ECE 385
Lab 5
An 8-Bit Multiplier in SystemVerilog

Li, Ziying
Kang, Xingjian

March 22, 2024

1 Introduction (Summarize the basic functionality of the multiplier circuit

In this lab, we build a multiplier in System Verilog for two 8-bit 2's compliment numbers. The multiplier circuit uses the continuous add-shift way which is very similar to the algorithm of pencil-and-paper method of multiplication except the final step for 2's Complement numbers depends on the sign bit.

2 Pre-lab question

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

Function	X	A	B	M	Comments for the next step
Clear A, Load B	0	0000 0000	0000 0111	1	M=1, multiplicand will be added to A
ADD	1	1100 0101	0000 0111	1	Shift XAB by one bit after ADD complete
SHIFT	1	1110 0010	1000 0011	1	M=1, Add S to A
ADD	1	1010 0111	1000 0011	1	Shift XAB by one bit after ADD complete
SHIFT	1	1101 0011	1100 0001	1	M=1, Add S to A
ADD	1	1001 1000	1100 0001	1	Shift XAB by one bit after ADD complete
SHIFT	1	1100 1100	0110 0000	0	M=0, Shift XAB
SHIFT	1	1110 0110	0011 0000	0	M=0, Shift XAB
SHIFT	1	1111 0011	0001 1000	0	M=0, Shift XAB
SHIFT	1	1111 0011	0001 1000	0	M=0, Shift XAB
SHIFT	1	1111 1001	1000 1100	0	M=0, Shift XAB
SHIFT	1	1111 1100	1100 0110	0	M=0, Shift XAB
SHIFT	1	1111 1110	0110 0011	1	8th shift done. Stop

3 Written description and diagrams of multiplier circuit

3.1 Summary of operation (Explain in words how operands are loaded, how the multiplier computes its result, how the result is stored, etc.

First, input the multiplier B by S, click ClearA_LoadB, so that the number is stored into register B and register A is cleared to 00000000. Second, input the multiplicand by S. Third, click run, XAB will repeat ADD and SHIFT for 8 times to complete multiplication. Last, the result is stored in XAB.

3.2 Top Level Block Diagram (This can be generated from the RTL viewer. Please only include the top level diagram and not the RTL view of every module.

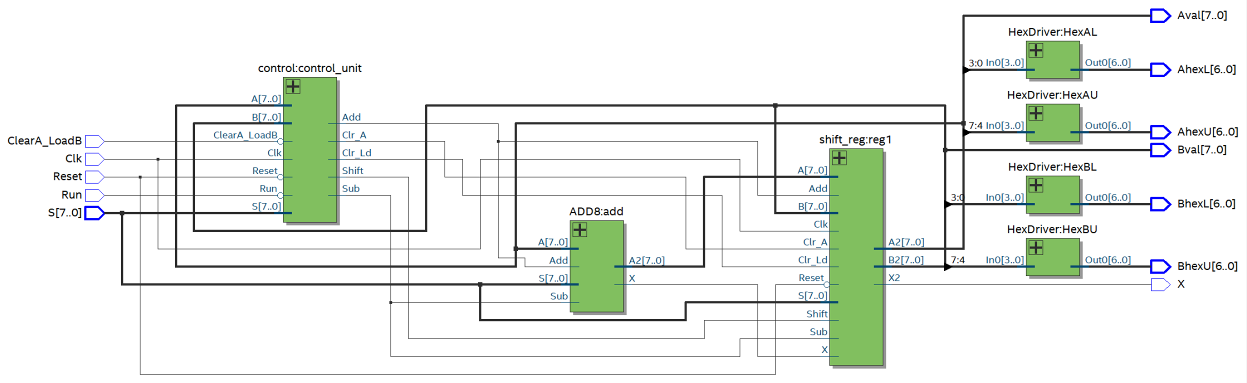

Figure 1: Enter Caption

3.3 Written Description of .sv Modules (List all modules used in a format shown in the appendix of this document

1) **Module:** *lab5_toplevel*

   Inputs: $[7:0]S, Clk, Reset, Run, ClearA_LoadB$

   Outputs: $[6:0]AhexU, [6:0]AhexL, [6:0]BhexU, [6:0]BhexL, [7:0]Aval, [7:0]Bval$

   Description: This is a positive-edge triggered top level bus to connect other sub modules.

