گزارش آزمایش سوم آزمایشگاه
طراحی سیستم دیجیتال

مزدک تیموریان

۴۰۱۱۰۱۴۹۵

مریم شیران

۴۰۰۱۰۹۴۴۶

مهدی بهرامیان

۴۰۱۱۷۱۵۹۳

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

تیر ۱۴۰۳

فهرست مطالب

۱ شرح آزمایش ۲

۱.۱ مقایسه کننده ۱ بیتی آبشاری ۲

۲.۱ مقایسه کننده ۴ بیتی ۲

۳.۱ مقایسه کننده سریالی ۳

۲ شبیه سازی و بررسی عملکرد مقایسه کننده ها ۴

۱.۲ مقایسه کننده ۴ بیتی ۴

۲.۲ مقایسه کننده سریالی ۵

۱ شرح آزمایش

۱.۱ مقایسه کننده ۱ بیتی آبشاری

این مقایسه کننده به این صورت کار میکند که وضعیت رقم های بزرگتر را میگیرد و اگر در آن رقوم دو عدد برابر بودند، خروجی برابر با نتیجه مقایسه دو رقم کنونی میشود و در غیر این صورت خروجی برابر با مقایسه رقوم بزرگتر میشود.

```
module casc_comp (
    input  i_g,
    input  i_e,
    input  i_s,
    input  a,
    input  b,
    output o_g,
    output o_e,
    output o_s
);
    assign o_g = i_g || i_e && (a && ~b);
    assign o_e = i_e && (a ^ ~b);
    assign o_s = i_s || i_e && (~a && b);
endmodule
```

همانطور که در کد وریلاگ بالا میبینید، این عملکرد با استفاده از عملگر های ساده منطقی پیاده سازی شده است.

۲.۱ مقایسه کننده ۴ بیتی

برای مقایسه کردن دو عدد ۴ رقمی کافیست ۴ مقایسه کننده آبشاری را پشت یکدیگر بگذاریم و حالت اولی را با تساوی شروع کنیم.

```
module full_comp #(
    parameter N = 4
) (
    input [N-1:0] a,
    input [N-1:0] b,
    output g,
    output e,
    output s
);
    wire [N:0] midg, mide, mids;
    assign midg[N] = 0;
    assign mide[N] = 1;
    assign mids[N] = 0;
    generate
        genvar i;
        for (i = N; i > 0; i = i - 1) begin : casc
            casc_comp _casc (
                .i_g(midg[i]),
                .i_e(mide[i]),
                .i_s(mids[i]),
                .a  (a[i-1]),
                .b  (b[i-1]),
                .o_g(midg[i-1]),
                .o_e(mide[i-1]),
                .o_s(mids[i-1])
            );
        end
    endgenerate
    assign g = midg[0];
    assign e = mide[0];
    assign s = mids[0];
endmodule
```

همانطور که در کد وریلاگ بالا مشاهده میکنید، این عملکرد با استفاده از یک بلوک generate پیاده سازی شده است.

۳.۱ مقایسه کننده سریالی

برای مقایسه کردن دو عدد سریالی کافیست برای وضعیت مقایسه دو عدد تا کنون ۳ ثبات نگهداریم که این ثبات ها را میتوان با استفاده از تکنیک master-slave طراحی کنیم. سپس آنها را به ورودی مقایسه کننده آبشاری بدهیم و خروجی آنرا به ورودی ثبات ها و خروجی کلی مدارمان وصل کنیم.

```
module serial_comp (
    input  serial_a,
    input  serial_b,
    input  clk,
    input  rst,
    output g,
    output e,
    output s
);
    wire mg = rst ? 0 : clk ? mg : g;
    wire qg = rst ? 0 : clk ? mg : qg;
    wire me = rst ? 1 : clk ? me : e;
    wire qe = rst ? 1 : clk ? me : qe;
    wire ms = rst ? 0 : clk ? ms : s;
    wire qs = rst ? 0 : clk ? ms : qs;

    casc_comp _cmp (
        .i_g(qg),
        .i_e(qe),
        .i_s(qs),
        .a  (serial_a),
        .b  (serial_b),
        .o_g(g),
        .o_e(e),
        .o_s(s)
    );
endmodule
```

به طور کلی توصیف ما به صورت کد وریلاگ بالا قابل پیاده سازی است.

۲ شبیه سازی و بررسی عملکرد مقایسه کننده ها

۱.۲ مقایسه کننده ۴ بیتی

برای آزمون کارایی این مدار کافیست در یک تست_بنچ ورودی های مختلفی را به مدار داد و رفتار مدار را آزمود.

```
# 12 vs 12 ⇒ 010
# 14 vs 12 ⇒ 100
# 14 vs  4 ⇒ 100
#  3 vs  4 ⇒ 001
#  3 vs  3 ⇒ 010
#  7 vs  3 ⇒ 100
#  7 vs 15 ⇒ 001
```

همانطور که در نتیجه اجرای تست بنچ میبینید، نتیجه این مقایسه ها به درستی محاسبه شده اند. (معنی بیت ها به ترتیب : $a > b|a = b|a < b$)

۲.۲ مقایسه کننده سریالی

برای آزمایش مقایسه کننده سریالی کافیست تعدادی دفعه مدار را ریست کنیم و سپس در هر مرحله ورودی های مورد نظر را به صورت رقم به رقم، هر رقم در یک کلاک وارد میکنیم. در این صورت انتظار میرود که عملکرد مدار به این صورت باشد که اولین کلاکی که ورودی ها متفاوت بودند، نتیجه مقایسه را تعیین کند.

```
# reset
# 00 ⇒ 010
# 11 ⇒ 010
# 00 ⇒ 010
# 11 ⇒ 010
# 10 ⇒ 100
# reset
# 00 ⇒ 010
# 01 ⇒ 001
# 10 ⇒ 001
# 00 ⇒ 001
# reset
# 00 ⇒ 010
# 10 ⇒ 100
# 00 ⇒ 100
# 01 ⇒ 100
# 11 ⇒ 100
```

همانطور که میبینید عملکرد مدار در این ۳ آزمون درست بوده است.