LogiCORE IP Floating-Point Operator v6.0

DS816 January 18, 2012 **Product Specification**

Introduction

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

Features

- Supported operators
  - multiply
  - add/subtract
  - divide
  - square-root
  - comparison
  - reciprocal
  - reciprocal square root
  - conversion from floating-point to fixed-point
  - conversion from fixed-point to floating-point
  - conversion between floating-point types
- Compliance with *IEEE-754 Standard* [Ref 1] (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
- Optimizations for speed and latency
- Fully synchronous design using a single clock
- For use with CORE Generator™ and Xilinx System Generator for DSP which are available in the Xilinx ISE® 13.4 software

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family(1)	Virtex-7, Kintex-7, Artix™-7, Zynq™-7000, Virtex-6, Spartan-6
Supported User Interfaces	AXI4-Stream
	Resources
Configuration	See Table 27 to Table 36
Provided with Core	
Documentation	Product Specification C- Model User Guide
Design Files	Netlist
Example Design	Not Provided
Test Bench	VHDL
Constraints File	Not Provided
Simulation Model	VHDL Verilog C-Model
Supported S/W Driver(2)	N/A
Tested Design Tools	
Design Entry Tools	CORE Generator 13.4 System Generator for DSP 13.4
Simulation(3)	Mentor Graphics ModelSim Cadence Incisive Enterprise Simulator (IES) Synopsys VCS and VCS MX ISim
Synthesis Tools	N/A
Support	
Provided by Xilinx, Inc.	

1. For a complete listing of supported devices, see the release notes for this core.
2. Standalone driver details can be found in the EDK or SDK directory (*<install_directory>*/doc/usenglish/xilinx_drivers.htm). Linux OS and driver support information is available from http://wiki.xilinx.com
3. For the supported version of the tools, see the ISE Design Suite 13: Release Notes Guide

© Copyright 2011–2012 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. ARM is a registered trademark of ARM in the EU and other countries. The AMBA trademark is a registered trademark of ARM Limited. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. All other trademarks are the property of their respective owners.

Overview

The Xilinx Floating-Point Operator 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 3. When a user selects an operation that requires only one operand, the B input channel is omitted.

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* [Ref 1] to allow for non-standard sizes. 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 Operator 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 $1 \le b_0.b_1b_2\ldots b_{p-1} < 2$ 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 1.

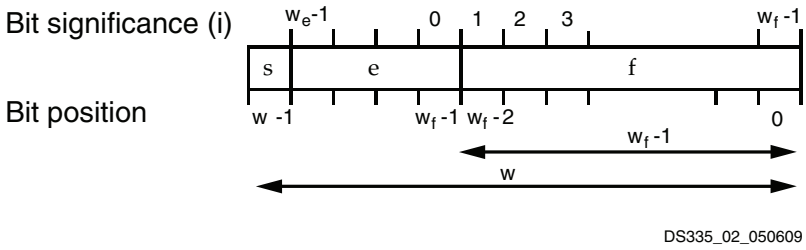

Figure 1: **Bit Fields within the Floating-Point Representation**

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 shown in Figure 1.

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	$2^{w_e - 1}$ -1 (that is, $e = 11...11$)	Any non-zero field. For results that are NaN the most significant bit of fraction is set (that is, $f = 10...00$)
$\pm\infty$	sign of ∞	$2^{w_e - 1}$ -1 (that is, $e = 11...11$)	Zero (that is, $f = 00...00$)
± 0	sign of 0	0	Zero (that is, $f = 00...00$)
denormalized	sign of number	0	Any non-zero field

In Table 1 the sign bit is undefined when a result is a NaN. The core generates NaNs with the sign bit set to 0 (that is, positive). 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 Operator core complies with much of the *IEEE-754 Standard* [Ref 1]. 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 Operator 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 can 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 Operator 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 might be very small differences between values generated by the Floating-Point Operator 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* [Ref 1]), is currently supported. This mode is often referred to as Round to Nearest Even, as values are rounded to the nearest representable value, with ties rounded to the nearest value with a zero least significant bit.

Signaling and Quiet NaNs

The *IEEE-754 Standard* requires provision of Signaling and Quiet NaNs. However, the Xilinx Floating-Point Operator 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 is not 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.

Accuracy of Results

Compliance to the *IEEE-754 Standard* requires that elementary arithmetic operations produce results accurate to half of one Unit in the Last Place (ULP). The Xilinx Floating-Point Operator satisfies this requirement for the

multiply, add/subtract, divide, square-root and conversion operators. The reciprocal and reciprocal square-root operators produce results which are accurate to one ULP.

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 2 (although it is still a weighted two's complement number).

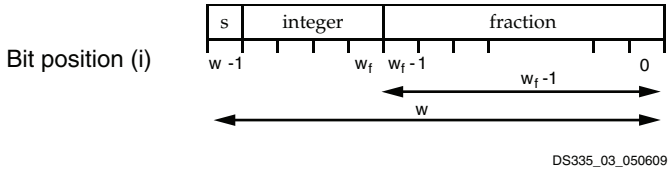

Figure 2: **Bit Fields within the Fixed-Point Representation**

In Figure 2, 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.

Pinout

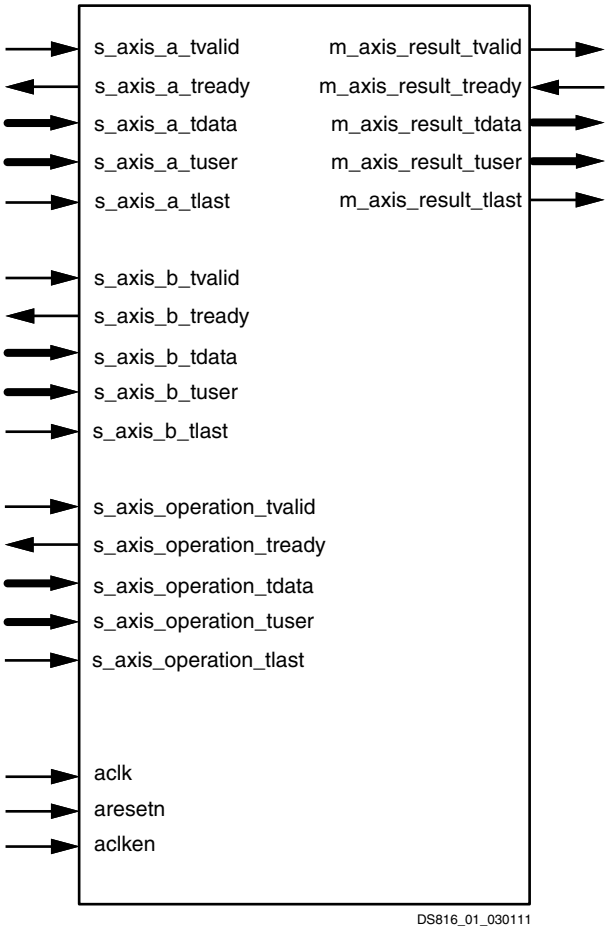

Figure 3: **Core Schematic Symbol**

Port Description

The ports employed by the core were shown in Figure 3. They are described in more detail in Table 2. All control signals are active high.

