ECE 385

Spring 2023

Experiment #4

An 8-Bit Multiplier in SystemVerilog

Dylan Bautista, Matthew Wei

Lab Section NL/ Feb 27 11:00 AM

Nick Lu

Experiment #4

An 8-Bit Multiplier in SystemVerilog

I. Introduction/Purpose of Circuit:

The function of the Multiplier Circuit built in this experiment is to display the result of multiplying two, 8-bit 2's complement numbers. By using 8 switches on the FPGA board as input, upon entering one multiplicand, pressing load, entering the other multiplier, and clicking Run, the product should be visible on the LED display in hexadecimal format. The circuit should also allow for continuous multiplications of the current product.

II. Pre-Lab

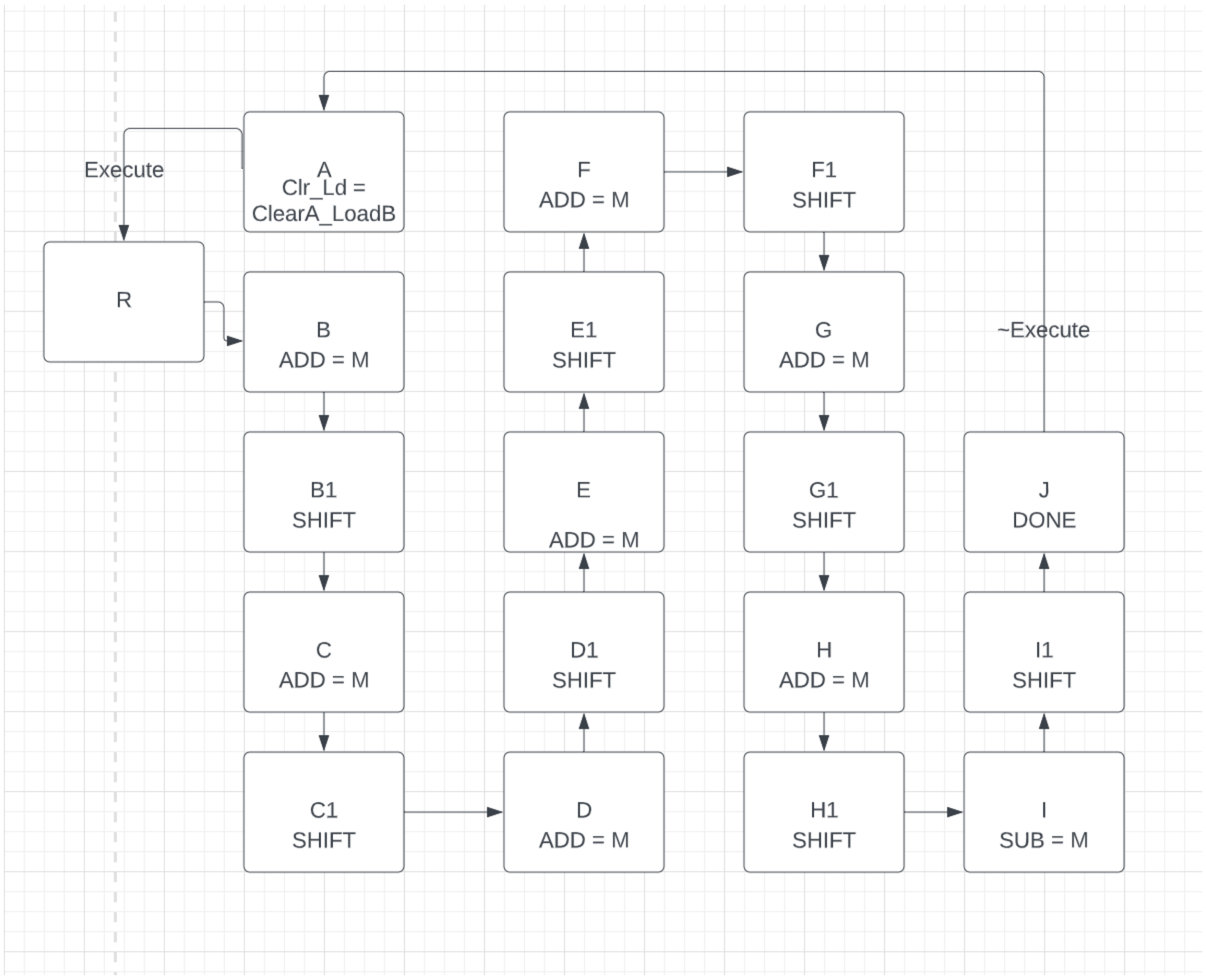
Initial Values: X = 0, A = 00000000, B =0000 0111 (achieved using ClearA_LoadB signal), S = 11000101, M is the least significant bit of the multiplier (Register B).

