

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 3	12
a)	12
b)	16
c)	18
Câu 5	20
a)	20
b)	24
c)	26
Câu 6	33
a)	34
b)	38
c)	45

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 PG Generator.	12
6	Sơ đồ logic của Carry Generator.	13
7	Sơ đồ logic của bộ cộng 4-bit CLA.	14
8	Block diagram của bộ cộng 4-bit CLA.	15
9	Block diagram của bộ cộng 32-bit CLA.	16
10	Mux 2-1 cho toán hạng thứ nhất Op1.	21
11	Mux 4-1 cho toán hạng thứ Op2.	22
12	Sơ đồ logic của bộ ALU.	22
13	4-bit Carry Lookahead Adder.	23
14	8-bit Carry Lookahead Adder.	24
15	Sơ đồ logic của bộ Comparator 2-bit.	35
16	Sơ đồ logic của bộ Comparator 4-bit.	36
17	Sơ đồ logic của bộ Comparator 8-bit.	37
18	Bộ so sánh và swap giá trị trong Bitonic Merger Sort.	38
19	Bộ sắp xếp Bitonic Merger Sort.	38

Danh sách bảng

1	Bảng sự thật của bộ phát hiện bit 1 (Leading one position) cho 4 bit.	1
2	Bảng tính toán để thiết kế ALU.	20
3	Bảng lựa chọn toán hạng thứ hai.	21
4	Trường hợp đặc biệt được kiểm tra bằng Directed Test.	27
5	Bảng sự thật cho bộ so sánh 2-bit $A < B$	34

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	Test trường hợp 24-bit không có bit 1.	9
7	Kết quả của trường hợp 24-bit không có bit 1.	9
8	Test 24 trường hợp vị trí bit 1 cho bộ LOPD 24-bit.	9
9	Kết quả của TestCase1.	9
10	Test 100 trường hợp đầu vào ngẫu nhiên cho bộ LOPD 24-bit.	10
11	Kết quả của TestCase2.	10
12	Kết quả của tổng kết của bài test.	11
13	Chương trình mô tả CLA 4-bit.	16
14	Chương trình mô tả CLA 32-bit.	17
15	Giải thuật chứng minh kết quả của bộ cộng CLA 32-bit.	18
16	Sinh 100 mẫu random và thực hiện kiểm tra.	18
17	Kết quả test từng mẫu.	19
18	Kết quả của tổng kết của bài test.	19
19	Chương trình mô tả CLA 4-bit.	24
20	Chương trình mô tả CLA 8-bit.	25
21	Chương trình mô tả bộ ALU 8-bit.	25
22	Thực hiện Directed Test.	27
23	Kết quả mô phỏng Directed Test.	28
24	Thực hiện kiểm tra zero case.	31
25	Kết quả mô phỏng Directed Test.	31
26	Thực hiện Random Test.	31
27	Kết quả mô phỏng Directed Test.	32
28	Kết quả của tổng kết của bài test.	32
29	Chương trình mô tả bộ so sánh 2-bit.	38
30	Chương trình mô tả bộ so sánh 4-bit.	39
31	Chương trình mô tả bộ so sánh 8-bit.	39
32	Chương trình mô tả một đơn vị của bộ Bitonic Merger Sort.	40
33	Chương trình mô tả bộ Bitonic Merger Sort Block-4.	40
34	Chương trình mô tả bộ Bitonic Merger Sort Block-8.	42
35	Chương trình mô tả bộ Bitonic Merger Sort 8 phần tử đầu vào.	44
36	Giải thuật chứng minh bộ Bitonic Merger Sort 8 phần tử.	45
37	Test trường hợp các đầu vào đều là không.	46
38	Kết quả khi cho 8 phần tử ngõ vào đều là không.	47

39	Test 1 trường hợp ngẫu nhiên.	47
40	Kết quả khi cho 1 trường hợp ngẫu nhiên.	48
41	Test 100 trường hợp ngẫu nhiên.	48
42	Kết quả khi cho 100 trường hợp ngẫu nhiên.	48
43	Kết quả của tổng kết của bài test.	49
44	Kết quả khi cho 8 phần tử ngõ vào đều là không.	49
45	Kết quả khi cho 1 trường hợp ngẫu nhiên.	50
46	Kết quả khi cho 100 trường hợp ngẫu nhiên.	50
47	Kết quả của tổng kết của bài test.	51

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.

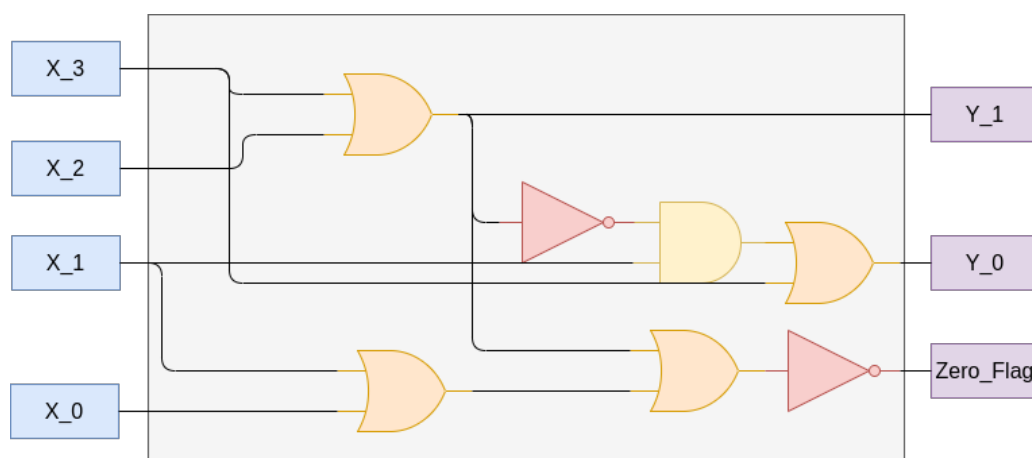
a) Thiết kế mạch chỉ được dùng các standard cell trên.

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

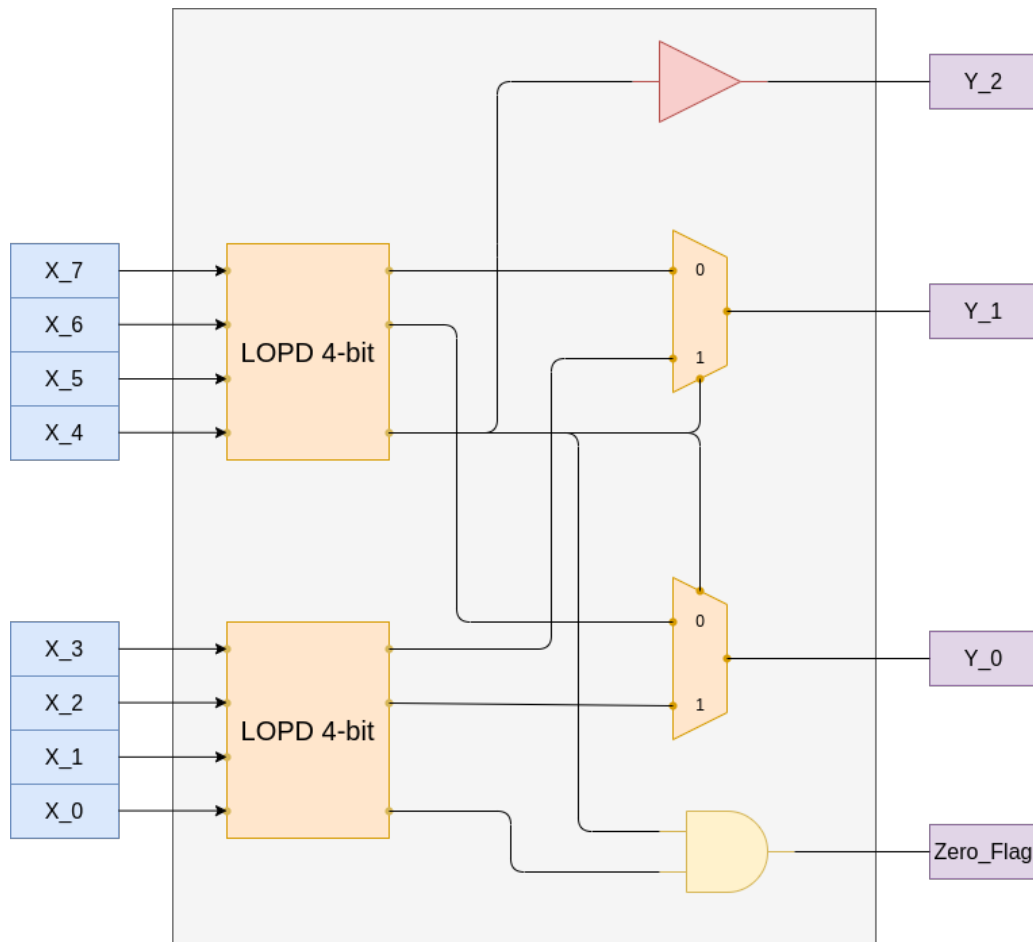
Bảng 1: Bảng sự thật của bộ phát hiện bit 1 (Leading one position) cho 4 bit.

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

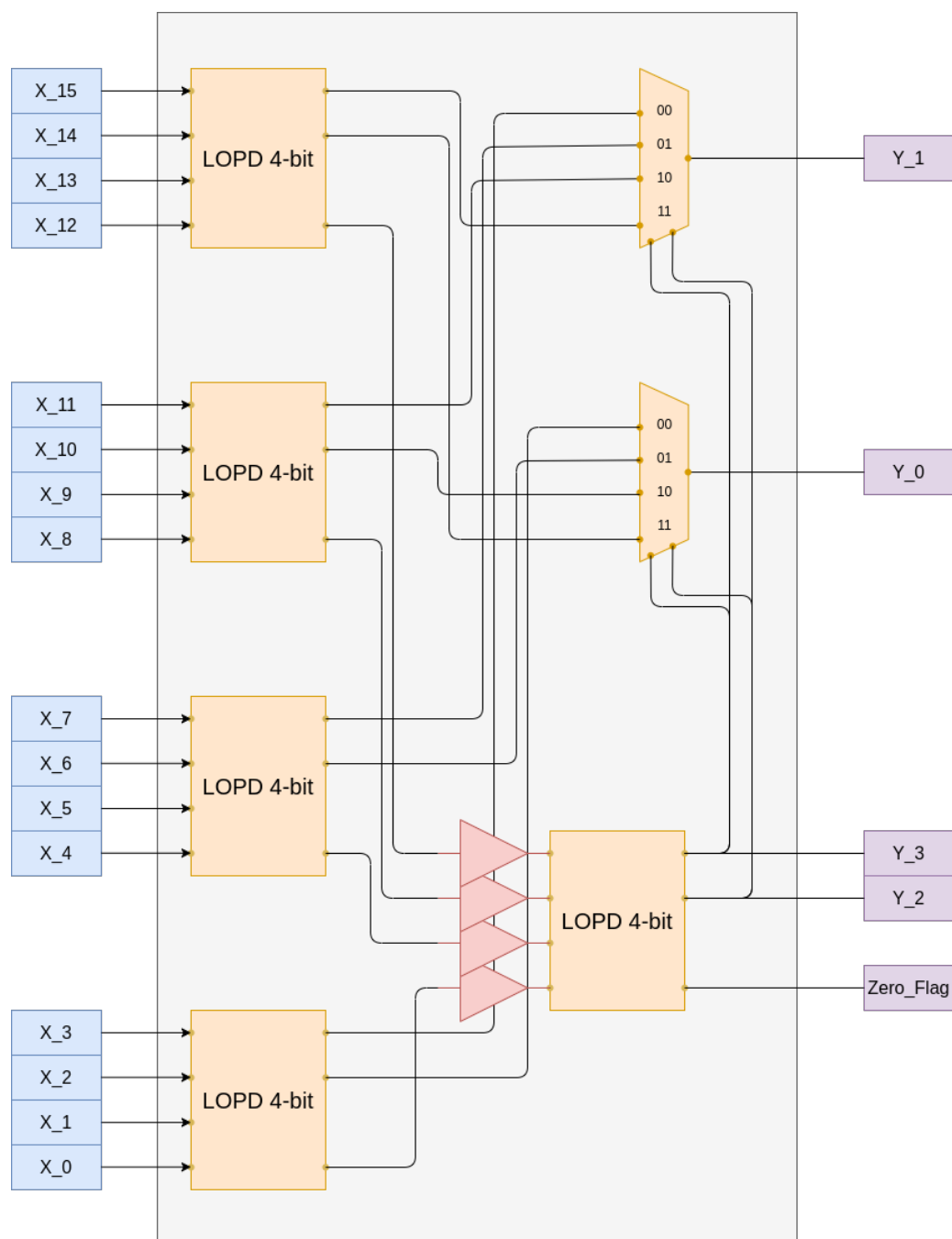


Hình 1: Sơ đồ logic của bộ LOPD 4bit.

