Overview

The Xilinx® LogiCORE™ IP Xilinx Complex Multiplier implements high-performance, optimized complex multipliers based on user-specified options.

All operands and the results are represented in signed two's complement format. The operand widths and the result width are parameterizable.

Features

- Drop-in module for Virtex®-6, Virtex-5, Virtex-4, Spartan®-6, Spartan-3/XA, Spartan-3E/XA and Spartan-3A/XA/AN/3A DSP FPGAs
- 8-bit to 63-bit input precision and up to 127-bit output precision
- Supports truncation or unbiased rounding
- 3 or 4 real multiplier implementation options
- Option to use LUTs or embedded multipliers/XtremeDSP slices
- Resource estimation in the Xilinx CORE Generator™ graphical user interface (GUI)
- For use with Xilinx CORE Generator™, Xilinx AccelDSP™ Synthesis Tool and Xilinx System Generator for DSP™ v11.1 or later.

Functional Description

There are two basic architectures to implement complex multiplication, given two operands: $a = a_r + ja_i$ and $b = b_r + jb_i$, yielding an output $p = ab = p_r + jp_i$.

Direct implementation requires four real multiplications:

$$p_r = a_r b_r - a_i b_i \qquad \text{Equation 1}$$

$$p_i = a_r b_i + a_i b_r \qquad \text{Equation 2}$$

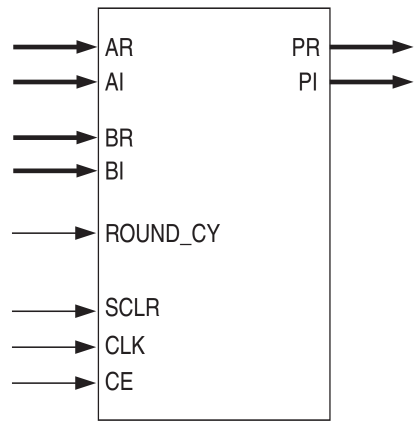
By exploiting that

$$p_r = a_r b_r - a_i b_i = a_r(b_r + b_i) - (a_r + a_i)b_i \qquad \text{Equation 3}$$

$$p_i = a_r b_i + a_i b_r = a_r(b_r + b_i) + (a_i - a_r)b_r \qquad \text{Equation 4}$$

a three real multiplier solution can be devised, which trades off one multiplier for three pre-combining adders and increased multiplier wordlength.

Pinout



xip307

Figure 1: **Core Schematic Symbol**

This section describes the Complex Multiplier core signals as shown in Figure 1 and described in Table 1. All control inputs are Active High. Should an Active Low input be required for a specific control pin, an inverter must be placed in the path to the pin and will be absorbed appropriately during synthesis and/or mapping.

Table 1: **Core Signal Pinout**

Name	Direction	Description
AR[N-1:0]	Input	Real component of operand *A*, N bits wide
AI[N-1:0]	Input	Imaginary component of operand *A*, N bits wide
BR[M-1:0]	Input	Real component of operand *B*, M bits wide
BI[M-1:0]	Input	Imaginary component of operand *B*, M bits wide
CLK	Input	Rising-edge clock
CE	Input	Active high Clock Enable
SCLR	Input	Active high Synchronous Clear (SCLR/CE priority is configurable)
ROUND_CY	Input	Carry input to facilitate unbiased rounding
PR[X:Y]	Output	Real component of product *P*
PI[X:Y]	Output	Imaginary component of product *P*

CORE Generator Graphical User Interface

The Complex Multiplier core GUI has a number of fields to set parameter values for the particular instantiation required. This section provides a description of each GUI field.