Function	X	A	B	M	Comments for Next Step
Clear A, Load B, Reset	0	0000 0000	*0000 0111*	1	Since M = 1, multiplicand (available from switches S) will be added to A
ADD	0	1100 0101	*0000 0111*	1	Shift XAB by one bit after ADD complete
SHIFT	0	0110 0010	1*000 0011*	1	Add S to A since M=1
ADD	0	0010 0111	1*000 0011*	1	Shift XAB by one bit after ADD complete
SHIFT	0	0001 0011	11*00 0001*	1	Add S to A since M=1
ADD	1	1001 1000	11*00 0001*	1	Shift XAB by one bit after ADD complete
SHIFT	1	1100 1100	011*0 0000*	0	Do not add S to A since M = 0. Shift XAB
SHIFT	1	1110 0110	0011 *0000*	0	Do not add S to A since M = 0. Shift XAB
SHIFT	1	1111 0011	0001 1*000*	0	Do not add S to A since M = 0. Shift XAB
SHIFT	1	1111 1001	1000 11*00*	0	Do not add S to A since M = 0. Shift XAB
SHIFT	1	1111 1100	1100 011*0*	0	Do not add S to A since M = 0. Shift XAB
SHIFT	1	1111 1100	0110 0011	1	8th shift done. Stop. 16-bit Product in AB.