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

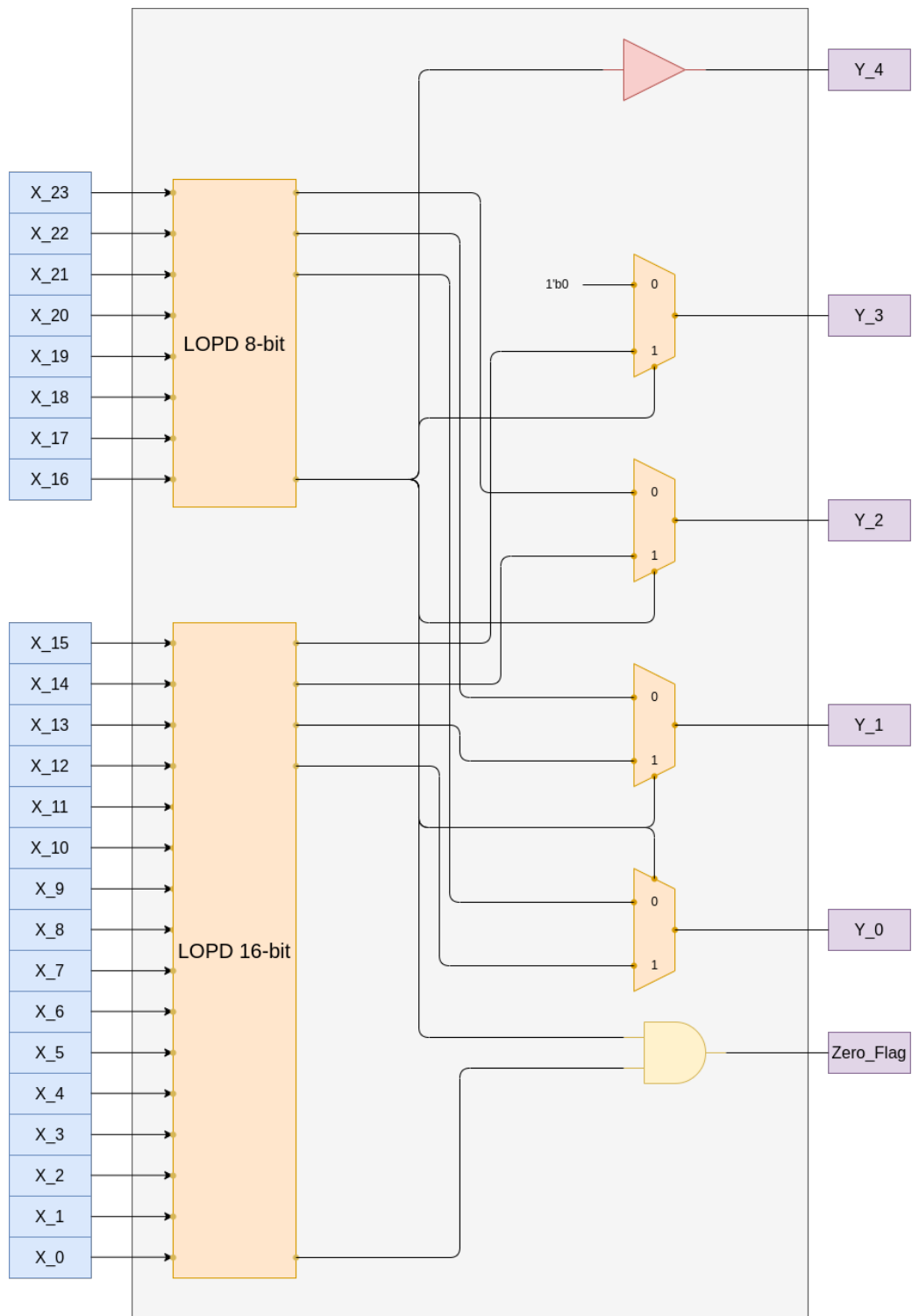


Hình 2: Sơ đồ logic của bộ LOPD 8bit.



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.

- TestCase0: Thực hiện test với đầu vào là 24'b0.

```

1      repeat (1) begin
2          @(posedge i_clk);
3          #1;
4          i_addr = i_addr + 1;
5          i_data = 24'b0;
6          @(negedge i_clk);
7          #1;
8          $display("[TIME: %5t] [%s] i_data = %b (%d) \t| o_one_position = %b (%d) \t| o_zero_flag = %
b", $time, "Zero", i_data, i_data, o_one_position, o_one_position, o_zero_flag);
9          $display("=> %4s: Expect: %8h, DUT: %8h ", (Test_LOPD(i_data) == o_one_position) ? "PASS" :
"FAIL", o_one_position, Test_LOPD(i_data));
10         test_count = test_count + 1;
11         test_pass = (Test_LOPD(i_data) == o_one_position) ? test_pass + 1 : test_pass;
12     end
13

```

Listing 6: Test trường hợp 24-bit không có bit 1.

Kết quả

```

[TIME: 41000] [Zero] i_data = 000000000000000000000000 (      0)      | o_one_position = 00000
( 0)      | o_zero_flag = 1
=> PASS: Expect: 00000000, DUT: 00000000

```

Listing 7: Kết quả của trường hợp 24-bit không có bit 1.

- 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.

```

1      bit_pos = 1;
2      repeat (24) begin
3          @(posedge i_clk);
4          #1;
5          i_addr = i_addr + 1;
6          i_data = bit_pos;
7          @(negedge i_clk);
8          #1;
9          $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);
10         $display("=> %4s: Expect: %8h, DUT: %8h ", (Test_LOPD(i_data) == o_one_position) ? "PASS" :
"FAIL", o_one_position, Test_LOPD(i_data));
11         test_count = test_count + 1;
12         test_pass = (Test_LOPD(i_data) == o_one_position) ? test_pass + 1 : test_pass;
13         bit_pos = bit_pos << 1'b1;
14     end
15

```

Listing 8: 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 = 000000000000000000000000100 ( 4) | o_one_position = 00010 (
2) | o_zero_flag = 0
=> PASS: Expect: 00000002, DUT: 00000002
[TIME: 71000] [Direcly] i_data = 0000000000000000000000001000 ( 8) | o_one_position = 00011 (
3) | o_zero_flag = 0
=> PASS: Expect: 00000003, DUT: 00000003
...
[TIME: 261000] [Direcly] i_data = 010000000000000000000000000 ( 4194304) | o_one_position = 10110
(22) | o_zero_flag = 0
=> PASS: Expect: 00000016, DUT: 00000016
[TIME: 271000] [Direcly] i_data = 1000000000000000000000000000 ( 8388608) | o_one_position = 10111
(23) | o_zero_flag = 0
=> PASS: Expect: 00000017, DUT: 00000017
```

Listing 9: Kết quả của TestCase1.

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

```
1 repeat (100) begin
2   @(posedge i_clk);
3   #1;
4   bit_pos = $urandom_range(0, SIZE_DATA-1);
5   i_data = 24'b1 << bit_pos;
6   if ($urandom_range(0, 1)) begin
7     i_data |= $urandom_range(0, (1 << SIZE_DATA) - 1);
8   end
9   #5;
10  $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 10: Test 100 trường hợp đầu vào ngẫu nhiên cho bộ LOPD 24-bit.

Kết quả

```
[TIME: 281000] [Random] i_data = 0001000000000000000000000000 ( 1048576) | o_one_position = 10100
(20) | o_zero_flag = 0
=> PASS: Expect: 00000014, DUT: 00000014
[TIME: 291000] [Random] i_data = 0000000000000000000000000000100 ( 4) | o_one_position = 00010 (
2) | o_zero_flag = 0
=> PASS: Expect: 00000002, DUT: 00000002
[TIME: 301000] [Random] i_data = 0000000000000000000000000000001 ( 1) | o_one_position = 00000 (
0) | o_zero_flag = 0
=> PASS: Expect: 00000000, DUT: 00000000
[TIME: 311000] [Random] i_data = 0100000000000000000000000000000 ( 4194304) | o_one_position = 10110
(22) | o_zero_flag = 0
=> PASS: Expect: 00000016, DUT: 00000016
...
[TIME: 1241000] [Random] i_data = 01000000000000000000000000000 ( 4194304) | o_one_position = 10110
(22) | o_zero_flag = 0
=> PASS: Expect: 00000016, DUT: 00000016
[TIME: 1251000] [Random] i_data = 00000000010000000000000000000 ( 16384) | o_one_position = 01110
(14) | o_zero_flag = 0
=> PASS: Expect: 0000000e, DUT: 0000000e
```

```
[TIME: 1261000] [Random] i_data = 00000010000000000000000000 ( 131072) | o_one_position = 10001
(17) | o_zero_flag = 0
=> PASS: Expect: 00000011, DUT: 00000011
[TIME: 1271000] [Random] i_data = 00000000001000000000000000 ( 8192) | o_one_position = 01101
(13) | o_zero_flag = 0
=> PASS: Expect: 0000000d, DUT: 0000000d
```

Listing 11: Kết quả của TestCase2.

- Kết quả tổng kết.

```
=====
=====TEST SUMMARY=====
Total test cases:    125
Passed              :    125
Failed              :      0
Pass rate           : 100.00%
=====
```

Listing 12: Kết quả của tổng kết của bài test.

Câu 3

Thiết kế mạch tính tổng của 2 số 32-bit sử dụng giải thuật CLA (Carry Look-Ahead Adder) trong lý thuyết. Lưu ý: tách thành các bộ cộng 4-bit CLA.

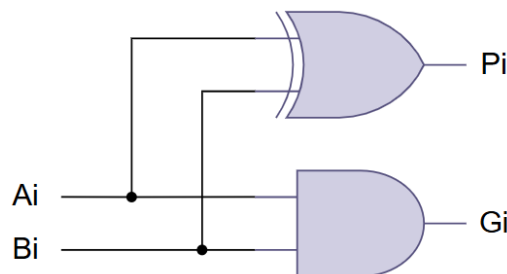
Cho các standard cell là: cổng not, các cổng logic 2, 3, 4 ngõ vào.

a) Thiết kế mạch theo phương pháp đã cho và chỉ được dùng các standard cell trên.

Theo lý thuyết, ta định nghĩa được 2 tín hiệu quan trọng là:

- **Generate (G_i)**: Sinh carry ngay tại bit đó: $G_i = A_i \& B_i$.
- **Propagate (P_i)**: Cho phép carry từ bit trước đi qua: $P_i = A_i \oplus B_i$.

Dựa vào đó, ta thiết kế PG Generator (Propagate-Generate Generator) như sau:

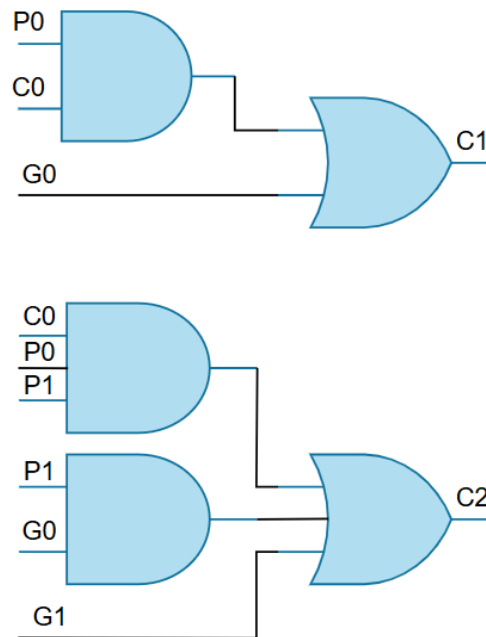


Hình 5: Sơ đồ logic của PG Generator.

Các tín hiệu G_i và P_i chính là đầu vào cho **Carry Generator** ở cấp kế tiếp. Biểu thức cho carry kế tiếp: $C_{i+1} = G_i | (P_i \& C_{in})$. Nếu thực hiện thế vào liên tục, ta được biểu thức như sau:

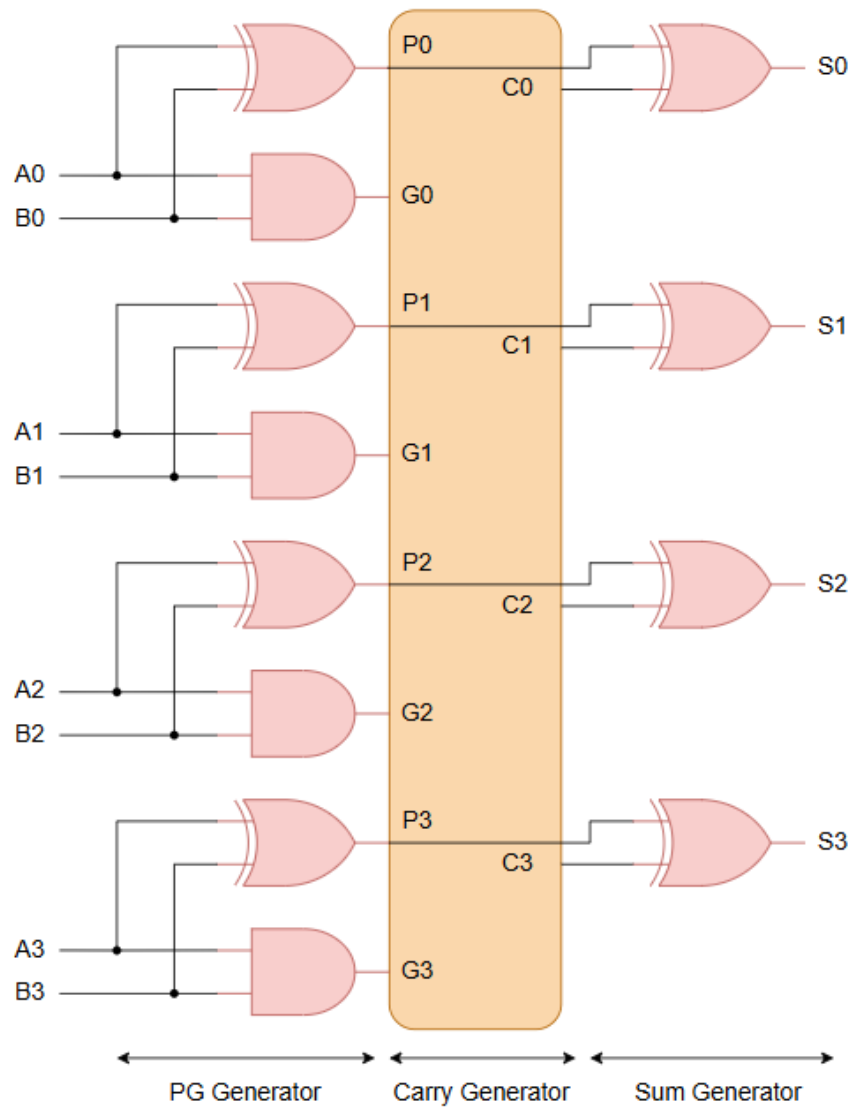
$$C_1 = G_0 | (P_0 \& C_{in}) \rightarrow C_2 = G_2 | (G_0 \& P_1) | (P_1 \& P_2 \& C_{in}) \dots$$

