VIP 37921 JUNIOR PARTICIPATION
FALL 2024

School of Electrical and Computer Engineering, Purdue University

SoCET – System on Chip Extension Technologies

Floating Point Unit in RISC-V

Fall 2024

Name:
Saandiya KPS Mohan

Submission Date:
11/27/2024

Advisor:
Dr. Mark C Johnson, Cole Nelson, Om P Kotwal

Summary:

The project aimed to design and implement a 32-bit IEEE 754 compliant floating-point unit (FPU) as part of a RISC-V processor architecture. The primary focus was on creating modules for addition, subtraction, and multiplication that adhered to the IEEE standard. These modules required handling complex tasks such as normalization, alignment of operands, rounding, and addressing special cases like NaN, infinity, and subnormal numbers. The project achieved a significant milestone by completing the design and simulation of a functional FPU. I designed and implemented the floating-point adder and subtractor modules, ensuring proper handling of both common and edge cases. Through rigorous testbench creation and debugging, I verified their correctness under randomized and directed test scenarios. Furthermore, I collaborated with my team to integrate these modules into the larger FPU system and contributed to refining its overall performance and compatibility within the RISC-V architecture. The result was a robust and efficient FPU that met all design requirements and provided valuable insights into hardware design practices.

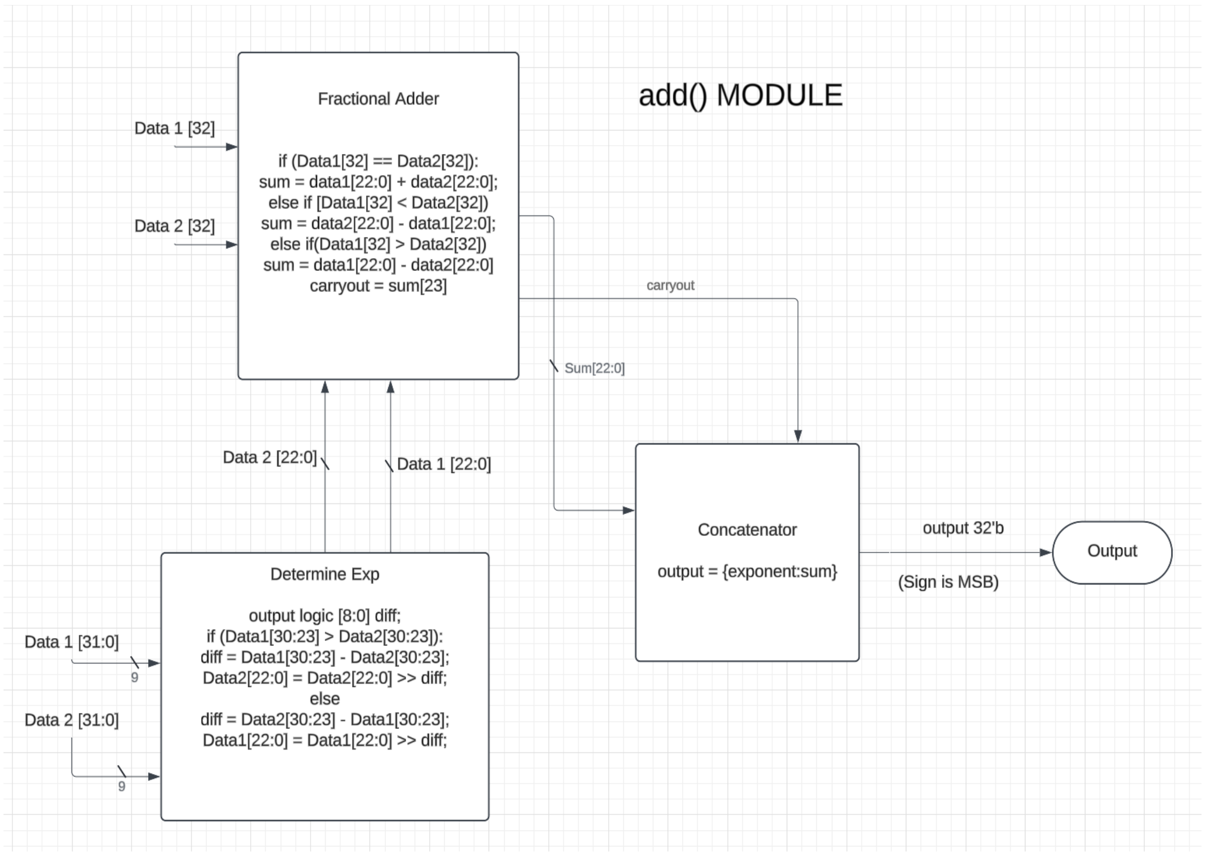
Goals:

- Develop a deep understanding of IEEE 754 floating-point arithmetic.
- Designing the floating-point adder module, handling all IEEE 754 cases.
- Gaining hands-on experience with System Verilog for RTL design.
- Writing effective testbenches to verify module functionality.
- Collaborating with team members to ensure seamless integration of modules into the FPU.
- Make a poster explaining the background, device architecture, benchmarks, results, and future work in a poster and a presentation for the SoCET and Undergraduate Research Expo.
- Create a framework for future teams that want to work with FPU and add more cores to the system.
- Create weekly logs for future students to read and understand the design of the coherency unit.
- Provide documentation for future students on the FPU design on GitHub/SharePoint.
- Finish Professional Development activities, such as research ethics and working in a multicultural team, were selected specifically for this project.

Initially, I planned to focus solely on the adder module. However, as the project progressed, I recognized the need to adapt to evolving team requirements. Midway through the semester, I expanded my responsibilities to include the multiplier module, which involved iterative design and debugging. Additionally, my goals shifted slightly to address challenges that arose during the project, such as refining the handling of subnormal numbers and optimizing the performance of the multiplier. Another change was the greater emphasis on debugging and integration. I took an active role in assisting team members with their modules, identifying and resolving compatibility issues during integration. These changes allowed me to broaden my skill set and contribute to the project's overall success more significantly.

Accomplishments and Learning:

- Designed and implemented a 32-bit floating-point adder and subtractor module compliant with the IEEE 754 standard.

