

---

# LogiCORE™ IP Fast Fourier Transform v7.0 Bit Accurate C Model

## *User Guide*

UG459 v3.0 June 24, 2009



Xilinx is providing this product documentation, hereinafter “Information,” to you “AS IS” with no warranty of any kind, express or implied. Xilinx makes no representation that the Information, or any particular implementation thereof, is free from any claims of infringement. You are responsible for obtaining any rights you may require for any implementation based on the Information. All specifications are subject to change without notice.

XILINX EXPRESSLY DISCLAIMS ANY WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE INFORMATION OR ANY IMPLEMENTATION BASED THEREON, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF INFRINGEMENT AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Except as stated herein, none of the Information may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx.

© 2007-2009 Xilinx, Inc. XILINX, the Xilinx logo, Virtex, Spartan, ISE, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. MATLAB is a registered trademark of The MathWorks, Inc. All other trademarks are the property of their respective owners.

---

## Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/17/07	1.0	Initial Xilinx release
09/19/08	2.0	Updated for Fast Fourier Transform v6.0
06/24/09	3.0	Updated for Fast Fourier Transform v7.0

---

# Contents

---

## Preface: About This Guide

Contents .....	5
Conventions .....	6
Typographical .....	6
Online Document .....	7

## Chapter 1: Introduction

Features .....	9
Overview .....	9
Additional Core Resources .....	10
Technical Support .....	10
Feedback .....	10
FFT v7.0 Bit Accurate C Model and IP Core .....	10
Document .....	10

## Chapter 2: User Instructions

Unpacking and Model Contents .....	11
Installation .....	12
Software Requirements .....	12

## Chapter 3: FFT v7.0 Bit Accurate C Model

FFT v7.0 C Model Interface .....	13
Create a State Structure .....	13
Simulate the FFT Core .....	15
Destroy the State Structure .....	20
C Model Example Code .....	20
Compiling with the FFT v7.0 C Model .....	20
Linux (32-bit and 64-bit) .....	20
Windows (32-bit and 64-bit) .....	20
FFT v7.0 MATLAB Software MEX Function .....	21
Building the MEX Function for Other MATLAB Software Versions .....	21
Installing and Running the MEX Function .....	22
MEX Function Example Code .....	25
Modeling Multichannel FFTs .....	26



# *Preface*

## *About This Guide*

---

This user guide provides information about the Xilinx LogiCORE™ IP Fast Fourier Transform v7.0 Bit Accurate C Model for 32-bit and 64-bit Linux and Windows platforms.

### **Contents**

This guide contains the following chapters:

- [Preface, “About this Guide”](#) introduces the organization and purpose of this guide and provides the conventions used in this document.
- [Chapter 1, “Introduction”](#) describes the core and related information, including additional resources, technical support, and submitting feedback to Xilinx.
- [Chapter 2, “User Instructions”](#) describes basic instructions for unpacking, the C model contents, and installation.
- [Chapter 3, “FFT v7.0 Bit Accurate C Model”](#) provides a description of the C model interface, example code, compiling with the C model, the MATLAB® software MEX function, and modeling multichannel FFTs.

## Conventions

This document uses the following conventions.

### Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
Courier font	Messages, prompts, and program files that the system displays	speed grade: - 100
<b>Courier bold</b>	Literal commands you enter in a syntactical statement	<b>ngdbuild</b> design_name
<i>Italics</i>	References to other manuals	See the <i>User Guide</i> for details.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are <i>not</i> connected.
Shading	Unsupported or reserved items	This feature is not supported
Brackets < >	User-defined variable	<project directory>, <filename>
Square brackets [ ]	Optional entry or parameter, with the exception of bus specifications. For bus specifications, brackets are required, for example <b>bus [7:0]</b> .	<b>ngdbuild</b> [option_name] design_name
Braces { }	A list of items from which you must choose one or more	<b>lowpwr = {on   off}</b>
Vertical bar	Separates items in a list of choices	<b>lowpwr = {on   off}</b>
Vertical ellipsis · · ·	Omitted repetitive material	IOB #1: Name = QOUT' IOB #2: Name = CLKIN' · · ·
Horizontal ellipsis ...	Omitted repetitive material	<b>allow block</b> block_name loc1 loc2 ... locn;
Notations	The prefix '0x' or the suffix 'h' indicate hexadecimal notation	A read of address 0x00112975 returns 45524943h
	An '_n' means the signal is active low	usr_teof_n is active low

## Online Document

The following conventions are used in this document for cross-references and links to URLs.

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See “ <a href="#">Additional Resources</a> ” for more information. See “ <a href="#">Title Formats</a> ” in <a href="#">Chapter 1</a> for detailed information.
<a href="#">Blue, underlined text</a>	Hyperlink to a website (URL)	Go to <a href="http://www.xilinx.com">www.xilinx.com</a> for the latest speed files.





# Chapter 1

## Introduction

---

The Xilinx® LogiCORE™ IP Fast Fourier Transform (FFT) v7.0 core has a bit accurate C model designed for system modeling. A MATLAB® software MEX function for seamless MATLAB software integration is also available.

