TRƯỜNG ĐẠI HỌC BÁCH KHOA TP. HỒ CHÍ MINH

Khoa Điện – Điện tử

Bộ môn Điện tử

_____***_____

BÁO CÁO CẤU TRÚC MÁY TÍNH

LAB 2: THỰC HIỆN THIẾT KẾ ALU

GVHD: Trần Hoàng Linh

Sinh viên thực hiện: Đinh Thế Bảo

MSSV: 1510152

Tp. Hồ Chí Minh, ngày 20 tháng 11 năm 2019

I. Mục đích

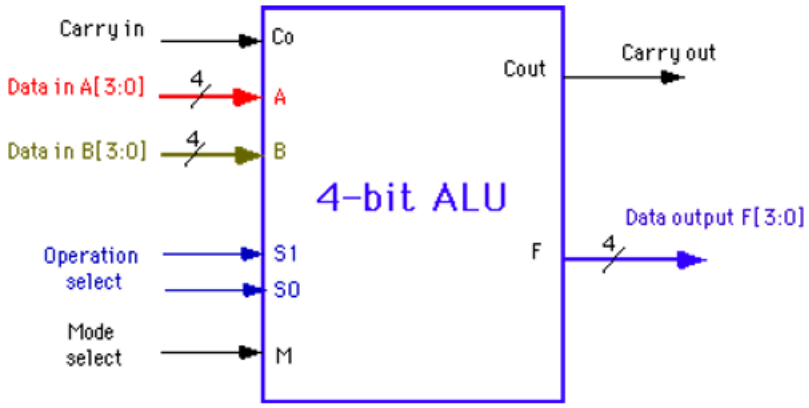
Thiết kế một bộ ALU đơn giản, đáp ứng được những yêu cầu:

- Độ dài toán hạng là 4 bit,
- Các ngõ nhập function-select gồm có: M, S0 và S1,
- Có thể thực hiện các tác vụ:

M	S1	S0	Chức năng	Tác vụ
0	0	0	Ai.Bi	AND
0	0	1	Ai + Bi	OR
0	1	0	Ai (+) Bi	XOR
0	1	1	~(Ai (+) Bi)	XNOR
1	0	0	A + C0	Cộng A với Carry
1	0	1	A+B+C0	Cộng A, B và Carry
1	1	0	A+B'+C0	Cộng A với bù B và Carry
1	1	1	A'+B+C0	Cộng B với bù A và Carry

Hình 1.1: Các chức năng của khối ALU

Sơ đồ khối của ALU 4 bit:



Hình 1.2 Sơ đồ khối ALU

II. Phương pháp thực hiện

2.1 Lựa chọn phương án thực hiện

Ta sẽ sử dụng nguyên tắc "chia để trị" để thực hiện xây dựng bộ ALU 4 bit. Tại đây, ta sẽ chia chương trình gốc thành nhiều chương trình con nhỏ hơn được gọi là module, mỗi module sẽ thực hiện một chức năng riêng biệt. Sau khi xây dựng xong các module, ta chỉ cần ghép chúng lại với nhau sẽ được chương trình ta cần xây dựng.

Mặc dù việc chia nhỏ chương trình thành các module này sẽ khiến thời gian xử lý dữ liệu dài hơn việc lập trình từ trên xuống dưới. Tuy nhiên bằng cách này, ta có thể giảm bớt độ phức tạp của việc lập trình, dễ dàng kiểm tra, bảo trì đồng thời, khi xảy ra lỗi, bằng cách kiểm tra kết quả đầu ra của từng module, ta có thể dễ dàng tìm ra vị trí lỗi.

Áp dụng vào Lab này, ta có thể nhận thấy ALU 4 bit có thể được tạo ra bằng cách ghép 4 bộ ALU 1 bit lại với nhau.

Để thiết kế ra 1 bộ ALU 1 bit, ta thường có 2 cách:

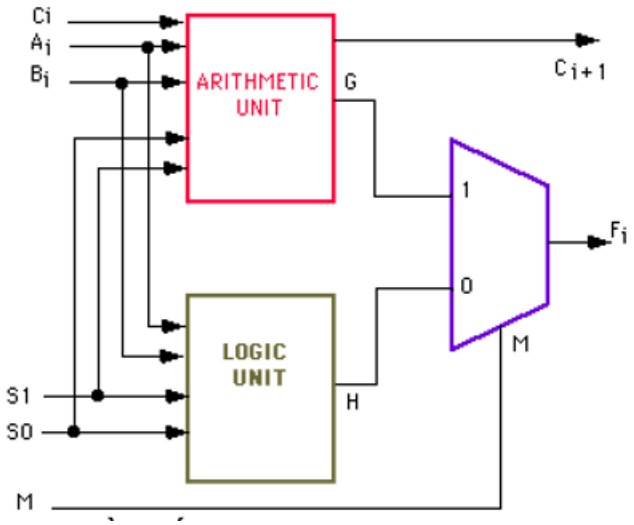
Cách 1: dùng bảng sự thật, xác định dữ liệu ngõ ra theo dữ liệu ngõ vào, từ đó thực hiện thiết kế.

Cách 2: ta nhận thấy bộ ALU thực hiện 2 chức năng riêng biệt là tính toán toán học và tính toán logic. Vậy nên ta có thể chia khối ALU thành 2 khối là AU (Arithmetic Unit) và LU (Logic Unit).

Với cách 1 bộ ALU sẽ thực hiện phép toán nhanh hơn tuy nhiên lại mất nhiều thời gian thiết kế hơn. Còn cách 2 cho phép giảm thời gian thiết kế nhưng đổi lại tốc độ tính toán sẽ chậm hơn.

Trong bài LAB này, em sẽ dùng phương pháp số 2 để thiết kế bộ ALU 1 bit.

Sơ đồ thiết kế ALU 1 bit phân theo chức năng:



Hình 2.1: Sơ đồ khối của bộ ALU 1 bit chia theo chức năng

2.2 Thực hiện thiết kế bộ ALU 1 bit:

2.2.1 Thiết kế bộ Arithmetic Unit

Dựa vào bảng chức năng, tại hình 1.1, ta có thể nhận thấy khối AU sẽ thực hiện chức năng thực hiện phép cộng 1 bit có cờ nhớ, vậy nên ta sẽ sử dụng bộ full adder 1 bit đã được thiết kế ở LAB 1 làm cơ sở để thiết kế nên khối AU.

