

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

—oOo—



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

Câu 2	1
a)	1
b)	4
c)	8
Câu 6	9
a)	10
b)	14
c)	21

Danh sách hình vẽ

1	Sơ đồ logic của bộ LOPD 4bit.	1
2	Sơ đồ logic của bộ LOPD 8bit.	2
3	Sơ đồ logic của bộ LOPD 16bit.	3
4	Sơ đồ logic của bộ LOPD 24-bit.	4
5	Sơ đồ logic của bộ Comparator 2-bit.	11
6	Sơ đồ logic của bộ Comparator 4-bit.	12
7	Sơ đồ logic của bộ Comparator 8-bit.	13
8	Bộ so sánh và swap giá trị trong Bitonic Sort.	14
9	Bộ sắp xếp Bitonic Sort.	14

Danh sách bảng

1	Bảng sự thật của bộ phát hiện bit 1 (Leading one position) cho 4 bit.	1
2	Bảng sự thật cho bộ so sánh 2-bit $A < B$	10

List of Listings

1	Chương trình mô tả LOPD 4-bit.	5
2	Chương trình mô tả LOPD 8-bit.	5
3	Chương trình mô tả LOPD 16-bit.	6
4	Chương trình mô tả LOPD 24-bit.	7
5	Giải thuật chứng minh kết quả của bộ LOPD 24-bit.	8
6	Chương trình mô tả bộ so sánh 2-bit.	14
7	Chương trình mô tả bộ so sánh 4-bit.	15
8	Chương trình mô tả bộ so sánh 8-bit.	15
9	Chương trình mô tả một đơn vị của bộ Bitonic Sort.	16
10	Chương trình mô tả bộ Bitonic Sort Block-4.	16
11	Chương trình mô tả bộ Bitonic Sort Block-8.	18
12	Chương trình mô tả bộ Bitonic Sort 8 phần tử đầu vào.	20
13	Giải thuật chứng minh bộ Bitonic Sort 8 phần tử.	21

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.

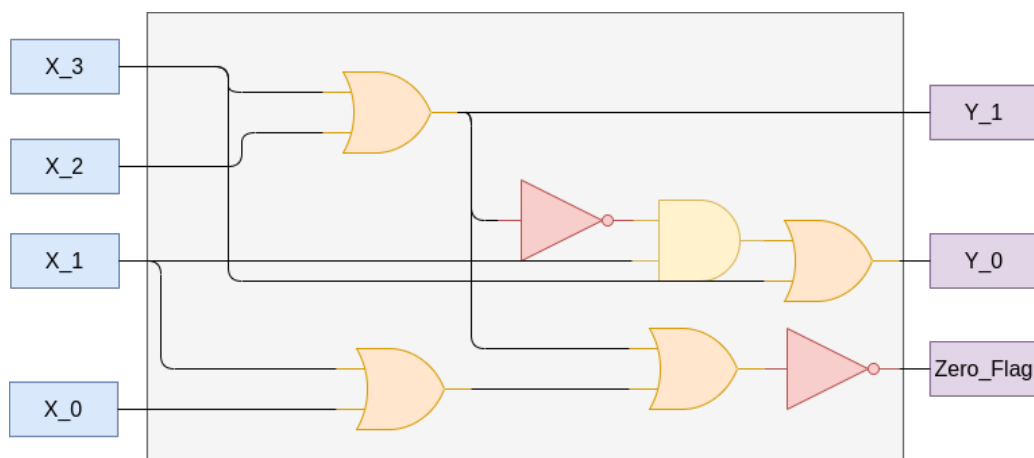
a) Thiết kế mạch chỉ được dùng các standard cell trên.

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

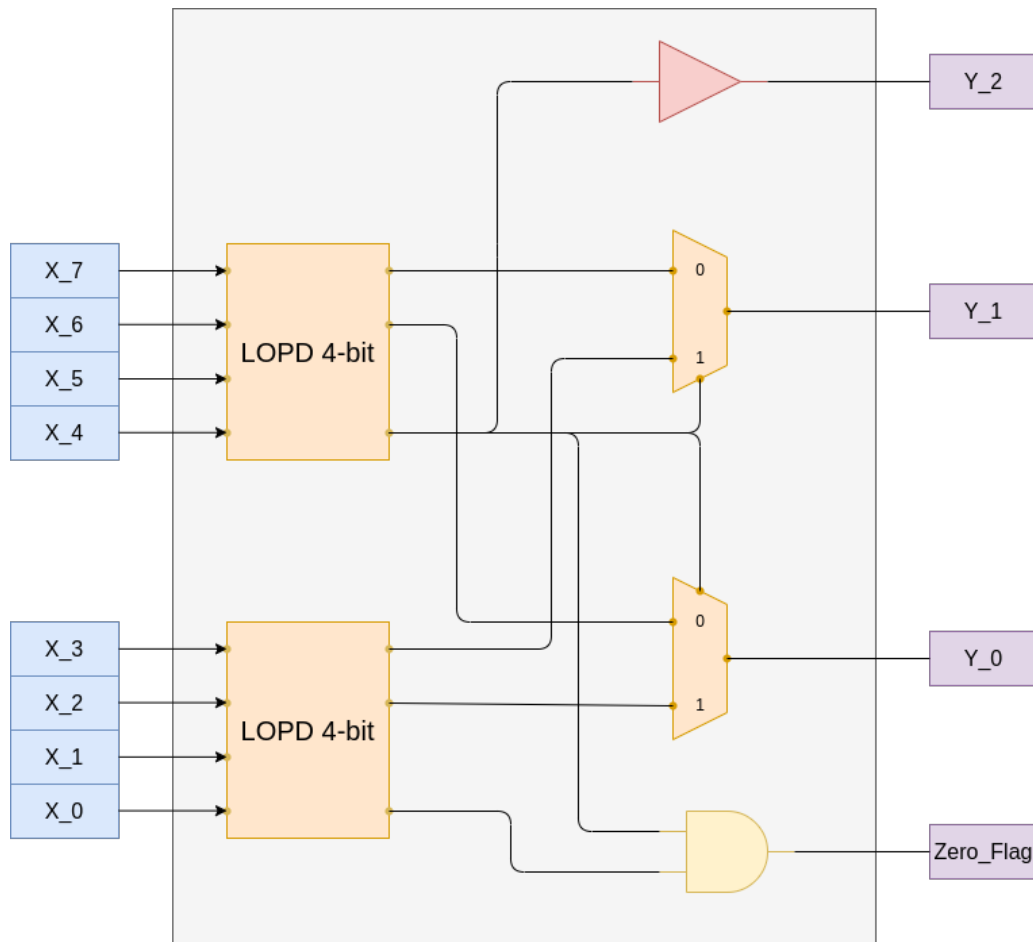
Bảng 1: Bảng sự thật của bộ phát hiện bit 1 (Leading one position) cho 4 bit.