Table 2: **Core Signal Pinout**

Name	Direction	Optional	Description
aclk	Input	yes	Rising-edge clock
aclken	Input	yes	Active high clock enable (optional)
aresetn	Input	yes	Active low synchronous clear (optional, always takes priority over aclken)
s_axis_a_tvalid	Input	no	TVALID for channel A
s_axis_a_tready	Output	yes	TREADY for channel A
s_axis_a_tdata	Input	no	TDATA for channel A. See TDATA Packing for internal structure
s_axis_a_tuser	Input	yes	TUSER for channel A
s_axis_a_tlast	Input	yes	TLAST for channel A

Table 2: **Core Signal Pinout *(Cont'd)***

Name	Direction	Optional	Description
s_axis_b_tvalid	Input	no	TVALID for channel B
s_axis_b_tready	Output	yes	TREADY for channel B
s_axis_b_tdata	Input	no	TDATA for channel B. See TDATA Packing for internal structure
s_axis_b_tuser	Input	yes	TUSER for channel B
s_axis_b_tlast	Input	yes	TLAST for channel B
s_axis_operation_tvalid	Input	no	TVALID for channel OPERATION
s_axis_operation_tready	Output	yes	TREADY for channel OPERATION
s_axis_operation_tdata	Input	no	TDATA for channel OPERATION. See TDATA Packing for internal structure
s_axis_operation_tuser	Input	yes	TUSER for channel OPERATION
s_axis_operation_tlast	Input	yes	TLAST for channel OPERATION
m_axis_result_tvalid	Output	no	TVALID for channel RESULT
m_axis_result_tready	Input	yes	TREADY for channel RESULT
m_axis_result_tdata	Output	no	TDATA for channel RESULT. See TDATA Subfield for internal structure
m_axis_result_tuser	Output	yes	TUSER for channel RESULT
m_axis_result_tlast	Output	yes	TLAST for channel RESULT

All AXI4-Stream port names are lower case, but for ease of visualization, upper case is used in this document when referring to port name suffixes, such as TDATA or TLAST.

A Channel (s_axis_a_tdata)

Operand `A` input.

B Channel (s_axis_b_tdata)

Operand `B` input.

aclk

All signals are synchronous to the `aclk` input.

aclken

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

aresetn

When `aresetn` is asserted, the core control circuits are synchronously set to their initial state. Any incomplete results are discarded, and `m_axis_result_tvalid` is not generated for them. While `aresetn` is asserted `m_axis_result_tvalid` is synchronously deasserted. The core is ready for new input one cycle after `aresetn` is deasserted, at which point slave channel `tvalids` are asserted. `aresetn` takes priority over `aclken`. If `aresetn` is required to be gated by `aclken`, then this can be done externally to the core.

`aresetn` must be driven low for a minimum of two clock cycles to reset the core.

Operation Channel (s_axis_operation_tdata)

The operation channel 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 s_axis_operation_tdata**

FP Operation		s_axis_operation_tdata(5:0)
Add		000000
Subtract		000001
Compare (Programmable)	Unordered[1]	000100
	Less Than	001100
	Equal	010100
	Less Than or Equal	011100
	Greater Than	100100
	Not Equal	101100
	Greater Than or Equal	110100

1. An unordered comparison returns true when either (or both) of the operands are NaN, indicating that the operands' magnitudes cannot be put in size order.

Result Channel (m_axis_result_tdata)

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.

Table 4: **Condition Code Summary**

Compare Operation	m_axis_result_tdata(3:0)				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	1	0	0	A > B
	1	0	0	0	A, B or both are NaN.

The following flag signals provide exception information. Additional detail on their behavior can be found in the *IEEE-754 Standard*. The exception flags are not presented as discrete signals in Floating-Point Operator v6.0, but instead are provided in the RESULT channel `m_axis_result_tuser` subfield. For more details, see Output Result Channel.

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* [Ref 1], the following are invalid operations:

1. Any operation on a signaling NaN. (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. A special case is sqrt(-0), which is defined to be -0 by the *IEEE-754 Standard*.
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.

CORE Generator Graphical User Interface

The Floating-Point Operator 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 can be one of the following:

- Add/Subtract
- Multiply
- Divide
- Square-root
- Compare
- Reciprocal
- Reciprocal square root
- 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 `s_axis_operation` channel (with encoding defined earlier in Table 3).

When *Add/Subtract* or *Multiply* is selected, the level of 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 can be programmable or fixed. If programmable, then the compare operation performed should be supplied via the `s_axis_operation` channel (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, `m_axis_result_tdata(3:0)` indicates the result of the comparison operation. For other compare operations `m_axis_result_tdata(0:0)` 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:

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

  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:

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

  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 penultimate configuration screen lets you specify the following:

- Architecture Optimizations
- Family Optimizations

Architecture Optimizations

On Virtex®-6 and 7 series 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 level of XtremeDSP slice multiplier use to be specified.

Multiplier Usage

The level and type of multiplier usage 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-6 FPGA Family	Virtex-6 and 7 Series FPGA Families
No usage	Logic	Logic
Medium usage	DSP48A1+logic(1) in multiplier body	DSP48E1+logic(1) in multiplier body
Full usage	DSP48A1 used in multiplier body	DSP48E1 used in multiplier body
Max usage	DSP48A1 multiplier body and rounder	DSP48E1 multiplier body and rounder

1. Logic-assisted multiplier variant is available only for single and double precision formats in Virtex-6 and 7 Series FPGAs and single precision in 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)	Spartan-6 FPGA Family	Virtex-6 and 7 Series FPGA Families		
	Any	Other	Single	Double
No usage	Logic	Logic	Logic	Logic
Full usage	Not supported	Not supported	2 DSP48E1	3 DSP48E1

Final Configuration Screen

The final configuration screen lets you specify:

- Flow Control Options
- Latency and Rate Configuration
- Control Signals
- Optional Output Fields
- AXI Channel Options

Flow Control Options

These parameters allow the AXI4 interface to be optimized to suit the surrounding system.

- Flow Control
  - Blocking: When the core is configured to a Blocking interface, it waits for valid data to be available on all input channels before performing a calculation. Back pressure from downstream modules is possible.
  - NonBlocking: When the core is configured to use a NonBlocking interface, a calculation is performed on each cycle where all input channel TVALIDs are asserted. Back pressure from downstream modules is not possible.

- Optimize Goal
  - Resources: This option reduces the logic resources required by the AXI interface, at the expense of maximum achievable clock frequency.
  - Performance: This option allows maximum performance, at the cost of additional logic required to buffer data in the event of back pressure from downstream modules.
- RESULT channel has TREADY
  - Unchecking this option removes TREADY signals from the RESULT channel, disabling the ability for downstream modules to signal back pressure to the Floating-Point Operator core and upstream modules.

Latency and Rate Configuration

This parameter describes the number of cycles between an operand input and result output. The latency of all operators can be set between 0 and a maximum value that is dependent upon the parameters chosen. The maximum latency of the Floating-Point Operator core is tabulated for a range of width and operation types in Tables 10 through 18.

The latency values presented below represent the fully-pipelined latency of the internal Floating-Point Operator core. They do not include additional latency overhead due to AXI4 interface logic required when using a Blocking flow control scheme.

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 DSP48A1**

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

1. Single precision only.

Table 12: **Latency of Floating-Point Multiplication Using DSP48E1**

Fraction Width	Maximum Latency (clock cycles)		
	Medium Usage	Full Usage	Max Usage
single	8	8	6
double	15	15	16

Table 12: **Latency of Floating-Point Multiplication Using DSP48E1 *(Cont'd)***

Fraction Width	Maximum Latency (clock cycles)		
	Medium Usage	Full Usage	Max Usage
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 13: **Latency of Floating-Point Multiplication Using DSP48E1 and Low Latency Optimization**

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

Table 14: **Latency of Floating-Point Addition Using Full Usage and DSP48E1**

Width	Maximum Latency (clock cycles)
single	11
double	14

Table 15: **Latency of Floating-Point Addition Using Logic and Low-Latency Optimization on Virtex-6 and 7 Series FPGAs**

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

Table 16: **Latency of Floating-Point Addition Using Logic and Speed Optimization on 7 Series, Virtex-6 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 17: **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 18: **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

Table 19: **Latency of Floating-Point Reciprocal Using DSP48E1**

Fraction Width	Maximum Latency (clock cycles)	
	No Usage	Full Usage
single	36	29
double		35

Table 20: **Latency of Floating-Point Reciprocal Using DSP48A1**

Fraction Width	Maximum Latency (clock cycles)	
	No Usage	Full Usage
single	36	33
double		43

Table 21: **Latency of Floating-Point Reciprocal Square Root Using DSP48E1**

Fraction Width	Maximum Latency (clock cycles)	
	No Usage	Full Usage
single	37	32
double		112

Table 22: **Latency of Floating-Point Reciprocal Square Root Using DSP48A1**

Fraction Width	Maximum Latency (clock cycles)	
	No Usage	Full Usage
single	37	38

Note: Double precision reciprocal square root is not supported for Spartan-6 devices

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.

