1403

پروژه جبرانی طراحی سیستمهای دیجیتال

سوال 1

علی احمدوند-401110318

بنده سوال اول میانترم را برای پروژه‌ی جبرانی انتخاب کرده‌ام.

الف)

کد

یک ماژول استک میسازیم که یک ارایه‌ی 2^10 تایی از رج های n بیتی می‌باشد. تنها ورودی‌های اضافی‌ای که برای ماژول در نظر گرفتیم rst و clk است که به ترتیب ریست و کلاک هستند. ماژول ما بصورت آسنکرون ریست می‌شود و عملیاتهای خود را (هر 4 عملیات مدنظر سوال) را به صورت سنکرون انجام می‌دهد. در واقع عملیاتهای اصلی‌ای که باید بصورت سنکرون انجام شوند پوش و پاپ هستند چرا که نباید بیش از یکبار پوش یا پاپ کنیم که تنها با کلاک، کنترل این موضوع امکانپذیر است.
یک رجیستر 11 بیتی برای نگه داشتن اولین جای خالی در استک تحت عنوان head داریم. یک رجیستر 2n بیتی داریم که حاصل عملیات جمع یا تفریق در این رجیستر ریخته می‌شود و n بیت output به n بیت اول این رجیستر اساین شده است.

```
reg signed [n - 1:0] mem [1023:0];

reg [10:0] head = 1'b0;

reg signed [2 * n - 1:0] temp;


assign output_data[n - 1:0] = temp[n - 1:0];
```

بلوک always درون استک در صورت تغییر کلاک به 1 یا تغییر ریست به 0 فعال می‌شود. ابتدا ریست را چک می‌کند و سپس یک بلوک case برای مقدار opcode می‌گذاریم تا ببینیم چه عملیاتی را باید انجام بدهیم.

```
if (rst === 1'b0) begin

        head = 0;

        overflow = 1'b0;

        temp = 0;

end
```

برای عملیات پوش ابتدا مطمئن می‌شویم جای خالی‌ای در استک وجود دارد یا نه، اگر نبود پیام مناسب میفرستیم و اگر هم بود پوش را انجام می‌دهیم، بدین صورت که در جایی که head به آن اشاره می‌کند دیتای ورودی را قرار می‌دهیم و سپس head را آپدیت می‌کنیم.

```
3'b110:

        begin

                        if (head < (2 ** 10)) begin

                            mem[head] = input_data;

                            head = head + 1;

                            overflow = 1'bx;

                            temp = {n{1'bx}};

                        end

                        else begin

                            $display("stack is full, can't push!");

                        end
```

```
    end
```

به طریق مشابه برای پاپ بررسی می‌کنیم که آیا دیتایی درون استک هست یا نه و به درستی عملیات را انجام داده یا ارور متناسب می‌دهیم.

```
3'b111:
                    begin
                        if (head != 0) begin
                            head = head - 1;
                            overflow = 1'bx;
                            temp = {n{1'bx}};
                        end
                        else begin
                            $display("stack is empty, can't pop!");
                        end
                    end
```

همانطور که گفتیم عملیات جمع و ضرب، حاصل خود را در رجیستر 2n بیتی temp قرار می‌دهند. حال بر حسب اینکه آیا این مقدار برای رجیستر n بیتی باعث overflow می‌شود یا نه بیت overflow را 1 می‌کنیم، وگرنه 0 می‌کنیم. overflow در صورتی رخ می‌دهد که عدد نهایی بزرگتر مساوی $2^{(n-1)}$ باشد یا از $-2^{(n-1)}$ کوچکتر باشد.
همچنین پیش از انجام عملیات چک می‌کنیم که آیا حداقل 2 دیتا در استک قرار دارد یا نه که اگر نباشد نمی‌توانیم عملیات را انجام دهیم و پیام مناسب می‌دهیم.
جمع:

```
3'b100:
                    begin
                        if (head >= 2) begin
                            temp = mem[head - 1] + mem[head - 2];
                            if ((temp >= (2 ** (n - 1))) || (temp < -(2 ** (n - 1)))) begin
                                overflow = 1'b1;
                            end else begin
                                overflow = 1'b0;
                            end
                        end
                        else begin
                            $display("We don't have 2 elements in stack.");
                        end
                    end
```

