LogiCORE™ IP Fast Fourier Transform v5.0 Bit-Accurate C Model

User Guide

UG459 v1.0 October 17, 2007

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/17/07	1.0	Initial Xilinx release

Contents

About This Guide

This user guide provides information about the Xilinx LogiCORE™ IP Fast Fourier Transform v5.0 bit-accurate C model for 32-bit Linux, 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, a list of additional resources, and 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 v5.0 Bit-Accurate C Model" provides a description of the C model interface, example code, compiling with the C model, the MATLAB MEX function, and modelling multichannel FFTs.

Additional Resources

For additional information and resources, see www.xilinx.com/support. To go directly to a specific area of the support site, click a link in the table below.

Resource	Description/URL
Tutorials	Tutorials covering Xilinx design flows, from design entry to verification and debugging www.xilinx.com/support/techsup/tutorials/index.htm
Answer Browser	Database of Xilinx solution records www.xilinx.com/xlnx/xil_ans_browser.jsp
Application Notes	Descriptions of device-specific design techniques and approaches www.xilinx.com/xlnx/xweb/xil_publications_index.jsp?category=Application+Notes
Data Sheets	Device-specific information on Xilinx device characteristics, including readback, boundary scan, configuration, length count, and debugging www.xilinx.com/xlnx/xweb/xil_publications_index.jsp

Resource	Description/URL
Problem Solvers	Interactive tools that allow you to troubleshoot your design issues www.xilinx.com/support/troubleshoot/psolvers.htm
Tech Tips	Latest news, design tips, and patch information for the Xilinx design environment www.xilinx.com/xlnx/xil_tt_home.jsp

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`
`Courier bold`	Literal commands you enter in a syntactical statement	**`ngdbuild`** `design_name`
Italics	References to other manuals	See the *User Guide* for details.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* 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 **`bus[7:0]`**.	**`ngdbuild`** `[option_name] design_name`
Braces { }	A list of items from which you must choose one or more	**`lowpwr ={on|off}`**
Vertical bar |	Separates items in a list of choices	**`lowpwr ={on|off}`**

Vertical ellipsis
.
.
.
	Omitted repetitive material	`IOB #1: Name = QOUT'`
		`IOB #2: Name = CLKIN'`
		.
		.
		.

Horizontal ellipsis ...	Omitted repetitive material	**`allow block`** `block_name loc1 loc2 ... locn;`

Convention	Meaning or Use	Example
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 "Additional Resources" for more information. See "Title Formats" in Chapter 1 for detailed information.
Blue, underlined text	Hyperlink to a website (URL)	Go to www.xilinx.com for the latest speed files.

XILINX®

Chapter 1

Introduction

The Xilinx LogiCORE™ IP Fast Fourier Transform (FFT) v5.0 core has a bit-accurate C model designed for system modelling. A MATLAB MEX function for seamless MATLAB integration is also available.

Features

- Bit-accurate to FFT v5.0 core
- Dynamic link library
- Available for 32-bit Linux, 64-bit Linux and Windows platforms
- MATLAB MEX function (Windows only)
- Supports all features of the FFT core that affect numerical results
- Designed for rapid integration into a larger system model
- Example C++ code showing how to use the function is provided

Overview

The Xilinx LogiCORE IP FFT v5.0 has a bit-accurate C model for 32-bit Linux, 64-bit Linux, and 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 v5.0 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 MEX function for seamless MATLAB integration (Windows platform only).

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/xlnx/xebiz/designResources/ip_product_details.jsp?key=FFT.

Additional Core Resources

For detailed information and updates about the FFT v5.0 core, see the following documents, located on the FFT v5.0 product page at: www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=FFT

- Fast Fourier Transform v5.0 Product Specification (DS260)
- FFT v5.0 Release Notes

Technical Support

For technical support, go to www.xilinx.com/support. Questions are routed to a team with expertise using the FFT v5.0 core.

Xilinx provides technical support for use of this product as described in *LogiCORE IP Fast Fourier Transform v5.0 Bit-Accurate C Model User Guide (UG459)* and the *Fast Fourier Transform v5.0 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 v5.0 core and the accompanying documentation.

FFT v5.0 Bit-Accurate C Model Core

For comments or suggestions about the FFT v5.0 core, please submit a WebCase from 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 v5.0 core, please submit a WebCase from 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

XILINX®

Chapter 2

User Instructions

Unpacking and Model Contents

Unzip the FFT v5.0 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 v5.0 Bit-Accurate C Model**

File	Description
`README.txt`	release notes
`xfft_v5_0_bitacc_cmodel_ug459.pdf`	this file
`xfft_v5_0_bitacc_cmodel.h`	model header file
`run_bitacc_cmodel.c`	example code calling the model
`lin/`	model for 32-bit Linux
`libIp_xfft_v5_0_bitacc_cmodel.so`	model shared object library
`libSTL.so`	STL library, used by model
`lin64/`	model for 64-bit Linux
`libIp_xfft_v5_0_bitacc_cmodel.so`	model shared object library
`libSTL.so`	STL library, used by model
`nt/`	model for Windows
`libIp_xfft_v5_0_bitacc_cmodel.dll`	model dynamically linked library
`libIp_xfft_v5_0_bitacc_cmodel.libc`	model .lib file for compiling
`libSTL.dll`	STL library, used by model
`xfft_v5_0_bitacc_mex.mexw32`	MATLAB MEX interface to model

Installation

On Linux, ensure that the directory in which the files `libIp_xfft_v5_0_bitacc_cmodel.so` and `libSTL.so` are located is on your $LD_LIBRARY_PATH environment variable.