- **Component Name**: The name of the core component to be instantiated. The name must begin with a letter and be composed of the following characters: a to z, 0 to 9 and "_".
- **Input Options:** Select the required operand widths. The widths for each operand apply to both the real and imaginary components of each operand.
- **Construction and Optimization:** These options allow the choice of resources to use, and optimization for speed or lower resource use.
  - **Complex Multiplier Construction:** Allows the choice of using LUTs (slice logic) to construct the complex multiplier, or using embedded multipliers / XtremeDSP slices.
  - **Optimization Options:** Selects between Resource and Performance optimization.
    - If the complex multiplier is to be constructed from LUTs (Use LUTs), the only optimization option is Resources.
    - If constructed from embedded multipliers / XtremeDSP slices (Use Mults), Resource or Performance optimization can be selected. In general, Resource optimization will use the 3 real multiplier structure. The core uses the 4 real multiplier structure when the 3 real multiplier structure uses more multiplier resources. Performance optimization always uses the 4 real multiplier structure to allow the best clock frequency performance to be achieved.
- **Output Product Range:** Select the required MSB and LSB of the output product. The values are automatically initialized to provide the full-precision product when the A and B operand widths are set. The output is sign-extended if required. If rounding is required, set the Output LSB to a value greater than zero to enable the rounding options.
- **Rounding:** If the full-precision product is selected, no rounding options are available. When the Output LSB is greater than zero, either Truncation or Random Rounding can be selected. When Random Rounding is selected, the ROUND_CY pin will be enabled by the GUI so that it is present on the core. See the Rounding section for further details.
- **Control Signals:** Select which control signals should be present on the core, and their priority over each other
  - **Clock Enable:** Enables the clock enable (CE) pin on the core. All registers in the core will be enabled by this signal.

- **Synchronous Clear:** Enables the synchronous clear (SCLR) pin on the core. All registers in the core will be reset by this signal. This can increase resource use and degrade performance, as the number of SRL-based shift registers that can be used is reduced.
- **SCLR/CE Priority:** When both SCLR and CE pins are present, the priority of SCLR and CE can be selected. The fewest resources are used, and best performance is achieved, when SCLR overrides CE.

- **Latency and Resource Estimation Tab**: Clicking on the tab below the GUI symbol displays an estimate of the embedded multiplier / XtremeDSP slice resources used for a particular complex multiplier configuration. The latency of the configuration from the A/B inputs to the PR/PI outputs is also shown. These values update instantaneously with changes in the GUI, allowing tradeoffs in implementation to be evaluated immediately.

Using the Complex Multiplier IP core

The CORE Generator GUI performs error-checking on all input parameters. Resource estimation and 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 a number of options for simulation models:

- VHDL behavioral model in the xilinxcorelib library
- VHDL unisim structural model
- Verilog unisim structural model

Xilinx recommends that simulations utilizing unisim-based structural models are 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 models might produce incorrect results if simulation with a resolution other than 1 ps. See the "Register Transfer Level (RTL) Simulation Using Xilinx Libraries" section in *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 2 defines valid entries for the XCO parameters. Parameters are not case sensitive. Default values are displayed in bold.

Xilinx strongly suggests that XCO parameters are not manually edited in the XCO file; instead, use the CORE Generator GUI to configure the core and perform range and parameter value checking.

Table 2: **XCO Parameters**

XCO Parameter	Valid Values
component_name	ASCII text using characters: a..z, 0..9 and '_' ; starting with a letter
APortWidth	8 - 63 (default value is **16**)
BPortWidth	8 - 63 (default value is **16**)
OutputWidthHigh	APortWidth+BPortWidth (default value is **32**)
OutputWidthLow	0 - APortWidth+BPortWidth (default value is **0**)

Table 2: **XCO Parameters *(Cont'd)***

MultType	Use_LUTs, **Use_Mults**
OptimizeGoal	**Resources**, Performance
RoundMode	Truncate, Random_Rounding
ClockEnable	**false**, true
SyncClear	false, true
SclrCePriority	**SCLR_overrides_CE**, CE_overrides_SCLR

Migrating to Complex Multiplier v3.0 from Earlier Versions

XCO Parameter Changes

The CORE Generator core update functionality is used to update an existing XCO file from Complex Multiplier v2.1 or v2.0 to Complex Multiplier v3.0. The XCO parameters in Complex Multiplier v3.0 are different from the parameters in previous versions of the core.

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

Port Changes

There are no differences in port naming conventions, polarities, priorities or widths between versions. Clock enable is now an optional pin v3.0, and clock enable/synchronous clear priority is also now configurable.