Code AU:

```
module arithmetic_unit ( S0, S1, in_0, in_1, carry_in, G,carry_out);
    input S0, S1, in_0, in_1, carry_in;
    output reg G, carry_out;

    wire [1:0] s;
    reg g_1, carry_out_0;
    wire carry_out_1, carry_out_2, carry_out_3, carry_out_4;
    wire a, b, c, d,in_0_bu,in_1_bu;
    assign s[0] = S0;
    assign s[1] = S1;
    assign in_0_bu =~ in_0;
    assign in_1_bu =~ in_1;

    fullAdder fullAdder_1(  .A      (in_0),
                            .B      (0),
                            .Ci     (carry_in),
                            .Co     (carry_out_1),
                            .S  (a)
                        );

    fullAdder fullAdder_2(  .A      (in_0),
                            .B      (in_1),
                            .Ci     (carry_in),
                            .Co     (carry_out_2),
                            .S  (b)
                        );

    fullAdder fullAdder_3(  .A      (in_0),
                            .B      (in_1_bu),
                            .Ci     (carry_in),
                            .Co     (carry_out_3),
                            .S  (c)
                        );

    fullAdder fullAdder_4(  .A      (in_0_bu),
                            .B      (in_1),
                            .Ci     (carry_in),
                            .Co     (carry_out_4),
                            .S  (d)
```

```
                        );

    always @ (a or b or c or d or s)
    case (s)

        2'b00:begin
            G = a;
            carry_out = carry_out_1;
            end

        2'b01:begin
            G = b;
            carry_out =carry_out_2;
            end

        2'b10:begin
            G = c;
            carry_out = carry_out_3;
            end

        2'b11:begin
            G = d;
            carry_out = carry_out_4;
            end

        default:$display("Error in SEL");

    endcase

endmodule
```

Code test bench:

```
module testBenchAU;

    reg S0, S1, in_0, in_1, carry_in;
    wire G, carry_out;

    arithmetic_unit arithmetic_unit (   .S0         (S0),
                                        .S1         (S1),
                                        .in_0       (in_0),
                                        .in_1       (in_1),
                                        .carry_in   (carry_in),
                                        .G          (G),
                                        .carry_out  (carry_out)
                                        );
```

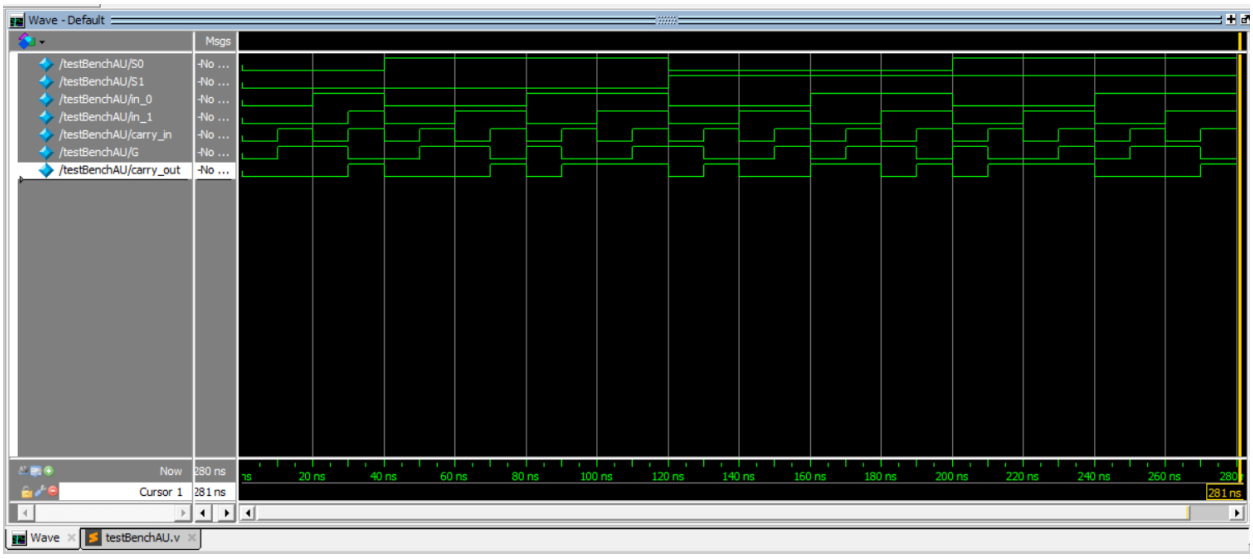
```
initial begin
    S1 = 1'b0;
    S0 = 1'b0;
    in_1 = 1'b0;
    in_0 = 1'b0;
    carry_in = 1'b0;
    #10
    S1 = 1'b0;
    S0 = 1'b0;
    in_1 = 1'b0;
    in_0 = 1'b0;
    carry_in = 1'b1;
    #10
    S1 = 1'b0;
    S0 = 1'b0;
    in_1 = 1'b0;
    in_0 = 1'b1;
    carry_in = 1'b0;
    #10
    S1 = 1'b0;
    S0 = 1'b0;
    in_1 = 1'b1;
    in_0 = 1'b1;
    carry_in = 1'b1;
    // S1 = 0, S0 = 1, A + B + Cin
        #10
    S1 = 1'b0;
    S0 = 1'b1;
    in_0 = 1'b0;
    in_1 = 1'b0;
    carry_in = 1'b0;
    #10
    S1 = 1'b0;
    S0 = 1'b1;
    in_0 = 1'b0;
    in_1 = 1'b0;
    carry_in = 1'b1;
    #10
    S1 = 1'b0;
    S0 = 1'b1;
    in_0 = 1'b0;
    in_1 = 1'b1;
    carry_in = 1'b0;
    #10
```

```
        S1 = 1'b0;
        S0 = 1'b1;
        in_0 = 1'b0;
        in_1 = 1'b1;
        carry_in = 1'b1;
        #10
        S1 = 1'b0;
        S0 = 1'b1;
        in_0 = 1'b1;
        in_1 = 1'b0;
        carry_in = 1'b0;
        #10
        S1 = 1'b0;
        S0 = 1'b1;
        in_0 = 1'b1;
        in_1 = 1'b0;
        carry_in = 1'b1;
        #10
        S1 = 1'b0;
        S0 = 1'b1;
        in_0 = 1'b1;
        in_1 = 1'b1;
        carry_in = 1'b0;
        #10
        S1 = 1'b0;
        S0 = 1'b1;
        in_0 = 1'b1;
        in_1 = 1'b1;
        carry_in = 1'b1;
        //S1 = 1, S0 = 0, A + B' + Cin
        #10
        S1 = 1'b1;
        S0 = 1'b0;
        in_0 = 1'b0;
        in_1 = 1'b0;
        carry_in = 1'b0;
        #10
        S1 = 1'b1;
        S0 = 1'b0;
        in_0 = 1'b0;
        in_1 = 1'b0;
        carry_in = 1'b1;
        #10
        S1 = 1'b1;
        S0 = 1'b0;
```

```
        in_0 = 1'b0;
        in_1 = 1'b1;
        carry_in = 1'b0;
        #10
        S1 = 1'b1;
        S0 = 1'b0;
        in_0 = 1'b0;
        in_1 = 1'b1;
        carry_in = 1'b1;
        #10
        S1 = 1'b1;
        S0 = 1'b0;
        in_0 = 1'b1;
        in_1 = 1'b0;
        carry_in = 1'b0;
        #10
        S1 = 1'b1;
        S0 = 1'b0;
        in_0 = 1'b1;
        in_1 = 1'b0;
        carry_in = 1'b1;
        #10
        S1 = 1'b1;
        S0 = 1'b0;
        in_0 = 1'b1;
        in_1 = 1'b1;
        carry_in = 1'b0;
        #10
        S1 = 1'b1;
        S0 = 1'b0;
        in_0 = 1'b1;
        in_1 = 1'b1;
        carry_in = 1'b1;
        //S1 = 1, S0 = 1, A' + B + Cin
        #10
        S1 = 1'b1;
        S0 = 1'b1;
        in_0 = 1'b0;
        in_1 = 1'b0;
        carry_in = 1'b0;
        #10
        S1 = 1'b1;
        S0 = 1'b1;
        in_0 = 1'b0;
        in_1 = 1'b0;
```

```
        carry_in = 1'b1;
        #10
        S1 = 1'b1;
        S0 = 1'b1;
        in_0 = 1'b0;
        in_1 = 1'b1;
        carry_in = 1'b0;
        #10
        S1 = 1'b1;
        S0 = 1'b1;
        in_0 = 1'b0;
        in_1 = 1'b1;
        carry_in = 1'b1;
        #10
        S1 = 1'b1;
        S0 = 1'b1;
        in_0 = 1'b1;
        in_1 = 1'b0;
        carry_in = 1'b0;
        #10
        S1 = 1'b1;
        S0 = 1'b1;
        in_0 = 1'b1;
        in_1 = 1'b0;
        carry_in = 1'b1;
        #10
        S1 = 1'b1;
        S0 = 1'b1;
        in_0 = 1'b1;
        in_1 = 1'b1;
        carry_in = 1'b0;
        #10
        S1 = 1'b1;
        S0 = 1'b1;
        in_0 = 1'b1;
        in_1 = 1'b1;
        carry_in = 1'b1;
        #10

        $finish;
    end
endmodule
```