### Features

- Bit accurate with FFT v7.0 core
- Dynamic link library
- Available for 32-bit Linux, 64-bit Linux, 32-bit Windows, and 64-bit Windows platforms
- MATLAB software MEX function
- Supports all features of the FFT core that affect numerical results
- Designed for rapid integration into a larger system model
- Example C++ and M code showing how to use the function is provided

### Overview

The Xilinx® LogiCORE IP FFT v7.0 has a bit accurate C model for 32-bit and 64-bit Linux and 32-bit and 64-bit Windows platforms. The model has an interface consisting of a set of C functions, which reside in a dynamic link library (shared library). Full details of the interface are given in “[FFT v7.0 C Model Interface](#)” in [Chapter 3](#). An example piece of C++ code showing how to call the model is provided. The model is also available as a MATLAB software MEX function for seamless MATLAB software integration.

The model is bit accurate but not cycle accurate, so it produces exactly the same output data as the core on a frame-by-frame basis. However, it does not model the core's latency or its interface signals.

The latest version of the model is available for download on the Xilinx™ LogiCORE IP FFT web page at: [www.xilinx.com/products/ipcenter/FFT.htm](http://www.xilinx.com/products/ipcenter/FFT.htm).

## Additional Core Resources

For detailed information and updates about the FFT v7.0 core, see the following documents, located on the FFT v7.0 product page at:

[www.xilinx.com/products/ipcenter/FFT.htm](http://www.xilinx.com/products/ipcenter/FFT.htm)

- *Fast Fourier Transform v7.0 Product Specification (DS260)*
- *FFT v7.0 Release Notes*

## Technical Support

For technical support, go to [www.xilinx.com/support](http://www.xilinx.com/support). Questions are routed to a team with expertise using the FFT v7.0 core.

Xilinx provides technical support for use of this product as described in *LogiCORE IP Fast Fourier Transform v7.0 Bit Accurate C Model User Guide (UG459)* and the *Fast Fourier Transform v7.0 Product Specification (DS260)*. Xilinx cannot guarantee functionality or support of this product for designs that do not follow these guidelines.

## Feedback

Xilinx welcomes comments and suggestions about the FFT v7.0 core and the accompanying documentation.

### FFT v7.0 Bit Accurate C Model and IP Core

For comments or suggestions about the FFT v7.0 core and bit accurate C model, please submit a WebCase from [www.xilinx.com/support/clearexpress/websupport.htm](http://www.xilinx.com/support/clearexpress/websupport.htm). Be sure to include the following information:

- Product name
- Core version number
- Explanation of your comments

### Document

For comments or suggestions about the FFT v7.0 core documentation, please submit a WebCase from [www.xilinx.com/support/clearexpress/websupport.htm](http://www.xilinx.com/support/clearexpress/websupport.htm). Be sure to include the following information:

- Document title
- Document number
- Page number(s) to which your comments refer
- Explanation of your comments

## Chapter 2

# User Instructions

## Unpacking and Model Contents

Unzip the FFT v7.0 C model zip file. This produces the directory structure and files shown in [Table 2-1](#).

**Table 2-1: Directory Structure and Files of the FFT v7.0 Bit Accurate C Model**

File	Description
README.txt	release notes
xfft_bitacc_cmodel_ug459.pdf	this file
xfft_v7_0_bitacc_cmodel.h	model header file
run_bitacc_cmodel.c	example code calling the C model
lin/	model for 32-bit Linux
libIp_xfft_v7_0_bitacc_cmodel.so	model shared object library
libstlport.so.5.1	STL library, used by model
lin64/	model for 64-bit Linux
libIp_xfft_v7_0_bitacc_cmodel.so	model shared object library
libstlport.so.5.1	STL library, used by model
nt/	model for 32-bit Windows
libIp_xfft_v7_0_bitacc_cmodel.dll	model dynamically linked library
libIp_xfft_v7_0_bitacc_cmodel.lib	model .lib file for compiling
stlport.5.1.dll	STL library, used by model
xfft_v7_0_bitacc_mex.mexw32	MATLAB® software MEX interface to model
nt64/	model for 64-bit Windows
libIp_xfft_v7_0_bitacc_cmodel.dll	model dynamically linked library
libIp_xfft_v7_0_bitacc_cmodel.lib	model .lib file for compiling
stlport.5.1.dll	STL library, used by model
matlab/	files for MATLAB software MEX function
make_xfft_v7_0_mex.m	M file to build MEX function

Table 2-1: Directory Structure and Files of the FFT v7.0 Bit Accurate C Model

File	Description
<code>xfft_v7_0_bitacc_mex.cpp</code>	C++ wrapper for MEX function
<code>run_xfft_v7_0_mex.m</code>	example code calling the MEX function

## Installation

On Linux, ensure that the directory in which the files `libIp_xfft_v7_0_bitacc_cmodel.so` and `libstlport.so.5.1` are located is in your `$LD_LIBRARY_PATH` environment variable.

On Windows, ensure that the directory in which the files `libIp_xfft_v7_0_bitacc_cmodel.dll` and `stlport.5.1.dll` are located is either in your `%PATH%` environment variable, or is the directory in which you will run your executable that calls the FFT v7.0 C model.