Latency changes

The latency of Complex Multiplier v3.0 might be different compared to the previous implementation, and the update process cannot account for this. The CORE Generator GUI for both versions should be consulted and the latency reports checked. If the latencies are different, additional registers might need to be added to the larger design, or additional latency accommodated.

System Generator for DSP Graphical User Interface

This section describes each tab of the System Generator GUI and details the parameters that differ from the CORE Generator GUI.

The Complex Multiplier core may be found in the Xilinx Blockset in the Math section. The block is called "Complex Multiplier".

See the System Generator for DSP Help page for the "Mult" block for more information on parameters not mentioned here.

Page 1:

Page 1 is used to specify the complex multiplier construction, optimization options and output width in a similar way to the CORE Generator GUI. Setting "Output LSB" to a value greater than zero enables the rounding options on Page 2.

Page 2:

Page 2 is used to specify output product rounding options and block control signals.

- **Output Rounding**: Selecting the "Random Rounding" option will add a ROUND_CY pin to the block to allow a carry-in bit to be input.

- **en:** Checking this box specifies if the core will have a clock enable pin (the equivalent of selecting the CE option in the CORE Generator GUI).
- **rst**: Checking this box specifies if the core will have a synchronous reset pin (the equivalent of selecting the SCLR option in the CORE Generator GUI).

Implementation:

This page is used only for System Generator for DSP FPGA area estimation, and has no equivalent parameters on the CORE Generator GUI.

Control Signals

Clock Enable

If the CE pin is present on the core, driving the pin Low will pause the core in its current state. All logic within the core will be paused. Driving the CE pin High will allow the core to continue processing.

If CE and SCLR priority is set such that SCLR overrides CE, asserting SCLR while CE is held Low will still reset the core.

Synchronous Clear

If the SCLR pin is present on the core, driving the pin High results in all registers being reset to their initial values.

If CE and SCLR priority is set such that CE overrides SCLR, SCLR will only reset the core when CE is asserted.

Rounding

In a DSP system, especially if the system contains feedback, the wordlength growth through the multiplier should be offset by quantizing the results. Quantization, or reduction in word-length, results in error, introduces quantization noise and can introduce bias. For best results it is favorable to select a quantization method which introduces zero mean noise and minimizes noise variance. Figure 2 illustrates the quantization method used for truncation.

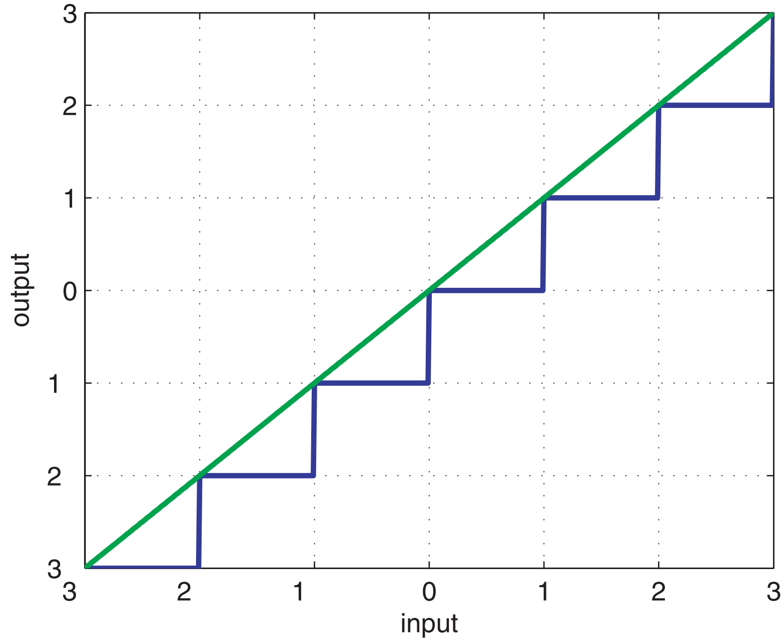


Figure 2: **Truncation**

For truncation the probability density function (PDF) of the noise is:

$$p(e) = \begin{cases} \frac{1}{\Delta} & -\Delta < e < 0 \\ 0 & \text{otherwise} \end{cases}$$ *Equation 5*

therefore the mean and the variance of the error introduced are:

$$m_e = \int_{-\Delta}^{0} e p(e) de = \frac{1}{\Delta} \int_{-\Delta}^{0} e de = -\frac{\Delta}{2}$$ *Equation 6*

$$\sigma_e^2 = \int_{0}^{\Delta} e^2 p(e) de = \frac{1}{\Delta} \int_{0}^{\Delta} e^2 de = \frac{\Delta^2}{3}$$ *Equation 7*

Implementing truncation has no cost in hardware; the fractional bits are simply trimmed.

