# Floating-Point

## Overview

The floating-point cores provide basic floating-point operations including addition, subtraction, multiplication, division, square root and float to integer and integer to float conversions. Multiple floating-point precisions are supported. Some common sample precisions are in the table below.

|  |  |
| --- | --- |
| Precision | Bits |
| Half | 16 |
| Single | 32 |
| Double | 64 |
| Double Extended | 80 |
| Quad | 128 |

Other precisions are possible but there has been no testing on the following:

|  |  |
| --- | --- |
| Precision | Bits |
| Two-Thirds | 24 |
| Half Double Extended | 40 |
| Triple half | 48 |
| Gambit | 52 |
| Triple | 96 |

The code is parameterized and virtually any precision is possible with some minor modifications to the code, which include modifying the EMSB and FMSB local parameters.

## Operations Supported

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Module | Test Bench | Test Vectors File |
| Addition / Subtraction | fpAddsub.v | fpAddsub\_tb.v | fpAddsub\_tv.txt |
| Multiplication | fpMul.v | fpMul\_tb.v | fpMul\_tv.txt |
| Division | fpDiv.v | fpDiv\_tb.v | fpDiv\_tv.txt |
| Square root | fpSqrt.v | fpSqrt\_tb.v | fpSqrt\_tv.txt |
| Integer to Float | i2f.v |  |  |
| Float to Integer | f2i.v |  |  |
| Normalization | fpNormalize.v |  |  |
| Rounding | fpRound.v |  |  |

**Representation**

The floating-point format is an IEEE-754 representation. The representation has the sign bit as it’s most significant bit followed by the sign of the exponent, followed by the exponent, and finally the mantissa. The mantissa is a magnitude with the most significant bit hidden. (The most significant bit of the mantissa would always be one, except for denormalized numbers.) The format makes it possible to perform many comparisons using integer instructions as well as floating-point ones.

Briefly, the double precision format is outlined below, single precision and other precisions are similar.

**Double Precision Format:**

|  |  |  |  |
| --- | --- | --- | --- |
| 63 | 62 | 61 52 | 51 0 |
| SM | SE | Exponent | Mantissa |

SM – sign of mantissa

SE – sign of exponent

The sign bit of the exponent is inverted compared to the standard two’s complement representation.

|  |  |
| --- | --- |
| SeEEEEEEEEEE |  |
| 11111111111 | Maximum exponent |
| …. |  |
| 01111111111 | exponent of zero |
| …. |  |
| 00000000000 | Minimum exponent |

The exponent ranges from -1024 to +1023 for double precision numbers

The following table lists the sizes of the exponents and mantissa for various formats:

|  |  |  |  |
| --- | --- | --- | --- |
| Precision | Exp. | Man. | Decimal Digits |
| 16 | 5 | 10 | 3.0 |
| 24 | 7 | 16 | 4.8 |
| 32 | 8 | 23 | 6.9 |
| 40 | 10 | 29 | 8.7 |
| 48 | 12 | 35 | 10.5 |
| 52 | 11 | 40 | 12.0 |
| 64 | 11 | 52 | 15.7 |
| 80 | 15 | 64 | 19.3 |
| 96 | 15 | 80 | 24.0 |
| 128 | 15 | 112 | 33.7 |

## Intermediate Representation

For an intermediate representation, the cores use a format similar to the IEEE format. The most significant bit is the sign bit, the next bit the sign of the exponent, followed by the exponent, then mantissa, just as in the IEEE format. However the mantissa is doubled in width and has two leading decimal places without a hidden bit. This intermediate format is output by all the core’s primitives and is the input to the normalization unit.

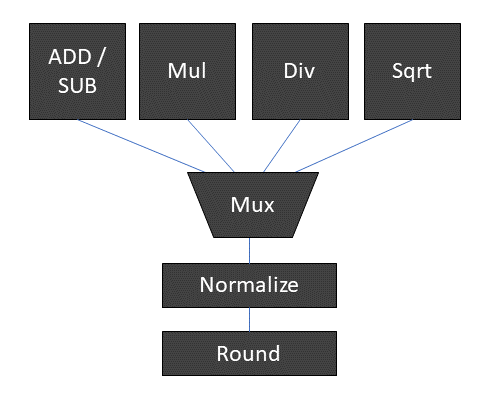
Intermediate Format

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 118 | 117 | 116 106 | 105 104 | 103 0 |
| SM | SE | Exponent | D2 | Mantissa fraction |

The core calculates to double the normal mantissa width (some components do not calculate to the full double width precision but instead compute a few extra bits beyond the normal precision). Output from primitives are then normalized and rounded to a lower precision. All the primitives output in the same intermediate format so that a single normalizer may be shared among multiple primitives.

## Floating Point Unit Organization

Following is an illustration of a suggested floating-point unit organization. This organization shares the normalization and rounding units to reduce the size of the floating-point unit.



## Pipelining

Add / subtract and multiply are fully pipelined and may begin a new operation on each clock cycle. Divide and square root take multiple clock cycles to execute. A new operation should not begin on the divider or square root core before the previous operation has finished.

## Latencies

The floating-point cores are optimized for low latency and not high clock frequency. For instance for addsub and multiply the latency of the primitive is only two clock cycles prior to normalization. They may be used at a higher clock frequency by taking into consideration the delays in producing an output. For instance the system may run at double the clock frequency, but the fp-cores could be fed a ½ frequency clock for use. New fp operations would be issued every other clock cycle then. When interfacing to a higher frequency clock the fp operations must be spread out over multiple clock cycles.

## Add / Subtract

Addition and subtraction are performed by the same component as the operation is virtually identical when differing operand signs are taken into consideration. An operation selection bit is fed to the addsub unit to select either addition or subtraction.

## Multiply

A multiply operation inherently produces double the selected precision output bits.

## Divider

The divider uses a radix 16 integer divide component. This component provides four quotient bits per clock cycle. The divider computes to double the precision selected.

## Square Root

Square root uses the standard long-hand division algorithm producing a single bit of the result per clock cycle. Note that two bits of the operand are consumed per clock cycle. Results are computed out to double the selected precision.

## NaN Values

The cores generate NaN (not-a-number) values in some circumstances. NaN values are propagated by the core. The exact values used for various NaN’s depend on the precision and are defined in fp\_defines.v. The least significant bits of the values are in common among all precisions and listed in the table below.

|  |  |
| --- | --- |
| Non-Number | NaN Value |
| Subtraction of infinities | ---1 |
| Infinity / Infinity | ---2 |
| zero / zero | ---3 |
| Infinity \* zero | ---4 |
| square root of infinity | ---5 |
| square root of negative number | ---6 |