## Software Requirements

The FFT v7.0 C model and MEX function were compiled and tested with the following software:

Table 2-2: Supported Software

Platform	C++ Compiler	MATLAB Software
64-bit Linux	GCC 4.1.1	See note 1 and note 2
32-bit Linux	GCC 4.1.1	See note 1 and note 2
64-bit Windows	Microsoft Visual Studio 2005	See note 1
32-bit Windows	Microsoft Visual Studio 2005 (Visual C++ 8.0)	2008a (also see note 1)

### Note:

1. M file scripts are provided in the zip file to allow the MEX function to be compiled for other versions of MATLAB software, and versions on different operating systems. See [“FFT v7.0 MATLAB Software MEX Function”](#) for details.
2. MATLAB software requires at least GCC version 4.0.0 to build MEX functions for Linux platforms.

## Chapter 3

# FFT v7.0 Bit Accurate C Model

## FFT v7.0 C Model Interface

**Note:** An example C++ file, `run_bitacc_cmodel.c` is included that demonstrates how to call the FFT C model. See that file for examples of using the interface described below.

The C model is used through three functions, declared in the header file `xfft_v7_0_bitacc_cmodel.h`:

```
struct xilinx_ip_xfft_v7_0_state* xilinx_ip_xfft_v7_0_create_state
(
    struct xilinx_ip_xfft_v7_0_generics generics
);

int xilinx_ip_xfft_v7_0_bitacc_simulate
(
    struct xilinx_ip_xfft_v7_0_state* state,
    struct xilinx_ip_xfft_v7_0_inputs inputs,
    struct xilinx_ip_xfft_v7_0_outputs* outputs
);

void xilinx_ip_xfft_v7_0_destroy_state
(
    struct xilinx_ip_xfft_v7_0_state* state
);
```

To use the model, first create a state structure using the first function, `xilinx_ip_xfft_v7_0_create_state`. Then run the model using the second function, `xilinx_ip_xfft_v7_0_bitacc_simulate`, passing the state structure, an inputs structure, and an outputs structure to the function. Finally, free up memory allocated for the state structure using the third function, `xilinx_ip_xfft_v7_0_destroy_state`. Each of these functions is described fully in the following sections.

### Create a State Structure

The first function, `xilinx_ip_xfft_v7_0_create_state`, creates a new state structure for the FFT C model, allocating memory to store the state as required, and returns a pointer to that state structure. The state structure contains all information required to define the FFT being modeled. The function is called with a structure containing the core's generics: these are all of the parameters that define the bit accurate numerical performance of the core, represented as integers, and are shown in [Table 3-1](#).

Table 3-1: FFT C Model Generics

Generic	Description	Permitted Values	
C_NFFT_MAX	$\log_2$ (maximum transform length)	3-16	
C_ARCH	Architecture	1 = Radix-4, Burst I/O 2 = Radix-2, Burst I/O 3 = Pipelined, Streaming I/O 4 = Radix-2 Lite, Burst I/O	
C_HAS_NFFT	Run-time configurable transform length	0 = no, 1 = yes	
C_USE_FLT_PT	Core interface	0 = Fixed Point	1 = Single-Precision Floating Point
C_INPUT_WIDTH	Input data width (bits)	8-34	32
C_TWIDDLE_WIDTH	Phase factor width (bits)	8-34	24-25
C_HAS_SCALING	Scaling option: unscaled or determined by C_HAS_BFP	0 = unscaled, 1 = see C_HAS_BFP	0
C_HAS_BFP	Scaling option: Applicable if C_HAS_SCALING=1	0 = scaled, 1 = block floating point	0
C_HAS_ROUNDING	Rounding:	0 = truncation, 1 = convergent rounding	0

**Note:** C\_CHANNELS is not a generic used in the C model. The model is always single channel. To model multiple channels in a multichannel FFT, see “[Modeling Multichannel FFTs](#)”.

The xilinx\_ip\_xfft\_v7\_0\_create\_state function fails with an error message and returns a NULL pointer if any generic or combination of generics is invalid.

## Simulate the FFT Core

After a state structure has been created, it can be used as many times as required to simulate the FFT core. A simulation is run using the second function, `xilinx_ip_xfft_v7_0_bitacc_simulate`. Call this function with the pointer to the existing state structure and structures that hold the inputs and outputs of the C model. The input structure members are shown in [Table 3-2](#).

**Table 3-2: Members of the Inputs Structure**

Member	Type	Description
<code>nfft</code>	<code>int</code>	Transform length
<code>xn_re</code>	<code>double*</code>	Pointer to array of doubles: real part of input data
<code>xn_re_size</code>	<code>int</code>	Number of elements in <code>xn_re</code> array
<code>xn_im</code>	<code>double*</code>	Pointer to array of doubles: imaginary part of input data
<code>xn_im_size</code>	<code>int</code>	Number of elements in <code>xn_im</code> array
<code>scaling_sch</code>	<code>int*</code>	Pointer to array of ints containing scaling schedule
<code>scaling_sch_size</code>	<code>int</code>	Number of elements in <code>scaling_sch</code> array
<code>direction</code>	<code>int</code>	Transform direction: 1=forward FFT, 0=inverse FFT (IFFT)

### Notes:

#### General

1. You are responsible for allocating memory for arrays in the inputs structure.
2. `nfft` input is only used with run-time configurable transform length (that is, `C_HAS_NFFT = 1`). If the transform length is fixed (that is, `C_HAS_NFFT = 0`), `C_NFFT_MAX` is used for `nfft`. In this case, `nfft` should be equal to `C_NFFT_MAX`, and a warning is printed if it is not (but the model continues, using `C_NFFT_MAX` for `nfft` and ignoring the `nfft` value in the inputs structure).
3. `xn_re` and `xn_im` must have  $2^{nfft}$  elements. `xn_re_size` and `xn_im_size` must be set to  $2^{nfft}$ .
4. `xn_re` and `xn_im` may be in natural or bit/digit-reversed sample index order. The C model produces samples in the same ordering format as they were input.

