دانشکده مهندسی کامپیوتر

آزمایشگاه طراحی سیستم‌های دیجیتال

آزمایش چهارم - توصیف رفتاری

دکتر اجلالی، مهندس اثنی عشری

امیرمهدی کوششی — ۹۸۱۷۱۰۵۳
ایمان محمدی — ۹۹۱۰۲۲۰۷
شایان صالحی — ۹۹۱۰۵۵۶۱

۲۴ مرداد ۱۴۰۲

فهرست مطالب

۱ مقدمه

در این آزمایش قصد داریم تا پشته‌ای (LIFO) به پهنای ۴ بیت و عمق ۸ طراحی کنیم. مطابق دستور کار ورودی‌ها و خروجی‌های ما به شکل زیر می‌باشد.

Inputs:	Clk	Clock signal
	RstN	Reset signal
	Data_In	4-bit data into the stack
	Push	Push Command
	Pop	Pop Command
Outputs:	Data_Out	4-bit output data from stack
	Full	Full=1 indicates that the stack is full
	Empty	Empty=0 indicates that the stack is empty

مطابق دستورکار، ما ۴ بیت ورودی می‌گیریم که نشان دهنده‌ی دیتایی هست که می‌خواهیم در استک push کنیم. همچنین دو ورودی نیز داریم که push و pop می‌باشد. در این کد نیز کلاک و reset not یا همان RstN داریم.

خروجی‌های این مدار نیز ۴ بیت data out داریم که نشان دهنده‌ی همان دیتایی است که از استک pop کرده‌ایم. همچنین دو خروجی Full نیز که زمانی که همه‌ی ۸تا خانه‌ی آن‌ها پر شود روشن می‌شود و خروجی Empty نیز زمانی که همه‌ی خانه‌های استک خالی باشد و به عبارتی که همه‌ی دیتاهای آن را pop کرده‌باشیم و یا استک را reset کرده باشیم این خروجی فعال می‌شود.

۲ کد

با هر بار کلاک زدن اتفاقات زیر در استک رخ می‌دهد.
۱- در صورتی که RstN صفر باشد، پوینتر ریست می‌شود و به خانه‌ی اول استک اشاره می‌کند. دقت کنید که RstN یا همان reset not نات ریست است و اگر بخواهیم ریست کنیم باید RstN را صفر یا به عبارت دیگر ریست را یک کنیم.

۲- زمانی که push ما ۱ باشد و استک‌مان هنوز فضا برای ذخیره‌سازی داشته باشد یا به عبارتی full نباشد، دیتا را در استک قرار داده و پوینتر را یکی اضافه می‌کنیم. در صورتی که استک full باشد وارد این شرط نمی‌شویم و اتفاقی نمی‌افتد.

۳- زمانی که pop ما ۱ باشد و استک empty نباشد، اخرین خانه‌ای که درون آن دیتا ذخیره کردیم را خوانده و در data out قرار می‌دهیم و پوینتر را یک خانه کم می‌کنیم. در صورتی که استک empty بود وارد این شرط نمی‌شویم و به عبارتی اتفاقی نمی‌افتد.

در نهایت در پایان هر عملیات full و empty را محاسبه می‌کنیم و متناسب با آن آن‌ها را مقداردهی می‌کنیم. full زمانی ۱ می‌شود که پوینتر ما عدد ۸ باشد و یا به عبارتی به آخرین خانه اشاره کند. برای این کار ما پوینتر

را با عدد ۸ xor می‌کنیم. اگر حاصل این xor صفر باشد به این معنی است که پوینتر همان ۸ هست و یا به عبارتی داریم:

$$XOR(pointer, 8) = 0 \longrightarrow pointer = 8$$

پس اگر حاصل آن ۰ شد آن را reduction nor می‌کنیم و حاصل آن ۱ خواهد شد و در full می‌ریزیم.

