بسمه تعالی

گزارش کار سوم آزمایشگاه طراحی سیستم های دیجیتال

توصیف جریان داده

استاد

دکتر انصاری

نویسنده

سیدعماد امام جمعه – ۴۰۰۱۰۸۷۷۴

آرش ضیایی رازبان – ۴۰۰۱۰۵۱۰۹

محمدعرفان سلیما – ۴۰۰۱۰۵۰۱۴

دانشگاه صنعتی شریف

تابستان 1402

مقدمه

در ۲ بخش این آزمایش هدف طراحی یک comparator با روش data-flow با استفاده از دستور assign است. در بخش اول این مقایسه کننده به صورت ترکیبی و با ساختار سلسله مراتبی طراحی شده و در بخش دوم به صورت ترتیبی طراحی شده و ساختار سلسله مراتبی ندارد. کدهای Verilog در VSCode زده شده و با iVerilog کامپایل و اجرا شده است.

۱

پیاده سازی

پیاده سازی بخش اول شامل 2 ماژول bit_comparator که معادل cascadable 1-bit comparator و comparator است که معادل مقایسه کننده 4 بیتی خواسته شده نهایی است. پیاده سازی این 2 ماژول در وریلاگ به شکل زیر است:

```
module bit_comparator (input wire a, b, Gin, Ein, Lin, output wire Gout, Eout,
Lout);
    assign Eout = Ein && (a == b);
    assign Gout = a > b || ((a == b) && Gin);
    assign Lout = a < b || ((a == b) && Lin);
endmodule
```

که در آن Gin, Ein, Lin حاصل compare کردن بیت های قبلی با ارزش کمتر است که به این ماژول ورودی داده میشود و Gout, Eout, Lout خروجی های مدارند. توجه کنید که 1 بودن G به معنای بزرگتر بودن a است. و همچنین فرض شده که دقیقا یکی از ورودی های Gin, Ein, Lin 1 هستند و با این فرض تضمین میشود از بین خروجی ها هم فقط یکی 1 باشد.

```
module comparator(input [3:0]a, b, output G, E, L);
    wire [3:0]Gi;
    wire [3:0]Ei;
    wire [3:0]Li;

    bit_comparator bc0(a[0], b[0], 1'b0, 1'b1, 1'b0, Gi[0], Ei[0], Li[0]);
    bit_comparator bc1(a[1], b[1], Gi[0], Ei[0], Li[0], Gi[1], Ei[1], Li[1]);
    bit_comparator bc2(a[2], b[2], Gi[1], Ei[1], Li[1], Gi[2], Ei[2], Li[2]);
    bit_comparator bc3(a[3], b[3], Gi[2], Ei[2], Li[2], Gi[3], Ei[3], Li[3]);

    assign G = Gi[3];
    assign E = Ei[3];
    assign L = Li[3];
endmodule
```

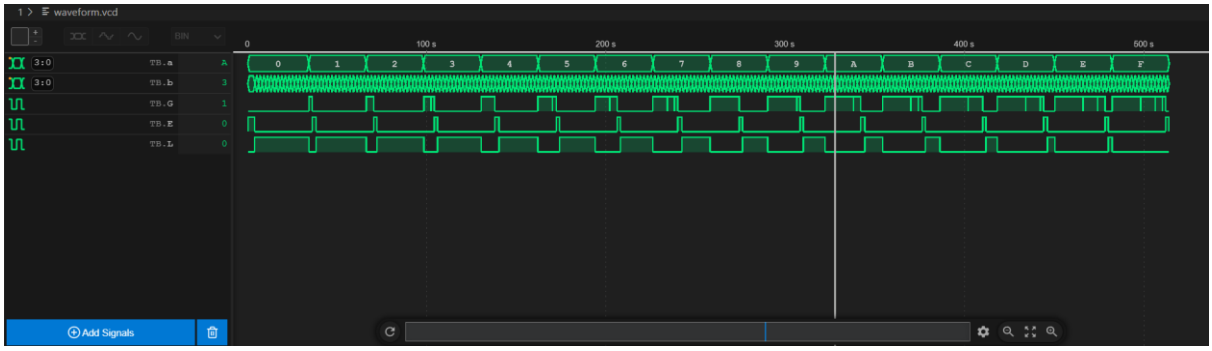
در این ماژول از 4 instance از ماژول bit_comparator استفاده شده که به صورت cascaded یک مقایسه کننده 4 بیتی تولید کنند. Gi, Ei, Li سیم هایی هستند که صرفا برای ایجاد ارتباط بین ماژول ها به کار میروند.

