ECE 385

Spring 2020

Experiment #5

8 Bit Multiplier in System Verilog

Freddy Zhang and Mehul Dugar

Section: ABO (Thu. 3PM)

Gene Shiue

Introduction:

In this week's lab we were tasked to build an 8-bit multiplier in System Verilog that made use of the Add-Shift Algorithm. The multiplier takes in two numbers as parameters (operands), let one of them be C and the other B. B will be stored in a register. Now the multiplier multiplies the Least significant bit of B with the C and adds the result to another register A. If there is carryout from the sum of A and C, then the carryout bit will be stored in X. Then, register B will shift one bit right with the least significant bit of A be the shift in bit, and A will shift right one bit as well with X as the shift in bit. The multiplication and shifting will repeat a total of 8 times. However, on the eighth multiplier operation, if the least significant bit of B is 1, then the value of C will be subtracted from the value stored in register A because B would be a negative number. The two registers, A and B, will be the product of the two operands with A being the first 8 most significant bits and B being the 8 least significant bits.

We made use of 19 states to create a Moore machine to implement the control unit. These 19 states were broken down to shift, add and reset states. The final result was stored as a 16 bit 2's complement number.

Pre-lab Questions:

Rework the multiplication example on page 5.2 of the lab manual, as in compute 00000111 * 11000101 in a table similar to the example.

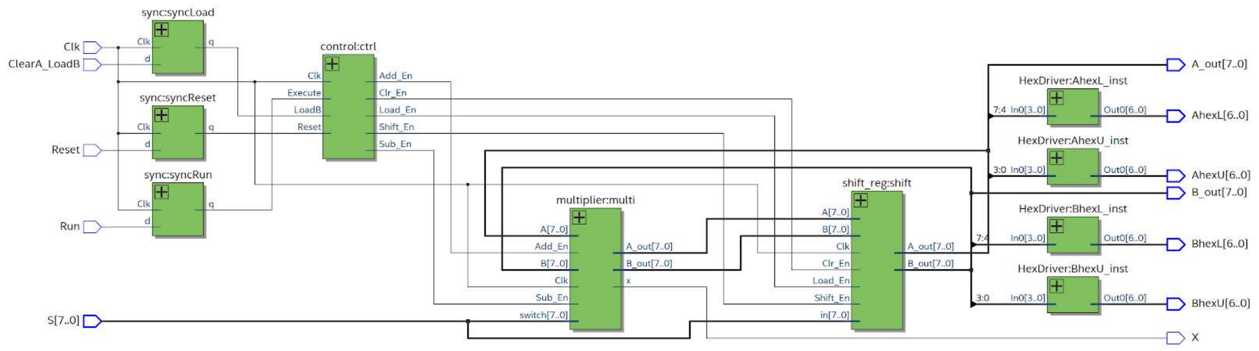
Functions	X	A	B`	M	Comments for next step
Clear A, Load B	0	00000000	00000111	1	Since M = 1, multiplicand (available from switches S) will be added to A.
ADD	1	11000101	00000111	1	Shift XAB by one bit after ADD complete
SHIFT	1	11100010	10000011	1	Second ADD into A, so add S to A
ADD	1	10100111	10000011	1	Shift XAB by one bit after ADD complete
SHIFT	1	11010011	11000001	1	Third ADD into A, so add S to A
ADD	1	10011000	11000001	1	Shift XAB by one bit after ADD complete
SHIFT	1	11001100	01100000	0	Since M = 0, shift without adding
SHIFT	1	11100110	00110000	0	Since M = 0, shift without adding
SHIFT	1	11110011	00011000	0	Since M = 0, shift without adding
SHIFT	1	11111001	10001100	0	Since M = 0, shift without adding

SHIFT	1	11111100	11000110	0	8th Shift so no ADD even if M = 1.
SHIFT	1	11111110	01100011	1	Final SHIFT, product is in AB

Summary of Operation:

One operand is stored into register B by using the LoadB_ClearA button. The value loaded into register B is determined by the value of the switches when the button was pushed. The second operand is the value of the switches when the run button is pushed. Once the run button is pressed, the two operands will go through the add-shift algorithm explained previously and store the product into both the A and B registers together. The answer is 16 bits long with the A register being the most significant 8 bits and the B register being the least significant 8 bits. If the user wants to compute the result again, pressing run will truncate the product to the least significant 8 bits in register B and multiply the value in register B with the values of the switches.