Kết quả mô phỏng



Bảng sự thật:

S1	S0	Biểu thức	In_0	In_1	Carry_in	G	Carry_out
0	0	A + Ci	0		0	0	0
			0		1	1	0
			1		0	1	0
			1		1	0	1
0	1	A + B + Ci	0	0	0	0	0
			0	0	1	1	0
			0	1	0	1	0
			0	1	1	0	1
			1	0	0	1	0
			1	0	1	0	1
			1	1	0	0	1
			1	1	1	1	1
1	0	A + B’ + Ci	0	0	0	1	0
			0	0	1	0	1
			0	1	0	0	0
			0	1	1	1	0
			1	0	0	0	1
			1	0	1	1	1
			1	1	0	1	0
			1	1	1	0	1
1	1	A’ + B + Ci	0	0	0	1	0
			0	0	1	0	1
			0	1	0	0	1
			0	1	1	1	1
			1	0	0	0	0

			1	0	1	1	0
			1	1	0	1	0
			1	1	1	0	1

Nhận xét: khối AU đúng với tất cả các trường hợp

2.2.2 Thiết kế bộ Logic Unit

Dựa vào bảng chức năng ALU hình 1.1, ta có thể thiết kế khối LU.

Code LU:

```
module logic_unit(S0, S1,in_0,in_1,H);

    input S0, S1, in_0, in_1;
    output wire H;

    reg g;
    wire [1:0] s;
    wire a, b, c, d;

    assign a = in_0 & in_1;
    assign b = in_0 | in_1;
    assign c = in_0 ^ in_1;
    assign d = ~(in_0 ^ in_1);
    assign H = g;
    assign s[0] = S0;
    assign s[1] = S1;

    always @ (a or b or c or d or s)

    case (s)

        0: g = a;
        1: g = b;
        2: g = c;
        3: g = d;
        default: $display ("Error in control");

    endcase

endmodule
```

Code test bench

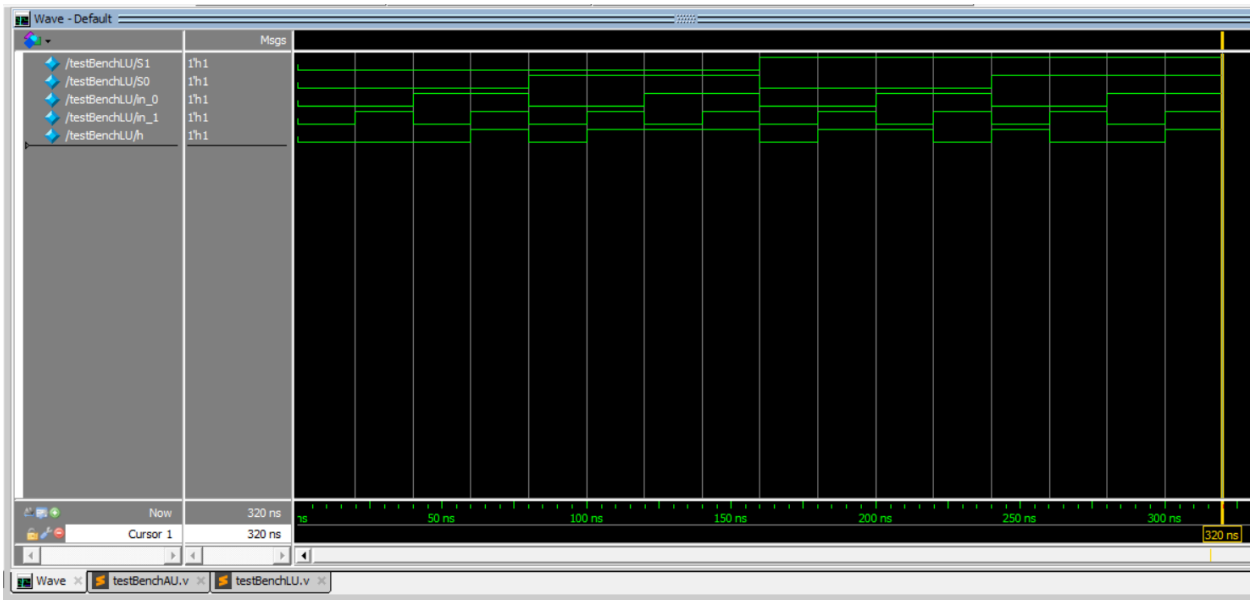
```
module testBenchLU;
```

```
reg S1;
reg S0;
reg in_0;
reg in_1;
wire H;
logic_unit logic_unit_1(    .S1      (S1),
                            .S0      (S0),
                            .in_0    (in_0),
                            .in_1    (in_1),
                            .H       (H)
                            );
initial begin
    // S1 = 0, S0 = 0, A and B
    S1 = 1'b0;
    S0 = 1'b0;
    in_0 = 1'b0;
    in_1 = 1'b0;
    #20
    S1 = 1'b0;
    S0 = 1'b0;
    in_0 = 1'b0;
    in_1 = 1'b1;
    #20
    S1 = 1'b0;
    S0 = 1'b0;
    in_0 = 1'b1;
    in_1 = 1'b0;
    #20
    S1 = 1'b0;
    S0 = 1'b0;
    in_0 = 1'b1;
    in_1 = 1'b1;
    #20

    S1 = 1'b0;
    S0 = 1'b1;
    in_0 = 1'b0;
    in_1 = 1'b0;
    #20
    S1 = 1'b0;
    S0 = 1'b1;
    in_0 = 1'b0;
    in_1 = 1'b1;
    #20
    S1 = 1'b0;
```

```
    S0 = 1'b1;
    in_0 = 1'b1;
    in_1 = 1'b0;
    #20
    S1 = 1'b0;
    S0 = 1'b1;
    in_0 = 1'b1;
    in_1 = 1'b1;
    #20

    S1 = 1'b1;
    S0 = 1'b0;
    in_0 = 1'b0;
    in_1 = 1'b0;
    #20
    S1 = 1'b1;
    S0 = 1'b0;
    in_0 = 1'b0;
    in_1 = 1'b1;
    #20
    S1 = 1'b1;
    S0 = 1'b0;
    in_0 = 1'b1;
    in_1 = 1'b0;
    #20
    S1 = 1'b1;
    S0 = 1'b0;
    in_0 = 1'b1;
    in_1 = 1'b1;
    #20

    S1 = 1'b1;
    S0 = 1'b1;
    in_0 = 1'b0;
    in_1 = 1'b0;
    #20
    S1 = 1'b1;
    S0 = 1'b1;
    in_0 = 1'b0;
    in_1 = 1'b1;
    #20
    S1 = 1'b1;
    S0 = 1'b1;
    in_0 = 1'b1;
    in_1 = 1'b0;
```

```
    #20
    S1 = 1'b1;
    S0 = 1'b1;
    in_0 = 1'b1;
    in_1 = 1'b1;
    #20
$finish;
end
endmodule
```