Tức là *tất cả các carry có thể tính song song* bằng các phép logic, không cần đợi ripple từng bit. Từ đó ta có thiết kế Carry Generator như sau:



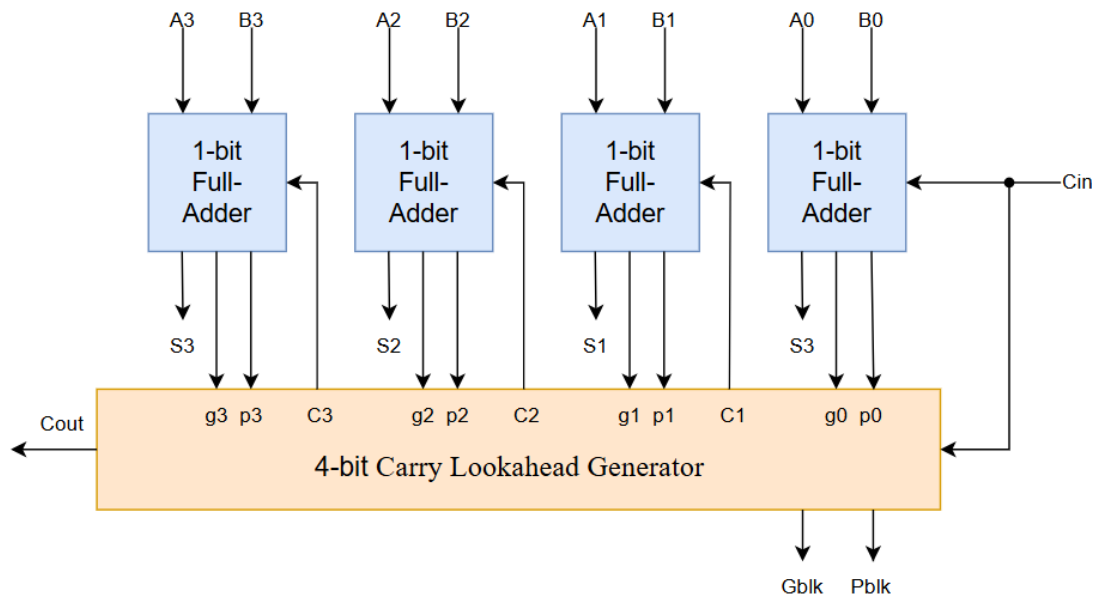
Hình 6: Sơ đồ logic của Carry Generator.

Sau khi biết các biết carry, ta tính tổng: $S_i = P_i \oplus C_{in}$. Từ đó, ta thiết kế bộ cộng CLA 4-bit như sau:



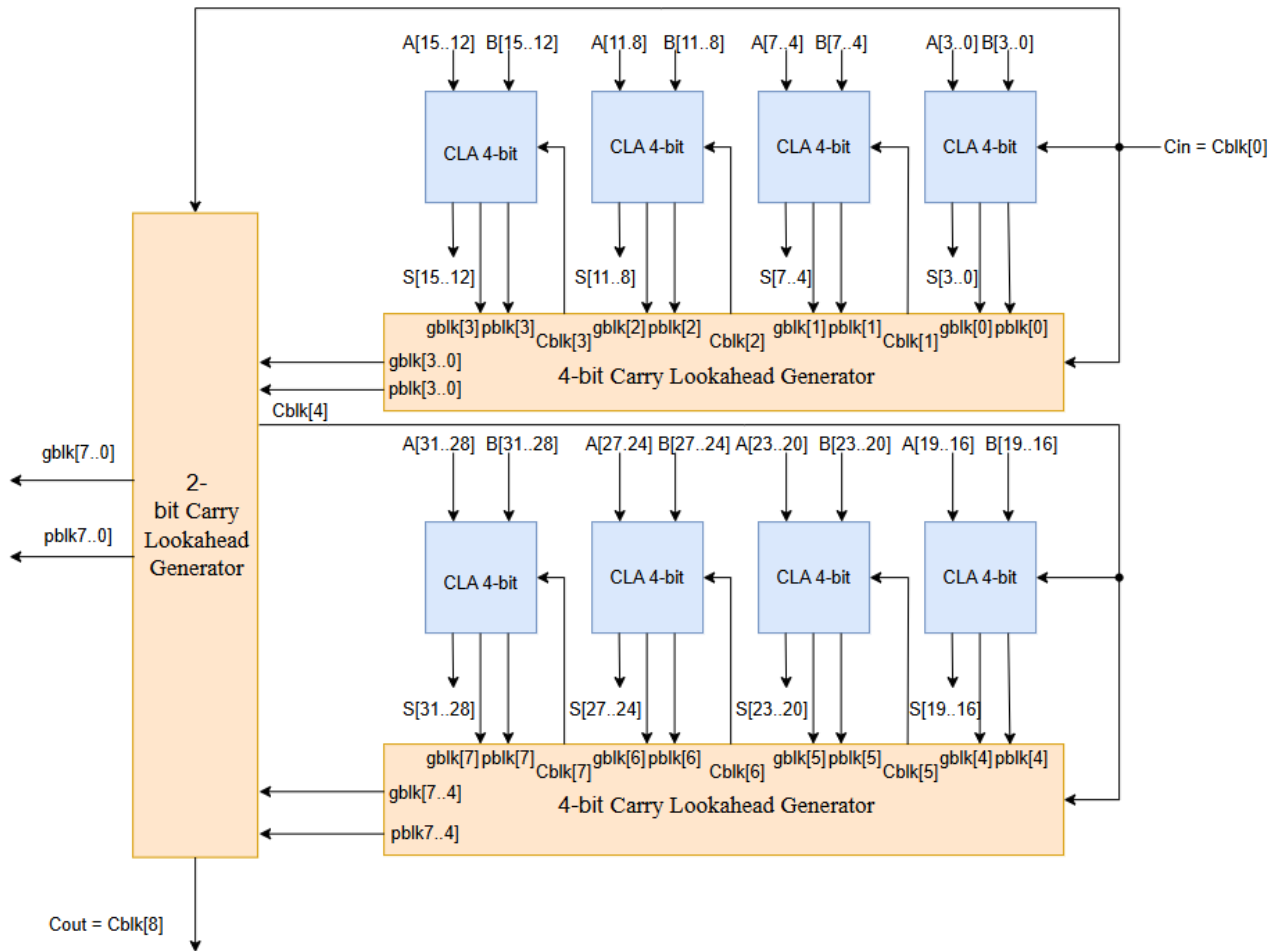
Hình 7: Sơ đồ logic của bộ cộng 4-bit CLA.

Ngoài ra, cũng là sơ đồ logic ấy, ta có thể biểu diễn thiết kế khác như hình bên dưới:



Hình 8: Block diagram của bộ cộng 4-bit CLA.

Từ những bộ trên, ta có được bộ tính tổng 2 số 32-bit bằng bộ cộng theo phương pháp CLA.



Hình 9: Block diagram của bộ cộng 32-bit CLA.

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

```

1  module cla_4bit (
2      input logic [3:0] A,
3      input logic [3:0] B,
4      input logic      Cin,
5      output logic      Gblk,
6      output logic      Pblk,
7      output logic [3:0] S,
8      output logic      Cout
9  );
10     logic [3:0] p, g;
11     logic c1, c2, c3;
12     assign p  = A ^ B;
13     assign g  = A & B;
14     assign c1 = g[0] | (p[0] & Cin);
15     assign c2 = g[1] | (g[0] & p[1]) | (p[1] & p[0] & Cin);
16     assign c3 = g[2] | (g[1] & p[2]) | (g[0] & p[1] & p[2]) | (p[2] & p[1] & p[0] & Cin);
17
18     logic and4, or3;
19     assign and4 = p[3] & p[2] & p[1] & p[0];
20     assign or3  = (g[2] & p[3]) | (g[1] & p[3] & p[2]) | (g[0] & p[3] & p[2] & p[1]);

```

```

21
22 assign Cout = g[3] | or3 | (and4 & Cin) ;
23 assign S[0] = p[0] ^ Cin;
24 assign S[1] = p[1] ^ c1;
25 assign S[2] = p[2] ^ c2;
26 assign S[3] = p[3] ^ c3;
27
28 assign Gblk = g[3] | or3;
29 assign Pblk = and4;
30
31
32 endmodule

```

Listing 13: Chương trình mô tả CLA 4-bit.

```

1 module cla_32bit (
2     input logic clk,
3     input logic rst_n,
4     input logic [31:0] A,
5     input logic [31:0] B,
6     input logic Cin,
7     output logic [31:0] Sum,
8     output logic Cout
9 );
10
11 logic [31:0] A_r, B_r;
12 logic Cin_r;
13
14 always_ff @(posedge clk or negedge rst_n) begin
15     if (!rst_n) begin
16         A_r <= '0;
17         B_r <= '0;
18         Cin_r <= 1'b0;
19     end else begin
20         A_r <= A;
21         B_r <= B;
22         Cin_r <= Cin;
23     end
24 end
25
26 logic [7:0] Pblk, Gblk;
27 logic [8:0] Cblk;
28 logic [31:0] Sum_c;
29 logic Cout_c;
30
31 assign Cblk[0] = Cin_r;
32
33 genvar i;
34 generate
35     for (i = 0; i < 8; i++) begin : BLK
36         cla_4bit u4 (
37             .A (A_r[4*i +: 4]),
38             .B (B_r[4*i +: 4]),
39             .cin (Cblk[i]),
40             .sum (Sum_c[4*i +: 4]),
41             .cout (),
42             .Gblk (Gblk[i]),
43             .Pblk (Pblk[i])
44         );
45         assign Cblk[i+1] = Gblk[i] | (Pblk[i] & Cblk[i]);
46     end
47 endgenerate
48

```

```

49     assign Cout_c = Cblk[8];
50
51
52     always_ff @(posedge clk or negedge rst_n) begin
53         if (!rst_n) begin
54             Sum <= '0;
55             Cout <= 1'b0;
56         end else begin
57             Sum <= Sum_c;
58             Cout <= Cout_c;
59         end
60     end
61 endmodule

```

Listing 14: Chương trình mô tả CLA 32-bit.

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

Nhóm em sử dụng dụng dẫu + làm golden model để thực hiện kiểm tra kết quả test như sau:

```

1     expected = {1'b0, tv_a} + {1'b0, tv_b} + tv_cin;

```

Listing 15: Giải thuật chứng minh kết quả của bộ cộng CLA 32-bit.

- Thực hiện kiểm thử (self-checking) với 100 mẫu random để áp vào DUT. So sánh với expected để cập nhật scoreboard.

```

1     $display("=== Start run_test (100 samples) ===");
2     for (idx = 0; idx < 100; idx++) begin
3         // --- generate testcase ---
4         tv_a = $urandom();
5         tv_b = $urandom();
6         tv_cin = $urandom_range(0,1);
7         A_tb = tv_a;
8         B_tb = tv_b;
9         Cin_tb = tv_cin;
10        @(posedge clk);
11        @(posedge clk);
12        #1;
13        // --- compute expected ---
14        expected = {1'b0, tv_a} + {1'b0, tv_b} + tv_cin;
15        got = {Cout_tb, Sum_tb};
16        // --- compare and display ---
17        if (got == expected) begin
18            pass_count++;
19            $display("PASS [%0d] A=0x%08h B=0x%08h Cin=%0d => {Cout,Sum}=0x%09h",
20                idx, tv_a, tv_b, tv_cin, got);
21        end else begin
22            fail_count++;
23            $display("FAIL [%0d] A=0x%08h B=0x%08h Cin=%0d => got=0x%09h (exp=0x%09h)",
24                idx, tv_a, tv_b, tv_cin, got, expected);

```

```

25         end
26     end
27

```

Listing 16: Sinh 100 mẫu random và thực hiện kiểm tra.

Kết quả

```

# === Start run_test (100 samples) ===
# PASS [0] A=0x2e45b278 B=0xd7aeae53 Cin=1 => {Cout,Sum}=0x105f460cc
# PASS [1] A=0x1e3a4d4a B=0x4ba544f7 Cin=1 => {Cout,Sum}=0x069df9242
# PASS [2] A=0xf331e3ec B=0x55820c15 Cin=0 => {Cout,Sum}=0x148b3f001
# PASS [3] A=0xfb325326 B=0x4cd94c49 Cin=0 => {Cout,Sum}=0x1480b9f6f
...
# PASS [97] A=0xb68bc493 B=0x18b2dfcc Cin=0 => {Cout,Sum}=0x0cf3ea45f
# PASS [98] A=0x3662a0de B=0x9cc1dc3f Cin=0 => {Cout,Sum}=0x0d3247d1d
# PASS [99] A=0xafde0bca B=0xcf523f4e Cin=1 => {Cout,Sum}=0x17f304b19

```

Listing 17: Kết quả test từng mẫu.

- Kết quả tổng kết.

```

# === Test summary ===
# Total samples = 100
# PASS = 100
# FAIL = 0
# === End run_test ===

```

Listing 18: Kết quả của tổng kết của bài test.

Câu 5

Thiết kế một ALU thực hiện theo tính toán như bảng sau. Trong đó, A và B là 2 ngõ vào 8-bit. Các tín hiệu lựa chọn S_1 , S_0 và C_{in} .

Bảng 2: Bảng tính toán để thiết kế ALU.

