Digital Logic
CSC 244L
Laboratory 7
Unsigned Multiplication and Arithmetic Logic Units

1 Objectives

1. Investigate a more complex arithmetic circuit;
2. Understand, design, and verify a SystemVerilog array multiplier to perform unsigned multiplication using partial products;
3. Build an arithmetic logic unit (ALU) that can perform four different operations; and
4. Design a sequential SV controller to manage shared resources ("bus") using staged operations with intermediate registers.

2 Pre-Laboratory Procedure

Illegible work and files that do not open in the D2L dropbox will result in reduced grades. It is your responsibility to ensure that what you turn in is readable by the TAs. Please read the submission instructions and follow them to ensure you receive all points. The circuit diagrams for pre-lab may be *neatly* hand drawn. To be completed *before* your lab meets (individually):

2.1 Read the entire lab procedure and Chapter 5.2 in your book (ALU, Multiplication) in your book;

2.2 The following will be turned in, *as a group*, to a D2L dropbox in your lab section before 2 p.m. on your lab day the week of Oct. 31. Complete these items (in this lab manual) for your pre-lab:

- 3.1, 3.2, 3.3, 3.4
- 4.3, 4.4

2.3 Bring soft copies of all your SV modules to lab to be compiled and loaded onto the FPGA for testing and demonstration.

For your SV, you must use **separate modules** with **one .txt file per Verilog module** for the various logic functions and memory elements. **Do not .zip your pre-lab files, and do not turn in your SV in '.sv' file format.** You must turn these in as a '.txt' file to receive pre-lab credit, '.sv' files are not readable in D2L.

By now you should have downloaded and installed Quartus Prime Lite. You should come to lab with **Verilog modules that compile the very first time**. You will ensure that your files compile correctly by compiling them at home with Quartus. A proactive student would also test that the compiled SV works on their DE10-Lite board.

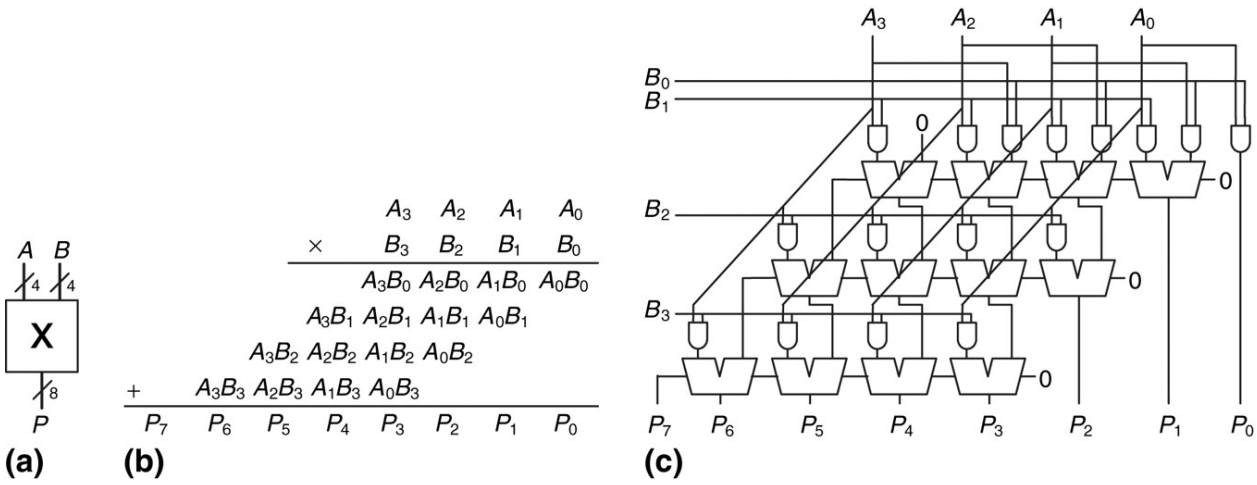

Figure 1: (a) Schematic of a 4-bit multiplier that implements $A \times B = P$. (b) A representation of the partial-product and sum method of unsigned binary multiplication. (c) A 4-bit array multiplier for unsigned binary numbers using partial products (AND gates) and cascaded full-adders.

