LogiCORE IP Floating-Point Operator v5.0

DS335 March 1, 2011 **Product Specification**

Introduction

The Xilinx® Floating-Point core provides designers with the means to perform floating-point arithmetic on an FPGA device. The core can be customized for operation, word length, latency, and interface.

Features

- Available for Kintex™-7, Virtex®-7, Virtex-6, Virtex-5, Virtex-4, Spartan®-6, Spartan-3/XA, Spartan-3E/XA, Spartan-3A/AN/3A DSP/XA FPGAs
- Supported operators:
  - multiply
  - add/subtract
  - divide
  - square-root
  - comparison
  - conversion from floating-point to fixed-point
  - conversion from fixed-point to floating-point
  - conversion between floating-point types
- Compliance with *IEEE-754 Standard* (with only minor documented deviations)
- Parameterized fraction and exponent wordlengths
- Use of XtremeDSP™ slice for multiply
- Use of XtremeDSP slice for single and double precision add/subtract operations on Virtex and Kintex families
- Optimizations for speed and latency
- Fully synchronous design using a single clock
- For use with CORE Generator™ software which is available in the Xilinx ISE® 13.1 software

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family[1]	Virtex-7, Kintex-7, Virtex-6, Virtex-5, Virtex-4, Spartan®-6, Spartan-3/XA, Spartan-3E/XA, Spartan-3A/3AN/3A DSP/XA
Supported User Interfaces	N/A
Provided with Core	
Documentation	Product Specification
Design Files	Netlist
Example Design	Not Provided
Test Bench	Not Provided
Constraints File	N/A
Simulation Model	Verilog, VHDL
Tested Design Tools	
Design Entry Tools	CORE Generator 13.1
Simulation	Mentor Graphics ModelSim 6.6d, Cadence Incisive Enterprise Simulator (IES) 10.2, Synopsys VCS and VCS MX 2010.06, ISIM 13.1
Synthesis Tools	Not Provided
Support	
Provided by Xilinx, Inc.	

1. For a complete listing of supported devices, see the release notes for this core.

Overview

The Xilinx Floating-Point core allows a range of floating-point arithmetic operations to be performed on FPGA. The operation is specified when the core is generated, and each operation variant has a common interface. This interface is shown in Figure 1. When a user selects an operation that requires only one operand, the B input is omitted.

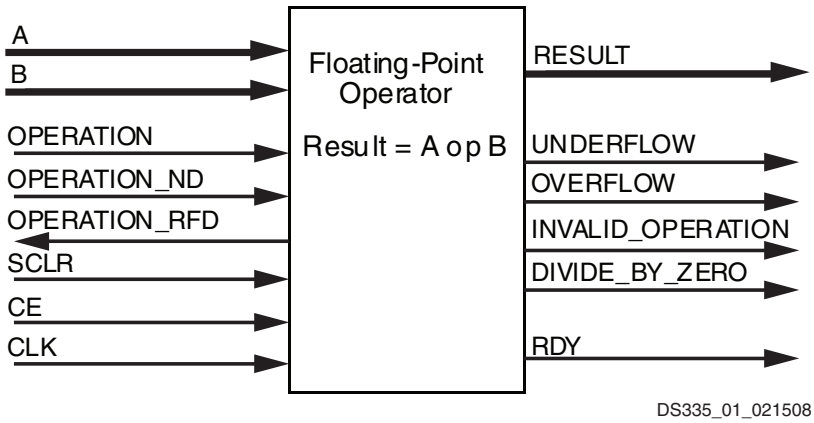

Figure 1: **Block Diagram of Generic Floating-Point Binary Operator Core**

Functional Description

The floating-point and fixed-point representations employed by the core are described in Floating-Point Number Representation and Fixed-Point Number Representation.

Floating-Point Number Representation

The core employs a floating-point representation that is a generalization of the *IEEE-754 Standard* to allow for non-standard sizes [Ref 1]. When standard sizes are chosen, the format and special values employed are identical to those described by the *IEEE-754 Standard*.

Two parameters have been adopted for the purposes of generalizing the format employed by the Floating-Point core. These specify the total format width and the width of the fractional part. For standard single precision types, the format width is 32 bits and fraction width 24 bits. In the following description, these widths are abbreviated to w and w_f, respectively.

A floating-point number is represented using a sign, exponent, and fraction (which are denoted as 's,' 'E,' and $b_0.b_1b_2\ldots b_{w_f-1}$, respectively).

The value of a floating-point number is given by: $v = (-1)^s 2^E b_0.b_1b_2\ldots b_{w_f-1}$

The binary bits, b_i, have weighting 2^{-i}, where the most significant bit b_0 is a constant 1. As such, the combination is bounded such that $1 \leq b_0.b_1b_2\ldots b_{p-1} < 2$ and the number is said to be normalized. To provide increased dynamic range, this quantity is scaled by a positive or negative power of 2 (denoted here as E). The sign bit provides a value that is negative when $s = 1$, and positive when $s = 0$.

The binary representation of a floating-point number contains three fields as shown in Figure 2.

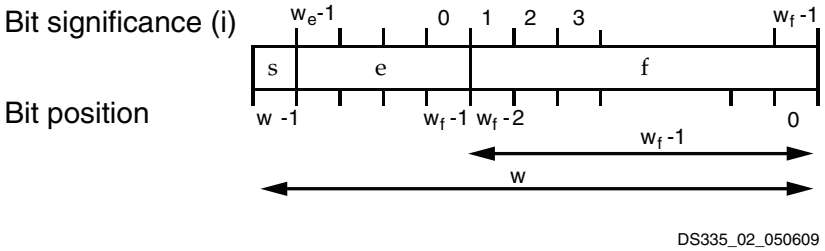

Figure 2: **Bit Fields within the Floating-Point Representation**

As b_0 is a constant, only the fractional part is retained, that is, $f = b_1 \ldots b_{w_f - 1}$. This requires only $w_f - 1$ bits. Of the remaining bits, one bit is used to represent the sign, and $w_e = w - w_f$ bits represent the exponent.

The exponent field, e, employs a biased unsigned integer representation, whose value is given by:

$$e = \sum_{i=0}^{w_e - 1} e_i 2^i$$

The index, i, of each bit within the exponent field is given in Figure 2.

The signed value of the exponent, E, is obtained by removing the bias, that is, $E = e - (2^{w_e - 1} - 1)$.

In reality, w_f is not the wordlength of the fraction, but the fraction with the hidden bit, b_0, included. This terminology has been adopted to provide commonality with that used to describe fixed-point parameters (as employed by Xilinx System Generator™ for DSP).

Special Values

A number of values for s, e and f have been reserved for representing special numbers, such as Not a Number (NaN), Infinity (∞), Zero (0), and denormalized numbers (see Denormalized Numbers for an explanation of the latter). These special values are summarized in Table 1.

