گزارش آزمایش چهارم

دانشگاه صنعتی شریف - زمستان ۹۹

آزمایشگاه طراحی سیستم‌های دیجیتال - دکتر اجلالی

نویسندگان:
سروش جهان‌زاد - ۹۸۱۰۰۳۸۹
علی حاتمی تاجیک - ۹۸۱۰۱۳۸۵

۱ مقدمه

در این سند سعی شده تا گزارشی بر نحوه انجام آزمایش چهارم درس آزمایشگاه طراحی سیستم‌های دیجیتال ارائه شود که در آن یک استک به عمق ۸ و عرض ۴ طراحی شده است. کدهای مربوط به هر بخش، فایل تست بنچ آن و مدیل مربوط به گزارش در کنار این گزارش آمده است. از صبر و بردباری شما در مطالعه این سند بسیار سپاسگزاریم.

با احترام

سروش جهان‌زاد، علی حاتمی تاجیک – بهار ۱۴۰۰

۲ ساخت استک

طراحی مدار

از ما خواسته شده است تا یک پشته با عمق هشت و عرض چهار بسازیم. برای ذخیره مقادیر استک از آرایه ای از رجیسترهای چهاربیتی stack استفاده شده است و برای نگهداری از مقدار خروجی استک نیز از یک رجیستر چهاربیتی out استفاده شده است. همینطور از یک رجیستر چهاربیتی emptyPos برای نگهداری اشاره‌گری به خانه خالی استک استفاده شده است.

سیگنال empty به صفر بودن emptyPos اساین شده است و full نیز به هشت بودن آن (توجه شود که emptyPos به خانه خالی بعدی اشاره میکند پس اگر صفر باشد در پوش بعدی درون خانه صفر مقدار نوشته خواهد شد و اگر هشت باشد یعنی هر هشت خانه استک پر شده است و ما در ایندکس نهم استک هستیم که غیرقابل نوشتن است).

مدار ما دو فرایند خواهد داشت. یکی چهار حالت عملیاتی که در ادامه توضیح داده خواهد شد و یک فرایند ریست که به کلاک وابسته نخواهد بود و هر زمان و برای هر مدت زمانی فشار داده شود باید استک به حالت اولیه خود برگردد. چون در سؤال ذکر نشده بود با فشار دادن ریست اطلاعات استک در آن باقی خواهد ماند اما مقدار رجیستر خروجی صفر خواهد شد. برای اینکه مشکل همراه شدن لبه بالارونده را با پالس ریست از بین ببریم داخل شرط بلاک always این شرط صفر بودن ریست را نیز با لبه بالارونده کلاک همراه می‌کنیم.

در چهار حالت در استک ما تغییر ایجاد می‌شود که جلوتر توضیح داده خواهند شد و در باقی موارد استک بدون تغییر باقی خواهد ماند:

- سیگنال پوش و پاپ همزمان فعال باشد اما داده‌ای برای پاپ کردن وجود نداشته باشد در این حالت دیتای ورودی در خانه اول استک باید نوشته شود و به اشاره‌گر یک واحد افزوده شود.
- سیگنال پوش و پاپ فعال باشد و داده برای پاپ کردن وجود داشته باشد. در این حالت داده خانه قبلی اشاره‌گر به خروجی می‌رود و سپس ورودی در همان خانه نوشته خواهد شد. در این حالت اشاره‌گر تغییر نخواهد کرد چون هم داده وارد و هم خارج شده است. همینطور در اینجا محدودیتی برای پر بودن استک برای پوش کردن نخواهیم داشت چون در جای آخرین دیتایی که وارد شده که حتماً جای قابل قبولی بوده است دیتای جدید نوشته خواهد شد.
- تنها سیگنال پوش فعال باشد و استک هم پر نباشد در این حالت پر بودن استک میتواند با استفاده از همان سیگنال خروجی که در ابتدا اساین شده است برسی شود. اگر استک پر نباشد داده در خانه خالی که اشارهگر به آن اشاره میکند نوشته می‌شود و اشارهگر یک واحد افزوده خواهد شد.
- تنها سیگنال پاپ فعال باشد و استک خالی نباشد. در این حالت خالی بودن استک با استفاده از همان سیگنال خروجی که در ابتدا اساین شده است برسی شده است. اگر استک خالی باشد از مقدار اشاره گر یکی کم می‌شود (به خانه‌ای که میخواهد پاپ شود اشاره میکند) و سپس مقدار آن ایندکس برگردادنه می‌شود. حال آن خانه یک خانه خالی برای نوشتن اطلاعات پوش احتمالی بعدی است و یک خانه خالی محسوب می‌شود و مقدار آن دیگر برای ما مهم نیست.

کد مدار

دو فرایند ریست شدن یا چهار حالت عملیاتی به وسیله دو بلاک always پیاده‌سازی شده‌اند. چهار سیگنال خروجی هم به این صورت اساین شده‌اند که رجیستری که مقدار خروجی را نگه میدارد به data_out خواهد رفت، برابری emptyPos با صفر به empty خواهد رفت و برابری آن با هشت به full خواهد رفت.

