Notes on IEEE 754 Floating-Point Number Format. Further details and examples can be found at http://weitz.de/ieee/ and https://en.wikipedia.org/wiki/IEEE_754

float16 (fewer resources, less accuracy)

The float16 library uses the IEEE 754 binary16 format for storing floating-point numbers:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+ -	exponent (+15)					mantissa (10 bits stored)									

The exponent is stored with a bias of +15, as per the standard.

EXAMPLES of float16		
Number	Binary Float	float16 hex and binary
0	0.0 x 2 ^ 0	0000 0000000000000000
1	1.0 x 2 ^ 0	3C00 0011110000000000
2	1.0 x 2 ^ 1	4000 0100000000000000
3.14	1.57 x 2 ^ 1	4248 0100001001001000
-100	-1.5625 x 2 ^ 6	D640 1101011001000000
inf	inf	7C00 0111110000000000
NaN	NaN	FFFF 1111111111111111

float32 (more accuracy)

The float32 library uses the IEEE 754 binary32 format for storing floating-point numbers:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+ -	exponent (+127)								mantissa (23 bits stored)																						

The exponent is stored with a bias of +127, as per the standard.

EXAMPLES of float32		
Number	Binary Float	float32 hex and binary
0	0.0 x 2 ^ 0	00000000 00000000000000000000000000000000
1	1.0 x 2 ^ 0	3F800000 00111111100000000000000000000000
2	1.0 x 2 ^ 1	40000000 01000000000000000000000000000000
3.1415927	1.5707964 x 2 ^ 1	40490FDB 01000000010010010000111111011011
-100	-1.5625 x 2 ^ 6	C2C80000 11000010110010000000000000000000
inf	inf	7F800000 01111111100000000000000000000000
NaN	NaN	FFFFFFFF 11111111111111111111111111111111

Each library provides algorithms for conversion between floating-point representation and integers, addition/subtraction, multiplication, division, square root and basic comparisons.

Numbers that are too large to store in the relevant format return the largest possible number in the relevant format, and too small to store, return zero. Errors, such as divide by zero, return INF or NaN as appropriate.

Usage

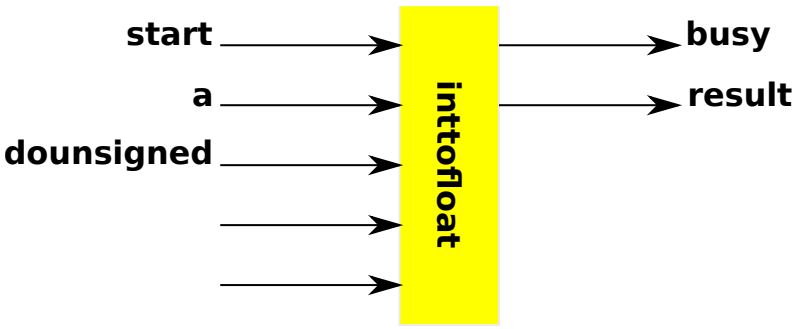
For all conversion and calculation algorithms in the library, hold “start” to 1 for 1 clock, and wait for “busy” to return to 0. “a” represents the first operand, “b” the second operand, and “result” is the result in the appropriate format. “addsub” and “dounsigned” are explained in the appropriate section when used.

For the comparisons, there are outputs for the result of the three comparisons, 1 for true, 0 for false.

NOTE: Due to the same name being used for the algorithms float16 and float32 cannot be used in the same project.