3 4-Bit Unsigned Array Multiplier Procedure

The goal of this section is to design a 4-bit unsigned integer multiplication circuit in SystemVerilog. The multiplier circuit will take two 4-bit inputs (the multiplicand, A, and multiplier, B) and return an 8-bit output (the product, P). The inputs, A and B, will each be implemented using four switches. The 8-bit product, P, will be displayed on two 7-segment displays in hexadecimal.

3.1 As part of pre-lab, complete the "pre-lab" column of Table 1.

3.2 Create a SV file named "partialproduct4.sv" that contains one SV `module` named PP4. Using any SV you wish (e.g., behavioral, procedural), create a 4-bit partial-product calculator using your 4-bit adder from Lab 7 and connecting it as in Fig. 2. Alternatively, you can make a single multiplication `module` "cell" (from Lecture 25) and connect four of them together using structural SV to create the same component in Fig. 2.

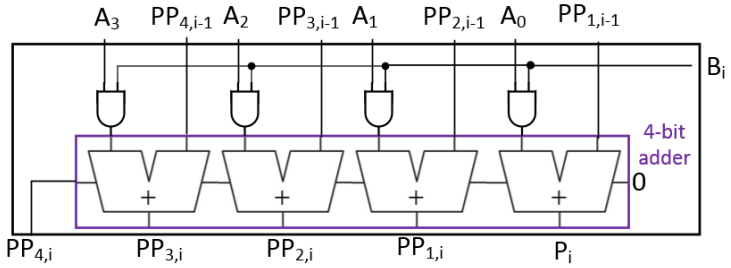

Figure 2: A 4-bit partial-product calculator for row i of the array multiplier. The partial product is calculated using the multiplicand A, bit i of the multiplier B_i, and the partial product output of the previous row $\text{PP}_{4:1,i-1}$. The output is the next partial product $\text{PP}_{4:1,i}$, and the product bit P_i.

3.3 Create a second SV file named "mult4.sv" that contains one SV `module` named mult4. Using ***structural*** SV, connect four PP4 modules together to make a 4-bit array multiplier as shown in Fig. 3