Top Level Block Diagram:



Written Description of .sv Modules:

Module : full_adder
Inputs : a, b, c
Outputs : s, cout
Description : This module creates a full adder

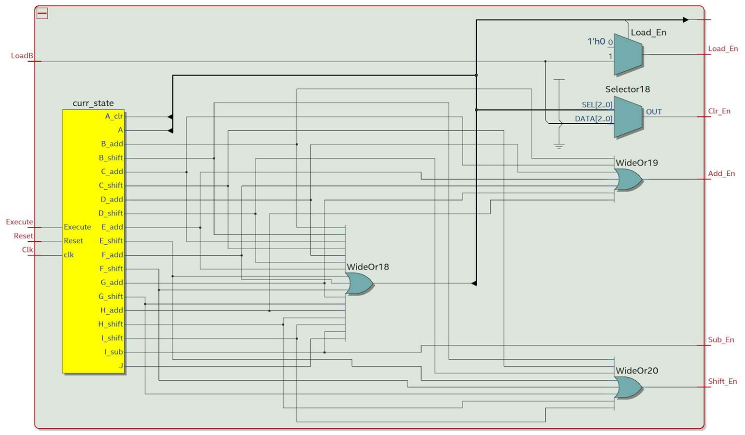
Module : adder
Inputs : [7:0] A, [7:0] B and cin
Outputs : sum
Description : This module creates a 9 bit ripple adder

Module : inverter
Inputs : [7:0] in
Outputs : [7:0] out

Description : This module inverts an 8 bit input

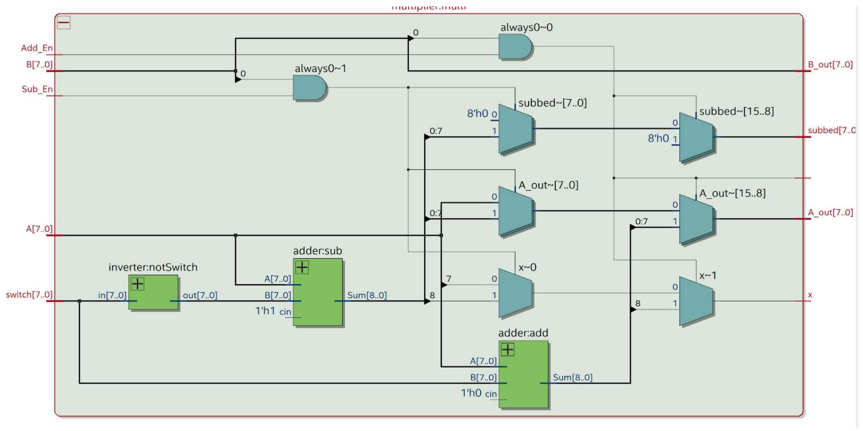
Module : HexDriver
Inputs : [3:0] in0
Outputs : [6:0] out0
Description : Drives the LED display on the FPGA board

Module : control
Inputs : Clk, Reset, Execute
Outputs : Shift_En, Add_En, Sub_En
Description : This module creates the Moore state machine



