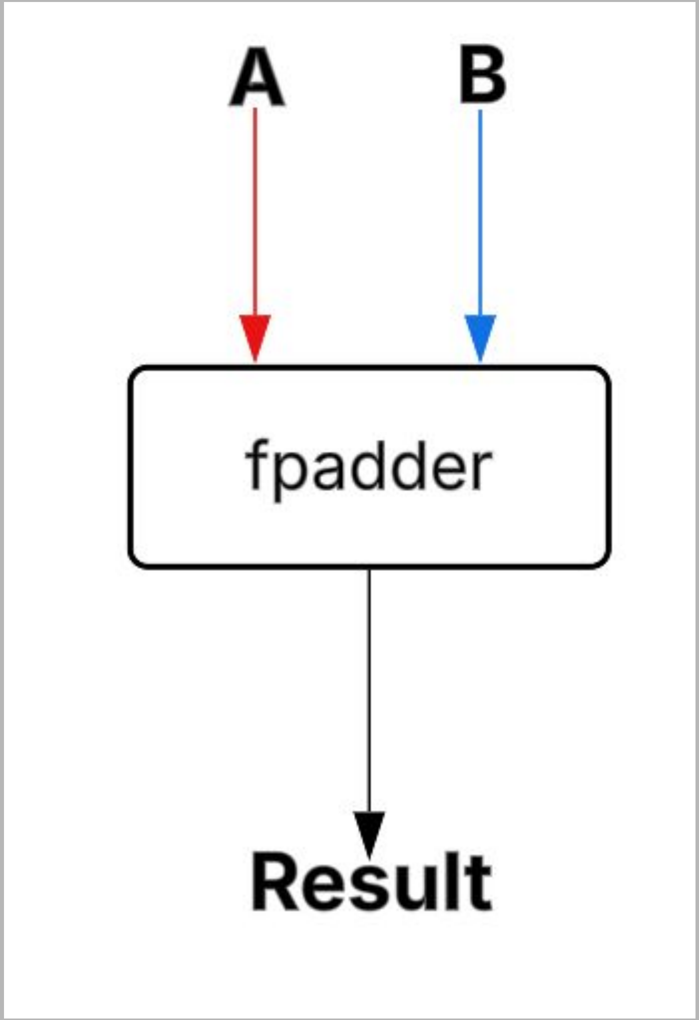
ECE (0x440EC000)
Introduction to SystemVerilog for Design and Verification

