14:332:493:02 – Fall 2017
Written By: Gregory Leonberg

Lab 2: The only time you have to do math

Introduction:

In this experiment, we are going to create a simple 4-bit Arithmetic and Logic Unit (ALU). First, a single bit full adder will be designed using basic Boolean logic. Then, a 4-bit ripple-carry adder will be created structurally using the single bit full adder. Finally, the power of operators will be leveraged in order to behaviorally describe a 16 function ALU.

Pre Lab – Operators and Conversions Review

Write the logic equations for a single bit full adder with inputs A, B, Cin, and outputs Y, Cout.

Draw a black box diagram for the single bit full adder described above.

Draw a block diagram of a 4-bit ripple-carry adder made of single bit full adders, with 4 bit inputs A and B, a single bit input Cin, a 4-bit output S, and a single bit output Cout.

Part 1 – Back to Digital Logic Design

Background:

A single bit-full adder is a simple circuit that can be used to perform binary arithmetic. By creating a truth table and using a Karnaugh map to minimize the logic, a gate level description can be made to represent the dataflow.

Shown below is a gate level diagram for a single-bit full adder:

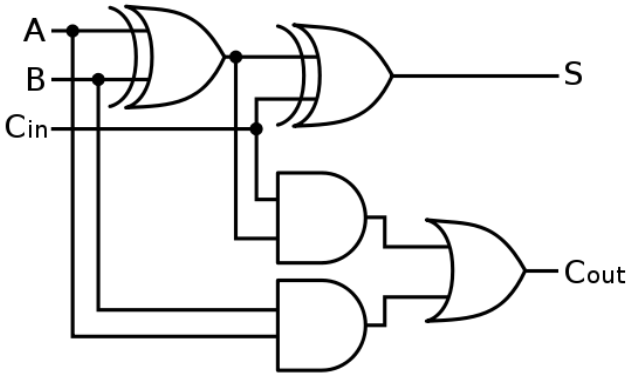

Figure 1: Single-Bit Full Adder Gate Diagram

Task(s):

- Use the gate level schematic provided above to create a single-bit full adder called "adder".
- Design an entity for the 4-bit ripple carry adder described above called "ripple_adder" by using a structural design methodology. Do so by placing and connecting 4 single-bit full adders based on the block diagram given below.
- Create a basic simulation testbench for the 4-bit ripple carry adder to prove its operation

Note: This part is NOT being put on the board

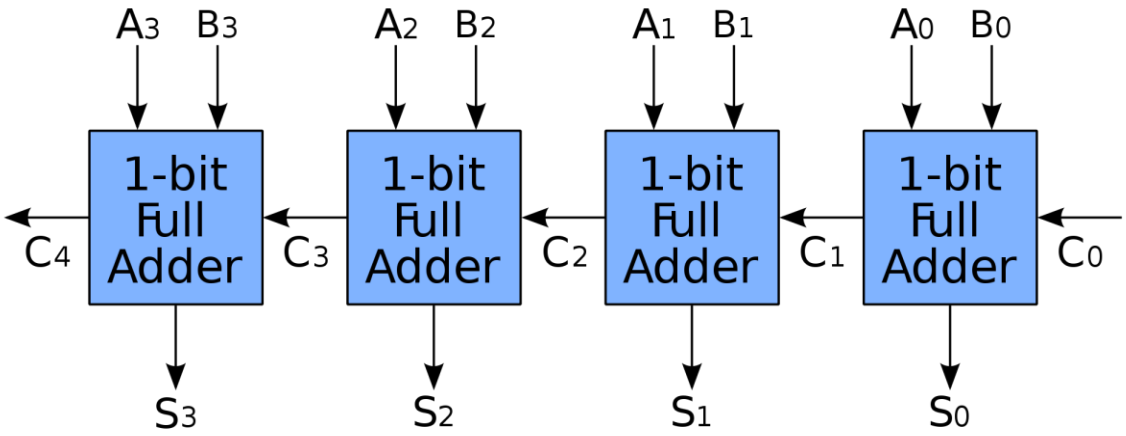

Figure 2: 4-bit Ripple Carry Adder Block Diagram

Part 2 –Somebody did the Work Already

Background:

Due to the power of VHDL and the operators defined in the numeric_std library, many of the basic Arithmetic and Logical functions have already been defined and are synthesize-able without having to create your own hardware implementations. For this part, we are going to leverage that in order to create a 4-bit, 16 function ALU with the following operations:

Opcode	Function
x"0"	A + B
x"1"	A - B
x"2"	A + 1
x"3"	A – 1
x"4"	0 – A
x"5"	A << 1 (shift left logical)
x"6"	A >> 1 (shift right logical)
x"7"	A >>> 1 (shift right arithmetic)
x"9"	A and B

x“A”	A or B
x“B”	A xor B
x“B”	A < B (signed)
x“C”	A > B (signed)
x“D”	A = B (as bit 0 of output)
x“E”	A < B (as bit 0 of output)
x“F”	A > B (as bit 0 of output)

Figure 3: ALU Opcode Table

Note: Many off these functions require certain types for their arguments, be careful with your type conversions.

Note: Shifting can be implemented with std_logic_vectors and some indexing tricks (think concatenation)

Hint: It may be worthwhile to look into the “case” structure.

Hint: see this https://www.doulos.com/knowhow/vhdl_designers_guide/numeric_std/

Task(s):

- Design the 16 function 4-bit ALU, called “myALU”, described above behaviorally by taking advantage of the numeric_std library and its associated operators and conversion functions.

- Create a top level design, called “ALUtester”, in which you instantiate the ALU and assign its output to the four LEDs on the Zybo board. Assign its three inputs Opcode, A, and B to three different 4-bit std_logic_vectors whose values are loaded in from the switches when buttons 2, 1, and 0 respectively are pressed on a rising edge of the clock. Pressing button 3 should clear all 3 signals to 0. The inputs from the buttons will be debounced using the module designed in Lab 1. The output of the ALU will be stored in a 4-bit std_logic_vector on each rising edge of the clock.

- Create the appropriate modified XDC file and put the design on the board in order to test it experimentally.

References:

https://upload.wikimedia.org/wikipedia/commons/thumb/5/5d/4bit_ripple_carry_adder.svg/2000px-4-bit_ripple_carry_adder.svg.png

http://gateoverflow.in/?qa=blob&qa_blobid=5724651430411155887