ComS / CprE 583 – Reconfigurable Computing
Homework #1
Assigned: August 24
Due: September 7 (12:00pm)

[Note from Joe: questions 1 and 4 relate to topics covered in the first week of class, while questions 2 and 3 regard LUT mapping which will be covered in the second week of class. Questions 1-3 are fairly straightforward, while 4 is considerably more challenging. Although we have not yet covered the details in class, you will be expected to modify some VHDL source and simulate the resulting designs. Think of it more as a puzzle involving design patterns rather than as a programming assignment.]

1) Consider the four classifications of coupling in a reconfigurable system as discussed in class [Hau98A, ComHau02A]. Find an example on the web or from your personal experience of a relatively recent (less than 5 years) commercialized implementation of each.

(a) Provide a brief (1 paragraph) description of each system. What is the architecture of the system? What are the intended applications? A complete datasheet is not necessary; provide just enough detail to give a clear picture about the product.

(b) Describe some of the obvious advantages of each of your selected systems. Describe some relative disadvantages. What generalizations can you make based on these?

2) Show a function of 9-bits that cannot be implemented with the F, G and H LUTs of a CLB in Xilinx 4000. Argue why not. Can you generalize it to a class of functions that cannot be implemented with F, G and H LUTs?

3) Assume that you wanted to implement a function with 100 2-input, 100 3-input, and 100 4-input gates. How many CLBs will you have to use in Xilinx 4000 family FPGA implementations. Estimate the average lengths of wires in terms of column/row units for this mapping (assume all of the 300 gates are equally likely connected to each other). Now consider a situation where 2Kb (2048 bits) of storage is required by your computation. How many CLBs will be needed in this case? How would the wire delays (wire spans in row/column units) compare for this scenario as opposed to the preceding 300 gate version?

4) See attached file *hw1.zip* which contains both a software and a partially-completed hardware implementation of the Bubble Sort algorithm. Specifically, you should find the following files:

- *BubbleSort.c* – an ANSI C implementation of the Bubble Sort algorithm. It should compile and run fine on any POSIX-compliant operating system. Take a look at the code to understand what is going on, but you will not need to modify it.
- *Comparator.vhd* – a VHDL implementation of the less than operator, as discussed in class. Take a look at it if VHDL mystifies you, but you will not need to modify it.
- *BubbleSort.vhd* – a partial implementation of the Bubble Sort algorithm, using the *Comparator* module described above. Similar to the version that was shown in class, this design can only handle inputs of length 8, and also is missing the last few stages of computation necessary to get a completely sorted output. Also of note is that this design assumes that the input bus and output bus can only handle 8 bits per cycle.
- *compile.do* – a Modelsim command macro file that will compile the two VHDL sources for you. Take a look at it to see the standard Modelsim syntax but you will not need to modify it. From within the Modelsim application, the command is "do filename.do" to execute any script.
- *simulate.do* – a second Modelsim command macro file that forces signals and runs simulation to test the initial version of the BubbleSort architecture.

(a) Select any modern PC or workstation that you have access to. Find the physical and technical specifications required to compute the functional density measure of [Deh00A]. Do the same for any commercial FPGA using the same process technology. You may have to make some assumptions about chip area (you can use die size as an approximation to layout area as need be). How do they compare?

(b) Using a similar style to what was shown in class, draw the completed architecture for the 8-input Bubble Sort design. Use this drawing and complete the VHDL implementation of the same. If you do not already have access to it, download a demo of the Modelsim HDL simulator in order to check your design. In a zip file, provide both the completed (and commented) BubbleSort.vhd file along with the vsim.wlf that shows the 8 elements being properly sorted.

(c) The base design both assumed a 100MHz clock (see the period on the iCLK signal) and that only 8-bit data values were being sorted, consistent with the 8-bit data bus provided into the design. Which design (hardware or software) runs faster? How does this relative speed difference compare to the functional density analysis performed in (a)?

(d) Let's relax these assumptions: vary the clock frequency between {50MHz, 75MHz, 100MHz, 125MHz, 150MHz} and the data type sizes between {8-bit, 16-bit, 32-bit, 64-bit}. How do the hardware and software designs now compare? Provide a table or chart detailing your findings. What can you infer from these findings? If you did not modify the software, then using *gcc* you should be able to compile for a larger data type using the –D_sort_u16_ flag, for example, while in the *BubbleSort.vhd* file it "should" be as simple as changing the gDATA_WIDTH VHDL generic, along with having different simulation parameters in *simulate.do*. It is not required to re-run the FPGA simulation for each of these input parameters; you can also solve analytically for the total time.