# Verify a Design: Report

EEE4701

Dr. Mike Borowczak

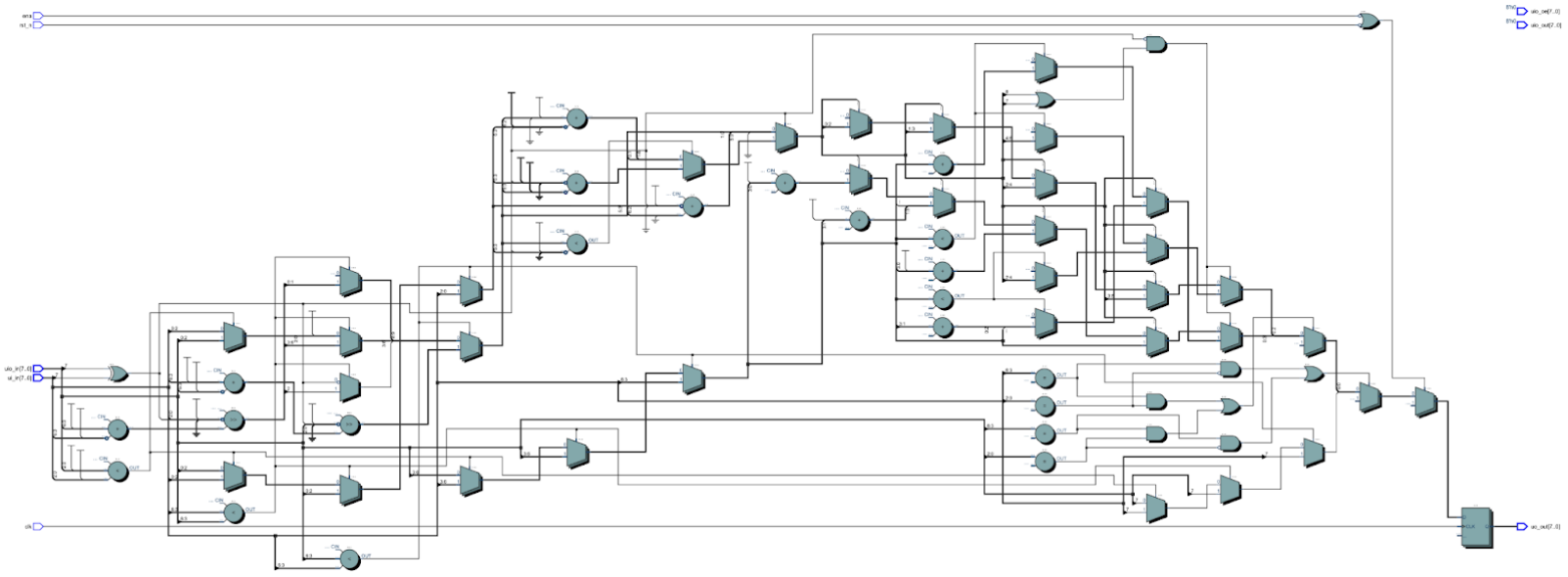
By: Michael Montoya, Evan Rees, Carter Clavell

## 1. Brief description:

The design we will be working with is an adder that can take two 8-bit arrays as an input and give an 8-bit output. The adder has additional features that consider things like infinity or rounding errors. It will also take the inputs and lay them out into three different sections. The first being a single bit for a sign. The next four bits will be for the exponent, and finally the last three bits for the mantissa.

We found our design from TinyTakeout 5. One of the unique things about our project is that it was made in two days by a single person.

Included below is a high level diagram of the design. It can also be found on our Github.



## 2. DUT Design Functionality:

For the adder, it does float addition like normal. It takes the 8 bit numbers we input and splits it into the 3 parts like described above.

First, if the sign bits are equal, then it will conduct binary addition on the mantissas'. If they are not then binary subtraction. It then compares the exponent portion of the two numbers, and takes the highest one.

Finally, it then adds the 4 portions back together into a single 8 bit number. The sign is either untouched or flipped depending on the binary addition/subtraction, the exponent is inherited from the highest binary exponent value, and finally the matiessa is the result of the binary addition/subtraction. This creates a final 8 bit result. Special cases such as infinity are also accounted for in the design when these values are detected the output is set accordingly.

**Test Bench Design Functionality:**

How we implemented our testbench is creating our own function in Verilog to convert an 8 bit number in the format: 1 bit as the sign, 4 bits as exponents and 3 bits being mantissa. Our function converts this value to its corresponding decimal value. It begins by extracting the components from the input. The sign bit is located at MSB and determines if the number is positive or negative.

