ECE 385

Fall 2021

Experiment 5

An 8-Bit Multiplier in SystemVerilog

Name: Li Boyan; Guo Moyang
Section: D225; evening
UID: 3190110007; 3190110316

Introduction

In this lab, we constructed 8-bits 2's complement multiplier in System Verilog, i.e., it can give the right sign number by doing multiplication extended to negative number. The signed 16-bits answer will be displayed in 4-digits Hex value on FPGA.

Pre-lab question

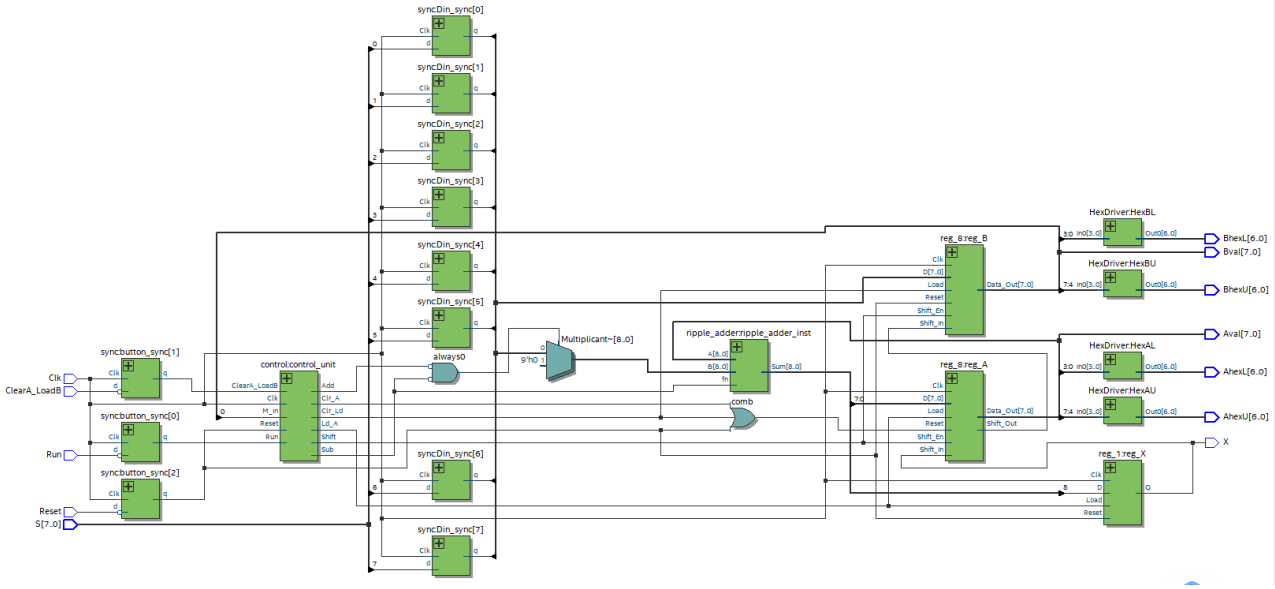
Function	X	A	B	M
Clear A, Load B	0	0000 0000	0000 0111	1
ADD	1	1100 0101	0000 0111	1
SHIFT	1	1110 0010	1000 0011	1
ADD	1	1010 0111	1000 0011	1
SHIFT	1	1101 0011	1100 0001	1
ADD	1	1001 1000	1100 0001	1
SHIFT	1	1100 1100	0110 0000	0
SHIFT	1	1110 0110	0011 0000	0
SHIFT	1	1111 0011	0001 1000	0
SHIFT	1	1111 1001	1000 1100	0
SHIFT	1	1111 1100	1100 0110	0
SHIFT	1	1111 1110	0110 0011	0