S_1S_0	$C_{in} = 0$	$C_{in} = 1$
00	$F = A + B$ (add)	$F = A + B + 1$
01	$F = A$ (transfer)	$F = A + 1$ (increment)
10	$F = \bar{B}$ (complement)	$F = \bar{B} + 1$ (negate)
11	$F = A + \bar{B}$	$F = A + \bar{B} + 1$ (subtract)

Cho các standard cell là: cổng not, các cổng logic 2 ngõ vào, mux 2-1, mux 4-1.

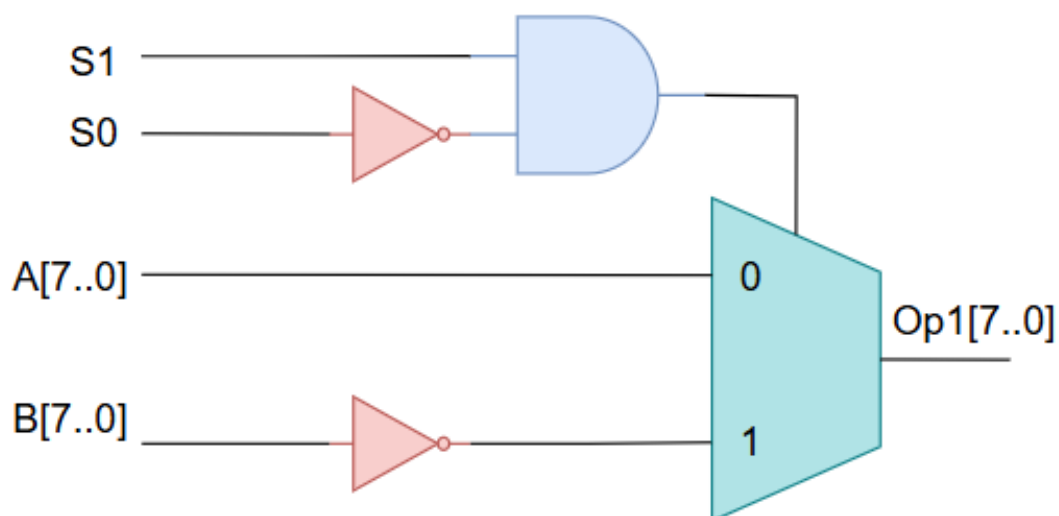
a) Thiết kế mạch chỉ sử dụng các standard cell trên và chỉ được thiết kế 1 bộ cộng.

Trước tiên, dựa vào Bảng 2, ta có thể rút ra được công thức tổng quát như sau:

$$F = A + B + C_{in}$$

Mà C_{in} sẽ được điều khiển trực tiếp để đưa vào bộ cộng nên không cần dùng mux. Tóm lại ta chỉ cần quan tâm đến 2 toán hạng còn lại gọi là Op1 và Op2.

Ngoài ra, quan sát thấy tại $S_1S_0 = 10$ chỉ có 1 toán hạng là $\bar{B} \rightarrow$ Dùng mux 2-1 để chọn ra trường hợp đặc biệt này, dưới đây là hình ảnh thiết kế.



Hình 10: Mux 2-1 cho toán hạng thứ nhất Op1.

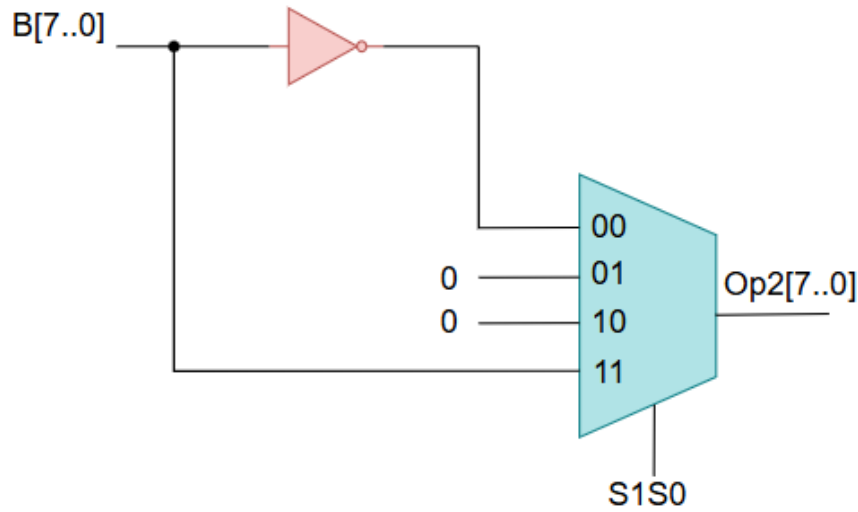
Hình 10 là thiết kế được tóm gọn lại. Thực chất, vì là A và B là số 8-bit nên sẽ là 8 bộ mux 2-1 cho mỗi một bit.

Sau khi có toán hạng thứ nhất, ta tiến đến toán hạng thứ hai dựa vào toán hạng thứ nhất như sau:

S1S0	Toán hạng 1 (Op1)	Toán hạng 2 (Op2)
00	A	B
01	A	0
10	\bar{B}	0
11	A	\bar{B}

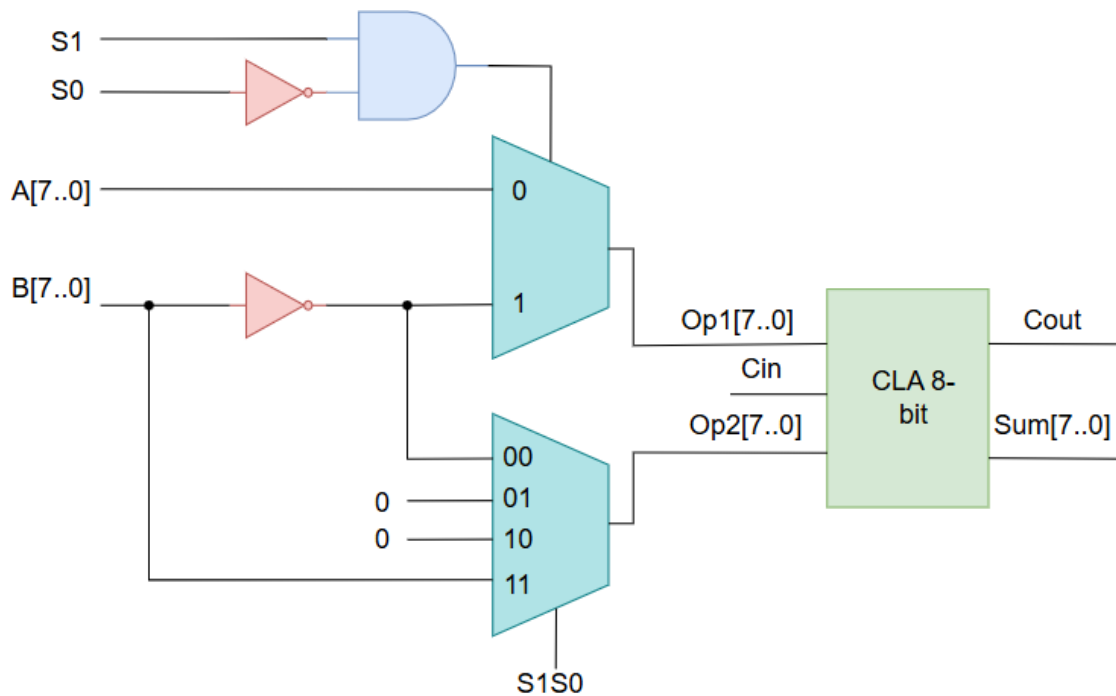
Bảng 3: Bảng lựa chọn toán hạng thứ hai.

Dựa vào bảng 3, nhóm em chọn mux 4-1 để thiết kế cho phù hợp với 4 trường hợp trên. Tương tự, vì là thiết kế cho mỗi bit nên thực chất sẽ có đến 8 bộ mux 4-1.



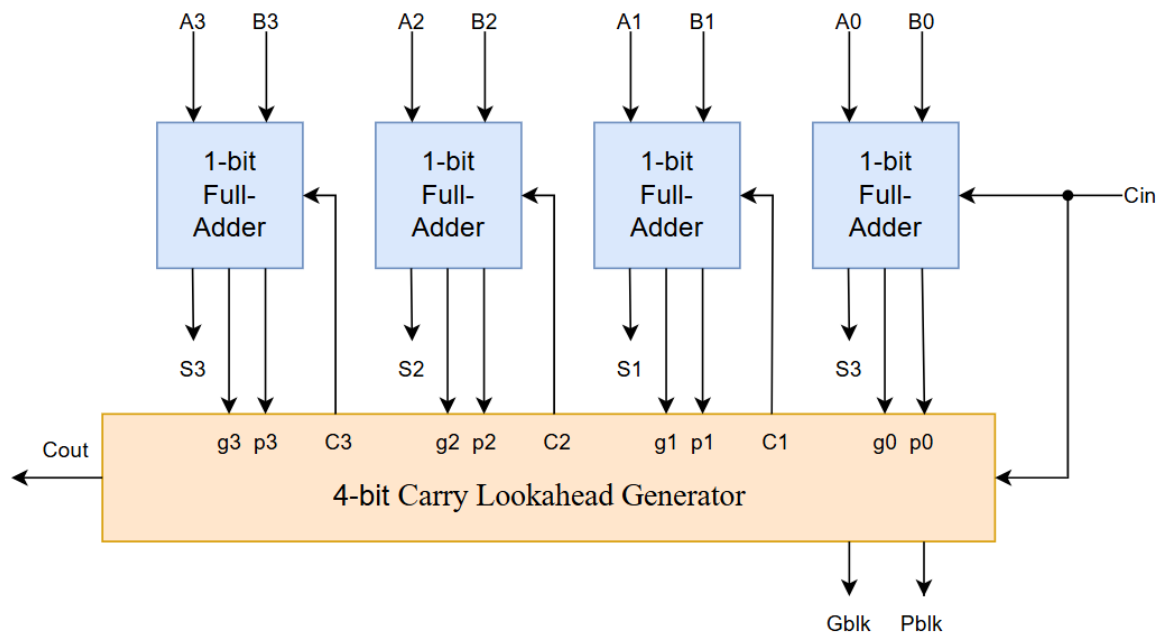
Hình 11: Mux 4-1 cho toán hạng thứ Op2.

Sau khi đã có đầy đủ hai toán hạng, việc còn lại là thiết kế bộ cộng với đầu vào là toán hạng thứ nhất và thứ hai đã chọn cùng với C_{in} . Tổng quát, bộ ALU sẽ có thiết kế như sau:



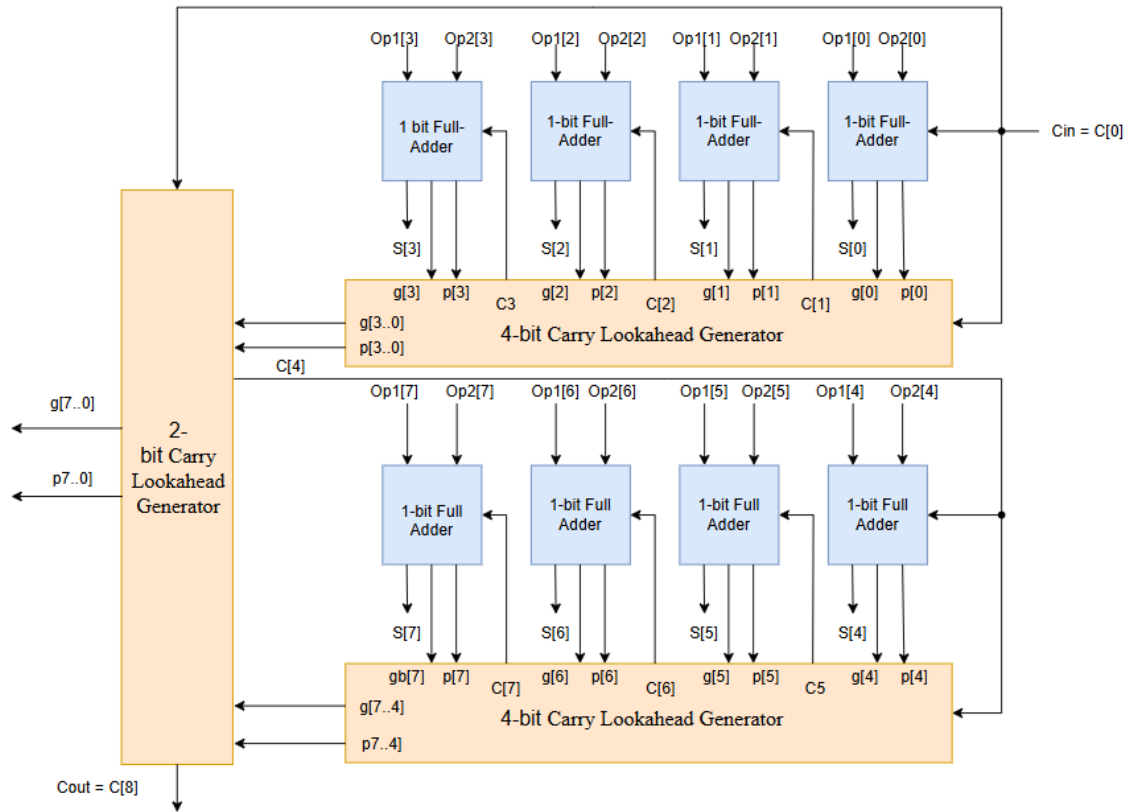
Hình 12: Sơ đồ logic của bộ ALU.

Đối với bộ cộng, nhóm em sử dụng bộ cộng CLA 8-bit. Trước hết, cần phải xây dựng một khối CLA 4-bit như sau.



Hình 13: 4-bit Carry Lookahead Adder.

