A Floating-Point Unit for Numerical Calculations

Jeff Walden– 6.111

5.11.06

Abstract

In this project we will implement a partially IEEE-754 compliant floating-point unit in Verilog. The FPU will implement addition, subtraction, and multiplication, and it will meet the exactness criteria specified by IEEE-754 and set exception flags as specified by IEEE-754. Due to time constraints, it will implement little of the rest of IEEE-754; however, that which it implements should be sufficient for many floating-point needs.

1 Overview

Numbers in computation may be divided into two categories: integers and floating-point numbers. Integers are precise and exactly store mathematical integers purely within the bounds dictated by their width; floating-point numbers store mathematical real numbers but even with unbounded width suffer some imprecision. This project will implement a floating-point unit supporting the basic mathematical operations from which most floating-point operations may be constructed, namely addition and subtraction, multiplication, and (if time permits) division.

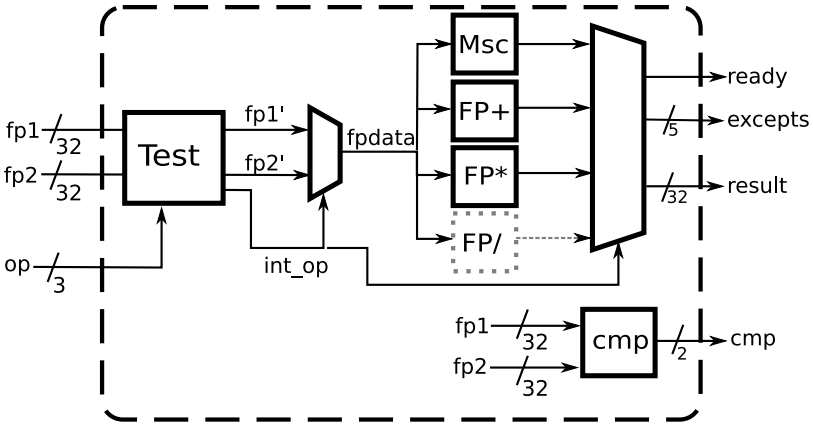
The implementation will aim for limited conformance with the IEEE-754 standard for binary floating-point arithmetic. Rounding-mode functionality will be limited to the round-to-nearest mode, and only single-format numbers will be implemented. No FPU-level support will be provided for determining square roots, calculating remainders, rounding to or from integer values, or converting between bases. However, the implementation will endeavor to support accurate value rounding (within the limits of the single-format number), positive and negative zero and infinity, a NaN (not a number) value, and floating-point exceptions, whenever possible. Deviations from the standard will be clearly noted in all reports.

2 Preliminary Implementation Overview

For simplicity, the initial FPU implementation will consist of a mux which selects among the results of the set of possible calculations. This will be helpful in getting a prototype together against which initial testing can occur. Once

a mostly-working prototype has been constructed, the design will be modified to accommodate faster processing on an operation-by-operation (and for special values on a value-by-value) basis. This should not be difficult to do using some form of tristate device, but we prefer to reason about a working but slow implementation over reasoning about an as-yet-unrealized plan for an implementation.

Figure 1: A high-level diagram of the FPU



2.1 Inputs

The inputs to the FPU will consist of the two floating-point numbers, each stored in 32 bits, and an operation code to indicate what calculation should be performed. The operation code is given in 3 bits to allow for possible functionality increases, although it is unlikely such increases will occur.

2.2 Test

This block converts the external operation code into an internal code (which may or may not be the same). The intent of this block is to combinationally determine when numerical calculation can be avoided entirely (e.g., if either operand is NaN, the entire operation may short-circuit). Its outputs are the aforementioned internal operation code and the two input numbers.

2.3 Output Mux

The output mux selects among the various results for each of the different floating-point operations and determines which one is to be sent to the module outputs. Its intent is to allow exactly one calculation to control the relevant

module outputs at any given time, allowing faster operations to finish before slower operations may have finished. It is not yet entirely clear whether this could be replaced with some form of tristate device, but in the likely case that it is, doing so should allow a slight speedup in processing the faster operations.

2.4 cmp

Comparison of floating-point numbers is relatively simple to implement, and since a comparison requires nothing more than the bit patterns which make up the numbers being compared, it is calculated separately from the main flow of control.

2.5 Outputs

The outputs of the FPU are a ready flag, five exception values, the result of the comparison between the two input floating-point numbers, and the result of the calculation. The ready flag allows for fast returns in the cases where the requested floating-point calculation is simple. The five exceptions are those specified by IEEE-754: invalid operation, divide by zero, overflow, underflow, and inexact. The result of the calculation will be accurate within the limits of the floating-point format; in other words, the result will be the number determined by performing the calculation exactly and using round-to-nearest to dtermine the answer.

3 FPU Operations

The implementation method used in calculation of each floating-point value will be as given in *On the Design of IEEE Compliant Floating Point Units*, Guy Even and Wolfgang Paul, May 2000 IEEE Transactions on Computers. This method reduces each calculation to the following steps: preprocessing each operand to bound precision, performing the operation, and rounding the result in the correct manner while setting exception flags as required.

3.1 Addition/Subtraction

The addition algorithm consists of aligning the floating points (and thus making the exponents equal), performing a sticky-bit computation (to ensure sufficient accuracy so that during final rounding no significant data is lost), adding the numbers (their exponents are equal, making this a relatively simple integral addition), and rounding the final result.

3.2 Multiplication

Multiplication of floating-point numbers consists of multiplying the integer data and adding the exponents; exceptions should be the only complication in implementing multiplication.

4 Additional Details

In addition to implementing the FPU, we will also implement a user interface by which interaction with the FPU is possible. We will allow the floating-point and operation inputs to the FPU to be set, and the result and exceptions will be displayed in some manner for the user to view. At the moment it is unclear what this interface will be, but ideally the interface will consist of a small calculator with input given by a keyboard and with output displayed on a VGA monitor.

Testing of the FPU will occur primarily through Verilog test modules. IEEE-754 verification suites will be used to verify the functionality which the FPU implements, and a test script to run these tests will be executed on a regular basis during development.