ECE 385

Fall 2021

Experiment #5

Lab 5
An 8-Bit Multiplier in System Verilog

Guanshujie Fu 3190110666

Xiaomin Qiu 3190110717

Lab Section D231 18:00 – 21:00

TA: Yuchuan Zhu

1. Introduction

a) The multiplier circuit could multiply two 8-bit 2's compliment numbers and return a correct sign extended 16-bit answer. The output is displayed through two 8-bit registers and a flip-flop to display the product's sign.

2. Pre-lab question

*a) Rework the multiplication example on page 5.2 of the lab manual, as in compute 00000111 * 11000101 in a table like the example S = 11000101*

Function	X	A	B	M	Comments for the next step
ClearA_LoadB	0	0000 0000	0000 0111	1	Since M = 1, multiplicand (available from switches S) 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	Add S to A since M = 1.
ADD	1	1010 0111	1000 0011	1	Shift XAB by one bit after ADD complete
SHIFT	1	1101 0011	1100 0001	1	Add S to A since M = 1.
ADD	1	1001 1000	1100 0001	1	Shift XAB by one bit after ADD complete
SHIFT	1	1100 1100	0110 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 1000	0	Do not add S to A since M = 0. Shift XAB.
SHIFT	1	1111 1001	1000 1100	0	Do not add S to A since M = 0. Shift XAB.
SHIFT	1	1111 1100	1100 0110	0	Do not add S to A since M = 0. Shift XAB.
SHIFT	1	1111 1110	0110 0011	0	8th shift done. Stop. 16-bit Product in AB.

3. Written description and diagrams of multiplier circuit

a) Summary of operation

i. how operands are loaded

Operands are loaded based on the current state and signal M, which is the last bit of register B. If M is a 0, the multiplier will not allow register A to load the answer but simply shift the data. When M is 1, the control allows register A to be loaded with the result of the addition of values stored in registers A and S.

ii. how the multiplier computes its result

When the control is adding the 8th bit of the input number, if that bit is a 1, then the

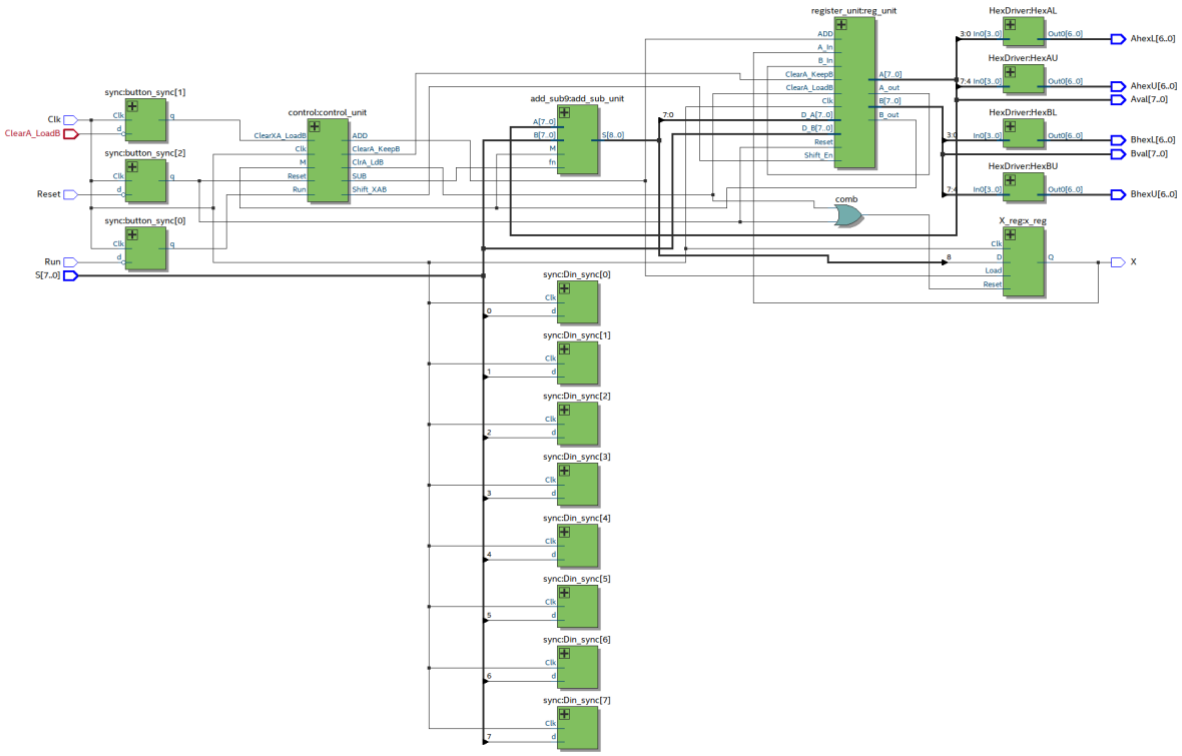
control outputs a subtraction signal that tells the 9-bit adder to do a subtraction instead of addition, then the result of that addition is once again stored into register A. The 9-bit adder is a combination of a 4-bit adder and a 5-bit adder. When a subtraction operation is required, the number being multiplied is *XOR*ed against that subtraction enable bit and addition is performed. For the subtraction operation to result in the correct 2's complement number, the carry-in bit of the adder is set to the subtraction enable bit so that the sum is right.