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

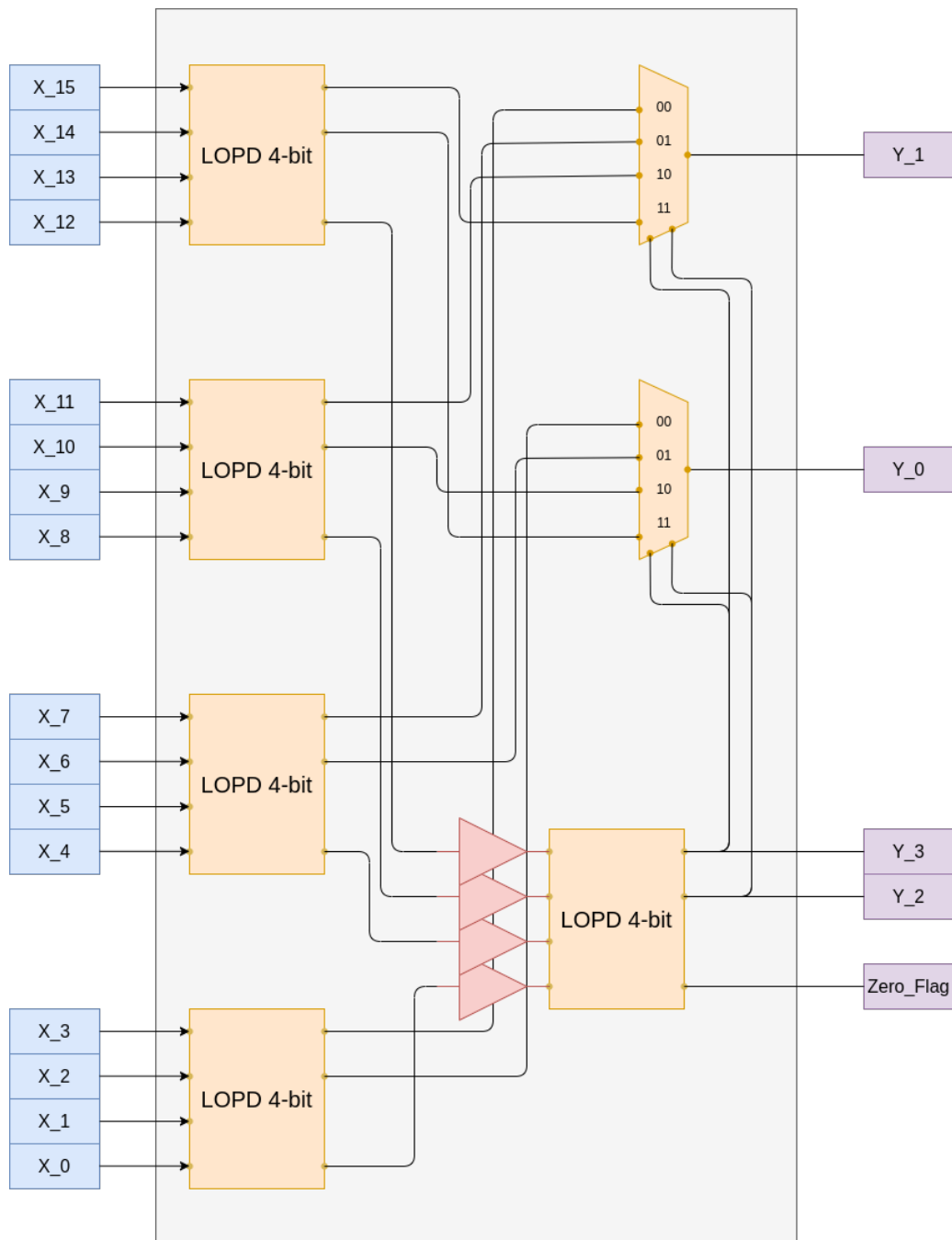


Hình 1: Sơ đồ logic của bộ LOPD 4bit.

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

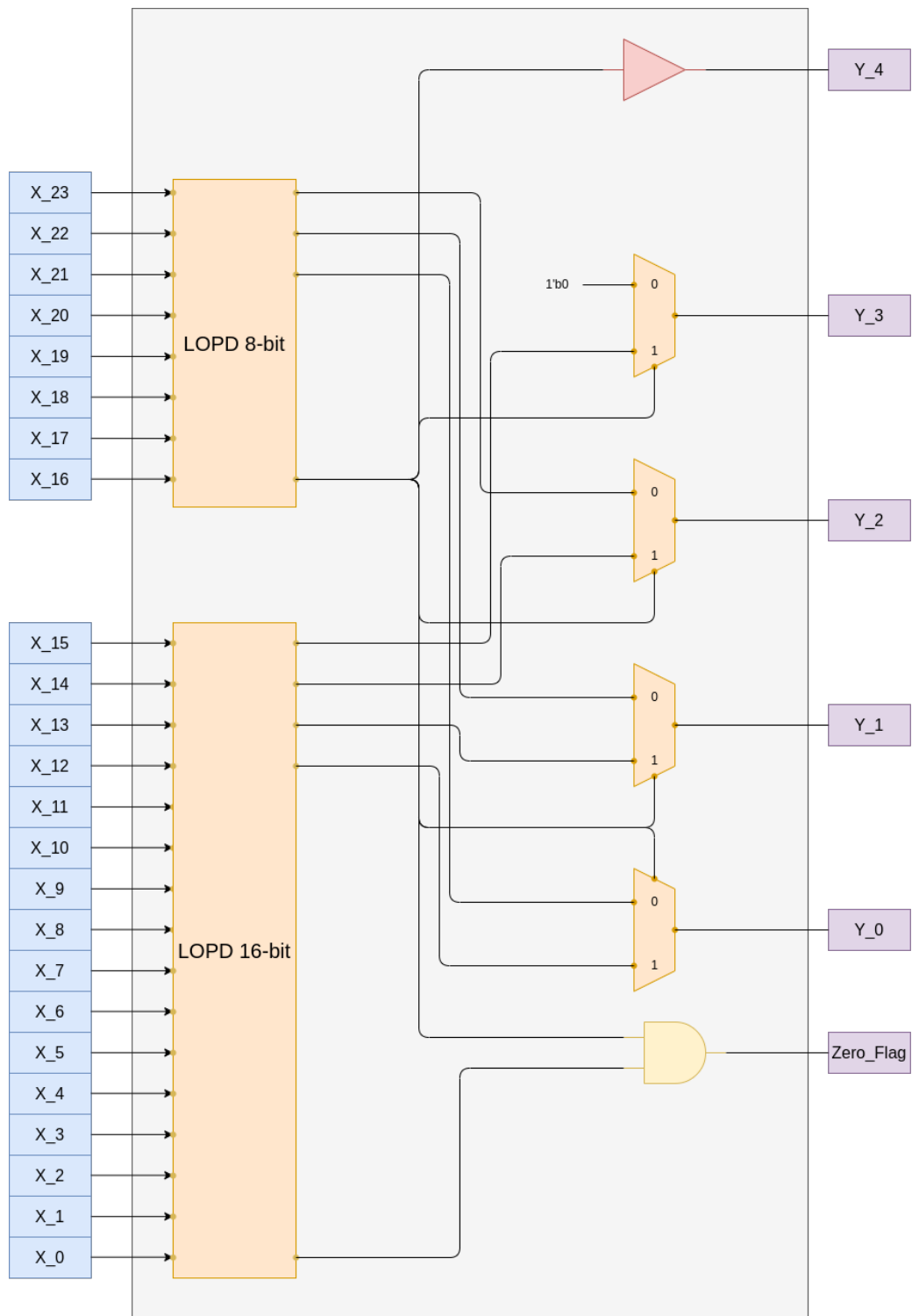


Hình 2: Sơ đồ logic của bộ LOPD 8bit.