در بلاک ابتدایی always که ریست شدن کنترل می‌شود هر زمانی که سیگنال ورودی ریست یک شود emptyPos و out صفر می‌شوند.

در بلاک always دیگر چهار حالت عملیاتی که پیشتر توضیح داده شد به وسیله بلاکهای if و else از هم جدا شده‌اند و هر موقع هرکدام اتفاق بیفتد عملیاتهای توضیح داده شده درون آن‌ها به همان ترتیب گفته شده انجام می‌شود. در ادامه کد مربوط به ماژول استک آمده است:

```
module stack(
    input clk, input rstN, input [3:0] data_in, input push, input pop,
    output [3:0] data_out, output full, output empty
    );
    // point to the empty index of stack
    reg [3:0]emptyPos = 0;
    // stack with depth 8 and width 4
    reg [3:0]stack[0:7];
    // out register
    reg [3:0] out = 0;
```

```
    // assign empty and full signals
    assign full = (emptyPos == 8);
    assign empty = (emptyPos == 0);
    assign data_out = out;

    // asynchronized reset stack signal
    always @ (rstN) begin
        emptyPos <= 0;
        out <= 0;
    end

    always @ (posedge clk && ~rstN) begin
        if (pop && push) begin
            if (empty) begin // If Pop and Push together but pop can't be done
                stack[emptyPos] = data_in;
                emptyPos = emptyPos + 1;
            end else begin // push and pop together
                out = stack[emptyPos - 1];
                stack[emptyPos - 1] = data_in;
            end
        end else if (pop && ~empty) begin // if only pop and pop is valid
            emptyPos = emptyPos - 1;
            out = stack[emptyPos];
        end else if (push && ~full) begin // if only push and push is valid
            stack[emptyPos] = data_in;
            emptyPos = emptyPos + 1;
        end
    end
endmodule
```

تست

مدار را در چند بخش تست میکنیم:

۱. ابتدا ۵ داده به استک پوش میکنیم (۵ داده درون استک)

۲. آخرین داده را پاپ میکنیم (۴ داده درون استک)

۳. یک داده (مثلاً چهار) را به استک پوش میکنیم و در همان کلاک داده قبلی را پاپ میکنیم (۴ داده درون استک است، و چهارمین عدد پوش شده در مرحله ۱ باید درون خروجی باشد)

۴. یک داده را پاپ میکنیم (باید همان داده پوش شده در مرحله قبل باشد، ۳ داده درون استک)

۵. ۶ داده به استک پوش میکنیم (۵ داده پوش خواهد شد. یکی اضافه است و باید از آن صرف نظر شود، ۸ داده درون استک و پر است)

۶. سه داده را پاپ میکنیم (۵ داده درون استک، داده‌ها باید صورت LIFO پاپ شوند)

۷. استک را ریست میکنیم (هیچ داده‌ای درون استک نیست و خالی است)

۸. دو بار پاپ میکنیم (هیچ داده‌ای درون استک نیست و پاپ ها غیر مجاز است و داده خروجی باید همان مقدار قبل باقی بماند)

۹. یک داده پوش کرده و در کلاک بعدی آنرا پاپ میکنیم (یک لحظه پر و در کلاک بعدی خالی می‌شود. برای درستی کاربرد بعد از پاپ کردن اضافه)

۱۰. داده ورودی را چند بار با خاموش بودن سیگنال پوش تغییر میدهیم تا ببینیم با صفر بودن سیگنال ها تغییری در مدار ایجاد نمیشود (همچنان استک خالی است)

کد تست

ماژولی که در بخش کد مدار نوشته بودیم را به وسیله کد زیر امتحان خواهیم کرد(بخشهای بالا درون کد مشخص شده اند):