همچنین برای empty، در صورتی که pointer صفر باشد و یا به عبارتی تمام بیت‌های آن صفر باشد، پس یعنی پوینتر به اولین خانه اشاره می‌کند و در آن مقداری ذخیره نشده است پس در این صورت empty را ۱ می‌کنیم. برای این کار نیز عملیات reduction nor را روی pointer انجام می‌دهیم.

در زیر می‌توانید عکس کد را مشاهده کنید.

```
1  module stack(input Clk, input RstN, input [3:0] Data_In, input Push, input
    Pop, output reg [3:0] Data_Out, output reg Full, output reg Empty);
2
3      reg [3:0] mem [7:0];
4      reg [3:0] pointer = 0;
5
6      always @(posedge Clk or negedge RstN)
7      begin
8          if (~RstN)
9              pointer = 0;
10         else if (Push == 1 & Full == 0)
11         begin
12             mem[pointer] = Data_In;
13             pointer = pointer + 1;
14         end
15         else if (Pop == 1 & Empty == 0)
16         begin
17             Data_Out = mem[pointer-1];
18             pointer = pointer - 1;
19             end
20         Full = ~|(pointer ^ 8);
21         Empty = ~|pointer;
22     end
23 endmodule
```

همانطور که در کد بالا مشاهده می‌کنید، ما یک آرایه دو بعدی گرفته‌ایم که هر خانه‌ی آن ۴ بیت دارد و به طور کلی ۸ خانه یا index دارد. ما از این آرایه که اسم آن mem است برای دیتاهایی که push شده است استفاده می‌کنیم و به عبارتی آن‌ها را در این آرایه ذخیره می‌کنیم. زمانی که pop نیز فعال شود متناسب با پوینتر خانه مرتبط با این آرایه را خوانده و در data out قرار می‌دهیم.

در این کد شما همچنین می‌توانید pointer و عملیات‌هایی که برای مقداردهی full و empty استفاده شده

است را نیز مشاهده کنید.

اگر مشاهده کنید می‌بینید که زمانی که دیتایی push می‌شود ما آن را در index پوینتر مموری‌مان قرار می‌دهیم و یا به عبارتی

$$mem[pointer] = Data_In$$

اما زمانی که می‌خواهیم دیتایی را pop کنیم آن را از خانه‌ی $pointer - 1$ می‌خوانیم. دلیل آن این است که ما زمانی که دیتایی را push کردیم pointer را یک خانه اضافه می‌کنیم. به همین دلیل زمانی که میخواهیم دیتایی را pop کنیم باید خانه‌ی قبلی آن را بخوانیم.