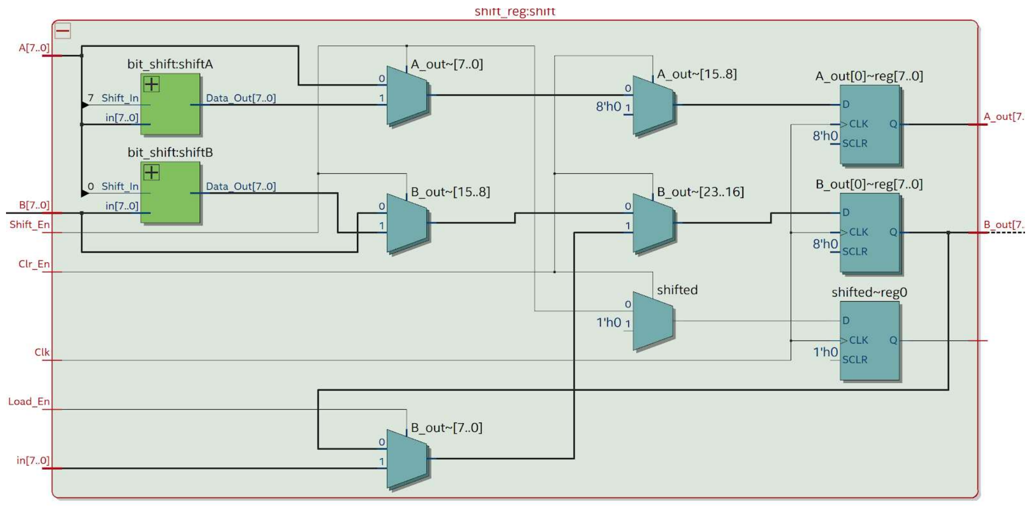
Module : lab5_toplevel
Inputs : Clk, Reset, ClearA_LoadB, Run, S
Outputs : X, AhexL, AhexU, BhexL, BhexU
Description : This module is used to instantiate all the modules

Module : multiplier
Inputs : [7:0] A, [7:0] B, [7:0] switch, Clk, Add_En, Sub_En, LdB
Outputs : [7:0] A_out, [7:0] B_out, X
Description : This module is used to perform the ADD step of the multiplication



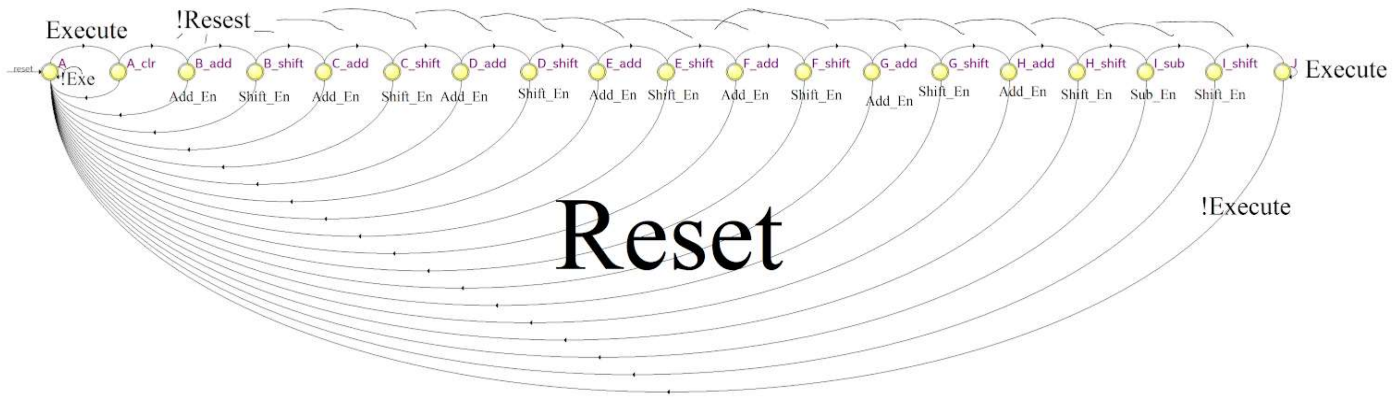
Module : bit_shift
Inputs : Clk, Reset, Shift_In, Shift_En
Outputs : Shift_Out, Data_Out
Description : This module is used to perform the shifting of bits on an 8 bit input

Module : shift_reg
Inputs : [7:0] A, [7:0] B, [7:0], Clk, Shift_En
Outputs : [7:0] A_out, [7:0] B_out
Description : This module is used to perform the SHIFT step of the multiplication