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 \rightarrow 16) và bộ LOPD 16-bit vào 16-bit thấp (từ 15 \rightarrow 0).



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

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

```

1  module LOPD_4bit(
2      input logic [3:0]      i_data ,
3      output logic [1:0]     o_pos_one,
4      output logic          o_zero_flag
5  );
6
7      //////////////////////////////////////////////////// LDO ////////////////////////////////////////////////////
8      // D[3] | D[2] | D[1] | D[0] | P0[3] | P0[2] | P0[1] | P0[0] | ZERO_FLAG |
9      // 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
10     // 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
11     // 0 | 0 | 1 | X | 0 | 0 | 1 | 0 | 0 |
12     // 0 | 1 | X | X | 0 | 1 | 0 | 0 | 0 |
13     // 1 | X | X | X | 1 | 0 | 0 | 0 | 0 |
14     //////////////////////////////////////////////////// LDOD ////////////////////////////////////////////////////
15     // D[3] | D[2] | D[1] | D[0] | P0[1] | P0[0] | ZERO_FLAG |
16     // 0 | 0 | 0 | 0 | 0 | 0 | 1 |
17     // 0 | 0 | 0 | 1 | 0 | 0 | 0 |
18     // 0 | 0 | 1 | X | 0 | 1 | 0 |
19     // 0 | 1 | X | X | 1 | 0 | 0 |
20     // 1 | X | X | X | 1 | 1 | 0 |
21
22     assign o_zero_flag = ~((o_pos_one[1] | (i_data[1] | i_data[0])));
23     assign o_pos_one[1] = i_data[3] | i_data[2];
24     assign o_pos_one[0] = ((~(i_data[3] | i_data[2])) & (i_data[1])) | (i_data[3]);
25
26 endmodule

```

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

```

1  module LOPD_8bit(
2      input logic [7:0]      i_data ,
3      output logic [2:0]     o_pos_one,
4      output logic          o_zero_flag
5  );
6
7      ////////////////////////////////////////////////////
8      // LOPD_4bit_unit_0
9      ////////////////////////////////////////////////////
10     logic w_zero_flag_0;
11     logic [1:0] w_pos_one_0;
12     LOPD_4bit LOPD_4bit_unit_0 (
13         .i_data      (i_data[3:0]),
14         .o_pos_one    (w_pos_one_0),
15         .o_zero_flag  (w_zero_flag_0)
16     );
17
18     ////////////////////////////////////////////////////
19     // LOPD_4bit_unit_1
20     ////////////////////////////////////////////////////
21     logic w_zero_flag_1;
22     logic [1:0] w_pos_one_1;
23     LOPD_4bit LOPD_4bit_unit_1 (
24         .i_data      (i_data[7:4]),
25         .o_pos_one    (w_pos_one_1),
26         .o_zero_flag  (w_zero_flag_1)
27     );
28
29     ////////////////////////////////////////////////////
30     // LOD_8bit_unit
31     ////////////////////////////////////////////////////
32     assign o_zero_flag = w_zero_flag_0 & w_zero_flag_1;
33     assign o_pos_one[2] = ~(w_zero_flag_1);

```



```

34 assign o_pos_one[1] = (w_zero_flag_1) ? w_pos_one_0[1] : w_pos_one_1[1];
35 assign o_pos_one[0] = (w_zero_flag_1) ? w_pos_one_0[0] : w_pos_one_1[0];
36
37 endmodule

```

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

```

1  module LOPD_16bit(
2      input logic [15:0]    i_data ,
3      output logic [3:0]    o_pos_one,
4      output logic         o_zero_flag
5  );
6
7
8  // ////////////////////////////////////////
9  // // LOPD_4bit_unit
10 // ////////////////////////////////////////
11 logic [1:0] w_one_position_0_0;
12 logic      w_zero_flag_0_0;
13 LOPD_4bit LOPD4BIT_0_0 (
14     .i_data      (i_data[3:0]),
15     .o_pos_one   (w_one_position_0_0),
16     .o_zero_flag (w_zero_flag_0_0)
17 );
18 logic [1:0] w_one_position_0_1;
19 logic      w_zero_flag_0_1;
20 LOPD_4bit LOPD4BIT_0_1 (
21     .i_data      (i_data[7:4]),
22     .o_pos_one   (w_one_position_0_1),
23     .o_zero_flag (w_zero_flag_0_1)
24 );
25 logic [1:0] w_one_position_0_2;
26 logic      w_zero_flag_0_2;
27 LOPD_4bit LOPD4BIT_0_2 (
28     .i_data      (i_data[11:8]),
29     .o_pos_one   (w_one_position_0_2),
30     .o_zero_flag (w_zero_flag_0_2)
31 );
32 logic [1:0] w_one_position_0_3;
33 logic      w_zero_flag_0_3;
34 LOPD_4bit LOPD4BIT_0_3 (
35     .i_data      (i_data[15:12]),
36     .o_pos_one   (w_one_position_0_3),
37     .o_zero_flag (w_zero_flag_0_3)
38 );
39
40 logic [1:0] w_one_position_1_0;
41 LOPD_4bit LOPD4BIT_1_0 (
42     .i_data      ({~w_zero_flag_0_3, ~w_zero_flag_0_2, ~w_zero_flag_0_1, ~w_zero_flag_0_2}),
43     .o_pos_one   (w_one_position_1_0),
44     .o_zero_flag (o_zero_flag)
45 );
46
47 // ////////////////////////////////////////
48 // // LOD_16bit_unit
49 // ////////////////////////////////////////
50 MUX_4_1 #(
51     .SIZE_DATA (1)
52 ) MUX_4_1_0(
53     .i_data_0      (w_one_position_0_0[0]),
54     .i_data_1      (w_one_position_0_1[0]),
55     .i_data_2      (w_one_position_0_2[0]),
56     .i_data_3      (w_one_position_0_3[0]),