# Original Floating Point Adder Design

0100 0100 0000 1110 1100 0000 0000 0000

Chris Kane-Pardy & Violet Monti

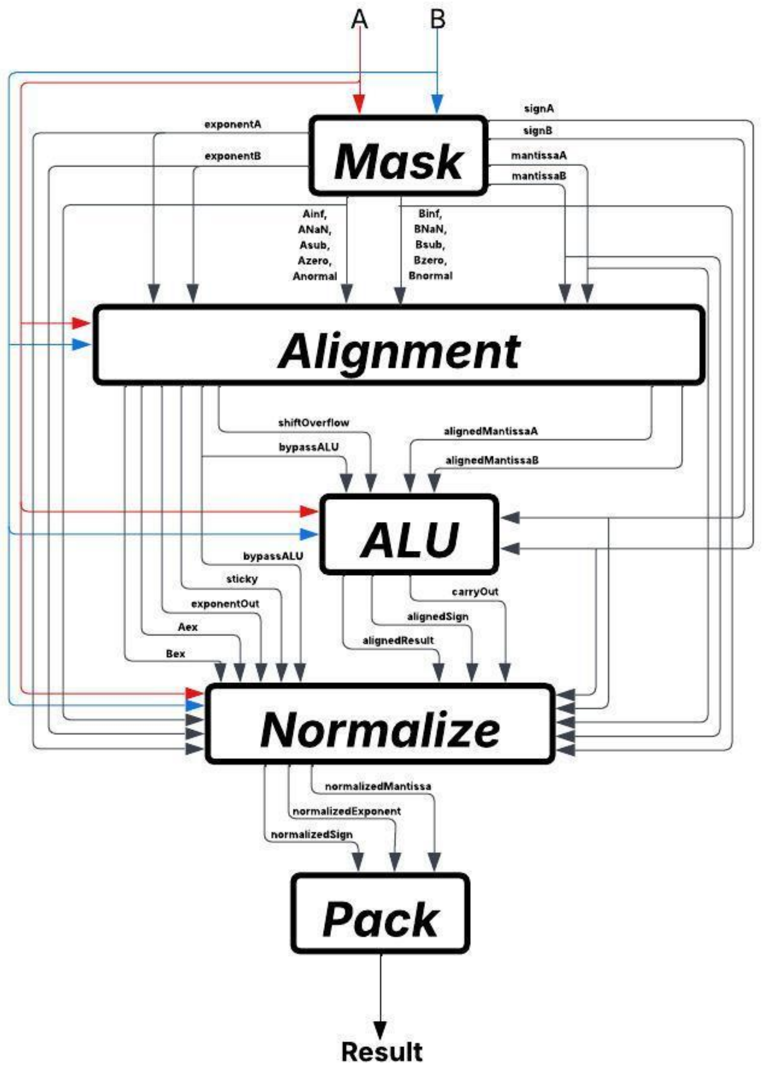


## Overview

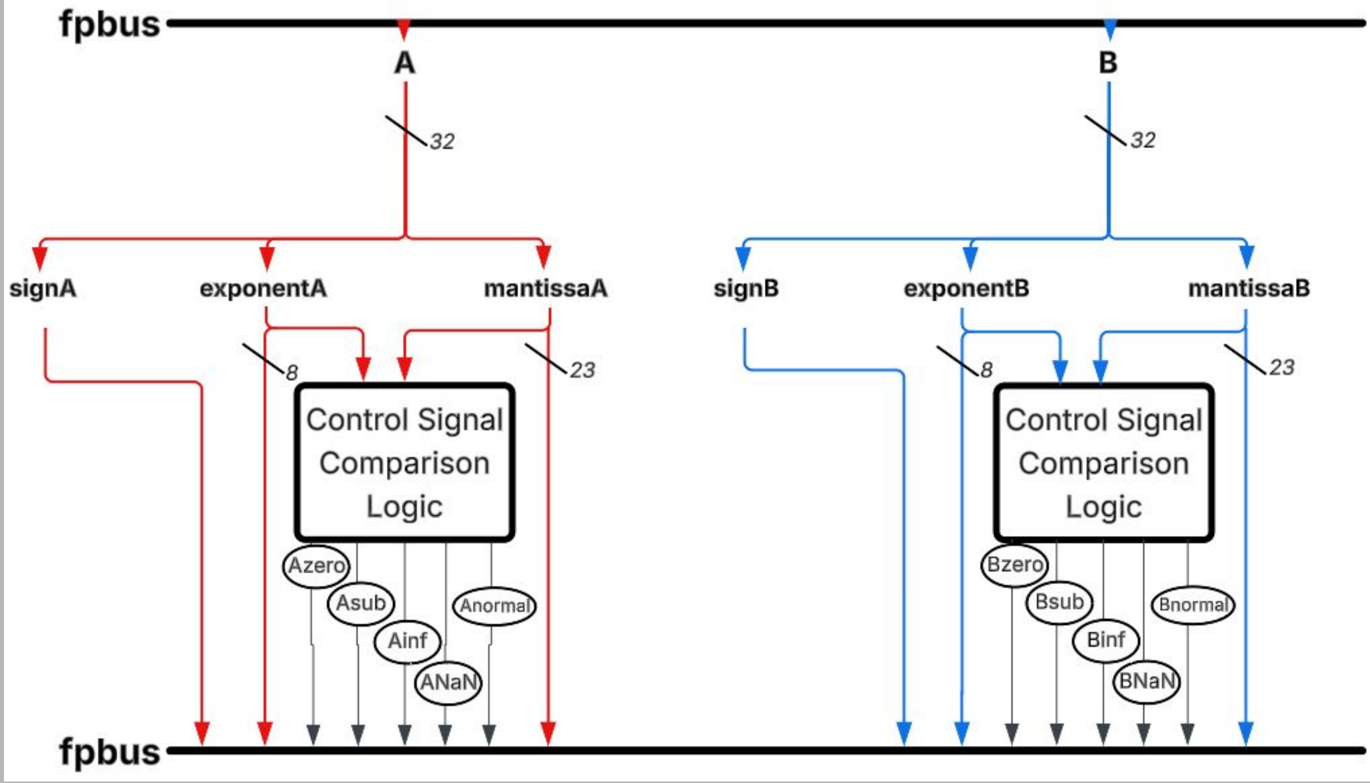
- Designed 5 submodules for Easy incremental Testing
- Easy support for the addition of different rounding modes (package)
- Decided not to parameterize due to lack of 16-bit floating point data type for accurate comparison
- Special handling for non-normal numbers
- IEEE-754 compliant
- Fully asynchronous to allow use in pipelines

