تمرین DSD/ جبرانی پایان‌ترم

مجتبی فراتین/ ۴۰۱۱۰۶۳۰۶

۱- طراحی یک Stack Based ALU

کد اصلی:

```
module STACK_BASED_ALU #(parameter n) (input clk, input [n-1:0] in, input signed
[2:0] opcode, output reg [n-1:0] out, output reg overflow);
    reg [n-1:0] memory [511:0];
    reg [2*n-2:0] overflowCheck;
    integer stackPointer = 0;
    always @(posedge clk) begin
        if(opcode == -4) begin
            out <= memory[stackPointer - 1] + memory[stackPointer - 2];
            overflowCheck <= memory[stackPointer - 1] + memory[stackPointer - 2];
            if (overflowCheck != out) #2 overflow <= 1;
            else #2 overflow <= 0;
        end
        else if(opcode == -3) begin
            out <= memory[stackPointer - 1] * memory[stackPointer - 2];
            overflowCheck <= memory[stackPointer - 1] * memory[stackPointer - 2];
            if (overflowCheck != out) #2 overflow <= 1;
            else #2 overflow <= 0;
        end
        else if(opcode == -2) begin
            memory[stackPointer] <= in;
            #1 stackPointer = stackPointer + 1;
        end
        else if(opcode == -1 && stackPointer != 0) begin
            out <= memory[stackPointer - 1];
            #2 memory[stackPointer - 1] <= 0;
            #3 stackPointer = stackPointer - 1;
        end
    end
endmodule
```

در این کد ابتدا یک حافظه به عمق 512 و طول دلخواه (4 و 8 و 16 و 32) ساختیم. یک اینتجر برای اشاره‌گر به حافظه نیاز داریم. یک بیت هم به اورفلو اختصاص می‌دهیم. سپس شروع به طراحی ALU می‌کنیم. بر اساس آپکدها شرط می‌گذاریم و اگر طبق سوال نیاز به خروجی داشت در خروجی می‌ریزیم و اگر نه عملیات را انجام می‌دهیم و در جایی که اشاره‌گر نشان می‌دهد حاصل را می‌ریزیم. برای چک کردن بیت اورفلو در عملیات‌های جمع و ضرب حاصل نگه‌داری شده در حافظه را با حاصل واقعی جمع یا ضرب مقایسه می‌کنیم. اگر برابر بود اورفلو صفر است و رخ نداده و اگر برابر نبود یعنی اورفلو رخ داده و در بیت مربوطه یک می‌ریزیم.

کد تست 4 بیتی:

```
module test ();
    reg clk;
    parameter mn = 4;
    wire overflow;
    reg [3:0] in;
    reg signed [2:0] opcode;
    wire [3:0] out;
    STACK_BASED_ALU #(.n(mn)) STACK_BASED_ALU_Test(clk, in, opcode, out,
overflow);

    always begin
        #5 clk = ~clk;
    end

    initial begin
        clk = 0;
        opcode <= 3; in <= 5;
        #10 opcode <= 2; in <= 5;
        #10 opcode <= -2; in <= 3;
        #10 opcode <= -2; in <= 5;
        #10 opcode <= -4; in <= 3;
        #10 opcode <= -3; in <= 3;
        #10 opcode <= -1; in <= 3;
```

```
        #10 opcode <= -2; in <= 1;
        #10 opcode <= -2; in <= 2;
        #10 opcode <= -2; in <= 3;
        #10 opcode <= -2; in <= 4;
        #10 opcode <= -2; in <= 5;
        #10 opcode <= -4; in <= 3;
        #10 opcode <= -3; in <= 3;
        #10 opcode <= -1; in <= 3;
        #10 opcode <= -1; in <= 3;
        #10 opcode <= -1; in <= 3;
        #10 opcode <= -1; in <= 3;
        #10 opcode <= -1; in <= 3;
        #10 opcode <= -1; in <= 3;
        #10 opcode <= -1; in <= 3;
    end

    initial begin
        $monitor("output: %d, overflow: %b", out, overflow);
    end
endmodule
```