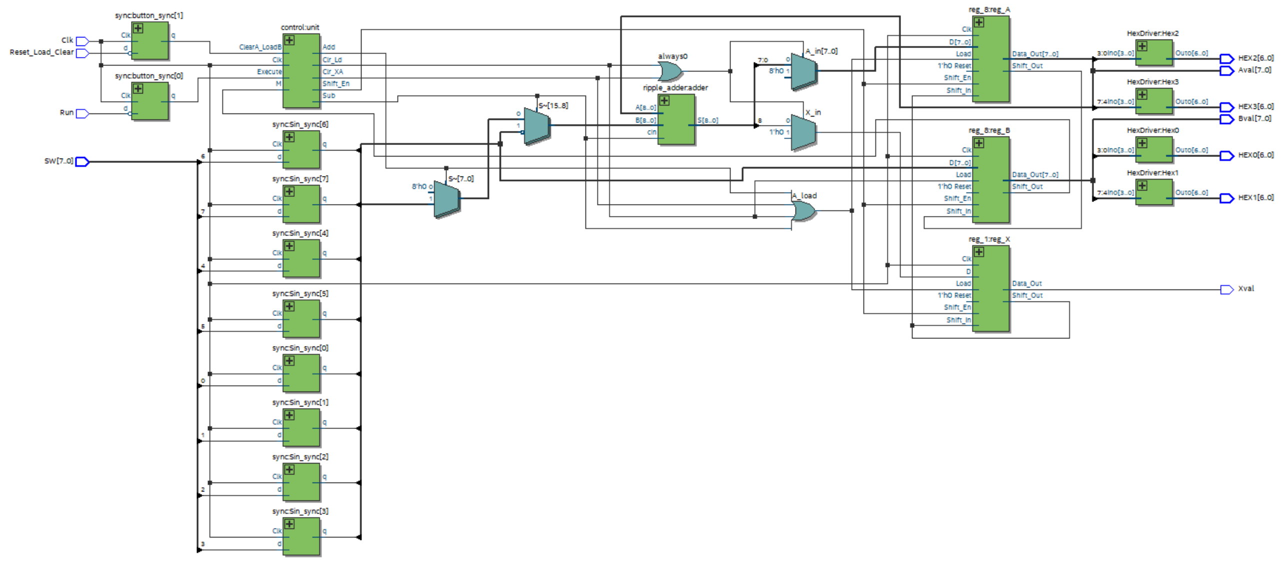
III. Written description of Circuit

The Multiplier Circuit is operated by loading in first the value of register B by using the on-board switches and the Reset_Load_Clear button. Then, the user will set the switches to represent the multiplicand and click the Run button to compute the product. The multiplying system utilizes an ADD/SHIFT method where A initially holds the multiplicand and B the multiplier, after the first step. The least significant bit of B is used to indicate (1) if sign extended S should be added to sign extended A with the result being stored in A and bit value X. Following every ADD step, there is an arithmetic right shift of all XAB bits. Otherwise (0), just a singular shift occurs. Once the 8th bit is reached, S is subtracted from A if the least significant bit is 1, or normal operation continues. To account for the SHIFT, ADD, SUB, and RESET steps for all 8 bits, corresponding states are created within the files, which set signals for external logical functionality. The 16-bit result is displayed on the FPGA hex displays. If the Run button is released then pressed again, another multiplication between the values represented in switches and register B will execute.

State Diagram for Control Unit



IV. Block Diagram:



Top Level Design RLT Diagram

V. Modules and Simulation:

Modules

full_adder.sv

Inputs: A, B, C_in

Outputs: S, C_out

Description: Basic full adder taking in two bits and a carrying and outputting their sum and carry-out bit.

Purpose: To be used in the ripple adder module to implement a 8-bit adder.

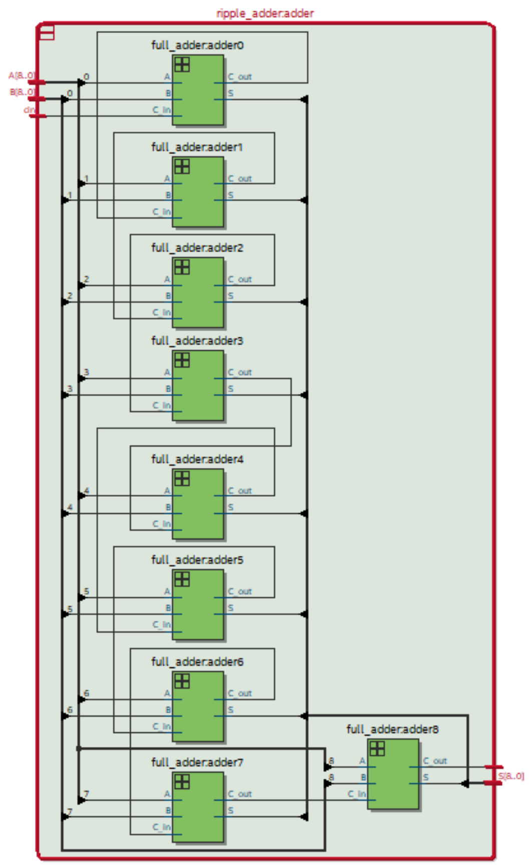
ripple_adder.sv

Inputs: [8:0] A, B, cin

Outputs: [8:0] S, cout

Description: Consists of 8 full adders with the carry-out of one feeding into the carry-in of one.

Purpose: Used as the adder to get the values of the multiplication product.