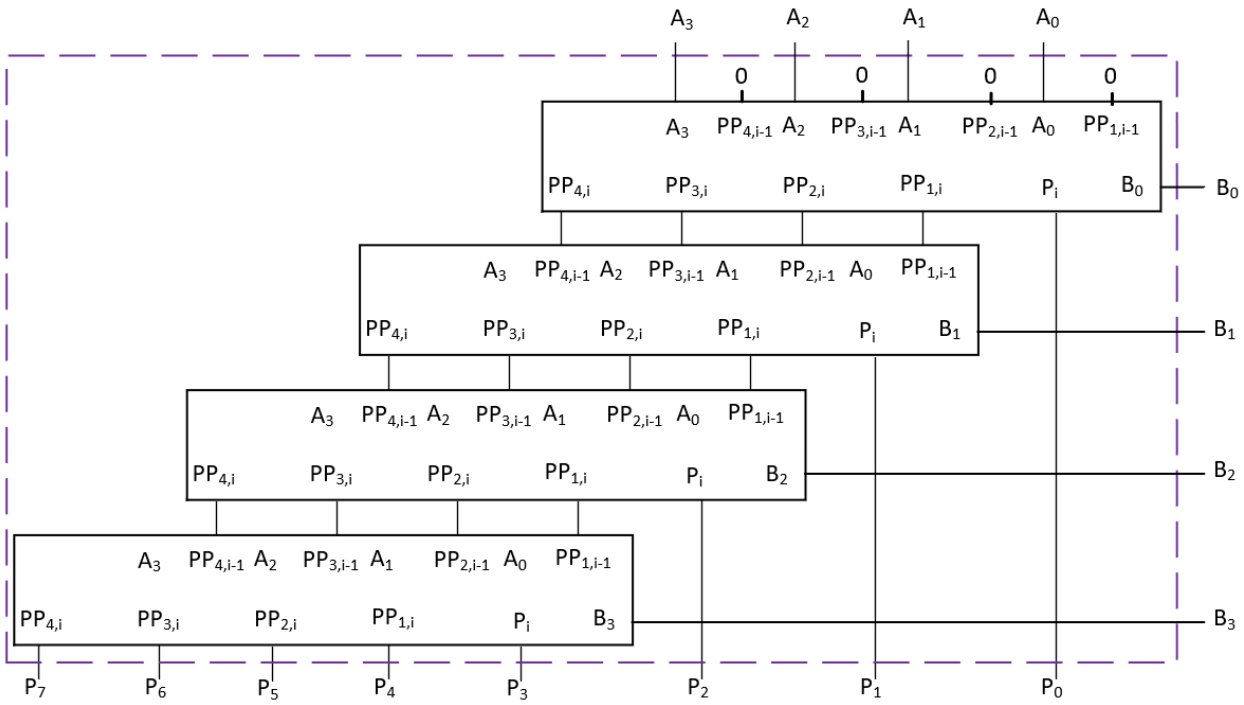

Figure 3: A 4-bit array multiplier that calculates $A \times B = P$, where A and B are 4-bits wide and P is 8-bits.

3.4 Make a top-level SV file that connects the mult4 module. Assign SW[7:4] as your A input and SW[3:0] as your B input. Connect A, B, and P to 7-segment displays using a hex decoder (**note:** you should already have one of these from a prior laboratory experiment. If not, you need to create a 4-input combinational circuit that outputs 0-F on an active-low 7-segment display for the inputs $0000_2 - 1111_2$). You will need one hex digit for each of your A and B inputs, and two digits for your P output.

3.5 Load your top-level SV module to your FPGA. Verify the operation of your SV module by checking that the outputs match your pre-lab in Table 1. If they do not, either debug your SV, recalculate your pre-lab, or both.

3.6 Show your working FPGA implementation and table to your TA, and have them sign off on your lab sheet.

Table 1: Unsigned multiplication

A			B			Pre-lab Product A×B			FPGA Product A×B		
decimal	binary	hex	decimal	binary	hex	decimal	binary	hex	decimal	binary	hex
1	0001	1	10	1010	A	10	1010	A	10	1010	A
3	0011	3	15	1111	F	45	00101101	2D	45	00101101	2D
6	0110	6	11	1011	B	66	01000010	42	66	01000010	42
13	1101	D	5	0101	5	65	01000001	41	65	01000001	41
9	1001	9	2	0010	2	18	00010010	12	18	00010010	12
15	1111	F	15	1111	F	225	11100001	E1	225	11100001	E1

4 Arithmetic Logic Unit (ALU) Procedure

The goal of this section is to design an ALU, and add some additional hardware to share physical resources by controlling access in multiple stages using a sequential controller. Dr. Hansen will provide you the top-level SV file and module headers for the project. The ALU will use a 2-bit control signal, ALUControl, to select from one of four operations, given in Table 2.

Table 2: ALU Operations

ALUControl	**Operation**
2'b00	ADD
2'b01	SUB
2'b10	AND
2'b11	OR

4.1 Download the provided SV files from the D2L Laboratory 7 page.

4.2 The top-level SV file and module ("ALUcontroller.sv") has been completed for you by Dr. Hansen (except the 7-segment connections), implementing the hardware shown in Fig. 4. Not pictured are the clock debouncer (CLKb to CLKb_deb), controller (to determine active-high register enables), and the 7-segment hex decoders. You will need to replace the placeholder 7-segment structural SV in the `ALUcontroller` module with your 7-segment module definition.

