به نام خدا

درس : آزمایشگاه طراحی سیستم های دیجیتال

استاد : دکتر انصاری

آزمایش چهارم

گروه ۲:

سیدعماد امام جمعه ۴۰۰۱۰۸۷۷۴

آرش ضیایی رازبان ۴۰۰۱۰۵۱۰۹

محمدعرفان سلیما ۴۰۰۱۰۵۰۱۴

تیر ۱۴۰۲

مقدمه و هدف :

در این آزمایش قصد داریم که یک پشته با عمق ۸ و پهنای ۴ بیت با مشخصات زیر را به صورت رفتاری و با استفاده از زبان وریلاگ توصیف کنیم.

Inputs:

Clk	clock signal
RstN	Reset signal
Data_In	٤-bit data into the stack
Push	push command
Pop	pop command

Outputs:

Data_Out	٤-bit output data from stack
Full	Full = ١ indicates that the stack is full
Empty	Empty = ١ indicates that the stack is empty

پیاده سازی:

حال به توصیف پشته در وریلاگ می پردازیم:

ابتدا باید پر و خالی بودن پشته را مشخص کنیم. با توجه به اینکه از توصیف رفتاری استفاده می کنیم، بنابراین داریم:

```
reg [STACK_DEPTH-1:0] count;

assign Full = (count == STACK_DEPTH);
assign Empty = (count == 0);
```

در اینجا، متغیری به نام count تعریف می کنیم که با هر بار push به داخل پشته مقدار آن افزایش می یابد و با هر بار pop از پشته، مقدار آن کاهش می یابد. لذا، هنگامی که مقدار count با مقدار عمق پشته (STACK_DEPTH) برابر باشد، باید سیگنال Full یک شود که به معنای پر بودن پشته است و همینطور اگر مقدار count برابر با صفر باشد، پس پشته خالی است و سیگنال Empty باید ۱ شود.

اکنون به بررسی عملیات پشته می پردازیم:

۱. عملیات Push : در این عملیات باید به داخل پشته، عدد Data_In را اضافه کنیم و بعد از آن باید مقدار count را افزایش دهیم.
۲. عملیات Pop : در این عملیات باید عضو نهایی که اضافه شده است را از پشته برداریم و داخل Data_Out قرار دهیم. همچنین باید مقدار count را کاهش دهیم چرا که یک عدد از داخل پشته برداشته شده است.

در نتیجه همیشه می توانیم با مقدار count، ایندکسی از پشته که باید در آن push و یا از آن pop کنیم را داشته باشیم.

بنابراین، تنها حالتی باقی می ماند که بخواهیم پشته را reset کنیم. از آنجایی که سیگنال RstN به صورت active-low می باشد، پس زمانی باید reset را انجام دهیم که سیگنال RstN صفر باشد.

حال با توجه به توضیحات بالا، مدار را در وریلاگ به صورت رفتاری توصیف می کنیم:

```
module stack(Clk, RstN, Data_In, Push, Pop, Data_Out, Full, Empty);
    parameter STACK_DEPTH = 8;
    parameter STACK_WIDTH = 4;

    input wire Clk, RstN, Push, Pop;
    input wire[STACK_WIDTH-1:0] Data_In;

    output Full, Empty;
    output reg [STACK_WIDTH-1:0] Data_Out;

    reg [STACK_WIDTH-1:0] stack [STACK_DEPTH-1:0];
    reg [STACK_DEPTH-1:0] count;

    assign Full = (count == STACK_DEPTH);
    assign Empty = (count == 0);
    integer i;
    always @(posedge Clk, negedge RstN) begin
      if(~RstN) begin
        count = 0;
        for(i = 0; i < STACK_DEPTH; i = i + 1)begin
          stack[i] = 0;
        end
      end
      else if(Pop && !Push && !Empty) begin
        Data_Out = stack[count - 1];
        count = count - 1;
      end
      else if(Push && !Pop && !Full) begin
        stack[count] = Data_In;
        $display("stack[%d] = %d", count, stack[count]);
        count = count + 1;
      end
    end
  endmodule
```

شبیه سازی:

برای بررسی کارکرد پشته، ابتدا باید پشته را reset کنیم. در این عملیات تمام خانه های پشته با مقدار صفر، مقداردهی اولیه می کنیم. همچنین، مقدار count را نیز صفر قرار می دهیم. در این حالت با توجه به دستورات assign نیز سیگنال Full برابر با صفر و Empty برابر با یک می شود.