```

```

57     .i_select      ({o_pos_one[3], o_pos_one[2]}),
58     .o_data        (o_pos_one[0])
59 );
60 MUX_4_1 #(
61     .SIZE_DATA     (1)
62 ) MUX_4_1_1(
63     .i_data_0       (w_one_position_0_0[1]),
64     .i_data_1       (w_one_position_0_1[1]),
65     .i_data_2       (w_one_position_0_2[1]),
66     .i_data_3       (w_one_position_0_3[1]),
67     .i_select       ({o_pos_one[3], o_pos_one[2]}),
68     .o_data         (o_pos_one[1])
69 );
70
71 assign o_pos_one[2] = w_one_position_1_0[0];
72 assign o_pos_one[3] = w_one_position_1_0[1];
73
74 endmodule
75
76 module MUX_4_1 #(
77     parameter SIZE_DATA = 1
78 )(
79     input logic [SIZE_DATA-1:0] i_data_0 ,
80     input logic [SIZE_DATA-1:0] i_data_1 ,
81     input logic [SIZE_DATA-1:0] i_data_2 ,
82     input logic [SIZE_DATA-1:0] i_data_3 ,
83     input logic [1:0] i_select ,
84     output logic [SIZE_DATA-1:0] o_data
85 );
86 reg [SIZE_DATA-1:0] w_o_mux;
87 always_comb begin : MUX_4_1_1
88     case (i_select)
89         2'b00:
90             w_o_mux = i_data_0;
91         2'b01:
92             w_o_mux = i_data_1;
93         2'b10:
94             w_o_mux = i_data_2;
95         2'b11:
96             w_o_mux = i_data_3;
97         default:
98             w_o_mux = '0;
99     endcase
100 end
101 assign o_data = w_o_mux;
102
103 endmodule

```

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

```

1  module Question2 #(
2      parameter SIZE_DATA    = 24 ,
3      parameter SIZE_LOPD    = 5
4  )(
5      input logic [SIZE_DATA-1:0] i_data ,
6      output logic [SIZE_LOPD-1:0] o_one_position ,
7      output logic o_zero_flag
8  );
9
10 logic [15:0] LOPD16_i_data;
11 assign LOPD16_i_data = i_data[15:0];
12 logic [3:0] LOPD16_o_pos_one;
13 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:

