گزارش کار چهارم آزمایشگاه طراحی سیستم های دیجیتال

توصیف رفتاری یک پشته با عمق هشت و پهنای چهار

استاد:

دکتر اجلالی

نویسنده:

امیررضا آذری - 99101087

بزرگمهر ضیا - 99100422

غزل طحان - 99106374

دانشگاه صنعتی شریف

تابستان 1402

فهرست

هدف آزمایش

می خواهیم یک پشته (با توصیف رفتاری) با عمق 8 و پهنای داده ی 4 بیت طراحی کنیم که دارای ورودی ها و خروجی های زیر باشد:

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

طراحی ماژول استک

از آنجایی که باید پشته را با توصیغ رفتاری طراحی کنیم، پس مجاز به استفاده از register ها هستیم. همان طور که در شکل 1 مشاهده میکنید سیگنال های ورودی و خروجی را طبق خواسته دستور کار آزمایش تعریف کرده ایم. شایان توجه است که سیگنال خروجی emty و سیگنال ورودی rstN، active low هستند؛ یعنی درصورتی که پشته خالی باشد emty صفر میشود و درصورتی که بخواهیم پشته را ریست کنیم باید ورودی rstN را صفر کنیم.

```
module Stack(
    output reg [3:0] data_out,
    output reg full,
    output reg empty, // active-low empty flag
    input [3:0] data_in,
    input push,
    input pop,
    input clk,
    input rstN // synchronized active-low reset signal
    );
```

شکل 1. ماژول استک

هم چنین برای طراحی پشته مان متغیر های نشان داده شده در شکل 2 را تعریف میکنیم. Sp همان متغیر استک پوینتر برای اشاره کردن به خانه های استک است؛ این متغیر به اولین خانه خالی استک اشاره کند بنابراین درصورتی که صفر باشد، استک خالی است و درصورتی که هشت باشد، استک پر میباشد (این متغیر چهاربیتی است چون عمق پشته 8 است). Stack_mem یک آرایه با هشت خانه ی چهاربیتی برای نمایش استک است. یک شمارنده با نام index نیز تعریف کرده ایم تا در ادامه در for loop از آن استفاده کنیم.

```
reg [3:0] sp = 4'b0000; // points to the first empty location on the stack
reg [3:0] stack_mem [7:0];

integer index = 0;
```

شکل 2. ماژول استک

حال تمامی سیگنال های ورودی این پشته را حساس به لبه بالا رونده کلاک و سنکرون طراحی میکنیم. در شکل 3 شروع طراحی منطق لبه بالارونده کلاک را مشاهده میکنیم.

```
always @(posedge clk) begin
```

شکل 3. ماژول استک

ابتدا کارکرد سیگنال ریست را در پشته طرحی میکنیم. همان طور که در شکل 4 مشاهده میکنیم با فعال شدن این سیگنال تمامی خانه های حافظه صفر میشوند. همچنین استک پوینتر نیز صفر شده و سیگنال empty نیز فعال میشود. هم چنین سیگنال full و data_out نیز صفر میشوند.

```
if (rstN == 0) begin
        for (index = 0; index < 8; index = index + 1) begin
                stack_mem [index] = 0;
        end
        sp = 0;
        empty = 0;
        full = 0;
        data_out = 0;
end
```

شکل 4. ماژول استک

حال اگر با آمدن لبه بالارونده کلاک، سیگنال ریست فعال نبود، به سراغ بررسی سیگنال های push و pull میرویم. همان طور که در شکل 5 میبینیم، اگر هر دو فرمان push و pull با هم صادر شده باشد، برنامه هیچ کاری نمیکند. اگر تنها push فعال بود و استک از داده پر نبود، داده ی ورودی در اولین خانه خالی استک ذخیره شده و استک پوینتر یکی اضافه میشود. اگر تنها pop فعال بود و استک خالی نبود، از استک پوینتر یکی کم شده (تا به آخرین خانه پر اشاره کند) و مقدار آخرین خانه ی استک نیز خروجی داده میشود (و حال، مقدار درون این خانه از

حافظه بی اعتبار میشود). حال در ادامه بررسی میکنیم که استک پر یا خالی شده است یا نه و در این صورت سیگنال های مربوطه را فعال میکنیم و طراحی این ماژول را تمام میکنیم.

```
        else begin
                if (push==1 && pop==1) begin
                // do nothing
                end
                else if (push==1 && full==0) begin
                        stack_mem [sp] = data_in;
                        sp = sp + 1;
                end
                else if (pop==1 && empty==1) begin
                        sp = sp - 1;
                        data_out = stack_mem[sp];
                end

                if (sp == 0) begin
                        empty = 0;
                        full = 0;
                end else if (sp == 8) begin
                        empty = 1;
                        full = 1;
                end else begin
                        empty = 1;
                        full = 0;
                end
        end
end
endmodule
```

