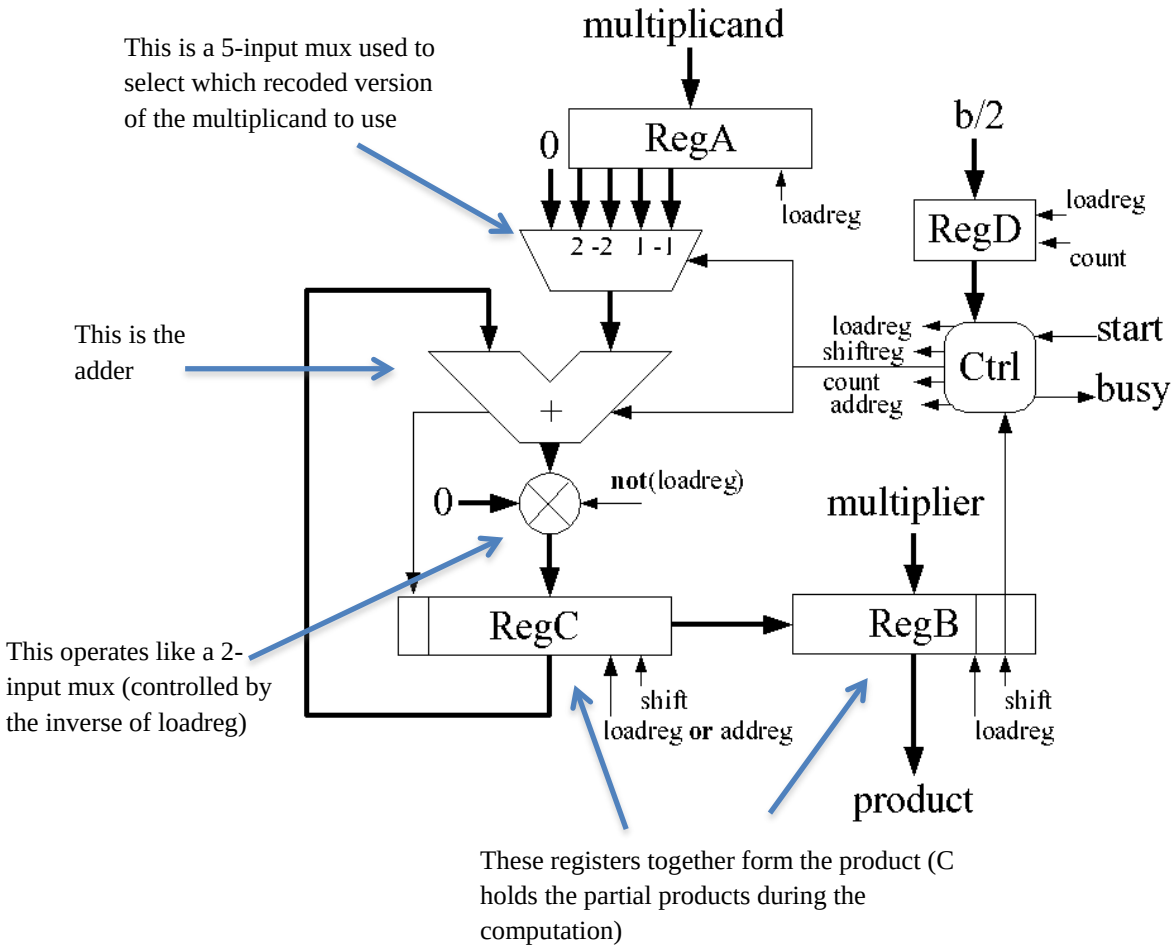
ECE3270 Digital System Design
Lab 4: Bit-pair Recoded Multiplier

Lab Overview: The purpose of this lab is to implement a Bit-pair Recoded fixed point multiplier in VHDL. The top level entity should be a structural design, and all lower level components should be kept completely modular. The multiplier should be tested in ModelSim with appropriate testbench(es) that test for all cases (+/- and varying ranges). You will map your top level entity to an OpenCL project which will call the multiplier multiple times. You will complete a full report using the LaTeX template and include discussion about the clock cycle latency of one multiplication. You will submit all edited files (vhd, xml, cpp, cl) to the assign server as Assignment 4. When submitting your report, be certain this is the **FINAL VERSION** of your report. Multiple submissions **ARE NOT ALLOWED** and will **NO LONGER** be accepted for submitting the wrong version.

Part I

You are to design an 8-bit shift-add multiplier using bit-pair recoding as discussed in class. Each component in the design should be a separate module that you will connect in your top-level structural entity. Part 2 will require that you extend the design to more than 8-bits, so a modular and generic design will be required.
Model your design based on the datapath below. Processing begins when the *start* signal is high and the *busy* signal is low (note, *busy* is set by the state machine, **Ctrl**). The following cycle loads the *multiplicand* and the *multiplier* into their corresponding registers and the *busy* signal becomes high (multiplication is in progress). The machine will complete add/shifts until the product is complete, at which point the *busy* signal becomes low for at least one clock cycle after which the machine can begin again (whenever *start* is high as well).



Part II

You are to extend your bit-pair multiplier from Part I to multiply two 16-bit numbers. Once completing this, you should map your multiplier as a component for use in OpenCL in *bitpair.vhd*. You can expect the multiplicand in the upper 16 bits of *datain* and the multiplier in the lower 16. You will map your output to *dataout*. You will need to map the *ivalid, ovalid,* and *oready* pins to the multiplier. The *ivalid* pin is set high for **exactly** one clock cycle when *datain* is guaranteed to have correct data and you must read the data while *ivalid* is high. The *ovalid* pin should be set high for **exactly** one clock cycle whenever your multiplier is complete. And the *oready* pin should be set high whenever your multiplier is not busy and is ready to receive the next output.

You will need to edit *bitpair_rtl.xml* to modify the latency to match that of your machine and to add all necessary files that your design uses. You will also need to modify the XML file to state that the hardware **will** stall and **will not** necessarily complete in the same number of clock cycles.

This setup is required because of the use of the *ovalid* and *oready* pins applying **backpressure** to the host system. *ovalid* tells the host that the data on the *dataout* line is valid and is **only** valid for one clock cycle. This allows your system to have a flexible latency, if needed. *oready* tells the host that the hardware is ready to accept information on the *datain* line, effectively stalling the host. If you cannot build these into the project easily, you can handle them with some logic statements based on internal bitpair signals to follow the appropriate functionality.

View the README.txt file for proper usage of the bitpair executable. Note that input and output files, if you choose to alter these, are binary files with the following format.

*4 byte int length, 2 byte short * length vec1, 2 byte short * length vec2*

Length is the length of the vectors. The remainder of each file is 2 *length* sized vectors of the **short** datatype. You will be given a few test files, but you are responsible for writing other test cases to fully exercise your multiplier. The output file is set up the same as the input file, but with a *4 byte int * length output* dataset replacing the 2 vectors. You will find a few sample vectors in the *vectors* folder, as well as an executable with source code. Feel free to modify this source to create vectors different than the ones generated by default.

The OpenCL host will print timings for both the kernel execution and the host CPU execution times for computing the product of the vectors. Include a comparison of these timings for different sized vectors that includes graphs displaying the data (vector sizes on the x-axis and timings on the y-axis). Discuss any speedup or lack thereof in the report.