سپس با استفاده از یک حلقه ی for تمام هشت خانه ی پشته را پر می کنیم. برای اینکار از سیگنال Push استفاده می کنیم.

```
RstN = 0;
#5 RstN = 1;
for(i = 0; i < STACK_DEPTH; i = i + 1) begin
    Data_In = temp; Push = 1;
    #10 temp = temp + 1;
    $display("Full = %d, Empty = %d", Full, Empty);
end
```

درستی کارکرد پشته را نیز با اجرای تست می توانید مشاهده کنید:

```
Full = 0, Empty = 1
stack[  0] =  0
Full = 0, Empty = 0
stack[  1] =  1
Full = 0, Empty = 0
stack[  2] =  2
Full = 0, Empty = 0
stack[  3] =  3
Full = 0, Empty = 0
stack[  4] =  4
Full = 0, Empty = 0
stack[  5] =  5
Full = 0, Empty = 0
stack[  6] =  6
Full = 0, Empty = 0
stack[  7] =  7
Full = 1, Empty = 0
```

همانطور که مشخص است، در ابتدا پشته خالی بوده پس سیگنال Empty برابر با یک است و با هشت بار اجرا کردن دستور Push، پشته با اعداد مورد نظر پر شده است و در اخر نیز سیگنال Full برابر با یک شده است.

حال دستور Pop را تست می کنیم. به این منظور، سیگنال Push را برابر با صفر و سیگنال Pop را برابر با یک قرار می دهیم. بنابراین داریم:

```
Push =0; Pop = 1;
#10 $display("Data_Out = %d",Data_Out);
Push =0; Pop = 1;
#10 $display("Data_Out = %d",Data_Out);
Push =0; Pop = 1;
#10 $display("Data_Out = %d",Data_Out);
Push =0; Pop = 1;
#10 $display("Data_Out = %d",Data_Out);
```

با توجه به تصویر بالا دستور Push را چهار بار انجام داده ایم و خروجی را نیز در تصویر می توانید مشاهده کنید:

```
Data_Out =  7
Data_Out =  6
Data_Out =  5
Data_Out =  4
```

با توجه به اینکه آخرین عضو اضافه شده در پشته قبل از دستورات بالا، عدد هفت بود. درنتیجه، با اولین دستور Push عدد هفت خارج شده است که نشان می دهد این دستور نیز به درستی کار می کند. اعداد دیگر نیز به درستی خارج شده اند.

حال برای بررسی بیشتر نیز، دو بار دستور Push را با اعداد ۱۱ و ۱۲ انجام می دهیم. سپس دستور Pop را اجرا می کنیم.

```
Data_In = 11; Push = 1; Pop = 0;
#10
Data_In = 12; Push = 1; Pop = 0;
#10
Push =0; Pop = 1;
#10 $display("Data_Out = %d",Data_Out);
```

در اینجا خروجی باید عدد دوازده باشد، که تصویر نشان دهنده ی درستی کارکرد پشته است:

(همچنین، سیگنال های Full , Empty نیز در اینجا صفر هستند چرا که پشته نه خالی و نه پر است.)

