

# گزارش آزمایش چهارم



دانشگاه صنعتی شریف - زمستان ۹۹

آزمایشگاه طراحی سیستم‌های دیجیتال - دکتر اجاللی

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 مدیله 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 اشاره گری 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. برای سیگنال rstN بلاک always را حساس به لبه بالارونده کلاک و لبه پایین رونده سیگنال ریست نات (این سیگنال را Active Low در نظر گرفتیم) تعریف کرده ایم. اگر سیگنال ریست فعال بود (چه در بالارونده کلاک چه در بالارونده ریست)، استک را ریست می کنیم و کار دیگری انجام نخواهیم داد. با این کار از تلاقی دوبلاکی شدن ریست و کلاک جلوگیری می کنیم.

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

- سیگنال پوش و پاپ همزمان فعال باشد اما داده ای برای پاپ کردن وجود نداشته باشد. در این حالت دیتای ورودی در خانه اول استک باید نوشته شود و به اشاره گر یک واحد افزوده شود.
- سیگنال پوش و پاپ فعال باشد و داده برای پاپ کردن وجود داشته باشد. در این حالت داده خانه قبلی اشاره گر به خروجی می رود و سپس ورودی در همان خانه نوشته خواهد شد. در این حالت اشاره گر 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 بررسی شود. اگر استک پر نباشد داده در خانه خالی که اشاره گر به آن اشاره میکند نوشته می شود و اشاره گر 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 بررسی شده است. اگر استک خالی باشد از مقدار اشاره گر یکی کم می شود (به خانه ای که میخواهد پاپ شود اشاره میکند) و سپس مقدار آن ایندکس برگردانده می شود. حال آن خانه یک خانه خالی برای نوشتن اطلاعات پوش احتمالی بعدی است و یک خانه خالی محسوب می شود و مقدار آن دیگر برای ما مهم نیست.

### کد مدار

دو فرایند ریست شدن یا چهار حالت عملیاتی به وسیله یک بلاک always پیاده سازی شده اند. چهار سیگنال خروجی هم به این صورت اساین شده اند که رجیستری که مقدار خروجی را نگه می دارد به data\_out خواهد رفت، برابری emptyPos با صفر به empty خواهد رفت و برابری آن با هشت به full خواهد رفت.

در بلاک always ابتدا فعال بودن ریست (صفر بودن آن، چون rstN به صورت Active Low است) چک می شود و سپس چهار حالت عملیاتی که پیش تر توضیح داده شد به وسیله بلاکهای if و else از هم جدا شده اند و هر موقع هر کدام اتفاق بیفتد عملیاتهای توضیح داده شده درون آن ها به همان ترتیب گفته شده انجام می شود. 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;

always @ (posedge clk or negedge rstN) begin
    if (~rstN) begin
        emptyPos <= 0;
        out <= 0;
    end else 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 - 1];
    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 پاپ‌ها 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 سیگنال‌ها 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 = 1;
    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)
    );
endmodule

```

```

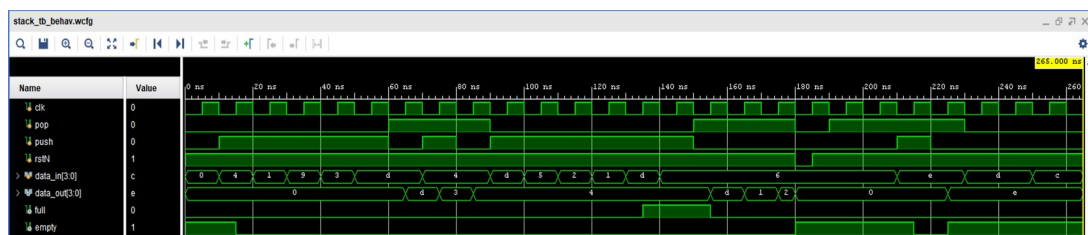
// Generating Clock Signal
initial begin
    forever
        #5 clk = ~clk;
end
integer i = 0;
initial begin
    #10;
    // push 5 data every clock
    push = 1;
    for (i = 0; i < 5; i = i + 1) begin data_in = $random; #10; end
    // pop last data
    push = 0;
    pop = 1; #10;
    // push pop in 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 = 0; #5 rstN = 1; #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 نتیجه 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 xa7a15tcs324-2l استفاده می‌کنیم. پس از انجام سنتز، مشاهده می‌کنیم که همه چیز به درستی پیش رفته و آزمایش موفقیت‌آمیز بوده است.

IMPLEMENTED DESIGN - constrs\_2 | xa7a15kcg324-2i

Sources Netlist

stack

- Nets (95)
- Leaf Cells (87)

I/O Port Bus Properties

data\_out

data\_out\_OBUF[3]\_inst data\_out[3]

data\_out\_OBUF[2]\_inst data\_out[2]

data\_out\_OBUF[1]\_inst data\_out[1]

data\_out\_OBUF[0]\_inst data\_out[0]

General I/O Ports Magnify Power

Project Summary Device stack.v stack\_tb.v

Tcl Console Messages Log Reports Design Runs I/O Ports Power DRC Methodology Timing

Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination	IN_TERM
data_in (4)	IN			<input checked="" type="checkbox"/>	35	LVC MOS18	1.800				NONE	NONE	
data_out (4)	OUT			<input checked="" type="checkbox"/>	15	LVC MOS18	1.800		12	SLOW	NONE	FP_VTT_50	
clk	IN		E15	<input checked="" type="checkbox"/>	15	LVC MOS18	1.800				NONE	NONE	
empty	OUT		K16	<input checked="" type="checkbox"/>	15	LVC MOS18	1.800		12	SLOW	NONE	FP_VTT_50	
full	OUT		J15	<input checked="" type="checkbox"/>	15	LVC MOS18	1.800		12	SLOW	NONE	FP_VTT_50	
pop	IN		K15	<input checked="" type="checkbox"/>	15	LVC MOS18	1.800				NONE	NONE	
push	IN		J18	<input checked="" type="checkbox"/>	15	LVC MOS18	1.800				NONE	NONE	
rstN	IN		J17	<input checked="" type="checkbox"/>	15	LVC MOS18	1.800				NONE	NONE	

شکل ۲: سنتز پذیری