IEEE-754 FLOATING-POINT ADDER

AN IMPLEMENTATION OF THE FP ADDER IN C AND VERILOG

MOMEN ANANI AND MOHAMMAD MATAR

12112318 - 12112195

INTRODUCTION

Floating point operations are critical in various computational tasks but are often not natively supported by HDLs like Verilog. This assignment aims to implement a floating-point adder first in C and then Verilog. The objective is to understand and experiment with the algorithm and functionality of floating-point arithmetic operations in both software and hardware contexts.

Methodology

For the C code, it was straight forward implementation; just follow the algorithm and how the hardware unit works to achieve the desired functionality of the IEEE-754 adder.

We first implemented the MyFloat struct that was the basic block of the prototypes in the program

```
struct MyFloat {
unsigned int fraction;
unsigned int exponent;
unsigned int sign;
};
```

The main function prototypes for the code are initiated as follows

```
struct MyFloat toMyFloat(float f);
struct MyFloat MyFloatAdder(struct MyFloat input1, struct MyFloat input2);
float toFloat(struct MyFloat mf);
float random_float_range(float min, float max);
```

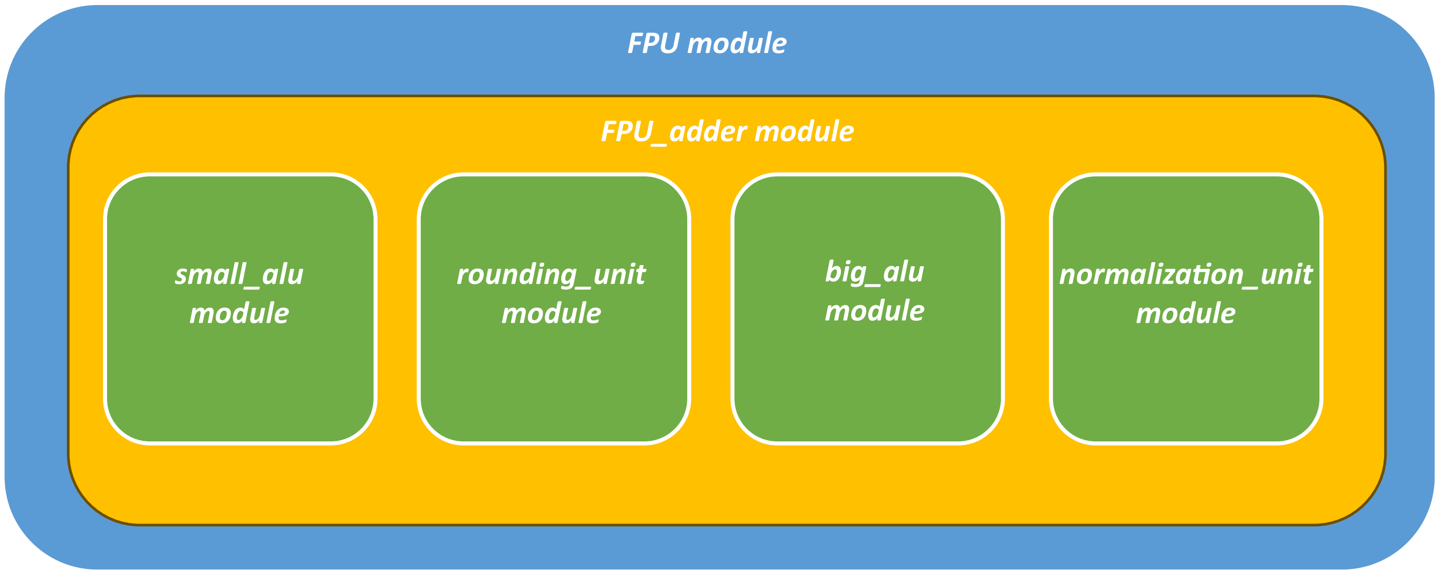
the toMyFloat function converts the standard float type to the user defined struct type.

The MyFloatAdder follows multiple steps mainly:

1. Checking for zero summation
2. Determine which exponent is larger and which is smaller to align them for summation
3. Perform addition/subtraction based on the sign bit
4. Normalizing and checking for overflow and underflow and handle NaN/inf exceptions

For the Verilog part, we took the same logic and made it resemble the hardware unit a bit more.