   Purpose: Take top level inputs from switch and call all the sub modules

2) **Module:** *shift_reg*

   Inputs: X, Reset, Clk, [7:0]A, [7:0]B, [7:0]S, Clr_Ld, Shift, Add, Sub, Clr_A

   Outputs: x2, [7:0]A2, [7:0]B2

   Description: This is a positive-edge triggered 16-bit register with asynchronous reset and synchronous load. **Purpose:** This module is used to create the registers that store operands A and B in the adder circuit.

3) **Module name:** *control*

   Inputs: Clk, Reset, Run, ClearA_LoadB, [7:0]A, [7:0]B, [7:0]S

   Outputs: Clr_Ld, Shift, Add, Sub, Clr_A

   Description: This is a positive-edge triggered control unit with synchronous control signal.

   Purpose: control unit that takes exterior button signal and output inner control signal to different module.

4) **Module:** *ADD8*

   Inputs: [7:0]A, [7:0]S, Add, Sub

Outputs: [7:0]A2, X

Description: This is a positive-edge triggered 8-bit + 8-bit add.

Purpose: Use 9 full_adder to calculate A2 and X bit by bit.

5) **Module:** *full_adder*

   Inputs: A, B, Cin

   Outputs: $S, Cout$

   Description: This is a positive-edge triggered adder.

   Purpose: Takes 1-bit A, B and 1 carry-in bit Cin as inputs and provides a carry-out bit $Cout$ and the result S.

3.4 State Diagram for Control Unit (This can be done in a program like Visio, but if the Quartus state diagram generator is used, you must label the states and transitions or you will lose points!

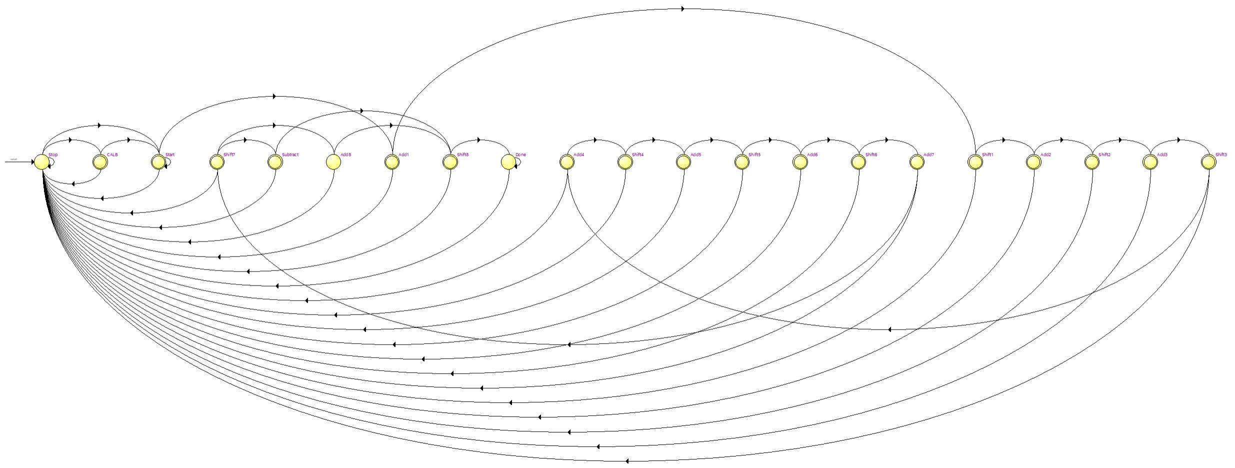

Figure 2: Enter Caption

4 Annotated pre-lab simulation waveforms

4.1 Create your own testbench for your multiplier which displays the following (4 operations where operands have signs (+*+), (+*-), (-*+) and (-*-) Waveform must have notes that clearly show the operands as well as the result, etc.