Kết quả mô phỏng:



Bảng sự thật:

S1	S0	Biểu thức	In_0	In_1	H
0	0	A and B	0	0	0
			0	1	0
			1	0	0
			1	1	1
0	1	A or B	0	0	0
			0	1	1
			1	0	1
			1	1	1
1	0	A xor B	0	0	0
			0	1	1
			1	0	1
			1	1	0
1	1	A xnor B	0	0	1
			0	1	0

			1	0	0
			1	1	1

Nhận xét: kết quả mô phỏng đúng với kết quả tính toán theo lý thuyết

2.2.3 Thiết kế bộ Mux 2:1

Sau khi hoàn thành 2 bộ Arithmetic Unit và Logic Unit, để có thể ghép hai bộ đó trở thành module ALU 1 bit, ta cần 1 bộ Mux 2:1 để thực hiện điều đó.

Code Mux 2:1

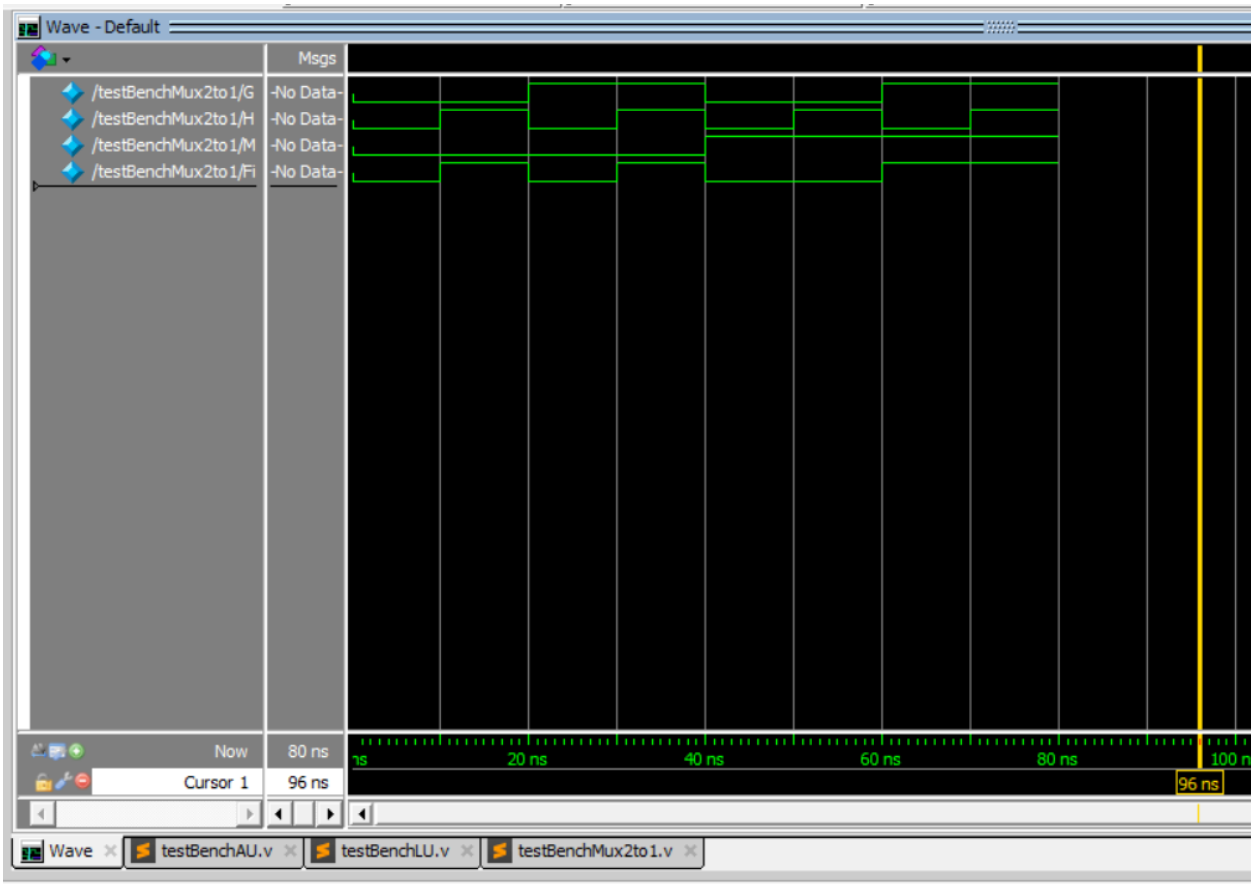
```
module mux2to1(G, H, M, Fi);

    input G; //Arithmetic Unit
    input H; //Logic Unit
    input M;
    output Fi;

    reg Fi;

    always @(G, H, M)
    begin
        if(M == 0)
            Fi = H;
        else
            Fi = G;
    end

endmodule
```

Code test bench:

```
module testBenchMux2to1;

    reg G, H, M;
    wire Fi;

    mux2to1 mux2to1 (   .G  (G),
                        .H  (H),
                        .M  (M),
                        .Fi (Fi)
                        );

    initial begin
```

```
        G = 1'b0;
        H = 1'b0;
        M = 1'b0;
        #10

        G = 1'b0;
        H = 1'b1;
        M = 1'b0;
        #10

        G = 1'b1;
        H = 1'b0;
        M = 1'b0;
        #10

        G = 1'b1;
        H = 1'b1;
        M = 1'b0;
        #10

        G = 1'b0;
        H = 1'b0;
        M = 1'b1;
        #10

        G = 1'b0;
        H = 1'b1;
        M = 1'b1;
        #10

        G = 1'b1;
        H = 1'b0;
        M = 1'b1;
        #10

        G = 1'b1;
        H = 1'b1;
        M = 1'b1;
        #10
        $finish;

    end

endmodule
```

Kết quả mô phỏng:



Bảng sự thật

G	H	M	Fi
0	0	0	0
0	1	0	1
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	0
1	0	1	1
1	1	1	1

Nhận xét: Kết quả mô phỏng giống như kết quả tính toán.

2.2.4 Thiết kế bộ ALU 1 bit

Bằng cách ghép 3 bộ AU, LU, Mux 2:1 ta đã thiết kế ở trên, chúng ta sẽ có 1 bộ ALU 1 bit.

