CSCE 230L – Lab 7: Arithmetic and Logic Unit (ALU)

February 26, 2013

Objectives

- Make a 16-bit full adder
- Add other components and include the full adder to create a basic ALU that can be used in the project. The components are on the last page.

Useful References

Section 9.1 and Appendix A in the textbook

What to Turn In

- Pre-lab questions (due at the beginning of lab; in paper)
  - All questions total to 20 points
- Files and questions from lab (submit on handin)
  - Archive of Quartus project (.qar)
  - Simulation results
    - Full Adder
    - ALU

Answer all the questions in a single text file. For the project files, create a Quartus archive file (.qar). Simulation results need to be submitted separately, the archive file does not include them.

Name (1 point):

Pre-Lab (19 points)

1. When adding two binary numbers, we know that for each bit, the sum and carry out can be determined with the following expressions:

   $S_i = A_i \oplus B_i \oplus C_{in}$ $C_{out} = A_iB_i + A_iC_{in} + B_iC_{in}$

   Draw the gate diagram for both S_i and C_{out} using gates available in Quartus (i.e. don't use a 3-input XOR, only 2-input XOR is acceptable).

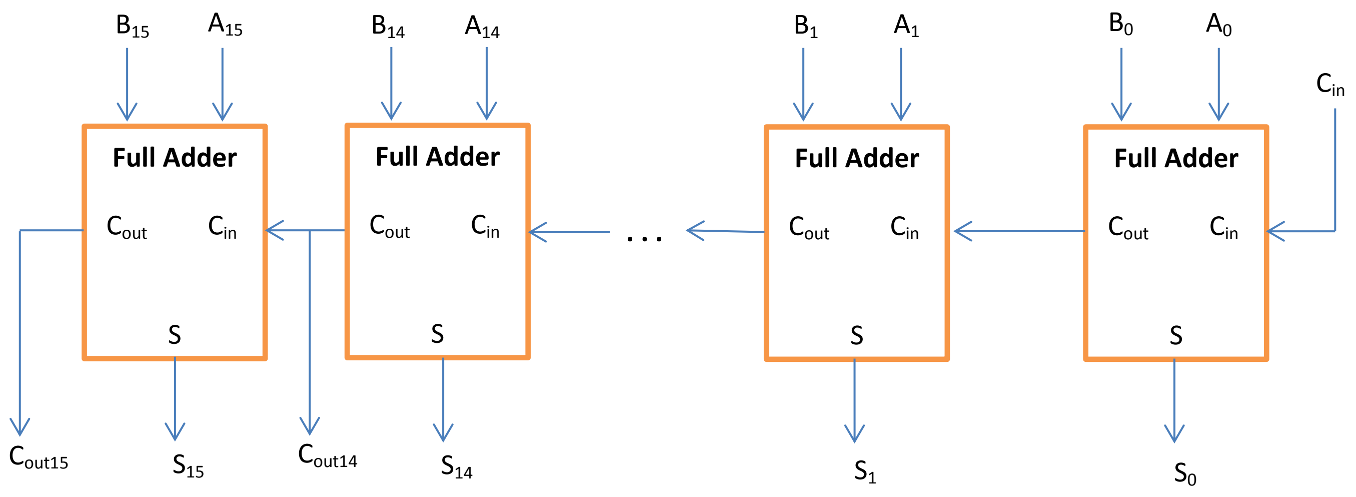
2. Question 1 is what makes up a 1-bit full adder (FA). Now given inputs A[3..0], B[3..0], and C_{in}, use multiple full adders to create a 4-bit adder. The outputs should be S[3..0] and C_{out}.

3. What should C_{in} be in each of the following cases?
   a. Computing A + B.
   b. Computing A – B, or in the computers view: A + (-B).

Lab

Part 1

This part should be straight forward. We want to create a 16-bit full adder for the ALU. To do this in incremental steps, first create a 1-bit FA, with inputs A, B, and C_{in}, and outputs S and C_{out}. Basically, draw in Quartus what was done in pre-lab question 1. Once completed, create a symbol file for it, and then use that to make a 16-bit FA. The following picture gives the general idea. Once the 16-bit adder is implemented, test a few cases with a simulation to give you confidence in the correctness of your design.



