بسمه تعالی

گزارش کار چهارم آزمایشگاه طراحی سیستم های دیجیتال

توصیف رفتاری

استاد: دکتر اجلالی

نویسندگان

مریم شیران ۴۰۰۱۰۹۴۴۶

مهدی بهرامیان ۴۰۱۱۷۱۵۹۳

مزدک تیموریان ۴۰۱۱۰۱۴۹۵

دانشگاه صنعتی شریف

تابستان ۱۴۰۳

فهرست مطالب

۱ آزمایش چهارم: توصیف رفتاری

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

۱-۱ مقدمه

در این آزمایش، می‌خواهیم با استفاده از زبان وریلاگ یک پشته (استک) با ۸ خانه حافظه طراحی کنیم. هر خانه حافظه در این پشته ۴ بیتی است.

۱-۲ طراحی ماژول

ماژول طراحی شده لازم است ورودی و خروجی های زیر را دارا باشد:

ورودی‌ها:

Clk (سیگنال ساعت): این سیگنال زمانی برای هماهنگی عملکرد عناصر مدار در مدارهای همگام استفاده می‌شود.

RstN (سیگنال ریست): این سیگنال برای بازنشانی پشته به حالت اولیه‌اش استفاده می‌شود. معمولاً به صورت فعال پایین است، به این معنی که ریست زمانی اتفاق می‌افتد که این سیگنال '۰' باشد.

Data_In (داده ورودی ۴ بیتی به پشته): این داده‌ای است که به پشته فشار داده می‌شود. عرض آن ۴ بیت است.

Push (دستور فشار دادن): این سیگنال فرمان نشان می‌دهد که داده باید به پشته فشار داده شود.

Pop (دستور خارج کردن): این سیگنال فرمان نشان می‌دهد که داده باید از پشته خارج شود. هر گاه مقدار آن یک بشود و پشته خالی (empty) نباشد، مقدار موجود در data_out به‌روزرسانی شده و مقدار موجود در خانه‌ای از حافظه استک که stack_pointer منهای یک به آن اشاره می‌کند ؛ روی data_out نوشته میشود.

خروجی‌ها:

Data_Out (داده خروجی ۴ بیتی از پشته): این داده‌ای است که از پشته خارج می‌شود. عرض آن ۴ بیت است.

Full (پشته پر): هنگامی که این سیگنال '۱' باشد، به این معنی است که پشته نمی‌تواند داده‌های بیشتری را قبول کند زیرا پر است.

Empty (پشته خالی): هنگامی که این سیگنال '۰' باشد، به این معنی است که پشته خالی است و داده‌ای برای خارج کردن وجود ندارد.

مواردی که تا کنون گفته شد ورودی ها و خروجی ها بودند. برای طراحی به ۲ متغیر دیگر نیز داریم.

اشاره‌گر پشته:

این متغیر یک رجیستر با عرض ۴ بیت است. علی رغم اینکه پشته در کل ۸ خانه دارد و با ۳ بیت هم می‌توان تمام خانه‌ها را آدرس‌دهی کرد ما نیاز به ۴ بیت داریم. چرا که این متغیر همیشه به اولین خانه خالی در حافظه پشته اشاره می‌کند، اگر پشته پر باشد، مقدار این متغیر باید ۸ باشد و در صورت استفاده از رجیستر ۳ بیتی، سرریز رخ داده و مقدار آن نامعتبر می‌شود.

حافظه پشته:

این متغیر یک آرایه از رجیسترهاست که عرض هر وکتور آن ۴ بیت و عمق آن ۸ درایه است. ذخیره‌سازی مقادیر ورودی به پشته در این آرایه انجام می‌شود.

۳-۱ کدنویسی ماژول اصلی در وریلاگ

با توجه به جدول زیر که رفتار مد نظر را توصیف میکند؛ کد مربوطه در وریلاگ به شرح زیر میشود.

Push	Pop	Stack State	Operation	Result
1	0	Not Full	Push	Item added to stack
1	0	Full	Push	No change
0	1	Not Empty	Pop	Item removed from stack
0	1	Empty	Pop	No change
1	1	Any	Push then Pop	No change
0	0	Any	No operation	No change