```

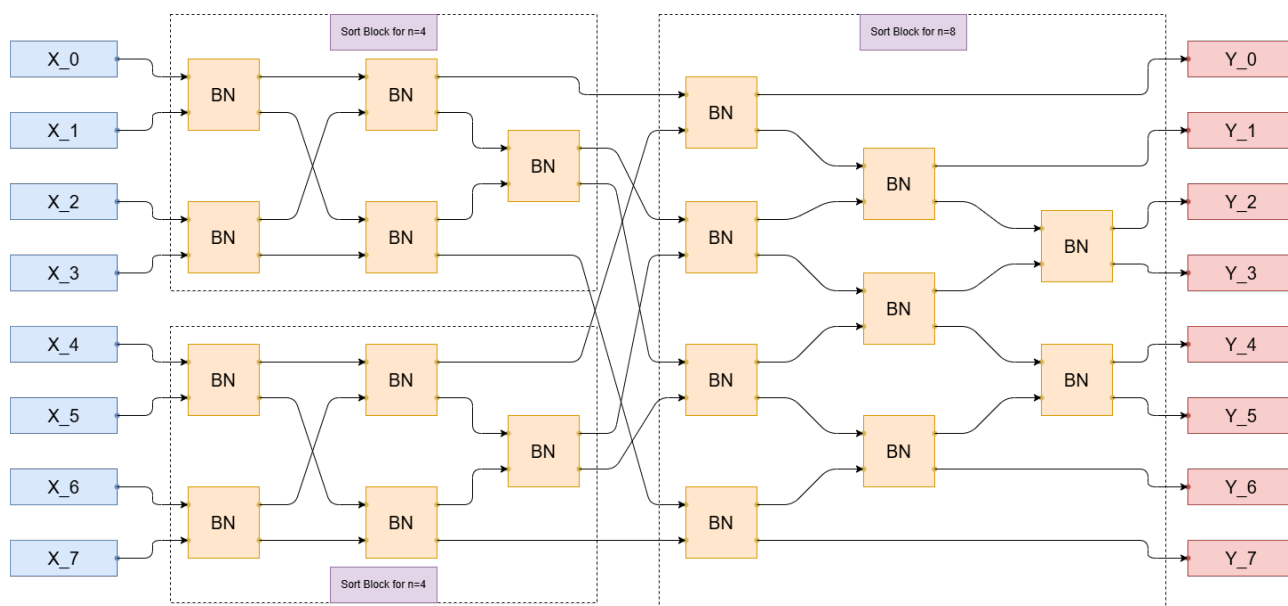
1 function automatic logic [SIZE_LOP-1:0] Test_LOPD(
2     input logic [SIZE_DATA-1:0] f_i_data
3 );
4 logic [SIZE_DATA-1:0] t_temp;
5 int cnt_position_1;
6 begin
7     t_temp = f_i_data;
8     cnt_position_1 = 0;
9
10    if(t_temp == 0) begin
11        Test_LOPD = 0;
12    end else begin
13        while (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

```

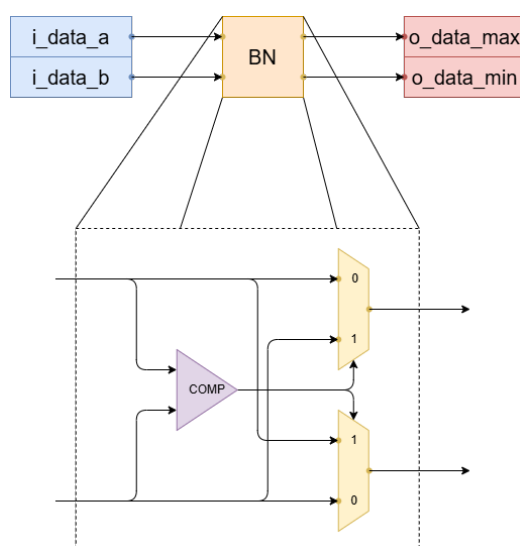
Listing 5: Giải thuật chứng minh kết quả của bộ LOPD 24-bit.

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, \dots, x_8 , và cho ngõ ra là y_1, y_2, \dots, 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.

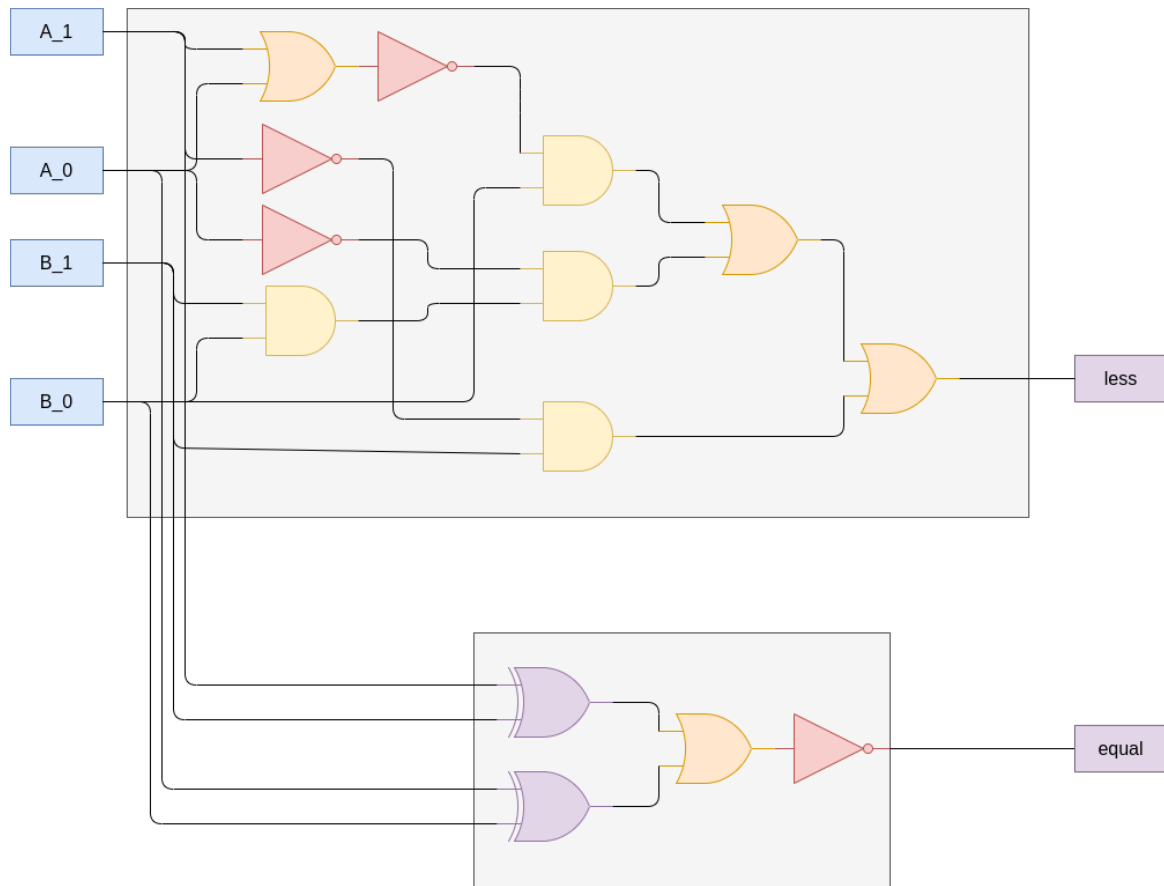
- Bộ so sánh 8bit

+ 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

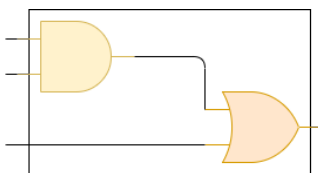