#### FFTs with Fixed-Point Interface

1. Data in `xn_re` and `xn_im` must all be in the range  $-1.0 \leq \text{data} < +1.0$ .
2. Data in `xn_re` and `xn_im` is of type `double`, but the model requires data in signed two's-complement, fixed-point format with precision given by `C_INPUT_WIDTH`. The data has a sign bit, then the binary point, and then  $(C\_INPUT\_WIDTH - 1)$  fractional bits. The model checks the input data to see if it fits within this format and precision. If not, it prints a warning, and internally rounds it using convergent rounding (halfway values are rounded to the nearest even number) to the required precision. To accurately model the FFT core, prequantize the input data to this required precision before passing it to the model.
3. `scaling_sch` and `scaling_sch_size` are ignored entirely unless fixed scaling is used (`C_HAS_SCALING = 1` and `C_HAS_BFP = 0`).

4. `scaling_sch` is an array of integers, each of which indicates the scaling to be done in a stage of the FFT processing. `scaling_sch[0]` is the scaling in the first stage, `scaling_sch[1]` the scaling in the second stage, and so on. *Note that this is the reverse of the scaling schedule vector applied to the IP core.* The number of elements in the `scaling_sch` array and the value of `scaling_sch_size` must be equal to the number of stages in the FFT. This is dependent on the architecture, and on `nfft`, the point size of the transform:
  - a. Radix-4, Burst I/O (`C_ARCH = 1`) or Pipelined, Streaming I/O (`C_ARCH = 3`):  
 $\text{stages} = \text{ceil}(\text{nfft}/2)$
  - b. Radix-2, Burst I/O (`C_ARCH = 2`) or Radix-2 Lite, Burst I/O (`C_ARCH = 4`):  
 $\text{stages} = \text{nfft}$
5. If `C_HAS_NFFT = 0`, `C_NFFT_MAX` is used for `nfft`. The scaling in each stage is an integer in the range 0-3, which indicates the number of bits the intermediate result will be shifted right. So 0 indicates no scaling, 1 indicates a division by 2, 2 indicates a division by 4, and 3 indicates a division by 8. Again, `scaling_sch[0]` is the scaling in the first stage, `scaling_sch[1]` the scaling in the second stage, and so on. Insufficiently large scaling results in overflow, indicated by the overflow output.

#### FFTs with Floating-Point Interface

1. Data in `xn_re` and `xn_im` must all be representable in IEEE-754 single precision 32-bit format.
2. Data in `xn_re` and `xn_im` is of type double, but the model requires data in single-precision format, such that the values may be represented in the 32-bit float datatype. The double values are explicitly cast to the float datatype internally. No range checking is performed by the model prior to casting to float.
3. The model checks the input data for denormalized numbers and if one is found, that sample is set to zero at the input to the model.
4. If an Infinity or Not A Number (NaN) value is detected in the input data, all outputs in that frame are invalidated and set to NaN in the output structure.

The outputs structure, a pointer which is passed to the `xilinx_ip_xfft_v7_0_bitacc_simulate` function, has the members shown in [Table 3-3](#).

**Table 3-3: Members of the Outputs Structure**

Member	Type	Description
<code>xk_re</code>	<code>double*</code>	Pointer to array of doubles: real part of output data
<code>xk_re_size</code>	<code>int</code>	Number of elements in <code>xk_re</code> array
<code>xk_im</code>	<code>double*</code>	Pointer to array of doubles: imaginary part of output data
<code>xk_im_size</code>	<code>int</code>	Number of elements in <code>xk_im</code> array
<code>blk_exp</code>	<code>int</code>	Block exponent (if block floating point is used)
<code>overflow</code>	<code>int</code>	Overflow occurred (if fixed scaling is used with a fixed-point interface, or if a floating point interface is used)



**Notes:****General**

1. You are responsible for allocating memory for the outputs structure and for arrays in the outputs structure.
2. `xk_re` and `xk_im` must have at least  $2^{nfft}$  elements. You must set `xk_re_size` and `xk_im_size` to indicate the number of elements in `xk_re` and `xk_im` before calling the FFT function. On exit, `xk_re_size` and `xk_im_size` are set to the number of elements that contain valid output data in `xk_re` and `xk_im`.
3. The C model produces data in the same ordering format as the input data. Hence, if `xn_re` and `xn_im` were provided in natural sample index order (0,1,2,3...), `xk_re` and `xk_im` samples will also be in natural sample index order.