سر تا سر کد، به منظور خوانایی و توضیح کامنت گذاری شده است.

```
module stack_behavioural (
   input clk,              // Clock signal
   input rstN,             // Active-low reset signal
   input [3:0] data_in,    // Data input for push operation
   input push,             // Push operation signal
   input pop,              // Pop operation signal
   output reg [3:0] data_out, // Data output for pop operation
   output reg full,        // Flag indicating if the stack is full
   output reg empty        // Flag indicating if the stack is empty
);

// Define an 8-depth stack memory with 4-bit wide data
reg [3:0] stack_mem [7:0];
```

```
// Stack pointer to keep track of the stack position, points to the last free index in memory
reg [3:0] stack_pointer;

// Integer variable for loop indexing
integer i;

// Always block triggered on the positive edge of the clock
always @(posedge clk) begin
   if (!rstN) begin   // If reset signal is active (low)
      // Initialize the stack memory to zero
      for (i = 0; i < 8; i = i + 1) begin
         stack_mem[i] <= 0;
      end
      // Reset the stack pointer to zero
      stack_pointer <= 0;
      // Set the full flag to 0 (stack is not full)
      full <= 0;
      // Set the empty flag to 1 (stack is empty)
      empty <= 1;
   end else begin
      // If push signal is active, pop signal is not active, and stack is not full
      if (push && !pop && !full) begin
         // Push the data into the stack memory at the current stack pointer position
         stack_mem[stack_pointer] <= data_in;
         // Increment the stack pointer
```

```
            stack_pointer <= stack_pointer + 1;
            // Update the full flag: stack is full if stack pointer is 8
            full <= (stack_pointer == 8);
            // Update the empty flag: stack is empty if stack pointer is 0
            empty <= 0;
        end
        // If pop signal is active, push signal is not active, and stack is not empty
        else if (pop && !push && !empty) begin
            // Decrement the stack pointer
            stack_pointer <= stack_pointer - 1;
            // Pop the data from the stack memory at the new stack pointer
position
            data_out <= stack_mem[stack_pointer];
            // Update the full flag: stack is full if stack pointer is 8
            full <= 0;
            // Update the empty flag: stack is empty if stack pointer is 0
            empty <= (stack_pointer == 0);
        end
    end
end

endmodule
```

۴-۱ کدنویسی ماژول تست در وریلاگ

هم اکنون ماژول تست زیر را نوشته و در ادامه آن را اجرا خواهیم کرد تا از صحت کد خود مطمین شویم.

سر تا سر کد، به منظور خوانایی و توضیح کامنت گذاری شده است.

```
module testbench;

 // Declare testbench signals
 reg clk;              // Clock signal
 reg rstN;             // Reset signal (active low)
 reg [3:0] data_in;    // 4-bit data input
 reg push;             // Push control signal
 reg pop;              // Pop control signal

 // Output signals from the stack module
 wire [3:0] data_out;  // 4-bit data output
 wire full;            // Full flag
 wire empty;           // Empty flag

 // Clock period constant
 localparam CLK_PERIOD = 10;

 // Generate clock signal with a period of CLK_PERIOD
 always #(CLK_PERIOD/2) clk = ~clk;
```

```
// Instantiate the stack_behavioural module
stack_behavioural sb (
  .clk(clk),
  .rstN(rstN),
  .data_in(data_in),
  .push(push),
  .pop(pop),
  .data_out(data_out),
  .full(full),
  .empty(empty)
);
// Initial block for setup
initial begin
  clk = 0;          // Initialize clock signal
  push = 0;         // Initialize push signal
  pop = 0;          // Initialize pop signal
  rstN = 0;         // Initialize reset signal (active low)
  data_in = 4'b0000; // Initialize data input to 0
  $dumpfile("testbench.vcd"); // Specify VCD file for waveform dump
  $dumpvars(0, testbench);    // Dump all variables in the testbench module
end

// Initial block for test sequence
initial begin
  rstN = 0; // Assert reset signal to reset the stack
  #(CLK_PERIOD*2); // Wait for 2 clock cycles
```

```
rstN = 1; // Deassert reset signal to start normal operation

pop = 1; // Attempt to pop from empty stack

#(CLK_PERIOD*3); // Wait for 3 clock cycles
pop = 0; // Deassert pop signal
push = 1; // Assert push signal
data_in = 4'b1101; // Load 13 into data input

#(CLK_PERIOD*1); // Wait for 1 clock cycle
data_in = 4'b1111; // Load 15 into data input

#(CLK_PERIOD*1); // Wait for 1 clock cycle
data_in = 4'b0010; // Load 2 into data input

#(CLK_PERIOD*1); // Wait for 1 clock cycle
data_in = 4'b1001; // Load 9 into data input

#(CLK_PERIOD*1); // Wait for 1 clock cycle
push = 0; // Deassert push signal
pop = 1; // Assert pop signal

#(CLK_PERIOD*1); // Wait for 1 clock cycle
push = 1; // Assert push signal
pop = 0; // Deassert pop signal
data_in = 4'b0011; // Load 3 into data input
```

```
  #(CLK_PERIOD*1); // Wait for 1 clock cycle
  data_in = 4'b1001; // Load 9 into data input

  #(CLK_PERIOD*1); // Wait for 1 clock cycle
  push = 0; // Deassert push signal
  pop = 1; // Assert pop signal

  #(CLK_PERIOD*8.5); // Wait for 8.5 clock cycles
  $finish; // End simulation
 end

endmodule
```