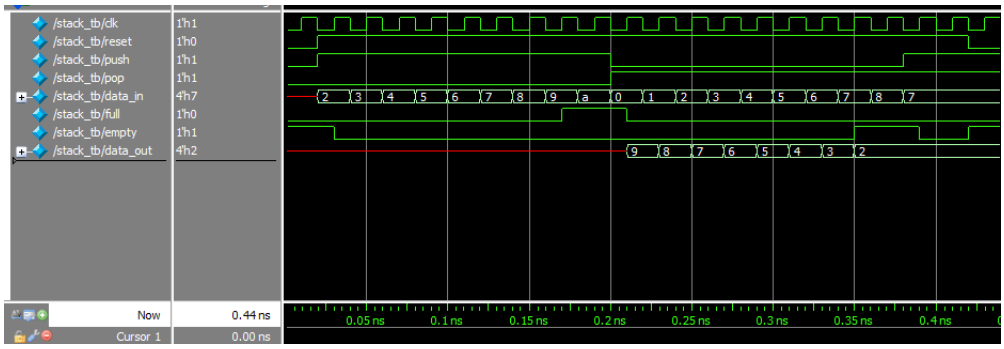
۱.۲ Test Bench

برای این ماژول نیز یک تست بنچ نوشته شده است که می‌توانید در زیر مشاهده کنید:

```
module stack_tb();

    reg         clk = 0, reset = 0, push = 0, pop = 0;
    reg [3:0]   data_in;
    wire        full, empty;
    wire [3:0]  data_out;


stack st(
    .Clk(clk),
    .RstN(reset),
    .Push(push),
    .Pop(pop),
    .Data_In(data_in),
    .Data_Out(data_out),
    .Full(full),
    .Empty(empty)
);

always #10 clk = ~clk;

initial begin
    $monitor("data_out = %d, Full = %d, Empty = %d"
, data_out, full, empty);
    #20 reset = 1;
    push = 1;
    for (data_in = 2; data_in < 11; data_in = data_in + 1) begin
        #20;
    end
    push = 0;
    pop = 1;
    for (data_in = 0; data_in < 9; data_in = data_in + 1) begin
      #20;
    end
    push = 1;
    data_in = 7;
    #20;
    #20 reset = 0;
    #20;
    $stop;
end

endmodule
```

۲.۲ Wave Form

در شکل زیر ویوفرم تست بنچ بالا آمده است.



ابتدا چند بار (۱۰ مرتبه) عملیات push انجام می‌شود. پس از اینکه تمام اعداد ۲ تا ۹ که مجموعا ۸ عدد هستند در این پشته قرار میگیرند، خروجی Full ۱ میشود و دیگر در داخل پشته چیزی قرار نمیگیرد. سپس شروع به pop کردن از پشته میکنیم. همانطور که مشا هده میشود، ویژگی LIFO در این پشته برقرار است و ابتدا ۹، سپس ۸ و ... از پشته خارج میشوند و در نهایت که پشته خالی است خروجی Empty ۱ میشود.

۳.۲ FPGA

حال کد را بر روی FPGA پیاده می‌کنیم. ابتدا از طریق pin planner پین‌های مرتبط به ورودی و خروجی را مشخص می‌کنیم.

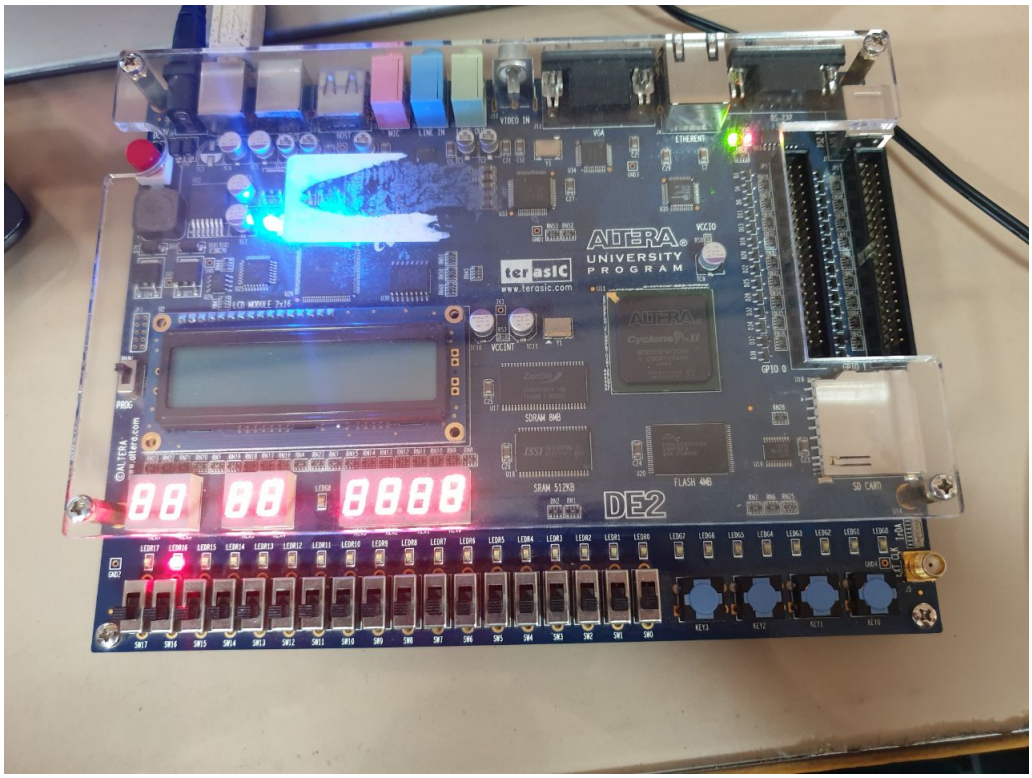
	Node Name	Direction	Reserved	Location
1	Clk	Input		PIN_P25
2	Data_In[3]	Input		PIN_T7
3	Data_In[2]	Input		PIN_P2
4	Data_In[1]	Input		PIN_P1
5	Data_In[0]	Input		PIN_N1
6	Data_Out[3]	Output		PIN_AC22
7	Data_Out[2]	Output		PIN_AB21
8	Data_Out[1]	Output		PIN_AF23
9	Data_Out[0]	Output		PIN_AE23