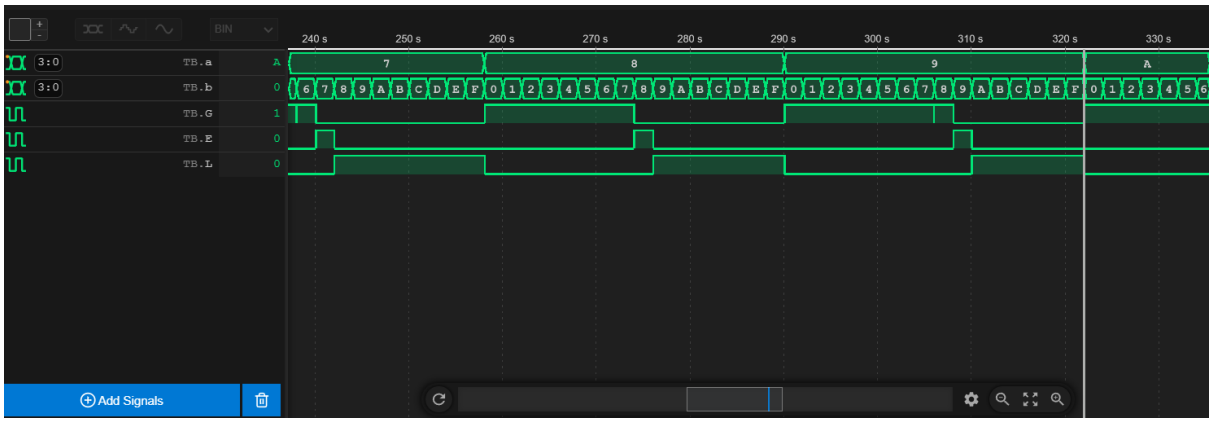
Test Bench

برای تست کردن ماژول های بالا تست بنچ صفحه بعد را طراحی کردیم که تک تک حالات ورودی ممکن را تولید میکند.

```
module TB;
    reg [3:0] a, b;
    wire G, E, L;

    comparator cmp(a, b, G, E, L);

    integer i, j;

    initial begin
        $dumpfile("waveform.vcd");
        $dumpvars(0, TB);
        a = 0; b = 0;
        for(i = 0; i < 16; i ++) begin
            for(j = 0; j < 16; j ++) begin
                #2 a = i; b = j;
            end
        end
        #2 $finish();
    end
endmodule
```

WaveForm حاصل شبیه سازی ماژول بالا به شکل زیر خواهد بود که چون جزئیات آن زیاد است به شرح بخشی از آن میپردازیم.