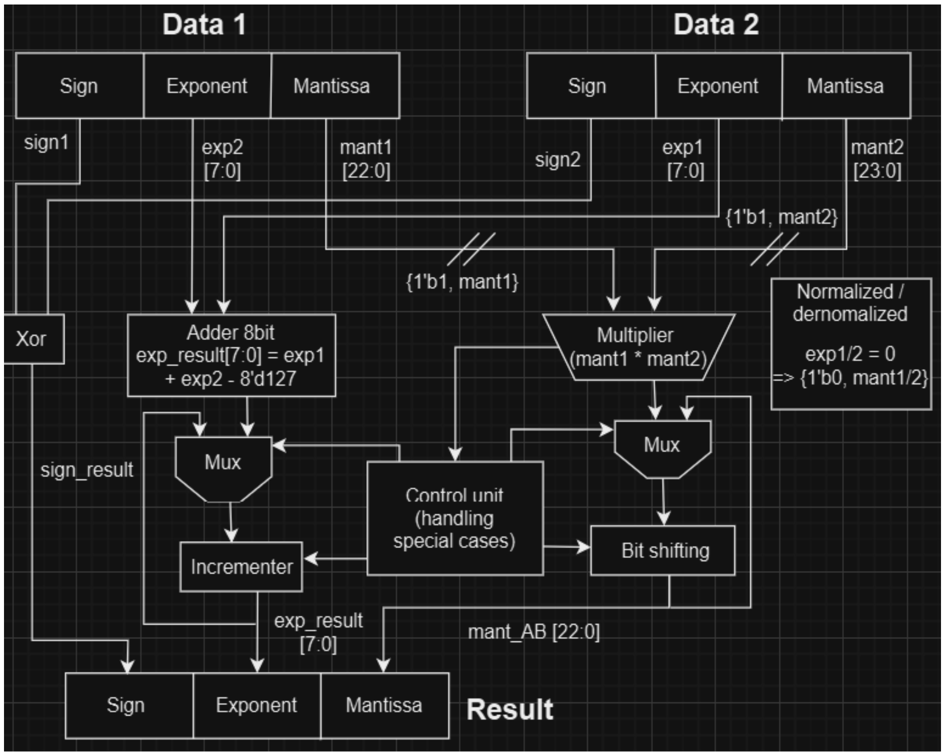

add() MODULE
Fractional Adder
if (Data1[32] == Data2[32]):
sum = data1[22:0] + data2[22:0];
else if [Data1[32] < Data2[32])
sum = data2[22:0] - data1[22:0];
else if(Data1[32] > Data2[32])
sum = data1[22:0] - data2[22:0]
carryout = sum[23]
Data 1 [32]
Data 2 [32]
carryout
Sum[22:0]
Data 2 [22:0]
Data 1 [22:0]
Concatenator
output = {exponent:sum}
output 32'b
(Sign is MSB)
Output
Determine Exp
output logic [8:0] diff;
if (Data1[30:23] > Data2[30:23]):
diff = Data1[30:23] - Data2[30:23];
Data2[22:0] = Data2[22:0] >> diff;
else
diff = Data2[30:23] - Data1[30:23];
Data1[22:0] = Data1[22:0] >> diff;
Data 1 [31:0]
Data 2 [31:0]
9
9

- Achieved proper operand alignment, normalization, and rounding. The module accurately handled special cases such as NaN, infinity, and subnormal values.
- Incorporated a robust normalization process using case statements for efficient shifting, ensuring the result adhered to the standard's precision requirements.
- Developed a comprehensive testbench to validate the module against directed and randomized test cases, ensuring accuracy under a wide range of conditions.
- Integrated the adder, subtractor and multiplier modules into the larger FPU system.
- Debugged module interactions and ensured compatibility between components.
- Contributed to finalizing the FPU's architecture by participating in team discussions and offering design optimizations to improve area and performance.
- Presented at the Undergraduate Research Expo and a SoCET meeting.
- Provided documentation for the Floating-Point Unit on GitHub/SharePoint for future designs.
- Finished all the PD Activities I planned for this project and learned a lot about ethics in research, intellectual property, working in a multicultural team, handling situations in these teams, and respecting everyone.
- Provided design logs weekly on the design of the floating-point unit, the research done for the project, and another discussion in the weekly meetings.

Teamwork:

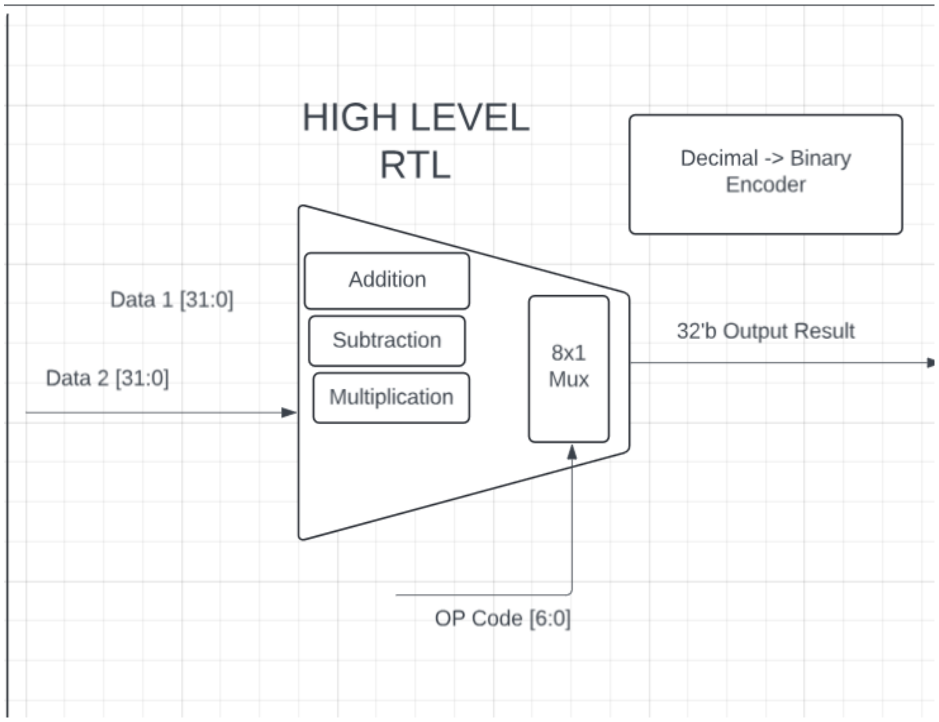
I worked with Justin Sanchez, Duc Pham Minh and Jain Iftesam on this project, and here are some of the contributions that they made:

- Justin Sanchez:
  - Worked on testing the adder module with me.
  - Worked on the final presentation at SoCET and the poster presentation we gave at the Undergraduate Research Expo.
  - Worked on the abstract for the Undergraduate Research Expo.
  - Presented the final presentation to the SoCET team, explaining the testing methods and IEE754 format.
- Duc Pham Minh:
  - Designed an incomplete multiplication module.
  - Worked on the slides for presentations, especially the multiplication slides.