Control Signals

Pins for the following global signals are optional:

- ACLKEN: Active high clock enable.
- ARESETn: Active low synchronous reset. Must be driven low for a minimum of two clock cycles to reset the core.

Optional Output Fields

The following exception signals are optional and are added to `m_axis_result_tuser` when selected:

- UNDERFLOW, OVERFLOW, INVALID_OPERATION and DIVIDE_BY_ZERO.
- See TLAST and TUSER Handling for information on the internal packing of the exception signals in `m_axis_result_tuser`.

AXI Channel Options

The following sections allow configuration of additional AXI4 channel features:

- A Channel Options
  - Enables TLAST and TUSER input fields for the A operand channel, and allows definition of the TUSER field width.
- B Channel Options
  - Enables TLAST and TUSER input fields for the B operand channel (when present), and allows definition of the TUSER field width.
- OPERATION Channel Options
  - Enables TLAST and TUSER input fields for the OPERATION channel (when present), and allows definition of the TUSER field width.
- Output TLAST Behavior
  - When at least one TLAST input is present on the core, this option defines how the `m_axis_result_tlast` signal should be generated. Options are available to pass any of the input TLAST signals without modification, or to logically OR or AND all input TLASTs.

Using the Floating-Point Operator 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 RTL-based simulation model in XilinxCoreLib
- Verilog UNISIM-based structural simulation model

The models required can 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 can 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 23 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 23: **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, Reciprocal Rec_Square_Root Fixed_to_float, Float_to_fixed, Float_to_float
Add_Sub_Value	**Both**, Add, Subtract
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 18 (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**).
Has_ARESETn	**False**, True

Table 23: **XCO Parameters *(Cont'd)***

XCO Parameter	XCO Values
Has_ACLKEN	**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
Flow_Control	**Blocking**, NonBlocking
Axi_Optimize_Goal	**Resources**, Performance
Has_RESULT_TREADY	**True**, False
Has_A_TLAST	**False**, True
Has_A_TUSER	**False**, True
A_TUSER_Width	Integer with range 1 to 256. Default is **1**.
Has_B_TLAST	**False**, True
Has_B_TUSER	**False**, True
B_TUSER_Width	Integer with range 1 to 256. Default is **1**.
Has_OPERATION_TLAST	**False**, True
Has_OPERATION_TUSER	**False**, True
OPERATION_TUSER_Width	Integer with range 1 to 256. Default is **1**.
RESULT_TLAST_Behv	**Null**, Pass_A_TLAST, Pass_B_TLAST, Pass_OPERATION_TLAST, OR_All_TLASTs, AND_all_TLASTs

Core Use through System Generator for DSP

The Floating-Point Operator core is available through Xilinx System Generator, a DSP design tool that enables the use of The Mathworks model-based design environment Simulink® for FPGA design. The Floating-Point Operator is used within DSP math building blocks provided in the Xilinx blockset for Simulink. The blocks that provide floating-point operations using the Floating-Point Operator core are:

- AddSub
- Mult
- CMult (Constant Multiplier)
- Divide
- Reciprocal
- SquareRoot
- Reciprocal SquareRoot
- Relational (provides compare operations)
- Convert (provides fixed to float, float to fixed, float to float)

See the System Generator for DSP User Guide for more information.

Bit Accurate C Model

The Floating-Point Operator core has a bit-accurate C model designed for system modeling and selecting parameters before generating a core. The model is bit-accurate but not cycle-accurate, so it produces exactly the same output data as the core on a per-sample basis. However, it does not model the core latency or interface signals.

The Floating-Point Operator C model API is broadly based on the APIs of the GNU Multiple Precision Arithmetic (GMP) library [Ref 2], the GNU Multiple Precision Floating-Point Reliable (MPFR) library [Ref 3] and the GNU Multiple Precision Integers and Rationals (MPIR) library [Ref 4], which provide a library of floating point mathematical functions. This simplifies the usage of the C model for users already utilizing these libraries.

The C model is available as a dynamically linked library for 32-bit and 64-bit Windows, and 32-bit and 64-bit Linux platforms. The C model is an optional output of CORE Generator software, listed under Output Product Selection. Ensure that "C Simulation Model" is selected and then generate the core. The C model is generated in `<component_name>/cmodel/` as a zip file for each supported platform. Alternatively, the C model zip files are available for download from the Xilinx LogiCORE IP Floating Point Operator web page. Unzip the zip file for the correct platform to install the C model. A `README.txt` file describes the contents of the installed directory structure and any further platform-specific installation instructions. A user guide (UG812) is also provided which gives a full description of the C model including instructions on installation and interfacing to the model.

Demonstration Test Bench

When the core is generated using CORE Generator, a demonstration test bench is created. This is a simple VHDL test bench that exercises the core.

The demonstration test bench source code is one VHDL file: `demo_tb/tb_<component_name>.vhd` in the CORE Generator output directory. The source code is comprehensively commented.

Using the Demonstration Test Bench

The demonstration test bench instantiates the generated Floating-Point Operator core. If the CORE Generator project options were set to generate a structural model, a VHDL or Verilog netlist named `<component_name>.vhd` or `<component_name>.v` was generated. If this file is not present, generate it using the netgen program, for example:

```
netgen -sim -ofmt vhdl <component_name>.ngc <component_name>.vhd
```

Compile the netlist and the demonstration test bench into the work library (see your simulator documentation for more information on how to do this). Then simulate the demonstration test bench. View the test bench's signals in your simulator's waveform viewer to see the operations of the test bench.

The Demonstration Test Bench in Detail

The demonstration test bench performs the following tasks:

- Instantiates the core
- Generates an input data frame consisting of one or the sum of two complex sinusoids
- Generates a clock signal
- Drives the core's input signals to demonstrate core features
- Checks that the core's output signals obey AXI4 protocol rules (data values are not checked in order to keep the test bench simple)
- Provides signals showing the separate fields of AXI4 TDATA and TUSER signals

The demonstration test bench drives the core input signals to demonstrate the features and modes of operation of the core. The operations performed by the demonstration test bench are appropriate for the configuration of the generated core, and are a subset of the following operations:

1. An initial phase where the core is initialized and no operations are performed
2. Perform a single operation, and wait for the result
3. Perform 100 consecutive operations with incrementing data
4. Perform operations while demonstrating the AXI4 control signals' use and effects.
5. If ACLKEN is present: Demonstrate the effect of toggling `aclken`.
6. If ARESETn is present: Demonstrate the effect of asserting `aresetn`.
7. Demonstrate the handling of special floating-point values (NaN, zero, infinity).

Customizing the Demonstration Test Bench

The clock frequency of the core can be modified by changing the `CLOCK_PERIOD` constant.

AXI4-Stream Considerations

The conversion to AXI4-Stream interfaces brings standardization and enhances interoperability of Xilinx IP LogiCORE™ solutions. Other than general control signals such as `aclk`, `aclken` and `aresetn`, all inputs and outputs to and from the Floating-Point Operator core are conveyed via AXI4-Stream channels. A channel consists of TVALID and TDATA always, plus several optional ports and fields. In the Floating-Point Operator, the optional ports supported are TREADY, TLAST and TUSER. Together, TVALID and TREADY perform a handshake to transfer a message, where the payload is TDATA, TUSER and TLAST. The Floating-Point Operator operates on the operands contained in the TDATA fields and outputs the result in the TDATA field of the output channel. The Floating-Point Operator does not use TUSER and TLAST inputs as such, but the core provides the facility to convey these fields with the same latency as for TDATA. This facility is expected to ease use of the Floating-Point Operator in a system. For example, the Floating-Point Operator might be operating on streaming packetized data. In this example, the core could be configured to pass the TLAST of the packetized data channel, thus saving the system designer the effort of constructing a bypass path for this information.