**FFTs with Fixed-Point Interface**

1. Data in `xk_re` and `xk_im` has the correct precision to model the FFT:
  - a. If the FFT is scaled or has block floating point (`C_HAS_SCALING` = 1, `C_HAS_BFP` = 0 or 1, respectively), data in `xk_re` and `xk_im` is all in the range  $-1.0 \leq \text{data} < +1.0$ , the precision is `C_INPUT_WIDTH` bits with `C_INPUT_WIDTH`-1 fractional bits. For example, if `C_INPUT_WIDTH` = 8, output data is precise to  $2^{-7} = 0.0078125$ , and is in the range -1.0 to +0.9921875, and the binary representation of the output data has the format `s.fyyyyyy`, where `s` is the sign bit and `f` is a fractional bit.
  - b. If the FFT is unscaled (`C_HAS_SCALING` = 0), data in `xk_re` and `xk_im` grows beyond  $\pm 1.0$ , such that the binary point remains in the same place and there are still (`C_INPUT_WIDTH` - 1) fractional bits after the binary point. In total, the output precision is (`C_INPUT_WIDTH` + `C_NFFT_MAX` + 1) bits. For example, if `C_INPUT_WIDTH` = 8 and `C_NFFT_MAX` = 3, output data is precise to  $2^{-7} = 0.0078125$ , and is in the range -16.0 to +15.9921875, and the binary representation of the output data has the format `siiii.fyyyyyy`, where `s` is the sign bit, `i` is an integer bit, and `f` is a fractional bit.
2. `blk_exp` is the integer block exponent. It is only valid (and non-zero) if block floating point is used (`C_HAS_SCALING` = 1 and `C_HAS_BFP` = 1). It indicates the total number of bits that intermediate values were shifted right during the FFT processing. For example, if `blk_exp` = 5, the output data has been divided by 32 relative to the magnitude of the input data.
3. Overflow indicates if overflow occurred during the FFT processing. It is only valid (and non-zero) if fixed scaling is used (`C_HAS_SCALING` = 1 and `C_HAS_BFP` = 0). A value of 0 indicates that overflow did not occur; a value of 1 indicates that overflow occurred at some stage in the FFT processing. To avoid overflow, increase the scaling at one or more stages in the scaling schedule (`scaling_sch` input).
4. If overflow occurred with the Pipelined, Streaming I/O architecture (`C_ARCH` = 3), due to differences between the FFT core and the model in the order of operations within the processing stage, the data in `xk_re` and `xk_im` may not match the `XK_RE` and `XK_IM` outputs of the FFT core. The `xk_re` and `xk_im` data must be ignored if the overflow output is 1. This is the only case where the model is not entirely bit accurate to the core.

### FFTs with Floating-Point Interface

1. Data in `xk_re` and `xk_im` has the correct precision to model the FFT. The double precision output can be cast to single precision without introducing error.
2. Overflow indicates if floating point exponent overflow occurred during the FFT processing. A value of 0 indicates that overflow did not occur; a value of 1 indicates that overflow occurred. Overflow is not set when a NaN value is present on the output. NaN values can only occur at the FFT output when the input data frame contains samples with value NaN or  $\pm$  Infinity.
3. If overflow occurred, the output sample which overflowed will be set to  $\pm$  Infinity, depending on the sign of the internal result.

The `xilinx_ip_xfft_v7_0_bitacc_simulate` function checks the input and output structures for errors. If the model finds a problem, it prints an error message and returns a value `xilinx_ip_xfft_v7_0_bitacc_simulate` function as shown in [Table 3-4](#).

Table 3-4: xilinx\_ip\_xfft\_v7\_0\_bitacc\_simulate Function Return Values

Return Value	Meaning
0	Success.
1	state structure is NULL.
2	outputs structure is NULL.
3	state structure is corrupted (Radix-4, Burst I/O architecture).
4	state structure is corrupted (Radix-2 [Lite], Burst I/O architecture).
5	state structure is corrupted (Pipelined, Streaming I/O architecture).
6	nfft in inputs structure out of range (Radix-4, Burst I/O architecture).
7	nfft in inputs structure out of range (other architectures).
8	xn_re in inputs structure is a NULL pointer.
9	xn_re_size in inputs structure is incorrect.
10	data value in xn_re in inputs structure out of range -1.0 to < +1.0 (fixed point input data only).
11	xn_im in inputs structure is a NULL pointer.
12	xn_im_size in inputs structure is incorrect.
13	data value in xn_im in inputs structure is out of range -1.0 to < +1.0 (fixed point input data only).
14	scaling_sch in inputs structure is a NULL pointer.
15	scaling_sch_size in inputs structure is incorrect (Radix-4, Burst I/O or Pipelined, Streaming I/O architectures).
16	scaling_sch_size in inputs structure is incorrect (Radix-2, Burst I/O or Radix-2 Lite, Burst I/O architectures).
17	scaling value in scaling_sch in inputs structure out of range 0-3.
18	scaling value for final stage in scaling_sch in inputs structure out of range 0-1 when nfft is odd and architecture is Radix-4, Burst I/O or Pipelined, Streaming I/O.
19	direction in inputs structure is out of range 0-1.
20	xk_re in outputs structure is a NULL pointer.
21	xk_im in outputs structure is a NULL pointer.
22	xk_re_size in outputs structure is too small.
23	xk_im_size in outputs structure is too small.