# Interface (*fpbus*)

- Common interconnect for all signals reused by multiple submodules
- Allows for built-in naming checks
- 33 total signals
- 5 modports (one for each submodule)



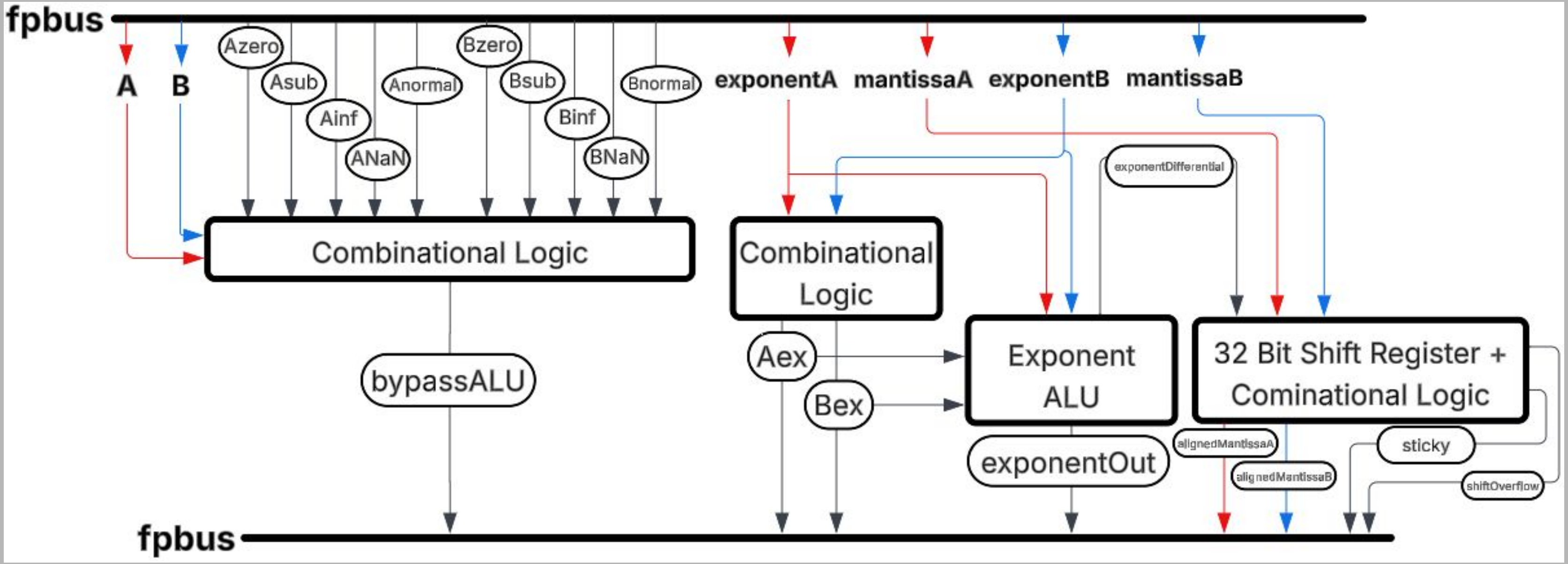
## Masking Module (*Mask*)



