ECE 385

EXPERIMENT #4

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 using logical operations and then run that multiplier on the Urbana 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)
```

Consider the same example, slightly modified to be more suitable for logical implementation (why?). The following illustrates the add-shift method that you will use to multiply the contents of register B and switches S, leaving the result in registers AB. Note that the initial values are set by setting the switches to 11000101, pressing Reset_ClearA_LoadB, and then changing the switches to 00000111 and pressing Run.

Initial Values: X = 0, A = 00000000, B = 11000101 (achieved using Reset_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 are arithmetically right shifted by one bit while the value of X is preserved.

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 logical CRA/CLA/CSA adder modules that you have created in previous labs. In other words, <u>do not use</u> the available SystemVerilog arithmetic operations "+" (add) and "-" (subtract) for this experiment. For future labs that are more complex, 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 into the controller which will reset the controller in the initial/start state. Note that this input may be shared with the LoadB and Clear A functionality. All pushbuttons should be de-bounced, to prevent multiple operations on a single button press. A partial block diagram of the circuit is shown in Figure 1. Use this along with previously provided SystemVerilog modules to implement the data-path of the multiplier circuit.

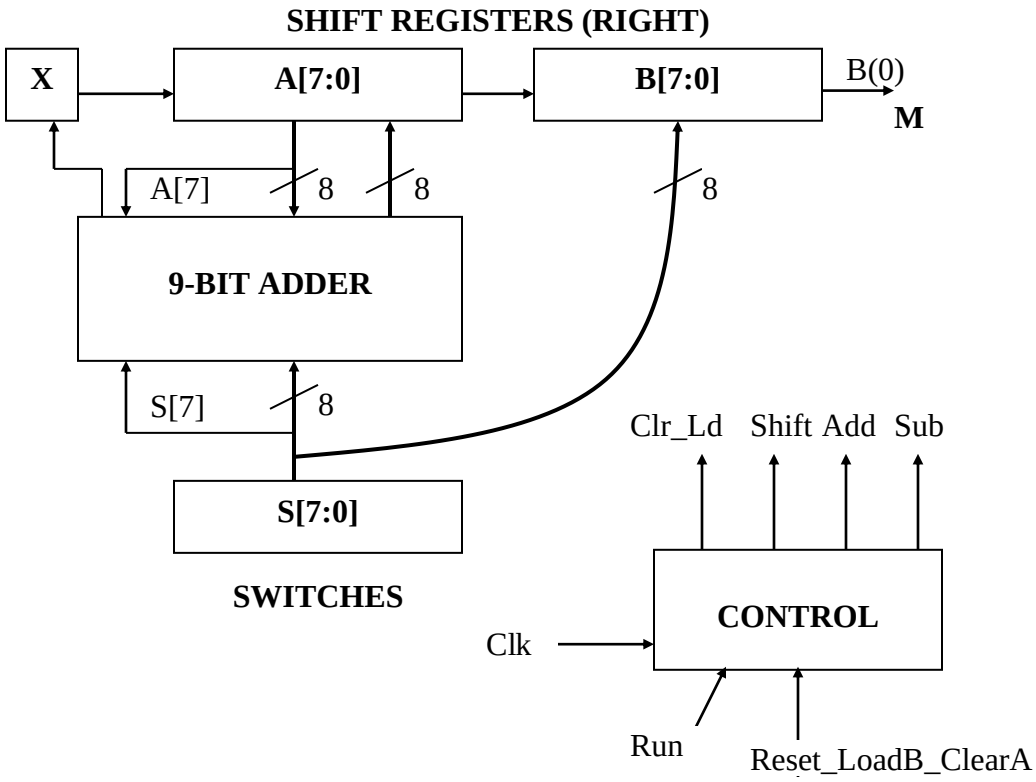

Figure 1: Incomplete Block Diagram

Your top-level module should have the following inputs and outputs:

Inputs

SW – logic [7:0]

Clk, Reset_Load_Clear, Run– logic

Outputs

hex_grid - logic [3:0]

hex_seg – logic [7:0]

Aval, Bval – logic [7:0]

Xval –logic

4.4

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 clear 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 the 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 (consecutive) 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.

III. PRE-LAB

Design, document, implement, and test the 8-bit multiplier in SystemVerilog. Submit your code prior to your demo according to the procedures on the course website.

LAB

Follow the Lab 4 demo information on the course website. As usual, the pin assignments are provided below, but feel free to reuse the XDC file from previous labs, as this lab uses the same peripherals (LEDs and switches).