خروجی تست 4 بیتی:

```
# output:  x, overflow: x
# output:  8, overflow: x
# output:  8, overflow: 0
# output: 15, overflow: 0
# output:  5, overflow: 0
# output:  9, overflow: 0
# output:  9, overflow: 1
# output:  4, overflow: 1
# output:  4, overflow: 0
# output:  5, overflow: 0
# output:  4, overflow: 0
# output:  3, overflow: 0
# output:  2, overflow: 0
# output:  1, overflow: 0
# output:  3, overflow: 0
```

با توجه به قسمت مقدار دهی در تست، خروجی در مرحله اول چیزی ندارد. چون عملیاتی انجام نمی‌دهد با توجه به آپکد. سپس دو مقدار 3 و 5 را در استک قرار می‌دهد. سپس جمع انجام می‌دهد. بعد از آن ضرب انجام می‌دهد. در هیچ کدام از این دو عملیات اورفلو نداریم پس صفر است. بعد از آن یک مرحله از استک بر می‌داریم و مقادیر 1 تا 5 را در استک قرار می‌دهیم. دوباره عملیات ضرب و جمع را انجام می‌دهیم. از آنجایی که جمع دو خانه بالایی استک یعنی 4 و 5، 9 است اورفلو نداریم. اما بعد از آن که ضرب انجام می‌شود حاصل 20 است و در چهار بیت جا نمی‌شود. برای همین خروجی ضرب غلط است. سپس 7 مرحله برمی‌داریم. با اینکه استک 5 خانه پر داشت اما با توجه به شرطی که در کد اصلی نوشته شده این 7 مرحله بدون اینکه کد خراب شود انجام می‌شود و مشکلی ندارد.

کد تست 8 بیتی:

```
module test ();
    reg clk;
    parameter mn = 8;
    wire overflow;
    reg [7:0] in;
    reg signed [2:0] opcode;
    wire [7:0] out;
    STACK_BASED_ALU #(.n(mn)) STACK_BASED_ALU_Test(clk, in, opcode, out,
overflow);

    always begin
        #5 clk = ~clk;
    end

    initial begin
        clk = 0;
        opcode <= 3; in <= 5;
        #10 opcode <= 2; in <= 5;
        #10 opcode <= -2; in <= 3;
        #10 opcode <= -2; in <= 5;
        #10 opcode <= -4; in <= 3;
        #10 opcode <= -3; in <= 3;
        #10 opcode <= -1; in <= 3;
        #10 opcode <= -2; in <= 1;
        #10 opcode <= -2; in <= 2;
        #10 opcode <= -2; in <= 3;
        #10 opcode <= -2; in <= 4;
        #10 opcode <= -2; in <= 5;
        #10 opcode <= -4; in <= 3;
        #10 opcode <= -3; in <= 3;
```

```
        #10 opcode <= -1; in <= 3;
        #10 opcode <= -1; in <= 3;
        #10 opcode <= -1; in <= 3;
        #10 opcode <= -1; in <= 3;
        #10 opcode <= -1; in <= 3;
        #10 opcode <= -1; in <= 3;
        #10 opcode <= -1; in <= 3;
        #10 opcode <= -2; in <= 40;
        #10 opcode <= -2; in <= 30;
        #10 opcode <= -4; in <= 3;
        #10 opcode <= -3; in <= 3;
        #10 opcode <= -1; in <= 3;
    end

    initial begin
        $monitor("output: %d, overflow: %b", out, overflow);
    end
endmodule
```

خروجی تست 8 بیتی:

```
# output:   x, overflow: x
# output:   8, overflow: x
# output:   8, overflow: 0
# output:  15, overflow: 0
# output:   5, overflow: 0
# output:   9, overflow: 0
# output:   9, overflow: 1
# output:  20, overflow: 1
# output:  20, overflow: 0
# output:   5, overflow: 0
# output:   4, overflow: 0
# output:   3, overflow: 0
# output:   2, overflow: 0
# output:   1, overflow: 0
# output:   3, overflow: 0
# output:  70, overflow: 0
# output:  70, overflow: 1
# output: 176, overflow: 1
# output: 176, overflow: 0
# output:  30, overflow: 0
# output:  40, overflow: 0
```