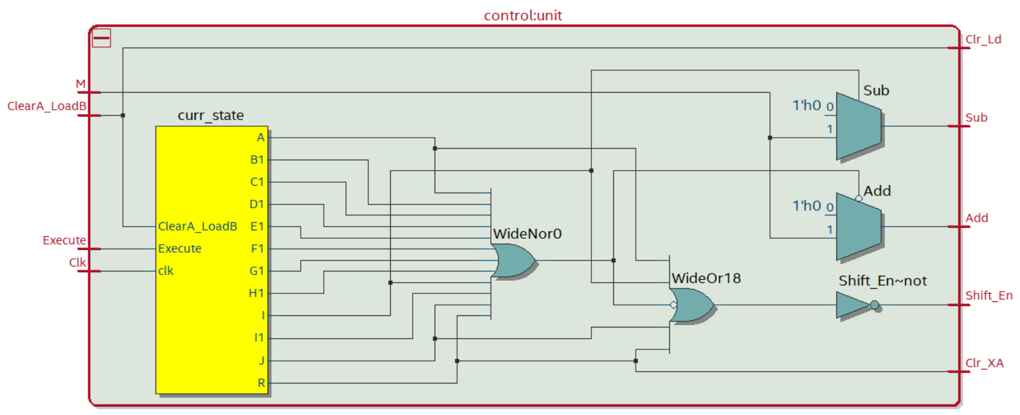
control.sv

Inputs: Clk, ClearA_LoadB, Execute, M

Outputs: Shift_En, Clr_Ld, Clr_XA, Add, Sub

Description: Takes in input signals and from their correctly outputs the corresponding output based on which state the design is currently in.

Purpose: Essentially holds our state machine. Directs our design onto what its next move should be. Loads shift registers, loads adder, shifts bits, etc.



HexDriver.sv

Inputs: [3:0] In0

Outputs: [6:0] Out0

Description: Module that holds information about the Hex LED displays on the FPGA board.

Purpose: Used so the correct output can be displayed onto the FPGAHEX displays.

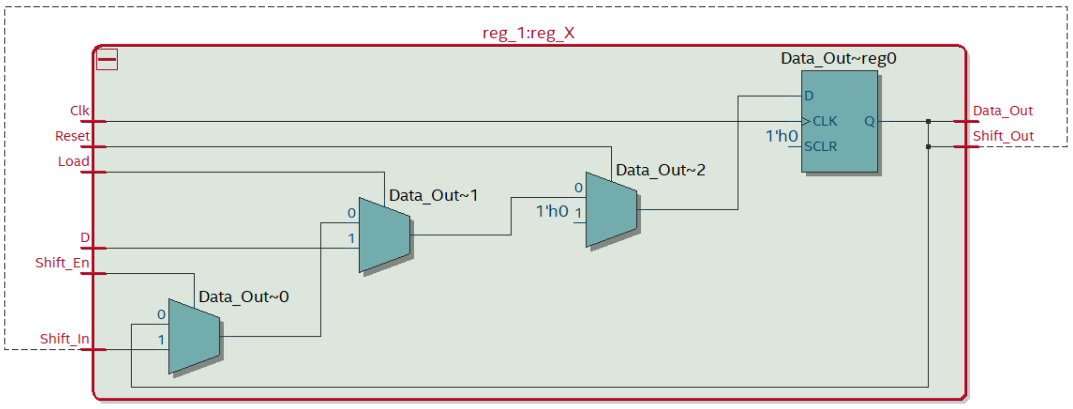
reg1.sv

Inputs: Clk, Reset, Shift_In, Load, Shift_En, D

Outputs: Shift_Out, Data_Out

Description: Basic one-bit shift register that either loads a bit, does nothing, or shifts a bit out.

Purpose: Used as the register that holds our value of X to determine the products sign.