The main structure of the code consisted of the following hierarchical design:



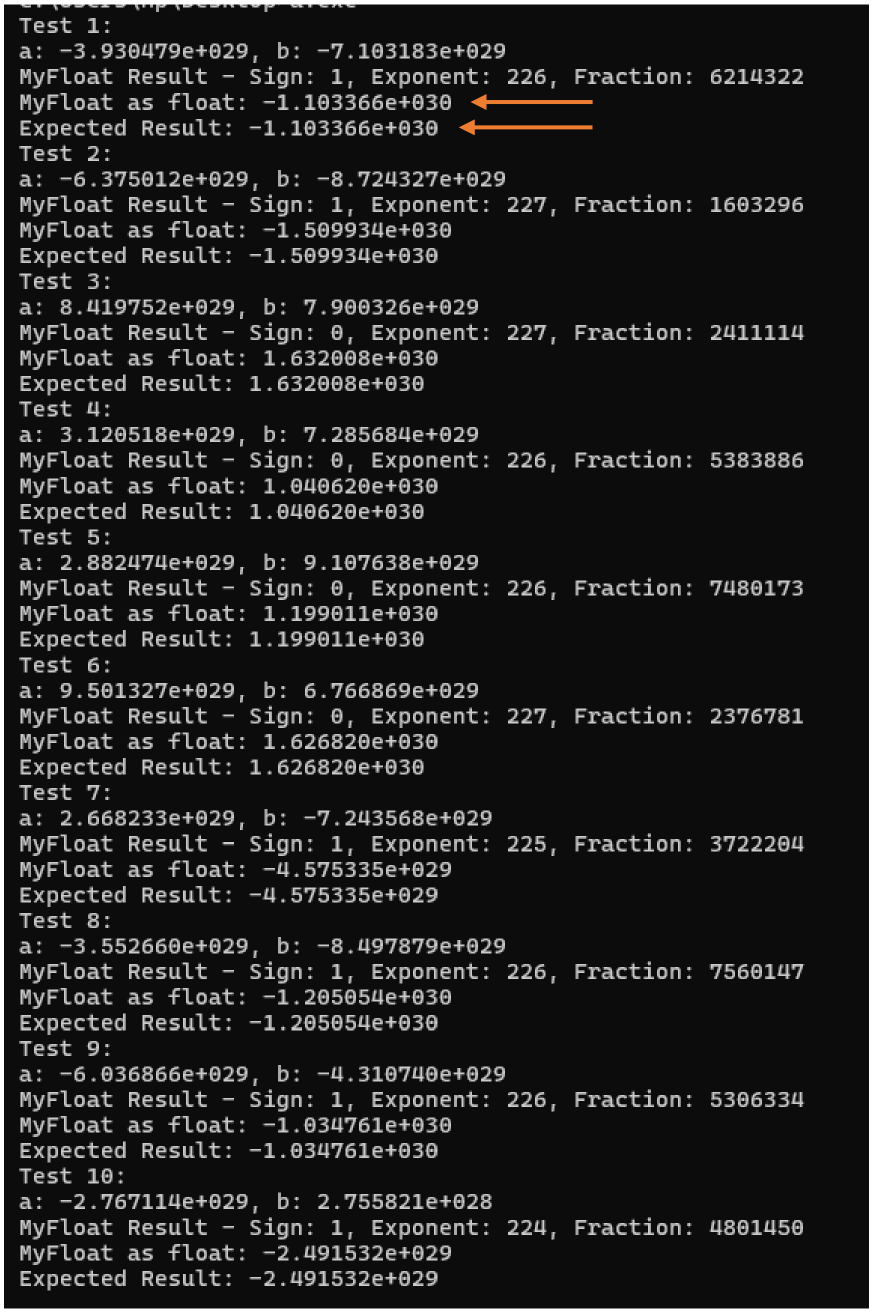
As the name suggests for each module, the corresponding functionality is applied to it.

You can notice that there is no control unit. Well, yes and no. The control unit is implemented implicitly in each unit defined in the FPU_adder module. After analysis of the diagram, implementing an explicit control unit would make the code more complex since it is already simple. *The control unit controls signals that are simple enough to be calculated and figured within other modules easily.*

Results and Discussion

For the C code, a random floating number was given as an input each time to test:

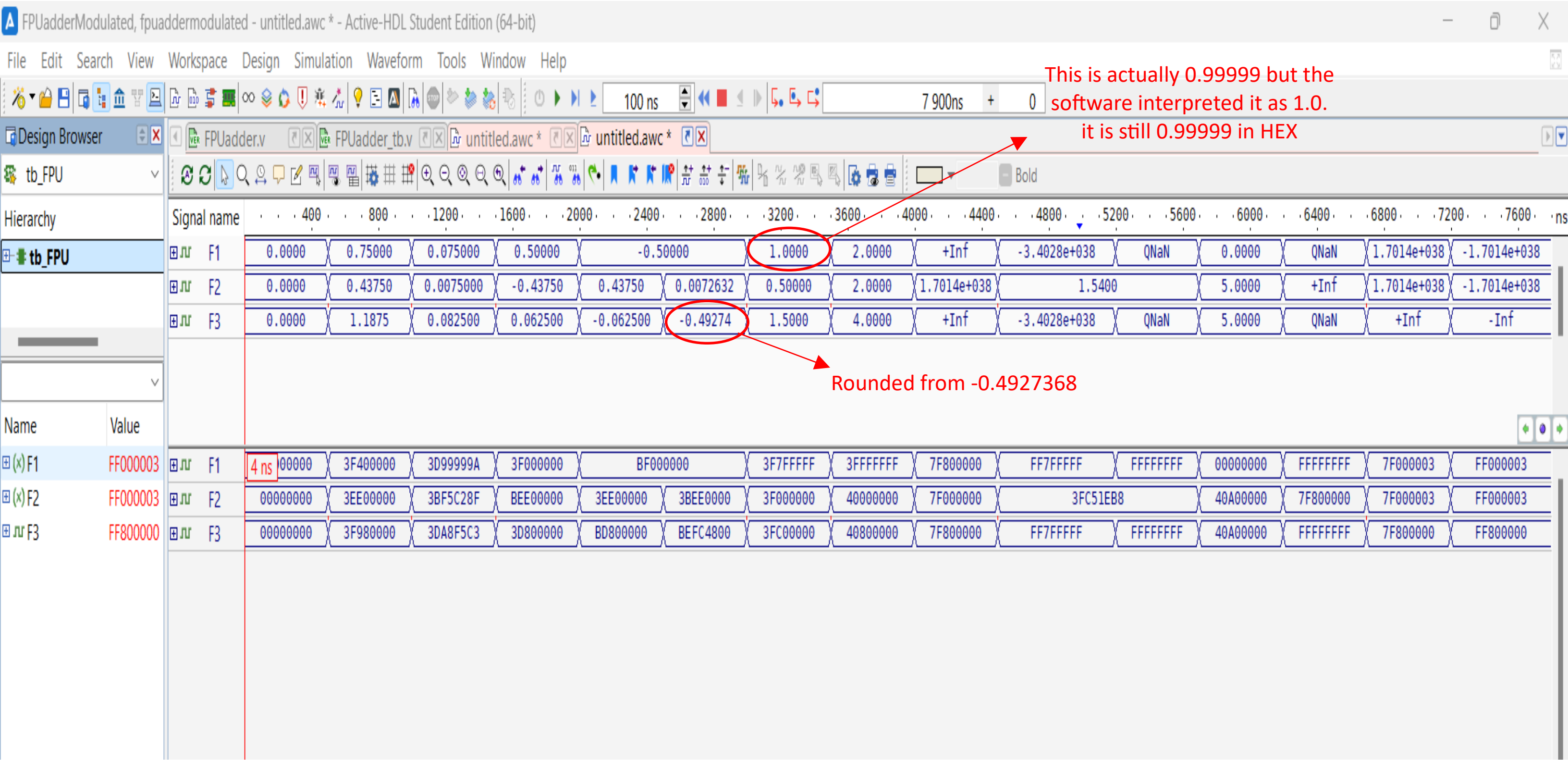
Here is the code result after running it:



```
Test 1:
a: -3.930479e+029, b: -7.103183e+029
MyFloat Result - Sign: 1, Exponent: 226, Fraction: 6214322
MyFloat as float: -1.103366e+030
Expected Result: -1.103366e+030
Test 2:
a: -6.375012e+029, b: -8.724327e+029
MyFloat Result - Sign: 1, Exponent: 227, Fraction: 1603296
MyFloat as float: -1.509934e+030
Expected Result: -1.509934e+030
Test 3:
a: 8.419752e+029, b: 7.900326e+029
MyFloat Result - Sign: 0, Exponent: 227, Fraction: 2411114
MyFloat as float: 1.632008e+030
Expected Result: 1.632008e+030
Test 4:
a: 3.120518e+029, b: 7.285684e+029
MyFloat Result - Sign: 0, Exponent: 226, Fraction: 5383886
MyFloat as float: 1.040620e+030
Expected Result: 1.040620e+030
Test 5:
a: 2.882474e+029, b: 9.107638e+029
MyFloat Result - Sign: 0, Exponent: 226, Fraction: 7480173
MyFloat as float: 1.199011e+030
Expected Result: 1.199011e+030
Test 6:
a: 9.501327e+029, b: 6.766869e+029
MyFloat Result - Sign: 0, Exponent: 227, Fraction: 2376781
MyFloat as float: 1.626820e+030
Expected Result: 1.626820e+030
Test 7:
a: 2.668233e+029, b: -7.243568e+029
MyFloat Result - Sign: 1, Exponent: 225, Fraction: 3722204
MyFloat as float: -4.575335e+029
Expected Result: -4.575335e+029
Test 8:
a: -3.552660e+029, b: -8.497879e+029
MyFloat Result - Sign: 1, Exponent: 226, Fraction: 7560147
MyFloat as float: -1.205054e+030
Expected Result: -1.205054e+030
Test 9:
a: -6.036866e+029, b: -4.310740e+029
MyFloat Result - Sign: 1, Exponent: 226, Fraction: 5306334
MyFloat as float: -1.034761e+030
Expected Result: -1.034761e+030
Test 10:
a: -2.767114e+029, b: 2.755821e+028
MyFloat Result - Sign: 1, Exponent: 224, Fraction: 4801450
MyFloat as float: -2.491532e+029
Expected Result: -2.491532e+029
```