شکل 5. ماژول استک

طراحی ماژول تست و ایجاد waveform

حال به سراغ طراحی ماژول تست میرویم. همان طور که در شکل 6 مشاهده میکنید ابتدا سیم ها و رجیستر های مورد نیاز را تعریف کرده و در ادامه از ماژول Stack اینستنس یا نمونه ای با نام stack میسازیم و سپس کلاک را طراحی کرده و دوره تناوب آن را 10ns قرار میدهیم.

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

    //clk
    initial begin
        clk = 1'b0;
    end
    always #5 clk = ~ clk;
```

شکل 6. ماژول تست

در ادامه و همان طور که در شکل های 7 و 8 مشاهده میکنیم، حالات مختلف سیگنال های Push و Pull را به ماژول میدهیم تا صحت خروجی این مدار را در حالات مختلف بتوانیم بررسی کنیم و طراحی این ماژول را نیز تمام میکنیم.

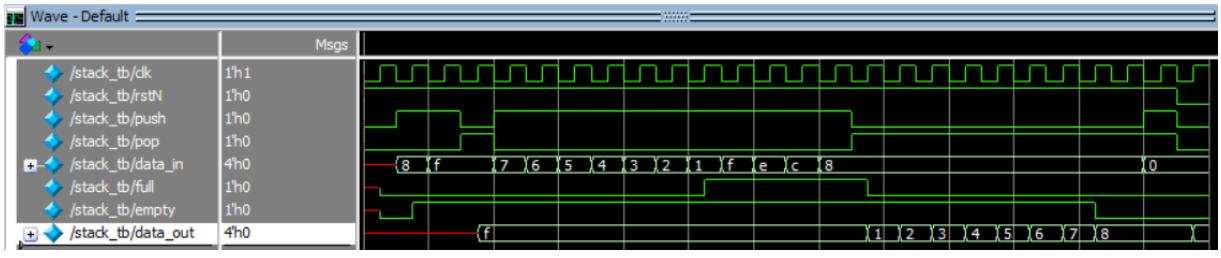
```
initial
    begin
            rstN <= 1'b1;
            push <= 1'b0;
            pop <= 1'b0;
            #10
            data_in <= 4'b1000;
            push <= 1'b1;
    //push 1000
            #10
            data_in <= 4'b1111;
            push <= 1'b1;
    //push 1111
            #10
            push <= 1'b0;
            pop <= 1'b1;
    //pop 1111
            #10
            pop <= 1'b0;
            push <= 1'b1;
            data_in <= 4'b0111;
    //push 0111
            #10
            push <= 1'b1;
            data_in <= 4'b0110;
    //push 0110
            #10
            push <= 1'b1;
            data_in <= 4'b0101;
    //push 0101
            #10
            push <= 1'b1;
            data_in <= 4'b0100;
    //push 0100
            #10
            push <= 1'b1;
            data_in <= 4'b0011;
    //push 0011
            #10
            push <= 1'b1;
            data_in <= 4'b0010;
    //push 0010
            #10
            push <= 1'b1;
            data_in <= 4'b0001;
    //push 0001
            #10
            push <= 1'b1;
            data_in <= 4'b1111;
    //push 1111
```

شکل 7. ماژول تست