Code ALU 1 bit:

```
module alu1bit (S1, S0, M, in_0, in_1, carry_in, carry_out, Fi);

    input S1, S0, M, in_0, in_1, carry_in;
    output carry_out, Fi;

    wire G;
    wire H;

    arithmetic_unit arithmetic_unit (   .S0         (S0),
                                        .S1         (S1),
                                        .in_0       (in_0),
                                        .in_1       (in_1),
                                        .carry_in   (carry_in),
                                        .G          (G),
                                        .carry_out  (carry_out)
                                        );

    logic_unit logic_unit_1(    .S1     (S1),
                                .S0     (S0),
                                .in_0   (in_0),
                                .in_1   (in_1),
                                .H      (H)
                            );

    mux2to1 mux2to1 (   .G  (G),
                        .H  (H),
                        .M  (M),
                        .Fi (Fi)
                        );

endmodule
```

Code test bench

```
module testBenchALU;

    reg S1, S0, M, in_0, in_1, carry_in;
    wire carry_out, Fi;

    alu1bit alu1bit (   .S1         (S1),
                        .S0         (S0),
                        .M          (M),
                        .in_0       (in_0),
                        .in_1       (in_1),
                        .carry_in   (carry_in),
                        .carry_out  (carry_out),
```

```
                    .Fi         (Fi)
                    );
initial begin
//M = 1
//S1 = 0, S0 = 0, A + Ci
    M = 1;
    S1 = 1'b0;
    S0 = 1'b0;
    in_1 = 1'b0;
    in_0 = 1'b0;
    carry_in = 1'b0;
    #10
    M = 1;
    S1 = 1'b0;
    S0 = 1'b0;
    in_1 = 1'b0;
    in_0 = 1'b0;
    carry_in = 1'b1;
    #10
    M = 1;
    S1 = 1'b0;
    S0 = 1'b0;
    in_1 = 1'b0;
    in_0 = 1'b1;
    carry_in = 1'b0;
    #10
    M = 1;
    S1 = 1'b0;
    S0 = 1'b0;
    in_1 = 1'b1;
    in_0 = 1'b1;
    carry_in = 1'b1;
// S1 = 0, S0 = 1, A + B + Cin
    #10
    M = 1;
    S1 = 1'b0;
    S0 = 1'b1;
    in_0 = 1'b0;
    in_1 = 1'b0;
    carry_in = 1'b0;
    #10
    M = 1;
    S1 = 1'b0;
    S0 = 1'b1;
    in_0 = 1'b0;
```

```
        in_1 = 1'b0;
        carry_in = 1'b1;
        #10
        M = 1;
        S1 = 1'b0;
        S0 = 1'b1;
        in_0 = 1'b0;
        in_1 = 1'b1;
        carry_in = 1'b0;
        #10
        M = 1;
        S1 = 1'b0;
        S0 = 1'b1;
        in_0 = 1'b0;
        in_1 = 1'b1;
        carry_in = 1'b1;
        #10
        M = 1;
        S1 = 1'b0;
        S0 = 1'b1;
        in_0 = 1'b1;
        in_1 = 1'b0;
        carry_in = 1'b0;
        #10
        M = 1;
        S1 = 1'b0;
        S0 = 1'b1;
        in_0 = 1'b1;
        in_1 = 1'b0;
        carry_in = 1'b1;
        #10
        M = 1;
        S1 = 1'b0;
        S0 = 1'b1;
        in_0 = 1'b1;
        in_1 = 1'b1;
        carry_in = 1'b0;
        #10
        M = 1;
        S1 = 1'b0;
        S0 = 1'b1;
        in_0 = 1'b1;
        in_1 = 1'b1;
        carry_in = 1'b1;
    //S1 = 1, S0 = 0, A + B' + Cin
```

```
        #10
        M = 1;
        S1 = 1'b1;
        S0 = 1'b0;
        in_0 = 1'b0;
        in_1 = 1'b0;
        carry_in = 1'b0;
        #10
        M = 1;
        S1 = 1'b1;
        S0 = 1'b0;
        in_0 = 1'b0;
        in_1 = 1'b0;
        carry_in = 1'b1;
        #10
        M = 1;
        S1 = 1'b1;
        S0 = 1'b0;
        in_0 = 1'b0;
        in_1 = 1'b1;
        carry_in = 1'b0;
        #10
        M = 1;
        S1 = 1'b1;
        S0 = 1'b0;
        in_0 = 1'b0;
        in_1 = 1'b1;
        carry_in = 1'b1;
        #10
        M = 1;
        S1 = 1'b1;
        S0 = 1'b0;
        in_0 = 1'b1;
        in_1 = 1'b0;
        carry_in = 1'b0;
        #10
        M = 1;
        S1 = 1'b1;
        S0 = 1'b0;
        in_0 = 1'b1;
        in_1 = 1'b0;
        carry_in = 1'b1;
        #10
        M = 1;
        S1 = 1'b1;
```

```
        S0 = 1'b0;
        in_0 = 1'b1;
        in_1 = 1'b1;
        carry_in = 1'b0;
        #10
        M = 1;
        S1 = 1'b1;
        S0 = 1'b0;
        in_0 = 1'b1;
        in_1 = 1'b1;
        carry_in = 1'b1;
    //S1 = 1, S0 = 1, A' + B + Cin
        #10
        M = 1;
        S1 = 1'b1;
        S0 = 1'b1;
        in_0 = 1'b0;
        in_1 = 1'b0;
        carry_in = 1'b0;
        #10
        M = 1;
        S1 = 1'b1;
        S0 = 1'b1;
        in_0 = 1'b0;
        in_1 = 1'b0;
        carry_in = 1'b1;
        #10
        M = 1;
        S1 = 1'b1;
        S0 = 1'b1;
        in_0 = 1'b0;
        in_1 = 1'b1;
        carry_in = 1'b0;
        #10
        M = 1;
        S1 = 1'b1;
        S0 = 1'b1;
        in_0 = 1'b0;
        in_1 = 1'b1;
        carry_in = 1'b1;
        #10
        M = 1;
        S1 = 1'b1;
        S0 = 1'b1;
        in_0 = 1'b1;
```