Figure 1. C code output for FP adder MyFloat.

For the Verilog testbench that was run on Active-HDL software, this was the waveform:


FPUadderModulated, fpuaddermodulated - untitled.awc * - Active-HDL Student Edition (64-bit)
File Edit Search View Workspace Design Simulation Waveform Tools Window Help
100 ns
7 900ns
0
Design Browser
FPUadder.v
FPUadder_tb.v
untitled.awc *
untitled.awc *
tb_FPU
Bold
Hierarchy
tb_FPU
Signal name
Name
Value
ns

This is actually 0.99999 but the software interpreted it as 1.0. it is still 0.99999 in HEX

Rounded from -0.4927368

```
Signal name  · 400 · 800 · 1200 · 1600 · 2000 · 2400 · 2800 · 3200 · 3600 · 4000 · 4400 · 4800 · 5200 · 5600 · 6000 · 6400 · 6800 · 7200 · 7600 · ns
F1  0.0000 | 0.75000 | 0.075000 | 0.50000   | -0.50000             | 1.0000  | 2.0000 | +Inf         | -3.4028e+038 | QNaN | 0.0000 | QNaN | 1.7014e+038 | -1.7014e+038
F2  0.0000 | 0.43750 | 0.0075000 | -0.43750 | 0.43750   | 0.0072632 | 0.50000 | 2.0000 | 1.7014e+038 | 1.5400              | 5.0000 | +Inf | 1.7014e+038 | -1.7014e+038
F3  0.0000 | 1.1875  | 0.082500 | 0.062500  | -0.062500 | -0.49274  | 1.5000  | 4.0000 | +Inf         | -3.4028e+038 | QNaN | 5.0000 | QNaN | +Inf        | -Inf

Name  Value
F1    FF000003   F1  4 ns 00000 | 3F400000 | 3D99999A | 3F000000 | BF000000            | 3F7FFFFF | 3FFFFFFF | 7F800000 | FF7FFFFF | FFFFFFFF | 00000000 | FFFFFFFF | 7F000003 | FF000003
F2    FF000003   F2  00000000 | 3EE00000 | 3BF5C28F | BEE00000 | 3EE00000 | 3BEE0000 | 3F000000 | 40000000 | 7F000000 | 3FC51EB8           | 40A00000 | 7F800000 | 7F000003 | FF000003
F3    FF800000   F3  00000000 | 3F980000 | 3DA8F5C3 | 3D800000 | BD800000 | BEFC4800 | 3FC00000 | 40800000 | 7F800000 | FF7FFFFF | FFFFFFFF | 40A00000 | FFFFFFFF | 7F800000 | FF800000
```

Figure 2. waveform output for floating and HEX radix

The test cases are chosen to test certain cases, all of which are explained in the testbench Verilog file.

Conclusion

Implementing the floating-point adder logic in C was straightforward due to the simplicity of the language and the ability to directly translate the block diagram into code, on the HDL side, the modular approach in Verilog proved to be efficient, allowing each part of the operation to be handled within its own unit. This made the code more organized and manageable it can also help in adding other arithmetic units in the future. *The implicit control* within each unit streamlined the design, reducing complexity while maintaining functionality. Overall, this modular approach enhances the scalability and flexibility of the floating-point arithmetic unit design.