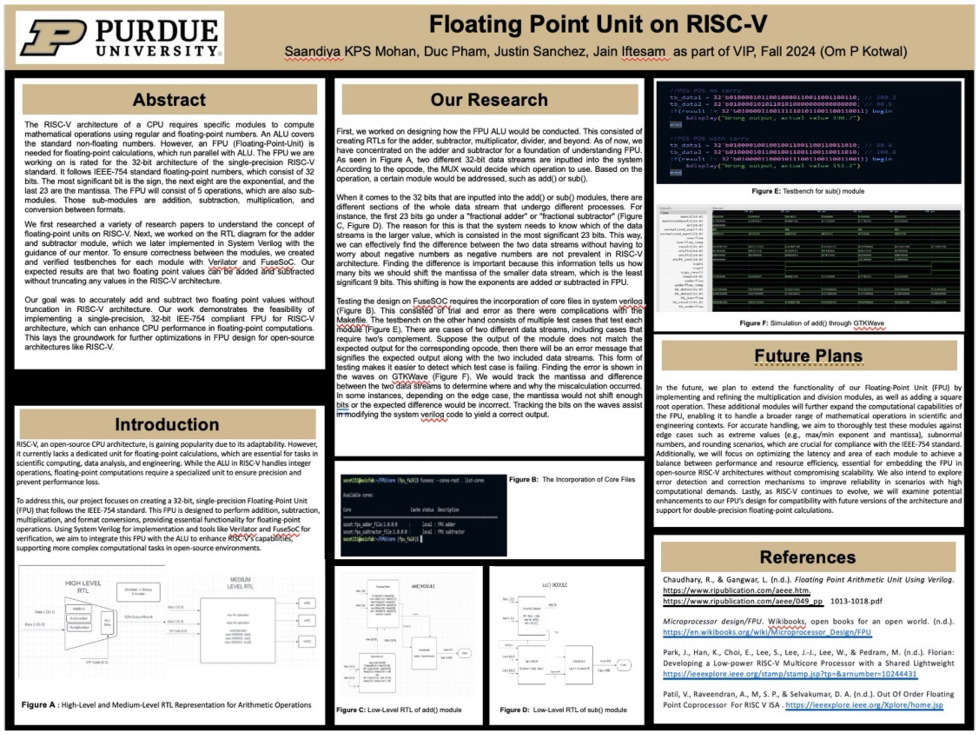


  - Presented the final presentation to the SoCET team, explaining the multiplication modules.
- Jain Iftesam:
  - Worked on the final presentation at SoCET and the poster presentation we gave at the Undergraduate Research Expo.
  - Worked on the abstract for the Undergraduate Research Expo.
  - Presented the final presentation to the SoCET team, explaining the background of FPU.
- Saandiya KPS Mohan:
  - Designed the single precision adder, subtractor and top module for FPU, including the decoder.

- Tested and debugged adder and subtractor module.
- Provided documentation for the floating point unit and created the diagrams for the top-level system on GitHub for future designs.



- I finished all the PD Activities I planned for this project and learned a lot about ethics in research, intellectual property, working in a multicultural team, handling situations in these teams, and respecting everyone.
- Made a poster explaining the project's details, which I presented to multiple judges at the Undergraduate Research Expo.
- Worked on the final presentation at SoCET and the poster presentation we gave at the Undergraduate Research Expo.
- Worked on the abstract for the Undergraduate Research Expo.
- Presented the final presentation to the SoCET team, explaining the adder, subtractor, top module and future plans.
- Provided design logs weekly on the design of the coherency unit, the research done for the project, and another discussion in the weekly meetings.