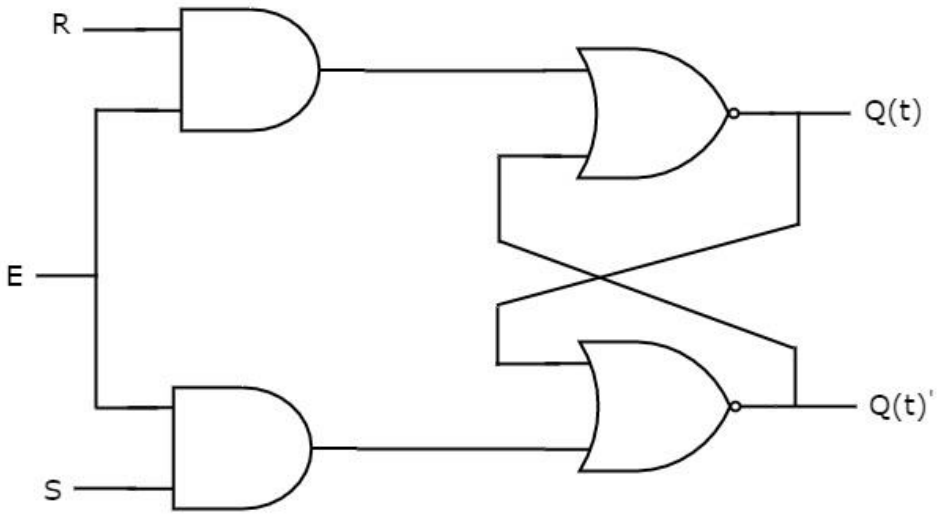


همانطور که میبینید ماژول به درستی بزرگ بودن یا کوچکتر بودن یا برابر بودن a و b را بررسی میکند پس ماژول به درستی کار میکند.

۲

پیاده سازی

در این بخش از آزمایش تنهای لازم است که یک ماژول مقایسه کننده طراحی کنیم. برای اینکه بتوانیم ساختار ترتیبی بر مبنای clock به مدارمان بدهیم، از یک D-Latch استفاده میکنیم که ساختار شماتیک زیر را دارد:



با توجه به ساختار بالا یک ماژول طراحی میکنیم که بیت های ۲ عدد را به ترتیب از MSB به LSB ورودی میگیرد. پس ارزش بیت های ورودی گرفته شده قبلی بیشتر است و اگر یکی از بیت های قبلی نتیجه بزرگ یا کوچکتر به ما بدهند آن نتیجه باقی خواهند ماند. پس در نهایت پیاده سازی این ماژول در Verilog به شکل زیر خواهد بود:

```
module comparator(input a, b, clk, reset, output G, E, L);
    wire Gn, En, Ln; // inverse of G E L
    wire Gin, Ein, Lin; // computed G E L based on previous state;

    assign Gin = (~reset) && (G || (E && (a > b)));
    assign Ein = reset || (E && (a == b));
    assign Lin = (~reset) && (L || (E && (a < b)));

    // Latches
    assign G = ~(Gn && ~(Gin && clk));
    assign Gn = ~(G && ~(~Gin && clk));
    assign E = ~(En && ~(Ein && clk));
    assign En = ~(E && ~(~Ein && clk));
    assign L = ~(Ln && ~(Lin && clk));
    assign Ln = ~(L && ~(~Lin && clk));

endmodule
```

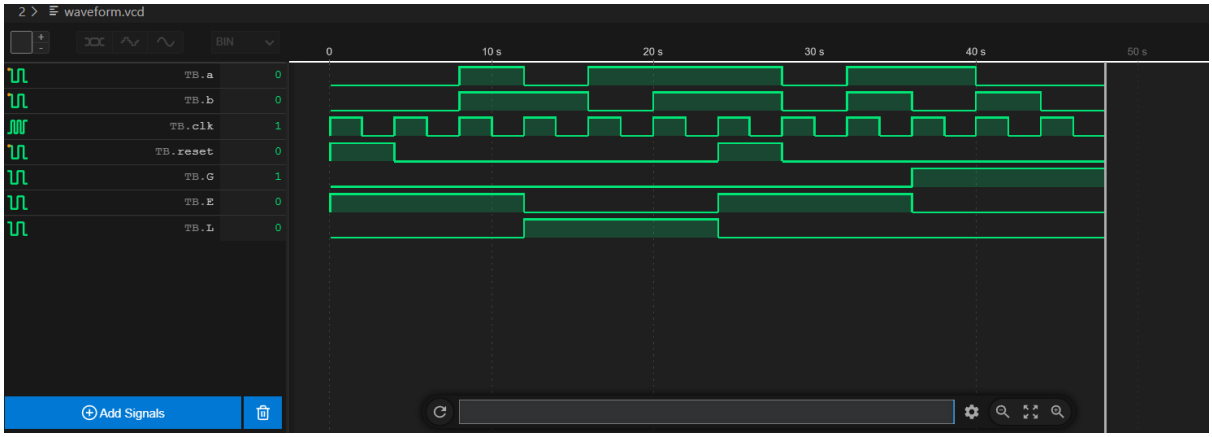
کامنت های کد گویای وظیفه بخش های مختلف هستند. ۳ assign اول به Gin, Ein, Lin هم منطق اصلی مدار هستند که با توجه به استیت های قبلی و ورودی ها latch را معین میکنند.