Table 1: **Special Values**

Symbol for Special Value	s Field	e Field	f Field
NaN	don't care	2^{w_e} -1 (that is, $e = 11 \ldots 11$)	Any non-zero field. For results that are NaN the most significant bit of fraction is set (that is, $f = 10 \ldots 00$)
$\pm\infty$	sign of ∞	2^{w_e} -1 (that is, $e = 11 \ldots 11$)	Zero (that is, $f = 00 \ldots 00$)
± 0	sign of 0	0	Zero (that is, $f = 00 \ldots 00$)
denormalized	sign of number	0	Any non-zero field

Note that in Table 1 the sign bit is undefined when a result is a NaN. The core generates NaNs with the sign bit set to 1 (that is, negative). Also, infinity and zero are signed. Where possible, the sign is handled in the same way as finite non-zero numbers. For example, $-0 + (-0) = -0$, $-0 + 0 = 0$ and $-\infty + (-\infty) = -\infty$. A meaningless operation such as $-\infty + \infty$ raises an invalid operation exception and produces a NaN as a result.

IEEE-754 Support

The Xilinx Floating-Point core complies with much of the *IEEE-754 Standard*. The deviations generally provide a better trade-off of resources against functionality. Specifically, the core deviates in the following ways:

- Non-Standard Wordlengths
- Denormalized Numbers
- Rounding Modes
- Signaling and Quiet NaNs

Non-Standard Wordlengths

The Xilinx Floating-Point core supports a greater range of fraction and exponent wordlength than defined in the *IEEE-754 Standard*.

Standard formats commonly implemented by programmable processors:

- **Single Format** – uses 32 bits, with a 24-bit fraction and 8-bit exponent.
- **Double Format** – uses 64 bits, with 53-bit fraction and 11-bit exponent.

Less commonly implemented standard formats are:

- **Single Extended** – wordlength extensions of 43 bits and above
- **Double Extended** – wordlength extensions of 79 bits and above

The Xilinx core supports formats with fraction and exponent wordlengths outside of these standard wordlengths.

Denormalized Numbers

The exponent limits the size of numbers that may be represented. It is possible to extend the range for small numbers using the minimum exponent value (0) and allowing the fraction to become denormalized. That is, the hidden bit b_0 becomes zero such that $b_0.b_1b_2\ldots b_{p-1} < 1$. Now the value is given by:

$$v = (-1)^s 2^{-\left(2^{w_e-1}-2\right)} 0.b_1b_2\ldots b_{w_f-1}$$

These denormalized numbers are extremely small. For example, with single precision the value is bounded $|v| < 2^{-126}$. As such, in most practical calculation they do not contribute to the end result. Furthermore, as the denormalized value becomes smaller, it is represented with fewer bits and the relative rounding error introduced by each operation is increased.

The Xilinx Floating-Point core does not support denormalized numbers. In FPGAs, the dynamic range can be increased using fewer resources by increasing the size of the exponent (and a 1-bit increase for single precision increases the range by 2^{256}). If necessary, the overall wordlength of the format can be maintained by an associated decrease in the wordlength of the fraction.

To provide robustness, the core treats denormalized operands as zero with a sign taken from the denormalized number. Results that would have been denormalized are set to an appropriately signed zero.

The support for denormalized numbers cannot be switched off on some processors. Therefore, there may be very small differences between values generated by the Floating-Point core and a program running on a conventional processor when numbers are very small. If such differences must be avoided, the arithmetic model on the conventional processor should include a simple check for denormalized numbers. This check should set the output of an operation to zero when denormalized numbers are detected to correctly reflect what happens in the FPGA implementation.

Rounding Modes

Only the default rounding mode, Round to Nearest (as defined by the *IEEE-754 Standard*), is currently supported.

Signaling and Quiet NaNs

The *IEEE-754 Standard* requires provision of Signaling and Quiet NaNs. However, the Xilinx Floating-Point core treats all NaNs as Quiet NaNs. When any NaN is supplied as one of the operands to the core, the result is a Quiet NaN, and an invalid operation exception will not be raised (as would be the case for signaling NaNs). The exception to this rule is floating-point to fixed-point conversion. For detailed information, see the behavior of INVALID_OP.

Fixed-Point Number Representation

For the purposes of fixed-point to floating-point conversion, a fixed-point representation is adopted that is consistent with the signed integer type used by Xilinx System Generator for DSP. Fixed-point values are represented using a two's complement number that is weighted by a fixed power of 2. The binary representation of a fixed-point number contains three fields as shown in Figure 3 (although it is still simply a weighted two's complement number).

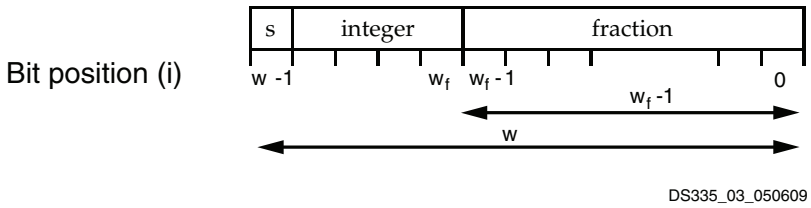

Figure 3: **Bit Fields within the Fixed-Point Representation**

In Figure 3, the bit position has been labeled with an index i. Based upon this, the value of a fixed-point number is given by:

$$v = (-1)^s 2^{w-1-w_f} + b_{w-2} \ldots b_{w_f} . b_{w_f-1} \ldots b_1 b_0$$

$$= (-1)^{b_{w-1}} 2^{w-1-w_f} + \sum_{0}^{w-2} 2^{i-w_f} b_i$$

For example, a 32-bit signed integer representation is obtained when a total width of 32 and a fraction width of 0 are specified. Round to Nearest is employed within the conversion operations.

To provide for the sign bit, the width of the integer field must be at least 1, requiring that the fractional width be no larger than w-1.

Port Description

The ports employed by the core were shown in Figure 1. They are described in more detail in Table 2. All control signals are active High.

Table 2: **Core Ports**

Name	Width	Direction	Description
A[1]	w	INPUT	Operand A
B[1]	w	INPUT	**Operand B:** Only present on binary operation.
OPERATION[1]	6	INPUT	**Operation:** Specifies the operation to be performed. Implemented when the core is configured for both add and subtract operations, or as a programmable comparator.
OPERATION_ND	1	INPUT	**New Data**: Must be set High to indicate that operand A, operand B and OPERATION, the latter when required as described above, are valid.
OPERATION_RFD	1	OUTPUT	**Ready For Data**: Set High by core to indicate that it is ready for new operands.
SCLR	1	INPUT	**Synchronous Reset** (optional): This resets control path and not the data path (that is, only RDY and OPERATION_RFD outputs are affected). SCLR takes priority over CE.
CE	1	INPUT	**Clock Enable** (optional)
CLK	1	INPUT	Clock
RESULT	w	OUTPUT	**Result Output**: Result of operation.
UNDERFLOW	1	OUTPUT	**Underflow:** Set High by core when underflow occurs. Supplied in synchronism with associated RESULT.
OVERFLOW	1	OUTPUT	**Overflow:** Set High by core when overflow occurs. Supplied in synchronism with associated RESULT.
INVALID_OP	1	OUTPUT	**Invalid Operation:** Set High by core when operands cause an invalid operation. Supplied in synchronism with associated RESULT.
DIVIDE_BY_ZERO	1	OUTPUT	**Divide By Zero:** Set High by a divide operation to indicate that a division by zero was performed. Supplied in synchronism with associated RESULT.
RDY	1	OUTPUT	**Output Ready**: Set High by core when RESULT is valid.

