Floating Point Coprocessor ASIC Implementation
ECE 337
Final Proposal
Wednesday 11:30 Lab Section 3

Harry Jung
Matt King
Andy Shi
Tasha Weidler
TA: Yunus Akhtar

March 23, 2015

1. Executive Summary

In many aspects of computing, performance is defined as the throughput of the system. *How quickly can the system process data?* This is an important question to ask when there is a precedence on the time it takes to reach a solution as well as the validity of the solution. Hardware applications large and small may contain many processing units, usually consisting of a primary central processing unit (CPU) and many coprocessors that augment the functionality of the main processor. Offloading specific tasks to dedicated coprocessors for tasks like I/O integration, arithmetic functions, and graphics processing can significantly increase the throughput of the overall system.

The objective of this design project is to create a floating point arithmetic coprocessor (FPU) capable of addition and subtraction, as well as more advanced functions such as multiplication and calculation of sine values. The purpose of the FPU in context of a larger system is to handle floating point arithmetic, reserving CPU resources for higher-level program management. The design of the FPU will focused on a specific function, making an ASIC implementation ideal. The remainder of this proposal will detail the technical aspects of an ASIC implementation of the FPU.

2. Design Specifications

2.0 System Usage

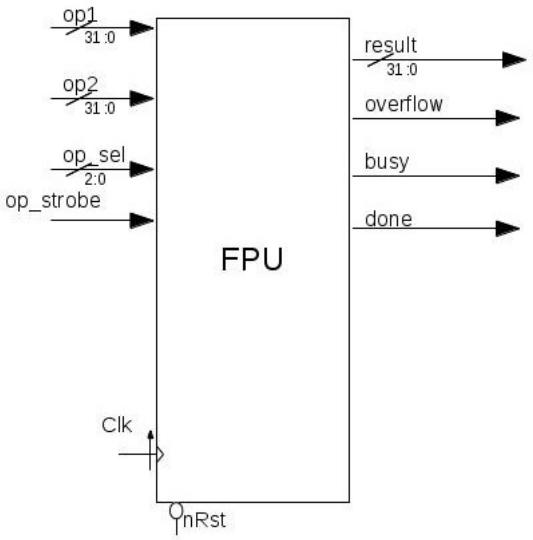

Figure 1: Chip Layout Overview

2.1 Operational Characteristics

Signal Name	Type	Num. Bits	Description
clk	input	1	100 MHz System Clock.
n_rst	input	1	System wide active low reset.
op_strobe	input	1	This signal strobes the FPU to tell it that new data is being input and to begin a new operation.
op1	input	32	Input operand 1. This operand must be a valid number conforming to the IEEE standard 754. Input will be in normalized form.
op2	input	32	Input operand 2. This operand must be a valid number conforming to the IEEE standard 754. Input will be in a normalized form.
op_sel	input	3	Operation select. `000`: add, `001`: sub, `010`: mul, `011`: sine, `100`: cosine.
result	output	32	Output result. The output will also be a number conforming to the IEEE standard 754.
overflow	output	1	Error condition where output is too large to be represented. The resultant value will be invalid.
busy	output	1	This signal will be asserted when the FPU is unable to take in any more requests.
done	output	1	This signal will be asserted for a single clock cycle when an operation is completed.

This design will make use of the IEEE 754 Standard binary32 floating point number representation. This standard uses 32-bit data to represent floating point numbers consisting of a sign bit, 8-bit exponent width, and a 23-bit fraction. Since this standard describes many types of floating point data and operations, some generalizations and assumptions will be made for the simplicity of this design, such as using a single rounding scheme as well as exception handling of NaN and infinity conditions.

The Operation of the design will be achieved by use of an input decoding module to determine the operation requested, and then blocks for individual functions. The FPU will be designed to facilitate operations in parallel. If an addition operation is being performed and the module receives a multiply operation, it will start the multiply operation before the addition operation is completed. This ensures maximum usage of the available resources to increase the overall throughput of the module.

The flow of data through each block will be similar for each operation requested. There will be a sub-block to "pre-process" the data to allow the floating point numbers to operated on, the operation sub-block itself, and then the "post-process" sub-block to normalize data. The input and output decode blocks will handle sending data too and from each operation block.