Bảng 2: Bảng sự thật cho bộ so sánh 2-bit $A < B$.

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

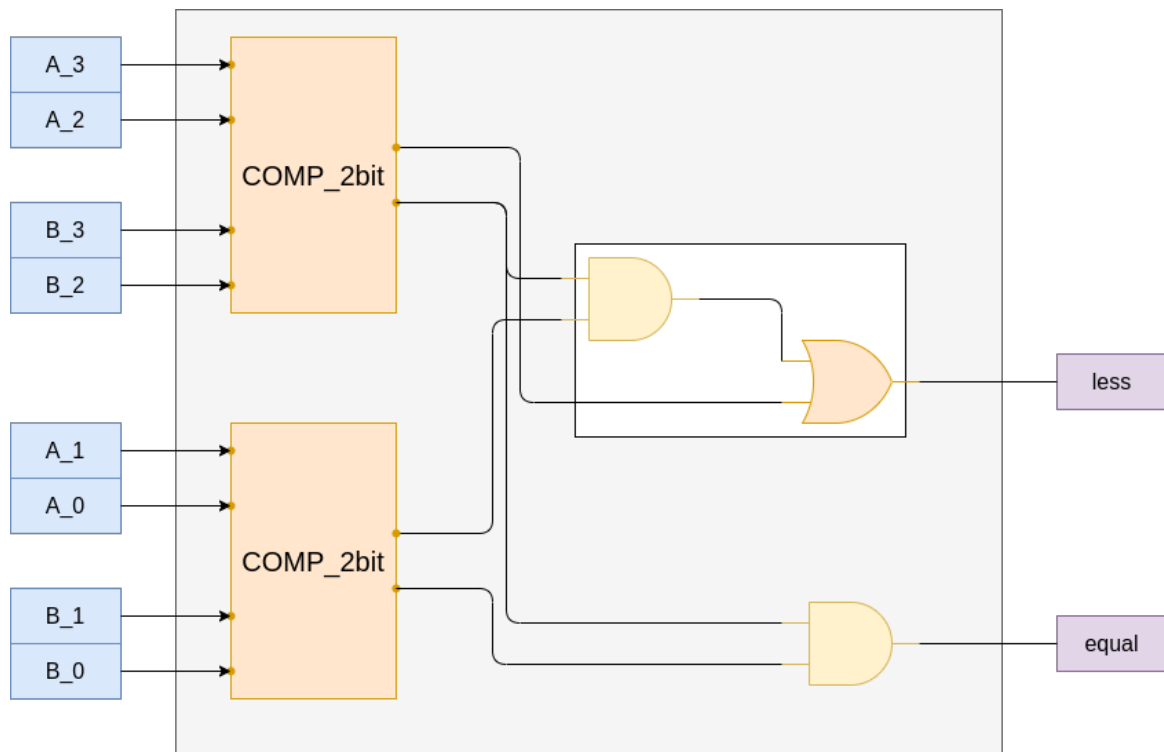


Hình 5: Sơ đồ logic của bộ Comparator 2-bit.

+ Triển khai bộ so sánh 4-bit



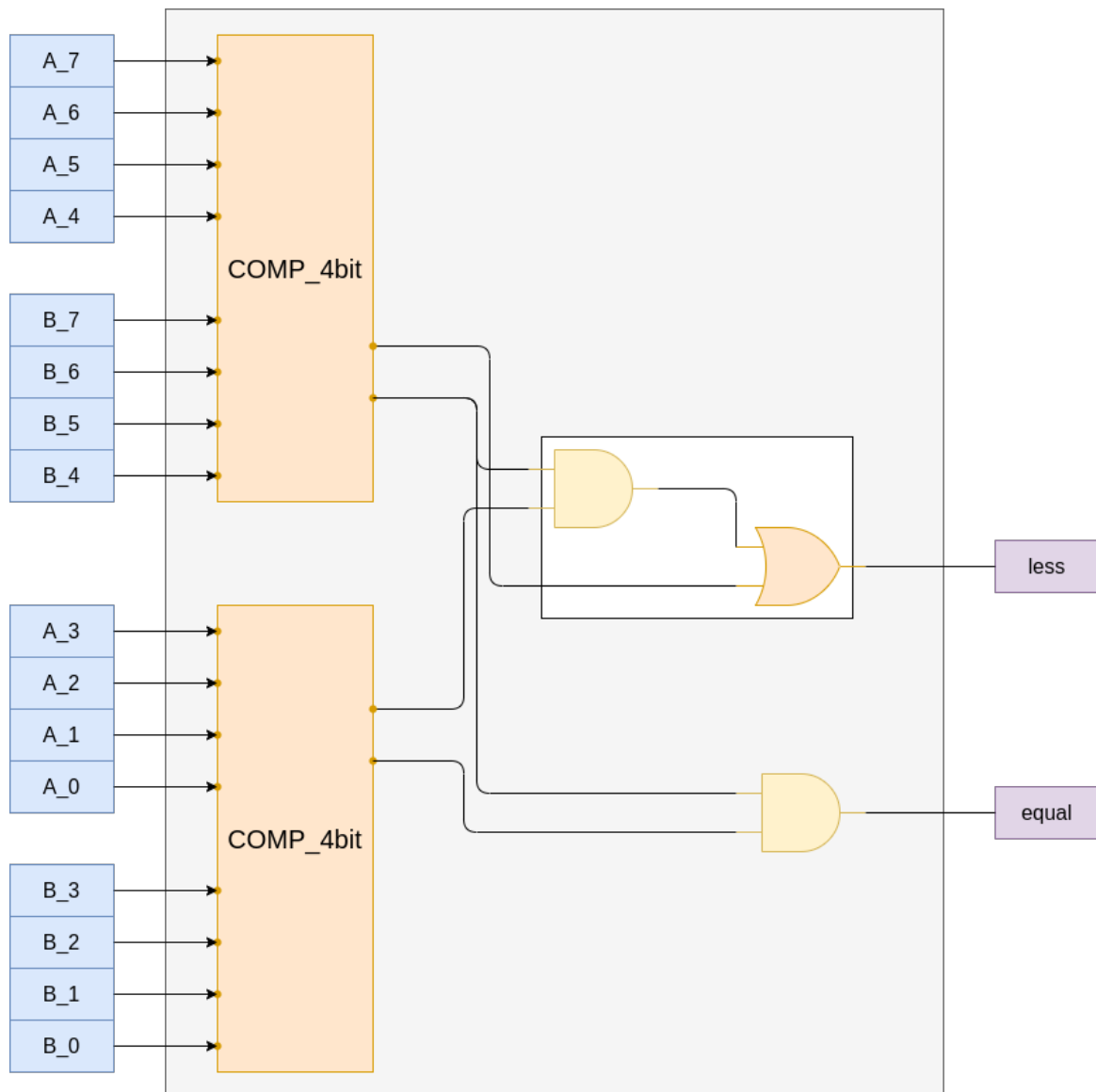
Sử dụng bộ lan truyền dấu so sánh trên, với nếu các bit trên có xuất hiện bit less thì ngõ ra lan truyền bit less, còn nếu bit thấp thì nếu tầng trên bằng nhau và các tầng thấp có bit less ngõ ra lan truyền bit less.



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

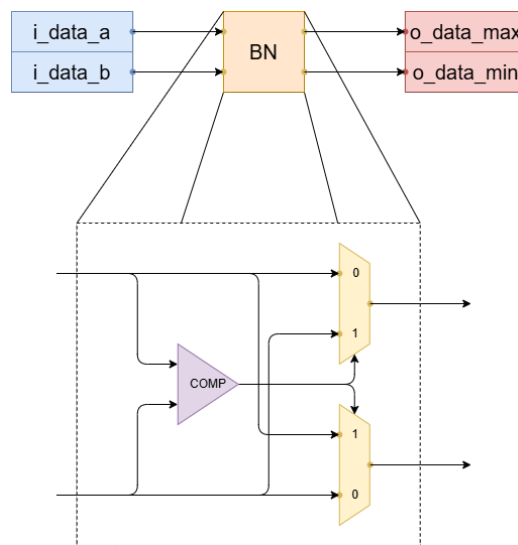
+ Triển khai bộ so sánh 8-bit

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



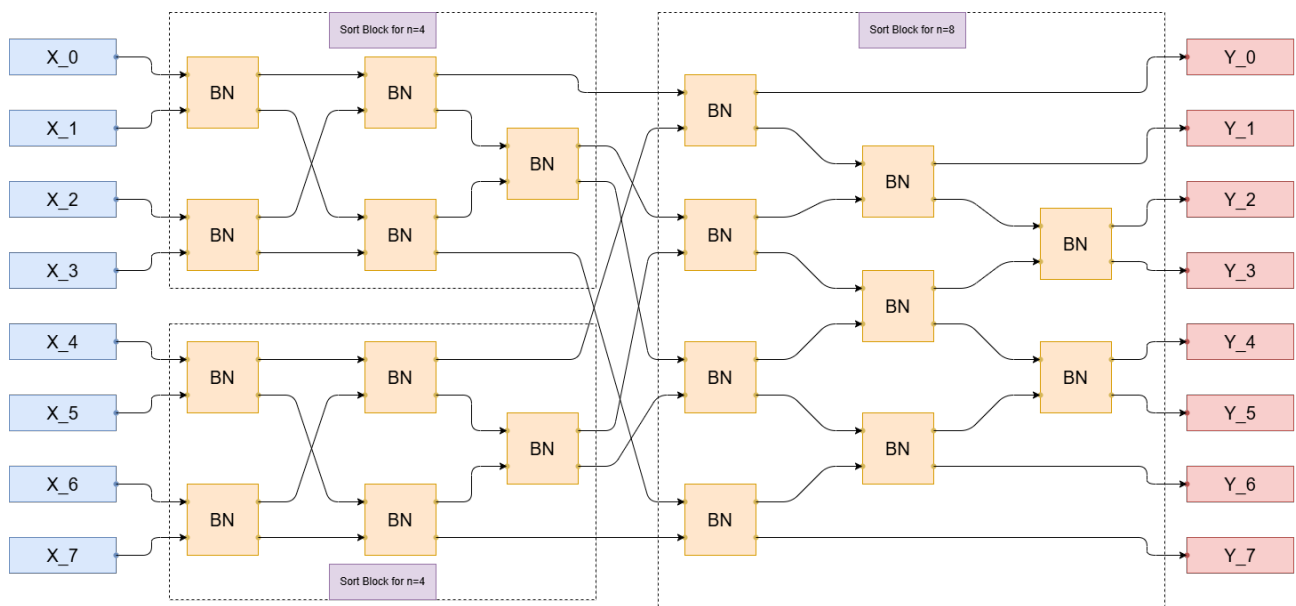
Hình 7: Sơ đồ logic của bộ Comparator 8-bit.

- 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.