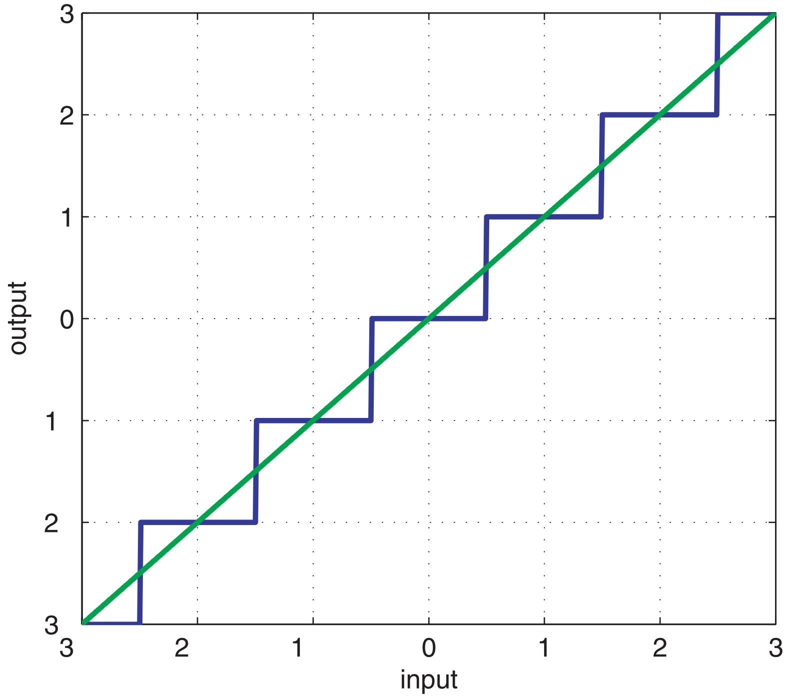


Figure 3: **Rounding**

For rounding the (PDF) of the noise is:

$$p(e) = \begin{cases} \dfrac{1}{\Delta} & -\Delta/2 < e < \Delta/2 \\ 0 & \text{otherwise} \end{cases} \quad \textit{Equation 8}$$

the mean and the variance of the error introduced are:

$$m_e = \int_{-\Delta/2}^{\Delta/2} ep(e)de = \frac{1}{\Delta}\int_{-\Delta/2}^{\Delta/2} ede = 0 \quad \textit{Equation 9}$$

$$\sigma_e^2 = \int_{-\Delta/2}^{\Delta/2} e^2 p(e)de = \frac{1}{\Delta}\int_{-\Delta/2}^{\Delta/2} e^2 de = \frac{\Delta^2}{12} \quad \textit{Equation 10}$$

Therefore, the ideal rounder introduces no DC bias to the signal flow. If the full product word (for example, $a_r b_r$ - $a_i b_i$) is represented with B_P bits, and the actual result of the core (for example, p_r) is represented with B_R bits, then bits B_P-1...B_P-B_R are the integer part, and B_P-B_R-1..0 are the fractional part of the result.

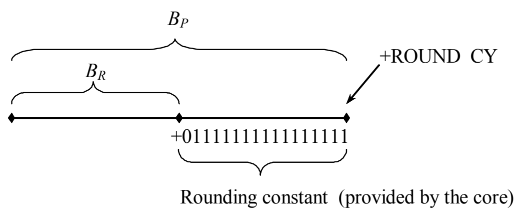


Figure 4: **Rounding to B_r Bits from B_p Bits**

To implement the rounding function shown in Figure 3, 0.5 (represented in B_P.B_P-B_R format) has to be added to the full product word then the lower B_P-B_R bits need to be truncated. However, if the fractional part is exactly 0.5, this method always rounds up, which introduces positive bias to the computation. Also, if the rounding constant is -1 (Figure 4), 0.5 would be always rounded down, introducing negative bias.

