền dấu so sánh trên, với nếu các bit trên có có xuất hiện bit less thì ngõ ra lan truyền bit less, còn nếu bit thấp thì nếu tầng trên bằng nhau và các tầng thấp có bit less ngõ ra lan truyền bit less.



Hình 6: Sơ đồ logic của bộ Comparator 4-bit.

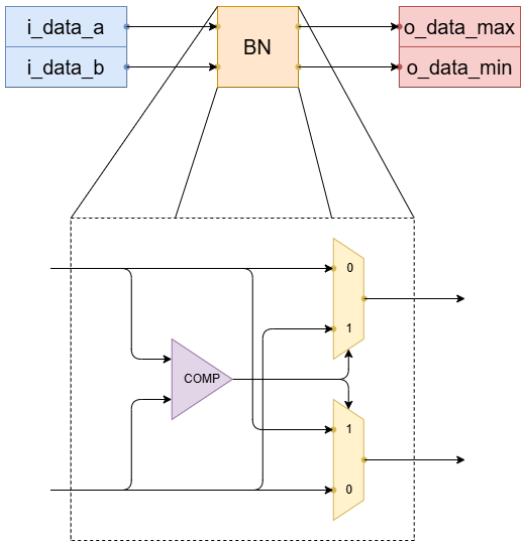
+ Triển khai bộ so sánh 8-bit

  Tương tự với bộ 4-bit, thì 8-bit được triển khai với 2 bộ 4-bit.



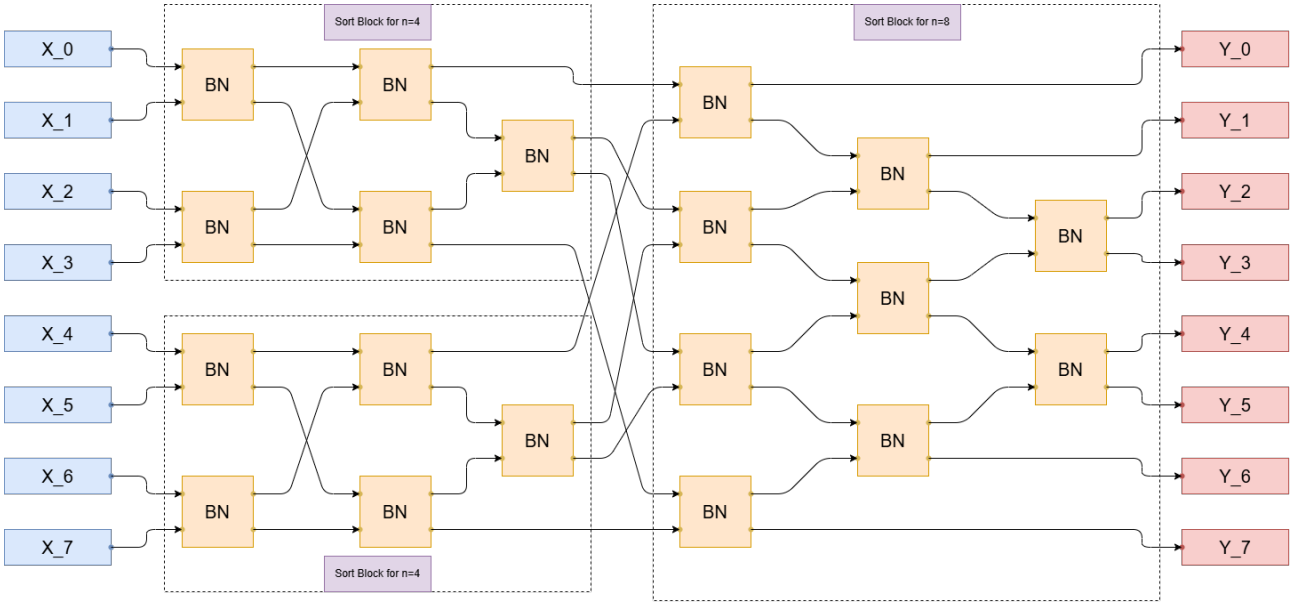
Hình 7: Sơ đồ logic của bộ Comparator 8-bit.