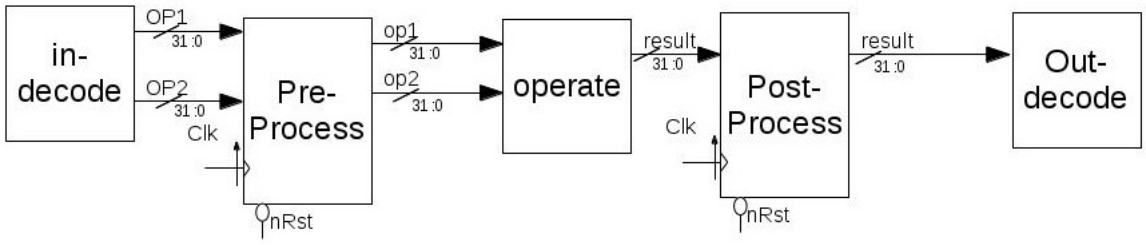

Figure 2: Data Flow through Block

The functionality of the module will be split into several functional blocks. Descriptions and signals associated with each block are described below:

Input Decode

- **Module Name:** indecode
- **File Name:** indecode.sv
- This block takes all the system level inputs and determines which operation is requested and what to do with the data input.
- In order to facilitate the parallel operation of the arithmetic operations, the decode block will add the requested opcode to a FIFO buffer. This allows the output block to determine the order in which the operations were requested.
- The arithmetic units will use a shared operand bus, and each operation will have a dedicated start signal to tell that block the values on the bus are for that operation and to begin computing the result.
- This block is purely combinational, acting as a multiplexer for the different operations, and ties the system level inputs to the specific blocks in the module.

Signal Name	Type	Num. Bits	Description
op_strobe	input	1	This signal strobes the FPU to tell it that new data is being input and to begin a new operation.
op1	input	32	Input operand 1. This operand must be a valid number conforming to the IEEE standard 754. Input will be in normalized form.
op2	input	32	Input operand 2. This operand must be a valid number conforming to the IEEE standard 754. Input will be in a normalized form.
op_sel	input	3	Operation select. `000`: add, `001`: sub, `010`: mul, `011`: sine, `100`: cosine.
op1_out	output	32	Output to the shared operand 1 bus.
op2_out	output	32	Output to the shared operand 2 bus.
add_start	output	1	Signal to tell the add block to begin computation.
mul_start	output	1	Signal to tell the multiply block to begin computation.
sine_start	output	1	Signal to tell the sine block to begin computation.
opcode_out	output	1	Output of opcode to FIFO buffer.
read_data	output	1	Signal to tell the FIFO buffer to read in the opcode.

Add & Subtract

- **Module Name:** addsub
- **File Name:** addsub.sv
- This block takes the two operands as inputs and adds them together. The `mode` input switches to subtraction by using radix complement.
- The operands will be input in a normalized format, however the block will handle processing the operands such that the exponents will match, allowing the operation to take place. The block will then normalize the result prior to outputting the value.

Signal Name	Type	Num. Bits	Description
clk	input	1	100 MHz System Clock.
n_rst	input	1	System wide active low reset.
add_start	input	1	Signal to tell block to begin computation.
mode	input	1	Chooses add or subtract. An input of `0` will be addition, and an input of `1` will be subtraction.
op1	input	32	Input operand 1. This operand must be a valid number conforming to the IEEE standard 754.
op2	input	32	Input operand 2. This operand must be a valid number conforming to the IEEE standard 754.
add_result	output	32	Output result. The output will also be a number conforming to the IEEE standard 754.
add_done	output	1	This signal is asserted when the add or subtract operation is complete.
add_overflow	output	1	Error condition where output is too large to be represented. The resultant value will be invalid.

Multiply

- **Module Name:** multiply
- **File Name:** multiply.sv
- This block takes the two operands as inputs and multiplies them together.
- The block will handle adjusting the exponents to allow the operation to take place and then normalize the data prior to output.

Signal Name	Type	Num. Bits	Description
clk	input	1	100 MHz System Clock.
n_rst	input	1	System wide active low reset.
mul_start	input	1	Signal to tell the block to begin computation.
op1	input	32	Input operand 1. This operand must be a valid number conforming to the IEEE standard 754.
op2	input	32	Input operand 2. This operand must be a valid number conforming to the IEEE standard 754.
mul_result	output	32	Output result. The output will also be a number conforming to the IEEE standard 754.
mul_done	output	1	This signal is asserted when the multiply operation is complete.
mul_overflow	output	1	Error condition where output is too large to be represented. The resultant value will be invalid.

Sine/Cosine

- **Module Name:** sincos
- **File Name:** sincos.sv
- This block takes one radian input and outputs the sine of that value.
- This block will use a taylor series representation of the sine or cosine value. The series is an addition of the input divided by increasing factorials. This block will make use of the inverse of these factorials stored in memory, and then multiply it by the input.
- This block will have dedicated add and multiply components to free the add and multiply blocks to allow parallel operations to take place.