If 0.5 is rounded using a static rule, the resulting quantization will always introduce bias. To avoid bias rounding has to be randomized. Therefore, the core adds a rounding constant, and an extra 1 should be added with ½ probability, thus dithering the exact rounding threshold. Typical round carry sources being used extensively as control signals are listed in Table 3.

Table 3: **Unbiased Rounding Sources**

0.5 Rounding Rule	Round Carry Source
Round towards 0	-MSB(P)
Round towards +/- infinity	MSB(P)
Round towards nearest even	LSB(P)

Rounding of the results is not trivial when multiple, cascaded XtremeDSP slices are involved in the process, such as evaluation of equations 10 or 11. The sign of the output (MSB_o) can not be predicted from the operands before the actual multiplications and additions take place, and would incur additional latency or resource to implement outside the XtremeDSP slices. Therefore an external signal should be used to feed the round carry input, through the *ROUND_CY* pin.

A good candidate for a source can be a clock-dividing flip-flop, or any 50% duty cycle random signal, which is not correlated with the fractional part of the results. For predictable behavior (as for bit-true modeling) the *ROUND_CY* signal might need to be connected to a CLK independent source in the user design, such as an LSB of one of the complex multiplier inputs.

Nevertheless, even when a static rule is used (such as tying *ROUND_CY* = '0'), bias and quantization error are reduced compared to using truncation.

In many cases, for XtremeDSP slice implementation, the addition of the rounding constant is 'free', as the C port and carry-in input may be utilized. In devices without XtremeDSP slices, the addition of rounding typically requires an extra slice-based adder and an additional cycle of latency.

Hardware Implementation

Three Real Multiplier Solution

The three real multiplier implementation maps well to devices which have a pre-adder as part of the XtremeDSP slice (such as Spartan-3A DSP devices), saving slice resources.

In general, the three multiplier solution will use more slice resources (LUTs/flipflops) and have a lower maximum achievable clock frequency than the four multiplier solution.

Four Real Multiplier Solution

The four real multiplier solution makes maximum use of embedded multiplier/XtremeDSP slice resources, and has higher clock frequency performance than the three real multiplier solution, in many cases reaching the maximum clock frequency of the FPGA device.

It will still consume slice resources for pipeline balancing, but this slice cost is always less than that required by the equivalent three real multiplier solution.

LUT-based Solution

The core offers the option to build the complex multiplier using LUTs only. While this option uses a significant number of slices, achieves a lower maximum clock frequency and uses more power than embedded multiplier/XtremeDSP slice implementations, it might be suitable for applications where embedded multipliers/XtremeDSP slices are in limited supply (such as Virtex-4 and Virtex-5 LX devices), or where lower clock rates are in use.

The three real multiplier configuration is used exclusively when LUT implementation is selected.

Performance and Resource Usage

Table 4 through Table 7 provide performance and resource usage information for a number of different core configurations.

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 to implement a single Complex Multiplier. 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 Virtex-5 FPGA test cases in Table 4 used an XC5VLX50-FF676 (-1 speed grade) device and ISE speed file version "PRODUCTION 1.64 2009-02-10, STEPPING level 0".

Table 4: **Virtex-5 Performance and Resource Usage**

Complex Multiplier Configuration	Core Latency (Cycles)	Maximum Clock Frequency (MHz)	LUT/FF Pairs	LUT6s	FFs	XtremeDSP Slices
16x16 Use Mults Resources	6	450	117	117	83	3
16x16 Use Mults Performance	4	450	0	0	0	4
16x16 Use LUTs Resources	6	374	1095	1035	1080	0
32x16 Use Mults Resources	9	439	380	190	314	6
32x16 Use Mults Performance	6	450	110	50	110	8
32x16 Use LUTs Resources	7	296	2067	1923	2004	0
32x32 Use Mults Resources	13	365	660	464	594	12
32x32 Use Mults Performance	10	450	286	241	286	16
32x32 Use LUTs Resources	7	247	3810	3721	3697	0

The Spartan-3A DSP FPGA test cases in Table 5 used an XC3SD3400-FG676 (-4 speed grade) device and ISE speed file version "PRODUCTION 1.33 2009-02-10".