برای این تست به تست قسمت قبل چند نمونه اضافه کردیم. بعد از خالی کردن استک عددهای 30 و 40 را در آن ریختیم. جمع و ضرب را انجام دادیم. جمع خروجی 70 می‌دهد که در 8 بیت جا می‌شود اما ضربشان 1200 می‌شود که نیاز به بیت‌های بیشتری دارد و برای همین جواب غلط می‌دهد و اورفلو را هم یک می‌کند. از طرفی ضرب 4 در 5 قسمت قبل را درست انجام داده و خروجی 20 داده است. در آخر هم دوباره بر می‌داریم و استک را خالی می‌کنیم.

کد تست 16 بیتی:

```
module test ();
    reg clk;
    parameter mn = 16;
    wire overflow;
    reg [15:0] in;
    reg signed [2:0] opcode;
    wire [15:0] out;
    STACK_BASED_ALU #(.n(mn)) STACK_BASED_ALU_Test(clk, in, opcode, out,
overflow);

    always begin
        #5 clk = ~clk;
    end

    initial begin
        clk = 0;
        opcode <= 3; in <= 5;
        #10 opcode <= 2; in <= 5;
        #10 opcode <= -2; in <= 3;
        #10 opcode <= -2; in <= 5;
        #10 opcode <= -4; in <= 3;
        #10 opcode <= -3; in <= 3;
        #10 opcode <= -1; in <= 3;
        #10 opcode <= -2; in <= 1;
        #10 opcode <= -2; in <= 2;
        #10 opcode <= -2; in <= 3;
        #10 opcode <= -2; in <= 4;
        #10 opcode <= -2; in <= 5;
```

```
        #10 opcode <= -4; in <= 3;
        #10 opcode <= -3; in <= 3;
        #10 opcode <= -1; in <= 3;
        #10 opcode <= -1; in <= 3;
        #10 opcode <= -1; in <= 3;
        #10 opcode <= -1; in <= 3;
        #10 opcode <= -1; in <= 3;
        #10 opcode <= -1; in <= 3;
        #10 opcode <= -1; in <= 3;
        #10 opcode <= -2; in <= 40;
        #10 opcode <= -2; in <= 30;
        #10 opcode <= -4; in <= 3;
        #10 opcode <= -3; in <= 3;
        #10 opcode <= -1; in <= 3;
        #10 opcode <= -2; in <= 300;
        #10 opcode <= -2; in <= 500;
        #10 opcode <= -4; in <= 3;
        #10 opcode <= -3; in <= 3;
        #10 opcode <= -1; in <= 3;
        #10 opcode <= -1; in <= 3;
    end

    initial begin
        $monitor("output: %d, overflow: %b", out, overflow);
    end
endmodule
```

خروجی تست 16 بیتی:

```
# output:     x, overflow: x
# output:     8, overflow: x
# output:     8, overflow: 0
# output:    15, overflow: 0
# output:     5, overflow: 0
# output:     9, overflow: 0
# output:     9, overflow: 1
# output:    20, overflow: 1
# output:    20, overflow: 0
# output:     5, overflow: 0
# output:     4, overflow: 0
# output:     3, overflow: 0
# output:     2, overflow: 0
# output:     1, overflow: 0
# output:     3, overflow: 0
# output:    70, overflow: 0
# output:    70, overflow: 1
# output:  1200, overflow: 1
# output:  1200, overflow: 0
# output:    30, overflow: 0
# output:   800, overflow: 0
# output:   800, overflow: 1
# output: 18928, overflow: 1
# output: 18928, overflow: 0
VSIM 11> run
# output:   500, overflow: 0
# output:   300, overflow: 0
```

در این تست هم به تست‌های قبلی اضافه می‌کنیم. خروجی ضرب 30 در 40 درست داده شده است و 1200 است. اما در اینجا عددهای 300 و 500 را در استک ریخته‌ایم و عملیات‌های ضرب و جمع را پیاده کرده‌ایم. جمع‌شان 800 و بدون اورفلو شده است که درست است. اما خروجی ضربشان چون در 16 بیت جا نمی‌شده غلط است. بعد از آن هم استک را خالی کرده‌ایم.