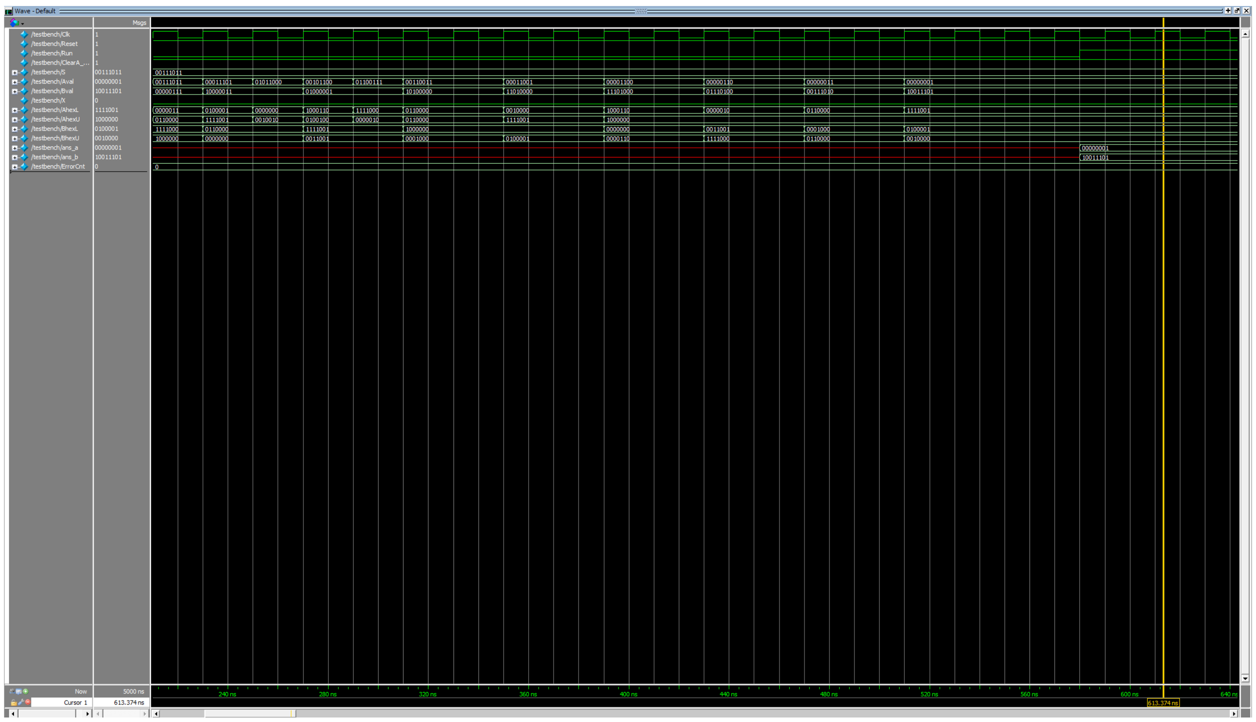

Figure 3: 7*59

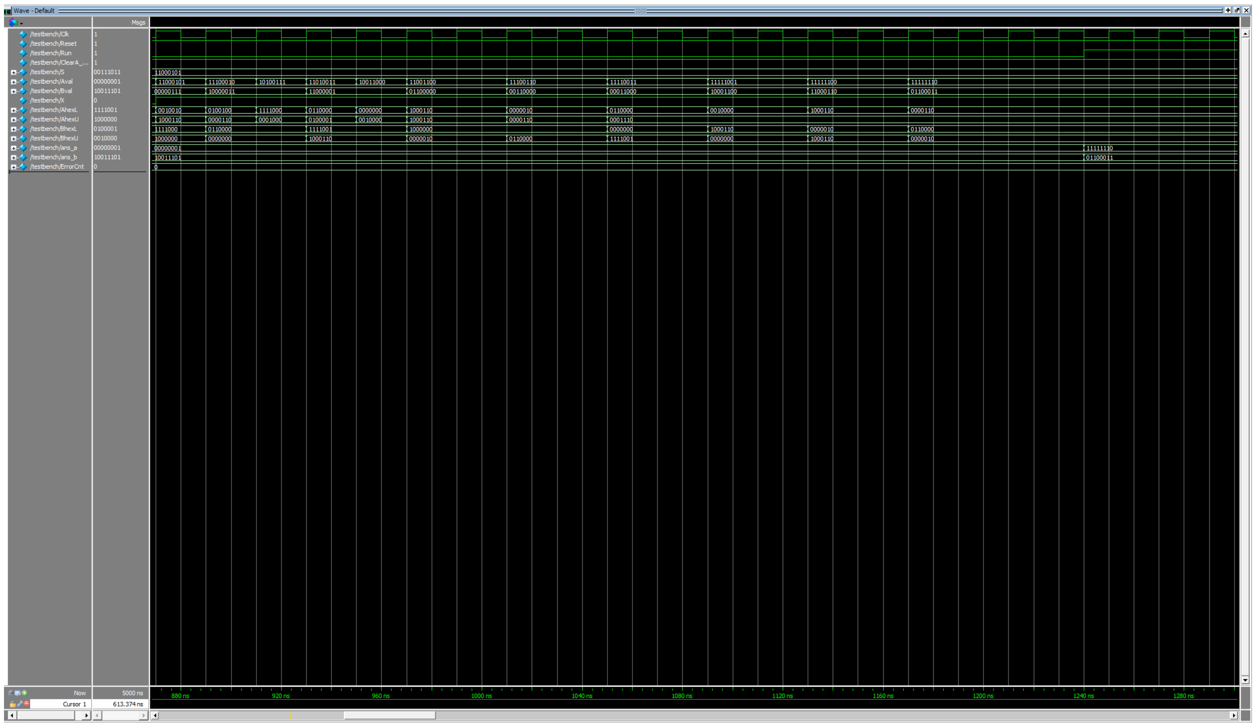

Figure 4: 7*(-59)