[1] A, B, and OPERATION are not registered on the input to the core, as the assumption is that they will be coming from registered outputs of other cores. Should this be required, registers can be added to these inputs externally to the core.

A

Operand A input.

B

Operand B input.

CLK

All signals are synchronous to the CLK input.

CE

When CE is deasserted, the clock is disabled, and the state of the core and its outputs are maintained. Note that SCLR takes priority over CE.

SCLR

When SCLR is asserted, the core control circuits are synchronously set to their initial state. Any incomplete results are discarded, and RDY will not be generated for them. While SCLR is asserted both OPERATION_RFD and RDY are synchronously deasserted. The core is ready for new input one cycle after SCLR is deasserted, at which point OPERATION_RFD is asserted. SCLR takes priority over CE. If SCLR is required to be gated by CE, then this can be done externally to the core.

OPERATION

OPERATION is present when add and subtract operations are selected together, or when a programmable comparator is selected. The operations are binary encoded as specified in Table 3.

Table 3: **Encoding of OPERATION**

FP Operation		OPERATION(5 downto 0)
Add		000000
Subtract		000001
Compare (Programmable)	Unordered	000100
	Less Than	001100
	Equal	010100
	Less Than or Equal	011100
	Greater Than	100100
	Not Equal	101100
	Greater Than or Equal	110100

OPERATION_ND should be asserted when operands are valid on inputs A and B and the FP Operation is valid on OPERATION (should it be required). Deasserting OPERATION_ND prevents the initiation of new operations and the subsequent assertion of RDY.

Note: OPERATION_ND is required to synchronize operations when the core is configured to perform a multi-cycle divide or square root.

OPERATION_RFD

OPERATION_RFD is asserted by the core to indicate that it is ready to accept new operands on inputs A, B and OPERATION. A new operation is initiated by the core when both OPERATION_ND and OPERATION_RFD are asserted together.

RESULT

If the operation is compare, then the valid bits within the result depend upon the compare operation selected. If the compare operation is one of those listed in Table 3, then only the least significant bit of the result indicates whether the comparison is true or false. If the operation is condition code, then the result of the comparison is provided by 4-bits using the encoding summarized in Table 4. See *IEEE-754 Standard* for a more complete listing of the meanings of all the valid comparison results.

Table 4: **Condition Code Summary**

Compare Operation	RESULT(3 downto 0)				Result
	3	2	1	0	
Programmable				0	A OP B = False
				1	A OP B = True
Condition Code	Unordered	>	<	EQ	Meaning
	0	0	0	1	A = B
	0	0	1	0	A < B
	0	0	1	1	A <= B
	0	1	0	0	A > B
	0	1	0	1	A >= B
	0	1	1	0	A <> B
	1	See Standard			A, B or both are NaN.

The following signals provide exception information. Additional detail on their behavior can be found in the *IEEE-754 Standard*.

UNDERFLOW

Underflow is signaled when the operation generates a non-zero result which is too small to be represented with the chosen precision. The result is set to zero. Underflow is detected after rounding.

Note: A number that becomes denormalized before rounding is set to zero and underflow signaled.

OVERFLOW

Overflow is signaled when the operation generates a result that is too large to be represented with the chosen precision. The output is set to a correctly signed ∞ .

INVALID_OP

Invalid operation is signaled when the operation performed is invalid. According to the *IEEE-754 Standard*, the following are invalid operations:

1. Any operation on a signaling NaN. (Note that this is not relevant to the core as all NaNs are treated as Quiet NaNs).
2. Addition or subtraction of infinite values where the sign of the result cannot be determined. For example, magnitude subtraction of infinities such as $(+\infty) + (-\infty)$.
3. Multiplication where $0 \times \infty$.
4. Division where $0/0$ or ∞/∞.
5. Square root if the operand is less than zero.
6. When the input of a conversion precludes a faithful representation that cannot otherwise be signaled (for example NaN or infinity).

When an invalid operation occurs, the associated result is a Quiet NaN. In the case of floating-point to fixed-point conversion, NaN and infinity raise an invalid operation exception. If the operand is out of range, or an infinity, then an overflow exception is raised. By analyzing the two exception signals it is possible to determine which of the three types of operand was converted. (See Table 5.)

Table 5: **Invalid Operation Summary**

Operand	Invalid Operation	Overflow	Result
+ Out of Range	0	1	011...11
- Out of Range	0	1	100...00
+ Infinity	1	1	011...11
- Infinity	1	1	100...00
NaN	1	0	100...00

When the operand is a NaN the result is set to the most negative representable number. When the operand is infinity or an out-of-range floating-point number, the result is saturated to the most positive or most negative number, depending upon the sign of the operand.

Note: Floating-point to fixed-point conversion does not treat a NaN as a Quiet NaN, because NaN is not representable within the resulting fixed-point format, and so can only be indicated through an invalid operation exception.

DIVIDE_BY_ZERO

`DIVIDE_BY_ZERO` is asserted when a divide operation is performed where the divisor is zero and the dividend is a finite non-zero number. The result in this circumstance is a correctly signed infinity.

RDY

`RDY` is asserted by the core to indicate that `RESULT` is valid. `RDY` can be used to qualify the result of a multi-cycle operation (that is, divide or square root operations with rate greater than 1).

Example Timing

An example of signal timing is given in Figure 4 for square-root with latency 4 and rate 3. The result is provided four cycles after an active OPERATION_ND. In this example, new inputs are applied every three cycles, in accordance with the maximum rate and OPERATION_RFD output. (Data could be applied less frequently, in which case OPERATION_RFD would stay high until OPERATION_ND was asserted with the new input.) The RDY output indicates when RESULT, and any exception flags, are valid. In this example, an invalid operation exception has been generated with result R2.

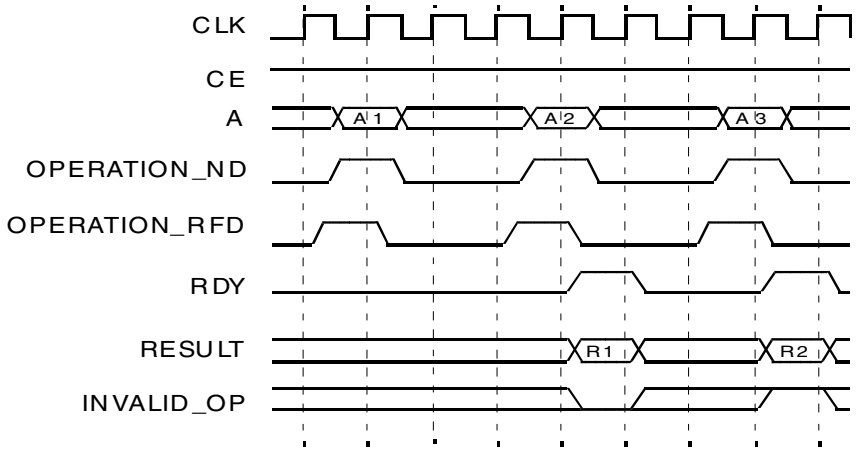

Figure 4: **Example Timing Diagram**

CORE Generator Graphical User Interface

The Floating Point core GUI provides several screens with fields to set the parameter values for the particular instantiation required. This section provides a description of each GUI field.