For further details on AXI4-Stream interfaces see [Ref 5] and [Ref 6].

Basic Handshake

Figure 4 shows the transfer of data in an AXI4-Stream channel. TVALID is driven by the source (master) side of the channel and TREADY is driven by the receiver (slave). TVALID indicates that the value in the payload fields (TDATA, TUSER and TLAST) is valid. TREADY indicates that the slave is ready to receive data. When both TVALID and TREADY are true in a cycle, a transfer occurs. The master and slave set TVALID and TREADY respectively for the next transfer appropriately.

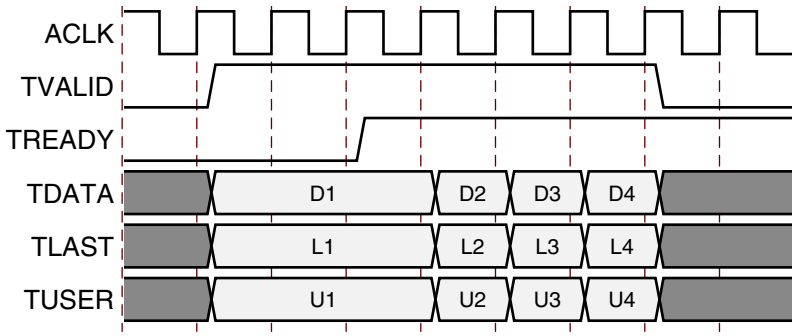

Figure 4: **Data Transfer in an AXI4-Stream Channel**

Non-Blocking Mode

The term Non-Blocking means that lack of data on one input channel does not block the execution of an operation if data is received on another input channel. The full flow control of AXI4-Stream is not always required. Blocking or Non-Blocking behavior is selected via the Flow Control parameter or GUI field. The core supports a Non-Blocking mode in which the AXI4-Stream channels do not have TREADY, that is, they do not support back pressure. The choice of Blocking or Non-Blocking applies to the whole core, not each channel individually. Channels still have the non-optional TVALID signal, which is analogous to the New Data (ND) signal on many cores prior to the adoption of AXI4-Stream interfaces. Without the facility to block dataflow, the internal implementation is much simplified, so fewer resources are required for this mode. This mode is recommended for users wishing to move to this version from a pre-AXI4-Stream core version with minimal change.

When all of the present input channels receive an active TVALID, an operation is validated and the output TVALID (suitably delayed by the latency of the core) is asserted to qualify the result. Operations occur on every enabled clock cycle and data is presented on the output channel payload fields regardless of TVALID. This is to allow a minimal migration from previous core versions. Figure 5 shows the Non-Blocking behavior for a case of an adder with latency of one cycle.

Warning: For performance, `aresetn` is registered internally, which delays its action by a clock cycle. The effect of this is that any transaction input in the cycle following the de-assertion of `aresetn` is reset by the action of `aresetn`, resulting in an output data value of zero. `m_axis_result_tvalid` is also inactive for this cycle.

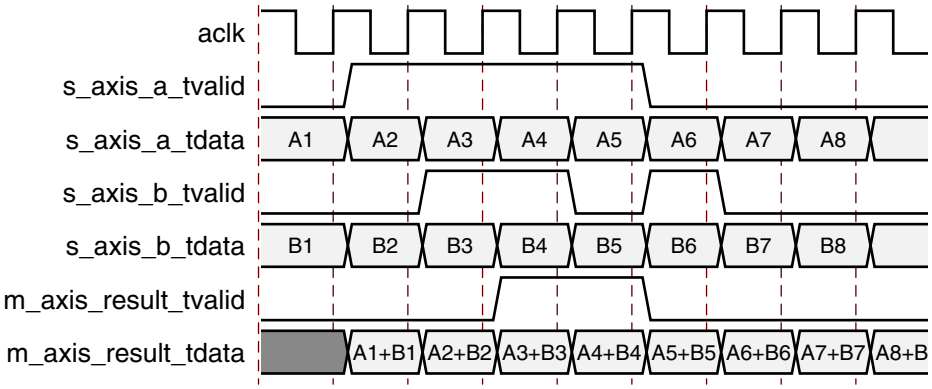

Figure 5: **Non-Blocking Mode**

Blocking Mode

The term Blocking means that operation execution does not occur until fresh data is available on all input channels. The full flow control of AXI4-Stream aids system design because the flow of data is self-regulating. Data loss is prevented by the presence of back pressure (TREADY), so that data is only propagated when the downstream datapath is ready to process the data.

The Floating-Point Operator has one, two or three input channels and one output channel. When all input channels have validated data available, an operation occurs and the result becomes available on the output. If the output is prevented from off-loading data because TREADY is low then data accumulates in the output buffer internal to the core. When this output buffer is nearly full the core will stop further operations. This prevents the input buffers from off-loading data for new operations so the input buffers fill as new data is input. When the input buffers fill, their respective TREADYs are deasserted to prevent further input. This is the normal action of back pressure.

The inputs are tied in the sense that each must receive validated data before an operation is prompted. Therefore, there is an additional blocking mechanism, where at least one input channel does not receive validated data while others do. In this case, the validated data is stored in the input buffer of the channel.

After a few cycles of this scenario, the buffer of the channel receiving data fills and TREADY for that channel is deasserted until the starved channel receives some data. Figure 6 shows both blocking behavior and back pressure for the case of an adder. The first data on channel A is paired with the first data on channel B, the second with the second and so on. This demonstrates the 'blocking' concept. The diagram further shows how data output is delayed not only by latency, but also by the handshake signal `m_axis_result_tready`. This is 'back pressure'. Sustained back pressure on the output along with data availability on the inputs eventually leads to a saturation of the core's buffers, leading the core to signal that it can no longer accept further input by deasserting the input channel TREADY signals. The minimum latency in this example is 2 cycles, but it should be noted that in Blocking operation latency is not a useful concept. Instead, as the diagram shows, the important idea is that each channel acts as a queue, ensuring that the first, second, third data samples on each channel are paired with the corresponding samples on the other channels for each operation.

Also note that the core buffers have a greater capacity than implied by the diagram.

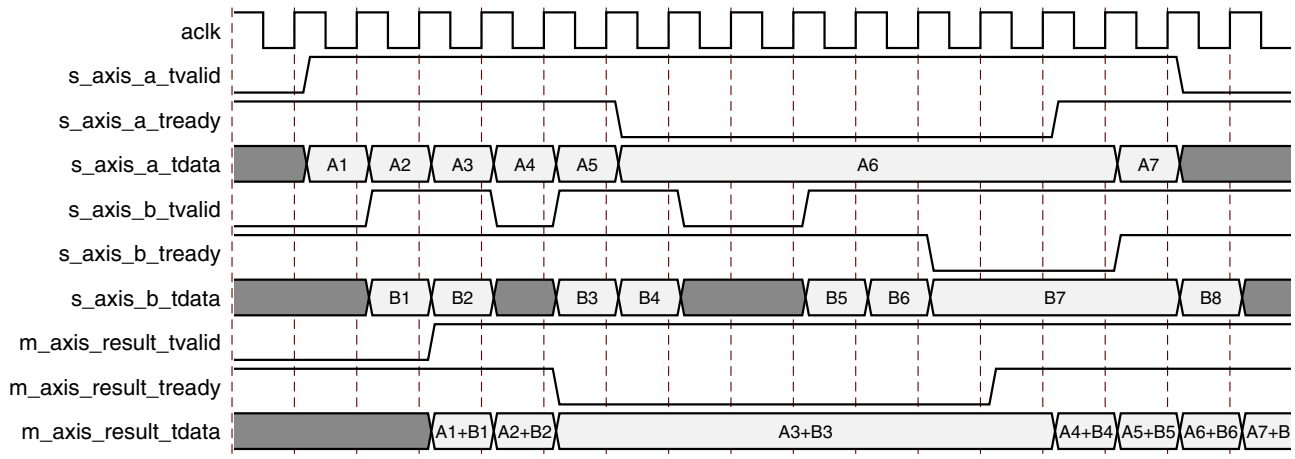

Figure 6: **Blocking Mode**

TDATA Packing

Fields within an AXI4-Stream interface are not given arbitrary names. Normally, information pertinent to the application is carried in the TDATA field. To ease interoperability with byte-oriented protocols, each subfield within

TDATA which could be used independently is first extended, if necessary, to fit a bit field which is a multiple of 8 bits. For example, say the Floating-Point Operator is configured to have an A operand with a custom precision of 11 bits (5 exponent and 6 mantissa bits). The operand would occupy bits (10:0). Bits (15:11) would be ignored. The bits added by byte orientation are ignored by the core and do not result in additional resource use.

