ECE 385

EXPERIMENT #5

An 8-Bit Multiplier in SystemVerilog

I. OBJECTIVE

In this experiment, you will design a multiplier in SystemVerilog for two 8-bit 2's compliment numbers and then run that multiplier on the DE10-Lite FPGA board.

II. INTRODUCTION

You will use a simple add-shift algorithm to multiply two numbers. The algorithm is very similar to the pencil-and-paper method of multiplication except the final step for 2's Complement numbers depends on the sign bit. Consider the following example to calculate 8-bit 00000111 (7, Multiplicand) x 11000101 (-59, Multiplier)

```
  00000111        7 (multiplicand)
x 11000101      x (-)59 (multiplier)
 00000111       (-)413
+00000000x
+00000111xx
+00000000xxx
+00000000xxxx
+00000000xxxxx
+00000111xxxxxx
-00000111xxxxxxx  Subtract (or Add 2's comp of 00000111)
1111111001100011 (2's comp of result=0000000110011101=413)
```

Let us see how to perform multiplication using the add-shift method that you will use to multiply the contents of register B and switches S, leaving the result in registers AB:

Initial Values: X = 0, A = 00000000, B = 11000101 (achieved using ClearA_LoadB signal), S = 00000111, M is the least significant bit of the multiplier (Register B).

Function	X	A	*B*	M	Comments for the next step
Clear A, LoadB, Reset	0	0000 0000	*11000101*	1	Since M = 1, multiplicand (available from switches S) will be added to A.
ADD	0	0000 0111	*11000101*	1	Shift XAB by one bit after ADD complete
SHIFT	0	0000 0011	1 *1100010*	0	Do not add S to A since M = 0. Shift XAB.
SHIFT	0	0000 0001	11 *110001*	1	Add S to A since M = 1.
ADD	0	0000 1000	11 *110001*	1	Shift XAB by one bit after ADD complete
SHIFT	0	0000 0100	011 *11000*	0	Do not add S to A since M = 0. Shift XAB.
SHIFT	0	0000 0010	0011 *1100*	0	Do not add S to A since M = 0. Shift XAB.
SHIFT	0	0000 0001	00011 *110*	0	Do not add S to A since M = 0. Shift XAB.
SHIFT	0	0000 0000	100011 *11*	1	Add S to A since M = 1
ADD	0	0000 0111	100011 *11*	1	Shift XAB by one bit after ADD complete
SHIFT	0	0000 0011	1100011 *1*	1	Subtract S from A since 8^{th} bit M = 1.
SUB	1	1111 1100	1100011 *1*	1	Shift XAB after SUB complete
SHIFT	1	1111 1110	01100011	1	8^{th} shift done. Stop. 16-bit Product in AB.

In the ADD state, the values of A and S are first sign-extended to 9 bits, and then summed together. The 9-bit results (not including the Cout) are then stored into XA. In the SHIFT state, the entire 17 bits of XAB is arithmetically right-shifted by one bit.

When M = 0, an ADD does not need to be performed. In that case, the ADD cycle can be omitted or a zero can be added to A. In addition, since we are using a 2's complement representation, we need to consider negative numbers. If A is negative, then XA will contain the correct partial sum and the sign will be preserved since the shift operation will perform an arithmetic shift on XAB. If B is negative (the most significant bit = 1), then M will be 1 after the seventh shift (see the example above). In that case a subtract operation is performed since the 8^{th} bit of B has negative weight with 2's complement representation.

The 9-bit Adder/Subtractor should be designed using Full Adder primitives that you create. In other words, do not use the available SystemVerilog arithmetic operations "+" (add) and "-" (subtract) for this experiment. In future, you may use these operations in your designs.