```

1 module COMP_2bit(
2     input logic [1:0] i_data_a,
3     input logic [1:0] i_data_b,
4     output logic      o_less,
5     output logic      o_equal

```

```

6 );
7 // assign o_less = ((~i_data_a[1] & ~i_data_a[0]) & i_data_b[0]) | (~i_data_a[0] & (i_data_b[1] &
8 i_data_b[0])) | (~i_data_a[1] & i_data_b[1]);
9 // assign o_equal = ~(i_data_a ^ i_data_b);
10 assign o_less = ((~(i_data_a[1] | i_data_a[0])) & i_data_b[0]) | (~i_data_a[0] & (i_data_b[1] &
11 i_data_b[0])) | (~i_data_a[1] & i_data_b[1]);
12 assign o_equal = ~(i_data_a ^ i_data_b);
13 endmodule

```

Listing 6: Chương trình mô tả bộ so sánh 2-bit.

```

1 module COMP_4bit(
2     input logic [3:0] i_data_a,
3     input logic [3:0] i_data_b,
4     output logic      o_less,
5     output logic      o_equal
6 );
7     logic w_less_low, w_equal_low;
8     logic w_less_high, w_equal_high;
9
10    COMP_2bit u_low (
11        .i_data_a (i_data_a[1:0]),
12        .i_data_b (i_data_b[1:0]),
13        .o_less   (w_less_low),
14        .o_equal  (w_equal_low)
15    );
16
17    COMP_2bit u_high (
18        .i_data_a (i_data_a[3:2]),
19        .i_data_b (i_data_b[3:2]),
20        .o_less   (w_less_high),
21        .o_equal  (w_equal_high)
22    );
23
24    assign o_less = w_less_high | (w_equal_high & w_less_low);
25    assign o_equal = w_equal_high & w_equal_low;
26 endmodule

```

Listing 7: Chương trình mô tả bộ so sánh 4-bit.

```

1 module COMP_less #(
2     parameter SIZE_DATA = 8
3 )(
4     input logic [SIZE_DATA-1:0] i_data_a,
5     input logic [SIZE_DATA-1:0] i_data_b,
6     output logic      o_less
7 );
8     logic w_less_low, w_equal_low;
9     logic w_less_high, w_equal_high;
10
11    COMP_4bit u_low (
12        .i_data_a (i_data_a[3:0]),
13        .i_data_b (i_data_b[3:0]),
14        .o_less   (w_less_low),
15        .o_equal  (w_equal_low)
16    );
17
18    COMP_4bit u_high (
19        .i_data_a (i_data_a[7:4]),
20        .i_data_b (i_data_b[7:4]),
21        .o_less   (w_less_high),
22        .o_equal  (w_equal_high)

```

```

23     );
24
25     assign o_less = w_less_high | (w_equal_high & w_less_low);
26 endmodule

```

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

```

1  module Compare_and_Swap_unit #(
2      parameter IS_ASC      = 1 ,
3      parameter SIZE_DATA   = 8
4  )(
5      input logic [SIZE_DATA-1:0] i_data_a ,
6      input logic [SIZE_DATA-1:0] i_data_b ,
7      output logic [SIZE_DATA-1:0] o_data_max ,
8      output logic [SIZE_DATA-1:0] o_data_min
9  );
10
11  logic w_compare;
12  COMP_less #(
13      .SIZE_DATA (SIZE_DATA)
14  ) COMP_UNIT (
15      // o_less = i_data_a < i_data_b
16      .i_data_a (i_data_a),
17      .i_data_b (i_data_b),
18      .o_less (w_compare)
19  );
20  // COMP_parallel_prefix_binary #(
21  //     .SIZE_DATA (SIZE_DATA)
22  // ) COMP_UNT (
23  //     .i_data_a (i_data_a),
24  //     .i_data_b (i_data_b),
25  //     .o_less (w_compare)
26  // );
27
28  generate
29      if (IS_ASC) begin
30          assign o_data_max = w_compare ? i_data_b : i_data_a;
31          assign o_data_min = w_compare ? i_data_a : i_data_b;
32      end else begin
33          assign o_data_max = w_compare ? i_data_a : i_data_b;
34          assign o_data_min = w_compare ? i_data_b : i_data_a;
35      end
36  endgenerate
37
38  endmodule

```

Listing 9: Chương trình mô tả một đơn vị của bộ Bitonic Sort.

```

1  module Bitonic_Block4 #(
2      parameter IS_ASC      = 1 ,
3      parameter SIZE_DATA   = 8
4  )(
5      input logic [SIZE_DATA-1:0] i_data_0 ,
6      input logic [SIZE_DATA-1:0] i_data_1 ,
7      input logic [SIZE_DATA-1:0] i_data_2 ,
8      input logic [SIZE_DATA-1:0] i_data_3 ,
9      output logic [SIZE_DATA-1:0] o_data_0 ,
10     output logic [SIZE_DATA-1:0] o_data_1 ,
11     output logic [SIZE_DATA-1:0] o_data_2 ,
12     output logic [SIZE_DATA-1:0] o_data_3
13 );
14

```

```

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;

```

```

79 assign o_data_1 = w_data_max_2_0;
80 assign o_data_2 = w_data_min_2_0;
81 assign o_data_3 = w_data_min_1_1;
82
83 endmodule

```

Listing 10: Chương trình mô tả bộ Bitonic Sort Block-4.

```

1  module Bitonic_Block8 #(
2      parameter IS_ASC      = 1 ,
3      parameter SIZE_DATA   = 8
4  ) (
5      input logic [SIZE_DATA-1:0] i_data_0 ,
6      input logic [SIZE_DATA-1:0] i_data_1 ,
7      input logic [SIZE_DATA-1:0] i_data_2 ,
8      input logic [SIZE_DATA-1:0] i_data_3 ,
9      input logic [SIZE_DATA-1:0] i_data_4 ,
10     input logic [SIZE_DATA-1:0] i_data_5 ,
11     input logic [SIZE_DATA-1:0] i_data_6 ,
12     input logic [SIZE_DATA-1:0] i_data_7 ,
13     output logic [SIZE_DATA-1:0] o_data_0 ,
14     output logic [SIZE_DATA-1:0] o_data_1 ,
15     output logic [SIZE_DATA-1:0] o_data_2 ,
16     output logic [SIZE_DATA-1:0] o_data_3 ,
17     output logic [SIZE_DATA-1:0] o_data_4 ,
18     output logic [SIZE_DATA-1:0] o_data_5 ,
19     output logic [SIZE_DATA-1:0] o_data_6 ,
20     output logic [SIZE_DATA-1:0] o_data_7
21 );
22
23 ////////////////////////////////////////////////////
24 // Internal Logics
25 ////////////////////////////////////////////////////
26 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;
27 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;
28 wire [SIZE_DATA-1:0] w_data_max_1_0, w_data_max_1_1, w_data_max_1_2;
29 wire [SIZE_DATA-1:0] w_data_min_1_0, w_data_min_1_1, w_data_min_1_2;
30 wire [SIZE_DATA-1:0] w_data_max_2_0, w_data_max_2_1;
31 wire [SIZE_DATA-1:0] w_data_min_2_0, w_data_min_2_1;
32
33 ////////////////////////////////////////////////////
34 // SubModules
35 ////////////////////////////////////////////////////
36
37 Compare_and_Swap_unit #(
38     .IS_ASC      (IS_ASC),
39     .SIZE_DATA   (SIZE_DATA)
40 ) CAS_0_0 (
41     .i_data_a     (i_data_0),
42     .i_data_b     (i_data_1),
43     .o_data_max   (w_data_max_0_0),
44     .o_data_min   (w_data_min_0_0)
45 );
46 Compare_and_Swap_unit #(
47     .IS_ASC      (IS_ASC),
48     .SIZE_DATA   (SIZE_DATA)
49 ) CAS_0_1 (
50     .i_data_a     (i_data_2),
51     .i_data_b     (i_data_3),
52     .o_data_max   (w_data_max_0_1),
53     .o_data_min   (w_data_min_0_1)
54 );
55 Compare_and_Swap_unit #(

```