The GUI allows configuration of the following:

- Core operation
- Wordlength
- Implementation optimizations, such as use of XtremeDSP slices
- Optional pins

Main Configuration Screen

The main configuration screen allows the following parameters to be specified:

- Component Name
- Operation Selection

Component Name

The component name is used as the base name of the output files generated for the core. Names must start with a letter and be composed using the following characters: a to z, 0 to 9, and "_".

Operation Selection

The floating-point operation may be one of the following:

- Add/Subtract
- Multiply
- Divide
- Square-root
- Compare
- Fixed-to-float
- Float-to-fixed
- Float-to-float

When *Add/Subtract* is selected, it is possible for the core to perform both operations, or just add or subtract. When both are selected, the operation performed on a particular set of operands is controlled by the `OPERATION` input (with encoding defined earlier in Table 3).

When *Add/Subtract* or *Multiply* is selected, the level of embedded multiplier or XtremeDSP slice usage can be specified according to FPGA family as described in the Penultimate Configuration Screen section.

When *Compare* is selected, the compare operation may be programmable or fixed. If programmable, then the compare operation performed should be supplied via the `OPERATION` input (with encoding defined earlier in Table 3). If a fixed operation is required, then the operation type should be selected.

When *Float-to-float* conversion is selected, and exponent and fraction widths of the input and result are the same, the core provides a means to condition numbers, that is, convert denormalized numbers to zero, and signaling NaNs to quiet NaNs.

Second and Third Configuration Screens

Depending on the configuration you select from the first screen, the second and third configuration screens let you specify the precision of the operand and result.

Precision of the Operand and Results

This parameter defines the number of bits used to represent quantities. The type of the operands and results depend on the operation requested. For fixed-point conversion operations, either the operand or result is fixed-point. For all other operations, the output is specified as a floating-point type.

Note: For the condition-code compare operation, RESULT(3 downto 0) indicates the result of the comparison operation. For other compare operations RESULT(0 downto 0) provides the result.

Table 6 defines the general limits of the format widths.

Table 6: **General Limits of Width and Fraction Width**

Format Type	Fraction Width		Exponent/Integer Width		Width	
	Min	Max	Min	Max	Min	Max
Floating-Point	4	64	4	16	4	64
Fixed-Point	0	63	1	64	4	64

There are also a number of further limits for specific cases which are enforced by the GUI:

- The exponent width (that is., Total Width-Fraction Width) should be chosen to support normalization of the fractional part. This can be calculated using:

Minimum Exponent Width = ceil[$\log_2$(Fraction Width+3)] + 1

  For example, a 24-bit fractional part requires an exponent of at least 6 bits (for example, {ceil [$\log_2$ (27)]+1}).

- For conversion operations, the exponent width of the floating-point input or output is also constrained by the Total Width of the fixed-point input or output to be a minimum of:

Minimum Exponent Width = ceil[$\log_2$(Total Width+3)] + 1

  For example, a 32-bit integer requires a minimum exponent of 7 bits.

A summary of the width limits imposed by exponent width is provided in Table 7.

Table 7: **Summary of Exponent Width Limits**

Floating-Point Fraction Width or Fixed-Point Total Width	Minimum Exponent Width
4 to 5	4
6 to 13	5
14 to 29	6
30 to 61	7
61 to 64	8

Penultimate Configuration Screen

The final configuration screen lets you specify the following:

- Architecture Optimizations
- Family Optimizations
- Latency
- Cycles per Operation

Architecture Optimizations

On Virtex-5, Virtex-6, Virtex-7, and Kintex-7 FPGAs, for double precision multiplication and addition/subtraction operations, it is possible to specify a latency optimized architecture, or speed optimized architecture. The latency optimized architecture offers reduced latency at the expense of increased resources.

Family Optimizations

- Multiplier Usage allows the type and level of embedded multiplier usage to be specified.

Multiplier Usage

The level of embedded multiplier usage can be specified. The level and type of multiplier usage also depend upon the operation and FPGA family. Table 8 summarizes these options for multiplication.

Table 8: **Impact of Family and Multiplier Usage on the Implementation of the Multiplier**

Multiplier Usage	Spartan-3E/A FPGA Families	Spartan-3A DSP and Spartan-6 FPGA Families	Virtex-4 FPGA Family	Virtex-5, Virtex-6, Virtex-7, Kintex-7 FPGA Families
No usage	Logic	Logic	Logic	Logic
Medium usage	Not supported	DSP48A/A1+logic[1] in multiplier body	DSP48+logic[1] in multiplier body	DSP48E/E1+logic[1] in multiplier body
Full usage	MULT18X18	DSP48A/A1 used in multiplier body	DSP48 used in multiplier body	DSP48E/E1 used in multiplier body
Max usage	Not supported	DSP48A/A1 multiplier body and rounder	DSP48 multiplier body and rounder	DSP48E/E1 multiplier body and rounder

Notes:

1. Logic-assisted multiplier variant is available only for single and double precision formats in Virtex-4, Virtex-5, Virtex-6, Virtex-7, and Kintex-7 FPGAs and single precision in Spartan-3A DSP and Spartan-6 FPGAs.

Table 9 summarizes these options for addition/subtraction.

Table 9: **Impact of Family, Precision, and Multiplier Usage on the Implementation of the Adder/Subtractor**

Multiplier Usage (only valid values listed)	Other Families	Virtex-4 FPGA Family			Virtex-5, Virtex-6, Virtex-7, Kintex-7 FPGA Families		
	Any	Other	Single	Double	Other	Single	Double
No usage	Logic	Logic	Logic	Logic	Logic	Logic	Logic
Full usage	Not supported	Not supported	4 DSP48	3 DSP48	Not supported	2 DSP48E/E1	3 DSP48E/E1

Latency

This parameter describes the number of cycles between an operand input and result output. The latency of all operators (apart from the logic-assisted, double-precision multiplier on Virtex-4 devices) can be set between 0 and a maximum value that is dependent upon the parameters chosen. The maximum latency of the Floating-Point core is tabulated for a range of width and operation types in Tables 10 through 21.

The maximum latency of the divide and square root operations is Fraction Width + 4, and for compare operation it is two cycles. The float-to-float conversion operation is three cycles when either fraction or exponent width is being reduced; otherwise it is two cycles. Note that it is two cycles, even when the input and result widths are the same, as the core provides conditioning in this situation (see Operation Selection for further details).