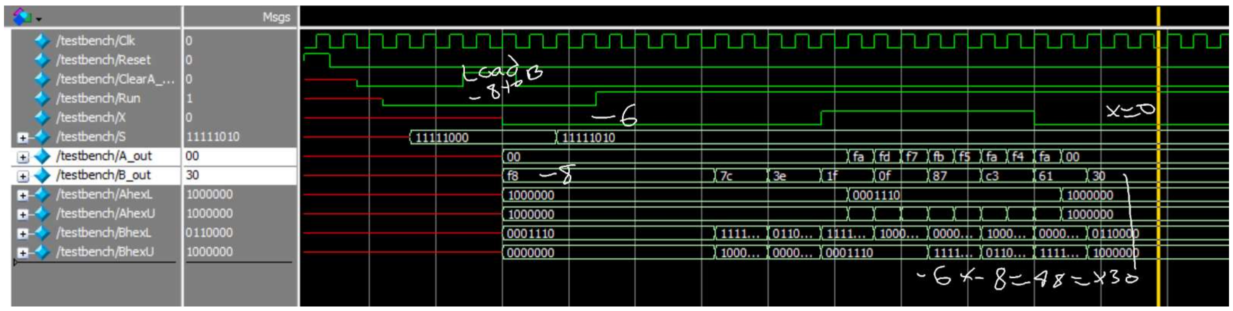
Module : testbench
Inputs :
Outputs :
Description : This module is used to test the operations given in the prelab (+*+, -*-, +*-, -*+)

State Diagram:

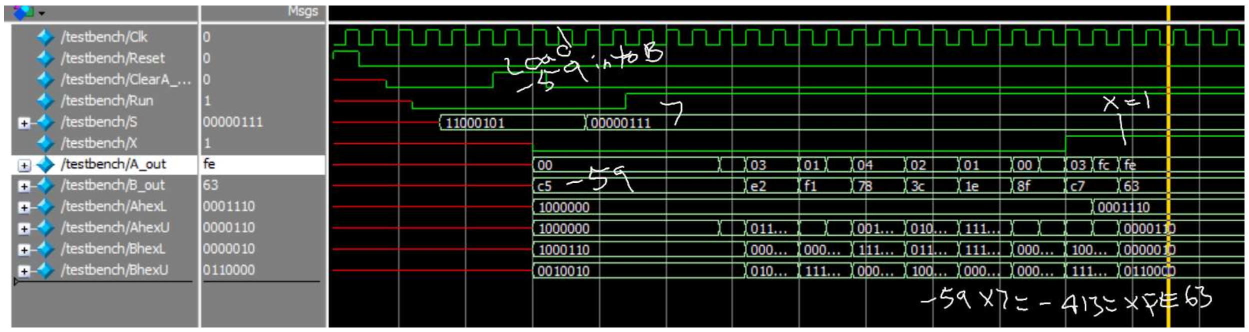


Annotated Pre-lab Waveforms:

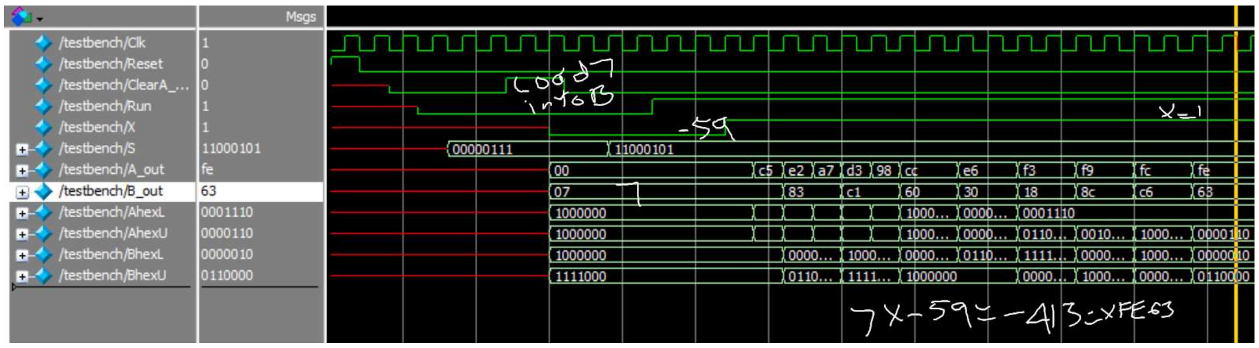
-*-



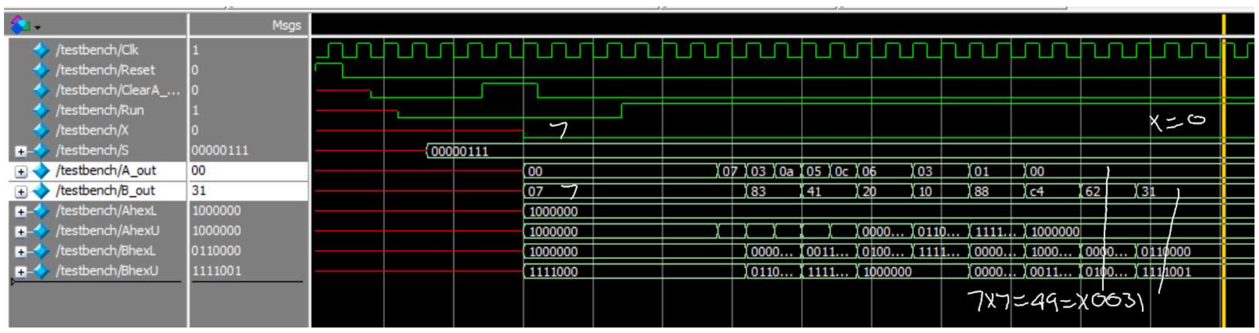
-*+



+*-



+*+



Post Lab Questions:

Refer to the Design Resources and Statistics in IQT.30-32 and complete the following design statistics table.

LUT	113
DSP	0
Memory (BRAM)	0
Flip-Flop	38
Frequency	253.74 MHz
Static Power	98.62 mW
Dynamic Power	0 mW
Total Power	156.93 mW

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

   X register is used for two purposes, it is firstly used to store the signs of the multiplicand. Secondly, it is used to determine whether we add the corresponding bits between the multiplier and the multiplicand. The X register gets set whenever the circuit adds the value in register A and the value from the switches in the 9-bit adder. The most significant bit of the 9-bit sum is stored in the X register. X only gets cleared whenever the LoadB_ClearA button is pressed or a new multiplication operation is started.

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

   One limitation of the continuous multiplication is that if the product is greater than 8 bits, the 8 bits will be truncated to the eight least significant bits because the circuit will clear the value in register A and only keep the value from register B and multiply off of that. This means the algorithm will fail if the product requires more than eight bits and another multiplication operation was executed.

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

It has advantages such as requiring lesser extend bits to be stored in comparison to the "pencil-and-paper" method as the latter requires you to store every extend bit. Additionally, the method implemented in the lab is faster than the traditional method owing to the fact that in the traditional method, each bit in the multiplicand needs to be multiplied by each bit in the multiplier.

One disadvantage however would be the fact that it is quite complex to implement in comparison to the traditional method (requires much less logic).

Conclusion:

Initially, one big problem in our implementation was forgetting to store the output of the shift register back into the shift register if the shift register was not being shifted. This caused our circuit to have unknown values in the shift register which resulted in unknown values for the output. Our implementation of the design worked as expected. One of the error's we faced was the fact that X bit was being incorrectly stored, so this was causing the answers to be slightly different from the expected value. We correctly implemented the 9-bit adder and inserting the most significant bit into X. However, the problem arose when we neglected to shift X into the most significant bit of register A. This caused the A register to shift incorrectly which resulted in the incorrect answer.

This lab seemed pretty well structured. Most of the confusion came from getting comfortable with the SystemVerilog and understanding how to use it. In hindsight, the actual lab was pretty simple. The hardest part was understanding how to connect everything together.