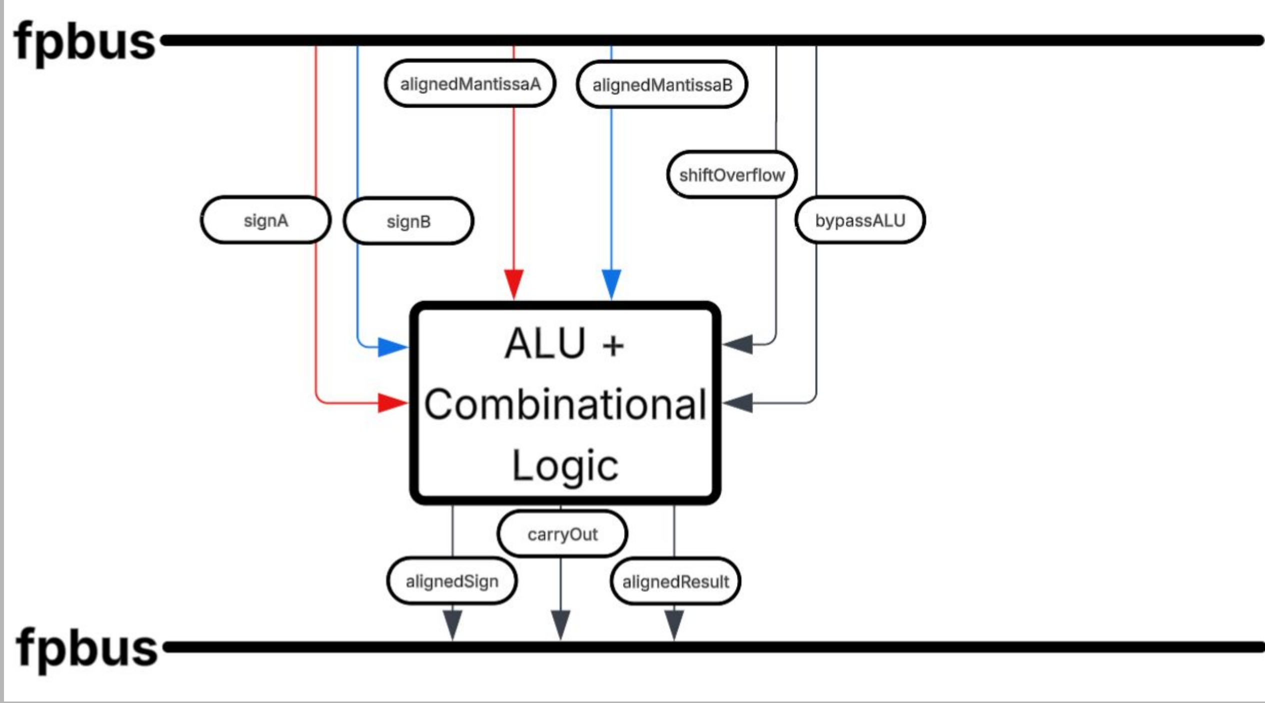
- Splits each input directly into a Sign, Exponent, and Mantissa value using bit slicing
- Computes and generates the control signals for both A and B to determine if there are non-normal numbers present in either addend

## Alignment Module (*Alignment*)

- Adds implicit ones and normalizes the lesser mantissa
- Signal for bypassing ALU for special cases such as NaN, Inf, Zeros
- Calculates Sticky Bit & Shift Overflow for proper rounding