Table 10: **Latency of Floating-Point Multiplication Using Logic Only**

Fraction Width	Maximum Latency (clock cycles)
4 to 5	5
6 to 11	6
12 to 23	7
24 to 47 (inc. single)	8
48 to 64 (inc. double)	9

Table 11: **Latency of Floating-Point Multiplication Using MULT18X18S**

Fraction Width	Maximum Latency (clock cycles)
4 to 17	4
18 to 34 (inc. single)	6
35 to 51	7
52 to 64 (inc. double)	8

Table 12: **Latency of Floating-Point Multiplication Using DSP48A/A1**

Fraction Width	Maximum Latency (clock cycles)		
	Medium Usage	Full Usage	Max Usage
4 to 17		6	5
18 to 34 (inc. single)	9[1]	11	10
35 to 51		18	17
52 to 64 (inc. double)		27	26

Notes:

1. Single precision only.

Table 13: **Latency of Floating-Point Multiplication Using DSP48**

Fraction Width	Maximum Latency (clock cycles)		
	Medium Usage	Full Usage	Max Usage
4 to 17		6	8
18 to 34 (inc. single)	9[1]	10	11
35 to 51		15	16
52 to 64 (inc. double)	17[2]	22	23

Notes:

1. Single precision only.
2. Double precision only.

Table 14: **Latency of Floating-Point Multiplication Using DSP48E/E1**

Fraction Width	Maximum Latency (clock cycles)		
	Medium Usage	Full Usage	Max Usage
single	8	8	6
double	15	15	16
4 to 17		6	8
18 to 24		8	9
25 to 34		10	11
35 to 41		12	13
42 to 51		15	16
52 to 58		18	19
59 to 64		22	23

Table 15: **Latency of Floating-Point Multiplication Using DSP48E/E1 and Low Latency Optimization**

Fraction Width	Maximum Latency (clock cycles)
	Max Usage
double	10

Table 16: **Latency of Floating-Point Addition Using Full Usage and DSP48/DSP48E/DSP48E1**

Width	Maximum Latency (clock cycles)	
	DSP48	DSP48E/E1
single	16	11
double	15	14

Table 17: **Latency of Floating-Point Addition Using Logic on Families Other Than Virtex-5, Virtex-6, Virtex-7, Kintex-7, and Spartan-6 FPGAs**

Fraction Width	Maximum Latency (clock cycles)
4, 5	9
6 to 14	10
15	11
16, 17	12
18 to 29 (inc. single)	13
30 to 62 (inc. double)	14
63, 64	15

Table 18: **Latency of Floating-Point Addition Using Logic and Low-Latency Optimization on Virtex-5, Virtex-6, Virtex-7, and Kintex-7 FPGAs**

Fraction Width	Maximum Latency (clock cycles)
single	8
double	8

Table 19: **Latency of Floating-Point Addition Using Logic and Speed Optimization on Virtex-5, Virtex-6, Virtex-7, Kintex-7, and Spartan-6 FPGAs**

Fraction Width	Maximum Latency (clock cycles)
4 to 13	8
14	9
15	10
16, 17	11
18 to 61 (single, double)	12
62 to 64	13

Table 20: **Latency of Fixed-Point to Floating-Point Conversion**

Operand Width	Maximum Latency (Cycles)
4 to 8	5
9 to 32	6
33 to 64	7

Table 21: **Latency of Floating-Point to Fixed-Point Conversion**

Maximum of (A Fraction Width+1) and Result Width	Maximum Latency (Cycles)
5 to 16	5
17 to 64	6
65	7

Cycles per Operation

The 'Cycles per Operation' GUI parameter describes the minimum number of cycles that must elapse between inputs. This rate can be specified. A value of 1 allows operands to be applied on every clock cycle, and results in a fully-parallel circuit. A value greater than 1 enables hardware reuse. The resources consumed by the core reduces as the number of cycles per operation is increased. A value of 2 approximately halves the resources used. A fully sequential implementation is obtained when the value is equal to Fraction Width+1 for the square-root operation, and Fraction Width+2 for the divide operation.

Final Configuration Screen

The final configuration screen lets you specify the Optional Control and Exception Pins.

Optional Control and Exception Pins

Pins for the following signals are optional:

- **Handshake and Control Signals:** `OPERATION_ND`, `OPERATION_RDY`, `RDY`, `CE` and `SCLR` control signals are optional.
- **Exception Signals:** `UNDERFLOW`, `OVERFLOW`, `INVALID_OPERATION` and `DIVIDE_BY_ZERO` signals are optional. The `DIVIDE_BY_ZERO` signal is available only when the divide operation is selected.

Using the Floating Point IP Core

The CORE Generator GUI performs error-checking on all input parameters. Resource estimation and optimum latency information are also available.

Several files are produced when a core is generated, and customized instantiation templates for Verilog and VHDL design flows are provided in the .veo and .vho files, respectively. For detailed instructions, see the CORE Generator software documentation.

Simulation Models

The core has two options for simulation models:

- VHDL behavioral model in xilinxcorelib library
- Verilog UniSim-based structural simulation model

The models required may be selected in the CORE Generator project options.

Xilinx recommends that simulations utilizing UniSim-based structural models be run using a resolution of 1 ps. Some Xilinx library components require a 1 ps resolution to work properly in either functional or timing simulation. The UniSim-based structural simulation models may produce incorrect results if simulated with a resolution other than 1 ps. See the "Register Transfer Level (RTL) Simulation Using Xilinx Libraries" section in Chapter 6 of the Synthesis and Simulation Design Guide for more information. This document is part of the ISE Software Manuals set available at www.xilinx.com/support/software_manuals.htm.

XCO Parameters

Table 22 defines valid entries for the XCO parameters. Parameters are not case sensitive. Default values are displayed in bold. Xilinx strongly recommends that XCO parameters not be manually edited in the XCO file; instead, use CORE Generator software GUI to configure the core and perform range and parameter value checking.

Table 22: **XCO Parameters**

XCO Parameter	XCO Values
Component_Name	Name must begin with a letter and be composed of the following characters: a to z, A to Z, 0 to 9 and "_".
Operation_Type	**Add_Subtract**, Multiply, Divide, Square_Root, Compare, Fixed_to_float, Float_to_fixed, Float_to_float
Add_Sub_Value	**Both**, Add, Subtract

Table 22: **XCO Parameters**

XCO Parameter	XCO Values
C_Compare_Operation	**Programmable**, Unordered, Less_Than, Equal, Less_Than_Or_Equal, Greater_Than, Not_Equal, Greater_Than_Or_Equal, Condition_Code
A_Precision_Type	**Single**, Double, Int32, Custom
C_A_Exponent_Width	Integer with range summarized in Table 6 and Table 7. Required when A_Precision_Type is Custom.
C_A_Fraction_Width	Integer with range summarized in Table 6 and Table 7. Required when A_Precision_Type is Custom.
Result_Precision_Type	**Single**, Double, Int32, Custom.
C_Result_Exponent_Width	Integer with range summarized in Table 6 and Table 7. Required when Result_Precision_Type is Custom.
C_Result_Fraction_Width	Integer with range summarized in Table 6 and Table 7. Required when Result_Precision_Type is Custom.
C_Optimization	**Speed_Optimized**, Low_Latency
C_Mult_Usage	**No_Usage**, Medium_Usage, Full_Usage, Max_Usage
Maximum_Latency	False, **True**
C_Latency	Integer with range 0 to the maximum latency of core as summarized by Tables 10 through 21 (default is **maximum latency**). Required when Maximum_Latency is False.
C_Rate	Integer with range 1 to maximum rate as described in Cycles per Operation (default is **1**).
C_Speed	Maximum_Speed
C_Has_OPERATION_ND	**False**, True
C_Has_OPERATION_RFD	**False**, True
C_Has_RDY	**False**, True
C_Has_SCLR	**False**, True
C_Has_CE	**False**, True
C_Has_UNDERFLOW	**False**, True
C_Has_OVERFLOW	**False**, True
C_Has_INVALID_OP	**False**, True
C_Has_DIVIDE_BY_ZERO	**False**, True