inttofloat

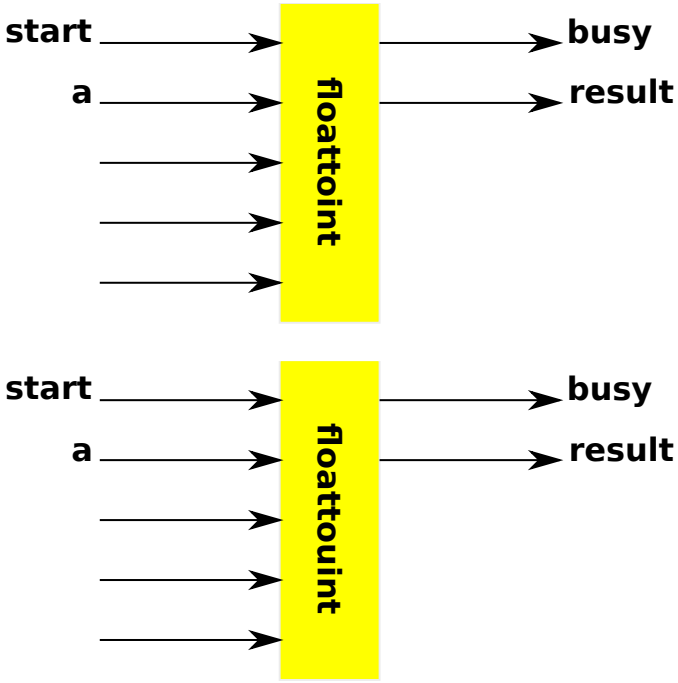
Converts integers (16 bit for float16, 32 bit for float32) to floating-point.



Signal	Meaning
“start”	Start the conversion by holding to 1 for 1 clock cycle.
“a”	Integer to convert to floating point (16 bit for float16, 32 bit for float32).
“dounsigned”	Set to 1 to treat a as an unsigned integer.
“busy”	Set to 1 whilst the conversion takes place.
“result”	Result of the conversion of a into floating point as float16 or float32.

floattoint and floattouint

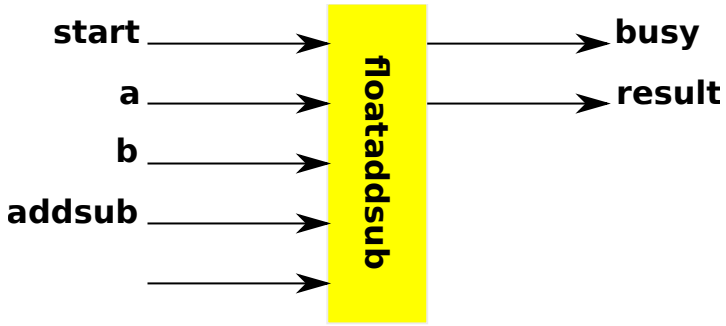
Converts floating point numbers (16 bit for float16, 32 bit for float32) to integers. floattoint converts floating point numbers to signed integers. floattouint converts floating point numbers to unsigned integers, with negative numbers returning 0.



Signal	Meaning
“start”	Start the conversion by holding to 1 for 1 clock cycle.
“a”	Floating point number to convert to an integer.
“busy”	Set to 1 whilst the conversion takes place.
“result”	Result of the conversion of a into an integer.

floataddsub

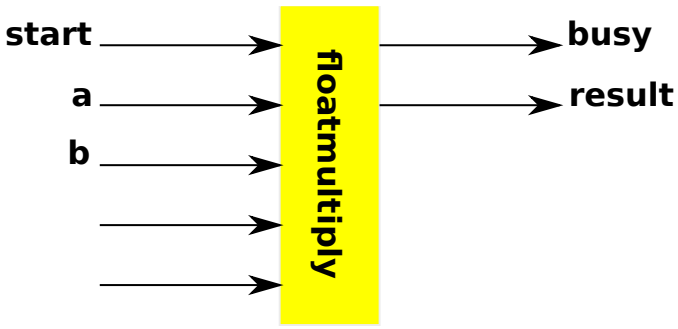
Performs addition or subtraction (16 bit for float16, 32 bit for float32) of two floating point numbers.



Signal	Meaning
“start”	Start the additiion or subtraction by holding to 1 for 1 clock cycle.
“a”	First floating point operand.
“b”	Second floating point operand.
“addsub”	Control when 0 do addition, or 1 do subtraction.
“busy”	Set to 1 whilst the operation takes place.
“result”	Result of the “addsub = 0” a +b, or “addsub = 1” a – b.

floatmultiply

Performs multiplication (16 bit for float16, 32 bit for float32) of two floating point numbers.