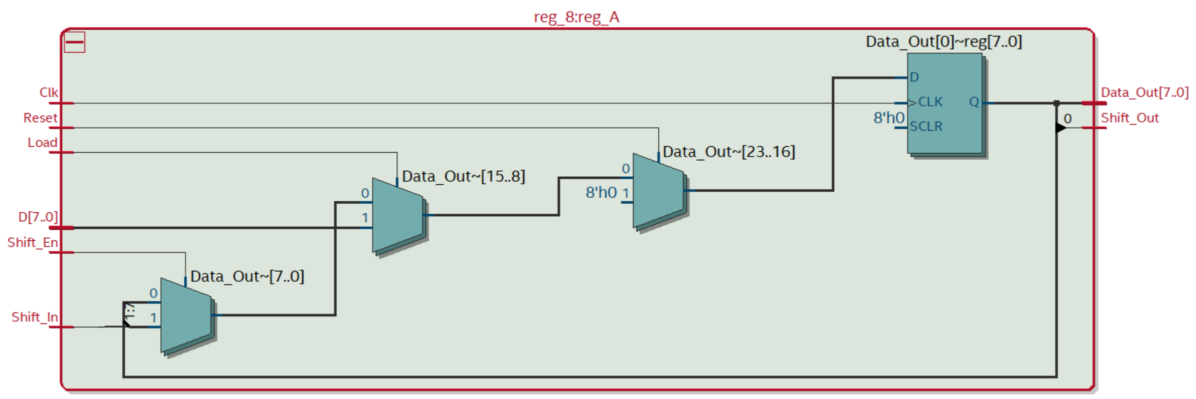
reg_8.sv

Inputs: Clk, Reset, Shift_In, Load, Shift_En, [7:0] D

Outputs: Shift_Out, [7:0] Data_Out

Description: Basic 8-bit shift register that either loads a value, does nothing, or shifts a bit out.

Purpose: Used as the registers that hold our values of A and B or our values that we want to multiply.



Multiplier.sv

Inputs: Clk, Reset_Load,CLar, Run, [7:0] SW

Outputs: [7:0] Aval, Bavl, Xval, [6:0] HEX0, HEX1, HEX2, HEX3

Description: Top-level design where all of our individual modules are combined into one. This is where the actual inputs and outputs of the design and individual modules are hooked up together.

Purpose: Connects our shift registers, adders, synchronizers, control unit, hexdrivers, and everything else with each other so that the design can properly implement a multiplicaition operation.

synchronizers.sv

Inputs: Clk, Reset, d

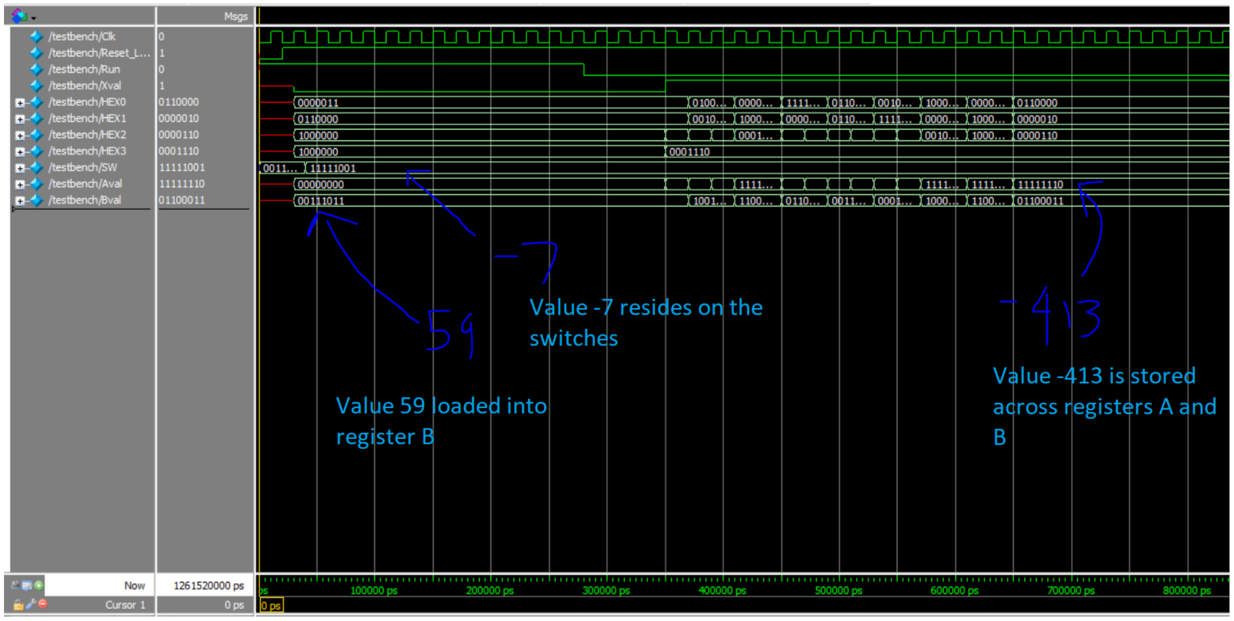
Outputs: q

Description: Contains modules that synchronize the code, so it properly works with the FPGA board.