- **Bộ Swap 8bit**



Hình 8: Bộ so sánh và swap giá trị trong Bitonic Sort.

- **Bitonic Sort**



Hình 9: Bộ sắp xếp Bitonic Sort.

b) Viết chương trình HDL mô tả mạch đã cho.

```
module COMP_2bit(
    input  logic [1:0] i_data_a,
    input  logic [1:0] i_data_b,
    output logic       o_less,
    output logic       o_equal
```

```
);
    // assign o_less  = ((~i_data_a[1] & ~i_data_a[0]) & i_data_b[0]) | (~i_data_a[0] & (i_data_b[1] &
     i_data_b[0])) | (~i_data_a[1] & i_data_b[1]);
    // assign o_equal = ~|(i_data_a ^ i_data_b);
    assign o_less  = ((~(i_data_a[1] | i_data_a[0])) & i_data_b[0]) | (~i_data_a[0] & (i_data_b[1] &
     i_data_b[0])) | (~i_data_a[1] & i_data_b[1]);
    assign o_equal = ~|(i_data_a ^ i_data_b);
endmodule
```

Listing 11: Chương trình mô tả bộ so sánh 2-bit.

```
module COMP_4bit(
    input  logic [3:0] i_data_a,
    input  logic [3:0] i_data_b,
    output logic       o_less,
    output logic       o_equal
);
    logic w_less_low, w_equal_low;
    logic w_less_high, w_equal_high;

    COMP_2bit u_low (
        .i_data_a (i_data_a[1:0]),
        .i_data_b (i_data_b[1:0]),
        .o_less   (w_less_low),
        .o_equal  (w_equal_low)
    );

    COMP_2bit u_high (
        .i_data_a (i_data_a[3:2]),
        .i_data_b (i_data_b[3:2]),
        .o_less   (w_less_high),
        .o_equal  (w_equal_high)
    );

    assign o_less  = w_less_high | (w_equal_high & w_less_low);
    assign o_equal = w_equal_high & w_equal_low;
endmodule
```

Listing 12: Chương trình mô tả bộ so sánh 4-bit.

```
module COMP_less #(
    parameter SIZE_DATA = 8
)(
    input  logic [SIZE_DATA-1:0] i_data_a,
    input  logic [SIZE_DATA-1:0] i_data_b,
    output logic                 o_less
);
    logic w_less_low, w_equal_low;
    logic w_less_high, w_equal_high;

    COMP_4bit u_low (
        .i_data_a (i_data_a[3:0]),
        .i_data_b (i_data_b[3:0]),
        .o_less   (w_less_low),
        .o_equal  (w_equal_low)
    );

    COMP_4bit u_high (
        .i_data_a (i_data_a[7:4]),
        .i_data_b (i_data_b[7:4]),
        .o_less   (w_less_high),
        .o_equal  (w_equal_high)
```

```
    );

    assign o_less = w_less_high | (w_equal_high & w_less_low);
endmodule
```

Listing 13: Chương trình mô tả bộ so sánh 8-bit.

```
module Compare_and_Swap_unit #(
    parameter IS_ASC        = 1 ,
    parameter SIZE_DATA     = 8
)(
    input logic  [SIZE_DATA-1:0]    i_data_a    ,
    input logic  [SIZE_DATA-1:0]    i_data_b    ,
    output logic [SIZE_DATA-1:0]    o_data_max  ,
    output logic [SIZE_DATA-1:0]    o_data_min
);

logic w_compare;
COMP_less #(
    .SIZE_DATA  (SIZE_DATA)
) COMP_UNIT (
    // o_less = i_data_a < i_data_b
    .i_data_a       (i_data_a),
    .i_data_b       (i_data_b),
    .o_less         (w_compare)
);
// COMP_parallel_prefix_binary #(
//     .SIZE_DATA  (SIZE_DATA)
// ) COMP_UNT (
//     .i_data_a       (i_data_a),
//     .i_data_b       (i_data_b),
//     .o_less         (w_compare)
// );

generate
    if(IS_ASC) begin
        assign o_data_max = w_compare ? i_data_b : i_data_a;
        assign o_data_min = w_compare ? i_data_a : i_data_b;
    end else begin
        assign o_data_max = w_compare ? i_data_a : i_data_b;
        assign o_data_min = w_compare ? i_data_b : i_data_a;
    end
endgenerate

endmodule
```