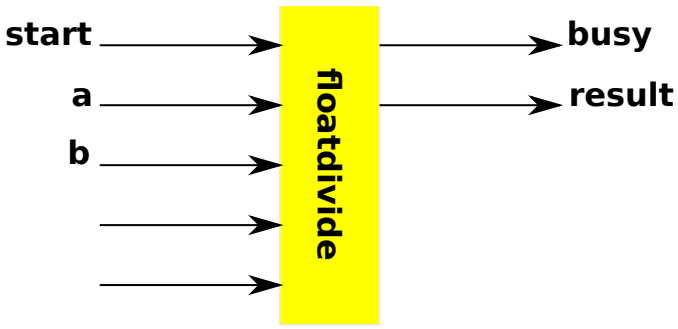


Signal	Meaning
“start”	Start the multiplication by holding to 1 for 1 clock cycle.
“a”	First floating point operand.
“b”	Second floating point operand.
“busy”	Set to 1 whilst the operation takes place.
“result”	Result of the a * b.

NOTE: The multiplication as written expects yosys to infer DSP multipliers. The algorithm douintmul within float16.ice or float32.ice performs the actual multiplication, as is configured for 18 bit DSP multipliers, known to work on the ULX3S and DE10NANO FPGA boards that I own.

floatdivide

Performs division (16 bit for float16, 32 bit for float32) of two floating point numbers.

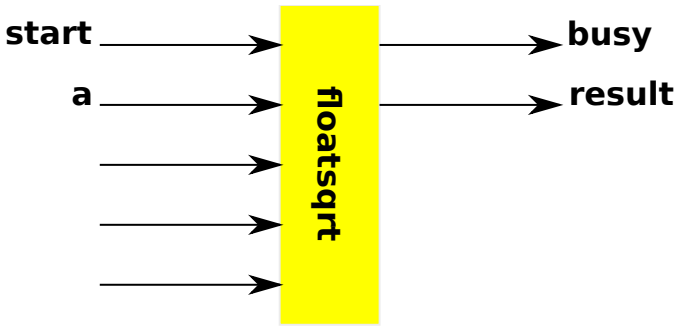


Signal	Meaning
“start”	Start the division by holding to 1 for 1 clock cycle.
“a”	First floating point operand.
“b”	Second floating point operand.
“busy”	Set to 1 whilst the operation takes place.
“result”	Result of the a / b.

floatsqrt

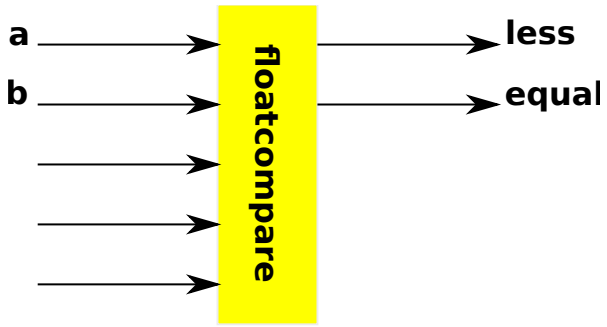
Adapted from https://projectf.io/posts/square-root-in-verilog/

Performs square root (16 bit for float16, 32 bit for float32) of a floating point number.



Signal	Meaning
“start”	Start the square root by holding to 1 for 1 clock cycle.
“a”	Floating point number to square root.
“busy”	Set to 1 whilst the operation takes place.
“result”	Result of the √a.

Comparisons: floatcompare



Adpated from Berkeley SoftFloat https://github.com/ucb-bar/berkeley-softfloat-3

Performs comparisons (16 bit for float16, 32 bit for float32) of two floating point numbers. For flexibility, the individual circuits for each comparison are available, plus an extra circuit for <=.

Signal	Meaning
“a”	First floating point operand.
“b”	Second floating point operand.
“lessthan”	Returns 1 if a < b.
“equalto”	Returns 1 if a == b.