If the xilinx\_ip\_xfft\_v7\_0\_bitacc\_simulate function returns 0 (zero), it completed successfully and the outputs of the model are in the outputs structure.

## Destroy the State Structure

Finally, the state structure must be destroyed to free up memory used to store the state, using the third function, `xilinx_ip_xfft_v7_0_destroy_state`, called with the pointer to the existing state structure.

If the generics of the core need to be changed, destroy the existing state structure and create a new state structure using the new generics. There is no way to change the generics of an existing state structure.

## C Model Example Code

An example C++ file, `run_bitacc_cmodel.c`, is provided. This demonstrates the steps required to run the model: set up generics, create a state structure, create inputs and outputs structures, simulate the FFT, use the outputs, and finally destroy the state structure. The code works for all legal combinations of generics. Simply modify the `const int` declarations of generics at the start of function `main()`. The code also illustrates how to model a multichannel FFT; see [“Modeling Multichannel FFTs.”](#)

The example code can be used to test your compilation process. See [“Compiling with the FFT v7.0 C Model.”](#)

## Compiling with the FFT v7.0 C Model

Place the header file, `xfft_v7_0_bitacc_cmodel.h`, with your other header files.

Compilation varies from platform to platform.

### Linux (32-bit and 64-bit)

To compile the example code, `run_bitacc_cmodel.c`, first ensure that the directory in which the files `libIp_xfft_v7_0_bitacc_cmodel.so` and `libstlport.so.5.1` are located is present on your `$LD_LIBRARY_PATH` environment variable. These shared libraries are referenced during the compilation and linking process.

Place the header file and C++ source file in a single directory. Then in that directory, compile using the GNU C++ Compiler:

```
g++ -x c++ run_bitacc_cmodel.c -o run_bitacc_cmodel -L.  
-lIp_xfft_v7_0_bitacc_cmodel -Wl,-rpath,.
```

### Windows (32-bit and 64-bit)

Link to the import library `libIp_xfft_v7_0_bitacc_cmodel.lib`. For example, for Microsoft Visual Studio.NET, in Project Properties, under Linker -> Input, for Additional Dependencies, specify `libIp_xfft_v6_0_bitacc_cmodel.lib`.

## FFT v7.0 MATLAB Software MEX Function

The FFT v7.0 model is available as a compiled MATLAB® software MEX function for seamless integration with MATLAB software. The FFT MEX function provides a pre-compiled MATLAB software interface to the FFT C model. The FFT MEX function and FFT C model produce identical results and both are bit accurate to the FFT core.

A pre-compiled FFT MEX function is supplied in the `nt` directory in the file `xfft_v7_0_bitacc_mex.mexw32`. It was built using MATLAB software 7.6.0 (R2008a).

### Building the MEX Function for Other MATLAB Software Versions

A C++ wrapper and compilation script are also provided to allow the MEX function to be built for MATLAB software versions other than R2008a and for other operating systems.

The FFT v7.0 C model does not support the LCC compiler shipped with MATLAB software.

Xilinx has verified that the Microsoft Visual Studio 2005 (v8.0) C++ compiler can successfully be used to build the MEX function on 32-bit Windows.

Xilinx has also verified that GCC version 4.1.1 can successfully be used to build the MEX function on 32-bit and 64-bit Linux.

To build the MEX function:

1. Start the MATLAB software.
2. Change directory to the unzipped FFT v7.0 C model installation
3. Copy the `xfft_v7_0_bitacc_cmodel.h` header file into the `matlab` directory
  - a. If building on a Windows platform, also copy the `libIp_xfft_v7_0_bitacc_cmodel.lib` file from the `nt` or `nt64` directory (as appropriate) into the `matlab` directory
  - b. If building on a Linux platform, also copy the `libIp_xfft_v7_0_bitacc_cmodel.so` file from the `lin` or `lin64` directory (as appropriate) into the `matlab` directory.
4. Change directory to the `matlab` directory
5. Use the **`mex -setup`** command at the MATLAB software command line to set up the compiler. For more details on the `mex` command and the arguments it accepts, type **`help mex`** at the MATLAB software command line
6. Execute the `make_xfft_v7_0_mex.m` file in the MATLAB software to build the MEX function. It will check that the necessary files from the preceding steps 3a) or 3b) are present.
7. Verify that a file named `xfft_v7_0_bitacc_mex.mex<suffix>` is now present in the `matlab` directory. The `<suffix>` portion of the filename depends on the platform the function was compiled on.

## Installing and Running the MEX Function

To install the FFT MEX function, place the MEX file in your MATLAB software working directory, or in the MATLAB software, change directory to the directory containing the MEX file.

**Note:** For Windows platforms, the correct `libIp_xfft_v7_0_bitacc_cmodel.dll` and `stlport.5.1.dll` files must be copied from the `nt` or `nt64` (as appropriate) directory to the directory where the FFT MEX function has been installed before running the MEX function.

**Note:** For Linux platforms, the `libIp_xfft_v7_0_bitacc_cmodel.so` and `libstlport.so.5.1` files must be copied from the `lin` or `lin64` directory (as appropriate) to the directory where the FFT MEX function has been installed before running the MEX function, or be visible via the `$LD_LIBRARY_PATH` environment variable. `$LD_LIBRARY_PATH` must be set correctly before starting MATLAB software.