Table 5: **Spartan-3A DSP Performance and Resource Usage**

Complex Multiplier Configuration	Core Latency (Cycles)	Maximum Clock Frequency (MHz)	Slices	LUT4s	FFs	XtremeDSP Slices
16x16 Use Mults Resources	6	250	32	64	64	3
16x16 Use Mults Performance	4	250	0	0	0	4
16x16 Use LUTs Resources	6	174	574	1053	1053	0
32x16 Use Mults Resources	11	191	233	273	297	6
32x16 Use Mults Performance	7	250	74	80	110	8
32x16 Use LUTs Resources	7	145	1067	1948	1965	0
32x32 Use Mults Resources	17	181	392	573	615	12
32x32 Use Mults Performance	12	250	201	224	318	16
32x32 Use LUTs Resources	7	129	1953	3721	3697	0

The Spartan-3A FPGA test cases in Table 6 used an XC3S1400A-FG676 (-4 speed grade) device and ISE speed file version "PRODUCTION 1.41 2009-02-10".

Table 6: **Spartan-3A Performance and Resource Usage**

Complex Multiplier Configuration	Core Latency (Cycles)	Maximum Clock Frequency (MHz)	Slices	LUT4s	FFs	Embedded Multipliers
16x16 Use Mults Resources	5	195	136	117	264	3
16x16 Use Mults Performance	4	195	98	66	194	4
16x16 Use LUTs Resources	6	172	574	1053	1053	0
32x16 Use Mults Resources	6	147	273	328	537	6
32x16 Use Mults Performance	5	162	282	222	546	8
32x16 Use LUTs Resources	7	142	1067	1948	1965	0
32x32 Use Mults Resources	7	132	537	670	1051	12
32x32 Use Mults Performance	6	132	658	574	1286	16
32x32 Use LUTs Resources	7	132	1953	3721	3697	0

The Virtex-4 FPGA test cases in Table 7 used an XC4VLX40-FF668 (-10 speed grade) device and ISE speed file version "PRODUCTION 1.69 2009-02-10, STEPPING level 1".

Table 7: **Virtex-4 Performance and Resource Usage**

Complex Multiplier Configuration	Core Latency (Cycles)	Maximum Clock Frequency (MHz)	Slices	LUT4s	FFs	XtremeDSP Slices
16x16 Use Mults Resources	6	379	61	117	83	3
16x16 Use Mults Performance	4	400	0	0	0	4
16x16 Use LUTs Resources	6	307	574	1023	1101	0
32x16 Use Mults Resources	9	315	226	192	314	6
32x16 Use Mults Performance	6	400	74	50	110	8
32x16 Use LUTs Resources	7	243	1067	1903	2035	0
32x32 Use Mults Resources	13	304	384	473	599	12
32x32 Use Mults Performance	10	400	201	194	318	16
32x32 Use LUTs Resources	7	212	1953	3721	3697	0

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*.

Ordering Information

This core may be downloaded from the Xilinx IP Center for use with the Xilinx CORE Generator v11.1 and later. The Xilinx CORE Generator system is shipped with Xilinx ISE Foundation™ Series Development software.

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

Information on additional Xilinx LogiCORE modules is available on the Xilinx IP Center.

Revision History

Date	Version	Description of Revisions
5/21/04	1.0	Initial Xilinx release of the core.
11/11/04	2.0	Document updated to support Complex Multiplier core v2.0 and Xilinx software v6.3i.
4/28/05	2.1	Document updated to support Complex Multiplier core v2.1 and Xilinx software v7.1i.
12/22/08	3.0-draft-C	Flowed into current DS000-IP template, v3.1
4/29/09	3.0	Documented updated to support Complex Multiplier core v3.0 and ISE 11.1

Notice of Disclaimer