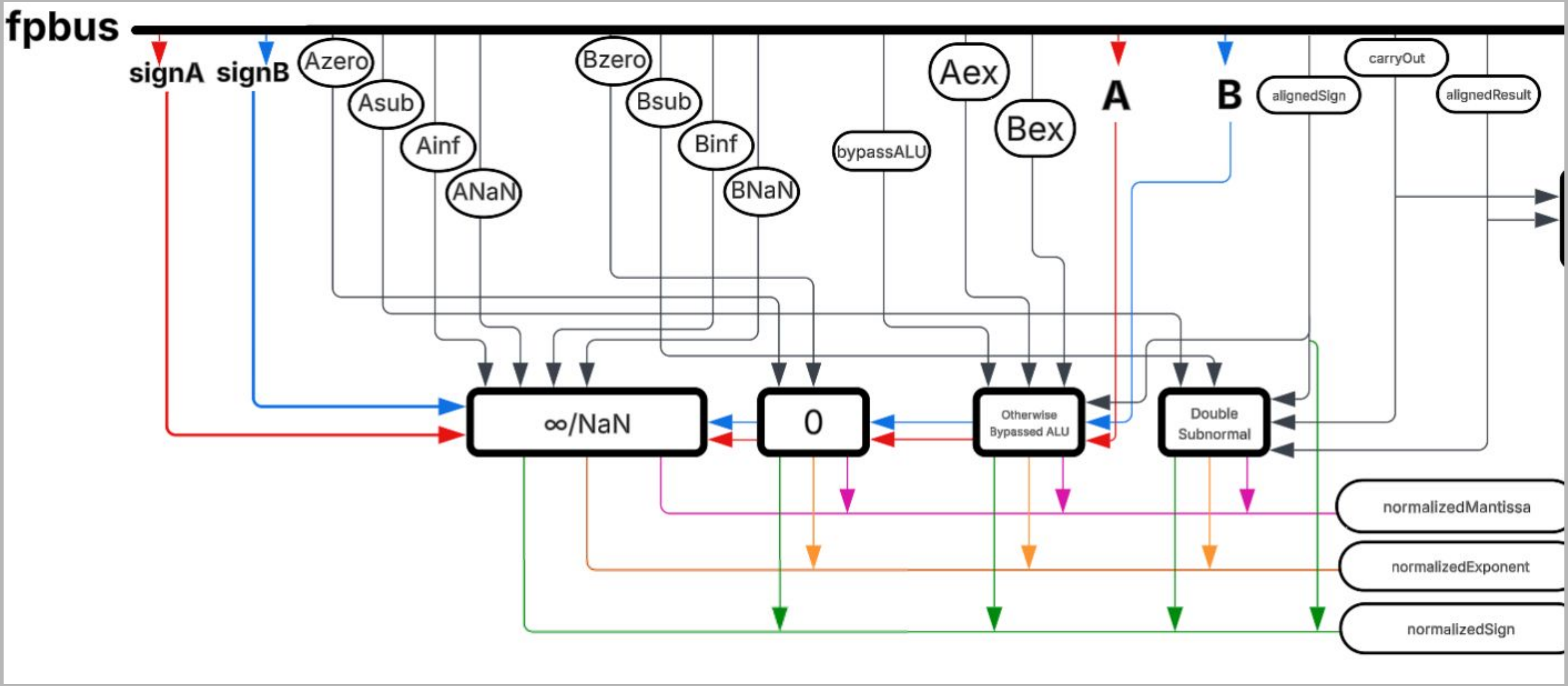
## ALU Module (*ALU*)



- Adds (or subtracts) the two aligned mantissas, plus additional subtraction for rounding accuracy
- Bypassed by special cases (handled during normalization)
- Tracks “carry out” for use in normalization
- Determines final sign bit to be used with a normal result