ما برای این آزمایش، کلاک خود را سوییچ SW2 و RstN را سوییچ SW16 قرار داده‌ایم. همچنین سوییچ‌های push و pop به ترتیب SW15 و SW14 می‌باشد. همچنین ۴ بیت Data In که دیتای ورودی push می‌باشد به ترتیب از بیت پر ارزش SW13 الی SW10 می‌باشد.

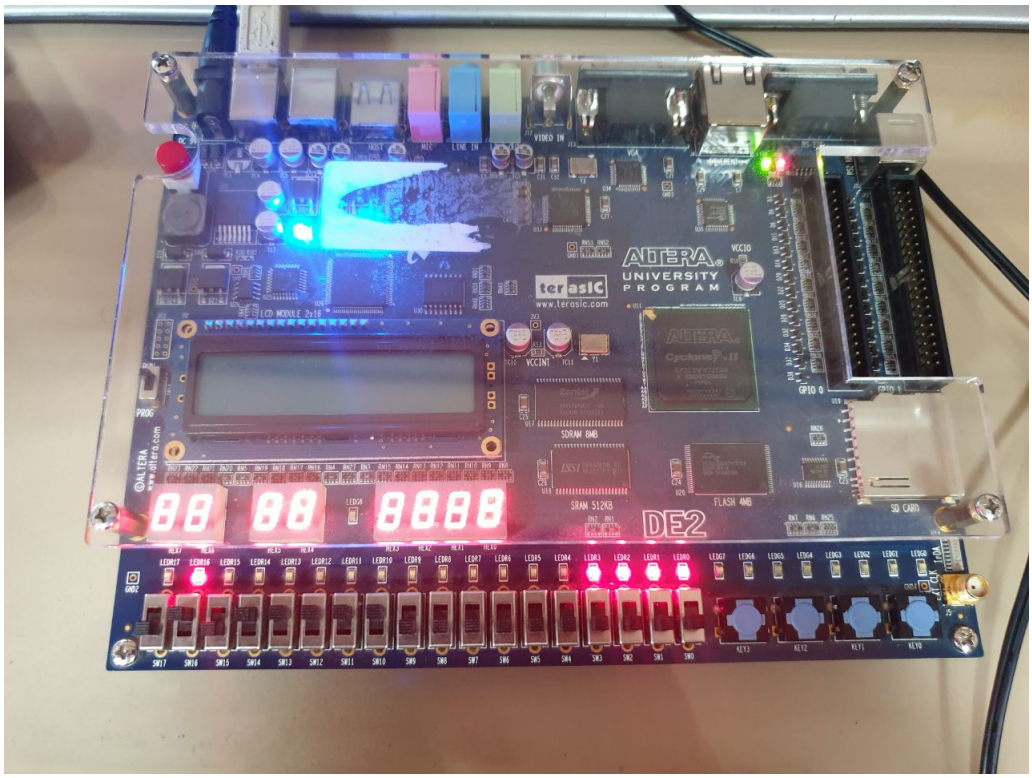
خروجی‌های full و empty نیز به ترتیب LED های ۱۷ و ۱۶ می‌باشد. Data Out حاصل از عملیات pop که به عنوان خروجی باید نمایش داده شود نیز به ترتیب از بیت پر ارزش LED3 الی LED0 می‌باشد.