The FFT MEX function is called `xfft_v7_0_bitacc_mex`. Enter this function name without arguments at the MATLAB software command line to see usage information. The FFT MEX function syntax is:

```
[output_data, blk_exp, overflow] = xfft_v7_0_bitacc_mex(generics, nfft,
input_data, scaling_sch, direction)
```

The function's inputs are shown in [Table 3-5](#).

**Table 3-5: FFT MEX Function Inputs**

Input	Description	Permitted values	
generics	Core parameters. Single-element, 9-field structure containing all relevant generics defining the core		
generics.C_NFFT_MAX	$\log_2$ (maximum transform length)	3-16	
generics.C_ARCH	Architecture	1 = Radix-4, Burst I/O, 2 = Radix-2, Burst I/O, 3 = Pipelined, Streaming I/O, 4 = Radix-2 Lite, Burst I/O	
generics.C_HAS_NFFT	Run-time configurable transform length	0 = no, 1 = yes	
generics.C_USE_FLT_PT	Core interface	0 = fixed point	1 = single precision floating point
generics.C_INPUT_WIDTH	Input data width	8-34 bits	32 bits
generics.C_TWIDDLE_WIDTH	Phase factor width	8-34 bits	24-25 bits
generics.C_HAS_SCALING	Type of scaling	0 = unscaled, 1 = other	0
generics.C_HAS_BFP	Type of scaling if <code>C_HAS_SCALING = 1</code>	0 = scaled, 1 = block floating point	0
generics.C_HAS_ROUNDING	Type of rounding	0 = truncation, 1 = convergent rounding	0

Table 3-5: FFT MEX Function Inputs (*Continued*)

Input	Description	Permitted values	
nfft	$\log_2$ (transform length) for this transform. Single integer.	Maximum value is C_NFFT_MAX. Minimum value is 6 for Radix-4, Burst I/O architecture, or 3 for other architectures.	Maximum value is C_NFFT_MAX. Minimum value is 6 for Radix-4, Burst I/O architecture, or 3 for other architectures.
input_data	Input data. 1D array of complex data with $2^{\text{nfft}}$ elements.	All components must be in the range of $-1.0 \leq \text{data} < +1.0$ .	All components must be representable in the MATLAB software single datatype (equivalent to a float in C++).
scaling_sch	Scaling schedule. 1D array of integer values size S = number of stages. For Radix-4 and Streaming architectures, $S = \text{nfft}/2$ , rounded up to the next integer. For Radix-2 and Radix-2 Lite architectures, $S = \text{nfft}$ .	Each value corresponds to scaling to be performed by the corresponding stage and must be in the range 0 to 3. <code>scaling_sch[1]</code> is the scaling for the first stage.	N/A
direction	Transform direction. Single integer.	1 = forward FFT, 0 = inverse FFT (IFFT).	1 = forward FFT, 0 = inverse FFT (IFFT).

**Notes:**

1. nfft input is only used for run-time configurable transform length (that is, `generics.C_HAS_NFFT = 1`). It is ignored otherwise and `generics.C_NFFT_MAX` is used instead.
2. For fixed-point input FFTs (that is, `generics.C_USE_FLT_PT = 0`), to ensure identical numerical behavior to the hardware, pre-quantize the `input_data` values to have precision determined by `C_INPUT_WIDTH`. This is easily achieved using the MATLAB software built-in `quantize` function.
3. `scaling_sch` input is only used for a fixed-point input, scaled FFT (that is, `generics.C_USE_FLT_PT = 0`, `generics.C_HAS_SCALING = 1` and `generics.C_HAS_BFP = 0`). It is ignored otherwise.
4. `input_data` may be in natural or bit/digit-reversed sample index order. The MEX function produces samples in the same ordering format as they were input.

The function's outputs are shown in Table 3-6.

Table 3-6: FFT MEX Function Outputs

Output	Description	Validity
output_data	Output data. 1D array of complex data with $2^{n_{fft}}$ elements.	Always valid.
blk_exp	Block exponent. Single integer.	Only valid if using block floating point (if <code>generics.C_HAS_SCALING = 1</code> and <code>C_HAS_BFP = 1</code> ). Will be zero otherwise.
overflow	Overflow. Single integer. 1 indicates overflow occurred, 0 indicates no overflow occurred.	Only valid with a scaled FFT (if <code>generics.C_HAS_SCALING = 1</code> and <code>generics.C_HAS_BFP = 0</code> ) or an FFT with floating point interfaces (that is, <code>generics.C_USE_FLT_PT = 1</code> ). Will be zero otherwise.

### Notes:

#### General

1. There is no need to create and destroy state, as must be done with the C model; this is handled internally by the FFT MEX function.
2. The FFT MEX function performs extensive checking of its inputs. Any invalid input results in a message reporting the error and the function terminates.
3. The MEX function produces data in the same order as the input data. Hence, if `input_data` was provided in natural sample index order (0,1,2,3...), `output_data` samples will also be in natural sample index order.

