با اسمه تعالی

نحوه اجرای کد ها به صورت قسمت های جدا از هم باهدف ساده سازی برنامه برای شما انجام شده است .

در بخش اول ماژول اول که ماژول`(Logic_Shift)` قرار دارد 8 ورودی و یک خروجی تعریف کردیم

```
    input s0,
    input s1,
    input s2,
    input s3,
    input Ai,
    input Bi,
    input Ai_1,
    input Aii,
    output reg F
```

و دو وایر به نام های (**E1 , D1**)

در ادامه ماژول (**Mux_gate_1to1 U1**) صدا زده میشود.

```
Mux_gate_1to1 U1(
        .S0(s0),
        .S1(s1),
        .A(Ai),
        .B(Bi),

        .Ei(E1)
    );
```

در این ماژول ورودی های (S0 , S1 , Ai , Bi) تابع `(Logic_Shift)` را برای ماژول مولتی پلکسر ارسال میشود.

ماژول (`Mux_gate_1to1`):

```
module Mux_gate_1to1(

    input S0,
    input S1,
    input A,
    input B,
    output reg Ei
);

always @(*)
    begin
        case({S1,S0})
            2'b00 :  Ei = A & B;
            2'b01 :  Ei = A | B;
            2'b10 :  Ei = A ^ B;
            2'b11 :  Ei = ~(A) ;
            default:  Ei = 1'bx;
        endcase
    end


endmodule
```

- **@(*)** نشان‌دهنده‌ی حساسیت به تغییرات همه‌ی ورودی‌ها است، به این معنی که هر زمان که یکی از ورودی‌ها تغییر کند، بلوک **always** اجرا می‌شود.

ورودی‌های S0 و S1 سیگنال‌های کنترلی هستند که تعیین می‌کنند کدام یک از داده‌های (And , Or , Xor , Not) اجرا شده و در آخر اگر مقداری نداشته باشد به صورت پیش فرض مقدار Ei برار با (X) یک بیتی از نوع باینری قرار میگیرد(برای اینکه دیگر خروجی **always** در نظر گرفته نشود).

این بلوک به منظور تعیین خروجی F بر اساس مقادیر ورودی‌ها طراحی شده است.

- این ماژول به منظور انتخاب یکی از داده‌های ورودی بر اساس سیگنال‌های کنترلی طراحی شده است. انتخاب‌ها و عملیات‌های مختلف مولتی‌پلکسر با توجه به مقادیر ورودی‌ها و سیگنال‌های کنترلی انجام می‌شود.

در ادامه به ماژول مادر برمیگردیم و ماژول (**Mux_Gate_4to1_FA U2**) را صدا میزیم.

```
Mux_Gate_4to1_FA U2(
        .S0(s0),
        .S1(s1),
        .CIN(s2),
        .A0(Ai),
        .B0(Bi),
        .O(E1),

        .D(D1)
    );
```

ماژول (Mux_Gate_4to1) یک مولتی‌پلکسر چهار به یک به همراه یک ماژول (Full_Adder U1) است.

```
module Mux_Gate_4to1_FA(
    input S0,
    input S1,
    input CIN,
    input A0,
    input B0,
    input O,

    output D
);

    wire reg Y0;

    assign Y0 = (S0) ? B0 : ~B0;
    assign D = (S1) ? Y0 : ~Y0;

      /*
    می توانیم ماژول فول ادر را همراه با مولتی پلکسر در این قسمت بنویسیم در عملکرد تفاوتی ندارد.
    */

    Full_Adder U1(
        .X1(A0),
        .X2(Y0),
        .cin(CIN),

        .S(D)
    );


endmodule
```

- ورودی‌های S0 و S1 سیگنال‌های کنترلی هستند که تعیین می‌کنند کدام یک از داده‌های A0 و B0 به عنوان خروجی که D است انتخاب شود.

- CIN ورودی حمل‌برای Full Adder است.

- خروجی D نتیجه‌ی انتخاب مولتی‌پلکسر و افزودن داده‌ها با استفاده از **Full Adder** است.

- مقادیر Y0 و D با توجه به مقادیر S0 و S1 تعیین می‌شوند.

- این ماژول به منظور انتخاب یکی از داده‌های ورودی بر اساس سیگنال‌های کنترلی و انجام عملیات افزودن طراحی شده است.

- **Full Adder** به عنوان یک بخش مهم در پردازش‌های منطقی استفاده شده است.

- ماژول (**Full Adder**) که در مولتی پلکسر فراخوانی شده بود.

```
module Full_Adder(
    input X1,
    input X2,
    input cin,

    output S,
    output Cout
);
    wire  Xor1 , And1 , And2 ;

    assign  Xor1 = (X1 ^ X2);
    assign S = (cin ^ Xor1);

    assign And1 = (cin & Xor1);
    assign And2 = (X1 & X2);
    assign Cout = (And1 | And2);

endmodule
```

ورودی‌ها:

- X1 و X2: این ورودی‌ها داده‌های ورودی به **Full Adder** هستند.