PURDUE UNIVERSITY

Floating Point Unit on RISC-V

Saandiya KPS Mohan, Duc Pham, Justin Sanchez, Jain Iftesam as part of VIP, Fall 2024 (Om P Kotwal)

Abstract

The RISC-V architecture of a CPU requires specific modules to compute mathematical operations using regular and floating-point numbers. An ALU covers the standard non-floating numbers. However, an FPU (Floating-Point-Unit) is needed for floating-point calculations, which run parallel with ALU. The FPU we are working on is rated for the 32-bit architecture of the single-precision RISC-V standard. It follows IEEE-754 standard floating-point numbers, which consist of 32 bits. The most significant bit is the sign, the next eight are the exponential, and the last 23 are the mantissa. The FPU will consist of 5 operations, which are also sub-modules. Those sub-modules are addition, subtraction, multiplication, and conversion between formats.

We first researched a variety of research papers to understand the concept of floating-point units on RISC-V. Next, we worked on the RTL diagram for the adder and subtractor module, which we later implemented in System Verilog with the guidance of our mentor. To ensure correctness between the modules, we created and verified testbenches for each module with Verilator and FuseSoC. Our expected results are that two floating point values can be added and subtracted without truncating any values in the RISC-V architecture.

Our goal was to accurately add and subtract two floating point values without truncation in RISC-V architecture. Our work demonstrates the feasibility of implementing a single-precision, 32-bit IEE-754 compliant FPU for RISC-V architecture, which can enhance CPU performance in floating-point computations. This lays the groundwork for further optimizations in FPU design for open-source architectures like RISC-V.

Introduction

RISC-V, an open-source CPU architecture, is gaining popularity due to its adaptability. However, it currently lacks a dedicated unit for floating-point calculations, which are essential for tasks in scientific computing, data analysis, and engineering. While the ALU in RISC-V handles integer operations, floating-point computations require a specialized unit to ensure precision and prevent performance loss.

To address this, our project focuses on creating a 32-bit, single-precision Floating-Point Unit (FPU) that follows the IEEE-754 standard. This FPU is designed to perform addition, subtraction, multiplication, and format conversions, providing essential functionality for floating-point operations. Using System Verilog for implementation and tools like Verilator and FuseSoC for verification, we aim to integrate this FPU with the ALU to enhance RISC-V's capabilities, supporting more complex computational tasks in open-source environments.

Figure A : High-Level and Medium-Level RTL Representation for Arithmetic Operations

Our Research

First, we worked on designing how the FPU ALU would be conducted. This consisted of creating RTLs for the adder, subtractor, multiplicator, divider, and beyond. As of now, we have concentrated on the adder and subtractor for a foundation of understanding FPU. As seen in Figure A, two different 32-bit data streams are inputted into the system According to the opcode, the MUX would decide which operation to use. Based on the operation, a certain module would be addressed, such as add() or sub().

When it comes to the 32 bits that are inputted into the add() or sub() modules, there are different sections of the whole data stream that undergo different processes. For instance, the first 23 bits go under a "fractional adder" or "fractional subtractor" (Figure C, Figure D). The reason for this is that the system needs to know which of the data streams is the larger value, which is consisted in the most significant 23 bits. This way, we can effectively find the difference between the two data streams without having to worry about negative numbers as negative numbers are not prevalent in RISC-V architecture. Finding the difference is important because this information tells us how many bits we should shift the mantissa of the smaller data stream, which is the least significant 9 bits. This shifting is how the exponents are added or subtracted in FPU.

Testing the design on FuseSOC requires the incorporation of core files in system verilog (Figure B). This consisted of trial and error as there were complications with the Makefile. The testbench on the other hand consists of multiple test cases that test each module (Figure E). There are cases of two different data streams, including cases that require two's complement. Suppose the output of the module does not match the expected output for the corresponding opcode, then there will be an error message that signifies the expected output along with the two included data streams. This form of testing makes it easier to detect which test case is failing. Finding the error is shown in the waves on GTKWave (Figure F). We would track the mantissa and difference between the two data streams to determine where and why the miscalculation occurred. In some instances, depending on the edge case, the mantissa would not shift enough bits or the expected difference would be incorrect. Tracking the bits on the waves assist in modifying the system verilog code to yield a correct output.

```
socet21@asicfab:~/FPU/core (fpu_fa24)$ fusesoc --cores-root . list-cores
Available cores:
Core                              Cache status  Description
================================================================
socet:fpu_adder_file:1.0.0      :    local : FPU adder
socet:fpu_subtractor_file:1.0.0 :    local : FPU subtractor
socet21@asicfab:~/FPU/core (fpu_fa24)$
```