Test Bench

برای تست کردن ماژول بالا Test Bench زیر را طراحی میکنیم که اعداد 01011 با 01101 و 01100 با 01010 را به ترتیب مقایسه میکند.

```
module TB;
    reg a, b, clk = 1, reset;
    wire G, E, L;

    comparator cmp(a, b, clk, reset, G, E, L);

    always #2 clk = ~clk;

    initial begin
        $dumpfile("waveform.vcd");
        $dumpvars(0, TB);
        reset = 1; a = 0; b = 0;
        #4 reset = 0; a = 0; b = 0;
        #4 a = 1; b = 1;
        #4 a = 0; b = 1;
        #4 a = 1; b = 0;
        #4 a = 1; b = 1;
        #4 reset = 1;
        #4 reset = 0; a = 0; b = 0;
        #4 a = 1; b = 1;
        #4 a = 1; b = 0;
        #4 a = 0; b = 1;
        #4 a = 0; b = 0;
        #4 $finish();
    end
endmodule
```

پس از simulate کردن Test Bench بالا به WaveForm زیر خواهیم رسید:



میتوان دید که مدار در هر لحظه با توجه به بیت هایی که ورودی گرفته به درستی G,E,L را مقدار دهی میکند که میتوانیم نتیجه بگیریم مدار به درستی کار میکند.

پیاده سازی بر FPGA

مدار آزمایش اول به درستی بدون نیاز به تغییر پیاده سازی شد.

اما در مدار آزمایش دوم به دلیل اینکه نمیشد ورودی ها را همزمان با کلاک تغییر داد، Latch ها را با Flip-Flop جایگزین کردیم. که یعنی در واقع از دو Latch با clock های مکمل همدیگر استفاده میکنیم. کد نهایی Verilog به شکل زیر خواهد بود:

```
module comparator(input a, b, clk, reset, output G, E, L);
    wire Gn, En, Ln; // inverse of G E L
    wire Gin, Ein, Lin; // computed G E L based on previous state;

    assign Gin = (~reset) && (G || (E && (a > b)));
    assign Ein = reset || (E && (a == b));
    assign Lin = (~reset) && (L || (E && (a < b)));

    // 1st layer Latches
    assign G1 = ~(G1n && ~(Gin && ~clk));
    assign G1n = ~(G1 && ~(~Gin && ~clk));
    assign E1 = ~(E1n && ~(Ein && ~clk));
    assign E1n = ~(E1 && ~(~Ein && ~clk));
    assign L1 = ~(L1n && ~(Lin && ~clk));
    assign L1n = ~(L1 && ~(~Lin && ~clk));
    // 2nd layer Latches
    assign G2 = ~(G2n && ~(G1 && clk));
    assign G2n = ~(G2 && ~(~G1 && clk));
    assign E2 = ~(E2n && ~(E1 && clk));
    assign E2n = ~(E2 && ~(~E1 && clk));
    assign L2 = ~(L2n && ~(L1 && clk));
    assign L2n = ~(L2 && ~(~L1 && clk));

    assign G = G2;
    assign E = E2;
    assign L = L2;
endmodule
```

منابع:

Mano, Morris. *Computer system architecture*. Prentice-Hall of India, 2003.