Written description and diagrams of multiplier circuit

a. Summary of operation

We decide operands on the least significant bit of register B and the signals sent by control unit. We denote B [0] as M, if M = 1, ADD =1, we use 9-bits extended sign adder to compute the value in register A and the value in switches; if M = 1, SUB = 1, we perform subtraction rather than addition; if M=0, our program ignores the ADD and SUB signal and simply add 0 to register A, i.e., do nothing; when SHIFT = 1, both register A and register B will do right shift in 1-bit. After 8 times shift operation, the answer will be AB (16-bits).

b. Top Level Block Diagram

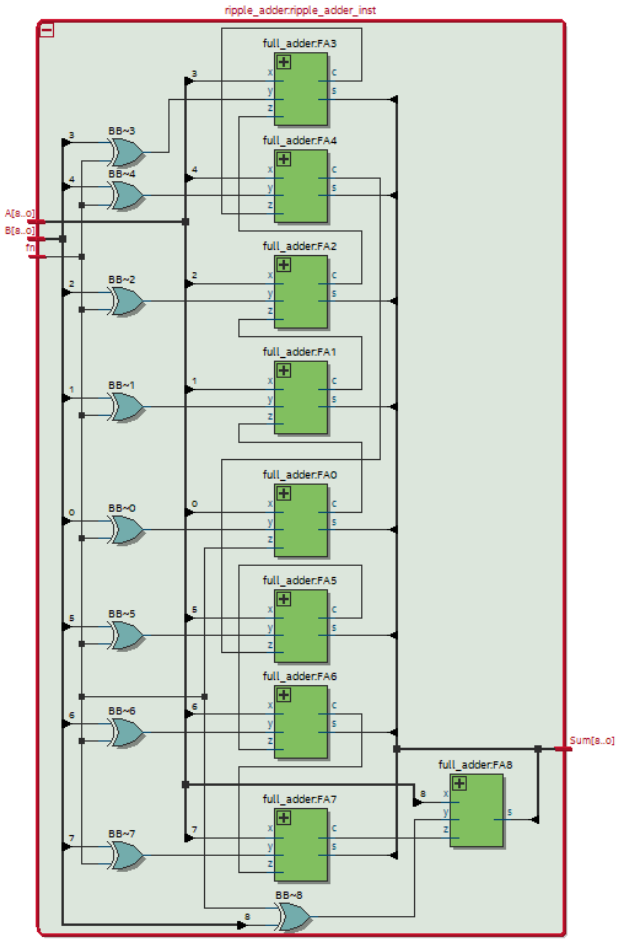


c. Written Description of .sv Modules

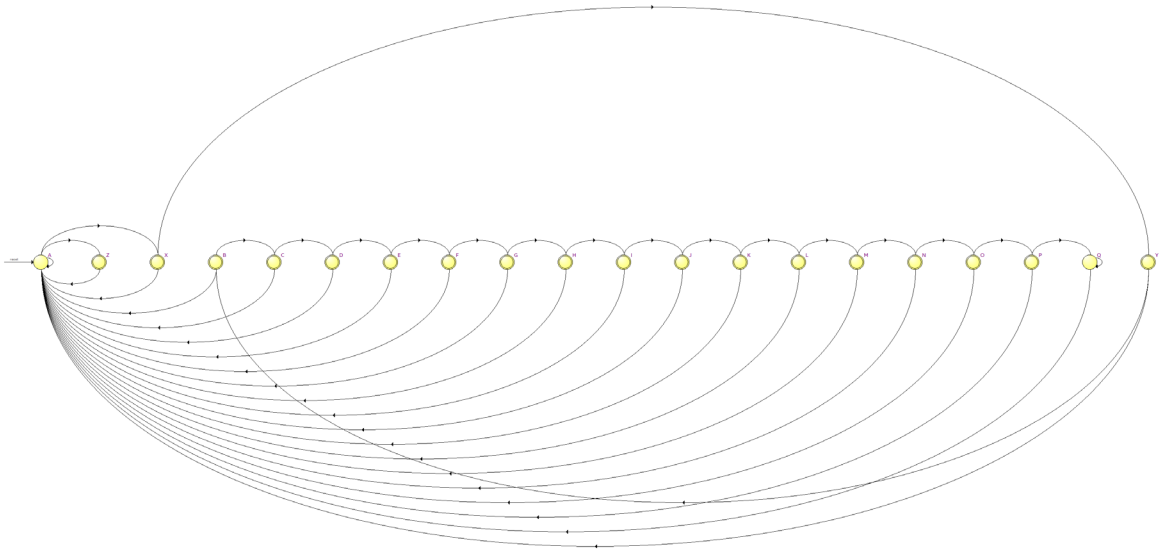
Module	**Inputs**	**Outputs**
Lab5_toplevel	Clk, Rest, ClearA_LoadB, Run, [7:0] S	[7:0] Aval, [7:0] Bval, X, [6:0] AhexL, [6:0] AhexU, [6:0] BhexL, [6:0] BhexU
Synchronizers	Clk, d	q
Control	Clk, Rest, ClearA_LoadB, Run, M_in	Clr_Ld, Shift, Add, Sub, Ld_A, Clr_A
Reg_1	Clk, Load, Rest, D	Q
Reg_8	Clk, Reset, Shift_In, Load, Shift_En, [7:0] D	[7:0] Data_Out, Shift_Out
Ripple_adder	[8:0] A, [8:0] B, fn	[8:0] Sum
HexDriver	[3:0] In0	[6:0] Out0

