fp_add_sub

Floating-Point Adder/Subtractor

January 1996, ver. 1 Functional Specification 2

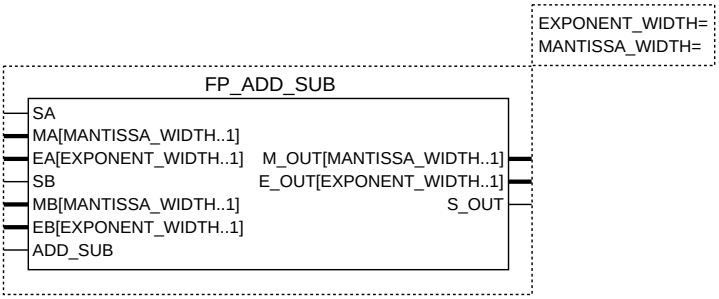
Features

- `fp_add_sub` reference design implementing a floating-point adder/subtractor
- Parameterized mantissa and exponent widths
- Optimized for FLEX 10K and FLEX 8000 device families
- High-speed operation
- Supported by schematic and text design entry methods, including VHDL, Verilog HDL, and the Altera Hardware Description Language (AHDL)

General Description

The `fp_add_sub` reference design implements a floating-point adder/subtractor with parameterized input widths. This function uses sign-mantissa-exponent notation with parameterized mantissa and exponent widths. See Figure 1.

Figure 1. fp_add_sub Symbol



Function Prototype

The AHDL Function Prototype for the `fp_add_sub` function is shown below:

```
FUNCTION fp_add_sub (sa, ma[mantissa_width..1], ea[exponent_width..1],
      sb, mb[mantissa_width..1], eb[exponent_width..1], add_sub)
  WITH (mantissa_width, exponent_width)
  RETURNS (m_out[mantissa_width..1], e_out[exponent_width..1],
      s_out);
```

Parameters

Parameters for the `fp_add_sub` function are provided in Table 1.

Table 1. fp_add_sub Parameters

Name	Default	Value	Description
`exponent_width`	7	Integers only	Width of all exponents (in bits)
`mantissa_width`	8	Integers only	Width of all mantissas (in bits)

Ports

Input and output ports for the `fp_add_sub` function are described in Table 2.

Table 2. Input & Output Ports

Port Type	Name	Description
Input	`sa`	Sign bit for the a input: 1 = positive, 0 = negative
Input	`ma[mantissa_width..1]`	Mantissa for the a input
Input	`ea[exponent_width..1]`	Exponent for the a input
Input	`sb`	Sign bit for the b input: 1 = positive, 0 = negative
Input	`mb[mantissa_width..1]`	Mantissa for the b input
Input	`eb[exponent_width..1]`	Exponent for the b input
Input	`add_sub`	Operation: 1 = add, 0 = subtract
Output	`m_out[mantissa_width..1]`	Mantissa for the output
Output	`e_out[exponent_width..1]`	Exponent for the output
Output	`s_out`	Sign bit for the output: 1 = positive, 0 = negative

Functional Description

Addition and subtraction are complex operations for floating-point numbers. With floating-point multiplication and division, the mantissa must be post-normalized. With floating-point addition and subtraction, however, the mantissas must be preprocessed so that the exponents are equal. For more information, go to "Floating-Point Addition & Subtraction" on page 4 of this functional specification.

In floating-point functions, the sign bit represents the sign of the mantissa: 1 for positive and 0 for negative. The mantissa is a positive number less than 1. A 0 is implied to the left of the binary point. After normalization, the most significant bit (MSB) is always 1. The exponent is in excess $2^{(n-1)}$ notation, where n is the number of bits in the exponent.

For example, the binary representation of the number 0.75×2^{1} is shown below. This example assumes 8 bits for the mantissa (M) and 7 bits for the exponent (E). S represents the sign bit.