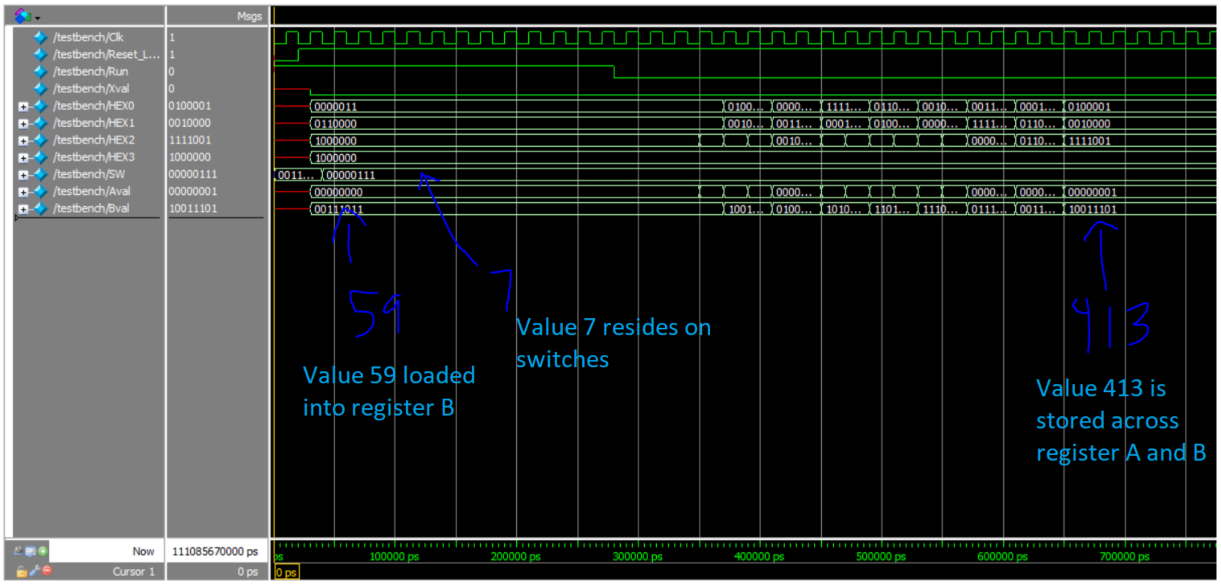
Purpose: Synchronizes the inputs on the board with the code so the FPGA can correctly be used to take in inputs for the code.

a. Must show 4 operations where operands have signs (+*+), (+*-), (-*+) and (-*-)