Module	Description	Purpose
Lab5_toplevel	Top-level module	It organizes other modules to perform the 8-bit multiplier
Synchronizers	A D flip-flop. Data go from d to q at the positive edge of clock	It works as a synchronizer to negate the reset, run and ClearA_LoadB signal synchronously.
Control	State machine	It works to send signal to different modules; and change states at the positive edge of clock
Reg_1	A D flip-flop.	It is used to store the extension bit X
Reg_8	8-bit registers	It is used to store the value in A and B; and do shift operand when needed.
Ripple_adder	9-bit ADD-SUB unit using ripple adder	It is used to perform addition and subtraction operand when needed
HexDriver	A combinational logic to generate hex driver value	It is used to display the answer on FPGA board

Diagram of module ‘Ripple_adder’ are attached to show how to extend subtraction to adder.



d. State Diagram for Control Unit



States transition:

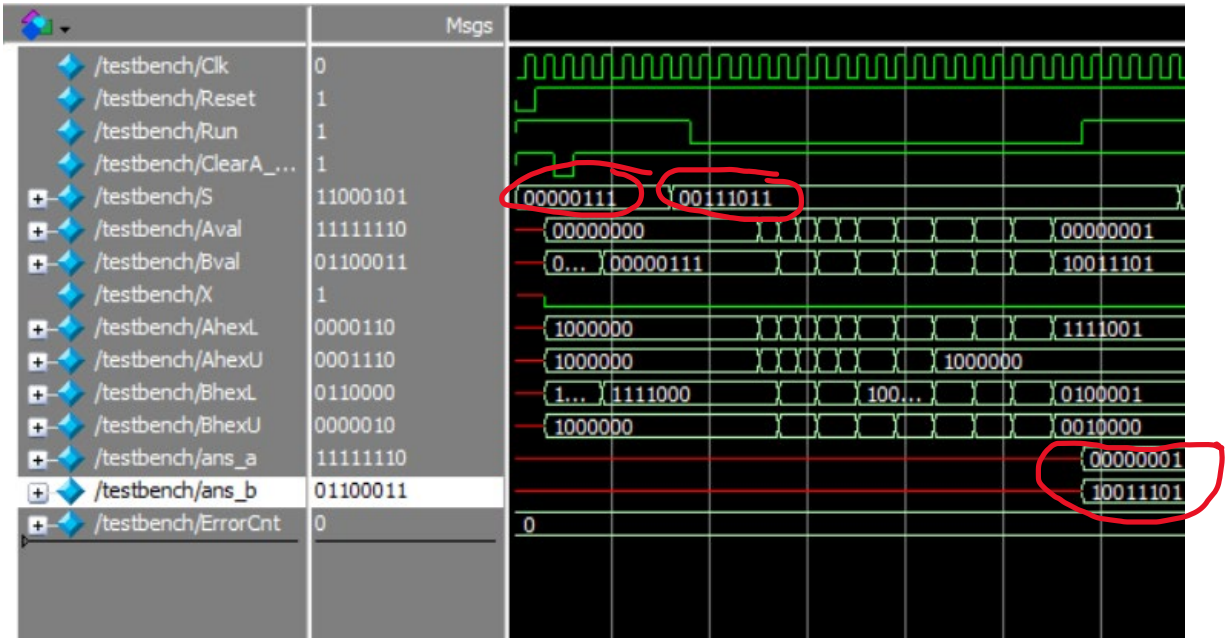
From	To	Transitions
A	Z	(ClearA_LoadB).(!Reset)
A	X	(Run).(!ClearA_LoadB).(!Reset)
A	A	(!Run).(!ClearA_LoadB) + (!Run).(ClearA_LoadB).(Reset) + (Run).(Reset)
B	A	(Reset)
B	C	(!Reset)
C	A	(Reset)
C	D	(!Reset)
D	A	(Reset)
D	E	(!Reset)
E	A	(Reset)
E	F	(!Reset)
F	A	(Reset)
F	G	(!Reset)
G	A	(Reset)
G	H	(!Reset)
H	A	(Reset)
H	I	(!Reset)
I	A	(Reset)
I	J	(!Reset)
J	A	(Reset)
J	K	(!Reset)
K	L	(!Reset)
K	A	(Reset)
L	M	(!Reset)
L	A	(Reset)
M	N	(!Reset)
M	A	(Reset)
N	O	(!Reset)
N	A	(Reset)
O	P	(!Reset)
O	A	(Reset)
P	Q	(!Reset)
P	A	(Reset)
Q	Q	(Run).(!Reset)
Q	A	(!Run) + (Run).(Reset)
X	Y	(!Reset)
X	A	(Reset)
Y	A	(Reset)
Y	B	(!Reset)
Z	A	

Totally, we have 20 states, starting states A and ending states Q. 16 States {Y, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P} are computing states, Y is computation1, B is shift1, C is computation2, D is shift2, E is computation3, F is shift3 and so on. There only remain two states: Z is state to perform the operation “clear register A and load register B”; X is state

to clear register A because we want to perform consecutive multiplication.

