

برای پیاده‌سازی رفتاری، از `@(in) always` استفاده میکنیم تا زمانی که ورودی `in` عوض شد، خروجی نیز بروزرسانی شود. توجه بفرمایید که `out[i]` زمانی برابر یک است که ورودی `i == in` باشد. بنابراین می‌توانیم به شکل زیر پیاده‌سازی را انجام دهیم.

```
module decoder2_4 (in, out0, out1, out2, out3);
    input [1:0] in;
    output reg out0, out1, out2, out3;

    always @(in) begin
        out0 = (in == 2'b00);
        out1 = (in == 2'b01);
        out2 = (in == 2'b10);
        out3 = (in == 2'b11);
    end
endmodule
```

پیاده‌سازی جریان داده‌ای نیز به طور مشابه خواهد بود. البته توجه بفرمایید که دیگر نیازی به اینکه `output reg` استفاده کنیم وجود ندارد.

```
module decoder2_4 (in, out0, out1, out2, out3);
    input [1:0] in;
    output out0, out1, out2, out3;

    assign out0 = (in == 2'b00);
    assign out1 = (in == 2'b01);
    assign out2 = (in == 2'b10);
    assign out3 = (in == 2'b11);
endmodule
```

در نهایت میتوانیم به شکل زیر تست بنویسیم.







```
module test;
    reg [1:0] in;
    wire out0, out1, out2, out3;

    decoder2_4 decoder2_4_instance(in, out0, out1, out2, out3);

    initial begin
        $monitor($time, " %b %b %b %b", out0, out1, out2, out3);

        #1 in = 2'b00;
        #1 in = 2'b01;
        #1 in = 2'b10;
        #1 in = 2'b11;
        #1 $finish;
    end
endmodule
```

نتیجه شکل موج برای هر دو به شکل زیر خواهد بود.

		/test/in	2'h3		0	1	2	3
		/test/out0	1'h0					
		/test/out1	1'h0					
		/test/out2	1'h0					
		/test/out3	1'h1					

توجه بفرمایید که نام ماژول‌ها متناسب با نوع پیاده‌سازی متفاوت است.

توجه بفرمایید که هر دو پیاده‌سازی در فایل decoder2_4 قرار دارند. یکی از آنها به صورت کامنت در آمده است.