You should design your control unit such that it executes one multiply operation when the Run press button is pressed. You can use symbolic states for the state machine in the controller for this experiment. You will need to have a Reset input that will reset the controller in the initial/start state. An **incomplete** block diagram of the circuit is shown in Figure 1:

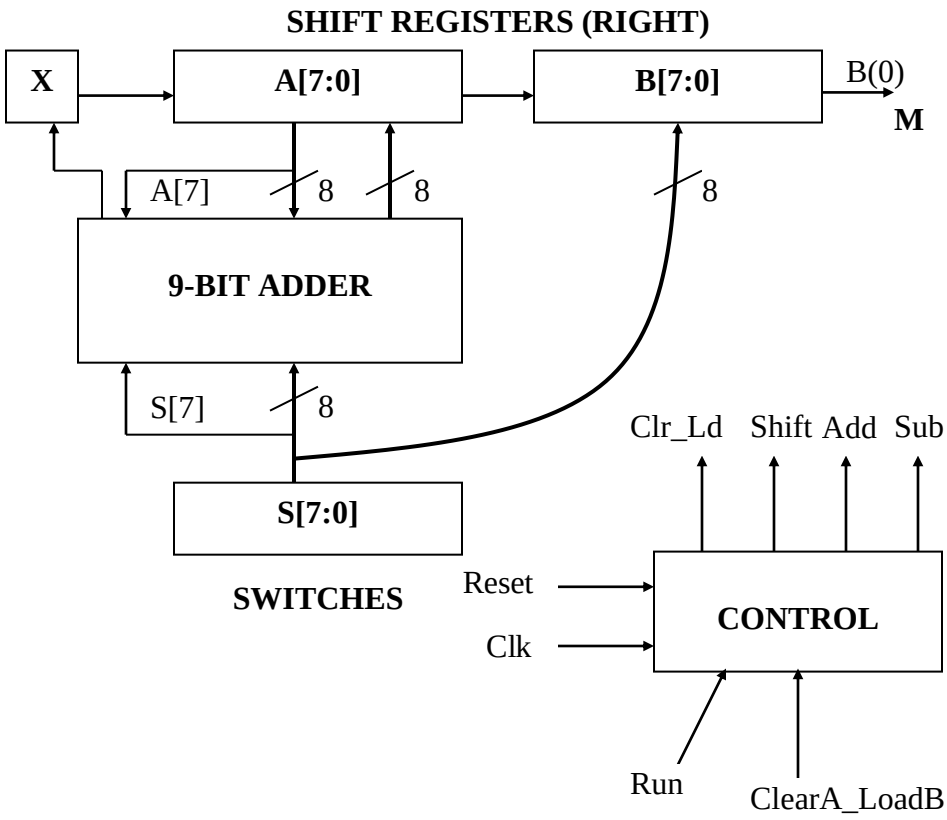

Figure 1: Incomplete Block Diagram

Your circuit should have the following inputs and outputs:

Inputs

SW – logic [7:0]

Clk, Reset_Load_Clear, Run– logic

Outputs

HEX1, HEX0, HEX3, HEX2– logic [6:0]

Aval, Bval – logic [7:0]

Xval –logic

To perform a multiplication, you will first load the multiplier to Register B by setting the switches (S) to represent the multiplier and pressing the Reset_Load_Clear button. Reset_Load_Clear button should also clears the X and A registers. Then you will set the switches

(S) to represent the multiplicand and press the Run button. Reset_Load_Clear should be released before Run button is pushed. Once the Run signal triggers the multiplication, the circuit should complete the multiply operation regardless of the status of Run signal. The circuit should stop once the multiplication is done and the correct result should be displayed by outputting AB on the hex displays. Another multiply operation can be triggered by releasing the Run button and pressing it again.

Your circuit should support consecutive multiplications to receive full demo points. Note that the Reset_Load_Clear buttons will not be pressed between consecutive presses of the Run button, so your circuit needs to clear X and A before the next multiplication execution starts to get the correct results.

Demo Points Breakdown:

1.0 point: Functional simulation completed successfully

1.0 point: Correct operation of the Reset_Load_Clear function on the DE10-Lite board

1.0 point: Correct operation of the *Multiplication* function on the DE10-Lite board. (++, +-, -+, --)

1.0 point: Correct operation of *consecutive* multiplications on the DE10-Lite board. (like -1 × -1 × …)

1.0 point: Execution cycle responds correctly (exactly one execution per press of the "Run" button)

III. PRE-LAB

A. Rework the 8-bit multiplication example presented in the table form at the beginning of this assignment. Use Multiplier B = 7, and Multiplicand S = -59. Note that this is different than the case when B = -59 and S = 7.

