EENG 420 - Computer Architecture - Lab 1 Report

Bryan SebaRaj

Professor Abhishek Bhattacharjee

January 31, 2025

Abstract

Simple arithmetic operations are taken for granted in modern computing from the perspective of the common software engineer. However, at the hardware level, these operations are non-trivial to implement. This lab focused on the implementation of a iterative multiplication and division module; Verilog was employed as the HDL to construct the multiplication and divison submodules, which each took in two 32-bit inputs and yielded a 64-bit output to be parsed as the product, quotient, or remainder from its respective operation. The submodules were integrated into a compound module, and the entire system was tested using a common testbench. The results of the simulation were analyzed to determine the cycle count of the system.

After confirming the correctness of the multiplication and division modules, and the compound module, the system was tested with additional test cases to ensure that unsigned remainder and division operations behaved as expected. In addition, edge case behaviors, such as division by zero, were tested to determine any outputs derived.

Design

Starting with the multiplication module, the prescribed datapath was followed. As prescribed, the iterative multiplication and division/remainder submodules were each divided into data and control paths to improve design clarity and enable for more straight forward development. However, when constructed the data path, rather than relying solely

on structural components, such as multiplexers, behavioral Verilog was employed to partially define the flow of data. Coming from the perspective of a software engineer, simple behavior logic most closely followed the simple control flow statements which comprise even the most basic of computer programs.

Regarding the specific implementation of multiplication, the control and data paths shared common wires for signal processing from the control path directing the data path and signal feedback from the data registers (i.e. the least significant bit in register B) in the data path to the control path. The latter was elected to prioritize proper separation between the control and dart paths, as while a data path with internal lsb logic avoids additional external wires and logic, this would involve the datapath making control decisions, i.e. whether to add the contents of register A to the result register. As a whole, the control path was designed as a state machine, implementing the val/rdy interface and interating over the 32 computational cycles, while the data path was designed as a simple iterative adder, shifting and reassigning registers when prompted.

Similarly, the division and remainder submodule was constructed using a similar approach. Aside from the trivial differences between iterative multiplication and divison/remainder as prescribed, additional registers and wires were utilized by the datapath in the division/remainder submodule to ensure correct handling of signed and unsigned operations.

No extensions were made to the prescribed datapath, and no extensions suggested by the lab manual were implemented.

Testing Methodology

When testing modules, a uniform testing stategy is trivially advantageous, but non-trivial to develop. Since the testing framework and unit tests for signed multiply, unsigned multiply, signed division, and signed remainder were already provided, only unit tests were unsigned division and unsigned remainder were developed. When constructed test cases, the following operations were considered, testing the quotient and remainder simultaneous, as the division/remainder submodule returned both the 32-bit quotient and the 32-bit remainder as one 64-bit output: division by zero (repeatable, but not necessarily correct output), division by one, division by a number that is not a factor of the dividend,

division by a number that is a factor of the dividend, division by the dividend, division by a number that is greater than the dividend, and division by a number that is less than the dividend.

Evaluation

The simulation results are as follows:

Register A	Operation	Register B	Result	Number of Cycles
0xdeadbeef	*	0x1000000	0xfdeadbeef0000000	33
0xf5fe4fbc	/	0x00004eb6	0xffffdf75	33
0x0a01b044	%	0xffffb14a	0x00003372	33
0xf5fe4fbc	/u	0x00004eb6	0x00032012	33
0x0a01b044	%u	0xffffb14a	0x0a01b044	33

See Appendix A for a screenshot of the simulation results.

These operations yield the expected results in the expected cycle count: 32 cycles for iterating through the computation and 1 cycle for signing the result, or maintaining its unsigned state.

Appendix A