Modeling

As described previously, the core performs floating-point arithmetic as defined by the *IEEE-754 Standard* for single and double precision formats for all but denormalized numbers. The latter are extremely small numbers which very rarely contribute to the final result of a calculation. The following c-code shows how to determine single precision results for a particular operation using standard c-code for comparison with the core:

```
#include <stdio.h>
main() {

  union int32_or_single {
    int i;
    float f;
  } a, b, r, rs;

  union int64_or_double {
    long long int i; // Please specify 64-bit integer type for platform
    double f;
  } rd;

  // Uncomment to assign values in decimal
  //a.f = 15161.0;
  //b.f = 1077.00048828125;
  // Assign values in hexadecimal
  a.i=0x466CE400;
  b.i=0x4486A004;

  //Do the deed (in this example subtraction)
  r.f=a.f - b.f;

  //Repeat, but this time do in double precision to avoid rounding
  rd.f=(double) a.f - (double) b.f;

  printf("a: Hex=%08X ", a.i);
  printf("Float=%.11f\n", a.f);
  printf("b: Hex=%08X ", b.i);
  printf("Float=%.11f\n", b.f);
  printf("Single result: Hex=%08X ", r.i);
  printf("Float=%.11f\n", r.f);
  printf("Double result: Hex=%08X%08X ",(int)(rd.i>>32),(int)(rd.i&0xFFFFFFFF));
  printf("Float=%.11f\n", rd.f);

  rs.f = (float) rd.f; // Round result from double to single

  printf("Double rounded to single: Hex=%08X ", rs.i);
  printf("Float=%.11f\n", rs.f);

  // Expect the following output:
  //  a: Hex=466CE400 Float=15161.00000000000
  //  b: Hex=4486A004 Float=1077.00048828125
  //  Single precision result: Hex=465C1000 Float=14084.00000000000
  //  Double precision result: Hex=40CB81FFF0000000 Float=14083.99951171875
  //  Double rounded to single: Hex=465C1000 Float=14084.00000000000
}
```

Migrating to Floating-Point Operator v5.0 from v4.0

The CORE Generator core update feature may be used to update an existing Floating-Point Operator XCO file from version 4.0 to version 5.0 of the Floating-Point Operator core. The core may then be regenerated to create a new netlist. See the CORE Generator documentation for more information on this feature.

Port Changes

There are no differences in port naming conventions, polarities, priorities or widths between versions.

Latency Changes

There are no latency differences between versions. If Latency Configuration is Manual, the latency used for v4.0 will be re-used when regenerating for v5.0.

Resource Utilization and Performance

The resource requirements and maximum clock rates achievable on Spartan-6, Spartan-3A DSP, Virtex-6 and Virtex-5 FPGAs are summarized as follows for the case of maximum latency and no SCLR or CE pins.

Note: Both LUT and FF resource usage and maximum frequency reduce with latency. Minimizing latency minimizes resources.

The maximum clock frequency results were obtained by double-registering input and output ports to reduce dependence on I/O placement. The inner level of registers used a separate clock signal to measure the path from the input registers to the first output register through the core.

The resource usage results do not include the above "characterization" registers and represent the true logic used by the core. LUT counts include SRL16s or SRL32s (according to device family).

The map options used were: "`map -pr b -ol high`."

The par options used were: "`par -t 1 -ol high`."

Clock frequency does not take clock jitter into account and should be derated by an amount appropriate to the clock source jitter specification.

The maximum achievable clock frequency and the resource counts may also be affected by other tool options, additional logic in the FPGA device, using a different version of Xilinx tools, and other factors.

It is possible to improve performance of the Xilinx Floating-Point Operator within a system context by placing the operator within an area group. Placement of both the logic slices and XtremeDSP slices can be contained in this way. If multiply-add operations are used, then placing them in the same group can be helpful. Groups can also include any supporting logic to ensure that it is placed close to the operators.

Custom Format: 17-Bit Fraction and 24-Bit Total Wordlength

The resource requirements and maximum clock rates achievable with 17-bit fraction and 24-bit total wordlength on Spartan-6 FPGAs are summarized in Table 23.

Table 23: **Characterization of 17-Bit Fraction and 24-Bit Total Wordlength on Spartan-6 FPGA (Part=XC6SLX16-2)**

Operation	Resources					Maximum Frequency (MHz)[1, 2]
	Embedded		Fabric			Spartan-6
	Type	Number	LUT-FF Pairs	LUTs	FFs	-2 Speed Grade
Multiply	DSP48A1 (max usage)	2	100	97	94	192
	DSP48A1 (full usage)	1	117	108	122	245
	Logic (no usage)	0	366	355	396	213
Add/Subtract	Logic (no usage)		341	332	432	236
Fixed to float	Int24 input		169	158	175	250
Float to fixed	Int24 result		162	157	187	250
Float to float	Single to 24-17 format		78	77	85	250
	24-17 to single		41	35	52	250
Compare	Programmable		47	46	19	250
Divide	C_RATE=1		614	606	750	237
	C_RATE=19		183	168	180	200
Sqrt	C_RATE=1		412	389	462	250
	C_RATE=18		122	109	153	250

Notes:

1. Area and maximum clock frequencies are provided as a guide and might vary with new releases of the Xilinx implementation tools.
2. Maximum clock frequencies are shown in MHz. Clock frequency does not take jitter into account and should be de-rated by an amount appropriate to the clock source jitter specification.

The resource requirements and maximum clock rates achievable with 17-bit fraction and 24-bit total wordlength on Spartan-3A DSP FPGAs are summarized in Table 24.

Table 24: **Characterization of 17-Bit Fraction and 24-Bit Total Wordlength on Spartan-3A DSP FPGA (Part = XC3SD3400A-4)**

Operation	Resources					Maximum Frequency (MHz)[1,2]
	Embedded		Fabric			Spartan-3A DSP
	Type	Number	Slices	LUTs	FFs	-4 Speed Grade
Multiply	DSP48A (max usage)	2	95	134	102	184
	DSP48A (full usage)	1	107	105	134	207
	Logic (no usage)	0	250	352	407	166
Add/Subtract	Logic (no usage)		333	471	429	194
Fixed to float	Int24 input		134	178	178	208
Float to fixed	Int24 result		152	243	186	206
Float to float	Single to 24-17 format		69	71	85	236
	24-17 to single		33	36	52	250
Compare	Programmable		47	75	19	206
Divide	C_RATE=1		426	466	753	194
	C_RATE=19		152	173	181	168
Sqrt	C_RATE=1		282	334	462	202
	C_RATE=18		121	160	156	182

Notes:

1. Area and maximum clock frequencies are provided as a guide and might vary with new releases of the Xilinx implementation tools.
2. Maximum clock frequencies are shown in MHz. Clock frequency does not take jitter into account and should be de-rated by an amount appropriate to the clock source jitter specification.