A and B Input Channels

TDATA Structure for A and B Channels

Input channels A and B carry data for use in calculations in their TDATA fields. See Figure 7.

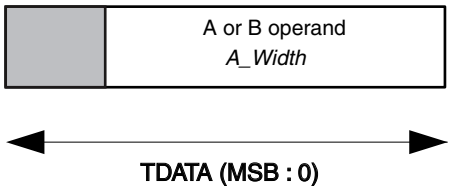

Figure 7: **TDATA Structure for A and B Channels**

Figure 8 illustrates how the previous example of a custom precision input with 11 bits maps to the TDATA channel.

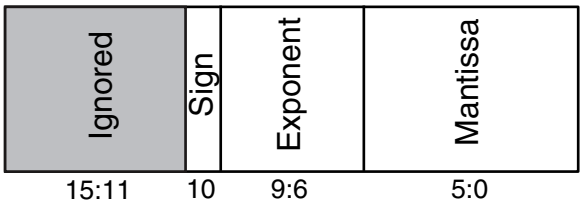

Figure 8: **Custom Precision Input (11 bits) Mapped to TDATA Channel**

TDATA Structure for OPERATION Channel

The OPERATION channel exists only when add and subtract operations are selected together, of when a programmable comparator is selected. The binary encoded operation code, as specified in Table 3, are 6 bits in length. However, due to the byte-oriented nature of TDATA, this means that TDATA has a width of 8 bits.

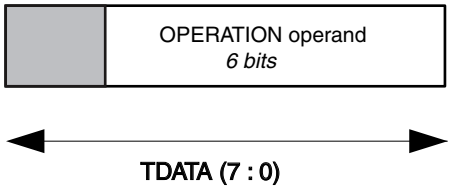

Figure 9: **TDATA Structure for OPERATION Channels**

TLAST and TUSER Handling

TLAST in AXI4-Stream is used to denote the last transfer of a block of data. TUSER is for ancillary information which qualifies or augments the primary data in TDATA. The Floating-Point Operator core operates on a per-sample basis where each operation is independent of any before or after. Because of this, there is no need for TLAST on a Floating-Point Operator core, nor is there any need for TUSER. The TLAST and TUSER signals are supported on each channel purely as an optional aid to system design for the scenario in which the data stream being passed through the Floating-Point Operator core does indeed have some packetization or ancillary field, but

which is not relevant to the core operation. The facility to pass TLAST and/or TUSER removes the burden of matching latency to the TDATA path, which can be variable, through the Floating-Point Operator core.

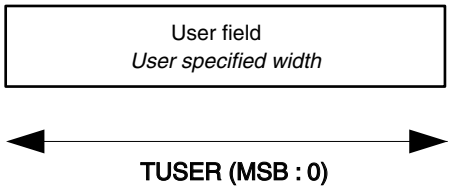

Figure 10: **TUSER Structure for A, B and OPERATION Channels**

TLAST Options

TLAST for each input channel is optional. Each, when present, can be passed via the Floating-Point Operator core, or, when more than one channel has TLAST enabled, can pass a logical AND or logical OR of the TLASTs input. When no TLASTs are present on any input channel, the output channel does not have TLAST either.

TUSER Options

TUSER for each input channel is optional. Each has user-selectable width. These fields are concatenated, without any byte-orientation or padding, to form the output channel TUSER field. The TUSER field from channel A will form the least significant portion of the concatenation, then TUSER from channel B, then TUSER from channel OPERATION.

For example, if channels A and OPERATION both have TUSER subfields with widths of 5 and 8 bits respectively, and no exception flag signals (underflow, etc.) are selected, the output TUSER is a suitably delayed concatenation of A and OPERATION TUSER fields, 13 bits wide, with A in the least significant 5 bit positions (4 down to 0).

Output Result Channel

TDATA Subfield

The internal structure of the RESULT channel TDATA subfield depends on the operation performed by the core.

For numerical operations (add, multiply, etc.) TDATA contains the numerical result of the operation and is a single floating-point or fixed-point number. The result width is sign-extended to a byte boundary if necessary. This is shown in Figure 11.

For Comparator operations, the result is either a 4 bit field (Condition Code) or a single bit indicating True or False. In both cases, the result is zero-padded to a byte boundary, as shown in Figure 12.

TUSER Subfield

The TUSER subfield is present if any of the input channels have an (optional) TUSER subfield, or if any of the exception flags (underflow, overflow, invalid operation, divide by zero) have been selected. The formatting of the TUSER fields is shown in Figure 13.

If any field of TUSER is not present, fields in more significant bit positions move down to fill the space. For example, if the overflow exception flag is selected, but the underflow exception flag is not, the overflow exception flag result moves to the least-significant bit position in the TUSER subfield.

No byte alignment is performed on TUSER fields. All fields present are immediately adjacent to one another with no padding between them or at the most significant bit.

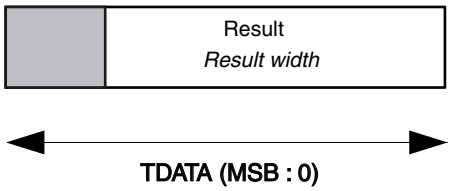

Figure 11: **TDATA Structure for Numerical Result Channel**

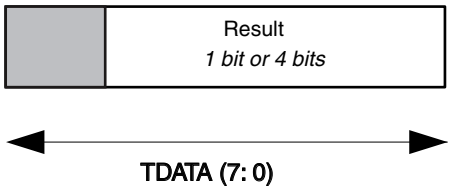

Figure 12: **TDATA Structure for Comparator Result Channel**

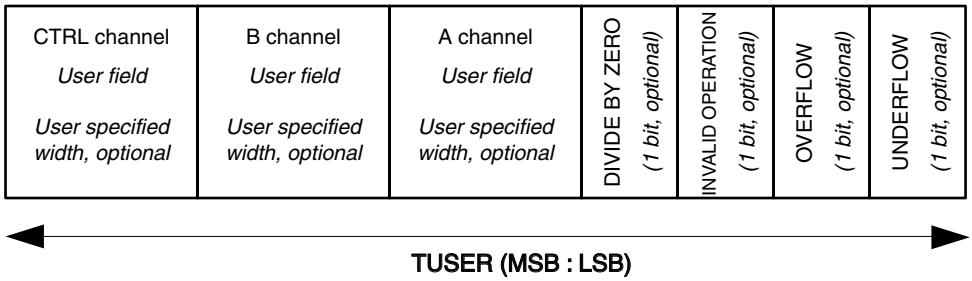

Figure 13: **TUSER Structure for Result Channel**

Migrating to Floating-Point Operator v6.0 from Earlier Versions

XCO Parameter Changes

The CORE Generator core upgrade functionality can be used to update an existing XCO file from versions 4.0 and 5.0 to Floating-Point Operator v6.0, but it should be noted that the upgrade mechanism alone does not create a core compatible with v6.0. See Instructions for Minimum Change Migration. Floating-Point Operator v6.0 has additional parameters for AXI4-Stream support. Figure 24 shows the changes to XCO parameters from versions 4.0 and 5.0 to version 6.0. For clarity, XCO parameters with no changes are not shown.