```
            #10
            push <= 1'b1;
            data_in <= 4'b1110;
        //push 1110
            #10
            push <= 1'b1;
            data_in <= 4'b1100;
        //push 1100
            #10
            push <= 1'b1;
            data_in <= 4'b1000;
        //push 1000
            #10
            push <= 1'b0;
            pop <= 1'b1;
            #10
            pop <= 1'b1;
            #10
            pop <= 1'b1;
            #10
            pop <= 1'b1;
            #10
            pop <= 1'b1;
            #10
            pop <= 1'b1;
            #10
            pop <= 1'b1;
            #10
            pop <= 1'b1;
            #10
            pop <= 1'b1;
            #10
            data_in <= 4'b0000;
            push <= 1'b1;
            pop <= 1'b1;
        //nothing!
            #10
            push <= 1'b0;
            pop <= 1'b0;
            rstN <= 1'b0;
            #10
            $stop;
    end
endmodule
```

شکل 8. ماژول تست

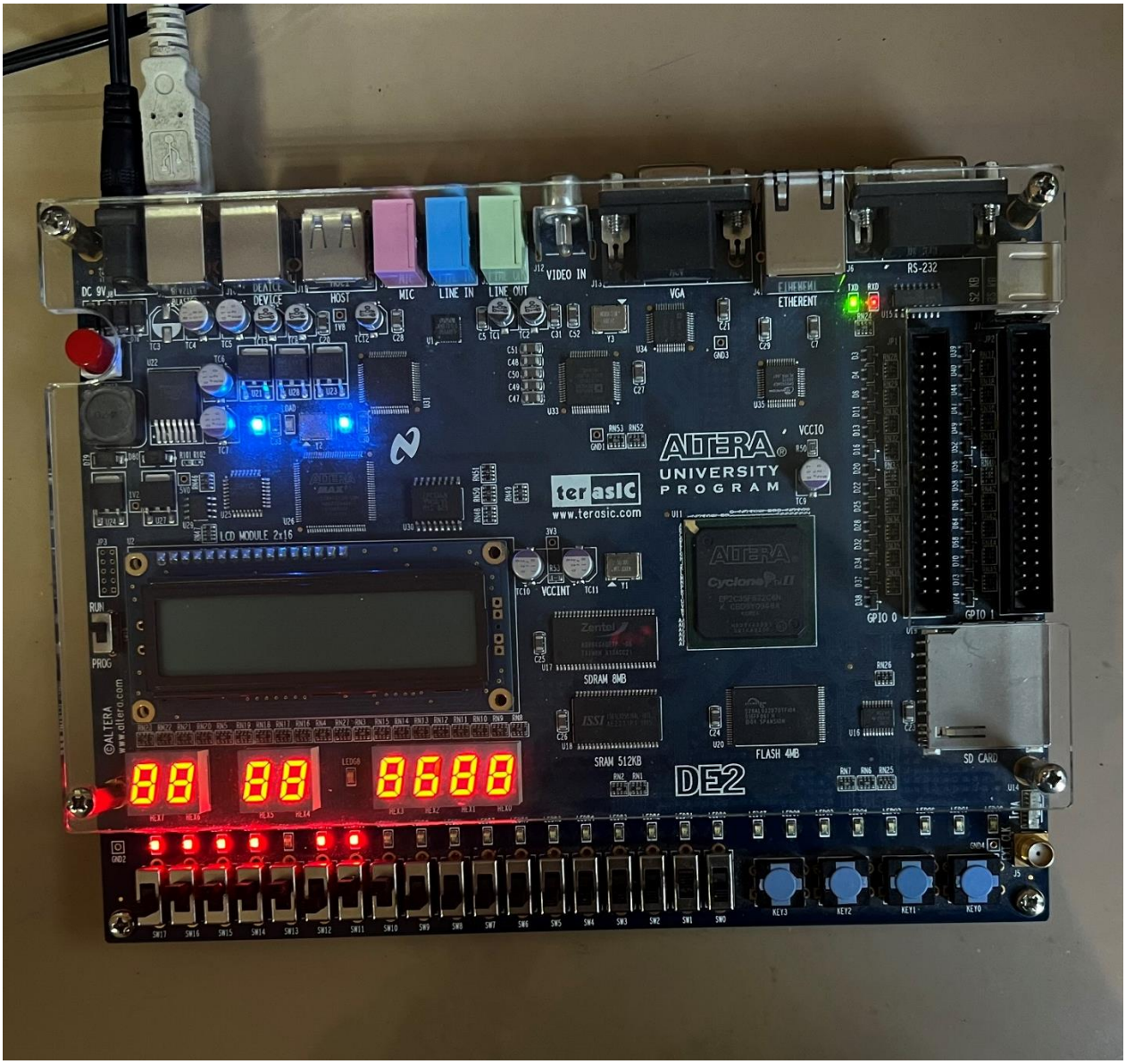