The resource requirements and maximum clock rates achievable with 17-bit fraction and 24-bit total wordlength on Virtex-6 FPGAs are summarized in Table 25.

Table 25: **Characterization of 17-Bit Fraction and 24-Bit Total Wordlength on Virtex-6 FPGA (Part = XC6VLX75-1)**

Operation	Resources					Maximum Frequency (MHz)[1,2]
	Embedded		Fabric			Virtex-6
	Type	Number	LUT-FF Pairs	LUTs	FFs	-1 Speed Grade
Multiply	DSP48E1 (max usage)	2	107	101	126	423
	DSP48E1 (full usage)	1	119	112	122	429
	Logic (no usage)	0	373	363	396	371
Add/ Subtract	Logic (no usage)		361	351	407	403
Fixed to float	Int24 input		171	170	139	349
Float to fixed	Int24 result		172	167	187	450
Float to float	Single to 24-17 format		91	88	85	450
	24-17 to single		46	45	52	450
Compare	Programmable		46	45	19	450
Divide	C_RATE=1		611	609	750	450
	C_RATE=19		199	192	180	374
Sqrt	C_RATE=1		410	403	462	450
	C_RATE=18		142	134	153	450

1. Area and maximum clock frequencies are provided as a guide and might vary with new releases of the Xilinx implementation tools.
2. Maximum clock frequencies are shown in MHz. Clock frequency does not take jitter into account and should be de-rated by an amount appropriate to the clock source jitter specification.

The resource requirements and maximum clock rates achievable with 17-bit fraction and 24-bit total wordlength on Virtex-5 are summarized in Table 26.

Table 26: **Characterization of 17-Bit Fraction and 24-Bit Total Wordlength on Virtex-5 FPGA (Part = XC5VLX30-1)**

Operation	Resources					Maximum Frequency (MHz)[1,2]
	Embedded		Fabric			Virtex-5
	Type	Number	LUT-FF Pairs	LUTs	FFs	-1 Speed Grade
Multiply	DSP48E (max usage)	2	139	66	127	410
	DSP48E (full usage)	1	137	87	122	410
	Logic (no usage)	0	416	334	395	334
Add/ Subtract	Logic (no usage)		452	338	407	404
Fixed to float	Int24 input		177	143	139	354
Float to fixed	Int24 result		202	155	187	414
Float to float	Single to 24-17 format		96	61	85	450
	24-17 to single		54	34	52	450
Compare	Programmable		53	45	19	450
Divide	C_RATE=1		773	450	751	433
	C_RATE=19		222	150	181	360
Sqrt	C_RATE=1		485	325	462	441
	C_RATE=18		161	127	153	429

Notes:

1. Area and maximum clock frequencies are provided as a guide and might vary with new releases of the Xilinx implementation tools.
2. Maximum clock frequencies are shown in MHz. Clock frequency does not take jitter into account and should be de-rated by an amount appropriate to the clock source jitter specification.

Single-Precision Format

The resource requirements and maximum clock rates achievable with single-precision format on Spartan-6 is summarized in Table 27.

Table 27: **Characterization of Single-Precision Format on Spartan-6 FPGAs (Part = XC6SLX16-2)**

Operation	Resources					Maximum Frequency (MHz)[1,2]
	Embedded		Fabric			Spartan-6
	Type	Number	LUT-FF Pairs	LUTs	FFs	-2 Speed Grade
Multiplier	DSP48A1 (max usage)	5	174	160	191	192
	DSP48A1 (full usage)	4	193	169	232	244
	DSP48A1 (medium usage)	1	495	488	545	233
	Logic (no usage)	0	649	637	692	203
Add/ Subtract	Logic (no usage)	0	482	457	590	245
Fixed to float	Int32 input		209	203	224	226
Float to fixed	Int32 result		205	199	233	240
Float to float	Single to double		44	42	68	250
Compare	Programmable		58	53	19	250
Divide	C_RATE=1		1,106	1,078	1,366	200
	C_RATE=26		219	207	222	197
Sqrt	C_RATE=1		685	671	809	217
	C_RATE=25		152	146	199	240

Notes:

1. Area and maximum clock frequencies are provided as a guide and might vary with new releases of the Xilinx implementation tools.
2. Maximum clock frequencies are shown in MHz. Clock frequency does not take jitter into account and should be de-rated by an amount appropriate to the clock source jitter specification.

The resource requirements and maximum clock rates achievable with single-precision format on Spartan-3A DSP is summarized in Table 28.

Table 28: **Characterization of Single-Precision Format on Spartan-3A DSP (XCS3SD3400A-4)**

Operation	Resources					Maximum Frequency (MHz)[1,2]
	Embedded		Fabric			Spartan-3A DSP
	Type	Number	Slices	LUTs	FFs	-4 Speed Grade
Multiplier	DSP48A (max usage)	5	150	184	204	202
	DSP48A (full usage)	4	170	180	245	199
	DSP48A (medium usage)	1	337	487	559	171
	Logic (no usage)	0	408	646	703	164
Add/ Subtract	Logic (no usage)	0	441	600	590	210
Fixed to float	Int32 input		172	230	226	186
Float to fixed	Int32 result		199	269	232	187
Float to float	Single to double		42	46	68	250
Compare	Programmable		55	91	19	185
Divide	C_RATE=1		779	827	1,367	179
	C_RATE=26		195	232	226	158
Sqrt	C_RATE=1		478	549	809	177
	C_RATE=25		157	214	202	173

Notes:

1. Area and maximum clock frequencies are provided as a guide and might vary with new releases of the Xilinx implementation tools.
2. Maximum clock frequencies are shown in MHz. Clock frequency does not take jitter into account and should be de-rated by an amount appropriate to the clock source jitter specification.

The resource requirements and maximum clock rates achievable with single-precision format on Virtex-6 DSP is summarized in Table 29.