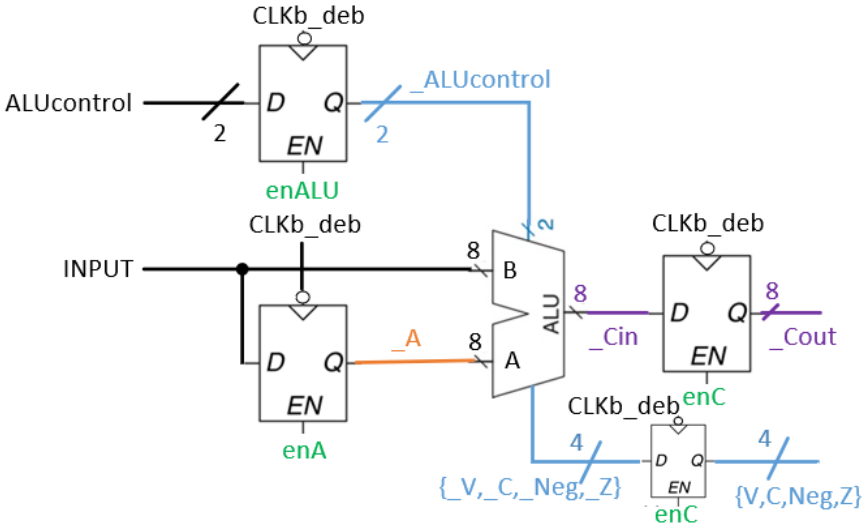

Figure 4: The top-level schematic for the ALU controller. The INPUT bus is shared between the A and B inputs to the ALU, with the A and ALUcontrol inputs buffered by a register. A `controller` module (not pictured) is used to determine the active-high register enables enA, enC, and enALU (green) to control the operation of the negative-edge triggered registers (triggered by a debounced clock signal, CLKb_deb).

4.3 Complete the `ALU` module in "ALU.sv" using any SV you wish. The ALU takes two 8-bit inputs, A and B, and determines the 8-bit Result based on the 2-bit ALUControl signal. The ALUControl bits are described in Table 2. You may implement this combinational circuit

however you wish, including using the built-in SV operators (i.e., '+', '−', '&', '|'). It is recommended you *paramaterize* your module by defining human-readable constants for each operation. Additionally, you should have four ALU status flags that indicate whether the result had overflow, had a carry, was negative, or was zero (V,C,N,Z, respectively). Use the logic from Fig. 5.17 in your book for these flags.

4.4 Complete the `controller` module in "controller.sv" using any SV you wish. This sequential controller takes as an input the debounced clock signal to *count* the operation step. Based on the current step, a combinational logic circuit determines the enA, enC, and enALU control signals. The required steps are:

a. save the INPUT to register A and save the ALUcontrol bits

b. save the result of A op B to the C register, and V,C,Neg,Z to the ALU status register (where A and op were the values saved from step a.), and B is the current value of the INPUT

It is recommended you re-watch Lecture 26 if this process does not make sense to you.

4.5 Assign the top-level I/O to the DE10-Lite board as described in Table 3. Note that the 7-segment outputs (Aseg, Bseg, and Cseg) are 2×7 two-dimensional signals. The first index chooses the HEX digit (0 or 1), and the second index chooses the segment (0 – 6, relating to the 7 segment display a–g).

Table 3: Required DE10-Lite Input/Output

Signal	**DE10-Lite I/O**
INPUT	SW[7:0]
ALUcontrol	SW[9:8]
CLKb	KEY0
CLK50M	50 MHz clock
Aseg[1][0:6]	HEX5
Aseg[0][0:6]	HEX4
Bseg[1][0:6]	HEX3
Bseg[0][0:6]	HEX2
Cseg[1][0:6]	HEX1
Cseg[0][0:6]	HEX0
V,C,Neg,Z	LEDR[3:0]

4.6 Load your top-level SV module to your FPGA. Verify the operation of your SV module by checking that the A register is updated on the first clock pulse (push the button), and the C register is updated as A op INPUT in the second clock pulse. **HINT:** you may wish to set the enable signals and the _ALUcontrol signal as LEDR outputs while debugging. The debugging procedure will be similar to that from Project 1. Describe your debugging process to your TA.

4.7 Show your working FPGA implementation and your completed, commented SV to your TA, and have them sign off on your lab sheet.

Laboratory 7 Signoff Sheet

TA Signature	Section
	3.6
	4.7