Figure B: The Incorporation of Core Files

Figure C: Low-Level RTL of add() module

Figure D: Low-Level RTL of sub() module

```
//POS POS no carry
tb_data1 = 32'b01000010110010000110011001100110; // 100.2
tb_data2 = 32'b01000010101101010000000000000000; // 90.5
if(result != 32'b01000011001111101011001100110011) begin
    $display("Wrong output, actual value 190.7")
end

//POS POS with carry
tb_data1 = 32'b01000010010010011001100110011010; // 50.4
tb_data2 = 32'b01000010110010011001100110011010; // 100.8
if(result != 32'b01000011000101110011001100110011) begin
    $display("Wrong output, actual value 151.2")
end
```

Figure E: Testbench for sub() module

Figure F: Simulation of add() through GTKWave

Future Plans

In the future, we plan to extend the functionality of our Floating-Point Unit (FPU) by implementing and refining the multiplication and division modules, as well as adding a square root operation. These additional modules will further expand the computational capabilities of the FPU, enabling it to handle a broader range of mathematical operations in scientific and engineering contexts. For accurate handling, we aim to thoroughly test these modules against edge cases such as extreme values (e.g., max/min exponent and mantissa), subnormal numbers, and rounding scenarios, which are crucial for compliance with the IEEE-754 standard. Additionally, we will focus on optimizing the latency and area of each module to achieve a balance between performance and resource efficiency, essential for embedding the FPU in open-source RISC-V architectures without compromising scalability. We also intend to explore error detection and correction mechanisms to improve reliability in scenarios with high computational demands. Lastly, as RISC-V continues to evolve, we will examine potential enhancements to our FPU's design for compatibility with future versions of the architecture and support for double-precision floating-point calculations.

References

Chaudhary, R., & Gangwar, L. (n.d.). *Floating Point Arithmetic Unit Using Verilog*. https://www.ripublication.com/aeee.htm. https://www.ripublication.com/aeee/049_pp 1013-1018.pdf

Microprocessor design/FPU. Wikibooks, open books for an open world. (n.d.). https://en.wikibooks.org/wiki/Microprocessor_Design/FPU

Park, J., Han, K., Choi, E., Lee, S., Lee, J.-J., Lee, W., & Pedram, M. (n.d.). Florian: Developing a Low-power RISC-V Multicore Processor with a Shared Lightweight https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10244431

Patil, V., Raveendran, A., M, S. P., & Selvakumar, D. A. (n.d.). Out Of Order Floating Point Coprocessor For RISC V ISA . https://ieeexplore.ieee.org/Xplore/home.jsp

Future Work:

As I transition to my senior design project, I will not be continuing with the current FPU development. However, there are several potential directions for future work that others could pursue based on the progress made this semester. One area is optimizing the existing modules for improved performance and reduced area utilization. This could involve refining the multiplier logic or implementing more efficient normalization techniques. Additionally, future efforts could focus on extending the FPU's capabilities to include more complex operations, such as division and square root computation. These additions would enhance the processor's versatility and provide a more comprehensive arithmetic system. Another potential avenue is testing and integrating the FPU into a broader processor design, ensuring seamless compatibility and evaluating its performance in real-world applications. For someone continuing this project, documenting and debugging integration challenges, along with further developing verification environments for large-scale testing, would be invaluable.

Appendix:

- https://purdue0-my.sharepoint.com/:p:/r/personal/sanch431_purdue_edu/Documents/Microsoft%20Teams%20Chat%20Files/FPU_Presentation.pptx?d=w59a86129fac34b26bffe3391f8aaa2d3&csf=1&web=1&e=EGkOMa
- **Floating Point Unit in RISC-V.mp4**
- **https://purdue0.sharepoint.com/:o:/r/sites/ENGR-ECE-O-SOCET/_layouts/15/Doc.aspx?sourcedoc=%7B9d92b34d-e064-4226-9957-cdfce583cf47%7D&action=editnew**