Pin Assignment Table

Port Name	IO Standard	Location	Comments
SW[0]	LVCMOS25	G1	On-board slider switch (SW0)
SW[1]	LVCMOS25	F2	On-board slider switch (SW1)
SW[2]	LVCMOS25	F1	On-board slider switch (SW2)
SW[3]	LVCMOS25	E2	On-board slider switch (SW3)
SW[4]	LVCMOS25	E1	On-board slider switch (SW4)
SW[5]	LVCMOS25	D2	On-board slider switch (SW5)
SW[6]	LVCMOS25	D1	On-board slider switch (SW6)
SW[7]	LVCMOS25	C2	On-board slider switch (SW7)
Bval[0]	LVCMOS33	C13	On-Board LED (LED0)
Bval[1]	LVCMOS33	C14	On-Board LED (LED1)
Bval[2]	LVCMOS33	D14	On-Board LED (LED2)
Bval[3]	LVCMOS33	D15	On-Board LED (LED3)
Bval[4]	LVCMOS33	D16	On-Board LED (LED4)
Bval[5]	LVCMOS33	F18	On-Board LED (LED5)

Bval[6]	LVCMOS33	E17	On-Board LED (LED6)
Bval[7]	LVCMOS33	D17	On-Board LED (LED7)
Aval[0]	LVCMOS33	C17	On-Board LED (LED8)
Aval[1]	LVCMOS33	B18	On-Board LED (LED9)
Aval[2]	LVCMOS33	A17	On-Board LED (LED10)
Aval[3]	LVCMOS33	B17	On-Board LED (LED11)
Aval[4]	LVCMOS33	C18	On-Board LED (LED12)
Aval[5]	LVCMOS33	D18	On-Board LED (LED13)
Aval[6]	LVCMOS33	E18	On-Board LED (LED14)
Aval[7]	LVCMOS33	G17	On-Board LED (LED15)
Xval	LVCMOS33	C9	On-Board RGB red (RGB0_R)
hex_gridA[0]	LVCMOS25	G6	On-Board eight-segment display grid
hex_gridA[1]	LVCMOS25	H6	On-Board eight-segment display grid
hex_gridA[2]	LVCMOS25	C3	On-Board eight-segment display grid
hex_gridA[3]	LVCMOS25	B3	On-Board eight-segment display grid
hex_segA[0]	LVCMOS25	E6	On-Board eight-segment display segment
hex_segA[1]	LVCMOS25	B4	On-Board eight-segment display segment
hex_segA[2]	LVCMOS25	D5	On-Board eight-segment display segment
hex_segA[3]	LVCMOS25	C5	On-Board eight-segment display segment
hex_segA[4]	LVCMOS25	D7	On-Board eight-segment display segment
hex_segA[5]	LVCMOS25	D6	On-Board eight-segment display segment
hex_segA[6]	LVCMOS25	C4	On-Board eight-segment display segment
hex_segA[7]	LVCMOS25	B5	On-Board eight-segment display segment
Clk	LVCMOS33	N15	50 MHz Clock from the on-board oscillators
Reset_Load_Clear	LVCMOS25	J2	On-Board Push Button (BTN0)
Run	LVCMOS25	J1	On-Board Push Button (BTN1)

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.

Demo Points Breakdown:

Refer to the demo point listing on the course webpage as this may change from semester to semester.

V. POST-LAB

1.) Refer to the Design Resources and Statistics in IVT and complete the following design statistics table.

LUT	
DSP	
Memory (BRAM)	

Flip-Flop	
Latches*	
Frequency	
Static Power	
Dynamic Power	
Total Power	

*The number of latches should be 0 in a fully synchronous FPGA design. Note that Vivado will create latches if your `always_comb` procedures do not synthesize into purely combinational logic. This indicates a **bug in your design** and should be fixed. Your TA will verify that your design has no latches during the demo.

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 would happen if you used the carry out of an 8-bit adder instead of output of a 9-bit adder for X?
- 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

   Summarize the basic functionality of the multiplier circuit.
2. Pre-lab question

Rework the multiplication example on page 4.2 of the lab manual, as in compute 11000101 * 00000111 in a table like the example. Note that the order of the multiplicand and multiplier are reversed from the example.

3. Written description and diagrams of multiplier circuit
    a. Summary of operation

        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 (elaborated design) viewer.
        ii. Please only include the top-level diagram and not the RTL view of every module.
    c. Written Description of SystemVerilog Modules
        i. List all modules used in the format shown in the appendix of the Lab 2.2 document.
        ii. Descriptions for modules which you have already written may be copy/pasted from (your own) previous lab reports.
        iii. You may insert expanded RTL diagrams of each individual module here if it is legible and useful.
    d. State Diagram for Control Unit
        i. This should be done using an external drawing tool.
        ii. Visio, Inkscape, and Draw.io are the most used for state diagrams.
        iii. Hand drawn (or diagrams inked on a tablet) are not acceptable.
4. Annotated pre-lab simulation waveforms.
    a. Must show 4 operations where operands have signs (+*+), (+*-), (-*+) and (-*-)
    b. Waveform must have annotations (labels) that clearly show the operands as well as the result, etc.
    c. This can be done with a single test bench, or with a separate test bench for each scenario.
5. Answers to post-lab questions
    a. Fill in the table shown on page 4.5 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 did not 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 does not get changed.