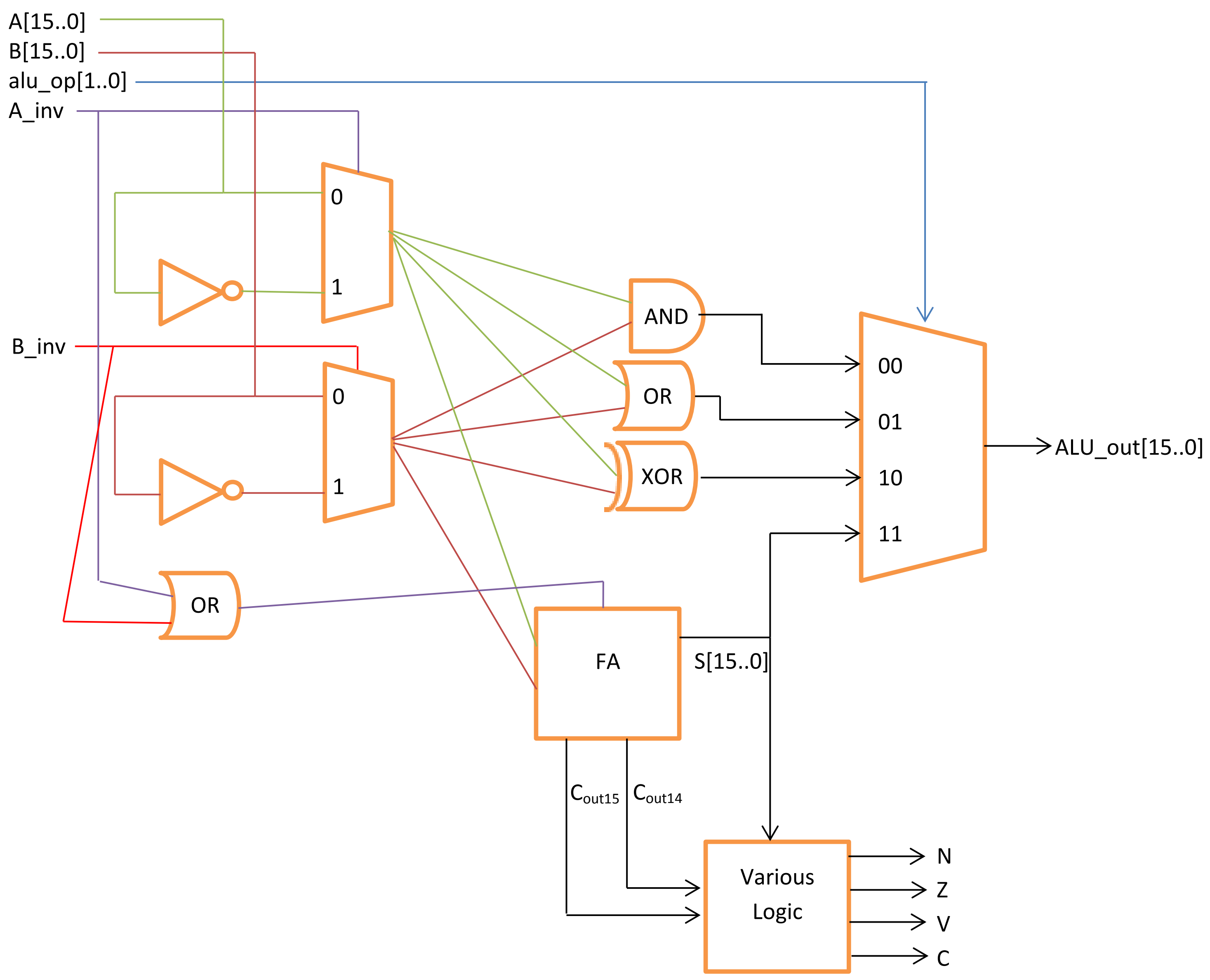
Part 2

With the adder implemented, the most complicated component of our ALU is finished. The other functionalities required for the project are AND, OR, and XOR. All of these are always computed and then a multiplexer is used to select the proper output. In addition to the output, the ALU needs to determine the status flags N (negative), Z (zero), V (overflow), C (carry). N is 1 when the output is negative, so when the sign bit is 1; Z is 1 when the output is all 0's; V is 1 when overflow has occurred which is when the sign bit of the output does not equal carry out; and lastly, C is simply the carry out value of the addition/subtraction. To summarize:

- N (negative) = S[15]
- Z (zero) corresponds to S[i] = 0 for i = 0, …, 15 (Figure out the Boolean function for this)
- V (overflow) = $C_{out15} \oplus C_{out14}$
- C (carry) = C_{out15}

Using the following picture as a guide, construct the ALU with the following inputs and outputs. Once finished, test the ALU with a functional simulation.

Inputs	**Outputs**
A[15..0]	ALU_out[15..0]
B[15..0]	N
A_inv, B_inv(will decide ADD or SUB)	Z
alu_op[1..0](will decide an operation of	V
AND, OR, XOR or ADD/SUB)	C



A[15..0]
B[15..0]
alu_op[1..0]
A_inv
B_inv
0
1
0
1
OR
AND
OR
XOR
FA
S[15..0]
Cout15
Cout14
00
01
10
11
ALU_out[15..0]
Various Logic
N
Z
V
C