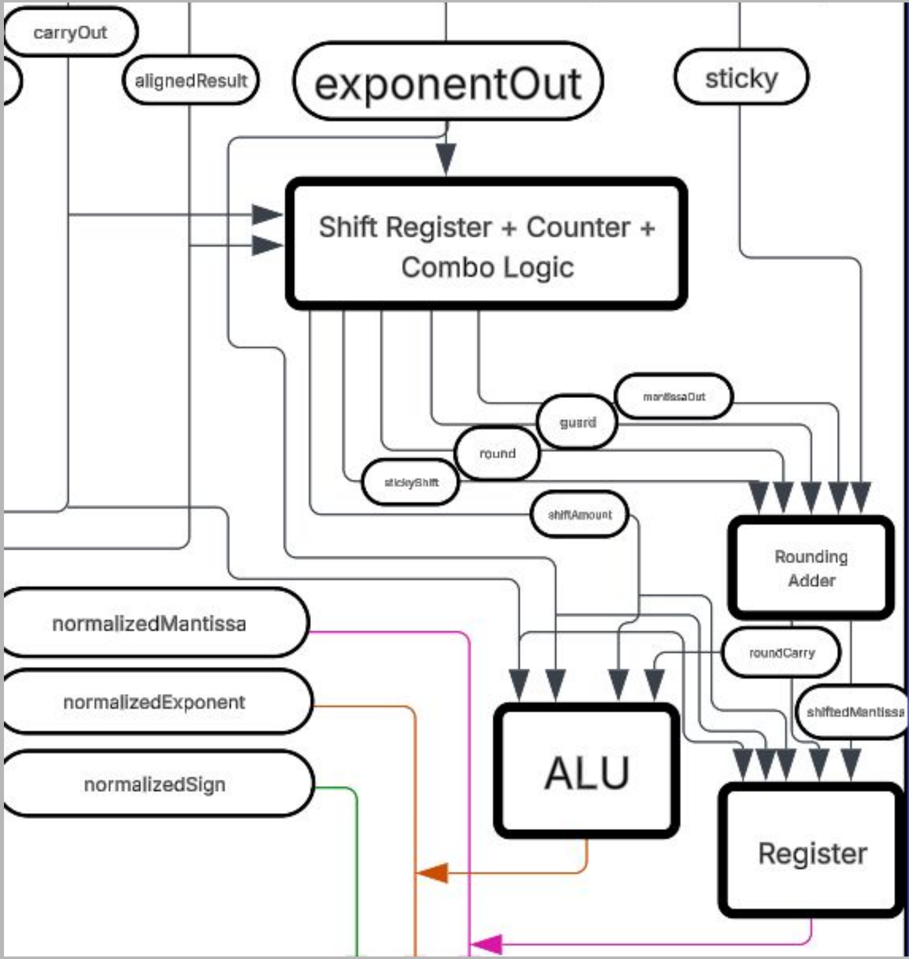
# Normalization Module (*Normalize*)