کد تست 32 بیتی:

```
module test ();
    reg clk;
    parameter mn = 32;
    wire overflow;
    reg [31:0] in;
    reg signed [2:0] opcode;
    wire [31:0] out;
    STACK_BASED_ALU #(.n(mn)) STACK_BASED_ALU_Test(clk, in, opcode, out,
overflow);
```

```
always begin
    #5 clk = ~clk;
end

initial begin
    clk = 0;
    opcode <= 3; in <= 5;
    #10 opcode <= 2; in <= 5;
    #10 opcode <= -2; in <= 3;
    #10 opcode <= -2; in <= 5;
    #10 opcode <= -4; in <= 3;
    #10 opcode <= -3; in <= 3;
    #10 opcode <= -1; in <= 3;
    #10 opcode <= -2; in <= 1;
    #10 opcode <= -2; in <= 2;
    #10 opcode <= -2; in <= 3;
    #10 opcode <= -2; in <= 4;
    #10 opcode <= -2; in <= 5;
    #10 opcode <= -4; in <= 3;
    #10 opcode <= -3; in <= 3;
    #10 opcode <= -1; in <= 3;
    #10 opcode <= -1; in <= 3;
    #10 opcode <= -1; in <= 3;
    #10 opcode <= -1; in <= 3;
    #10 opcode <= -1; in <= 3;
    #10 opcode <= -1; in <= 3;
    #10 opcode <= -1; in <= 3;
    #10 opcode <= -2; in <= 40;
    #10 opcode <= -2; in <= 30;
    #10 opcode <= -4; in <= 3;
    #10 opcode <= -3; in <= 3;
    #10 opcode <= -1; in <= 3;
    #10 opcode <= -2; in <= 300;
    #10 opcode <= -2; in <= 500;
    #10 opcode <= -4; in <= 3;
    #10 opcode <= -3; in <= 3;
    #10 opcode <= -1; in <= 3;
    #10 opcode <= -1; in <= 3;
    #10 opcode <= -2; in <= 200000;
    #10 opcode <= -2; in <= 500000;
    #10 opcode <= -4; in <= 3;
    #10 opcode <= -3; in <= 3;
    #10 opcode <= -1; in <= 3;
    #10 opcode <= -1; in <= 3;
```

```
    end

    initial begin
        $monitor("output: %d, overflow: %b", out, overflow);
    end
endmodule
```

خروجی تست 32 بیتی:

```
# output:          x, overflow: x
# output:          8, overflow: x
# output:          8, overflow: 0
# output:         15, overflow: 0
# output:          5, overflow: 0
# output:          9, overflow: 0
# output:          9, overflow: 1
# output:         20, overflow: 1
# output:         20, overflow: 0
# output:          5, overflow: 0
# output:          4, overflow: 0
# output:          3, overflow: 0
# output:          2, overflow: 0
# output:          1, overflow: 0
# output:          3, overflow: 0
# output:         70, overflow: 0
# output:         70, overflow: 1
# output:       1200, overflow: 1
# output:       1200, overflow: 0
# output:         30, overflow: 0
# output:        800, overflow: 0
# output:        800, overflow: 1
# output:     150000, overflow: 1
# output:     150000, overflow: 0
# output:        500, overflow: 0
# output:        300, overflow: 0
# output:     700000, overflow: 0
# output:     700000, overflow: 1
# output: 1215752192, overflow: 1
# output: 1215752192, overflow: 0
# output:     500000, overflow: 0
# output:     200000, overflow: 0
# output:         40, overflow: 0
```

در این تست خروجی ضرب 300 در 500 درست شده‌است و 150000 نمایش داده می‌شود. در ادامه عددهای 200000 و 500000 را در استک قرار داده‌ایم. سپس اول جمع و بعد ضربشان را خواستیم. در جمع عدد 700000 و بدون اورفلو نمایش داده شده که درست است. اما ضربشان باز هم اورفلو کرده و مقدار غلطی را خروجی داده است. سپس استک را خالی کرده‌ایم.