iii. *how the result is stored*

The multiplier circuit stores the multiplicand in register B, and the multiplier in switches S. The result product will be XAB and stored in registers.

b) Top Level Block Diagram

i. *This can be generated from the RTL viewer. Please only include the top-level diagram and not the RTL view of every module.*



c) *Written Description of .sv Modules*

i. List all modules

Modules	**Synchronizers.sv**
Inputs	logic Clk, d
Outputs	output logic q
Description	These are built by D flip-flops. Data is stored until it is on the positive edge of clock and then data will

	go from D to Q.
Purpose	These are synchronizers required for bringing asynchronous signals into the FPGA, like negating the reset signal, executing and ClearA_LoadB signal synchronously.

Modules	**X_reg.sv**
Inputs	logic Clk, Load, Reset, D
Outputs	logic Q
Description	This is a positive edge triggered flip-flop.
Purpose	This module could store the sign extension bit X.

Modules	**Reg_units.sv**
Inputs	logic Clk, Reset, A_In, B_In, ClearA_LoadB, ADD, Shift_En, ClearA_KeepB, logic [7:0] D_A, logic [7:0] D_B
Outputs	logic A_out, B_out, logic [7:0] A, logic [7:0] B
Description	This is the unit which combines two positive-edge triggered 8-bit register modules.
Purpose	It stores A and B values and output them directly by combining two Reg_8.sv modules.

Modules	**Reg_8.sv**
Inputs	logic Clk, Reset, Shift_In, Load, Shift_En, logic [7:0] D
Outputs	logic Shift_Out, logic [7:0] Data Out
Description	This is a positive edge triggered 8bit register with asynchronous reset and synchronous load. When load is high, data is loaded from din into the register on the positive edge of clk.
Purpose	This module is used to create the registers that store operands A or B in the adder circuit.

Modules	**Processor.sv**
Inputs	logic Clk, Reset, Run, ClearA_LoadB, logic [7:0] S
Outputs	logic [7:0] Aval, Bval,

	logic X, logic [6:0] AhexL, AhexU, BhexL, BhexU
Description	This is the top-level module that defines the registers, flip-flops, checks outputs from the state machine and uses these variables within the calls of the reg_8.sv modules and the 9-bit-adder module. After calling all the functions and compiling their data, it will operand the HexDriver module for the display of the data to take place on the FPGA.
Purpose	This module is the executioner of all the other modules. It transmutes variables from the state machine (control.sv) to other modules to give meaning and operation to the various states present within the state machine. Also, it establishes the X_reg to record the sign extension bit.