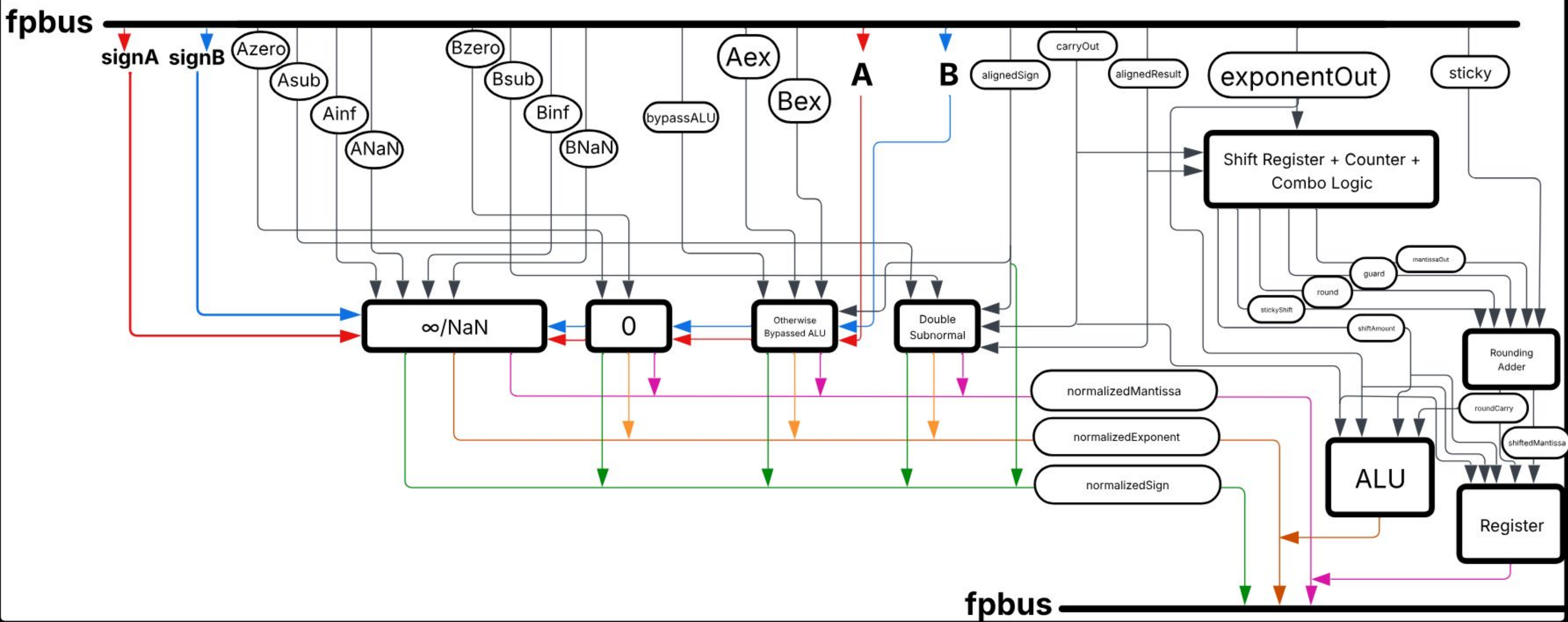
- Handles special cases that bypassed the ALU (±∞,±NaN,±0,±Subnormal)
- If inputs result in normal number, this portion will be bypassed



# Normalization Module (*Normalize*)

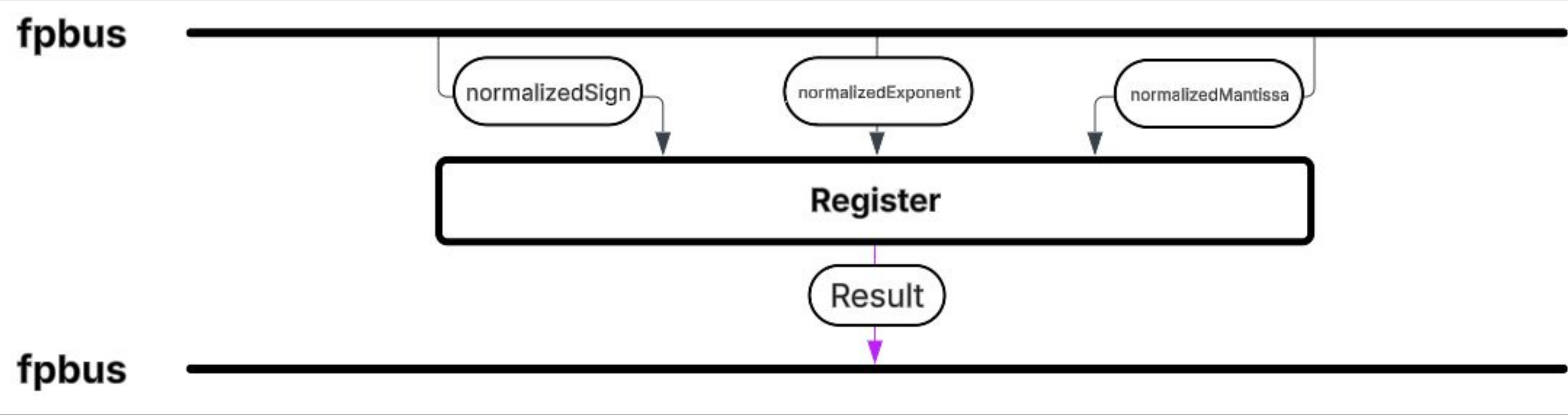
- Utilizes newly computed Sticky Bit, prior Sticky Bit, Round Bit, Guard Bit, and previous subtraction to accurately Round to Nearest "Even"
- Normalizes result by shifting and correcting Exponent
- Handles "carry outs" if mantissa shifts into the implicit one