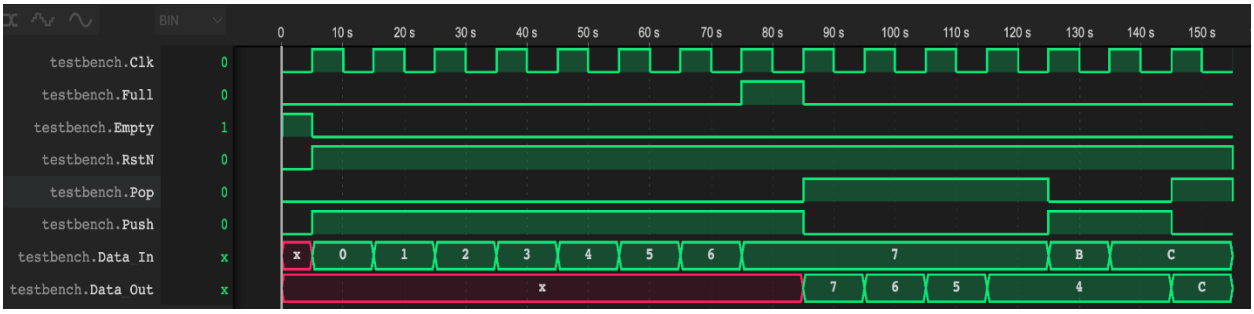
```
stack[  4] = 11
stack[  5] = 12
Data_Out = 12
Full = 0, Empty = 0
```

کد تست را نیز می توانید مشاهده کنید:

```
module testbench;
    parameter STACK_DEPTH = 8;
    parameter STACK_WIDTH = 4;
    reg Clk = 0, RstN = 0, Push = 0, Pop = 0;
    reg [STACK_WIDTH-1:0] Data_In;
    wire Full, Empty;
    wire [STACK_WIDTH-1:0] Data_Out;
    stack stackTest(Clk, RstN, Data_In, Push, Pop, Data_Out, Full, Empty);
    always #5 Clk = ~Clk;
    integer i;
    reg [STACK_WIDTH-1:0] temp = 8'b0;
    initial begin
        $dumpfile("wave.vcd");
        $dumpvars(0, testbench);
        RstN = 0;
        #5 RstN = 1;
        $display("Full = %d, Empty = %d", Full, Empty);
        for(i = 0; i < STACK_DEPTH; i = i + 1) begin
            Data_In = temp; Push = 1;
            #10 temp = temp + 1;
            $display("Full = %d, Empty = %d", Full, Empty);
        end
        Push =0; Pop = 1;
        #10 $display("Data_Out = %d",Data_Out);
        Push =0; Pop = 1;
        #10 $display("Data_Out = %d",Data_Out);
        Push =0; Pop = 1;
        #10 $display("Data_Out = %d",Data_Out);
        Push =0; Pop = 1;
        #10 $display("Data_Out = %d",Data_Out);

        Data_In = 11; Push = 1; Pop = 0;
        #10
        Data_In = 12; Push = 1; Pop = 0;
        #10
        Push =0; Pop = 1;
        #10 $display("Data_Out = %d",Data_Out);
        $display("Full = %d, Empty = %d", Full, Empty);
        $finish;
    end
endmodule
```

خروجی waveform را نیز در تصویر زیر قابل مشاهده است:



در نهایت خروجی Flow summery، RTL Viewer و Technology map viewer را مشاهده می کنید:

Flow Summary	
Flow Status	Successful - Sat Jul 15 13:30:50 2023
Quartus II 64-Bit Version	13.1.0 Build 162 10/23/2013 SJ Web Edition
Revision Name	stack
Top-level Entity Name	stack
Family	Cyclone IV GX
Total logic elements	74 / 14,400 (< 1 %)
Total combinational functions	60 / 14,400 (< 1 %)
Dedicated logic registers	44 / 14,400 (< 1 %)
Total registers	44
Total pins	14 / 81 (17 %)
Total virtual pins	0
Total memory bits	0 / 552,960 (0 %)
Embedded Multiplier 9-bit elements	0
Total GXB Receiver Channel PCS	0 / 2 (0 %)
Total GXB Receiver Channel PMA	0 / 2 (0 %)
Total GXB Transmitter Channel PCS	0 / 2 (0 %)
Total GXB Transmitter Channel PMA	0 / 2 (0 %)
Total PLLs	0 / 3 (0 %)
Device	EP4CGX15BF14C6
Timing Models	Final