B. Design, document, and implement the 8-bit multiplier in SystemVerilog.

You will need to bring the following to the lab:

1. Your code for the 8-bit multiplier. You can bring the code to the lab using a USB storage device, FTP, or any other method.

2. Block diagram of your design, with components, ports, and interconnections labeled.

3. A simulation of your design showing at least one full multiplication. You should set the radix of the Switches, Aval, and Bval signals to signed decimal for readability. (Radix is set by right-clicking on a signal and selecting *Properties*.)

IV. LAB

Follow the Lab 5 demo information on the course website.

Pin Assignment Table

Port Name	Location	Comments
SW[0]	PIN_C10	On-board slider switch (SW0)
SW[1]	PIN_C11	On-board slider switch (SW1)
SW[2]	PIN_D12	On-board slider switch (SW2)
SW[3]	PIN_C12	On-board slider switch (SW3)
SW[4]	PIN_A12	On-board slider switch (SW4)
SW[5]	PIN_B12	On-board slider switch (SW5)
SW[6]	PIN_A13	On-board slider switch (SW6)
SW[7]	PIN_A14	On-board slider switch (SW7)
SW[8]	PIN_B14	On-board slider switch (SW8)
SW[9]	PIN_F15	On-board slider switch (SW9)
LED[0]	PIN_A8	On-Board LED (LED0)
LED[1]	PIN_A9	On-Board LED (LED1)
LED[2]	PIN_A10	On-Board LED (LED2)
LED[3]	PIN_B10	On-Board LED (LED3)
LED[4]	PIN_D13	On-Board LED (LED4)
LED[5]	PIN_C13	On-Board LED (LED5)
LED[6]	PIN_E14	On-Board LED (LED6)
LED[7]	PIN_D14	On-Board LED (LED7)
LED[8]	PIN_A11	On-Board LED (LED8)
LED[9]	PIN_B11	On-Board LED (LED9)
HEX0[0]	PIN_C14	On-Board seven-segment display segment
HEX0[1]	PIN_E15	On-Board seven-segment display segment
HEX0[2]	PIN_C15	On-Board seven-segment display segment
HEX0[3]	PIN_C16	On-Board seven-segment display segment
HEX0[4]	PIN_E16	On-Board seven-segment display segment
HEX0[5]	PIN_D17	On-Board seven-segment display segment
HEX0[6]	PIN_C17	On-Board seven-segment display segment
HEX0[7]	PIN_D15	On-Board seven-segment display segment
HEX1[0]	PIN_C18	On-Board seven-segment display segment
HEX1[1]	PIN_D18	On-Board seven-segment display segment
HEX1[2]	PIN_E18	On-Board seven-segment display segment
HEX1[3]	PIN_B16	On-Board seven-segment display segment

HEX1[4]	PIN_A17	On-Board seven-segment display segment
HEX1[5]	PIN_A18	On-Board seven-segment display segment
HEX1[6]	PIN_B17	On-Board seven-segment display segment
HEX1[7]	PIN_A16	On-Board seven-segment display segment
HEX2[0]	PIN_B20	On-Board seven-segment display segment
HEX2[1]	PIN_A20	On-Board seven-segment display segment
HEX2[2]	PIN_B19	On-Board seven-segment display segment
HEX2[3]	PIN_A21	On-Board seven-segment display segment
HEX2[4]	PIN_B21	On-Board seven-segment display segment
HEX2[5]	PIN_C22	On-Board seven-segment display segment
HEX2[6]	PIN_B22	On-Board seven-segment display segment
HEX2[7]	PIN_A19	On-Board seven-segment display segment
HEX3[0]	PIN_F21	On-Board seven-segment display segment
HEX3[1]	PIN_E22	On-Board seven-segment display segment
HEX3[2]	PIN_E21	On-Board seven-segment display segment
HEX3[3]	PIN_C19	On-Board seven-segment display segment
HEX3[4]	PIN_C20	On-Board seven-segment display segment
HEX3[5]	PIN_D19	On-Board seven-segment display segment
HEX3[6]	PIN_E17	On-Board seven-segment display segment
HEX3[7]	PIN_D22	On-Board seven-segment display segment
HEX4[0]	PIN_F18	On-Board seven-segment display segment
HEX4[1]	PIN_E20	On-Board seven-segment display segment
HEX4[2]	PIN_E19	On-Board seven-segment display segment
HEX4[3]	PIN_J18	On-Board seven-segment display segment
HEX4[4]	PIN_H19	On-Board seven-segment display segment
HEX4[5]	PIN_F19	On-Board seven-segment display segment
HEX4[6]	PIN_F20	On-Board seven-segment display segment
HEX4[7]	PIN_F17	On-Board seven-segment display segment
HEX5[0]	PIN_J20	On-Board seven-segment display segment
HEX5[1]	PIN_K20	On-Board seven-segment display segment
HEX5[2]	PIN_L18	On-Board seven-segment display segment
HEX5[3]	PIN_N18	On-Board seven-segment display segment
HEX5[4]	PIN_M20	On-Board seven-segment display segment
HEX5[5]	PIN_N19	On-Board seven-segment display segment
HEX5[6]	PIN_N20	On-Board seven-segment display segment
HEX5[7]	PIN_L19	On-Board seven-segment display segment
Clk	PIN_P11	50 MHz Clock from the on-board oscillators
Reset_Load_Clear	PIN_B8	On-Board Push Button (KEY0)
Run	PIN_A7	On-Board Push Button (KEY1)