```

56     .IS_ASC          (IS_ASC),
57     .SIZE_DATA       (SIZE_DATA)
58 ) CAS_0_2 (
59     .i_data_a        (i_data_4),
60     .i_data_b        (i_data_5),
61     .o_data_max      (w_data_max_0_2),
62     .o_data_min      (w_data_min_0_2)
63 );
64 Compare_and_Swap_unit #(
65     .IS_ASC          (IS_ASC),
66     .SIZE_DATA       (SIZE_DATA)
67 ) CAS_0_3 (
68     .i_data_a        (i_data_6),
69     .i_data_b        (i_data_7),
70     .o_data_max      (w_data_max_0_3),
71     .o_data_min      (w_data_min_0_3)
72 );
73
74 Compare_and_Swap_unit #(
75     .IS_ASC          (IS_ASC),
76     .SIZE_DATA       (SIZE_DATA)
77 ) CAS_1_0 (
78     .i_data_a        (w_data_min_0_0),
79     .i_data_b        (w_data_max_0_1),
80     .o_data_max      (w_data_max_1_0),
81     .o_data_min      (w_data_min_1_0)
82 );
83 Compare_and_Swap_unit #(
84     .IS_ASC          (IS_ASC),
85     .SIZE_DATA       (SIZE_DATA)
86 ) CAS_1_1 (
87     .i_data_a        (w_data_min_0_1),
88     .i_data_b        (w_data_max_0_2),
89     .o_data_max      (w_data_max_1_1),
90     .o_data_min      (w_data_min_1_1)
91 );
92 Compare_and_Swap_unit #(
93     .IS_ASC          (IS_ASC),
94     .SIZE_DATA       (SIZE_DATA)
95 ) CAS_1_2 (
96     .i_data_a        (w_data_min_0_2),
97     .i_data_b        (w_data_max_0_3),
98     .o_data_max      (w_data_max_1_2),
99     .o_data_min      (w_data_min_1_2)
100 );
101
102 Compare_and_Swap_unit #(
103     .IS_ASC          (IS_ASC),
104     .SIZE_DATA       (SIZE_DATA)
105 ) CAS_2_0 (
106     .i_data_a        (w_data_min_1_0),
107     .i_data_b        (w_data_max_1_1),
108     .o_data_max      (w_data_max_2_0),
109     .o_data_min      (w_data_min_2_0)
110 );
111 Compare_and_Swap_unit #(
112     .IS_ASC          (IS_ASC),
113     .SIZE_DATA       (SIZE_DATA)
114 ) CAS_2_1 (
115     .i_data_a        (w_data_min_1_1),
116     .i_data_b        (w_data_max_1_2),
117     .o_data_max      (w_data_max_2_1),
118     .o_data_min      (w_data_min_2_1)
119 );

```

```

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 11: 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_o_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_o_data[0]),
31     .o_data_1     (w_o_data[1]),
32     .o_data_2     (w_o_data[2]),
33     .o_data_3     (w_o_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_o_data[4]),
44     .o_data_1     (w_o_data[5]),
45     .o_data_2     (w_o_data[6]),
46     .o_data_3     (w_o_data[7])
47 );
48
49 Bitonic_Block8 #(
50     .IS_ASC      (IS_ASC),

```

```

51         .SIZE_DATA      (SIZE_DATA)
52     ) BN_8_UNIT_0 (
53         .i_data_0        (w_0_data[0]),
54         .i_data_1        (w_0_data[4]),
55         .i_data_2        (w_0_data[1]),
56         .i_data_3        (w_0_data[5]),
57         .i_data_4        (w_0_data[2]),
58         .i_data_5        (w_0_data[6]),
59         .i_data_6        (w_0_data[3]),
60         .i_data_7        (w_0_data[7]),
61         .o_data_0        (w_sorted[0]),
62         .o_data_1        (w_sorted[1]),
63         .o_data_2        (w_sorted[2]),
64         .o_data_3        (w_sorted[3]),
65         .o_data_4        (w_sorted[4]),
66         .o_data_5        (w_sorted[5]),
67         .o_data_6        (w_sorted[6]),
68         .o_data_7        (w_sorted[7])
69     );
70
71     generate
72         for (i = 0; i < NUM_ELEM; i++) begin : PACK_OUTPUT
73             assign o_sorted[i*SIZE_DATA +: SIZE_DATA] = w_sorted[i];
74         end
75     endgenerate
76
77 endmodule

```

Listing 12: 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:

```

1  function automatic logic [SIZE_DATA*NUM_ELEM-1:0] f_ARR_sorted(
2      int                f_is_acs    ,
3      input logic [SIZE_DATA*NUM_ELEM-1:0] f_i_data
4  );
5      logic [SIZE_DATA-1:0] arr    [0:NUM_ELEM-1];
6      logic [SIZE_DATA-1:0] temp;
7      int i, j;
8
9      begin
10         for (i = 0; i < NUM_ELEM; i++) begin
11             arr[i] = f_i_data[i*SIZE_DATA +: SIZE_DATA];
12         end
13
14         for (i = 0; i < NUM_ELEM-1; i++) begin
15             for (j = 0; j < NUM_ELEM-1-i; j++) begin
16                 if (f_is_acs) begin
17                     if (arr[j] > arr[j+1]) begin
18                         temp    = arr[j];
19                         arr[j]   = arr[j+1];
20                         arr[j+1] = temp;
21                     end
22                 end else begin
23                     if (arr[j] < arr[j+1]) begin
24                         temp    = arr[j];

```



```
25         arr[j]    = arr[j+1];
26         arr[j+1] = temp;
27     end
28 end
29 end
30 end
31
32 for (i = 0; i < NUM_ELEM; i++) begin
33     f_ARR_sorted[i*SIZE_DATA +: SIZE_DATA] = arr[i];
34 end
35 end
36 endfunction
```

Listing 13: 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.