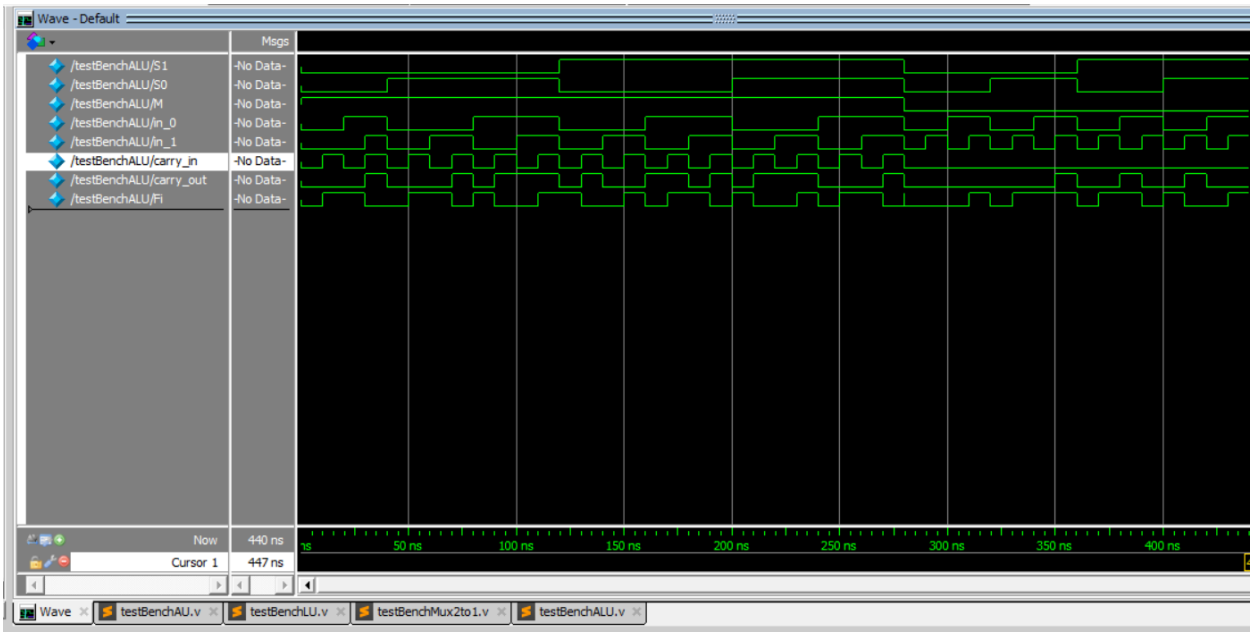
```
        in_1 = 1'b0;
        carry_in = 1'b0;
        #10
        M = 1;
        S1 = 1'b1;
        S0 = 1'b1;
        in_0 = 1'b1;
        in_1 = 1'b0;
        carry_in = 1'b1;
        #10
        M = 1;
        S1 = 1'b1;
        S0 = 1'b1;
        in_0 = 1'b1;
        in_1 = 1'b1;
        carry_in = 1'b0;
        #10
        M = 1;
        S1 = 1'b1;
        S0 = 1'b1;
        in_0 = 1'b1;
        in_1 = 1'b1;
        carry_in = 1'b1;
        #10
    //M = 0
    //S1 = 0, S0 = 0, A and B
        M = 0;
        S1 = 1'b0;
        S0 = 1'b0;
        in_0 = 1'b0;
        in_1 = 1'b0;
        carry_in = 1'b0;
        #10
        M = 0;
        S1 = 1'b0;
        S0 = 1'b0;
        in_0 = 1'b0;
        in_1 = 1'b1;
        carry_in = 1'b0;
        #10
        M = 0;
        S1 = 1'b0;
        S0 = 1'b0;
        in_0 = 1'b1;
        in_1 = 1'b0;
```

```
        carry_in = 1'b0;
        #10
        M = 0;
        S1 = 1'b0;
        S0 = 1'b0;
        in_0 = 1'b1;
        in_1 = 1'b1;
        carry_in = 1'b0;
        #10
    //S1 = 0, S0 = 1, A or B
        M = 0;
        S1 = 1'b0;
        S0 = 1'b1;
        in_0 = 1'b0;
        in_1 = 1'b0;
        carry_in = 1'b0;
        #10
        M = 0;
        S1 = 1'b0;
        S0 = 1'b1;
        in_0 = 1'b0;
        in_1 = 1'b1;
        carry_in = 1'b0;
        #10
        M = 0;
        S1 = 1'b0;
        S0 = 1'b1;
        in_0 = 1'b1;
        in_1 = 1'b0;
        carry_in = 1'b0;
        #10
        M = 0;
        S1 = 1'b0;
        S0 = 1'b1;
        in_0 = 1'b1;
        in_1 = 1'b1;
        carry_in = 1'b0;
        #10
    //S1 = 1, S0 = 0, A xor B
        M = 0;
        S1 = 1'b1;
        S0 = 1'b0;
        in_0 = 1'b0;
        in_1 = 1'b0;
        carry_in = 1'b0;
```

```
        #10
        M = 0;
        S1 = 1'b1;
        S0 = 1'b0;
        in_0 = 1'b0;
        in_1 = 1'b1;
        carry_in = 1'b0;
        #10
        M = 0;
        S1 = 1'b1;
        S0 = 1'b0;
        in_0 = 1'b1;
        in_1 = 1'b0;
        carry_in = 1'b0;
        #10
        M = 0;
        S1 = 1'b1;
        S0 = 1'b0;
        in_0 = 1'b1;
        in_1 = 1'b1;
        carry_in = 1'b0;
        #10
    //S1 = 1, S0 = 0, A xnor B
        M = 0;
        S1 = 1'b1;
        S0 = 1'b1;
        in_0 = 1'b0;
        in_1 = 1'b0;
        carry_in = 1'b0;
        #10
        M = 0;
        S1 = 1'b1;
        S0 = 1'b1;
        in_0 = 1'b0;
        in_1 = 1'b1;
        carry_in = 1'b0;
        #10
        M = 0;
        S1 = 1'b1;
        S0 = 1'b1;
        in_0 = 1'b1;
        in_1 = 1'b0;
        carry_in = 1'b0;
        #10
        M = 0;
```

```
        S1 = 1'b1;
        S0 = 1'b1;
        in_0 = 1'b1;
        in_1 = 1'b1;
        carry_in = 1'b0;
        #10
        $finish;
    end
endmodule
```

Kết quả mô phỏng:



Bảng sự thật:

M	S1	S0	Biểu thức	In_0	In_1	Carry_in	Fi	Carry_out
1	0	0	A + Ci	0		0	0	0
1	0	0		0		1	1	0
1	0	0		1		0	1	0
1	0	0		1		1	0	1
1	0	1	A + B + Ci	0	0	0	0	0
1	0	1		0	0	1	1	0
1	0	1		0	1	0	1	0
1	0	1		0	1	1	0	1
1	0	1		1	0	0	1	0
1	0	1		1	0	1	0	1
1	0	1		1	1	0	0	1
1	0	1		1	1	1	1	1

1	1	0	A + B’ + Ci	0	0	0	1	0
1	1	0		0	0	1	0	1
1	1	0		0	1	0	0	0
1	1	0		0	1	1	1	0
1	1	0		1	0	0	0	1
1	1	0		1	0	1	1	1
1	1	0		1	1	0	1	0
1	1	0		1	1	1	0	1
1	1	1	A’ + B + Ci	0	0	0	1	0
1	1	1		0	0	1	0	1
1	1	1		0	1	0	0	1
1	1	1		0	1	1	1	1
1	1	1		1	0	0	0	0
1	1	1		1	0	1	1	0
1	1	1		1	1	0	1	0
1	1	1		1	1	1	0	1
0	0	0	A and B	0	0	0		0
0	0	0		0	1	0		0
0	0	0		1	0	0		0
0	0	0		1	1	0		1
0	0	1	A or B	0	0	0		0
0	0	1		0	1	0		1
0	0	1		1	0	0		1
0	0	1		1	1	0		1
0	1	0	A xor B	0	0	0		0
0	1	0		0	1	0		1
0	1	0		1	0	0		1
0	1	0		1	1	0		0
0	1	1	A xnor B	0	0	0		1
0	1	1		0	1	0		0
0	1	1		1	0	0		0
0	1	1		1	1	0		1

2.2.5 Nhận xét kết quả

Dựa vào đồ thị sóng biểu thị kết quả mô phỏng của Model Sim và bảng chân trị được tính theo lý thuyết, ta có thể kết luận bộ ALU 1 bit được thiết kế chính xác.

2.3 Thực hiện thiết kế ALU 4 bit

Như phần lựa chọn phương án thực hiện, ta có thể thực hiện việc ghép 4 bộ ALU 1 bit lại với nhau để tạo ra bộ ALU 4 bit.