Astute readers will realize that Aval and Bval are given as outputs to the top-level, but are not included in the pin assignments, this is intentional. These outputs are included primarily for simulation, where reading the hex display outputs is not practical. Similarly, reading the outputs of your circuit in binary is not as convenient as reading them in hex. You are free to reuse the assignments from Lab 4 for these signals if you wish.

V. POST-LAB

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

LUT	
DSP	
Memory (BRAM)	
Flip-Flop	
Frequency	
Static Power	
Dynamic Power	
Total Power	

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.

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?
- What are the limitations of continuous multiplications? Under what circumstances will the implemented algorithm fail?
- What are the advantages (and disadvantages?) of the implemented multiplication algorithm over the pencil-and-paper method discussed in the introduction?

VI. REPORT

Write a report, you may follow the provided outline below, or make sure your own report outline includes at least the items enumerated below.

1. Introduction
    a. Summarize the basic functionality of the multiplier circuit

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 similar to the example
3. Written description and diagrams of multiplier circuit
    a. Summary of operation
        i. Explain in words how operands are loaded, how the multiplier computes its result, how the result is stored, etc.
    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 used in a format shown in the appendix of this document
        ii. You may insert expanded RTL diagrams of each individual module here if it is legible
    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. By default, the tool does not generate a very legible state diagram.
4. Annotated pre-lab simulation waveforms
    a. Must show 4 operations where operands have signs (+*+), (+*-), (-*+) and (-*-)
    b. Waveform must have notes that clearly show the operands as well as the result, etc.
5. Answers to post-lab questions
    a. Fill in the table shown in 5.6 with your design's statistics
    b. Answer all the post-lab questions. As usual, they may be in their own section or dispersed into the appropriate sections in the rest of the report.
6. Conclusion
    a. Discuss functionality of your design. If parts of your design didn't work, discuss what could be done to fix it
    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.

VII. APPENDIX

Module descriptions are an important part of the reports in ECE 385, and since this is the first significant FPGA lab, a brief example of how to write a module description is shown below.

Let's say you needed two 16-bit registers to store operands A and B in an adder that computes the sum of A and B. Here is example code of a 16-bit register with asynchronous reset and synchronous load that can be used for that purpose.

```
module reg16 (input [15:0] Din, input Clk, Load, Reset,
output logic [15:0] Dout);

always_ff @(posedge Clk or posedge Reset)
     begin
          if(Reset)
               Dout <= 16'h0000;
          else if(Load)
               Dout <= Din; //If load=1, perform parallel load
     on clock edge
     end
endmodule
```

And here is how a section of the report would describe it:

Module: `reg16.sv`

Inputs: `[15:0] Din, Clk, Load, Reset`

Outputs: `[15:0] Dout`

Description: This is a positive-edge triggered 16-bit 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 and B in the adder circuit.

Simple modules can have a description and purpose that are just a sentence or two each, but more complicated modules require more detailed descriptions.