Signal Name	Type	Num. Bits	Description
clk	input	1	100 Mhz System Clock.
n_rst	input	1	System wide active low reset.
sine_start	input	1	Signal telling the block to begin computation.
opx	input	32	Input operand 2. This operand must be a valid number conforming to the IEEE standard 754.
sine_result	output	32	Output result. The output will also be a number conforming to the IEEE standard 754.
sine_done	output	1	This signal is asserted when the sine/cosine operation is completed.

FIFO Buffer

- **Module Name:** fifobuff
- **File Name:** fifobuff.sv
- This block is a first in first out buffer responsible for storing the order in which the operations were requested. This allows the design to perform different operations in parallel and output the results in the order in which their requests were made.

Signal Name	Type	Num. Bits	Description
clk	input	1	100 Mhz System Clock.
n_rst	input	1	System wide active low reset.
read_data	input	1	Signal telling the buffer to read in the opcode.
opcode_in	input	3	Opcode data to be read into the buffer
opcode_out	output	3	Next opcode to be output.

Output Decode

- **Module Name:** outdecode
- **File Name:** outdecode.sv
- This block handles the system level outputs.
- Using the FIFO, this block determines which result value to output.

Signal Name	Type	Num. Bits	Description
add_result	input	32	Result value from add operation.
mul_result	input	32	Result value from multiply operation.
sine_result	input	32	Result value from sine operation.
add_done	input	1	Signal asserted when add operation is completed.
mul_done	input	1	Signal asserted when multiply operation is completed.
sine_done	input	1	Signal asserted when sine operation is completed.
add_overflow	input	1	Signal asserted when add block has encountered an overflow condition.
mul_overflow	input	1	Signal asserted when multiply block has encountered an overflow condition.
fifo_out	input	3	Opcode of next operation result to output.
result	output	32	Result of operation performed.
done	output	1	Signal asserted when operation requested is completed.
overflow	output	1	Signal asserted when operation requested has encountered an overflow condition. Output result is not valid.

2.2 Requirements

The target application for this design is to be interfaced with a primary processor via two 32-bit input buses, and a 2-bit operation select bus. Output will be interfaced to the primary processor via a 32-bit result bus and a single bit overflow error bus. This design calls for a relatively high pin count, and modifications could be made to optimize chip footprint by utilizing a single bi-directional 32-bit bus to handle transmission of operands and results. However, since the scope of this project is focused on specifically the FPU rather than integration, the goal will be to create a simple general solution that could be easily optimized for specific integration. The goal for the timing of the system will be to optimize the design such that a system clock of 100 MHz can be achieved.

It is assumed that all data passed to the FPU will be in a normalized format, and the CPU will never send the FPU invalid data not in this format. When the busy flag is asserted, the CPU will wait for the FPU to free resources before sending additional data.