Code ALU 4 bit:

```
module alu4bit (S1, S0, M, in_0, in_1, carry_in, carry_out, Fi);

    input S1, S0, M; //Control bit
    input [3:0] in_0, in_1; //4 bit data
    input carry_in;
    output [3:0] Fi;
    output carry_out;

    wire temp_in_0_0, temp_in_0_1, temp_in_0_2, temp_in_0_3;
    wire temp_in_1_0, temp_in_1_1, temp_in_1_2, temp_in_1_3;
    wire carry_temp_0, carry_temp_1, carry_temp_2;
    wire Fi_0, Fi_1, Fi_2, Fi_3;

    assign temp_in_0_0 = in_0[0];
    assign temp_in_0_1 = in_0[1];
    assign temp_in_0_2 = in_0[2];
    assign temp_in_0_3 = in_0[3];

    assign temp_in_1_0 = in_1[0];
    assign temp_in_1_1 = in_1[1];
    assign temp_in_1_2 = in_1[2];
    assign temp_in_1_3 = in_1[3];

    assign Fi[0] =  Fi_0;
    assign Fi[1] =  Fi_1;
    assign Fi[2] =  Fi_2;
    assign Fi[3] =  Fi_3;

    alu1bit alu1bit_0 ( .S1         (S1),
                        .S0         (S0),
                        .M          (M),
                        .in_0       (temp_in_0_0),
                        .in_1       (temp_in_1_0),
                        .carry_in   (carry_in),
                        .carry_out  (carry_temp_0),
                        .Fi         (Fi_0)
                        );

    alu1bit alu1bit_1 ( .S1         (S1),
                        .S0         (S0),
                        .M          (M),
                        .in_0       (temp_in_0_1),
                        .in_1       (temp_in_1_1),
```

```
                        .carry_in   (carry_temp_0),
                        .carry_out  (carry_temp_1),
                        .Fi         (Fi_1)
                        );

    alu1bit alu1bit_2 ( .S1         (S1),
                        .S0         (S0),
                        .M          (M),
                        .in_0       (temp_in_0_2),
                        .in_1       (temp_in_1_2),
                        .carry_in   (carry_temp_1),
                        .carry_out  (carry_temp_2),
                        .Fi         (Fi_2)
                        );

    alu1bit alu1bit_3 ( .S1         (S1),
                        .S0         (S0),
                        .M          (M),
                        .in_0       (temp_in_0_3),
                        .in_1       (temp_in_1_3),
                        .carry_in   (carry_temp_2),
                        .carry_out  (carry_out),
                        .Fi         (Fi_3)
                        );

endmodule
```