On Windows, ensure that the directory in which the files `libIp_xfft_v5_0_bitacc_cmodel.dll` and `libSTL.dll` are located is either on

your $PATH environment variable, or is the directory in which you will run your executable that calls the FFT v5.0 C model.

Chapter 3

FFT v5.0 Bit-Accurate C Model

FFT v5.0 C Model Interface

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

The C model is used through three functions, declared in the header file `xfft_v5_0_bitacc_cmodel.h`:

```
struct xilinx_ip_xfft_v5_0_state* xilinx_ip_xfft_v5_0_create_state
(
 struct xilinx_ip_xfft_v5_0_generics generics
);

int xilinx_ip_xfft_v5_0_bitacc_simulate
(
 struct xilinx_ip_xfft_v5_0_state* state,
 struct xilinx_ip_xfft_v5_0_inputs inputs,
 struct xilinx_ip_xfft_v5_0_outputs* outputs
);

void xilinx_ip_xfft_v5_0_destroy_state
(
 struct xilinx_ip_xfft_v5_0_state* state
);
```

To use the model, first create a state structure using the first function, xilinx_ip_xfft_v5_0_create_state. Then run the model using the second function, xilinx_ip_xfft_v5_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_v5_0_destroy_state. Each of these functions is described fully below.

Create a State Structure

The first function, xilinx_ip_xfft_v5_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 modelled. 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	log2(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_INPUT_WIDTH	Input data width (bits)	8-24
C_TWIDDLE_WIDTH	Phase factor width (bits)	8-24
C_HAS_SCALING	Scaling option: unscaled or determined by C_HAS_BFP	0 = unscaled, 1 = see C_HAS_BFP
C_HAS_BFP	Scaling option: if not unscaled, scaled or block floating point	0 = scaled, 1 = block floating point
C_HAS_ROUNDING	Rounding: Truncation or convergent rounding	0 = truncation, 1 = convergent rounding

Note: C_CHANNELS is not a generic used in the C model, as is incorrectly stated to be the case in the FFT v5.0 data sheet. This is an error in the data sheet. The model is always single channel. To model multiple channels in a multichannel FFT, see "Modelling Multichannel FFTs".

The xilinx_ip_xfft_v5_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_v5_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 inputs structure members are shown in Table 3-2.

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

Member	Type	Description
nfft	int	Transform length
xn_re	double*	Pointer to array of doubles: real part of input data
xn_re_size	int	Number of elements in xn_re array
xn_im	double*	Pointer to array of doubles: imaginary part of input data

Table 3-2: **Members of the Inputs Structure *(Continued)***

Member	Type	Description
xn_im_size	int	Number of elements in xn_im array
scaling_sch	int*	Pointer to array of ints containing scaling schedule
scaling_sch_size	int	Number of elements in scaling_sch array
direction	int	Transform direction: 1=forward FFT, 0=inverse FFT (IFFT)

Notes:

1. The user is responsible for allocating memory for arrays in the inputs structure.
2. nfft input is only used with run-time configurable transform length (C_HAS_NFFT = 1). If the transform length is fixed (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 this same value of 2^nfft.
4. Data in xn_re and xn_im must all be in the range -1.0 ≤ data < +1.0.
5. Data in xn_re and xn_im is of type double, but the FFT core being modelled 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 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, pre-quantize the input data to this required precision before passing it to the model.
6. scaling_sch and scaling_sch_size are ignored entirely unless fixed scaling is used (C_HAS_SCALING = 1 and C_HAS_BFP = 0).
7. scaling_sch is an array of integers, each of which indicates the scaling to be done in a stage of the FFT processing. 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:
   a. Radix-4, Burst I/O (C_ARCH = 1) or Pipelined, Streaming I/O (C_ARCH = 3): stages = ceil(nfft/2)
   b. Radix-2, Burst I/O (C_ARCH = 2) or Radix-2 Lite, Burst I/O (C_ARCH = 4): stages = nfft
8. 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. 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 will result in overflow, indicated by the overflow output.

The outputs structure, a pointer to which is passed to the xilinx_ip_xfft_v5_0_bitacc_simulate function, has the members shown in Table 3-3.

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

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

Notes:

1. The user is 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. 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 ≤ data < +1.0, the precision is C_INPUT_WIDTH bits, and the data has a sign bit, then the binary point, then (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.fffffff, 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 ± 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.fffffff, where s is the sign bit, i is an integer bit, and f is a fractional bit.
4. 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.
5. 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).
6. 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.