بعد از سنتز کردن این ماژول و بررسی خروجی های مدار، در شکل 9 میبینیم مدار خروجی های مورد انتظار را تولید کرده است.



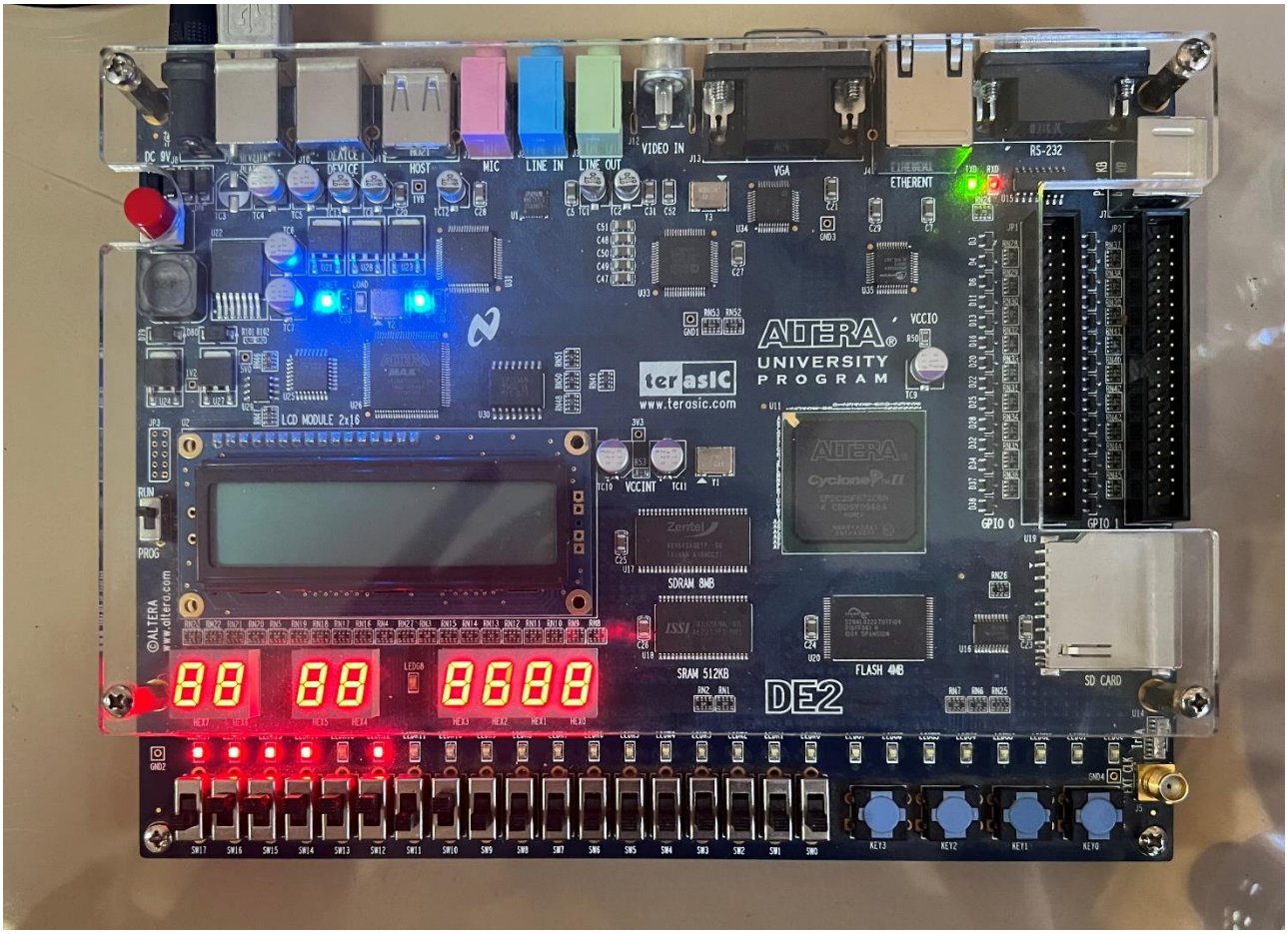
شکل 9. waveframe

کار در کلاس

در روز دوشنبه 23 ام مرداد ماه، در آزمایشگاه طراحی سیستم های دیجیتال، این مدار را روی بورد FPGA سنتز کردیم و با ورودی های مختلف، منطق مدار و خروجی های آن را بررسی کردیم که در تصاویر پایین قابل مشاهده هستند.



شکل 10. بورد FPGA (led های empty و full روشن شده اند به این معنی که استک پر شده است)



شکل 11. بورد FPGA (تنها led ی empty روشن شده به این معنی که استک نه خالی و نه پر است)

نتیجه گیری

در آزمایش چهارم، پشته ای با عمق 8 و پهنای داده ی 4 بیت را با توصیف رفتاری در زبان Verilog طراحی و تست کردیم؛ و درنهایت روی بورد FPGA پیاده سازی کرده و با اعمال تست کیس های متعدد، از صحت عملکرد آن مطمئن شدیم.

منابع و مراجع

- S. Palnitkar. Verilog HDL: A Guide to Digital Design and Synthesis. 2nd Edition, Prentice Hall, 2003.
- ACEX 1K Programmable Logic Family Data Sheet. Available at www.altera.com.
- ModelSim User's Manual. Available at www.actel.com.
- Introduction to the Quartus II Software. Available at www.altera.com