Dựa vào thiết kế này, ta nối 2 khối CLA 4-bit thành bộ cộng CLA 8-bit, cụ thể như hình dưới:



Hình 14: 8-bit Carry Lookahead Adder.

b) Viết chương trình HDL mô tả mạch đã cho và viết testbench cho mạch

```

1  module cla_4bit (
2      input logic [3:0] A,
3      input logic [3:0] B,
4      input logic cin,
5      output logic cout, Pblk, Gblk,
6      output logic [3:0] sum
7  );
8      logic [3:0] g, p;
9      assign g = A & B;
10     assign p = A ^ B;
11
12     logic c1, c2, c3;
13     assign c1 = g[0] | (p[0] & cin);
14     assign c2 = g[1] | (g[0] & p[1]) | (p[1] & p[0] & cin);
15     assign c3 = g[2] | (g[1] & p[2]) | (g[0] & p[2] & p[1]) | (p[2] & p[1] & p[0] & cin);
16     assign cout = g[3] | (g[2] & p[3]) | (g[1] & p[3] & p[2]) | (g[0] & p[3] & p[2] & p[1]) | (p[3] & p[2] & p[1] & p[0] & cin);
17
18     assign sum[0] = p[0] ^ cin;
19     assign sum[1] = p[1] ^ c1;
20     assign sum[2] = p[2] ^ c2;
21     assign sum[3] = p[3] ^ c3;
22     assign Pblk = p[3] & p[2] & p[1] & p[0];

```

```

23 assign Gblk = g[3] | (g[2]&p[3]) | (g[1]&p[3]&p[2]) | (g[0]&p[3]&p[2]&p[1]);
24
25 endmodule
26

```

Listing 19: Chương trình mô tả CLA 4-bit.

```

1  module cla_8bit (
2      input logic [7:0] A,
3      input logic [7:0] B,
4      input logic      Cin,
5      output logic [7:0] Sum,
6      output logic      Cout
7  );
8      logic [1:0] Gblk, Pblk;
9      logic [2:0] Cblk;
10     assign Cblk[0] = Cin;
11
12     genvar i;
13     generate
14         for (i = 0; i < 2; i = i + 1) begin : BLK
15             cla_4bit u4 (
16                 .A (A[4*i +: 4]),
17                 .B (B[4*i +: 4]),
18                 .cin (Cblk[i]),
19                 .sum (Sum[4*i +: 4]),
20                 .cout(),
21                 .Gblk(Gblk[i]),
22                 .Pblk(Pblk[i])
23             );
24
25             assign Cblk[i+1] = Gblk[i] | (Pblk[i] & Cblk[i]);
26         end
27     endgenerate
28
29     assign Cout = Cblk[2];
30 endmodule

```

Listing 20: Chương trình mô tả CLA 8-bit.

```

1  module alu_8bit (
2      input logic [7:0] A,
3      input logic [7:0] B,
4      input logic      S1,
5      input logic      S0,
6      input logic      Cin,
7      output logic [7:0] F,
8      output logic      Cout
9  );
10
11     logic [7:0] B_comp;
12     assign B_comp = ~B;
13
14     logic sel_op1;
15     assign sel_op1 = S1 & (~S0);
16
17     logic [7:0] Op1;
18
19     genvar i;
20     generate
21         for (i = 0; i < 8; i = i + 1) begin : OP1_MUX
22             assign Op1[i] = sel_op1 ? B_comp[i] : A[i];

```

```

23     end
24 endgenerate
25
26 logic [7:0] Op2;
27 always_comb begin
28     case ({S1, S0})
29         2'b00: Op2 = B;
30         2'b01: Op2 = 8'b0;
31         2'b10: Op2 = 8'b0;
32         2'b11: Op2 = B_comp;
33         default: Op2 = 8'b0;
34     endcase
35 end
36
37 cla_8bit u_adder (
38     .A (Op1),
39     .B (Op2),
40     .Cin (Cin),
41     .Sum (F),
42     .Cout (Cout)
43 );
44
45 endmodule

```

Listing 21: Chương trình mô tả bộ ALU 8-bit.

c) Viết testbench cho mạch.

Mục tiêu: Xác minh chức năng ALU 8-bit theo bảng S_1 , S_0 , C_{in} ; đảm bảo F và C_{out} của DUT khớp với mô hình tham chiếu trên tập mẫu Directed + Random (100 mẫu tổng).

Ở đây, phương pháp nhóm em áp dụng là Functional Verification - Self-checking testbench. Cụ thể hơn, đây là Direct + Random Testing kết hợp Reference Model Comparison.

Nhóm sử dụng Reference Model để tính toán giá trị expected. Sau đó, thực hiện task để so sánh với giá trị expected và thực hiện cập nhật PASS/FAIL. Nhóm thực hiện test 100 trường hợp với những trường hợp và phương pháp test khác nhau.

- TestCase0: Thực hiện Directed Test kiểm tra trường hợp đầu vào đặc biệt mà logic dễ sai như sau:

TH	Giá trị test A và B	Tác dụng test
1	$A = 8'h00,$ $B = 8'h00$	Kiểm tra tính cộng cơ bản ($0 + 0 = 0$) và xác nhận không có bit rác hoặc lỗi <i>sign extension</i> .
2	$A = 8'hFF,$ $B = 8'hFF$	Kiểm tra tràn (<i>overflow/carry</i>); khi $A = B = FF_h$, phép cộng cho FE_h và <i>carry</i> = 1.
3	$A = 8'hFF,$ $B = 8'h01$	Kiểm tra sinh <i>Cout</i> khi cộng giá trị lớn nhất với 1 và xác nhận mạch <i>carry propagate</i> hoạt động đúng ở bit thấp.
4	$A = 8'h80,$ $B = 8'h80$	Kiểm tra xử lý bit 7 (MSB), vì $0x80 = 1000_0000$. Với signed, đây là vùng âm; với unsigned, kiểm tra tràn.
5	$A = 8'hAA,$ $B = 8'h55$	Kiểm tra truyền <i>carry</i> xen kẽ. Các bit 1–0 xen kẽ giúp phát hiện lỗi nối bit hoặc XOR sai.
6	$A = 8'h55,$ $B = 8'hAA$	Kiểm tra tính đối xứng của phép cộng ($A + B = B + A$).
7	$A = 8'h0F,$ $B = 8'hF0$	Kiểm tra truyền <i>carry</i> giữa nibble thấp–cao, đánh giá hoạt động cộng giữa các bit chéo.

Bảng 4: Trường hợp đặc biệt được kiểm tra bằng Directed Test.

```

1  task automatic run_test();
2  logic [7:0] edgeA [0:6];
3  logic [7:0] edgeB [0:6];
4  int idx;
5  int s1, s0, c;
6  int rctl;
7  begin
8      total_tests = 0;
9      errors = 0;
10     test_count = 0;
11     test_pass = 0;
12
13     // --- Directed edge cases ---
14     edgeA[0] = 8'h00; edgeB[0] = 8'h00;
15     edgeA[1] = 8'hFF; edgeB[1] = 8'hFF;
16     edgeA[2] = 8'hFF; edgeB[2] = 8'h01;
17     edgeA[3] = 8'h80; edgeB[3] = 8'h80;
18     edgeA[4] = 8'hAA; edgeB[4] = 8'h55;
19     edgeA[5] = 8'h55; edgeB[5] = 8'hAA;
20     edgeA[6] = 8'h0F; edgeB[6] = 8'hF0;

```



```

21
22     $display("\n===== STARTING DIRECTED TESTS =====\n");
23     for (idx = 0; idx <= 6; idx++) begin
24         for (s1 = 0; s1 < 2; s1++) begin
25             for (s0 = 0; s0 < 2; s0++) begin
26                 for (c = 0; c < 2; c++) begin
27                     apply_and_check(edgeA[idx], edgeB[idx], s1, s0, c, "Direct");
28                 end
29             end
30         end
31     end
32 end
33 endtask
34

```

Listing 22: Thực hiện Directed Test.

Kết quả

```

# ===== STARTING DIRECTED TESTS =====
#
# [TIME: 1000] [Direct] A=00 B=00 S1S0=00 Cin=0 | F=00 Cout=0
# => PASS: Expect: 00 (0), DUT: 00 (0)
# -----
# [TIME: 2000] [Direct] A=00 B=00 S1S0=00 Cin=1 | F=01 Cout=0
# => PASS: Expect: 01 (1), DUT: 01 (1)
# -----
# [TIME: 3000] [Direct] A=00 B=00 S1S0=01 Cin=0 | F=00 Cout=0
# => PASS: Expect: 00 (0), DUT: 00 (0)
# -----
# [TIME: 4000] [Direct] A=00 B=00 S1S0=01 Cin=1 | F=01 Cout=0
# => PASS: Expect: 01 (1), DUT: 01 (1)
# -----
# [TIME: 5000] [Direct] A=00 B=00 S1S0=10 Cin=0 | F=ff Cout=0
# => PASS: Expect: ff (255), DUT: ff (255)
# -----
# [TIME: 6000] [Direct] A=00 B=00 S1S0=10 Cin=1 | F=00 Cout=1
# => PASS: Expect: 00 (0), DUT: 00 (0)
# -----
# [TIME: 7000] [Direct] A=00 B=00 S1S0=11 Cin=0 | F=ff Cout=0
# => PASS: Expect: ff (255), DUT: ff (255)
# -----
# [TIME: 8000] [Direct] A=00 B=00 S1S0=11 Cin=1 | F=00 Cout=1
# => PASS: Expect: 00 (0), DUT: 00 (0)
# -----
# [TIME: 9000] [Direct] A=ff B=ff S1S0=00 Cin=0 | F=fe Cout=1
# => PASS: Expect: fe (254), DUT: fe (254)
# -----
# [TIME: 10000] [Direct] A=ff B=ff S1S0=00 Cin=1 | F=ff Cout=1
# => PASS: Expect: ff (255), DUT: ff (255)
# -----
# [TIME: 11000] [Direct] A=ff B=ff S1S0=01 Cin=0 | F=ff Cout=0
# => PASS: Expect: ff (255), DUT: ff (255)
# -----
# [TIME: 12000] [Direct] A=ff B=ff S1S0=01 Cin=1 | F=00 Cout=1
# => PASS: Expect: 00 (0), DUT: 00 (0)
# -----
# [TIME: 13000] [Direct] A=ff B=ff S1S0=10 Cin=0 | F=00 Cout=0
# => PASS: Expect: 00 (0), DUT: 00 (0)
# -----
# [TIME: 14000] [Direct] A=ff B=ff S1S0=10 Cin=1 | F=01 Cout=0
# => PASS: Expect: 01 (1), DUT: 01 (1)
# -----

```

```

# [TIME: 15000] [Direct] A=ff B=ff S1S0=11 Cin=0 | F=ff Cout=0
# => PASS: Expect: ff (255), DUT: ff (255)
# -----
# [TIME: 16000] [Direct] A=ff B=ff S1S0=11 Cin=1 | F=00 Cout=1
# => PASS: Expect: 00 (0), DUT: 00 (0)
# -----
# [TIME: 17000] [Direct] A=ff B=01 S1S0=00 Cin=0 | F=00 Cout=1
# => PASS: Expect: 00 (0), DUT: 00 (0)
# -----
# [TIME: 18000] [Direct] A=ff B=01 S1S0=00 Cin=1 | F=01 Cout=1
# => PASS: Expect: 01 (1), DUT: 01 (1)
# -----
# [TIME: 19000] [Direct] A=ff B=01 S1S0=01 Cin=0 | F=ff Cout=0
# => PASS: Expect: ff (255), DUT: ff (255)
# -----
# [TIME: 20000] [Direct] A=ff B=01 S1S0=01 Cin=1 | F=00 Cout=1
# => PASS: Expect: 00 (0), DUT: 00 (0)
# -----
# [TIME: 21000] [Direct] A=ff B=01 S1S0=10 Cin=0 | F=fe Cout=0
# => PASS: Expect: fe (254), DUT: fe (254)
# -----
# [TIME: 22000] [Direct] A=ff B=01 S1S0=10 Cin=1 | F=ff Cout=0
# => PASS: Expect: ff (255), DUT: ff (255)
# -----
# [TIME: 23000] [Direct] A=ff B=01 S1S0=11 Cin=0 | F=fd Cout=1
# => PASS: Expect: fd (253), DUT: fd (253)
# -----
# [TIME: 24000] [Direct] A=ff B=01 S1S0=11 Cin=1 | F=fe Cout=1
# => PASS: Expect: fe (254), DUT: fe (254)
# -----
# [TIME: 25000] [Direct] A=80 B=80 S1S0=00 Cin=0 | F=00 Cout=1
# => PASS: Expect: 00 (0), DUT: 00 (0)
# -----
# [TIME: 26000] [Direct] A=80 B=80 S1S0=00 Cin=1 | F=01 Cout=1
# => PASS: Expect: 01 (1), DUT: 01 (1)
# -----
# [TIME: 27000] [Direct] A=80 B=80 S1S0=01 Cin=0 | F=80 Cout=0
# => PASS: Expect: 80 (128), DUT: 80 (128)
# -----
# [TIME: 28000] [Direct] A=80 B=80 S1S0=01 Cin=1 | F=81 Cout=0
# => PASS: Expect: 81 (129), DUT: 81 (129)
# -----
# [TIME: 29000] [Direct] A=80 B=80 S1S0=10 Cin=0 | F=7f Cout=0
# => PASS: Expect: 7f (127), DUT: 7f (127)
# -----
# [TIME: 30000] [Direct] A=80 B=80 S1S0=10 Cin=1 | F=80 Cout=0
# => PASS: Expect: 80 (128), DUT: 80 (128)
# -----
# [TIME: 31000] [Direct] A=80 B=80 S1S0=11 Cin=0 | F=ff Cout=0
# => PASS: Expect: ff (255), DUT: ff (255)
# -----
# [TIME: 32000] [Direct] A=80 B=80 S1S0=11 Cin=1 | F=00 Cout=1
# => PASS: Expect: 00 (0), DUT: 00 (0)
# -----
# [TIME: 33000] [Direct] A=aa B=55 S1S0=00 Cin=0 | F=ff Cout=0
# => PASS: Expect: ff (255), DUT: ff (255)
# -----
# [TIME: 34000] [Direct] A=aa B=55 S1S0=00 Cin=1 | F=00 Cout=1
# => PASS: Expect: 00 (0), DUT: 00 (0)
# -----
# [TIME: 35000] [Direct] A=aa B=55 S1S0=01 Cin=0 | F=aa Cout=0
# => PASS: Expect: aa (170), DUT: aa (170)
# -----
# [TIME: 36000] [Direct] A=aa B=55 S1S0=01 Cin=1 | F=ab Cout=0