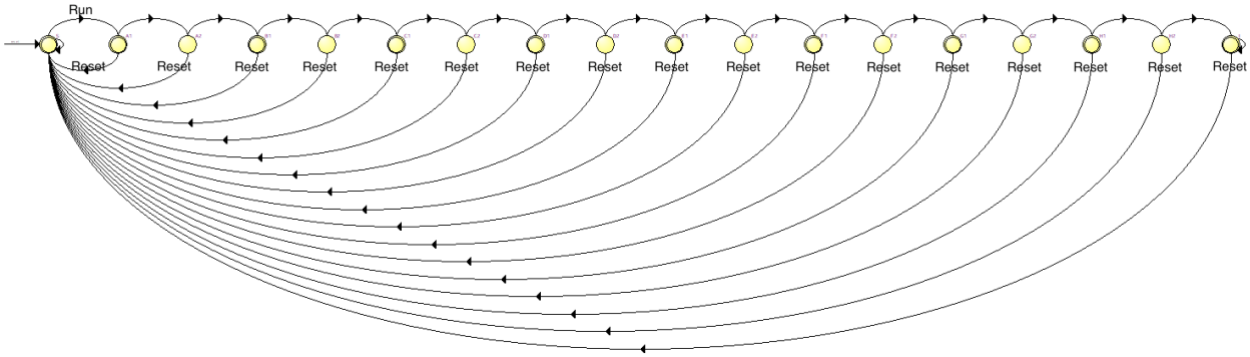
Modules	**HexDriver.sv**
Inputs	logic [3:0] In0
Outputs	logic [6:0] Out0
Description	This is a hex driver that transfers a 4-bit signal to represent a hex display on the FPGA. The 4 bits in are needed to have designations for hex values 0 to F.
Purpose	It helps to present the output hex-value on the board.

Modules	**Control.sv**
Inputs	logic Clk, Reset, Run, ClearXA_LoadB, M
Outputs	logic Shift_XAB, ADD, SUB, ClrA_LdB, ClearA_KeepB
Description	This is the state machine of the design. This module creates various states and available transitions for the states which are updated constantly in the top-level module Processor.sv.
Purpose	This module could control the operation of adders and registers and hold the states for the other modules. The module has 18 states with 8 add and shift states as well as a hold and start state. This module will then feed the outputs to the processor.sv module which will update the operation signal of the other modules based on the current state.

Modules	**Add_Sub_9.sv**
Inputs	logic [7:0] A, B, logic fn, M
Outputs	logic [8:0] S
Description	This module is a 9-bit carry_select_adder. Takes in two 8-bit binary signals and depending on the 8th number of B signal decides whether to add or subtract the S signal to the partial sum A. If the 8th B = 1 then the adder will subtract S from A. The result is the output as a bit extended sum with X as sign bit.
Purpose	The purpose of this module is to perform the operation of add or subtract depending on the 8th number of B.

d) State Diagram for Control Unit

i. 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!