fpbus
signA
signB
Azero
Asub
Ainf
ANaN
Bzero
Bsub
Binf
BNaN
bypassALU
Aex
Bex
A
B
alignedSign
carryOut
alignedResult
exponentOut
sticky
Shift Register + Counter + Combo Logic
mantissaOut
guard
round
stickyShift
shiftAmount
∞/NaN
0
Otherwise Bypassed ALU
Double Subnormal
normalizedMantissa
normalizedExponent
normalizedSign
Rounding Adder
roundCarry
shiftedMantissa
ALU
Register
fpbus

## Repacking Module (*Pack*)

- Very simple - concatenates normalizedSign, normalizedExponent, and normalizedMantissa into a single result for IEEE754 32-bit floating point representation



## Testing

- Every module was tested as we designed them to ensure no errors were made before proceeding (they still were!)

*Mask-*

- Basic functionality test with $2^{20}$ random 32-bit inputs

*Alignment-*

- Uses *Mask* outputs as inputs, with $2^{20}$ random test cases, including NaN, Infs, Subnormals, Zero and Normal Numbers to verify proper results and control signal assertion

## Testing

*ALU* -

- Determines expected exponent and mantissa separately, then checks those against *ALU* output (also computed using *Mask* and *Alignment*). Checks bypassALU signal to skip computation if necessary. Again, $2^{20}$ test cases.

*Normalize* -

- One static test for early functionality, and $2^{20}$ random tests, using all prior modules to compute all inputs to the *Normalize* module, and using *$shortrealtobits* for results checking.

# Testing

*Full System Testing-*

- 25 targeted tests cases (Case Analysis)
  - Done using a file, with hex value inputs for easy additions later on if deemed necessary
- $2^{25}$ constrained random non-NaN and non-Inf numbers
- $2^{25}$ random numbers including NaN and Inf
- Task for automatic results checking

# Bugs

Minor Issues

- Dozens of syntax related bugs in early development (some during later development as well)
- Improper handling of non-normal numbers (by IEEE-754 conventions), which were fixed during *Normalize*

Rounding Issues

1. We originally sampled GRS bits during the shifting process in *Alignment*, which resulted in rounding to ALWAYS be off by 1-3 (bits)
    - Moved to Normalize
2. Results still proved to be incorrect by 1-2 (bits)
    - Extended our ALU to use 32 bits, rather than the 26 we used previously (Mantissa + GRS), as a device capable of producing a 32-bit number almost definitely has 32-bit ALUs
    - Added second “Sticky Bit” still within alignment
3. Results were now consistently off by 1 bit
    - Added shift overflow check bit for 1’s lost with shifting, which accounted for any required subtraction that would usually be lost

## Results

- Accurate computation of “Normal” additions within precision
- Accurate rounding (adhering to IEEE-754 Round to Nearest “Even”)
- Handles special cases, Including NaN, +/- Inf, Subnormals (including those who graduate), etc
- Easily modifiable to support different rounding modes via loading different packages
- 0 Errors out of ~$2^{26}$ test cases

What would our next additions be?

- Implementation of more rounding modes
- 64-bit & 16-bit modes, which would be toggled via a parameter

## Resources

FP adder
FP converter

Floating Point Numbers and Rounding

IEEE 754 Floating Point Representation

ECE 571 & 586 Lecture Slides