۴.۲ تست FPGA

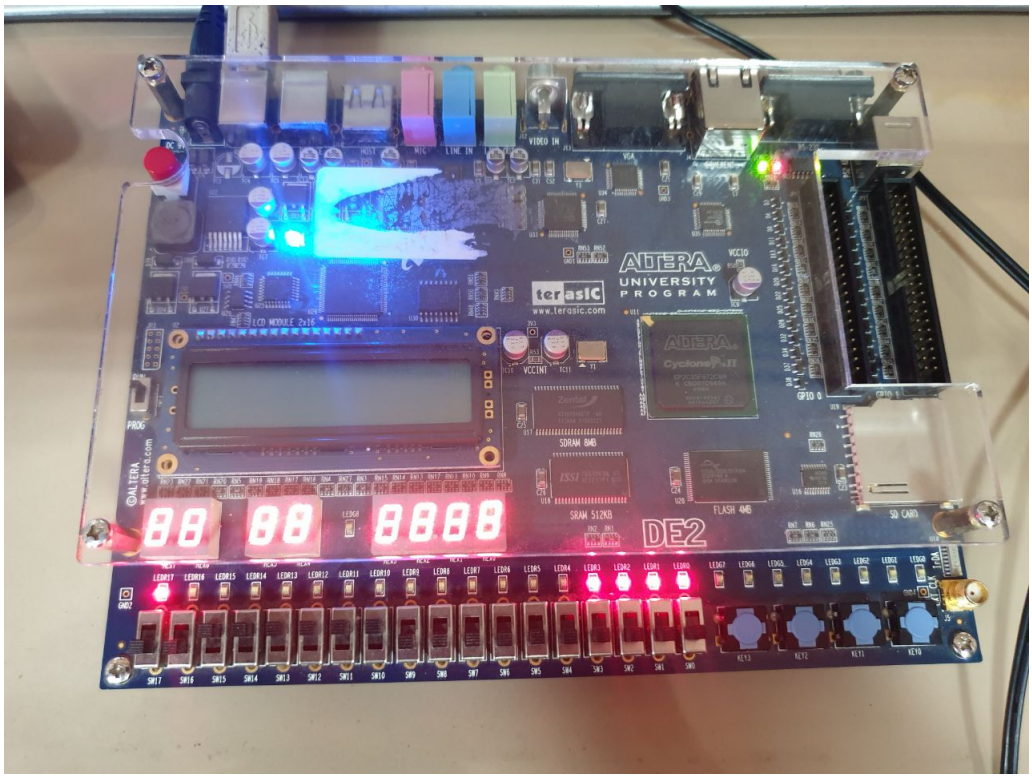
این تست لحظه‌ی ابتدایی start شدن FPGA را نشان می‌دهد که استک خالی است. همانظور که مشاهده می‌کنید چراغ empty روشن است.



در این ابتدا ۴ بیت 1111 را در استک push می‌کنیم و سپس آن را pop می‌کنیم. عکس زیر در واقع خروجی Data Out که از pop شدن آمده است را نشان می‌دهد. همچنین چراغ empty مجددا فعال می‌شود زیرا دیتایی که push کرده بودیم را pop کردیم و استک دوباره خالی شده است.



در این تست نیز دیتای ورودی 1111 را ۸ بار در استک push می‌کنیم. همانطورکه مشاهده می‌کنید چراغ full روشن شده است.



در این تست مقداری دیتا را push می‌کنیم. این تست برای این است که نشان دهیم زمانی که استک نیمه‌پر است هر دو چراغ empty و full خاموش می‌باشد.