EE486 final project

1 Goal

This project aims to improve on a complete design for an IEEE compatible floating point unit using the single precision numbers.

The unit is part of the open cores project (`http://www.opencores.org`). It is described in high level verilog and uses the verilog operators +,-,*,/,% to achieve the add, subtract, multiply, divide and remainder functions respectively. Obviously such a description can only be rendered to hardware if the synthesis tool used has already some blocks implementing the functions mentioned above.

Your duty is to apply the knowledge that you got in this class to implement those functions in logic gates so that your implementation can be used as the building blocks for the synthesis tool.

2 How to get started

1. Visit this website:

   `http://www.opencores.org/projects/fpu/`

   and download the unit.

2. Unpack the unit and familiarize yourself with it by reading the README file as well as the file FPU.pdf.

3. Your changes will be to the file primitives.v from the verilog subdirectory. This file describes three parts:

   (a) A combinational 27 bits integer adder/subtractor part that is used in the flaoting point adder

   (b) Two stage pipelined 24 bits integer multiplier used in the floating multiplier unit

   (c) Two stage pipelined integer divider taking a 50 bits dividend and a 24 bits divisor to produce a quotiant and a remainder.

4. We want you to implement these three parts in logic gates from the parts.v file that we provide to you. If you need some other logic blocks not in the parts file, build them from the parts that we give and include them in

your new primitives file. Your design should fulfill the following in order of priority:

(a) Correct functionality. This is the most important thing!

(b) High speed

(c) Small area

5. You are not supposed to change the number of inputs or outputs of the `add_sub27`, `mul_r2` and `div_r2` units. Your implementation must fit with the rest of the fpu without changing anything to the remaining files. Your implementation for the multiplier and divider should be also in two pipeline stages. Choose the breaking point carefuly. Notice that, in the divider, if you choose to implement a multiplicative algorithm (to share the multiplier and save on area) you still need to provide the remainder.

6. Final report: In addition to your new primitives.v verilog code, you will need to submit a brief report showing the block diagram, critical path and estimated speed and area for each function. We hope to receive just *one* verilog file and a *short* report.

3 Testing functionality and measuring the performance

You are supposed to write your own testing files to verify the correct functionality of your implementation. You are free to test it using the testing methodology provided with the whole FPU or to use your own. If you are going to test the whole FPU beware of any bugs that might exist in the other files! We will test the submitted files to ensure their functionality. As we said above, correct functionality is the most important thing.

You will be provided with a script that estimates the time and area needed for each block.

4 Collaboration

It is an individual project and you are expected to work on the project alone.

5 Dealines

27 February 2003: One page describing:

1. what kind of an adder you are building
2. if you are using Booth recoding then which type, what kind of an array or tree you are using for the multiplier
3. For the divider, what algorithm are you using.

6 March 2003: Augment the previous page with more details. This should be almost the full report with the exception of the speed and area final estimates. Free testing for functionality for those who submit by this date.

11 March 2003: Last day to submit the report and the project.

13 March 2003: Last day of the class, announcing the results of the project. A token prize for the best design.