```

```

# => PASS: Expect: ab (171), DUT: ab (171)
# -----
# [TIME: 37000] [Direct] A=aa B=55 S1S0=10 Cin=0 | F=aa Cout=0
# => PASS: Expect: aa (170), DUT: aa (170)
# -----
# [TIME: 38000] [Direct] A=aa B=55 S1S0=10 Cin=1 | F=ab Cout=0
# => PASS: Expect: ab (171), DUT: ab (171)
# -----
# [TIME: 39000] [Direct] A=aa B=55 S1S0=11 Cin=0 | F=54 Cout=1
# => PASS: Expect: 54 (84), DUT: 54 (84)
# -----
# [TIME: 40000] [Direct] A=aa B=55 S1S0=11 Cin=1 | F=55 Cout=1
# => PASS: Expect: 55 (85), DUT: 55 (85)
# -----
# [TIME: 41000] [Direct] A=55 B=aa S1S0=00 Cin=0 | F=ff Cout=0
# => PASS: Expect: ff (255), DUT: ff (255)
# -----
# [TIME: 42000] [Direct] A=55 B=aa S1S0=00 Cin=1 | F=00 Cout=1
# => PASS: Expect: 00 (0), DUT: 00 (0)
# -----
# [TIME: 43000] [Direct] A=55 B=aa S1S0=01 Cin=0 | F=55 Cout=0
# => PASS: Expect: 55 (85), DUT: 55 (85)
# -----
# [TIME: 44000] [Direct] A=55 B=aa S1S0=01 Cin=1 | F=56 Cout=0
# => PASS: Expect: 56 (86), DUT: 56 (86)
# -----
# [TIME: 45000] [Direct] A=55 B=aa S1S0=10 Cin=0 | F=55 Cout=0
# => PASS: Expect: 55 (85), DUT: 55 (85)
# -----
# [TIME: 46000] [Direct] A=55 B=aa S1S0=10 Cin=1 | F=56 Cout=0
# => PASS: Expect: 56 (86), DUT: 56 (86)
# -----
# [TIME: 47000] [Direct] A=55 B=aa S1S0=11 Cin=0 | F=aa Cout=0
# => PASS: Expect: aa (170), DUT: aa (170)
# -----
# [TIME: 48000] [Direct] A=55 B=aa S1S0=11 Cin=1 | F=ab Cout=0
# => PASS: Expect: ab (171), DUT: ab (171)
# -----
# [TIME: 49000] [Direct] A=0f B=f0 S1S0=00 Cin=0 | F=ff Cout=0
# => PASS: Expect: ff (255), DUT: ff (255)
# -----
# [TIME: 50000] [Direct] A=0f B=f0 S1S0=00 Cin=1 | F=00 Cout=1
# => PASS: Expect: 00 (0), DUT: 00 (0)
# -----
# [TIME: 51000] [Direct] A=0f B=f0 S1S0=01 Cin=0 | F=0f Cout=0
# => PASS: Expect: 0f (15), DUT: 0f (15)
# -----
# [TIME: 52000] [Direct] A=0f B=f0 S1S0=01 Cin=1 | F=10 Cout=0
# => PASS: Expect: 10 (16), DUT: 10 (16)
# -----
# [TIME: 53000] [Direct] A=0f B=f0 S1S0=10 Cin=0 | F=0f Cout=0
# => PASS: Expect: 0f (15), DUT: 0f (15)
# -----
# [TIME: 54000] [Direct] A=0f B=f0 S1S0=10 Cin=1 | F=10 Cout=0
# => PASS: Expect: 10 (16), DUT: 10 (16)
# -----
# [TIME: 55000] [Direct] A=0f B=f0 S1S0=11 Cin=0 | F=1e Cout=0
# => PASS: Expect: 1e (30), DUT: 1e (30)
# -----
# [TIME: 56000] [Direct] A=0f B=f0 S1S0=11 Cin=1 | F=1f Cout=0
# => PASS: Expect: 1f (31), DUT: 1f (31)
# -----

```

Listing 23: Kết quả mô phỏng Directed Test.

- TestCase1: test trường hợp toàn zero. Để xác nhận là mô phỏng ổn, đảm bảo mạch không tạo nhiễu hay glitch.

```

1      task automatic run_test();
2      logic [7:0] edgeA [0:6];
3      logic [7:0] edgeB [0:6];
4      int idx;
5      int s1, s0, c;
6      int rctl;
7      begin
8          total_tests = 0;
9          errors = 0;
10         test_count = 0;
11         test_pass = 0;
12         $display("\n===== ZERO CASE TEST =====\n");
13         apply_and_check(8'h00, 8'h00, 0, 0, 0, "Zero");
14     end
15 endtask
16

```

Listing 24: Thực hiện kiểm tra zero case.

Kết quả

```

# ===== ZERO CASE TEST =====
#
# [TIME: 57000] [Zero] A=00 B=00 S1S0=00 Cin=0 | F=00 Cout=0
# => PASS: Expect: 00 (0), DUT: 00 (0)
# -----

```

Listing 25: Kết quả mô phỏng Directed Test.

- TestCase2: Random Test sau khi chạy Directed test và zero case trước đó: Sinh ngẫu nhiên giá trị A , B , S_1 , S_0 , C_{in} bằng hàm \$urandom(seed). Dừng khi đạt 100 mẫu test tổng cộng.

```

1      task automatic run_test();
2      logic [7:0] edgeA [0:6];
3      logic [7:0] edgeB [0:6];
4      int idx;
5      int s1, s0, c;
6      int rctl;
7      begin
8          total_tests = 0;
9          errors = 0;
10         test_count = 0;
11         test_pass = 0;
12         seed = 32'hCAFEBABE;
13         while (total_tests < 100) begin
14             A = $urandom(seed) & 8'hFF;
15             B = $urandom(seed + 1) & 8'hFF;
16             rctl = $urandom(seed + 2) % 8;

```

```

17         S1 = (rctl >> 2) & 1;
18         S0 = (rctl >> 1) & 1;
19         Cin = rctl & 1;
20         #1;
21         apply_and_check(A, B, S1, S0, Cin, "Random");
22         seed = seed + 12345;
23     end
24 end
25 endtask
26

```

Listing 26: Thực hiện Random Test.

Kết quả

```

# ===== STARTING RANDOM TESTS =====
#
# [TIME: 59000] [Random] A=07 B=5b S1S0=11 Cin=1 | F=ac Cout=0
# => PASS: Expect: ac (172), DUT: ac (172)
# -----
# [TIME: 61000] [Random] A=46 B=63 S1S0=00 Cin=1 | F=aa Cout=0
# => PASS: Expect: aa (170), DUT: aa (170)
# -----
# [TIME: 63000] [Random] A=07 B=a0 S1S0=00 Cin=0 | F=a7 Cout=0
# => PASS: Expect: a7 (167), DUT: a7 (167)
# ...
# [TIME: 141000] [Random] A=8a B=79 S1S0=11 Cin=1 | F=11 Cout=1
# => PASS: Expect: 11 (17), DUT: 11 (17)
# -----
# [TIME: 143000] [Random] A=1f B=93 S1S0=11 Cin=1 | F=8c Cout=0
# => PASS: Expect: 8c (140), DUT: 8c (140)
# -----

```

Listing 27: Kết quả mô phỏng Directed Test.

- Kết quả tổng kết.

```

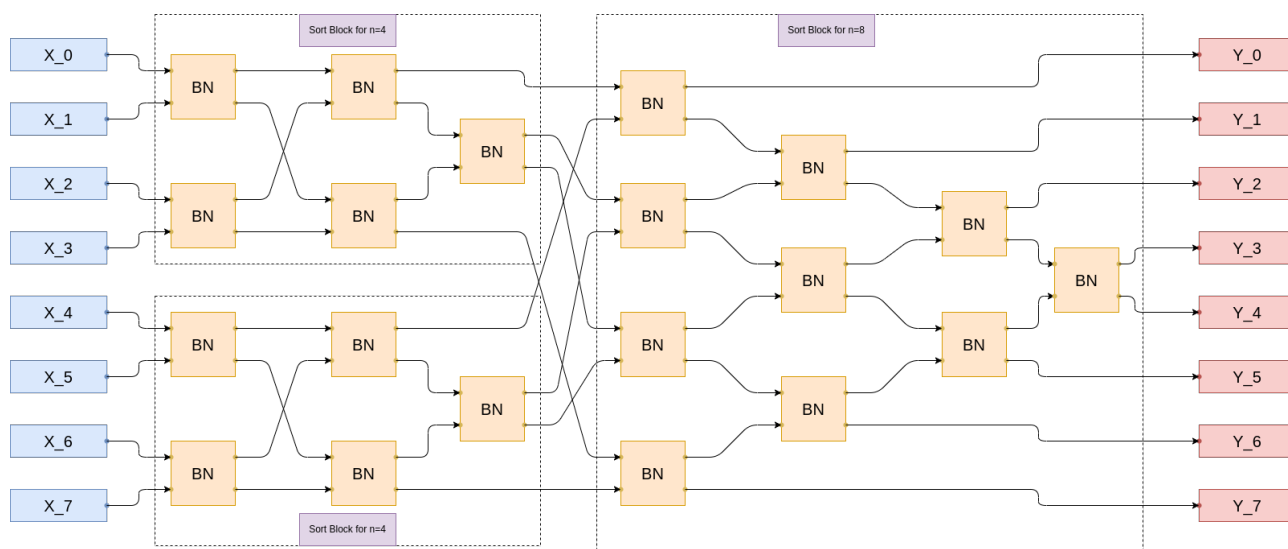
# =====
# ===== TEST SUMMARY =====
# Total test cases : 100
# Passed           : 100
# Failed           : 0
# Pass rate        : 100.00 %
# =====

```

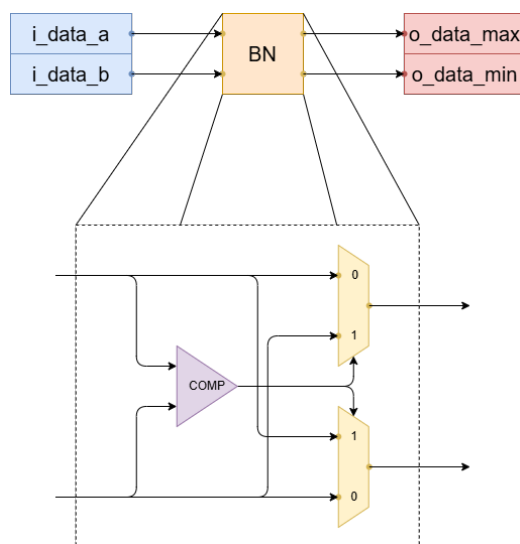
Listing 28: Kết quả của tổng kết của bài test.

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, \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 Merger 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.

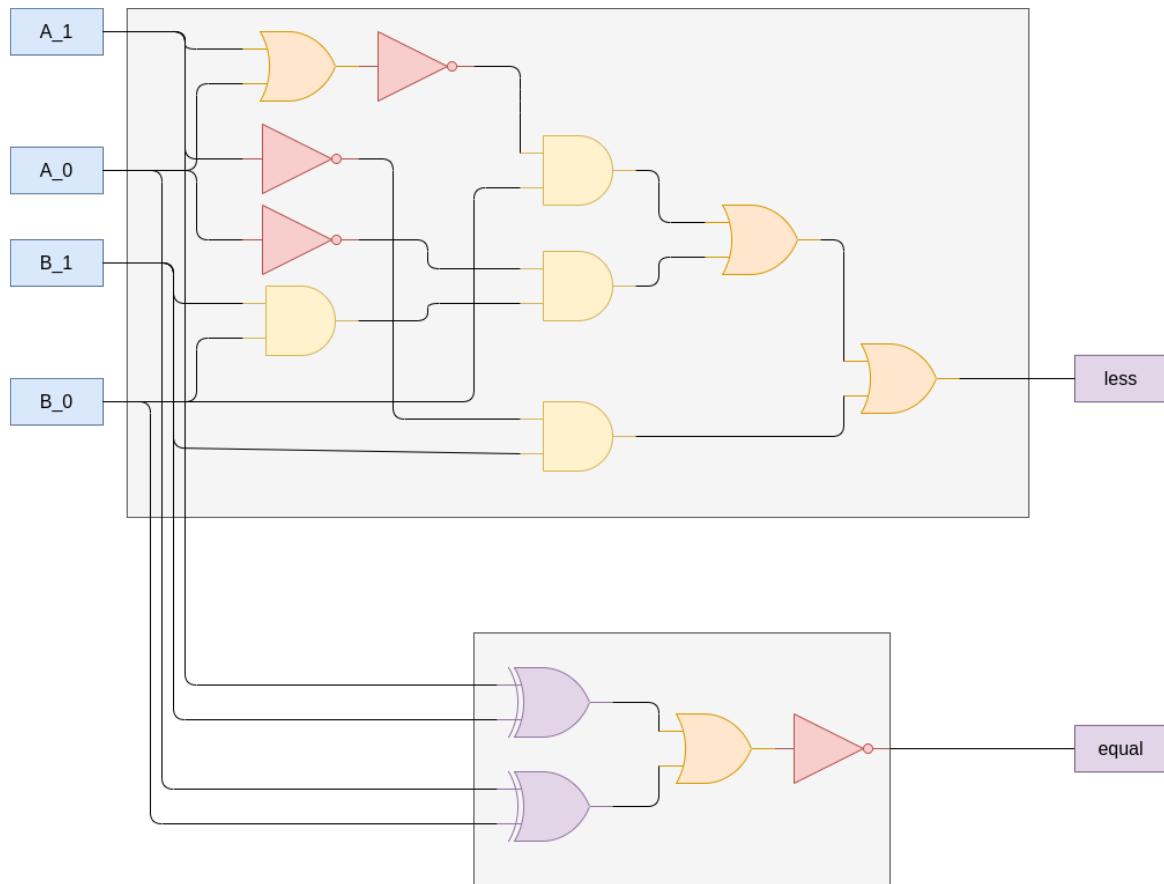
- Bộ so sánh 8bit

+ 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

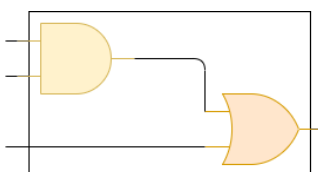
Bảng 5: Bảng sự thật cho bộ so sánh 2-bit $A < B$.