Annotated pre-lab simulation waveforms

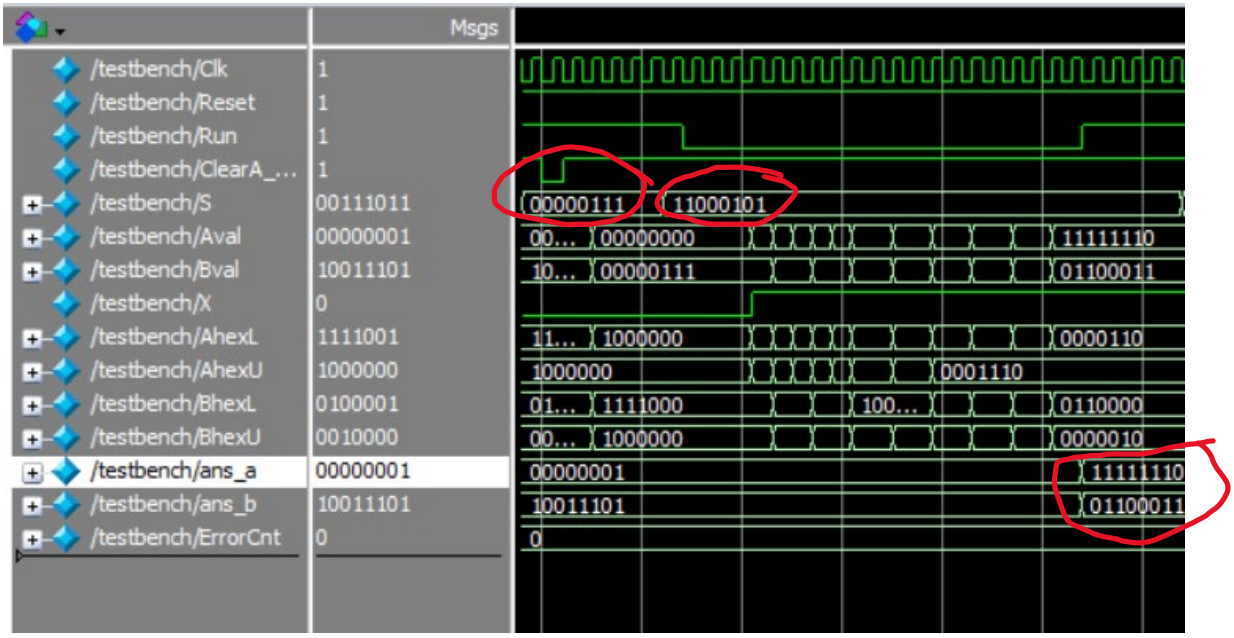
Condition 1: + and +



First S is multiplier and second S is multiplicand.

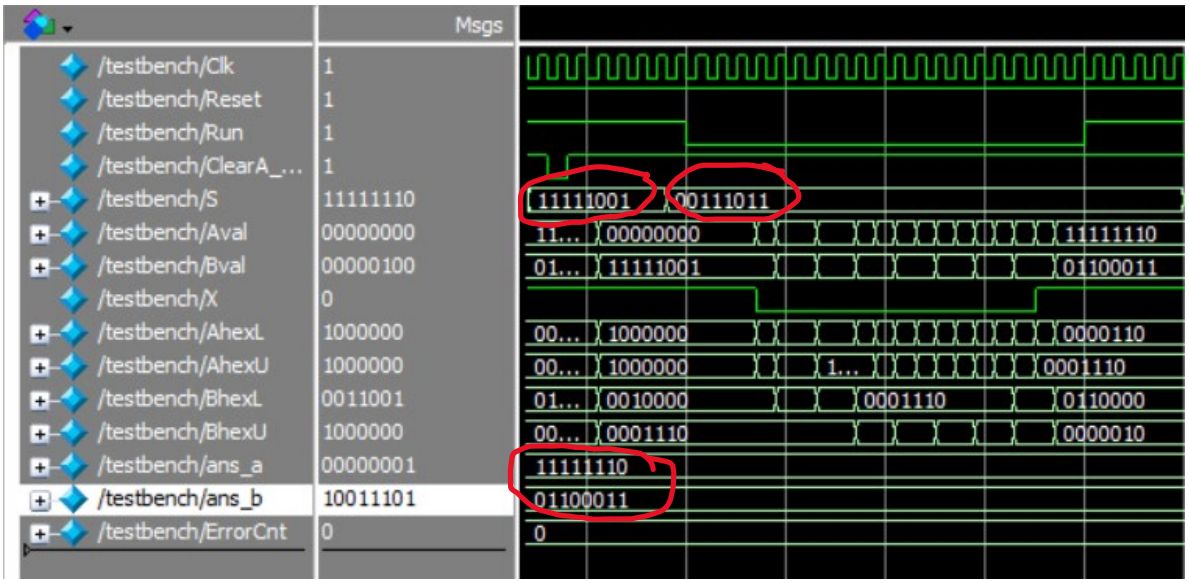
A = 0011 1011 (59) B = 0000 0111 (7)
AB = 0000 0001 1001 1101(413)