ضرب:

```
3'b101:
                    begin
                        if (head >= 2) begin
                            temp = mem[head - 1] * mem[head - 2];
                            if (temp >= (2 ** (n - 1)) || temp < -(2 ** (n - 1))) begin
                                overflow = 1'b1;
                            end else begin
                                overflow = 1'b0;
                            end
                        end
                        else begin
                            $display("We don't have 2 elements in stack.");
                        end
                    end
```

حالتی که نباید کاری انجام دهیم را هم در دیفالت می‌گذاریم:

```
default:
                    begin
                        overflow = 1'bx;
                        temp = {n{1'bx}};
                    end
```

استک بنده به این صورت است که در صورتیکه عملیات خواسته شده ضرب یا جمع نباشد مقدار بیت overflow و output خود را x می‌کند. اینکار را بدین جهت کردم که هنگام مانیتور کردن، مشخص باشد چه زمانی در حال پوش و پاپ هستیم و یا کاری انجام نمی‌دهیم و می‌شود به راحتی در صورتیکه هدف این بوده که به بیت overflow و output دست نزنیم، آن خطهایی که مربوط به x کردن این بیتها هستند را از کد حذف کنیم و در صورتیکه نیازی بهشان نبوده هم که هیچ.

تستبنچ

استک ما مقدار n را بصورت پارامتر می‌گیرد و در نتیجه ساخت استکهایی با سایز n متفاوت ممکن است. 4 استک با n های 4 و 8 و 16 و 32 میسازیم.
کلاک در هر 1 واحد زمانی تغییر حالیت می‌دهد و در ابتدا برابر با 0 است.
برای هر کدام از این استکها هم رج ورودی و سیمهای خروجی متناسب تعریف می‌کنیم که با ... ,_8 ,_4 مشخص شده اند.

```
myStack #(.n(4)) stack4(clk, rst, opcode_4, input_data_4, output_data_4, overflow_4);
myStack #(.n(8)) stack8(clk, rst, opcode_8, input_data_8, output_data_8, overflow_8);
myStack #(.n(16)) stack16(clk, rst, opcode_16, input_data_16, output_data_16, overflow_16);
myStack #(.n(32)) stack32(clk, rst, opcode_32, input_data_32, output_data_32, overflow_32);
```

تست آنها مشابه هم و پشت سر هم در یک بلوک initial نوشته شده است و از آنجایی که مشابه هم هستند در اینجا تنها یکیشان، یعنی 8 بیت را توضیح می‌دهیم.

در ابتدا 2 مقدار را در استک پوش می‌کنیم. سپس این 2 مقدار را در هم ضرب می‌کنیم و دو مقدار اولیه را جوری می‌دهیم که سرریز نکند. سپس مقدار دوم استک را پاپ می‌کنیم و مقدار جدیدی را درون آن پوش می‌کنیم و سپس ضرب می‌کنیم و مقدار جدید را جوری داده ایم که ضرب جدید سرریز کند. مقدار های داده شده به ترتیب 32 و 4- و 4 هستند که 128- سرریز نمی‌کند، ولی 128 سرریز می‌کند.

```
opcode_8 = 3'b110;
input_data_8 = 8'd32;
#2;
input_data_8 = -8'd4;
#2;
opcode_8 = 3'b101; // multiplying -4 with 32 to get -128 which will not overflow
#2
$display("time: %t, multiplying -4 by 32, not overflowing, result: %d, overflow: %b", $time, output_data_8,
overflow_8);
opcode_8 = 3'b111; // pop
#2
opcode_8 = 3'b110;
input_data_8 = 8'd4;
#2
opcode_8 = 3'b101; // multiplying 4 with 2 to get 128 which will overflow


#2;
$display("time: %t, multiplying 4 by 32, will overflow, result: %d, overflow: %b", $time, output_data_8,
overflow_8);
```

