IEEE 754 Half-Precision Fused Multiply-Add (FMA) Unit

ECEN 4233 Project Report

Nikolas Brouwer
Oklahoma State University
Electrical and Computer Engineering

Fall 2025

Contents

1 Introduction

This project implements a fused multiply-add (FMA) unit for IEEE 754 binary16 floating-point numbers. The unit computes:

$$R = X \cdot Y + Z \tag{1}$$

1.1 FMA Block Diagram

Figure 1 shows the completed FMA architecture.

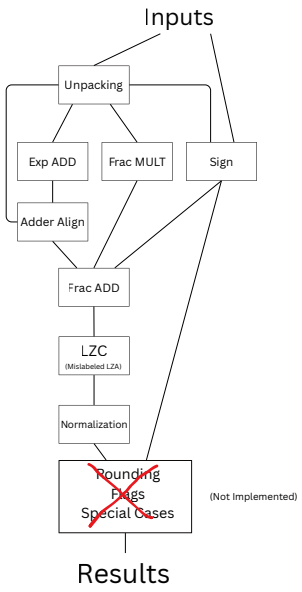

Figure 1: FMA block diagram.

1.2 Architecture Explanation

The architecture consists of the following major components:

1. Unpack - Take the X, Y, and Z inputs, and separate the exponent, sign, and fraction components. Figure out if the inputs are NaN, Inf, Zero, Subnormal, or the max value.
2. Multiply Unit - Multiply the fraction components of the X and Y inputs.
3. Exponent Adder - Add the exponents of X and Y, which is neccessary to find their product.
4. Alignment Shifter - Shift the fraction component of Z based on the exponent of the X and Y product.
5. Sign unit - Use the sign values from unpacking to find the sign.
6. Fraction Adder - Add the product of X and Y with the fraction aligned Z value.
7. Leading Zero Counter - Count the leading zeros after the addition step.
8. Normalization Unit - Normalize the addition value using the output from the LZC.

9. Rounding / Special Case Unit - Round the value if needed, adjust the value in special cases, and output the final result and flags. (This was not finished).

2 Test Results

2.1 Multiply Tests

Test	Description	Result
fmul0	Exp zero, RZ	PASS
fmul1	Positive normalized	PASS
fmul2	Signed normalized	PASS

Table 1: Multiply test results.

2.2 Add Tests

Test	Description	Result
fadd0	Exp zero	PASS
fadd1	Positive normalized	PASS
fadd2	Signed normalized	PASS

Table 2: Add test results.

2.3 FMA Tests

Test	Description	Result
fma0	Exp zero	PASS
fma1	Positive normalized	PASS
fma2	Signed normalized	PASSED 9994/10000
special_rz	Special cases, RNE	PASSED 4588/10000
baby_torture	Torture test	PASSED 484/3624

Table 3: FMA test results.

The fmul and fadd tests pass without issue. The first two fma tests passed, but fma2 failed 6 tests. All of the failed results were 1 value above the expected value, which suggests failed rounding. Many of the RNE tests failed as well, since the rounding unit was incomplete. Most of the baby_torture tests failed. Most of this was because of the incomplete rounding and special case unit, so the FMA was not well suited for this test. However, it is worth noting that some of the failed tests have expected values that don't make much sense. The first failed test was c3ec * 0000 + 3800, with an expected value of c3ec. Since X is being multiplies by 0, the result should be 3800, which is what my result was (it was also what the python program gave).

3 Synthesis Results

3.1 Area

Module	Area (μm^2)
Wrapper	2278
FMA Misc	347
FMA Add	5525
Unpack X	338
Unpack Y	314
Unpack Z	316
Align	3071
Exponent Add	294
LZC	4487
Multiplier	6471
Sign	42
Total	23482

Table 4: Area report.

3.2 Timing

Target frequency: 10 GHz

Metric	Value
Worst Negative Slack	-5.25

Table 5: Timing results.

The critical path was observed in:

> Unpack → Frac MULT → Frac ADD → LZC → Normalization → Finalization and Output

The slack was violated at the default frequency, and would need to be decreased for implementation.

3.3 Power

Measurement	Value	Value (with wrapper)
Switch Power (mW)	63	63
Int Power (mW)	19	34
Leak Power (nW)	148	154
Total Power (mW)	82	98

Table 6: Power results.

4 Project Management

4.1 Weekly Time Spent

Week	Hours	Activities
11/10	1	Setup
11/17	2	Unpack and EXP Add
11/24	18	FMUL (Complete), FADD (Complete), FMA
12/1	12	FMA Debugging, Synth, Report
Total	TODO	

Table 7: Weekly hours spent.

4.2 Reflections

Lessons learned:

- Incremental testing dramatically simplified debugging.
- Familiarization with assistance tools early in the process is helpful, since it lowers the barrier to entry, making them more approachable and reducing debugging time.
- Rounding logic and special cases caused most functional errors.
- Documentation during implementation was valuable.
- Debugging always takes longer than expected. Which is terrible, because it's expected to take a long time.

5 Conclusion

While the final FMA is reliable for FADD and FMUL, it runs into problems with edge cases for full FMA functionality. It does not handle special cases well, and has many errors rounding. Overall, its functions as a starting point, but improvements need to be made to be complete.

Future improvements could include:

- Reliable RNE rounding
- Hardware for special cases (NAN, INF, etc)
- Support for denormal numbers
- Pipelining
- LZA instead of LZC