Listing 14: Chương trình mô tả một đơn vị của bộ Bitonic Sort.

```
module Bitonic_Block4 #(
    parameter IS_ASC        = 1 ,
    parameter SIZE_DATA     = 8
)(
    input logic [SIZE_DATA-1:0]     i_data_0    ,
    input logic [SIZE_DATA-1:0]     i_data_1    ,
    input logic [SIZE_DATA-1:0]     i_data_2    ,
    input logic [SIZE_DATA-1:0]     i_data_3    ,
    output logic [SIZE_DATA-1:0]    o_data_0    ,
    output logic [SIZE_DATA-1:0]    o_data_1    ,
    output logic [SIZE_DATA-1:0]    o_data_2    ,
    output logic [SIZE_DATA-1:0]    o_data_3
);

```

```
15 ////////////////////////////////////////////////////
16 // Internal Logics
17 ////////////////////////////////////////////////////
18 wire [SIZE_DATA-1:0] w_data_max_0_0, w_data_max_0_1;
19 wire [SIZE_DATA-1:0] w_data_min_0_0, w_data_min_0_1;
20 wire [SIZE_DATA-1:0] w_data_max_1_0, w_data_max_1_1;
21 wire [SIZE_DATA-1:0] w_data_min_1_0, w_data_min_1_1;
22 wire [SIZE_DATA-1:0] w_data_max_2_0, w_data_min_2_0;
23
24 ////////////////////////////////////////////////////
25 // SubModules
26 ////////////////////////////////////////////////////
27
28 Compare_and_Swap_unit #(
29     .IS_ASC         (IS_ASC),
30     .SIZE_DATA      (SIZE_DATA)
31 ) CAS_0_0 (
32     .i_data_a       (i_data_0),
33     .i_data_b       (i_data_1),
34     .o_data_max     (w_data_max_0_0),
35     .o_data_min     (w_data_min_0_0)
36 );
37
38 Compare_and_Swap_unit #(
39     .IS_ASC         (IS_ASC),
40     .SIZE_DATA      (SIZE_DATA)
41 ) CAS_0_1 (
42     .i_data_a       (i_data_2),
43     .i_data_b       (i_data_3),
44     .o_data_max     (w_data_max_0_1),
45     .o_data_min     (w_data_min_0_1)
46 );
47
48 Compare_and_Swap_unit #(
49     .IS_ASC         (IS_ASC),
50     .SIZE_DATA      (SIZE_DATA)
51 ) CAS_1_0 (
52     .i_data_a       (w_data_max_0_0),
53     .i_data_b       (w_data_max_0_1),
54     .o_data_max     (w_data_max_1_0),
55     .o_data_min     (w_data_min_1_0)
56 );
57
58 Compare_and_Swap_unit #(
59     .IS_ASC         (IS_ASC),
60     .SIZE_DATA      (SIZE_DATA)
61 ) CAS_1_1 (
62     .i_data_a       (w_data_min_0_0),
63     .i_data_b       (w_data_min_0_1),
64     .o_data_max     (w_data_max_1_1),
65     .o_data_min     (w_data_min_1_1)
66 );
67
68 Compare_and_Swap_unit #(
69     .IS_ASC         (IS_ASC),
70     .SIZE_DATA      (SIZE_DATA)
71 ) CAS_2_0 (
72     .i_data_a       (w_data_min_1_0),
73     .i_data_b       (w_data_max_1_1),
74     .o_data_max     (w_data_max_2_0),
75     .o_data_min     (w_data_min_2_0)
76 );
77
78 assign o_data_0 = w_data_max_1_0;
```

```
assign o_data_1 = w_data_max_2_0;
assign o_data_2 = w_data_min_2_0;
assign o_data_3 = w_data_min_1_1;

endmodule
```