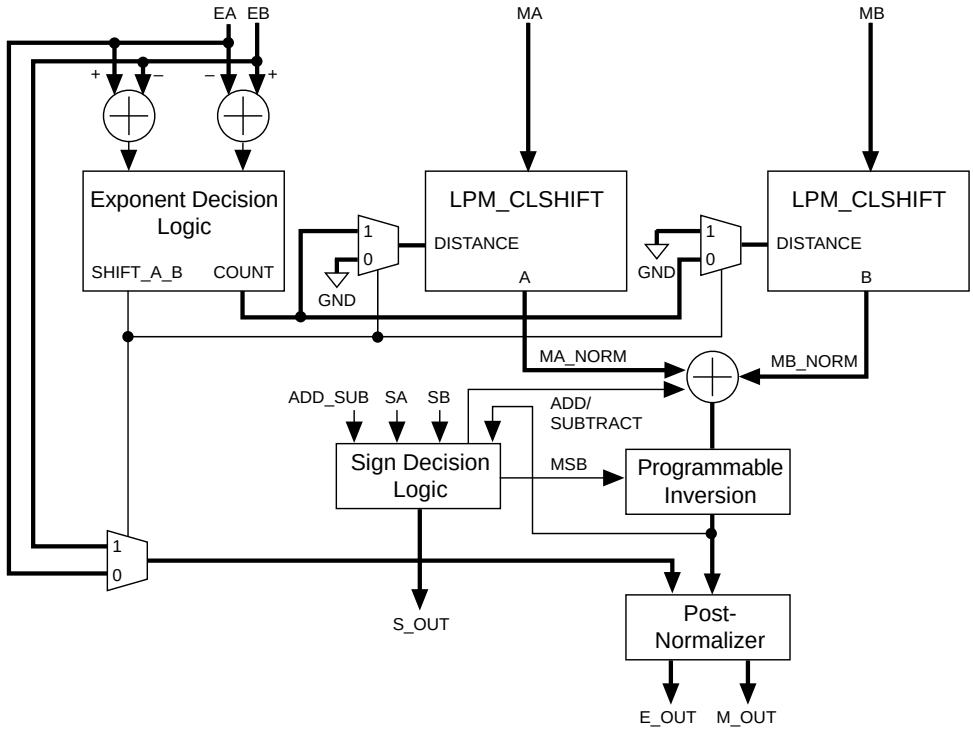
$S = 1$, $M = 11000000$, $E = 1000001$

Similarly, the binary representation of the number 0.625×2^{-1} is:

$S = 1$, $M = 10100000$, $E = 0111111$

Figure 2 shows the block diagram of the `fp_add_sub` floating-point adder.

Figure 2. fp_add_sub Block Diagram



Floating-Point Addition & Subtraction

To add or subtract two floating-point numbers, the mantissas must be aligned. Then, the exponents must be compared to determine which number is larger. If the difference between the exponents is slight, the number with the smaller exponent may be larger if mantissas MA and MB are not normalized, which reduces the function's precision. To avoid this problem, designers should ensure that all inputs are normalized by making sure that the MSB of each mantissa is 1.

The relative values of the exponents are checked by subtracting one exponent from the other. The mantissa with the larger exponent is retained, and the mantissa with the smaller exponent is right-shifted until the radix point is properly aligned (i.e., until the exponents are equal). If the exponents differ by more than the number of bits in the mantissa, the smaller number becomes insignificant. The shifting is performed by the LPM function `lpm_clshift`.

After the mantissas pass through the shifters, an unsigned integer adder/subtractor performs an operation that is determined by the sign of the inputs (`sa` and `sb`) and the `add_sub` port. The result of the adder is then passed through a programmable inverter, which is controlled by the sign decision logic. This process ensures that the mantissa has the proper sign. After the addition or subtraction has taken place, the post-normalizer normalizes the result, if necessary, by adjusting the mantissa and exponent of the result so that the MSB of the mantissa is 1.

Examples of floating-point addition and subtraction for 8-bit mantissa, 7-bit exponent floating-point numbers are provided below.

Example 1: Positive Number Plus Positive Number

$$
\begin{aligned}
& 0.3046875 \times 2^{45} && + && +0.34375 \times 2^{44} \\
= \; & +0.01001110 \times 2^{45} && + && +0.01011000 \times 2^{44} \\
= \; & +0.01001110 \times 2^{45} && + && +0.00101100 \times 2^{45} \\
= \; & +0.01111010 \times 2^{45} \\
= \; & +0.11110100 \times 2^{44} \\
= \; & +0.953125 \times 2^{44} && = && +1.677 \times 10^{13}
\end{aligned}
$$

Example 2: Negative Number Plus Positive Number

$$
\begin{aligned}
& -0.82421875 \times 2^{76} && + && +0.25390625 \times 2^{75} \\
= \; & -0.11010011 \times 2^{76} && + && +0.01000001 \times 2^{75} \\
= \; & -0.11010011 \times 2^{76} && + && +0.00100000 \times 2^{76} \\
= \; & -0.10110011 \times 2^{76} \\
= \; & -0.69921875 \times 2^{76} && = && -5.2831 \times 10^{22}
\end{aligned}
$$

In example 2, the mantissa shift causes a loss of precision.