The next number that is extracted is the exponent, which is stored as a biased value. The bias must be adjusted to obtain the real exponent which will be to subtract it by 7 to obtain the real exponent. The Mantissa is then extracted from the last 3 bits, it's also implied that the mantissa has an implicit leading 1; Meaning that the actual mantissa is calculated as 1.(mantissa).

The mantissa is iterated over from bit 2 to bit 0, with each bit representing a fractional binary value. These fractional values are added to the initial 1.0 mantissa. The function calculates the decimal value by applying the formula;

Floating point value = sign * mantissa_value * 2^exponent_value. If the sign bit is set to 1, the result is negated, indicating a negative number. The mantissa is scaled by the calculated exponent, and the function returns the decimal value.

3. **Test Plan:**

Our Test Plan is to make random variables to input into the adder, then verify the results on our side. We can do this by first making two random 8 bit numbers, inputting them into the adder and getting a result. We then convert the resulting 8 bit number into a float so that we can compare it to our test bench.

In our testbench we are converting the 8 bit inputs into floats in Verilog, then adding them directly. This gives us an accurate result whilst also creating valid inputs for our adder. We then compare the results directly to the adder's.

For our cover groups we will test as many cases as possible. We will test for infinity, high numbers, low numbers, medium numbers, zero, and both positive and negative. These cover groups should allow for excellent coverage of most scenarios the adder will face. We will also include a cover group for the enable pins and the like to verify those are working.

We expect about 200 to 400 tests are required for complete coverage. This is because we will need to properly test most scenarios and some may be hard to find. Although it may be possible to get good coverage with a little as 50 tests.

4. **Coverage Summary:**

In our coverage we were able to get 100% coverage with 175 Tests. We had two cover groups, one to test the enable and clk pins to verify those were working, and a cover group that looked at all the variants described in the test design. We made sure to look for infinity cases, High cases, Low cases, and the cases in each section of the 8 bit inputs.

We found that we got many zero cases in our random number generator, which is good as it helped find errors with small numbers in the adder. For more detailed information on the coverage data, please download the results.zip from EDAPlayground.

5. **Pass / Fail Summary:**

Our design had a 71.30% pass rate. We were able to code in any number of runs that we wanted to do. For this specific pass rate, we chose to run through 1000 times.

Interestingly, there are a few things that the test bed struggles with. One of them being that it really struggles with small numbers. Most of the floats were slightly above our tolerance which was set to 0.125. We set our tolerance to be small as it ensures that we have the highest accuracy possible. The smaller the margin of error, the better the adder and our test bench design will work.

One of the other things that it struggles with is very large positive and negative numbers. This is likely because of infinity cases. It does have infinity cases built into the DUT design, but it has a hard time determining what is an infinite value. Meaning that any large positive or negative number is considered infinity.

These two things are inherent issues with the adder and are the reason our accuracy will suffer a little bit. However, within our range, the values from the adder are extremely accurate because of how low we set the tolerance to be.

## 6. Summary:

The project report shows that the device under test (DUT) achieves about 70 percent accuracy, which is great. The main factor limiting this accuracy is the hardware design itself. Specifically, the DUT uses an 8-bit floating-point representation, which doesn't provide enough precision for complex calculations. The 3-bit mantissa is especially limiting, making it hard to represent numbers accurately. With such a small mantissa, the system struggles to handle small or large values properly, leading to rounding errors and lower overall accuracy.

Another issue is the 4-bit exponent, which restricts the range of numbers the DUT can handle. The exponent is responsible for scaling values, but with only 4 bits, the DUT can only represent a small range of exponents. This makes it difficult to manage very large or very small numbers, which further impacts the system's performance. Additionally, the adder in the design presents an issue where infinite cases can occur when all the bits are 1 in the result. This scenario was not accounted for in the testbench, which can lead to unexpected behavior or errors that are not properly tested or handled

While the design choices were made to keep the hardware simple, they create significant limitations. By increasing the bit-width for both the mantissa and exponent, the DUT could handle a wider range of numbers and achieve higher precision. However, this would also make the design more complex and resource-intensive, so it's important to balance these trade-offs.

7. **EDAPlayground:**

https://www.edaplayground.com/x/hDfY

## 8. Appendix:

Our Github

https://github.com/lavabrothers/VandV-Float-Adder-Project

Direct Link to Results
https://github.com/lavabrothers/VandV-Float-Adder-Project/tree/main/result

=== Test Summary ===
Total Tests Run: 175
Tests Passed: 122
Tests Failed: 53
Pass Rate: 69.71%

Total Coverage Summary

| SCORE | GROUP |
|---|---|
| 100.00 | 100.00 |