Table 24: **XCO Parameter Changes from v4.0 and v5.0 to v6.0**

Version 4.0 and 5.0	Version 6.0	Notes
C_Has_CE	Has_ACLKEN	Renamed only
C_Has_SCLR	Has_ARESETn	Renamed only. While the sense of the `aresetn` signal has changed (now active low), this XCO parameter determined whether or not the pin exists and has not changed.
C_Latency	C_Latency	Depending on the AXI4 Flow Control options selected (Blocking/NonBlocking), a minimum latency greater than previous core versions might be imposed.
	Flow_Control	New to version 6.0
	Axi_Optimize_Goal	New to version 6.0

Table 24: **XCO Parameter Changes from v4.0 and v5.0 to v6.0 *(Cont'd)***

Version 4.0 and 5.0	Version 6.0	Notes
	Has_RESULT_TREADY	New to version 6.0
	Has_A_TLAST	New to version 6.0
	Has_A_TUSER	New to version 6.0
	A_TUSER_Width	New to version 6.0
	Has_B_TLAST	New to version 6.0
	Has_B_TUSER	New to version 6.0
	B_TUSER_Width	New to version 6.0
	Has_OPERATION_TLAST	New to version 6.0
	Has_OPERATION_TUSER	New to version 6.0
	OPERATION_TUSER_Width	New to version 6.0
	RESULT_TLAST_Behv	New to version 6.0

For more information on this upgrade feature, see the CORE Generator software documentation.

Port Changes

Table 25 details the changes to port naming, additional or deprecated ports and polarity changes from v4.0 and v5.0 to v6.0.

Table 25: **Port Changes from v4.0 and v5.0 to v6.0**

Versions 4.0 and 5.0	Version 6.0	Notes
CLK	aclk	Rename only
CE	aclken	Rename only
SCLR	aresetn	Rename and change of sense (now active low). Must now be asserted for at least two clock cycles to effect a reset.
A(N-1:0)	s_axis_a_tdata(byte(N)-1:0)	byte(N) is to round N up to the next multiple of 8
B(N-1:0)	s_axis_b_tdata(byte(N)-1:0)	byte(N) is to round N up to the next multiple of 8
OPERATION(5:0)	s_axis_operation_tdata(7:0)	
RESULT(R-1:0)	m_axis_result_tdata(byte(R)-1:0)	byte(R) is to round R up to the next multiple of 8.
OPERATION_ND	Deprecated	Nearest equivalents are s_axis_<operand>_tvalid
OPERATION_RFD	Deprecated	Nearest equivalents are s_axis_<operand>_tready
RDY	Deprecated	Nearest equivalent is m_axis_result_tvalid
UNDERFLOW	Deprecated	Exception signals are now subfields of m_axis_result_tuser. See Figure 13 for data structure.
OVERFLOW	Deprecated	
INVALID_OP	Deprecated	
DIVIDE_BY_ZERO	Deprecated	
	s_axis_a_tvalid	TVALID (AXI4-Stream channel handshake signal) for each channel
	s_axis_b_tvalid	
	s_axis_operation_tvalid	
	m_axis_result_tvalid	
	s_axis_a_tready	TREADY (AXI4-Stream channel handshake signal) for each channel.
	s_axis_b_tready	
	s_axis_operation_tready	
	m_axis_result_tready	
	s_axis_a_tlast	TLAST (AXI4-Stream packet signal indicating the last transfer of a data structure) for each channel. The Floating-Point Operator does not use TLAST, but provides the facility to pass TLAST with the same latency as TDATA.
	s_axis_b_tlast	
	s_axis_operation_tlast	
	m_axis_result_tlast	
	s_axis_a_tuser(E-1:0)	TUSER (AXI4-Stream ancillary field for application-specific information) for each channel. The Floating-Point Operator does not use TUSER, but provides the facility to pass TUSER with the same latency as TDATA.
	s_axis_b_tuser(F-1:0)	
	s_axis_operation_tuser(G-1:0)	
	m_axis_result_tuser(H-1:0)	

Latency Changes

The latency of Floating-Point Operator v6.0 is different compared to v4.0 and v5.0 in general. The update process cannot account for this and guarantee equivalent performance.

Importantly, when in Blocking Mode, the latency of the core will be variable, so only the minimum possible latency can be determined.

When in Non-Blocking Mode, the latency of the core for equivalent performance is the same as that for the equivalent configuration of v4.0 and v5.0.

Instructions for Minimum Change Migration

To configure the Floating-Point Operator v6.0 to most closely mimic the behavior of previous versions the translation is as follows:

Parameters

Set Flow Control to NonBlocking and uncheck all AXI4 channel options (TUSER and TLAST).

Ports

Rename and map signals as detailed in Port Changes. Tie all TVALID signals on input channels (A, B, OPERATION) to '1'.

Remember to account for `aresetn` being active low, and the requirement to assert `aresetn` for at least two clock cycles to reset the core.

Performance

The fully-pipelined latency of the v6.0 core with a Non-Blocking interface configuration is the same as the v4.0 and v5.0 cores.

Resource Utilization and Performance

The resource requirements and maximum clock rates achievable on Virtex-7, Kintex™-7, Virtex-6 and Spartan®-6 FPGAs are summarized as follows for the case of maximum latency and no `aresetn` or `aclken` pins. Unless otherwise stated, Non-Blocking flow control is used for all configurations. For selected use cases, figures are provided for the Blocking and Performance flow control configuration which permits backpressure.

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 "characterization" registers above and represent the true logic used by the core. LUT counts include SRL16s or SRL32s.

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

The par options used were: "`par -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 might 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.

All results were produced using ISE 13.2 software.

Table 26: **Speed File Version**

FPGA Family	Speed File Version
Virtex-7	"PREVIEW 0.13 2011-05-25."
Kintex-7	"ADVANCED 1.01l 2011-05-25."
Virtex-6	"PRODUCTION 1.14 2011-05-24."
Spartan-6	"PRODUCTION 1.19a 2011-05-24."

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 Virtex-7 are summarized in Table 27.

Table 27: **Characterization of 17-Bit Fraction and 24-Bit Total Wordlength on Virtex-7 FPGAs (Part = XC7V585T-1)**

Operation	Resources					Maximum Frequency (MHz)[1][2]
	Embedded		Fabric			Virtex-7
	Type	Number	LUT-FF Pairs	LUTs	FFs	-1 Speed Grade
Multiply	DSP48E1 (max usage)	2	103	90	112	450
	DSP48E1 (full usage)	1	113	97	104	432
	Logic (no usage)	0	377	336	376	371
Add/Subtract	Logic (no usage)		369	301	393	461
Fixed to float	Int24 input		167	151	140	442
Float to fixed	Int24 result		160	143	184	531
Float to float	Single to 24-17 format		78	66	79	501
	24-17 to single		52	38	55	625
Compare	Programmable		41	39	13	550
Divide	RATE=1		547	479	710	491
	RATE=19		162	157	153	357
Square Root	RATE=1		392	326	446	501
	RATE=18		114	110	148	542
Multiply Flow Control: Blocking, Optimize Goal: Performance	DSP48E1 (max usage)	2	210	165	245	424
Add/Subtract Flow Control: Blocking, Optimize Goal: Performance	Logic (no usage)		475	395	531	493

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 Kintex-7 are summarized in Table 28.