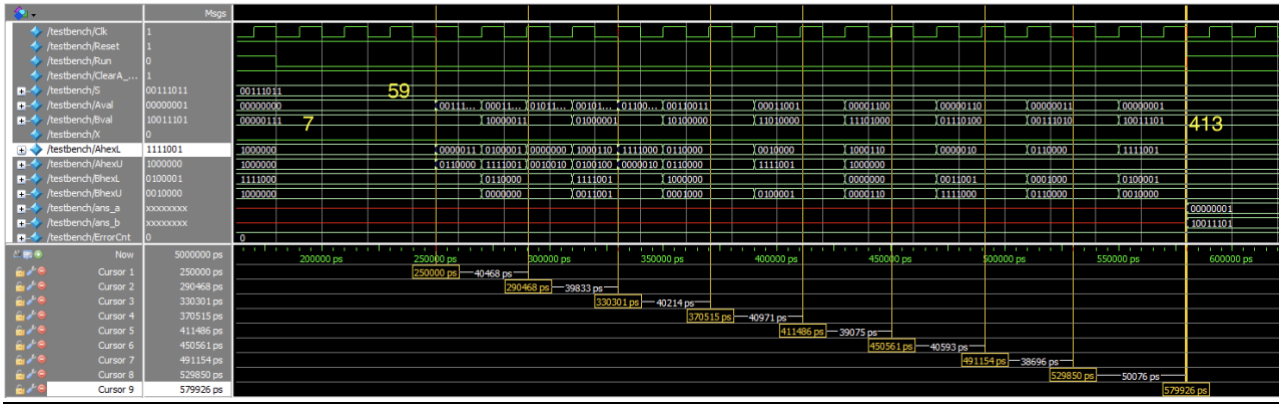
State Machine Diagram

4. Annotated pre-lab simulation waveforms

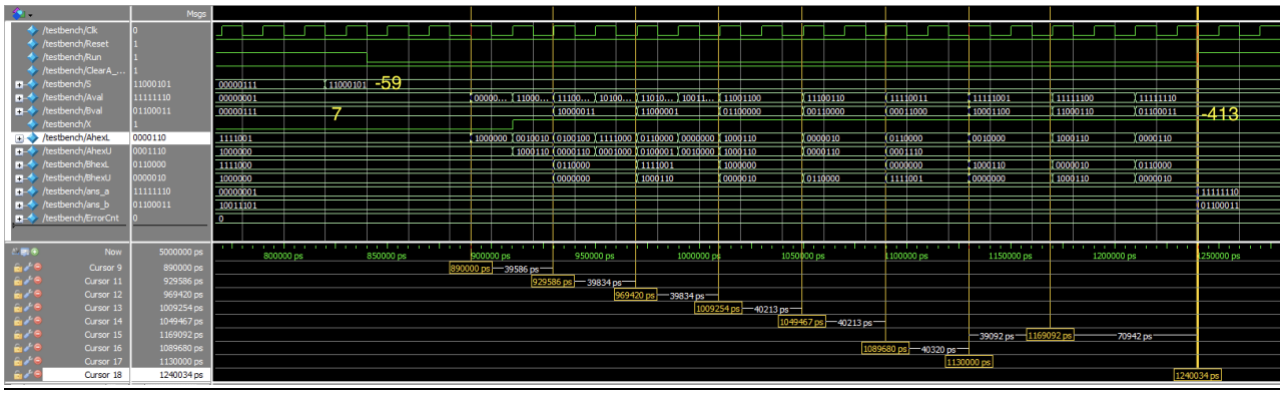
a) Create your own testbench for your multiplier which displays the following

i. 4 operations where operands have signs (++), (+*-), (-*+) and (-*-)*

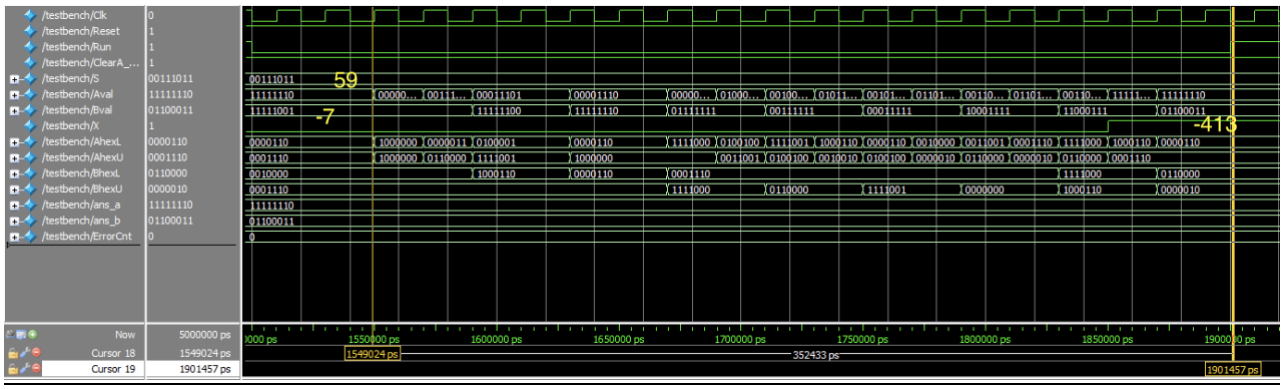
ii. Waveform must have notes that clearly show the operands as well as the result, etc.