سپس کار مشابهی برای جمع می‌کنیم. یعنی دو عدد جدید در استک پوش می‌کنیم و جمعشان می‌کنیم و جوری این اعداد را می‌دهیم که سرریز نکند. سپس پاپ می‌کنیم و عددی جدید می‌دهیم و جمع جدید را حساب می‌کنیم و عدد جدید را جوری می‌دهیم که جمعمان سرریز نکند. اعدادی که می‌دهیم 63 و 64 و 65 هستند که می‌دانیم 67 سرریز نمی‌کند ولی 68 می‌کند.

```
opcode_8 = 3'b110;
input_data_8 = 8'd63;
#2
input_data_8 = 8'd64;
#2
opcode_8 = 3'b100; // adding 63 to 64 which will not overflow
#2
$display("time: %t,  adding 63 with 64, will not overflow, result: %d, overflow: %b", $time, output_data_8,
overflow_8);
opcode_8 = 3'b111;
#2
```

```
opcode_8 = 3'b110;
input_data_8 = 8'd65;
#2
opcode_8 = 3'b100; // adding 63 to 65 which will overflow
#2
$display("time: %t, adding 63 with 65, will overflow, result: %d, overflow: %b", $time, output_data_8,
overflow_8);
opcode_8 = 3'b000;
```

در کل اعدادی که به عنوان ورودی به استک می‌دهیم در بخش ضرب (2^(n-3 و 4- و 4 هستند و برای جمع هم 2^(n – 2) – 1 و 2 ^ (n – 2) و 2 ^ (n – 2) + 1 می‌باشد.
نتیجه‌ی تستبنچ بصورت زیر است:

```
# checking 4 bit:
# time:                   6, multiplying -4 by 2, not overflowing, result:  -8, overflow: 0
# time:                  12, multiplying 4 by 2, will overflow, result:  -8, overflow: 1
# time:                  18,  adding 3 with 4, will not overflow, result:   7, overflow: 0
# time:                  24, adding 4 with 4, will overflow, result:  -8, overflow: 1
# checking 8 bit:
# time:                  30, multiplying -4 by 32, not overflowing, result: -128, overflow: 0
# time:                  36, multiplying 4 by 32, will overflow, result: -128, overflow: 1
# time:                  42,  adding 63 with 64, will not overflow, result:  127, overflow: 0
# time:                  48, adding 63 with 65, will overflow, result: -128, overflow: 1
# checking 16 bit:
# time:                  54, multiplying -4 by 8192, not overflowing, result: -32768, overflow: 0
# time:                  60, multiplying 4 by 32, will overflow, result: -32768, overflow: 1
# time:                  66,  adding 16383 with 16384, will not overflow, result:  32767, overflow: 0
# time:                  72, adding 16383 with 16385, will overflow, result: -32768, overflow: 1
# checking 32 bit:
# time:                  78, multiplying -4 by 2^29, not overflowing, result: -2147483648, overflow: 0
# time:                  84, multiplying 4 by 2^29, will overflow, result: -2147483648, overflow: 1
# time:                  90,  adding 2^31-1 with 2^31, will not overflow, result:  2147483647, overflow: 0
# time:                  96, adding 2^31 - 1 with 2^31 + 1, will overflow, result: -2147483648, overflow: 1
```

دقت کنید آنجاهایی که حاصل 2^(n – 1)- است و سرریز داشته ایم، در واقع مقدار درست برابر با 2^(n – 1) بوده است اما بخاطر سیستم مکمل دو، این مقدار 2^(n – 1)- ذخیره و اعلام می‌شود و به درستی سرریز تشخیص داده شده است. در نتیجه کارهای مذکور من جمله پوش و پاپ بدرستی انجام شده اند وگرنه مقدارهای داده شده اشتباه می‌بودند. در نتیجه استک ما بدرستی کار می‌کند.