- cin: ورودی حمل‌برای Full Adder است.

عملکرد Full Adder:

- Full Adder یک مدار منطقی است که دو عدد دودویی را با هم جمع می‌کند.

- در این ماژول، ابتدا با استفاده از یک XOR gate، خروجی Xor1 محاسبه می‌شود. این خروجی نتیجه‌ی XOR دو ورودی X1 و X2 است.

- سپس با استفاده از دو AND gate، خروجی And1 و And2 محاسبه می‌شود. And1 نتیجه‌ی AND بین cin و Xor1 است و And2 نتیجه‌ی AND بین X1 و X2 است.

- خروجی S نتیجه‌ی جمع دو عدد دودویی با استفاده از XOR gateها می‌باشد.

- خروجی Cout نتیجه‌ی حمل (carry) از جمع دو عدد دودویی با استفاده از AND gate ها می‌باشد.

این ماژول به منظور افزودن دو عدد دودویی با استفاده از Full Adder طراحی شده است. خروجی S نتیجه‌ی جمع دو عدد و خروجی Cout نتیجه‌ی حمل از جمع آن‌ها است.

و در پایان

```
always @(*)
        begin
            case ({s3,s2})
                2'b00 : F = D1;
                2'b01 : F = E1;
                2'b10 : F = Ai_1;
                2'b11 : F = Aii;
                default : F = 1'bx;
            endcase
        end
```

1.`always @(*)`:

- @(*) نشان‌دهنده‌ی حساسیت به تغییرات همه‌ی ورودی‌ها است، به این معنی که هر زمان که یکی از ورودی‌ها تغییر کند، بلوک always اجرا می‌شود.

2. عملکرد بلوک ' always ' :

- در این بلوک، با استفاده از مقادیر s3 و s2 ، یکی از مقادیر D1 ، E1 ، Ai_1 یا Aii به عنوان خروجی F انتخاب می‌شود.

- اگر مقادیر s3 و s2 نامعتبر باشند، خروجی F به عنوان 1’bx تعیین می‌شود.

به طور خلاصه، این بلوک always با توجه به مقادیر s3 و s2، یکی از مقادیر مشخص شده را به عنوان خروجی F انتخاب می‌کند.

ماژول واحد حساب ، منطق و شیفت (module Logic_Shift) :

```
module Logic_Shift(
    input s0,
    input s1,
    input s2,
    input s3,
    input Ai,
    input Bi,
    input Ai_1,
    input Aii,
    output reg F
);
    wire E1 , D1 ;

    Mux_gate_1to1 U1(
        .S0(s0),
        .S1(s1),
        .A(Ai),
        .B(Bi),

        .Ei(E1)
    );
    Mux_Gate_4to1_FA U2(
        .S0(s0),
        .S1(s1),
        .CIN(s2),
        .A0(Ai),
        .B0(Bi),
        .O(E1),

        .D(D1)
    );
    always @(*)
        begin
            case ({s3,s2})
                2'b00 : F = D1;
                2'b01 : F = E1;
                2'b10 : F = Ai_1;
                2'b11 : F = Aii;
                default : F = 1'bx;
            endcase
        end
endmodule
```

ماژول مولتی پلکسر 1به1 (Mux_gate_1to1):

```
module Mux_gate_1to1(

    input S0,
    input S1,
    input A,
    input B,
    output reg Ei
);

always @(*)
    begin
        case({S1,S0})
            2'b00 :  Ei = A & B;
            2'b01 :  Ei = A | B;
            2'b10 :  Ei = A ^ B;
            2'b11 :  Ei = ~(A) ;
            default:  Ei = 1'bx;
        endcase
    end


endmodule
```

ماژول مولتی پلکسر 4 به1 (Mux_gate_4to1):

```
module Mux_Gate_4to1_FA(
    input S0,
    input S1,
    input CIN,
    input A0,
    input B0,
    input O,

    output D
);

    wire reg Y0;

    assign Y0 = (S0) ? B0 : ~B0;
    assign D = (S1) ? Y0 : ~Y0;

    Full_Adder U1(
        .X1(A0),
        .X2(Y0),
        .cin(CIN),

        .S(D)
    );


endmodule
```

ماژول فول ادر (Full Adder):

```
module Full_Adder(
    input X1,
    input X2,
    input cin,

    output S,
    output Cout
);
    wire  Xor1 , And1 , And2 ;

    assign  Xor1 = (X1 ^ X2);
    assign S = (cin ^ Xor1);

    assign And1 = (cin & Xor1);
    assign And2 = (X1 & X2);
    assign Cout = (And1 | And2);

endmodule
```

ساخته شده در 1403/03/31

اسامی اعضای گروه: امیررضا فرجی،محمد مبین سعیدی،امیرحسین خدادادی