Từ bộ bảng sự thật 5, ta rút gọn kìa K về dạng như sau:

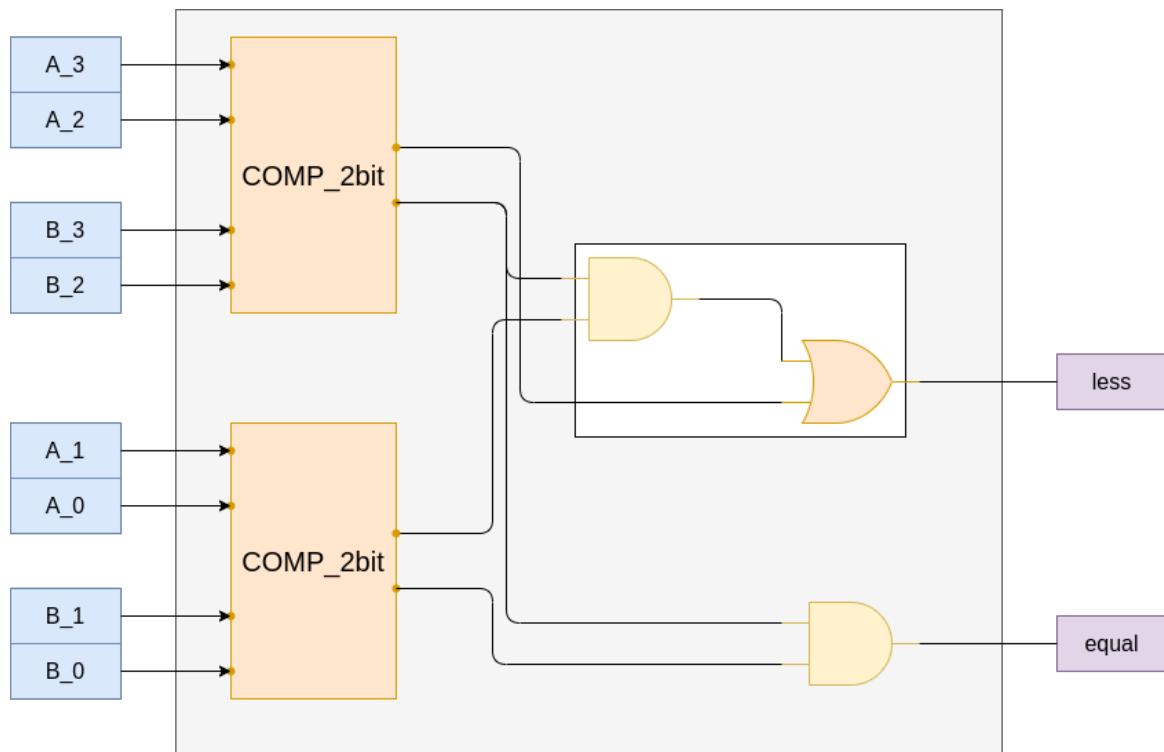


Hình 15: Sơ đồ logic của bộ Comparator 2-bit.

+ Triển khai bộ so sánh 4-bit



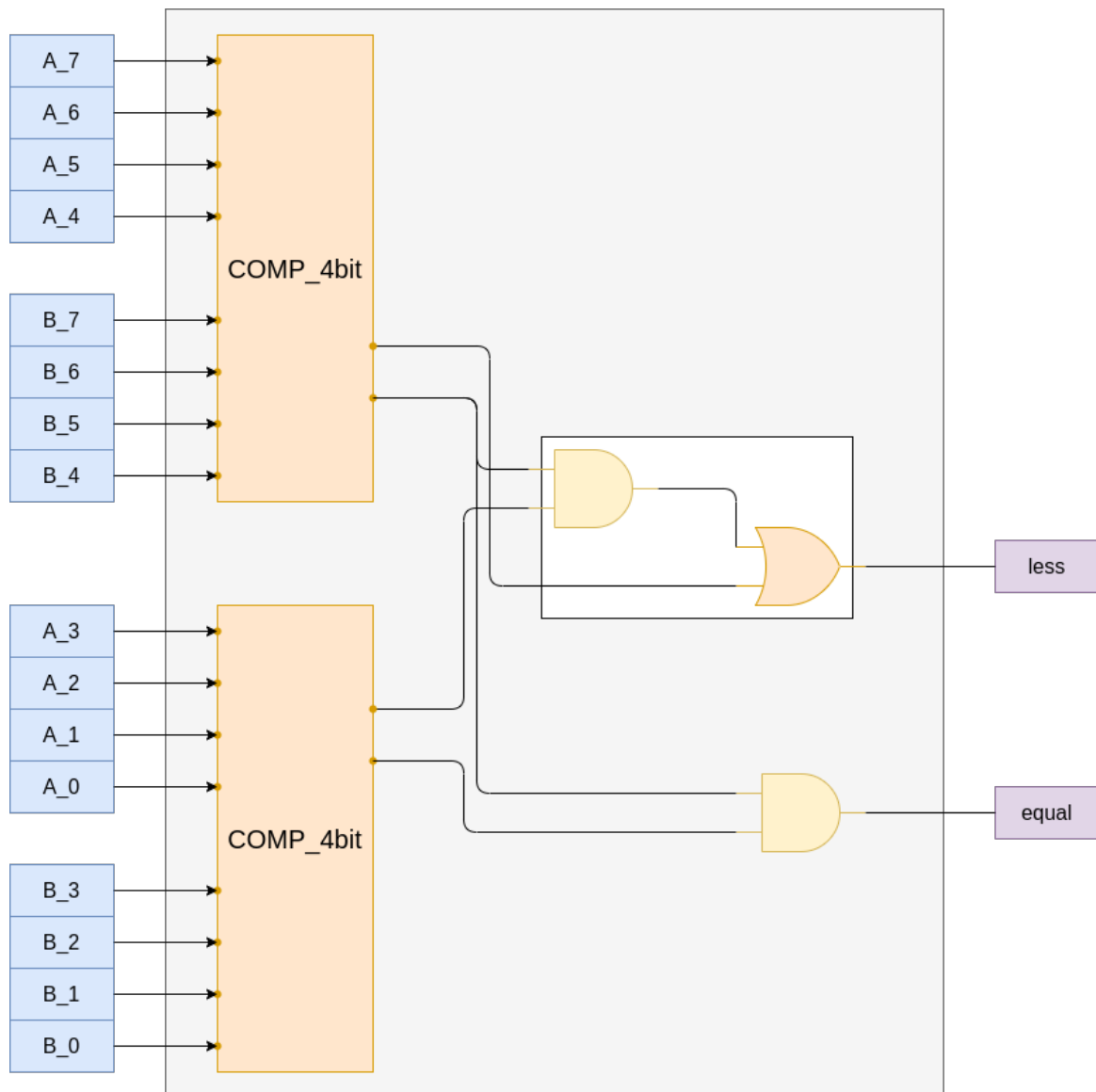
Sử dụng bộ lan truyền dấu so sánh trên, với nếu các bit trên có xuất hiện bit less thì ngõ ra lan truyền bit less, còn nếu bit thấp thì nếu tầng trên bằng nhau và các tầng thấp có bit less ngõ ra lan truyền bit less.



Hình 16: Sơ đồ logic của bộ Comparator 4-bit.

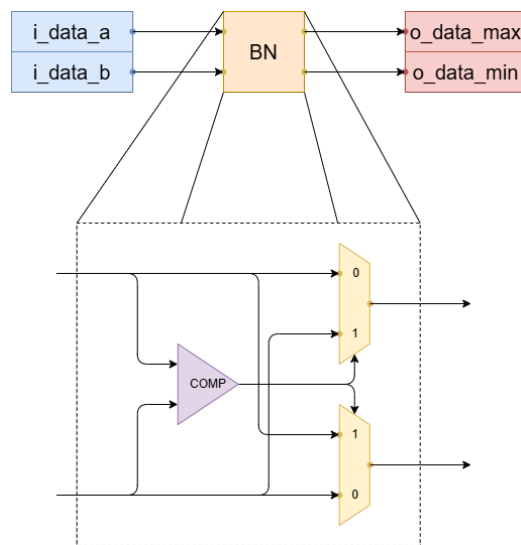
+ Triển khai bộ so sánh 8-bit

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



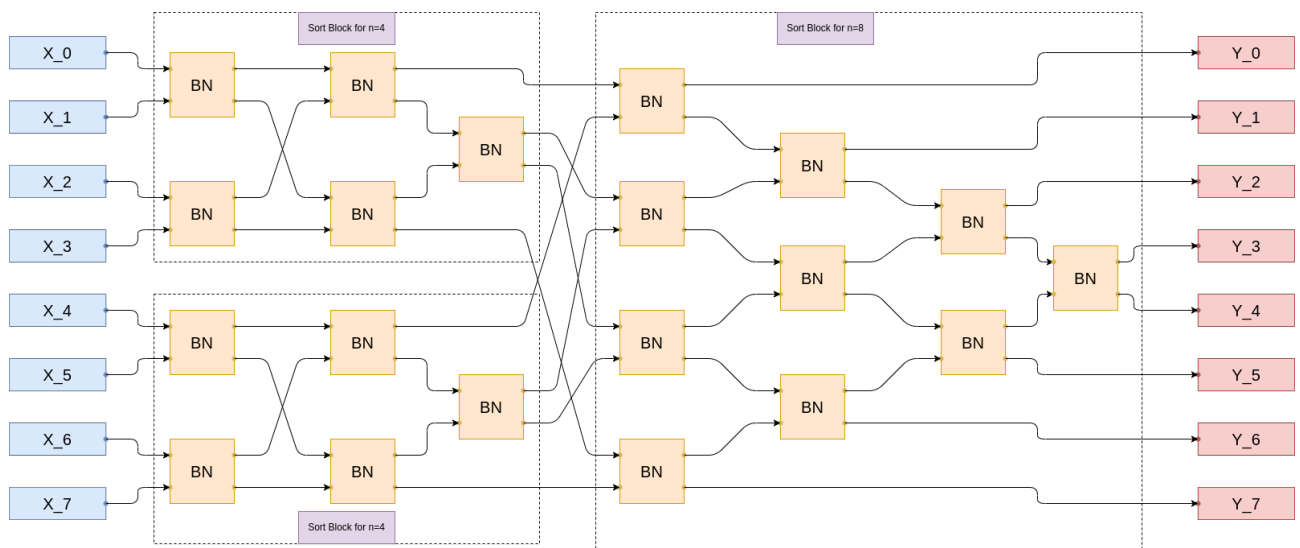
Hình 17: Sơ đồ logic của bộ Comparator 8-bit.

- Bộ Swap 8bit



Hình 18: Bộ so sánh và swap giá trị trong Bitonic Merger Sort.

