Homework 3: 2's Complement Multiplier

Patrick Laverty

Due March 3

1 Requirements

In this homework, we are required to construct a 16 x 8 multiplier that can handle 2's complement numbers. My implementation of this was to convert any signed number to unsigned and then perform multiplication, then convert back to signed. The only exception is when there are two signed numbers, then we return the unsigned version of the multiplication.

2 Finished Code

In this part, I ran the perl script given to create the multiplier. I then appended my code to take in temp values as inputs to test if they were signed or not. Because this is a little-endian language, we test the final bit to be 1.

mult.sv (snippets)

```
1 // Starting on line 10
2 assign Y = (Y_in[7] != 1) ? Y_in : ~Y_in + 00000001; // Conversion
3 assign X = (X_in[15] != 1) ? X_in : ~X_in + 000000000000001; // Conversion
4
5 // On line 306
6 assign Z_out = (((Y_in[7] != 1) && (X_in[15] != 1)) || ((Y_in[7] == X_in[15]))) ? Z :
  ~Z + 00000000000000000000001; // Conversion
```

I then put together my testbench to try with the entire range from -128 to 127. Each of these numbers out of the 256 range were then multiplied with some random number X that was flipped between positive and non-positive numbers on each clock edge.

mult_tb.sv

```
1  `timescale 1ps/1ps
2
3  module mult_tb ();;
4      logic clk = 1;
5
6      always clk = #5 ~clk;
7      logic signed [15:0] X;
8      logic signed [7:0] Y;
9      logic signed [23:0] Z;
10
11     initial begin
12         assign X = 1;
13         assign Y = -128;
14     end
15
16     mult mut (.X_in(X),
17               .Y_in(Y),
18               .Z_out(Z));
19
20     integer seed = 1000; // change seed to generate new random numbers
21
22     integer count = 0;
23     integer positive = 0;
24     always @(posedge clk) begin
25         #5 $display("␣␣X␣=␣%d", X);
26         #5 $display("*␣Y␣=␣%d", Y);
27         #5 $display("Value␣of␣Z␣=␣%d", Z);
28         if ((X * Y) == Z)
29             #5 $display("Passed!!!!");
30         else
31             #5 $display("FAILED!!!");
32
33         if (positive == 0) begin
34             assign X = $random(seed * count)%65536;
35             assign Y = Y + 1;
36             positive = 1;
37         end else begin
38             assign X = $urandom(seed * count)%65536;
39             assign Y = Y + 1;
40             positive = 0;
41         end
42         assign count = count + 1;
43     end
44 endmodule
```

Resulting Transcript (Because there are so many tests, I took a snippet)

```
# Passed!!!!
# Passed!!!!
#   X = -27770
# * Y =    14
# Value of Z =    -388780
# Passed!!!!
#   X =    2201
# * Y =    15
# Value of Z =     33015
# Passed!!!!
#   X =    -398
# * Y =    16
# Value of Z =     -6368
# Passed!!!!
#   X =   -9559
# * Y =    17
# Value of Z =    -162503
# Passed!!!!
#   X = -23356
# * Y =    18
# Value of Z =    -420408
# Passed!!!!
#   X = -21318
# * Y =    19
# Value of Z =    -405042
# Passed!!!!
#   X = -13502
# * Y =    20
# Value of Z =    -270040
# Passed!!!!
#   X =  32458
# * Y =    21
# Value of Z =     681618
# Passed!!!!
#   X =   -8444
# * Y =    22
# Value of Z =    -185768
# Passed!!!!
```

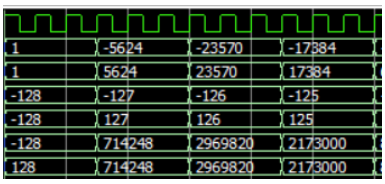

Figure 1: First half

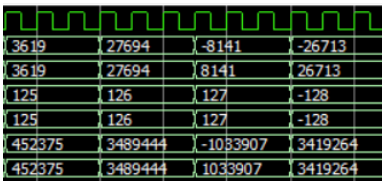

Figure 2: Second end

After doing so, I checked my simulation waveform to verify if it was correct as well:

Resulting Simulation

We can clearly see that the arithmetic is correct and assigning Z accordingly.

3 Conclusion

In this homework we understand how to deal with signed numbers whenever they appear. This tool can be used in many instances where if a signed number is involved we can then convert it to unsigned for proper use. Since the bit layout is little-endian, we must test the final bit to see if it is a signed number.