Example 3: Negative Number Plus Insignificant Positive Number

$$
\begin{aligned}
& -0.5 \times 2^{89} && + && +0.5 \times 2^{68} \\
= \; & -0.10000000 \times 2^{89} && + && +0.10000000 \times 2^{68} \\
= \; & -0.10000000 \times 2^{89} && + && +0.00000000 \times 2^{89} \\
= \; & -0.10000000 \times 2^{89} \\
= \; & -0.5 \times 2^{89}
\end{aligned}
$$

In example 3, the result is the same as the larger input value because 0.5×2^{68} is insignificant compared to -0.5×2^{89}.

Floating-Point Representation

Floating-point numbers can be represented by many different notations. The `fp_add_sub` function uses an implied leading zero for the mantissa, with an unsigned, *m*-bit mantissa and *n*-bit exponent, where *m* = `mantissa_width` and *n* = `exponent_width`. A separate sign bit is used to represent the sign of the mantissa.

The following examples of an 8-bit positive mantissa and a 7-bit exponent assume `mantissa_width` = 8 and `exponent_width` = 7. The numbers shown in Table 3 below should be adjusted accordingly if different parameter values are used.

An 8-bit positive mantissa allows fractions with numerators between 0 and 255. The implied leading zero limits the range of the mantissa between 0 and 0.9961, and the separate sign bit allows the mantissa to have a value between –0.9961 and +0.9961. Because the mantissa is in fractional form, a greater number of bits in the mantissa does not result in a larger mantissa, but offers greater precision. Table 3 lists examples of 8-bit mantissas with implied leading zeros.

Table 3. 8-Bit Mantissa Examples

Mantissa	Implied Zero	Binary Fraction	Decimal Fraction	Decimal
11001110	0.11001110	11001110 / 100000000	206 / 256	0.80469
00001100	0.00001100	00001100 / 100000000	12 / 256	0.04688
10100001	0.10100001	10100001 / 100000000	161 / 256	0.62891

For a 7-bit exponent, the exponent is represented in excess 64 format, i.e., for an *n*-bit exponent, the representation is excess $2^{(n-1)}$. Excess (offset) format allows both negative and positive exponents to be represented with positive numbers, which results in simpler calculations for exponent handling. To represent an exponent in excess $2^{(n-1)}$ format, add $2^{(n-1)}$ to the value of the exponent. For example, in excess 64 format, 64 is added to the actual exponent; thus, the maximum value for the exponent is +63, and the minimum value is –64. An exponent of 10 is represented as 74, and an exponent of –10 is represented as 54. The exponent 0 is represented as 64.

The following examples represent 8-bit mantissa, 7-bit exponent floating-point numbers. In this section, the subscripts "b" and "d" indicate that the number is a binary or a decimal number, respectively.

Example 1: Largest Positive Number

$+11111111_b\ 1111111_b$

$= +0.11111111_b \times 2^{(1111111_b - 1000000_b)}$

$= +0.11111111_b \times 2^{0111111_b}$

$= +0.11111111_b \times 2^{63_d}$

$= +11111111.0_b \times 2^{55_d}$

$= +255_d \times 2^{55_d}$

$= +9.187343239836e \times 10^{18}$

Example 2: Largest Negative Number

$-11111111_b\ 1111111_b$

$= -9.187343239836e \times 10^{18}$

Example 3: Smallest Number (Closest to Zero)

$\pm 10000000_b\ 0000000_b$

$= \pm 0.10000000_b \times 2^{(0000000_b - 1000000_b)}$

$= \pm 0.10000000_b \times 2^{(0_d - 64_d)}$

$= \pm 0.10000000_b \times 2^{-64_d}$

$= \pm 10000000.0_b \times 2^{-72_d}$

$= \pm 128_d \times 2^{-72_d}$

$= \pm 2.710505431214e \times 10^{-20}$

Example 4: Typical Value

$$
\begin{aligned}
&-11000111\ 1001001 \\
&= -0.11000111_{b} \times 2^{(1001001_{b} - 1000000_{b})} \\
&= -0.11000111_{b} \times 2^{1001_{b}} \\
&= -0.11000111_{b} \times 2^{9_{d}} \\
&= -11000111.0_{b} \times 2^{1_{d}} \\
&= -199_{d} \times 2 \\
&= -398
\end{aligned}
$$

2610 Orchard Parkway
San Jose, CA 95134-2020
(408) 894-7000
Applications Hotline:
(800) 800-EPLD
Customer Marketing:
(408) 894-7104
Literature Services:
(408) 894-7144