Listing 15: Chương trình mô tả bộ Bitonic Sort Block-4.

```
module Bitonic_Block8 #(
    parameter IS_ASC        = 1 ,
    parameter SIZE_DATA     = 8
)(
    input logic [SIZE_DATA-1:0]     i_data_0    ,
    input logic [SIZE_DATA-1:0]     i_data_1    ,
    input logic [SIZE_DATA-1:0]     i_data_2    ,
    input logic [SIZE_DATA-1:0]     i_data_3    ,
    input logic [SIZE_DATA-1:0]     i_data_4    ,
    input logic [SIZE_DATA-1:0]     i_data_5    ,
    input logic [SIZE_DATA-1:0]     i_data_6    ,
    input logic [SIZE_DATA-1:0]     i_data_7    ,
    output logic [SIZE_DATA-1:0]    o_data_0    ,
    output logic [SIZE_DATA-1:0]    o_data_1    ,
    output logic [SIZE_DATA-1:0]    o_data_2    ,
    output logic [SIZE_DATA-1:0]    o_data_3    ,
    output logic [SIZE_DATA-1:0]    o_data_4    ,
    output logic [SIZE_DATA-1:0]    o_data_5    ,
    output logic [SIZE_DATA-1:0]    o_data_6    ,
    output logic [SIZE_DATA-1:0]    o_data_7
);

//////////////////////////////////////////////////////
// Internal Logics
//////////////////////////////////////////////////////
wire [SIZE_DATA-1:0] w_data_max_0_0, w_data_max_0_1, w_data_max_0_2, w_data_max_0_3;
wire [SIZE_DATA-1:0] w_data_min_0_0, w_data_min_0_1, w_data_min_0_2, w_data_min_0_3;
wire [SIZE_DATA-1:0] w_data_max_1_0, w_data_max_1_1, w_data_max_1_2;
wire [SIZE_DATA-1:0] w_data_min_1_0, w_data_min_1_1, w_data_min_1_2;
wire [SIZE_DATA-1:0] w_data_max_2_0, w_data_max_2_1;
wire [SIZE_DATA-1:0] w_data_min_2_0, w_data_min_2_1;

//////////////////////////////////////////////////////
// SubModules
//////////////////////////////////////////////////////

Compare_and_Swap_unit #(
    .IS_ASC         (IS_ASC),
    .SIZE_DATA      (SIZE_DATA)
) CAS_0_0 (
    .i_data_a       (i_data_0),
    .i_data_b       (i_data_1),
    .o_data_max     (w_data_max_0_0),
    .o_data_min     (w_data_min_0_0)
);
Compare_and_Swap_unit #(
    .IS_ASC         (IS_ASC),
    .SIZE_DATA      (SIZE_DATA)
) CAS_0_1 (
    .i_data_a       (i_data_2),
    .i_data_b       (i_data_3),
    .o_data_max     (w_data_max_0_1),
    .o_data_min     (w_data_min_0_1)
);
Compare_and_Swap_unit #(
```

```
    .IS_ASC         (IS_ASC),
    .SIZE_DATA      (SIZE_DATA)
) CAS_0_2 (
    .i_data_a       (i_data_4),
    .i_data_b       (i_data_5),
    .o_data_max     (w_data_max_0_2),
    .o_data_min     (w_data_min_0_2)
);
Compare_and_Swap_unit #(
    .IS_ASC         (IS_ASC),
    .SIZE_DATA      (SIZE_DATA)
) CAS_0_3 (
    .i_data_a       (i_data_6),
    .i_data_b       (i_data_7),
    .o_data_max     (w_data_max_0_3),
    .o_data_min     (w_data_min_0_3)
);

Compare_and_Swap_unit #(
    .IS_ASC         (IS_ASC),
    .SIZE_DATA      (SIZE_DATA)
) CAS_1_0 (
    .i_data_a       (w_data_min_0_0),
    .i_data_b       (w_data_max_0_1),
    .o_data_max     (w_data_max_1_0),
    .o_data_min     (w_data_min_1_0)
);
Compare_and_Swap_unit #(
    .IS_ASC         (IS_ASC),
    .SIZE_DATA      (SIZE_DATA)
) CAS_1_1 (
    .i_data_a       (w_data_min_0_1),
    .i_data_b       (w_data_max_0_2),
    .o_data_max     (w_data_max_1_1),
    .o_data_min     (w_data_min_1_1)
);
Compare_and_Swap_unit #(
    .IS_ASC         (IS_ASC),
    .SIZE_DATA      (SIZE_DATA)
) CAS_1_2 (
    .i_data_a       (w_data_min_0_2),
    .i_data_b       (w_data_max_0_3),
    .o_data_max     (w_data_max_1_2),
    .o_data_min     (w_data_min_1_2)
);

Compare_and_Swap_unit #(
    .IS_ASC         (IS_ASC),
    .SIZE_DATA      (SIZE_DATA)
) CAS_2_0 (
    .i_data_a       (w_data_min_1_0),
    .i_data_b       (w_data_max_1_1),
    .o_data_max     (w_data_max_2_0),
    .o_data_min     (w_data_min_2_0)
);
Compare_and_Swap_unit #(
    .IS_ASC         (IS_ASC),
    .SIZE_DATA      (SIZE_DATA)
) CAS_2_1 (
    .i_data_a       (w_data_min_1_1),
    .i_data_b       (w_data_max_1_2),
    .o_data_max     (w_data_max_2_1),
    .o_data_min     (w_data_min_2_1)
);
```

```
120
121 assign o_data_0 = w_data_max_0_0;
122 assign o_data_1 = w_data_max_1_0;
123 assign o_data_2 = w_data_max_2_0;
124 assign o_data_3 = w_data_min_2_0;
125 assign o_data_4 = w_data_max_2_1;
126 assign o_data_5 = w_data_min_2_1;
127 assign o_data_6 = w_data_min_1_2;
128 assign o_data_7 = w_data_min_0_3;
129 endmodule
```