Table 28: **Characterization of 17-Bit Fraction and 24-Bit Total Wordlength on Kintex-7 FPGAs (Part = XC7K70T-1)**

Operation	Resources					Maximum Frequency (MHz)(1)(2)
	Embedded		Fabric			Kintex-7
	Type	Number	LUT-FF Pairs	LUTs	FFs	-1 Speed Grade
Multiply	DSP48E1 (max usage)	2	104	89	112	439
	DSP48E1 (full usage)	1	113	97	104	463
	Logic (no usage)	0	363	340	376	398
Add/Subtract	Logic (no usage)		363	312	393	471
Fixed to float	Int24 input		165	155	140	435
Float to fixed	Int24 result		154	144	184	525
Float to float	Single to 24-17 format		76	67	79	518
	24-17 to single		54	31	55	525
Compare	Programmable		43	39	13	525
Divide	RATE=1		564	490	710	506
	RATE=19		159	156	153	374
Square Root	RATE=1		388	329	446	525
	RATE=18		114	110	148	525
Multiply Flow Control: Blocking, Optimize Goal: Performance	DSP48E1 (max usage)	2	217	158	245	434
Add/Subtract Flow Control: Blocking, Optimize Goal: Performance	Logic (no usage)		499	383	531	490

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 29.

Table 29: **Characterization of 17-Bit Fraction and 24-Bit Total Wordlength 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)	2	113	76	112	408
	DSP48E1 (full usage)	1	112	99	104	408
	Logic (no usage)	0	359	336	376	405
Add/Subtract	Logic (no usage)		363	302	393	477
Fixed to float	Int24 input		160	152	140	422
Float to fixed	Int24 result		164	139	184	490
Float to float	Single to 24-17 format		77	67	79	475
	24-17 to single		59	31	55	490
Compare	Programmable		43	39	13	490
Divide	RATE=1		571	496	710	475
	RATE=19		165	153	153	388
Square Root	RATE=1		403	340	446	490
	RATE=18		111	108	148	490
Multiply Flow Control: Blocking, Optimize Goal: Performance	DSP48E1 (max usage)	2	216	155	245	408
Add/Subtract Flow Control: Blocking, Optimize Goal: Performance	Logic (no usage)		472	391	531	467

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-6 FPGAs are summarized in Table 30.

Table 30: **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	98	87	86	245
	DSP48A1 (full usage)	1	104	99	105	306
	Logic (no usage)	0	360	333	376	294
Add/Subtract	Logic (no usage)		374	304	409	324
Fixed to float	Int24 input		153	134	171	330
Float to fixed	Int24 result		165	147	186	330
Float to float	Single to 24-17 format		77	59	79	330
	24-17 to single		43	32	56	330
Compare	Programmable		43	39	13	330
Divide	RATE=1		610	598	711	330
	RATE=19		166	153	153	258
Square Root	RATE=1		401	396	446	330
	RATE=18		113	107	150	330
Multiply Flow Control: Blocking, Optimize Goal: Performance	DSP48A1 (max usage)	2	186	161	221	244
Add/Subtract Flow Control: Blocking, Optimize Goal: Performance	Logic (no usage)		457	383	546	311

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 Virtex-7 FPGAs is summarized in Table 31.

Table 31: **Characterization of Single-Precision Format on Virtex-7 FPGAs (Part = XC7V585T-1)**

Operation	Resources					Maximum Frequency (MHz)[1][2]
	Embedded		Fabric			Virtex-7
	Type	Number	LUT-FF Pairs	LUTs	FFs	-1 Speed Grade
Multiply	DSP48E1 (max usage)	3	120	103	105	463
	DSP48E1 (full usage)	2	160	128	160	459
	DSP48E1 (medium usage)	1	331	283	331	462
	Logic	0	665	629	669	473
Add/Subtract	DSP48E1 (speed optimized, full usage)	2	293	225	327	408
	Logic (speed optimized, no usage)	0	500	407	541	472
	Logic (low latency)	0	551	496	610	457
Fixed to float	Int32 input		189	179	217	485
Float to fixed	Int32 result		200	187	230	525
Float to float	Single to double		71	45	71	608
Compare	Programmable		51	48	13	528
Divide	RATE=1		1106	825	1316	482
	RATE=26		199	193	192	334
Square Root	RATE=1		634	546	785	482
	RATE=25		145	140	194	402
Reciprocal	DSP48E1 (full usage)	8	215	183	265	515
	Logic (no usage)	0	1360	1308	1220	372
Reciprocal Square Root	DSP48E1 (full usage)	9	429	375	407	434
	Logic (no usage)	0	2093	2044	1879	402
Multiply Flow Control: Blocking, Optimize Goal: Performance	DSP48E1 (max usage)	3	270	183	279	462
Add/Subtract Flow Control: Blocking, Optimize Goal: Performance	DSP48E1 (speed optimized, full usage)	2	418	345	505	424

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 Kintex-7 FPGAs is summarized in Table 32.

Table 32: **Characterization of Single-Precision Format on Kintex-7 FPGAs (Part = XC7K70T-1)**

Operation	Resources					Maximum Frequency (MHz)[1][2]
	Embedded		Fabric			Kintex-7
	Type	Number	LUT-FF Pairs	LUTs	FFs	-1 Speed Grade
Multiply	DSP48E1 (max usage)	3	125	99	105	463
	DSP48E1 (full usage)	2	170	117	160	463
	DSP48E1 (medium usage)	1	334	285	331	462
	Logic	0	666	629	669	449
Add/Subtract	DSP48E1 (speed optimized, full usage)	2	289	230	327	419
	Logic (speed optimized, no usage)	0	517	403	541	438
	Logic (low latency)	0	587	482	610	464
Fixed to float	Int32 input		185	183	217	523
Float to fixed	Int32 result		209	184	230	492
Float to float	Single to double		57	56	71	525
Compare	Programmable		51	48	13	522
Divide	RATE=1		1173	827	1316	420
	RATE=26		199	189	192	363
Square Root	RATE=1		636	556	785	499
	RATE=25		143	140	194	414
Reciprocal	DSP48E1 (full usage)	8	202	188	265	429
	Logic (no usage)	0	1350	1311	1220	402
Reciprocal Square Root	DSP48E1 (full usage)	9	409	375	407	493
	Logic (no usage)	0	2075	2045	1879	362
Multiply Flow Control: Blocking, Optimize Goal: Performance	DSP48E1 (max usage)	3	264	191	279	463
Add/Subtract Flow Control: Blocking, Optimize Goal: Performance	DSP48E1 (speed optimized, full usage)	2	453	313	505	420

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 FPGAs is summarized in Table 33.

Table 33: **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	116	103	105	408
	DSP48E1 (full usage)	2	160	128	160	408
	DSP48E1 (medium usage)	1	329	293	331	408
	Logic	0	665	629	669	423
Add/Subtract	DSP48E1 (speed optimized, full usage)	2	292	217	327	399
	Logic (speed optimized, no usage)	0	498	405	542	476
	Logic (low latency)	0	571	492	610	472
Fixed to float	Int32 input		189	182	218	477
Float to fixed	Int32 result		210	180	230	483
Float to float	Single to double		75	41	71	490
Compare	Programmable		52	48	13	490
Divide	RATE=1		929	831	1319	429
	RATE=26		197	190	192	380
Square Root	RATE=1		645	548	789	384
	RATE=25		145	141	194	409
Reciprocal	DSP48E1 (full usage)	8	211	181	263	472
	Logic (no usage)	0	1367	1308	1220	355
Reciprocal Square Root	DSP48E1 (full usage)	9	424	382	407	448
	Logic (no usage)	0	2077	2048	1879	343
Multiply Flow Control: Blocking, Optimize Goal: Performance	DSP48E1 (max usage)	3	263	191	279	408
Add/Subtract Flow Control: Blocking, Optimize Goal: Performance	DSP48E1 (speed optimized, full usage)	2	432	328	505	406

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-6 FPGAs is summarized in Table 34.