Simulation

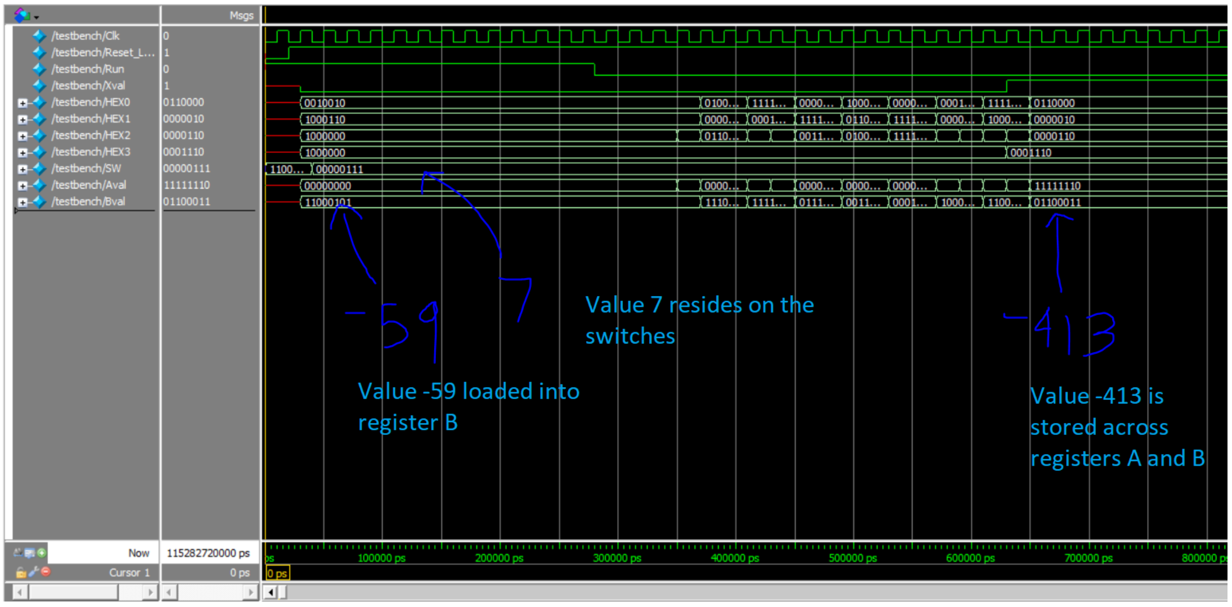


Simulation of -7*59 = -413


59
7
413
Value 7 resides on switches
Value 59 loaded into register B
Value 413 is stored across register A and B

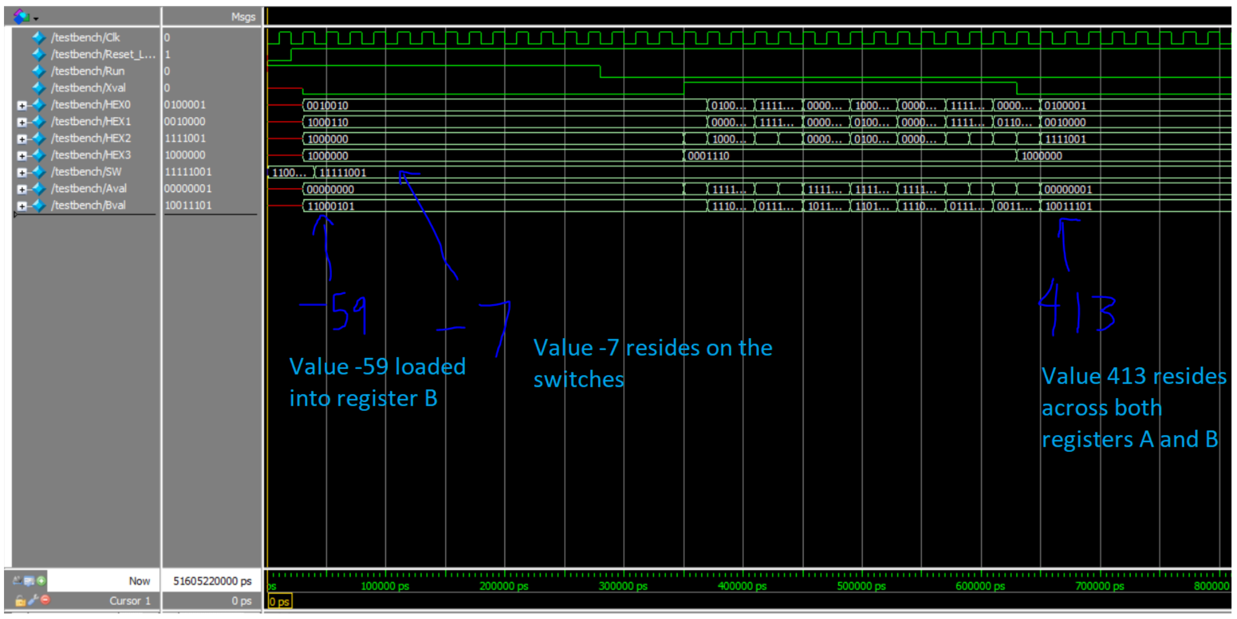
```
/testbench/Clk        1
/testbench/Reset_L...  1
/testbench/Run        0
/testbench/Xval       0
/testbench/HEX0       0100001
/testbench/HEX1       0010000
/testbench/HEX2       1111001
/testbench/HEX3       1000000
/testbench/SW         00000111
/testbench/Aval       00000001
/testbench/Bval       10011101
Now      111085670000 ps
Cursor 1             0 ps
```