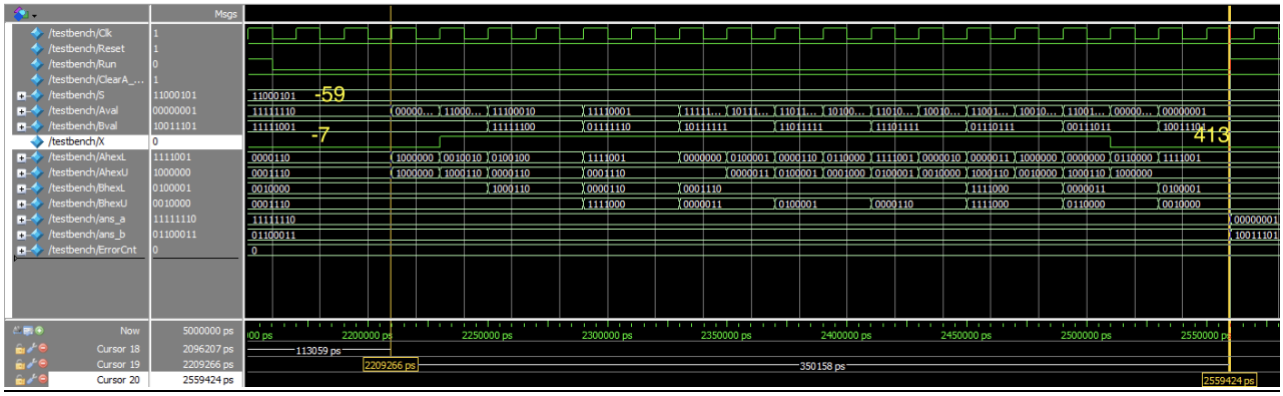
*Waveform of 7*59*



Waveform of 7-59*



*Waveform of -7*59*



Waveform of -7-59*

5. # Answers to two post-lab questions

a) *Fill in the table with your design's statistics*

LUT	99
DSP	0
Memory (BRAM)	0
Flip-Flop	39
Frequency	94.6MHz
Static Power	98.50mW
Dynamic Power	0.00mW
Total Power	143.00mW

b) *Write down several ideas on how the maximum frequency of your design could be increased or the gate count could be decreased.*

The maximum frequency could be increased by reducing the states in the state machine. It is possible to make the next state only depend on the least significant bit of register B instead adding and shifting for 8 times. As there are less states, the gates needed would be decreased and the maximum frequency would be increased. Also, we could replace all the XOR gates with NAND gated in the 8-bit adder module to decrease gates number.

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

The purpose of the register X is to hold the sign extension bit so that it would not get lost because of overflow. Also, the sign extension bit can be used to determine whether the addition or subtraction should be performed. If the addition results outputted by the 9-bit adder in a positive number, then the register holds a 0 (X = 0), if the output A is a negative number, the register holds 1 (X = 1).

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

The limitation of continuous multiplication is when the result of multiplication is more than 16 bits, the overflow will happen, and the algorithm will fail due to the overflow occurring in previous steps. The capability of the multiplication is only 16 bits in this lab.

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

Advantages:

Firstly, we do not have to keep track of the subtraction problem since it automatically determines whether it should subtract or not with the signal *M* and the state. Secondly, the multiplicand could be stored in B until shifted out, which allows continuous

multiplication. Also, less memory is needed as the implemented multiplication algorithm fits each step within 17-bits.

Disadvantages:

For the algorithm we implemented, though it can realize continuous multiplications, it might lead to overflow problem since it only has a 16-bit register to store the result calculated in each step. As we have stated in *e*) above, if the numbers used in continuous multiplication are too large which can lead to overflow, then the calculated result can be wrong due to the overflow occurring in previous steps.

6. Conclusion

a) Discuss functionality of your design. If parts of your design didn't work, discuss what could be done to fix it

Our design implements a 2's complement 8-bit multiplier. Our circuit mainly uses two 8-bit shift registers and an 8-bit switch. Our design works well, and the functionality is good.

b) 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.

We think the lab manual does not have a detailed introduction to the structure of what we need to implement. It only has four pages of description while we implemented more than four pages.