Table 34: **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	182	159	191	238
	DSP48A1 (full usage)	4	187	157	220	315
	DSP48A1 (medium usage)	1	498	452	519	278
	Logic (no usage)	0	641	607	669	298
Add/Subtract	Logic (no usage)	0	509	418	563	324
Fixed to float	Int32 input		187	185	219	330
Float to fixed	Int32 result		210	181	233	330
Float to float	Single to double		50	45	73	330
Compare	Programmable		51	49	13	330
Divide	RATE=1		1098	1063	1321	268
	RATE=26		203	190	192	240
Square Root	RATE=1		686	671	786	308
	RATE=25		143	137	195	330
Reciprocal	DSP48A1 (full usage)	10	229	183	286	247
	Logic (no usage)	0	1272	1205	1220	282
Reciprocal Square Root	DSP48A1 (full usage)	12	463	410	482	292
	Logic (no usage)	0	1961	1918	1879	253
Multiply Flow Control: Blocking, Optimize Goal: Performance	DSP48A1 (max usage)	5	305	239	365	239
Add/Subtract Flow Control: Blocking, Optimize Goal: Performance	Logic (no usage)	0	608	510	719	313

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-7 FPGAs are summarized in Table 35.

Table 35: **Characterization of Double-Precision Format on Virtex-7 FPGAs (Part = XC7V585T-1)**

Operation	Resources						Maximum Frequency (MHz)[1][2]
	Embedded		Fabric			18k Block RAMs	Virtex-7
	Type	Number	LUT-FF Pairs	LUTs	FFs		-1 Speed Grade
Multiply	DSP48E1 (max usage)	11	325	279	421	0	403
	DSP48E1 (full usage)	10	371	299	456	0	389
	DSP48E1 (medium usage)	9	439	356	510	0	378
	Logic	0	2361	2317	2418	0	318
	DSP48E1 (low latency, max usage)	13	309	230	298	0	403
Add/Subtract	DSP48E1 (speed optimized, full usage)	3	895	705	945	0	412
	Logic (speed optimized, no usage)	0	989	794	1029	0	436
	Logic (low latency, no usage)	0	1088	948	1162	0	364
Fixed to float	Int64 input		457	430	481	0	386
Float to fixed	Int64 result		408	376	446	0	427
Float to float	Double to single		109	86	102	0	461
Compare	Programmable		90	86	13	0	373
Divide	RATE=1		4062	3412	5915	0	375
	RATE=55		430	416	352	0	301
Square Root	RATE=1		2865	2005	3242	0	320
	RATE=54		294	288	379	0	321
Reciprocal	DSP48E1 (full usage)	14	420	382	536	0	385
Reciprocal Square Root	DSP48E1 (full usage)	75	2197	1837	2634	1	317
Multiply Flow Control: Blocking, Optimize Goal: Performance	DSP48E1 (max usage)	11	648	418	784	0	415
Add/Subtract Flow Control: Blocking, Optimize Goal: Performance	DSP48E1 (speed optimized, full usage)	3	1148	914	1283	0	380

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 Kintex-7 FPGAs are summarized in Table 36.

Table 36: **Characterization of Double-Precision Format on Kintex-7 FPGAs (Part = XC7K70T-1)**

Operation	Resources						Maximum Frequency (MHz)(1)(2)
	Embedded		Fabric			18K Block RAMs	Kintex-7
	Type	Number	LUT-FF Pairs	LUTs	FFs		-1 Speed Grade
Multiply	DSP48E1 (max usage)	11	330	270	421	0	463
	DSP48E1 (full usage)	10	349	311	456	0	435
	DSP48E1 (medium usage)	9	433	364	510	0	365
	Logic	0	2337	2317	2418	0	309
	DSP48E1 (low latency, max usage)	13	314	218	298	0	375
Add/Subtract	DSP48E1 (speed optimized, full usage)	3	885	739	945	0	428
	Logic (speed optimized, no usage)	0	967	818	1029	0	459
	Logic (low latency, no usage)	0	1033	940	1162	0	399
Fixed to float	Int64 input		449	434	481	0	395
Float to fixed	Int64 result		407	371	446	0	421
Float to float	Double to single		102	94	102	0	455
Compare	Programmable		90	86	13	0	474
Divide	RATE=1		3772	3544	5915	0	326
	RATE=55		430	415	352	0	278
Square Root	RATE=1		2202	1986	3242	0	303
	RATE=54		291	288	379	0	313
Reciprocal	DSP48E1 (full usage)	14	423	380	536	0	430
Reciprocal Square Root	DSP48E1 (full usage)	75	2157	1858	2634	1	334
Multiply Flow Control: Blocking, Optimize Goal: Performance	DSP48E1 (max usage)	11	611	450	784	0	427
Add/Subtract Flow Control: Blocking, Optimize Goal: Performance	DSP48E1 (speed optimized, full usage)	3	1161	908	1283	0	454

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-6 FPGAs are summarized in Table 37.

Table 37: **Characterization of Double-Precision Format on Virtex-6 FPGAs (Part = XC6VLX75-1)**

Operation	Resources						Maximum Frequency (MHz)[1][2]
	Embedded		Fabric			18k Block RAMs	Virtex-6
	Type	Number	LUT-FF Pairs	LUTs	FFs		-1 Speed Grade
Multiply	DSP48E1 (max usage)	11	332	270	421	0	407
	DSP48E1 (full usage)	10	363	305	456	0	398
	DSP48E1 (medium usage)	9	414	372	515	0	405
	Logic	0	2367	2309	2418	0	285
	DSP48E1 (low latency, max usage)	13	319	217	298	0	404
Add/Subtract	DSP48E1 (speed optimized, full usage)	3	848	708	945	0	394
	Logic (speed optimized, no usage)	0	917	783	1034	0	396
	Logic (low latency, no usage)	0	995	944	1163	0	376
Fixed to float	Int64 input		441	425	481	0	388
Float to fixed	Int64 result		413	372	446	0	373
Float to float	Double to single		101	94	102	0	488
Compare	Programmable		90	86	13	0	479
Divide	C_RATE=1		3667	3456	5915	0	309
	C_RATE=55		423	415	352	0	271
Square Root	C_RATE=1		2016	1957	3243	0	284
	C_RATE=54		293	288	379	0	284
Reciprocal	DSP48E1 (full usage)	14	426	378	536	0	382
Reciprocal Square Root	DSP48E1 (full usage)	75	2170	1862	2644	1	323
Multiply Flow Control: Blocking, Optimize Goal: Performance	DSP48E1 (max usage)	11	593	466	784	0	394
Add/Subtract Flow Control: Blocking, Optimize Goal: Performance	DSP48E1 (speed optimized, full usage)	3	1124	912	1283	0	382

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.

References

1. *ANSI/IEEE, IEEE Standard for Binary Floating-Point Arithmetic,* ANSI/IEEE Standard 754-1985. IEEE-754.
2. The GNU Multiple Precision Arithmetic (GMP) Library gmplib.org
3. The GNU Multiple Precision Floating-Point Reliable (MPFR) Library www.mpfr.org
4. The GNU Multiple Precision Integers and Rationals (MPIR) library www.mpir.org
5. *Xilinx AXI Reference Guide* (UG761)
6. AMBA 4 AXI4-Stream Protocol Version 1.0 Specification

Support

Xilinx provides technical support for this LogiCORE IP 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 LogiCORE IP module is included at no additional cost with the Xilinx ISE Design Suite software and is provided under the terms of the Xilinx End User License Agreement. Use the CORE Generator software included with the ISE Design Suite to generate the core. For more information, visit the core page.

Information about additional Xilinx LogiCORE IP modules is available at the Xilinx IP Center. For pricing and availability of other Xilinx LogiCORE IP modules and software, contact your local Xilinx sales representative.

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
06/22/11	1.0	Initial Xilinx release. Previous version of this data sheet is DS335.
10/19/11	1.1	Added System Generator for DSP information.
01/18/12	1.2	Bit Accurate C Model, page 19 updated to reflect new method of delivery of C model through CORE Generator. Software drivers row added to LogiCORE IP Facts Table.

Notice of Disclaimer

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at http://www.xilinx.com/warranty.htm; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: http://www.xilinx.com/warranty.htm#critapps.