LAB Assignment #4 for CMPE 415

Assigned: Fri., Nov 8th, 2007
Due: Thur., Nov 15th, 2007

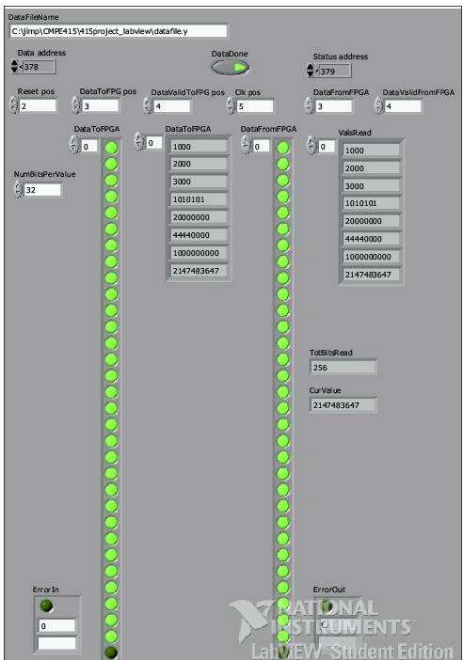
Description: Revise the LABVIEW and Verilog code from Labs #2 and #3 to read an entire file (LABVIEW) of 32 bit integers, transfer them to the FPGA and back.

This lab builds on the previous two labs and completes the basis work for the project. The Verilog code you have written for LAB #2 reads 8-bits. The first revision you'll need to make is to expand this to 32-bits. Once you have this working, you'll then create a memory CORE, using the core generator tool with ISE 9.2i. I will go over this in class on Nov. 8th after the demo. You should set the size of the memory to be 32-bits by 128 words (note: we'll later modify this to be larger). You will generate a single port memory which has 5 inputs, *addr* (address), *din* (data in), *we* (write enable), *clk* (clock) and *dout* (data out). It will operate synchronously with the clock, i.e., to write a value, set the value of *di* and *addr* and enable *we*. If you've configured your memory core to be sensitize to the rising edge of clk, then on the next rising edge, the values you've placed on *din* will be written to memory at the address given by *addr.* The written value will also simultaneously show up on the output of the memory, *dout*. To read a word from memory, set the *addr* and strobe the clock. It's that simple.

You also need to modify the LABVIEW code to read a file of integers, one per line and write each of the values as a sequence of bits to the parallel port, 1 bit at a time, as you've done for LAB #3. You'll also need to modify the read back portion to read a set of 32-bit integers. You should depend only on the **DataValidFromFPGA** to determine when the last value has been transferred from the FPGA. Although in this exercise you are simply transferring the data back and forth, for the project, you will process the data once it is loaded into the memory on the FPGA. The processing operation (to be decided) is likely to change the number of values that you will need to send back to LABVIEW. Therefore, you will need to use the **DataValidFromFPGA** signal from the FPGA to determine when the transfer is complete.

In my Verilog code, I added a fifth state to handle a memory address issue in the transition from reading data (state1) and writing it back (state4). State0 still the initialization state and state1 the read data state. I now have state2 and state3 to create two cycles of delay before reaching state4, which transfers data back. You may not need to add a state, and I may end up dropping this extra state once I add the code that does the processing since presumably this will take multiple clock cycles to complete. See my timing diagram for more information.

The front panel of my VI is shown below. It is nearly identical to the front panel from LAB #3.



Difference include a filename widget to allow the user to enter a path name to the data file. The data in and out arrays have been expanded to 32-bits and I've created arrays for the integer representations of the data.

YOU MUST USE THE SAME transparport.ucf and transparport.xcf files that I provided for LAB#3.

The Core Generator:
In order to create a memory on the FPGA, you will use the core generator application. Here are the steps to follow:

1) Under 'programs/ISE 9.2i/accessories', click on 'CORE Generator'
2) Select 'new project' under the file menu
3) Enter a name and directory
4) In the CGP form, select Spartian 2, xc2s200, fg200, -6. Under the Gen Tab, select 'Verilog' and ISE for vendor. Click Ok.
5) Expand Basic Elements in the list on left in CORE Generator, then Memory Elements
6) Select Single Port Block
7) Click Customize, type 32 for Width and 128 for Depth, click next, click next again
8) Choose the polarity for the Clk, write enable pins, etc.
9) Click Generate

Once the CORE Generator finishes, move the files to your FSM project directory, e.g., transtofromFPGA, or whatever you called it. The filename with extension '.veo' has a template for the module that you can cut-and-paste into your verilog file with the FSM. You'll need to change the names in parenthesis to match those you use in your code. In ISE, Add source to add the file with the '.v' extension. That's it...

Laboratory Report Requirements:

1) Turn in a commented copy of your LABVIEW code.
3) Prepare to demonstrate that both the LABVIEW code and the FPGA can communicate properly.

Each of you MUST work on it **independently** and turn in a separate report, i.e., this is NOT a group project.

Grading:
60% LABVIEW code correctly specifies desired functionality.
10% Meaningful comments in LABVIEW code.
30% Successful hardware demonstration.