Condition 2: - and +



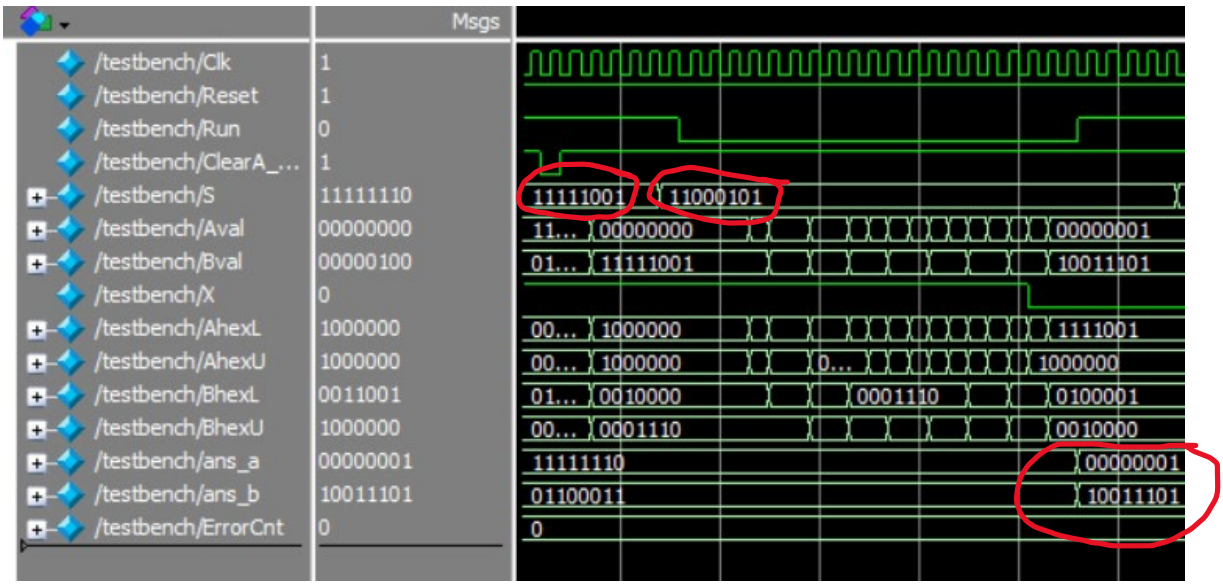
A = 1100 0101 (-59) B = 0000 0111 (7)
AB = 1111 1110 0110 0011(-413)

Condition 3: + and -



A = 0011 1011 (59) B = 1111 1001 (-7)
AB = 1111 1110 0110 0011(-413)

Condition 4: - and -



A = 1100 0101 (-59) B = 1111 1001 (-7)
AB = 0000 0001 1001 1101 (413)

Answers to post-lab questions

LUT	104
DSP	N/A
Memory	0
Flip-Flop	48
Frequency	63.03MHz
Static Power	98.51mW
Dynamic Power	2.48mW
Total Power	144.34mW

a. The easiest way to refine our design is to develop the state further condense. Our design use shift and compute states in exactly the number of bits, i.e., 8 bits; however, we can reuse some states and extend to a more general design (extend to more bits).
b. The purpose of register X is used to contain the sign number, because doing shift need to maintain the sign. When we do Run, we set X; when we do ClearA_LoadB, we clear X.
c. Continuous multiplication can easily lead to overflow, because we can only display the right number of 16-bits 2's complement; when the result is greater than 16-bits, the algorithm fails.
d. Advantage: It can automatically switch between addition and subtraction to get the answer fast.
   Disadvantage: It will lead to overflow problem.

Conclusion

a. Our design can do 8-bits 2's complement multiplication. When the least bit of B is 0, we do nothing, when the least bit of B is 1, the algorithm decides whether do addition or subtraction. It can do consecutive multiplication because we have a state to clear register A.
b. The problem is that we have class on Tuesday and Do demo on Thursday; however, lectures show more details on how to write a module related to the lab, for example, 9-bits ADD-SUB unit, which means we can hardly complete our demo before Tuesday. Therefore, I hope that we can get some tricks on how to write the core module related to the lab.