Code test bench:

```
module testBenchALU4bit;

    reg S1, S0, M;
    reg [3:0] in_0, in_1;
    reg carry_in;
    wire [3:0] Fi;
    wire carry_out;

    alu4bit alu4bit (   .S1         (S1),
                        .S0         (S0),
                        .M          (M),
                        .in_0       (in_0),
                        .in_1       (in_1),
                        .carry_in   (carry_in),
                        .carry_out  (carry_out),
                        .Fi         (Fi)
                        );
```

```
initial begin

//AU, M = 1
    //S1 = 0, S0 = 0, A + Ci
    M = 1;
    S0 = 0;
    S1 = 0;
    in_0 = 4'b0000;
    in_1 = 4'b0000;
    carry_in = 1'b0;
    #10
    M = 1;
    S0 = 0;
    S1 = 0;
    in_0 = 4'b0000;
    in_1 = 4'b0000;
    carry_in = 1'b1;
    #10
    M = 1;
    S0 = 0;
    S1 = 0;
    in_0 = 4'b0001;
    in_1 = 4'b0000;
    carry_in = 1'b0;
    #10
    M = 1;
    S0 = 0;
    S1 = 0;
    in_0 = 4'b0001;
    in_1 = 4'b0000;
    carry_in = 1'b1;
    #10
    M = 1;
    S0 = 0;
    S1 = 0;
    in_0 = 4'b1111;
    in_1 = 4'b0000;
    carry_in = 1'b0;
    #10
    //S1 = 0, S0 = 1, A + B + Ci
    M = 1;
    S0 = 1;
    S1 = 0;
    in_0 = 4'b0000;
```

```
        in_1 = 4'b0000;
        carry_in = 1'b0;
        #10
        M = 1;
        S0 = 1;
        S1 = 0;
        in_0 = 4'b0001;
        in_1 = 4'b0000;
        carry_in = 1'b1;
        #10
        M = 1;
        S0 = 1;
        S1 = 0;
        in_0 = 4'b0000;
        in_1 = 4'b0001;
        carry_in = 1'b0;
        #10
        M = 1;
        S0 = 1;
        S1 = 0;
        in_0 = 4'b0000;
        in_1 = 4'b0001;
        carry_in = 1'b1;
        #10
        M = 1;
        S0 = 1;
        S1 = 0;
        in_0 = 4'b1110;
        in_1 = 4'b0001;
        carry_in = 1'b0;
        #10
        M = 1;
        S0 = 1;
        S1 = 0;
        in_0 = 4'b1110;
        in_1 = 4'b0001;
        carry_in = 1'b1;
        #10
        M = 1;
        S0 = 1;
        S1 = 0;
        in_0 = 4'b1111;
        in_1 = 4'b0001;
        carry_in = 1'b0;
        #10
```

```
        M = 1;
        S0 = 1;
        S1 = 0;
        in_0 = 4'b1111;
        in_1 = 4'b0001;
        carry_in = 1'b1;
        #10
        M = 1;
        S0 = 1;
        S1 = 0;
        in_0 = 4'b1111;
        in_1 = 4'b1111;
        carry_in = 1'b0;
        #10
        M = 1;
        S0 = 1;
        S1 = 0;
        in_0 = 4'b1111;
        in_1 = 4'b1111;
        carry_in = 1'b1;
        #10
        //S1 = 1, S0 = 0, A + B' + Ci
        M = 1;
        S0 = 0;
        S1 = 1;
        in_0 = 4'b0000;
        in_1 = 4'b0000;
        carry_in = 1'b0;
        #10
        M = 1;
        S0 = 0;
        S1 = 1;
        in_0 = 4'b0001;
        in_1 = 4'b0000;
        carry_in = 1'b1;
        #10
        M = 1;
        S0 = 0;
        S1 = 1;
        in_0 = 4'b0000;
        in_1 = 4'b0001;
        carry_in = 1'b0;
        #10
        M = 1;
        S0 = 0;
```

```
        S1 = 1;
        in_0 = 4'b0000;
        in_1 = 4'b0001;
        carry_in = 1'b1;
        #10
        M = 1;
        S0 = 0;
        S1 = 1;
        in_0 = 4'b1110;
        in_1 = 4'b0001;
        carry_in = 1'b0;
        #10
        M = 1;
        S0 = 0;
        S1 = 1;
        in_0 = 4'b1110;
        in_1 = 4'b0001;
        carry_in = 1'b1;
        #10
        M = 1;
        S0 = 0;
        S1 = 1;
        in_0 = 4'b1111;
        in_1 = 4'b0001;
        carry_in = 1'b0;
        #10
        M = 1;
        S0 = 0;
        S1 = 1;
        in_0 = 4'b1111;
        in_1 = 4'b0001;
        carry_in = 1'b1;
        #10
        M = 1;
        S0 = 0;
        S1 = 1;
        in_0 = 4'b1111;
        in_1 = 4'b1111;
        carry_in = 1'b0;
        #10
        M = 1;
        S0 = 0;
        S1 = 1;
        in_0 = 4'b1111;
        in_1 = 4'b1111;
```

```
        carry_in = 1'b1;
        #10
        //S1 = 1, S0 = 1, A' + B + Ci
        M = 1;
        S0 = 1;
        S1 = 1;
        in_0 = 4'b0000;
        in_1 = 4'b0000;
        carry_in = 1'b0;
        #10
        M = 1;
        S0 = 1;
        S1 = 1;
        in_0 = 4'b0001;
        in_1 = 4'b0000;
        carry_in = 1'b1;
        #10
        M = 1;
        S0 = 1;
        S1 = 1;
        in_0 = 4'b0000;
        in_1 = 4'b0001;
        carry_in = 1'b0;
        #10
        M = 1;
        S0 = 1;
        S1 = 1;
        in_0 = 4'b0000;
        in_1 = 4'b0001;
        carry_in = 1'b1;
        #10
        M = 1;
        S0 = 1;
        S1 = 1;
        in_0 = 4'b1110;
        in_1 = 4'b0001;
        carry_in = 1'b0;
        #10
        M = 1;
        S0 = 1;
        S1 = 1;
        in_0 = 4'b1110;
        in_1 = 4'b0001;
        carry_in = 1'b1;
        #10
```

```
        M = 1;
        S0 = 1;
        S1 = 1;
        in_0 = 4'b1111;
        in_1 = 4'b0001;
        carry_in = 1'b0;
        #10
        M = 1;
        S0 = 1;
        S1 = 1;
        in_0 = 4'b1111;
        in_1 = 4'b0001;
        carry_in = 1'b1;
        #10
        M = 1;
        S0 = 1;
        S1 = 1;
        in_0 = 4'b1111;
        in_1 = 4'b1111;
        carry_in = 1'b0;
        #10
        M = 1;
        S0 = 1;
        S1 = 1;
        in_0 = 4'b1111;
        in_1 = 4'b1111;
        carry_in = 1'b1;
        #10

    //LU, M = 0
        //S1 = 0, S0 = 0, A and B
        M = 0;
        S0 = 0;
        S1 = 0;
        in_0 = 4'b0000;
        in_1 = 4'b0000;
        carry_in = 1'b0;
        #10
        M = 0;
        S0 = 0;
        S1 = 0;
        in_0 = 4'b0001;
        in_1 = 4'b0000;
        carry_in = 1'b1;
        #10
```

```
        M = 0;
        S0 = 0;
        S1 = 0;
        in_0 = 4'b0000;
        in_1 = 4'b0001;
        carry_in = 1'b0;
        #10
        M = 0;
        S0 = 0;
        S1 = 0;
        in_0 = 4'b1110;
        in_1 = 4'b0001;
        carry_in = 1'b0;
        #10
        M = 0;
        S0 = 0;
        S1 = 0;
        in_0 = 4'b1110;
        in_1 = 4'b0011;
        carry_in = 1'b1;
        #10
        M = 0;
        S0 = 0;
        S1 = 0;
        in_0 = 4'b1111;
        in_1 = 4'b0000;
        carry_in = 1'b0;
        #10
        M = 0;
        S0 = 0;
        S1 = 0;
        in_0 = 4'b1111;
        in_1 = 4'b1001;
        carry_in = 1'b1;
        #10
        M = 0;
        S0 = 0;
        S1 = 0;
        in_0 = 4'b1111;
        in_1 = 4'b1111;
        carry_in = 1'b0;
        #10
        //S1 = 0, S0 = 1, A or B
        M = 0;
        S0 = 1;
```

```
        S1 = 0;
        in_0 = 4'b0000;
        in_1 = 4'b0000;
        carry_in = 1'b0;
        #10
        M = 0;
        S0 = 1;
        S1 = 0;
        in_0 = 4'b0001;
        in_1 = 4'b0000;
        carry_in = 1'b1;
        #10
        M = 0;
        S0 = 1;
        S1 = 0;
        in_0 = 4'b0000;
        in_1 = 4'b0001;
        carry_in = 1'b0;
        #10
        M = 0;
        S0 = 1;
        S1 = 0;
        in_0 = 4'b1110;
        in_1 = 4'b0001;
        carry_in = 1'b0;
        #10
        M = 0;
        S0 = 1;
        S1 = 0;
        in_0 = 4'b1110;
        in_1 = 4'b0011;
        carry_in = 1'b1;
        #10
        M = 0;
        S0 = 1;
        S1 = 0;
        in_0 = 4'b1111;
        in_1 = 4'b0000;
        carry_in = 1'b0;
        #10
        M = 0;
        S0 = 1;
        S1 = 0;
        in_0 = 4'b1111;
        in_1 = 4'b1001;
```

```
        carry_in = 1'b1;
        #10
        M = 0;
        S0 = 1;
        S1 = 0;
        in_0 = 4'b1111;
        in_1 = 4'b1111;
        carry_in = 1'b0;
        #10
        //S1 = 1, S0 = 0, A xor B
        M = 0;
        S0 = 0;
        S1 = 1;
        in_0 = 4'b0000;
        in_1 = 4'b0000;
        carry_in = 1'b0;
        #10
        M = 0;
        S0 = 0;
        S1 = 1;
        in_0 = 4'b0001;
        in_1 = 4'b0000;
        carry_in = 1'b1;
        #10
        M = 0;
        S0 = 0;
        S1 = 1;
        in_0 = 4'b0000;
        in_1 = 4'b0001;
        carry_in = 1'b0;
        #10
        M = 0;
        S0 = 0;
        S1 = 1;
        in_0 = 4'b1110;
        in_1 = 4'b0001;
        carry_in = 1'b0;
        #10
        M = 0;
        S0 = 0;
        S1 = 1;
        in_0 = 4'b1110;
        in_1 = 4'b0011;
        carry_in = 1'b1;
        #10
```

```
        M = 0;
        S0 = 0;
        S1 = 1;
        in_0 = 4'b1111;
        in_1 = 4'b0000;
        carry_in = 1'b0;
        #10
        M = 0;
        S0 = 0;
        S1 = 1;
        in_0 = 4'b1111;
        in_1 = 4'b1001;
        carry_in = 1'b1;
        #10
        M = 0;
        S0 = 0;
        S1 = 1;
        in_0 = 4'b1111;
        in_1 = 4'b1111;
        carry_in = 1'b0;
        #10
        //S1 = 1, S0 = 1, A xnor B
        M = 0;
        S0 = 1;
        S1 = 1;
        in_0 = 4'b0000;
        in_1 = 4'b0000;
        carry_in = 1'b0;
        #10
        M = 0;
        S0 = 1;
        S1 = 1;
        in_0 = 4'b0001;
        in_1 = 4'b0000;
        carry_in = 1'b1;
        #10
        M = 0;
        S0 = 1;
        S1 = 1;
        in_0 = 4'b0000;
        in_1 = 4'b0001;
        carry_in = 1'b0;
        #10
        M = 0;
        S0 = 1;
```

```
        S1 = 1;
        in_0 = 4'b1110;
        in_1 = 4'b0001;
        carry_in = 1'b0;
        #10
        M = 0;
        S0 = 1;
        S1 = 1;
        in_0 = 4'b1110;
        in_1 = 4'b0011;
        carry_in = 1'b1;
        #10
        M = 0;
        S0 = 1;
        S1 = 1;
        in_0 = 4'b1111;
        in_1 = 4'b0000;
        carry_in = 1'b0;
        #10
        M = 0;
        S0 = 1;
        S1 = 1;
        in_0 = 4'b1111;
        in_1 = 4'b1001;
        carry_in = 1'b1;
        #10
        M = 0;
        S0 = 1;
        S1 = 1;
        in_0 = 4'b1111;
        in_1 = 4'b1111;
        carry_in = 1'b0;
        #10
        $finish;

    end


endmodule
```

Nhận xét kết quả: Kết quả mô phỏng giống với kết quả lý thuyết.