3. Design Architecture

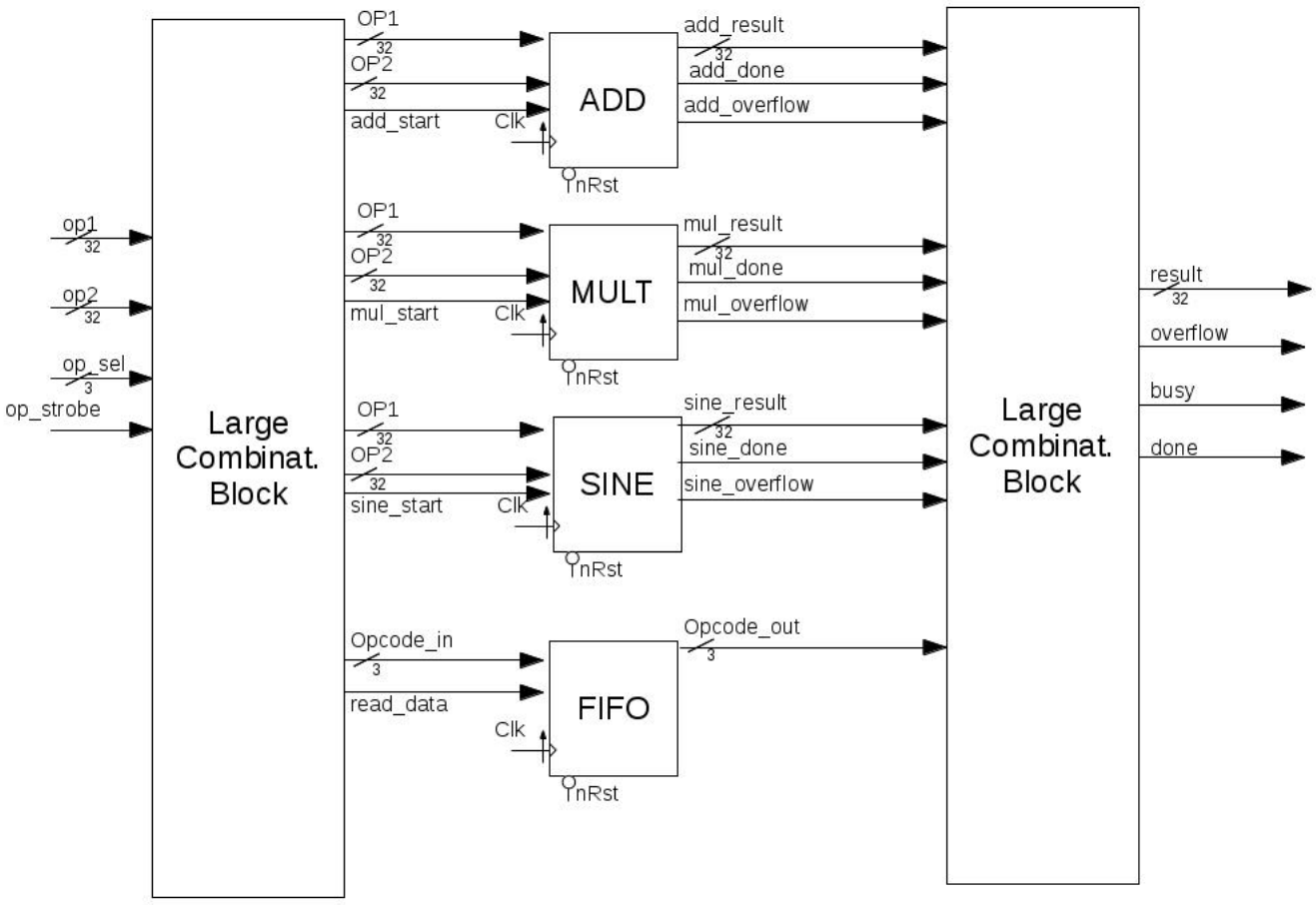

Figure 3: Design Architecture Diagram

4. Success Criteria

Fixed Criteria (12 points)

- Test benches (2 points): Test benches exist for all top level components and the entire design. The test benches must cover all of the functional requirements given in the design specific criteria.
- Design synthesis (4 points): The entire design synthesizes completely, without any inferred latches, timing arcs, or sensitivity list warnings.
- Source and mapped versions (2 points): The source and mapped version of the complete design must behave the same for all test cases. The mapped version must simulate without any timing errors except at time zero.
- Source and mapped versions (2 points): The source and mapped version of the complete design must behave the same for all test cases. The mapped version must simulate without any timing errors except at time zero.
- IC layout (2 points): A complete IC layout must be produced that passes all geometry and connectivity checks.
- Targets (2 points): The entire design complies with targets for area, pin count, throughput, and clock rate determined by course staff.

Design Specific Criteria (8 points)

- Operations (3 points): Demonstrate by simulation of Verilog test benches that the complete design is able to perform 5 different operations, add, subtract, multiply, sine, and cosine.
- Time efficient (1 point): Demonstrate by simulation of Verilog test benches that the complete design is able to perform the multiply, sine, and cosine operations within 4 clock cycles.
- Faulty values (1 point): Demonstrate by simulation of Verilog test benches that the complete design is able to detect and ignore any faulty opcodes or operands.
- FIFO (2 points): Demonstrate by simulation of Verilog test benches that the complete design is able to consecutively take in an add or subtract, a multiply, or a sine or cosine opcode, perform each of these operations at the same time, and produce the outputs in the order that the opcodes were received.
- Operation results (1 point): Demonstrate by simulation of Verilog test benches that the complete design is able to yield the correct results for each operation exactly for add, subtract, sine, and cosine, or within 3 decimal places for multiplication.

5. Project Timeline

- Week 11: Start modelsim and .sv files, project specification, RTLs, and basics.
- Week 12: Preliminary design budgets, Start coding basic blocks (add, subtract, multiple), Create FIFO algorithm, Start test bench for basic modules.
- Week 13: Design review and evaluations, Finish up basic blocks, start on sine/cosine blocks, Test bench for sine/cosine.
- Week 14: Project verification plans due, Finish up and connect all modules, Project documentations.
- Week 16: Final Presentations.

6. Division of Tasks

- Andy/Matt: Basic modules (add, subtract, multiple)
- Andy/Matt: Test bench for basic modules
- Harry/Tasha: FIFO, custom and standard lib
- Harry/Tasha: Testbench for FIFO
- Team: Sine/Cosine + test benches
- Team: Optimizations
- Team: Document preparations