Listing 16: Chương trình mô tả bộ Bitonic Sort Block-8.

```
1  module Bitonic_Sort #(
2      parameter IS_ASC    = 1,
3      parameter NUM_ELEM  = 8,
4      parameter SIZE_DATA = 8
5  )(
6      input  logic [(NUM_ELEM*SIZE_DATA)-1:0] i_data  ,
7      output logic [(NUM_ELEM*SIZE_DATA)-1:0] o_sorted
8  );
9
10     // Split flat bus into array
11     wire [SIZE_DATA-1:0] w_i_data [0:NUM_ELEM-1];
12     wire [SIZE_DATA-1:0] w_0_data [0:NUM_ELEM-1];
13     wire [SIZE_DATA-1:0] w_sorted [0:NUM_ELEM-1];
14
15     genvar i;
16     generate
17         for (i = 0; i < NUM_ELEM; i++) begin : UNPACK_INPUT
18             assign w_i_data[i] = i_data[i*SIZE_DATA +: SIZE_DATA];
19         end
20     endgenerate
21
22     Bitonic_Block4 #(
23         .IS_ASC         (IS_ASC),
24         .SIZE_DATA      (SIZE_DATA)
25     ) BN_4_UNIT_0 (
26         .i_data_0       (w_i_data[0]),
27         .i_data_1       (w_i_data[1]),
28         .i_data_2       (w_i_data[2]),
29         .i_data_3       (w_i_data[3]),
30         .o_data_0       (w_0_data[0]),
31         .o_data_1       (w_0_data[1]),
32         .o_data_2       (w_0_data[2]),
33         .o_data_3       (w_0_data[3])
34     );
35     Bitonic_Block4 #(
36         .IS_ASC         (IS_ASC),
37         .SIZE_DATA      (SIZE_DATA)
38     ) BN_4_UNIT_1 (
39         .i_data_0       (w_i_data[4]),
40         .i_data_1       (w_i_data[5]),
41         .i_data_2       (w_i_data[6]),
42         .i_data_3       (w_i_data[7]),
43         .o_data_0       (w_0_data[4]),
44         .o_data_1       (w_0_data[5]),
45         .o_data_2       (w_0_data[6]),
46         .o_data_3       (w_0_data[7])
47     );
48
49     Bitonic_Block8 #(
50         .IS_ASC         (IS_ASC),
```

```
        .SIZE_DATA      (SIZE_DATA)
    ) BN_8_UNIT_0 (
        .i_data_0       (w_0_data[0]),
        .i_data_1       (w_0_data[4]),
        .i_data_2       (w_0_data[1]),
        .i_data_3       (w_0_data[5]),
        .i_data_4       (w_0_data[2]),
        .i_data_5       (w_0_data[6]),
        .i_data_6       (w_0_data[3]),
        .i_data_7       (w_0_data[7]),
        .o_data_0       (w_sorted[0]),
        .o_data_1       (w_sorted[1]),
        .o_data_2       (w_sorted[2]),
        .o_data_3       (w_sorted[3]),
        .o_data_4       (w_sorted[4]),
        .o_data_5       (w_sorted[5]),
        .o_data_6       (w_sorted[6]),
        .o_data_7       (w_sorted[7])
    );

    generate
        for (i = 0; i < NUM_ELEM; i++) begin : PACK_OUTPUT
            assign o_sorted[i*SIZE_DATA +: SIZE_DATA] = w_sorted[i];
        end
    endgenerate

endmodule
```