Simulation of 7*59 = 413


-59
7
-413
Value 7 resides on the switches
Value -59 loaded into register B
Value -413 is stored across registers A and B

```
/testbench/Clk        0
/testbench/Reset_L...  1
/testbench/Run        0
/testbench/Xval       1
/testbench/HEX0       0110000
/testbench/HEX1       0000010
/testbench/HEX2       0000110
/testbench/HEX3       0001110
/testbench/SW         00000111
/testbench/Aval       11111110
/testbench/Bval       01100011
Now      115282720000 ps
Cursor 1             0 ps
```

Simulation of 7*-59 = -413



Simulation of –59*-7 = 413

VI. Conclusion

Overall, our design was correctly implemented and was able to work properly. It correctly calculates the correct values of each sign pair and operation. During our design process, we ran into a couple of bugs. One bug we ran into was continuous multiplication of –1 and –1 would result in the wrong values. This was due to us not clearing the A and X registers and after implementing that, it functioned correctly. Another bug we ran into was incorrect value loading into the registers. This was simply due to us incorrectly implementing the top-level and some simple fixes and additions caused that to work. Although not a bug, we initially thought there was an issue with some multiplication operations, but that was simply because the result caused overflow as it exceeded the range of a 16-bit 2’s complement bit value. Overall, this lab was a great experience of implementing more complex functions and designs using previously designed modules. The bugs we ran into were simply solved by referencing older modules or reading the lab documentation. It was clear on what we needed to do and how to correctly implement the design. The lectures also did a superb job of informing us on how the implemented multiplication algorithm operates and that made it a lot easier to code the design.

Post-Lab Questions

Refer to the Design Resources and Statistics in IQT and complete the following design statistics table. Come up with a few ideas on how you might optimize your design to decrease the total gate count and/or to increase maximum frequency by changing your code for the design.

	Multiplier
LUT	110
DSP	0
Memory (BRAM)	0
Flip-Flip	63
Frequency (MHz)	77.5
Static Power (mW)	89.97
Dynamic Power (mW)	1.63
Total Power (mW)	104.59

What is the purpose of the X register? When does the X register get set/cleared?

The X register is used to determine if the output (product) of the multiplication should be positive or negative. It gets cleared every time a new operation is performed because if it isn't cleared, then it won't perform the correct operation due to the bit remaining.

What would happen if you used the carry out of an 8-bit adder instead of output of a 9-bit adder for X?

It would result in some incorrect operations due to an incorrect bit value from the carry-out. It is not always the same sign as the most significant bit of register A.

What are the limitations of continuous multiplications? Under what circumstances will the implemented algorithm fail?

Continuous multiplication can lead to overflow if the product falls outside of our range. Since we are using 16 bit 2’s complement, we are limited by the max value of a 16-bit 2’s complement value. The implemented algorithm will fail if the product of two multiplications falls outside of the range.

What are the advantages (and disadvantages?) of the implemented multiplication algorithm over the pencil-and-paper method discussed in the introduction?

The implemented algorithm is better than the pencil-and-paper method because we are able to use less registers to hold our values. Had we used the pencil-and-paper method, we would have needed multiple registers to hold each of the values we needed to calculate the product. Using the implemented algorithm reduces that amount significantly. A disadvantage of the implemented method is that it takes longer to compute as the computation requires a bit shift after every calculation.