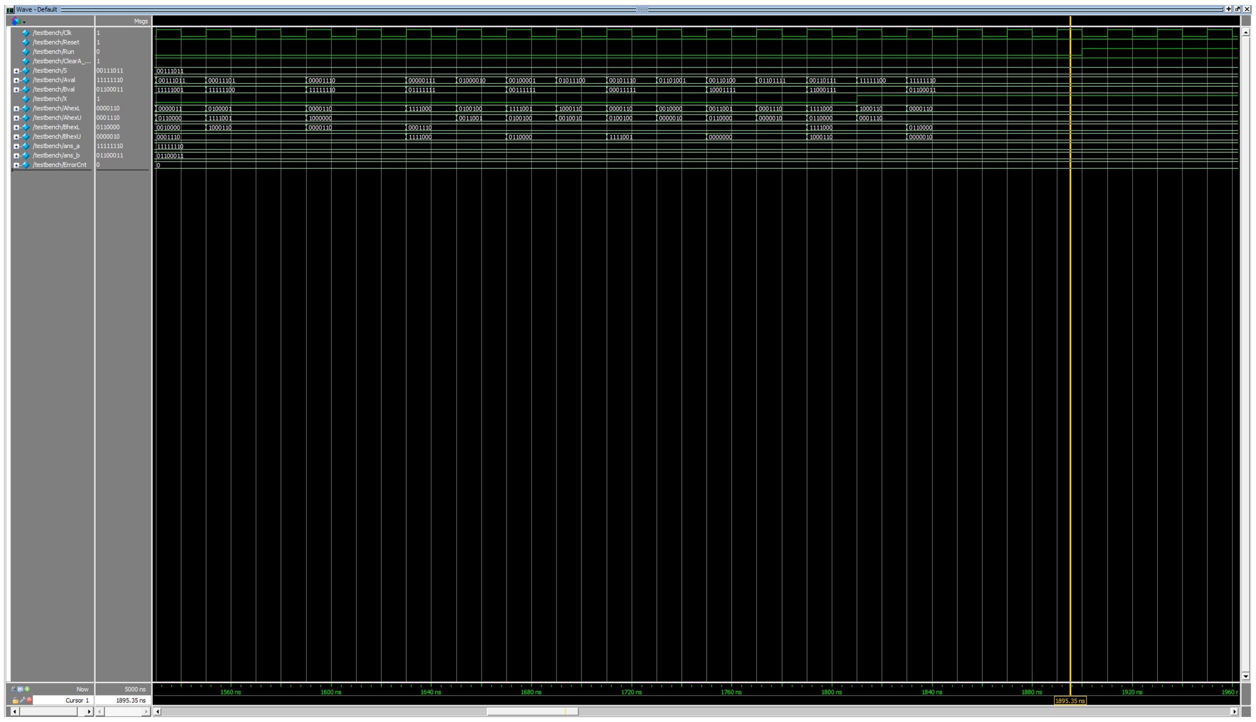

Figure 5: (-7)*59

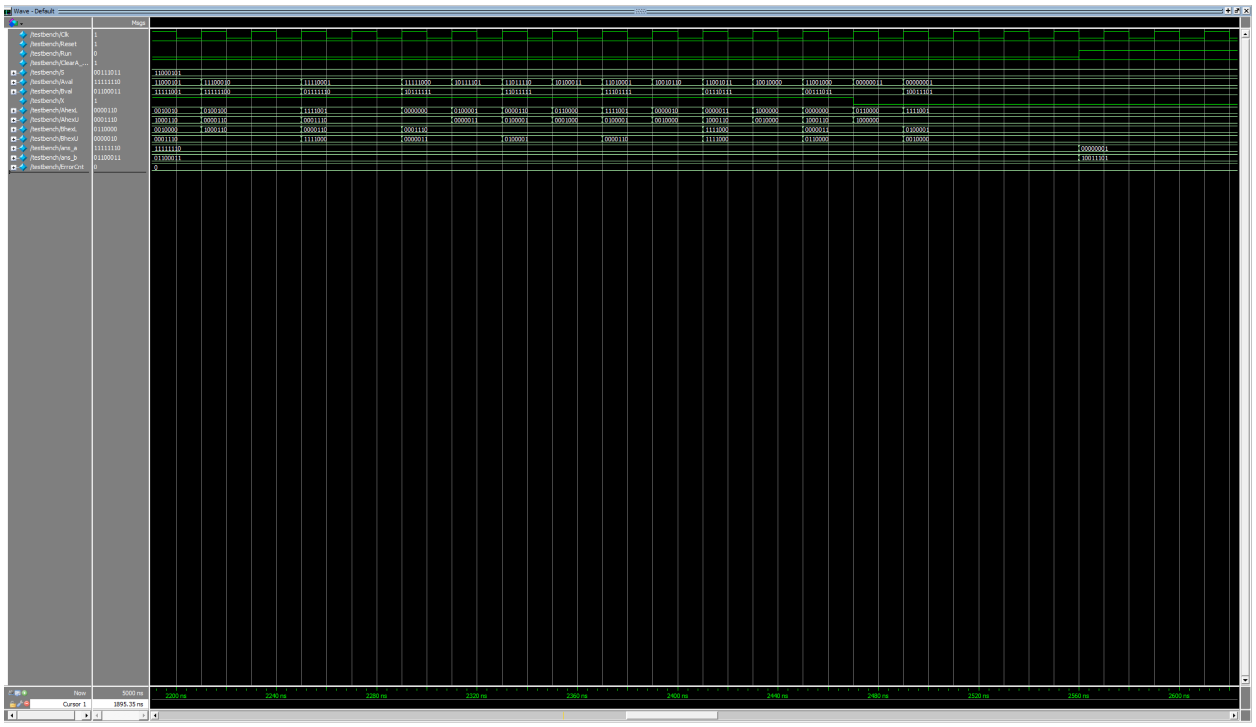

Figure 6: (-7)*(-59)

5 Answers to two post-lab questions (Fill in the table shown in 5.6 with your design's statistics Write down several ideas on how the maximum frequency of your design could be increased or the gate count could be decreased

5.1 design statistics table

LUT	106
DSP	0
Memory (BRAM)	0
Flip-Flop	38
Frequency	66.81MHz
Static Power	98.51mW
Dynamic Power	3.08mW
Total Power	144.94mW

5.2 Make sure your lab report answers at least the following questions

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

Because we do 8bit + 8bit addition, we need to have 9bit to store the result. So we use A for 8bits and X for the 9th bit.

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

It used complex way to add A and S to get X. If it is used to continuously calculate a product number outside [-128,127], it fails.

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

This continuous multiplication only used X, [7:0]A, [7:0]B, [7:0]S, [7:0]A2, [7:0]B2, for about 41 bits, saved a lot of memory space compared to pencil-paper method, which used more than 80 bits.

6 Conclusion (Discuss functionality of your design. If parts of your design didn't work, discuss what could be done to fix it. Was there anything ambiguous, incorrect, or unnecessarily difficult in the lab manual or given materials which can be improved for next semester? You can also specify what we did right so it doesn't get changed.

In this lab, we build an 8-bit multiplier by using continuous multiplication. We decompose it into 5 modules and used a clock signal to sync all modules.

Bugs encountered:

1. We didn't understand the addition for A and S at first and how to get X. So it takes time to fix. 2. At first, we clear A at the wrong state, so after it finished calculation, A is always 0.