Listing 17: Chương trình mô tả bộ Bitonic Sort 8 phần tử đầu vào.

c) Viết chương trình testbench cho mạch.

Đầu tiên nhóm em thực hiện triển khai chứng minh giải thuật kết quả đúng của bài toán sau:

```
   function automatic logic [SIZE_DATA*NUM_ELEM-1:0] f_ARR_sorted(
      int                                f_is_acs   ,
      input logic [SIZE_DATA*NUM_ELEM-1:0] f_i_data
   );
      logic [SIZE_DATA-1:0] arr   [0:NUM_ELEM-1];
      logic [SIZE_DATA-1:0] temp;
      int i, j;

      begin
         for (i = 0; i < NUM_ELEM; i++) begin
            arr[i] = f_i_data[i*SIZE_DATA +: SIZE_DATA];
         end

         for (i = 0; i < NUM_ELEM-1; i++) begin
            for (j = 0; j < NUM_ELEM-1-i; j++) begin
               if(f_is_acs) begin
                  if (arr[j] > arr[j+1]) begin
                     temp     = arr[j];
                     arr[j]   = arr[j+1];
                     arr[j+1] = temp;
                  end
               end else begin
                  if (arr[j] < arr[j+1]) begin
                     temp     = arr[j];
```

```
                    arr[j]   = arr[j+1];
                    arr[j+1] = temp;
                end
            end
        end
    end

    for (i = 0; i < NUM_ELEM; i++) begin
        f_ARR_sorted[i*SIZE_DATA +: SIZE_DATA] = arr[i];
    end
end
endfunction
```

Listing 18: Giải thuật chứng minh bộ Bitonic Sort 8 phần tử.

Nhóm em sử dụng giải thuật của Bubble Sort gần giống với giải thuật parallel của Bitonic Sort.