LAB Assignment #4 for ECE 443

Assigned: Wed., Sept 22, 2010
Due: Mon., Oct. 6, 2010

Description: Implement the Adder component of an ALU

This lab will give you experience designing a combinational component, i.e., an adder/subtractor component of a microprocessor, which uses conditional and selected signal assignment statements. The input/output signals to adder are defined as follows:
Input:
- A, B: input buses (12 bits wide)
- add_ins: Add A and B
- sub_ins: Subtract B from A (A-B)
- skwlf_ins
- skwgf_ins

Output:
- ADD_OUT: output bus (12 bits wide)
- skw_success: Boolean value that depends on instuction and MSB of adder

The input operands, *A* and *B*, to the **AddUnit** are 12 bits wide, as is the output of the AddUnit, labeled ADD_OUT. You need to implement a *ripple carry* adder, the simplest (and smallest) type of adder. The adder can be configured to subtract by XORing the B operand bits and setting the **carry-in** bit of the Adder to 1. You will perform subtraction under one of three conditions, if any of *sub_ins* or *skwlf_ins* or *skwgf_ins* are set to ‘1’. These signals model instructions from a microprocessor/controller
The output signal *skw_success* needs to be set as follows:
If *A* < *B*, then MSB of adder is ‘1’ after subtraction, otherwise it is ‘0’.
If *skwlf_ins* is ‘1’, then *skw_success* should be set to ‘1’ if the MSB of adder is ‘1’, else ‘0’
If *skwgf_ins* is ‘1’, then *skw_success* should be set to ‘1’ if the MSB of adder is ‘0’ AND the low order 11 bits are NOT all 0s, else ‘0’

(NOTE: The *add_ins* input does NOT connect to anything (at this point) but please leave it in the port list. Also, leave the *carry_out* signal of the high order 1-bit adder unconnected)

Be sure to use hierarchy in the implementation of the 12 bit adder, i.e., component instantiation. For example, the top-level of the ripple carry adder should be implemented using 12 full adders. The logic for a full adder can be obtained from any basic digital logic text (or on the web)

You should download the starter VHDL code that I’ve put on-line and run it first without modifying it. Make note of the WARNING messages you get during the synthesis so that when you later modify it to include your code, you will know which ones to ignore. You should observe the following behavior.

A) Set hyperterminal set to 38400, N81 and connect your USB to serial cable from your computer to the FPGA board.
B) Upload the synthesized design to the FPGA.

C) Reset the board after the upload by moving 'switch 3' (left most switch) from the up position to the down position and then back to the up position -- leave it in the up position.
D) Press the 'enter' pushbutton -- you should see "+0000" printed on the screen each time you press it. If you don't something is wrong with your serial connection.
E) To enter values for the two operands, do the following: Type up to 4 digits (value MUST be in range of -2048 to 2047) and hit 'enter' on the keyboard, type up to 4 more digits and hit enter again. If you then press the 'enter' pushbutton on the FPGA, the sum will be printed to the screen (this is the unmodified default behavior). If you enter another (third) number and press 'enter' on the keyboard, this new value over-writes the first value you entered. Each time you enter a value, it over-writes one of the values in the pair in a circular fashion.

Once you have tried this and confirmed that it works, you need to modify the main VHDL module (UARTNumberConvert.vhd). First create an instance of your ALU. Then find the line that says '-- ADD YOUR ADDER HERE'. Connect your input operands A and B to the signals 'A_op_reg' and 'B_op_reg' and your output to 'result_binary_val'. Connect your xxx_ins signals as shown below. Connect your skw_success signal to one of the LEDs (you can remove any assignments that I have made to the LEDs).

add_ins: NO buttons pressed -- just press Enter
sub_ins: up_pb_level
skwlf_ins: down_pb_level
skwgf_ins: left_pb_level

You need to comment out the statement below that line with the comment -- TEMPORARY that assigns to result_binary_val after you have added the AddComponent

Bear in mind that my code does not check for all possible error conditions. For example, if you enter a value smaller than -2048 or larger the 2047, your results will be difficult to interpret because of the overflow that occurred. Also, I do not enforce that you type only 4 characters before hitting enter on the keyboard. If you start noticing strange behavior, use the reset as described above. Report any bugs to me as soon as possible.

Laboratory Report Requirements:
1) Turn in a commented copy of your VHDL code.
2) Turn in the top-level schematic diagram that represents the synthesized schematic of the AddUnit component of your code.

Your lab grade will consist of two parts. The first part is associated with the in-class demo, and is worth 50% of the total grade (50 pts). Successful demonstration of the lab's stated requirements is worth 50 pts. Partial implementations will be given only partial credit. The second portion of the lab grade is derived from your lab report, derived according to the posted guidelines for preparing laboratory reports.