#### FFTs with Fixed-Point Interface

1. Input data is an array of complex double-precision floating-point data, but the FFT core being modeled requires data in signed two's-complement, fixed-point format with precision given by `C_INPUT_WIDTH`. The data has a sign bit, then the binary point, then  $(C\_INPUT\_WIDTH - 1)$  fractional bits. The FFT MEX function checks the input data to see if it fits within this format and precision. If not, it prints a warning, and internally rounds it using convergent rounding (halfway values are rounded to the nearest even number) to the required precision. To accurately model the FFT core, pre-quantize the input data to this required precision before passing it to the model. This can be done easily using the MATLAB software built-in quantize function.

Type `help quantizer/quantize` or `doc quantize` on the MATLAB software command line for more information.

2. Output data has the correct precision to model the FFT:
  - a. If the FFT is scaled or has block floating point (that is, `C_HAS_SCALING = 1`, `C_HAS_BFP = 0` or `1`, respectively), output data is all in the range  $-1.0 \leq \text{data} < +1.0$ , the precision is `C_INPUT_WIDTH` bits, with  $C\_INPUT\_WIDTH - 1$  fractional bits. For example, if `C_INPUT_WIDTH = 8`, output data is precise to  $2^{-7} = 0.0078125$ , and is in the range  $-1.0$  to  $+0.9921875$ , and the binary representation of the output data has the format `s.fxxxxxx`, where `s` is the sign bit and `f` is a fractional bit.



- b. If the FFT is unscaled ( $C\_HAS\_SCALING = 0$ ), output data grows beyond  $\pm 1.0$ , such that the binary point remains in the same place and there are still  $(C\_INPUT\_WIDTH - 1)$  fractional bits after the binary point. In total, the output precision is  $(C\_INPUT\_WIDTH + C\_NFFT\_MAX + 1)$  bits. For example, if  $C\_INPUT\_WIDTH = 8$  and  $C\_NFFT\_MAX = 3$ , output data is precise to  $2^{-7} = 0.0078125$ , and is in the range  $-16.0$  to  $+15.9921875$ , and the binary representation of the output data has the format  $siiii.fffff$ , where  $s$  is the sign bit,  $i$  is an integer bit, and  $f$  is a fractional bit.
3. `blk_exp` is the integer block exponent. It is only valid (and non-zero) if block floating point is used (that is,  $C\_HAS\_SCALING = 1$  and  $C\_HAS\_BFP = 1$ ). It indicates the total number of bits that intermediate values were shifted right during the FFT processing. For example, if `blk_exp` = 5, the output data has been divided by 32 relative to the magnitude of the input data.
4. `overflow` indicates if overflow occurred during the FFT processing. It is only valid (and non-zero) if fixed scaling is used (that is,  $C\_HAS\_SCALING = 1$  and  $C\_HAS\_BFP = 0$ ). A value of 0 indicates that overflow did not occur; a value of 1 indicates that overflow occurred at some stage in the FFT processing. To avoid overflow, increase the scaling at one or more stages in the scaling schedule (`scaling_sch` input).
5. If overflow occurred with the Pipelined, Streaming I/O architecture ( $C\_ARCH = 3$ ), due to differences between the FFT core and the model in the order of operations within the processing stage, the output data may not match the `XK_RE` and `XK_IM` outputs of the FFT core. The output data must be ignored if the overflow output is 1. This is the only case where the model is not entirely bit accurate to the core.

#### FFTs with Floating-Point Interface

1. Input data is an array of complex double-precision floating-point data, but the FFT core being modeled requires values in single precision (32-bit) format. The data must therefore be representable in the MATLAB software 'single' datatype, even if it is represented in the 'double' datatype. The value will be explicitly cast to the C++ 'float' datatype inside the MEX function.
2. Output data has the correct precision to model the FFT. The double-precision output array will contain single-precision values which are representable in the MATLAB software 'single' datatype without error.
3. `overflow` indicates if exponent overflow has occurred during the FFT processing. A value of 0 indicates that overflow did not occur; a value of 1 indicates that exponent overflow did occur.
4. If overflow occurred, the output sample which overflowed will be set to  $\pm$  Infinity, depending on the sign of the internal result.

## MEX Function Example Code

An example M file, `run_xfft_v7_0_mex.m`, is provided. This demonstrates the steps required to run the MEX function: set up generics, create input data, simulate the FFT, and use the outputs. The code works for all legal combinations of generics. Simply modify the declarations of generics at the top of the file. The code also illustrates how to model a multichannel FFT; see ["Modeling Multichannel FFTs."](#)

The example code can be used to test your MEX function compilation process. See ["Building the MEX Function for Other MATLAB Software Versions."](#)

## Modeling Multichannel FFTs

The FFT v7.0 C model and FFT v7.0 MEX function are single-channel models that do not directly model multichannel FFTs. However, it is very simple to model multichannel FFTs.

By definition, a multichannel FFT is equivalent to multiple identical single channel FFTs, each operating on different input data. Therefore a multichannel FFT can be modeled simply by running a single channel model several times on each channel's input data.

For the FFT v7.0 C model, the example C++ code provided, `run_bitacc_cmodel.c`, demonstrates how to model a multichannel FFT. This example code creates the FFT state structure, then uses a for loop to run the model on each channel's input data in turn, then finally destroys the state structure.

For the FFT MEX function, simply call the function on each channel's input data in turn.