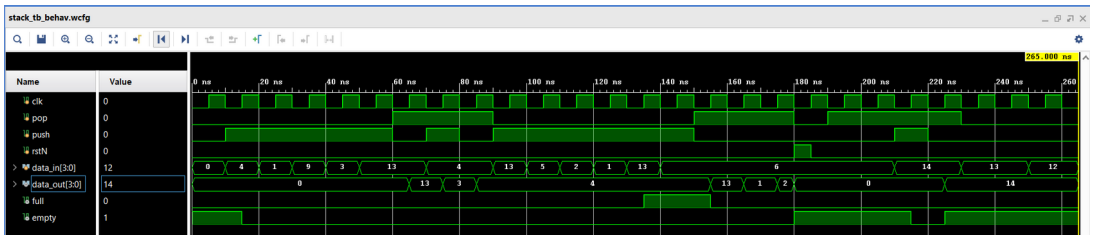
```
module stack_tb();
    reg clk = 0, pop = 0, push = 0, rstN = 0;
    reg [3:0]data_in = 0;
    wire [3:0] data_out;
    wire full, empty;
    stack uut(
        .clk (clk),.rstN (rstN),.data_in (data_in),.push (push),.pop (pop),
        .data_out (data_out),.full (full),.empty (empty)
    );
```

```
    // Generating Clock Signal
    initial begin
        forever
            #5 clk = ~clk;
    end
    integer i = 0;
    initial begin
        #10;
        // push 5 data every clock
        push = 1;
        for (i = 0; i<5; i = i + 1) begin data_in = $random; #10; end
        // pop last data
        push = 0;
        pop = 1; #10;
        // push pop in one clock
        data_in = 4;
        push = 1; #10;
        // pop again result must be 4
        push = 0; #10;
        // push 6 data (5 valid push / one invalid push)
        pop = 0; push =1;
        for (i = 0; i<6; i = i + 1) begin data_in = $random; #10; end
        // pop last 3 data (it should be the 5th data in last push part)
        push = 0;
        pop =1; #30;
        // reset the stack
        pop =0; rstN = 1; #5 rstN = 0; #5;
        // 2 invalid pop
        pop = 1; #20;


        // push a new data with pop on and pop it
        data_in = 14; push = 1; #10; // push data with push and pop on but pop is invalid
        push = 0; #10; //pop the last entry
        // change in data with push off
        pop = 0;
        for (i = 0; i<3; i = i + 1) begin data_in = $random; #10; end // change data in with push off
        #5 $finish;
    end
endmodule
```

نتیجه تست

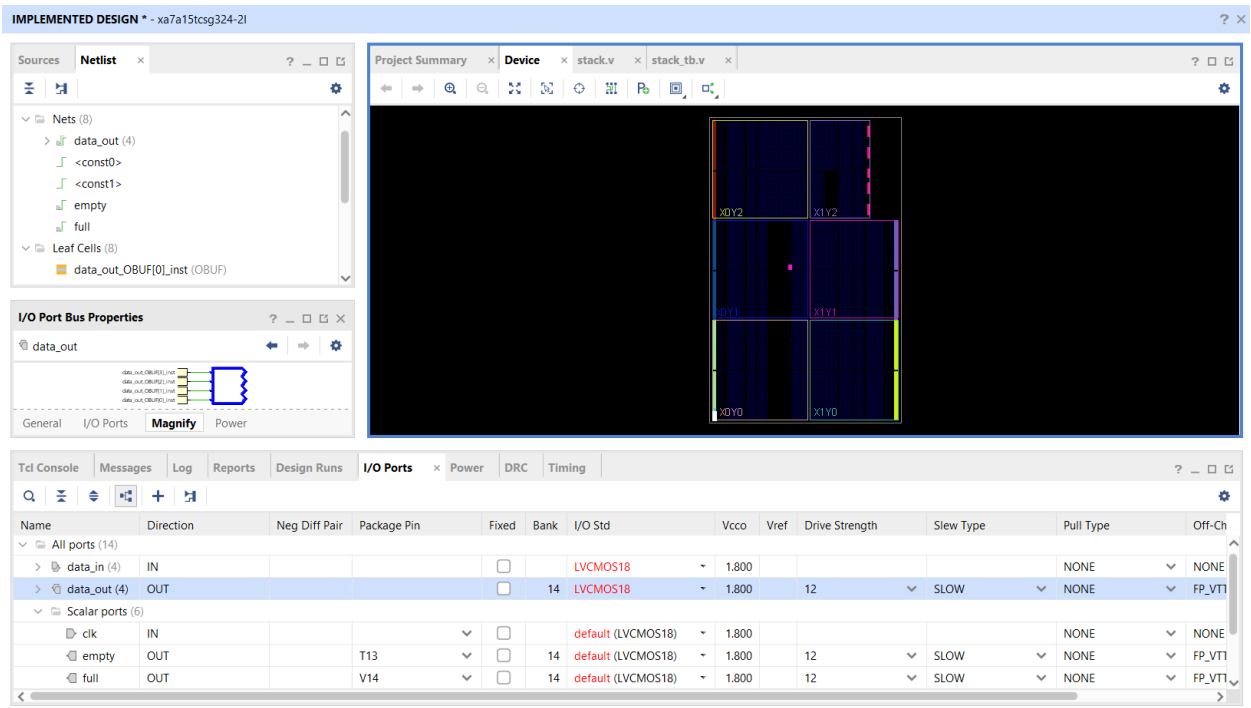
در شکل زیر تیجه اجرای تست بالا را میبینیم که خروجی ها تماماً مطابق انتظار توضیح داده شده در تست است:



شکل ۱: نتیجه تست مدار پشته طراحی شده

سنتز

در نهایت برای اطمینان از درستی کامل و سنتزپذیری مدار، با استفاده از ابزار سنتز ویوادو آن را برای یکی از نمونه‌های موجود در این نرم افزار سنتز می‌کنیم. در این آزمایش از xa7a15tcsg324-2l استفاده می‌کنیم. پس از انجام سنتز، مشاهده می‌کنیم که همه چیز به درستی پیش رفته و آزمایش موفقیت‌آمیز بوده است.



شکل ۲: سنتزپذیری