Table 29: **Characterization of Single-Precision Format on Virtex-6 FPGAs (Part = XC6VLX75-1)**

Operation	Resources					Maximum Frequency (MHz)[1,2]
	Embedded		Fabric			Virtex-6
	Type	Number	LUT-FF Pairs	LUTs	FFs	-1 Speed Grade
Multiply	DSP48E1 (max usage)	3	110	107	114	429
	DSP48E1 (full usage)	2	156	154	183	429
	DSP48E1 (medium usage)	1	344	336	378	390
	Logic	0	662	659	692	395
Add/ Subtract	DSP48E1 (speed optimized, full usage)	2	295	287	337	380
	Logic (speed optimized, no usage)	0	489	477	557	450
	Logic (low latency)	0	605	601	626	393
Fixed to float	Int32 input		205	200	224	411
Float to fixed	Int32 result		209	206	233	450
Float to float	Single to double		59	58	68	450
Compare	Programmable		54	53	19	450
Divide	C_RATE=1		1,099	1,071	1,366	433
	C_RATE=26		217	215	222	401
Sqrt	C_RATE=1		696	683	809	382
	C_RATE=25		188	181	199	386

Notes:

1. Area and maximum clock frequencies are provided as a guide and might vary with new releases of the Xilinx implementation tools.
2. Maximum clock frequencies are shown in MHz. Clock frequency does not take jitter into account and should be de-rated by an amount appropriate to the clock source jitter specification.

The resource requirements and maximum clock rates achievable with single-precision format on Virtex-5 is summarized in Table 30.

Table 30: **Characterization of Single-Precision Format on Virtex-5 FPGAs (Part = XC5VLX30-1)**

Operation	Resources					Maximum Frequency (MHz)[1,2]
	Embedded		Fabric			Virtex-5
	Type	Number	LUT-FF Pairs	LUTs	FFs	-1 Speed Grade
Multiply	DSP48E (max usage)	3	134	99	114	410
	DSP48E (full usage)	2	201	112	184	410
	DSP48E (medium usage)	1	402	268	378	384
	Logic	0	716	630	690	359
Add/Subtract	DSP48E (speed optimized, full usage)	2	378	245	337	410
	Logic (speed optimized, no usage)	0	629	432	558	420
	Logic (low latency)	0	707	562	626	377
Fixed to float	Int32 input		254	181	224	375
Float to fixed	Int32 result		257	199	233	373
Float to float	Single to double		75	43	68	450
Compare	Programmable		61	53	19	438
Divide	C_RATE=1		1,407	780	1,367	381
	C_RATE=26		262	187	223	337
Sqrt	C_RATE=1		835	533	809	406
	C_RATE=25		208	170	199	390

Notes:

1. Area and maximum clock frequencies are provided as a guide and might vary with new releases of the Xilinx implementation tools.
2. Maximum clock frequencies are shown in MHz. Clock frequency does not take jitter into account and should be de-rated by an amount appropriate to the clock source jitter specification.

Double-Precision Format

The resource requirements and maximum clock rates achievable with double-precision format on Virtex-6 FPGAs are summarized in Table 31.

Table 31: **Characterization of Double-Precision Format on Virtex-6 FPGAs (Part = XC6VLX75-1)**

Operation	Resources					Maximum Frequency (MHz)[1,2]
	Embedded		Fabric			Virtex-6
	Type	Number	LUT-FF Pairs	LUTs	FFs	-1 Speed Grade
Multiply	DSP48E1 (max usage)	11	357	328	497	429
	DSP48E1 (full usage)	10	372	351	494	395
	DSP48E1 (medium usage)	9	428	411	554	408
	Logic	0	2,373	2,352	2,444	295
	DSP48E1 (low latency, max usage)	13	268	257	311	361
Add/Subtract	DSP48E1 (speed optimized, full usage)	3	849	834	960	421
	Logic (speed optimized, no usage)	0	916	911	1,046	361
	Logic (low latency, no usage)	0	1,125	1,121	1,177	346
Fixed to float	Int64 input		470	455	504	363
Float to fixed	Int64 result		417	414	449	367
Float to float	Double to single		125	125	108	448
Compare	Programmable		93	92	19	439
Divide	C_RATE=1		4,650	4,646	5,996	277
	C_RATE=55		399	374	387	306
Sqrt	C_RATE=1		2,646	2,633	3,285	321
	C_RATE=54		379	372	385	339

Notes:

1. Area and maximum clock frequencies are provided as a guide and might vary with new releases of the Xilinx implementation tools.
2. Maximum clock frequencies are shown in MHz. Clock frequency does not take jitter into account and should be de-rated by an amount appropriate to the clock source jitter specification.

The resource requirements and maximum clock rates achievable with double-precision format on Virtex-5 FPGAs are summarized in Table 32.

Table 32: **Characterization of Double-Precision Format on Virtex-5 FPGAs (Part = XC5VLX30-1)**

Operation	Resources					Maximum Frequency (MHz)[1,2]
	Embedded		Fabric			Virtex-5
	Type	Number	LUT-FF Pairs	LUTs	FFs	-1 Speed Grade
Multiply	DSP48E (max usage)	11	527	322	498	410
	DSP48E (full usage)	10	524	342	495	359
	DSP48E (medium usage)	9	585	402	555	369
	Logic	0	2,485	2,293	2,443	250
	DSP48E (low latency, max usage)	13	346	191	311	404
Add/Subtract	DSP48E (speed optimized, full usage)	3	1,084	734	960	355
	Logic (speed optimized, no usage)	0	1,173	791	1,046	333
	Logic (low latency, no usage)	0	1,305	1,035	1,177	293
Fixed to float	Int64 input		559	404	504	306
Float to fixed	Int64 result		491	376	449	286
Float to float	Double to single		137	86	108	442
Compare	Programmable		100	92	19	365
Divide	C_RATE=1		6,113	3,220	5,997	258
	C_RATE=55		505	331	388	250
Sqrt	C_RATE=1		3,369	1,925	3,285	279
	C_RATE=54		418	349	385	267

Notes:

1. Area and maximum clock frequencies are provided as a guide and might vary with new releases of the Xilinx implementation tools.
2. Maximum clock frequencies are shown in MHz. Clock frequency does not take jitter into account and should be de-rated by an amount appropriate to the clock source jitter specification.

Support

Xilinx provides technical support for this LogiCORE product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled *DO NOT MODIFY*.

See the IP Release Notes Guide (XTP025) for further information on this core.

For each core, there is a master Answer Record that contains the Release Notes and Known Issues list for the core being used. The following information is listed for each version of the core:

- New Features
- Bug Fixes
- Known Issues

Ordering Information

This core may be downloaded from the Xilinx IP Center for use with the Xilinx CORE Generator software v13.1 and later. The Xilinx CORE Generator system is shipped with Xilinx ISE Design Suite development software.

To order Xilinx software, contact your local Xilinx sales representative.

References

1. *ANSI/IEEE, IEEE Standard for Binary Floating-Point Arithmetic,* ANSI/IEEE Standard 754-1985. IEEE-754.

Revision History

This table shows the revision history of this document.

Date	Version	Revision
04/28/05	1.0	Initial Xilinx release.
07/27/05	1.1	Document modified to include minor corrections and section on simulation.
01/18/06	2.0	Updated to version 2.0 of core, Xilinx tools v8.1i.
09/28/06	3.0	Updated to version 3.0 of core, Xilinx tools v8.2i.
04/25/08	4.0	Updated to version 4.0 of core, Xilinx tools v10.1.
06/24/09	5.0	Updated to version 5.0 of core, Xilinx tools v11.2.
03/01/11	6.0	Added support for Virtex-7 and Kintex-7.

Notice of Disclaimer