تست بنچ ارائه شده برای ماژول پشته‌ی Verilog به نام stack_behavioural طراحی شده است تا رفتار پشته را تحت شرایط مختلف شبیه‌سازی کند. در اینجا توضیحی از آنچه تست بنچ انجام می‌دهد به طور دقیق آورده شده است:

ابتدای تست بنچ

تولید سیگنال کلاک:

یک سیگنال کلاک (clk) با دوره‌ی ۱۰ واحد تولید می‌شود که هر ۵ واحد (نیم دوره) تغییر می‌کند.

تعریف سیگنال‌ها:

سیگنال‌های مختلف (clk، rstN، data_in، push، pop، data_out، full، empty) تعریف می‌شوند.

ماژول پشته stack_behavioural با استفاده از سیگنال‌های تعریف‌شده نمونه‌سازی می‌شود.

ذخیره‌سازی داده‌های شبیه‌سازی:

داده‌های شبیه‌سازی در فایلی به نام "testbench.vcd" ذخیره می‌شوند تا بعداً با مشاهده‌گرهای موجی مشاهده شوند.

۵-۱ شبیه سازی

مرحله‌ی بازنشانی:

پشته با تنظیم rstN به ۰ بازنشانی می‌شود.

این حالت به مدت دو دوره کلاک نگه داشته می‌شود (#(۲*CLK_PERIOD)).

شروع شبیه‌سازی:

پس از دو دوره کلاک، rstN به ۱ تنظیم می‌شود تا بازنشانی لغو شده و عملیات عادی آغاز شود.

پاپ از پشته‌ی خالی:

سیگنال pop به مدت سه دوره کلاک (#(۳*CLK_PERIOD)) در حالی که پشته هنوز خالی است به ۱ تنظیم می‌شود. این حالت، واکنش پشته را به عملیات پاپ روی پشته‌ی خالی آزمایش می‌کند.

عملیات پوش:

سیگنال push به ۱ تنظیم می‌شود تا داده‌ها به پشته فشرده شوند.

مقادیر ۱۳ (۴'b۱۱۰۱)، ۱۵ (۴'b۱۱۱۱)، ۲ (۴'b۰۰۱۰)، و ۹ (۴'b۱۰۰۱) به ترتیب و هر کدام برای یک دوره کلاک به پشته فشرده می‌شوند.

عملیات پاپ:

سیگنال push به ۰ تنظیم شده (push = ۰) و سیگنال pop به ۱ تنظیم می‌شود (pop = ۱) تا مقدار بالای پشته پاپ شود. این حالت به مدت یک دوره کلاک اتفاق می‌افتد.

تناوب پوش و پاپ:

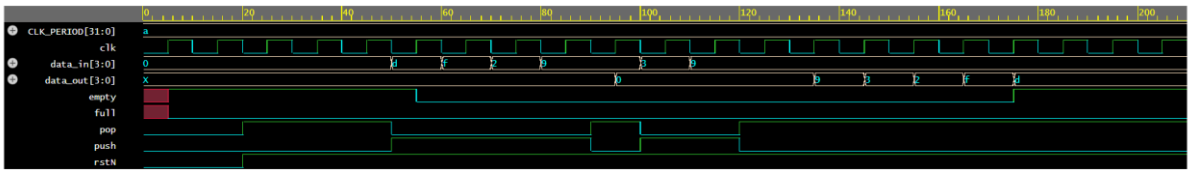
سیگنال push دوباره به ۱ تنظیم شده (push = ۱) و pop به ۰ تنظیم می‌شود (pop = ۰) تا مقدار ۳ (b۰۰۱۱'۴) به پشته فشرده شود.

مقدار دیگری ۹ (b۱۰۰۱'۴) به پشته فشرده می‌شود.

سیگنال push به ۰ تنظیم شده و سیگنال pop به ۱ تنظیم می‌شود تا مقدار بالای پشته دوباره پاپ شود.

پایان شبیه‌سازی:

شبیه‌سازی برای ۸.۵ دوره کلاک دیگر ادامه می‌یابد و سپس با فراخوانی $finish متوقف می‌شود.



در بالا نتیجه نمایش فایل vcd تولید شده از تست‌بنچ را میبینیم که با توضیحات مذکور مطابقت دارد.

۶-۱ منابع و مراجع

- وبسایت گیکزفورگیکز
- وبسایت یوتیوب
- وبسایت ویکی‌پدیا