- Bitonic Merger Sort



Hình 19: Bộ sắp xếp Bitonic Merger 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] & i_data_b
      //                    [0]) | (~i_data_a[1] & i_data_b[1]);

```

```

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

```

Listing 29: 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 30: 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 31: 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_UNIT (
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 32: Chương trình mô tả một đơn vị của bộ Bitonic Merger 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 33: Chương trình mô tả bộ Bitonic Merger 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 wire [SIZE_DATA-1:0] w_data_min_3_0, w_data_max_3_0;
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 Compare_and_Swap_unit #(
122     .IS_ASC         (IS_ASC),

```



```

123     .SIZE_DATA      (SIZE_DATA)
124 ) CAS_3_0 (
125     .i_data_a        (w_data_min_2_0),
126     .i_data_b        (w_data_max_2_1),
127     .o_data_max      (w_data_max_3_0),
128     .o_data_min      (w_data_min_3_0)
129 );
130
131 assign o_data_0 = w_data_max_0_0;
132 assign o_data_1 = w_data_max_1_0;
133 assign o_data_2 = w_data_max_2_0;
134 assign o_data_3 = w_data_max_3_0;
135 assign o_data_4 = w_data_min_3_0;
136 assign o_data_5 = w_data_min_2_1;
137 assign o_data_6 = w_data_min_1_2;
138 assign o_data_7 = w_data_min_0_3;
139 endmodule

```

Listing 34: Chương trình mô tả bộ Bitonic Merger 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
31     .o_data_0    (w_o_data[0]),
32     .o_data_1    (w_o_data[1]),
33     .o_data_2    (w_o_data[2]),
34     .o_data_3    (w_o_data[3])
35 );
36 Bitonic_Block4 #(
37     .IS_ASC    (IS_ASC),
38     .SIZE_DATA (SIZE_DATA)
39 ) BN_4_UNIT_1 (
40     .i_data_0    (w_i_data[4]),
41     .i_data_1    (w_i_data[5]),
42     .i_data_2    (w_i_data[6]),
43     .i_data_3    (w_i_data[7]),

```

```

44
45     .o_data_0      (w_0_data[4]),
46     .o_data_1      (w_0_data[5]),
47     .o_data_2      (w_0_data[6]),
48     .o_data_3      (w_0_data[7])
49 );
50
51 Bitonic_Block8 #(
52     .IS_ASC        (IS_ASC),
53     .SIZE_DATA      (SIZE_DATA)
54 ) BN_8_UNIT_0 (
55     .i_data_0       (w_0_data[0]),
56     .i_data_1       (w_0_data[4]),
57     .i_data_2       (w_0_data[1]),
58     .i_data_3       (w_0_data[5]),
59     .i_data_4       (w_0_data[2]),
60     .i_data_5       (w_0_data[6]),
61     .i_data_6       (w_0_data[3]),
62     .i_data_7       (w_0_data[7]),
63
64     .o_data_0       (w_sorted[0]),
65     .o_data_1       (w_sorted[1]),
66     .o_data_2       (w_sorted[2]),
67     .o_data_3       (w_sorted[3]),
68     .o_data_4       (w_sorted[4]),
69     .o_data_5       (w_sorted[5]),
70     .o_data_6       (w_sorted[6]),
71     .o_data_7       (w_sorted[7])
72 );
73
74 generate
75     for (i = 0; i < NUM_ELEM; i++) begin : PACK_OUTPUT
76         assign o_sorted[i*SIZE_DATA +: SIZE_DATA] = w_sorted[i];
77     end
78 endgenerate
79
80 endmodule

```

Listing 35: Chương trình mô tả bộ Bitonic Merger 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 36: Giải thuật chứng minh bộ Bitonic Merger Sort 8 phần tử.

Nhóm em sử dụng giải thuật của Bubble Sort gần giống với giải thuật parallel của Bitonic Merger Sort, và một task để hiển thị kết quả một cách trực quan.

```

1  task automatic Display_Result(
2      string                                t_type,
3      input logic [SIZE_DATA*NUM_ELEM-1:0] t_arr
4  );
5      logic [SIZE_DATA-1:0] t_t_arr    [0:NUM_ELEM-1];
6      int i, j;
7      begin
8          for (i = 0; i < NUM_ELEM; i++) begin
9              t_t_arr[i] = t_arr[i*SIZE_DATA +: SIZE_DATA];
10         end
11
12         for (j = 0; j < NUM_ELEM; j++) begin
13             $display("%s[%d] \t= %8h (%d)", t_type, j, t_t_arr[j], t_t_arr[j]);
14         end
15     end
16 endtask //automatic

```

Thực hiện với bộ sắp xếp tăng dần

- TestCase0: Thực hiện test với 8 phần tử đầu vào đều là 8'b0.

```

1      repeat (1) begin
2          @(posedge i_clk);
3          #1;
4          i_data = $urandom;
5          i_data = '0;
6          $display("\n");
7          Display_Result("Input", i_data);
8          @(negedge i_clk);
9          #1;
10         $display("\n");

```

```

11         Display_Result("Output", o_data);
12         $display("[TIME: %5t] [%s] i_data = %h (%d) \t| o_data = %h (%d)", $time, "Zero", i_data,
13         i_data, o_data, o_data);
14         $display("=> %4s: Expect: %8h, DUT: %8h ", (f_ARR_sorted(IS_ASC, i_data) == o_data) ? "PASS"
15         : "FAIL", f_ARR_sorted(IS_ASC, i_data), o_data);
16         test_count = test_count + 1;
17         test_pass = (f_ARR_sorted(IS_ASC, i_data) == o_data) ? test_pass + 1 : test_pass;
18     end

```

Listing 37: Test trường hợp các đầu vào đều là không.

Kết quả

```

===ACSENDING===

Input[      0] = 00000000 ( 0)
Input[      1] = 00000000 ( 0)
Input[      2] = 00000000 ( 0)
Input[      3] = 00000000 ( 0)
Input[      4] = 00000000 ( 0)
Input[      5] = 00000000 ( 0)
Input[      6] = 00000000 ( 0)
Input[      7] = 00000000 ( 0)

Output[     0] = 00000000 ( 0)
Output[     1] = 00000000 ( 0)
Output[     2] = 00000000 ( 0)
Output[     3] = 00000000 ( 0)
Output[     4] = 00000000 ( 0)
Output[     5] = 00000000 ( 0)
Output[     6] = 00000000 ( 0)
Output[     7] = 00000000 ( 0)

[TIME: 41000] [Zero] i_data = 0000000000000000 ( 0) | o_data =
0000000000000000 ( 0)
=> PASS: Expect: 00000000, DUT: 00000000

```

Listing 38: Kết quả khi cho 8 phần tử ngõ vào đều là không.

- Testcase1: Thực hiện và hiển thị một trường hợp để kiểm tra các hoạt động.

```

1     repeat (1) begin
2         @(posedge i_clk);
3         #1;
4         i_data = $urandom;
5         $display("\n");
6         Display_Result("Input", i_data);
7         @(negedge i_clk);
8         #1;
9         $display("\n");
10        Display_Result("Output", o_data);
11        $display("[TIME: %5t] [%s] i_data = %h (%d) \t| o_data = %h (%d)", $time, "Random", i_data,
12        i_data, o_data, o_data);
13        $display("=> %4s: Expect: %8h, DUT: %8h ", (f_ARR_sorted(IS_ASC, i_data) == o_data) ? "PASS"
14        : "FAIL", f_ARR_sorted(IS_ASC, i_data), o_data);
15        test_count = test_count + 1;
16        test_pass = (f_ARR_sorted(IS_ASC, i_data) == o_data) ? test_pass + 1 : test_pass;
17    end

```

Listing 39: Test 1 trường hợp ngẫu nhiên.

Kết quả

```

===ACSENDING===

Input [      0] = 0000009c (156)
Input [      1] = 0000005d ( 93)
Input [      2] = 0000001e ( 30)
Input [      3] = 0000000b ( 11)
Input [      4] = 00000000 (  0)
Input [      5] = 00000000 (  0)
Input [      6] = 00000000 (  0)
Input [      7] = 00000000 (  0)

Output [      0] = 00000000 (  0)
Output [      1] = 00000000 (  0)
Output [      2] = 00000000 (  0)
Output [      3] = 00000000 (  0)
Output [      4] = 0000000b ( 11)
Output [      5] = 0000001e ( 30)
Output [      6] = 0000005d ( 93)
Output [      7] = 0000009c (156)
[TIME: 51000] [Random] i_data = 000000000b1e5d9c (          186539420) | o_data = 9
c5d1e0b000000000 (11267194875344322560)
=> PASS: Expect: 9c5d1e0b00000000, DUT: 9c5d1e0b00000000

```

Listing 40: Kết quả khi cho 1 trường hợp ngẫu nhiên.

- Testcase1: Thực hiện và 100 trường hợp để kiểm tra các hoạt động.

```

1  repeat (100) begin
2      @(posedge i_clk);
3      #1;
4      i_data = $urandom_range(0, 2**((SIZE_DATA*NUM_ELEM)-1));
5      @(negedge i_clk);
6      #1;
7      $display("[TIME: %5t] [%s] i_data = %h (%d) \t| o_data = %h (%d)", $time, "Random", i_data,
8      i_data, o_data, o_data);
9      $display("=> %4s: Expect: %8h, DUT: %8h ", (f_ARR_sorted(IS_ASC, i_data) == o_data) ? "PASS"
10     : "FAIL", f_ARR_sorted(IS_ASC, i_data), o_data);
11     test_count = test_count + 1;
12     test_pass = (f_ARR_sorted(IS_ASC, i_data) == o_data) ? test_pass + 1 : test_pass;
13 end

```

Listing 41: Test 100 trường hợp ngẫu nhiên.

Kết quả

```

===ACSENDING===

[TIME: 61000] [Random] i_data = 000000009efdd502 (          2667435266) | o_data =
fdd59e0200000000 (18290699193062260736)
=> PASS: Expect: fdd59e0200000000, DUT: fdd59e0200000000
[TIME: 71000] [Random] i_data = 0000000048c2e0e4 (          1220731108) | o_data =
e4e0c24800000000 (16492395449924190208)

```

```

=> PASS: Expect: e4e0c24800000000, DUT: e4e0c24800000000
[TIME: 81000] [Random] i_data = 0000000092178378 ( 2451014520) | o_data =
9283781700000000 (10557413991666155520)
=> PASS: Expect: 9283781700000000, DUT: 9283781700000000
[TIME: 91000] [Random] i_data = 00000000bbd58ebc ( 3151335100) | o_data =
d5bcb8e000000000 (15401391044260003840)
=> PASS: Expect: d5bcb8e000000000, DUT: d5bcb8e000000000
...
[TIME: 1031000] [Random] i_data = 0000000011dab302 ( 299545346) | o_data =
dab3110200000000 (15758958221387104256)
=> PASS: Expect: dab3110200000000, DUT: dab3110200000000
[TIME: 1041000] [Random] i_data = 00000000ca972bfb ( 3398904827) | o_data =
fbca972b00000000 (18143480259754852352)
=> PASS: Expect: fbca972b00000000, DUT: fbca972b00000000
[TIME: 1051000] [Random] i_data = 000000009f2015e8 ( 2669680104) | o_data =
e89f201500000000 (16762151612662677504)
=> PASS: Expect: e89f201500000000, DUT: e89f201500000000

```

Listing 42: Kết quả khi cho 100 trường hợp ngẫu nhiên.

- Kết quả tổng kết.

```

=====
=====TEST SUMMARY=====
Total test cases:    102
Passed              :    102
Failed              :      0
Pass rate           : 100.00%
=====

```

Listing 43: Kết quả của tổng kết của bài test.

Thực hiện với bộ sắp xếp giảm dần

- TestCase0: Thực hiện test với 8 phần tử đầu vào đều là 8'b0.

```

===DESCENDING===

Input[      0] = 00000000 ( 0)
Input[      1] = 00000000 ( 0)
Input[      2] = 00000000 ( 0)
Input[      3] = 00000000 ( 0)
Input[      4] = 00000000 ( 0)
Input[      5] = 00000000 ( 0)
Input[      6] = 00000000 ( 0)
Input[      7] = 00000000 ( 0)

Output[      0] = 00000000 ( 0)
Output[      1] = 00000000 ( 0)
Output[      2] = 00000000 ( 0)
Output[      3] = 00000000 ( 0)
Output[      4] = 00000000 ( 0)
Output[      5] = 00000000 ( 0)
Output[      6] = 00000000 ( 0)
Output[      7] = 00000000 ( 0)
[TIME: 41000] [Zero] i_data = 000000000000000000 ( 0) | o_data =
0000000000000000 ( 0)

```

```
=> PASS: Expect: 00000000, DUT: 00000000
```

Listing 44: Kết quả khi cho 8 phần tử ngõ vào đều là không.

- Testcase1: Thực hiện và hiển thị một trường hợp để kiểm tra các hoạt động.

```
===DESCENDING===

Input[      0] = 0000009c (156)
Input[      1] = 0000005d ( 93)
Input[      2] = 0000001e ( 30)
Input[      3] = 0000000b ( 11)
Input[      4] = 00000000 (  0)
Input[      5] = 00000000 (  0)
Input[      6] = 00000000 (  0)
Input[      7] = 00000000 (  0)

Output[      0] = 0000009c (156)
Output[      1] = 0000005d ( 93)
Output[      2] = 0000001e ( 30)
Output[      3] = 0000000b ( 11)
Output[      4] = 00000000 (  0)
Output[      5] = 00000000 (  0)
Output[      6] = 00000000 (  0)
Output[      7] = 00000000 (  0)
[TIME: 51000] [Random] i_data = 00000000b1e5d9c (      186539420) | o_data = 000000000
b1e5d9c (      186539420)
=> PASS: Expect: 0b1e5d9c, DUT: 0b1e5d9c
```

Listing 45: Kết quả khi cho 1 trường hợp ngẫu nhiên.

- Testcase1: Thực hiện và 100 trường hợp để kiểm tra các hoạt động.

```
===DESCENDING===

[TIME: 61000] [Random] i_data = 000000009efdd502 (      2667435266) | o_data = 00000000029
ed5fd (      43963901)
=> PASS: Expect: 029ed5fd, DUT: 029ed5fd
[TIME: 71000] [Random] i_data = 0000000048c2e0e4 (      1220731108) | o_data = 00000000048
c2e0e4 (      1220731108)
=> PASS: Expect: 48c2e0e4, DUT: 48c2e0e4
[TIME: 81000] [Random] i_data = 0000000092178378 (      2451014520) | o_data =
0000000017788392 (      393773970)
=> PASS: Expect: 17788392, DUT: 17788392
...
[TIME: 1031000] [Random] i_data = 0000000011dab302 (      299545346) | o_data =
000000000211b3da (      34714586)
=> PASS: Expect: 0211b3da, DUT: 0211b3da
[TIME: 1041000] [Random] i_data = 00000000ca972bfb (      3398904827) | o_data = 0000000002
b97cafb (      731368187)
=> PASS: Expect: 2b97cafb, DUT: 2b97cafb
[TIME: 1051000] [Random] i_data = 000000009f2015e8 (      2669680104) | o_data =
0000000015209fe8 (      354459624)
=> PASS: Expect: 15209fe8, DUT: 15209fe8
```

Listing 46: Kết quả khi cho 100 trường hợp ngẫu nhiên.

- Kết quả tổng kết.

```
=====
=====TEST SUMMARY=====
Total test cases:      102
Passed                :    102
Failed                :         0
Pass rate              : 100.00%
=====
```

Listing 47: Kết quả của tổng kết của bài test.