Aaron Cohen
Riley Cox
Dan Engler
Nathan Fraly
Final Project Report
ECE 571, Spring 2024

Problem Statement

Design and verify a 32-bit floating point adder in SystemVerilog which accepts inputs and return output in IEEE 754 format.

Interface

The main design module, FloatAdder, is declared as follows:

FloatAdder(Op1, Op2, InputValid, Result, ResultValid, Clock, Reset,outputInvalid,inputInvalid);

The module takes as inputs two operands (Op1, Op2) in the form of 32-bit binary IEEE 754 which can also be passed in the form of the Float type defined as a packed struct in the floating point package. The module also takes a logic signal (InputValid), that lets the adder know that valid operands have been asserted, as well as Clock and Reset signals. The outputs are Result, again in 32-bit IEEE 754 format, and the ResultValid signal, which is asserted when the add operation is complete. Finally, outputInvalid, and inputInvalid are asserted for non-compatible outputs and inputs respectively. The outputInvalid signal is asserted if the output is DeNorm, NaN or Inf. Likewise inputInvalid is similarly asserted for such inputs.

Design

The module will accept inputs on the positive edge of the clock when the ResultValid signal is not asserted. Once the operation is complete, ResultValid is asserted and the module will take new inputs one clock cycle later. The operation should take between 2 and 3 clock cycles to complete; One clock cycle to clock in the inputs, one conditional clock cycle if rounding is needed, and one clock cycle that will round again if needed while outputting the results. Once the inputs are in, they are checked for validity while the exponents are compared using a subtractor. Flags for set if there is no difference and if the result is negative output logic. The sign of the larger exponent is propagated to the rounding module, which handles setting the output. The difference in the exponents is used to determine how far to shift the smaller addend's mantissa to the right. The shifted mantissa is then fed to the mantissa ALU, along with the larger addend's mantissa. The ALU will subtract if the two addends have different signs, otherwise add, with flags for to determine if the result will need to be normalized to the right, if the 2's complement of the result is needed, and if the result of the operation is 0, there is a flag used to determine if the overall output is 0. The output of the mantissa ALU is then complemented if needed, only when the exponents have no difference, the signs are not the

same, and the result is negative. This ensures that even if the larger mantissa wasn't properly selected, the correct sign is applied to the result. This result is then sent to the normalizer circuit, which has two paths; one that shifts the result to the right by 1, and another that shifts it to the left by an amount that will put the implied 1 back in the correct place. If we did not subtract mantissas and there is an overflow from the mantissa ALU, the right shifted mantissa is selected. Otherwise, the left shifted mantissa is selected. The selected mantissa is sent to the rounding circuit, which rounds based on what was shifted out of the smaller addend's mantissa before the mantissa ALU, as well as what is shifted off during normalizing. If rounding is needed, the rounded mantissa is sent back through the normalizing circuit using a multiplexer that selects between the first pass mantissa on the next clock cycle, which then feeds back into the rounding circuitry. Because the rounding module is registered, the rounding module is also responsible for determining if the output is valid, zero, signaling appropriately.

SystemVerilog Features

The design and testbench both employ various SystemVerilog constructs including always_ff, macros (used for debug code), constrained randomization for stimulus generation, coverage for verification, and assertions.

Verification Approach

A test bench was written which utilizes a modular, task-based approach. Tasks were created to generate various kinds of stimuli by constraining the random generation of floating point numbers in different ways. For example, different test cases were constructed to test the addition of any normal, the addition of any normals within a defined range of exponents, and the addition of normals with their additive inverse. The task for each test case is abstracted further with a generic task for applying stimulus and comparing with the output from a known good model. To assist with debugging during the initial phase of development, code was also added to print detailed information on the adder's internal signals at each stage of the processing pipeline. Equivalent code was added to the known good model to compare the output of each stage.

Known Issues

The rounding circuitry currently can cause the mantissa to be off by as much as two. This is due to a misunderstanding of rounding that was implemented and not caught until late in the testing process. These types of errors are detected and ignored in the testbench, but counted in a rudimentary scoreboard. They occur when one or more addends are negative, and rounding is needed. The sign is not properly accounted for in the rounding circuit, and attempts to fix the problem were not successful. In addition, we do not properly detect overflow with our outputValid detection, meaning adding the two largest non-infs that can be added will be treated as a valid output.