The xilinx_ip_xfft_v5_0_bitacc_simulate function checks the input and output structures for errors. If the model finds a problem, it will print an error message and return a value xilinx_ip_xfft_v5_0_bitacc_simulate function are shown in Table 3-4.

Table 3-4: **xilinx_ip_xfft_v5_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.
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.
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_v5_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_v5_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.

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 "Modelling Multichannel FFTs."

The example code can be used to test your compilation process. See "Compiling with the FFT v5.0 C Model."

Compiling with the FFT v5.0 C Model

Place the header file, `xfft_v5_0_bitacc_cmodel.h`, with your other header files.

Compilation varies from platform to platform.

Linux (Both 32-bit and 64-bit)

Reference the shared library files used by the model, `libSTL.so` and `libIp_xfft_v5_0_bitacc_cmodel.so`. For example, to compile the example code, `run_bitacc_cmodel.c`, first place the header file, C++ source file, and the two shared library files in a single directory. Then in that directory, compile using the GNU C++ Compiler v3.2.3:

```
g++ -x c++ run_bitacc_cmodel.c -o run_bitacc_cmodel -L. -lSTL -
lIp_xfft_v5_0_bitacc_cmodel
```

Windows

Link to the import library `libIp_xfft_v5_0_bitacc_cmodel.lib`. For example, for Microsoft Visual Studio.NET, in Project Properties, under Linker -> Input, for Additional Dependencies, specify `libIp_xfft_v5_0_bitacc_cmodel.lib`.

FFT v5.0 MATLAB MEX Function

Note: The FFT v5.0 MATLAB MEX function is available for the 32-bit Windows platform only.

The FFT v5.0 model is available as a compiled MATLAB MEX function for seamless integration with MATLAB running on Windows. The FFT MEX function provides a pre-compiled MATLAB 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.

The FFT MEX function is in the file `xfft_v5_0_bitacc_mex.mexw32`. It was built on MATLAB 7.3.0 (R2006b).

To install the FFT MEX function, place `xfft_v5_0_bitacc_mex.mexw32` in your MATLAB working directory, or in MATLAB, change directory to the directory containing `xfft_v5_0_bitacc_mex.mexw32`.

The FFT MEX function is called `xfft_v5_0_bitacc_mex`. Enter this function name without arguments at the MATLAB command line to see usage information.

The FFT MEX function syntax is:

```
[output_data, blk_exp, overflow] = xfft_v5_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, 8-field structure containing all relevant generics defining the core	
generics.C_NFFT_MAX	log2(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_INPUT_WIDTH	Input data width	8-24 bits
generics.C_TWIDDLE_WIDTH	Twiddle factor width	8-24 bits
generics.C_HAS_SCALING	Type of scaling	0 = unscaled, 1 = other
generics.C_HAS_BFP	Type of scaling if C_HAS_SCALING = 1	0 = scaled, 1 = block floating point
generics.C_HAS_ROUNDING	Type of rounding	0 = truncation, 1 = convergent rounding

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

Input	Description	Permitted values
nfft	log2(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.
input_data	Input data. 1D array of complex data with 2^nfft elements.	All components must be in the range -1.0 ≤ data < +1.0.
scaling_sch	Scaling schedule. 1D array of integer values size S = number of stages. For Radix-4 and Streaming architectures, S = nfft/2, rounded up to the next integer. For Radix-2 and Radix-2 Lite architectures, S = nfft.	Each value corresponds to scaling to be performed by the corresponding stage, and must be in the range 0 to 3.
direction	Transform direction. Single integer.	1 = forward FFT, 0 = inverse FFT (IFFT).

Notes:

1. nfft input is only used for run-time configurable transform length (generics.C_HAS_NFFT = 1). It is ignored otherwise and generics.C_NFFT_MAX is used instead.
2. 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 MATLAB'S built-in quantize function.
3. scaling_sch input is only used for a scaled FFT (generics.C_HAS_SCALING = 1 and generics.C_HAS_BFP = 0). It is ignored otherwise.

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^nfft elements.	Always valid.
blk_exp	Block exponent. Single integer.	Only valid if using block floating point (if generics.C_HAS_SCALING = 1 and C_HAS_BFP = 1). Will be zero otherwise.
overflow	Overflow. Single integer. 1 indicates overflow occurred, 0 indicates no overflow occurred.	Only valid with a scaled FFT (if C_HAS_SCALING = 1 and C_HAS_BFP = 0). Will be zero otherwise.

For example, the following MATLAB commands use the FFT MEX function to model a 1024-point forward FFT of random data with fixed scaling equivalent to 1/N:

```
generics.C_NFFT_MAX = 10
generics.C_ARCH = 1
generics.C_HAS_NFFT = 0
generics.C_INPUT_WIDTH = 16
generics.C_TWIDDLE_WIDTH = 16
generics.C_HAS_SCALING = 1
generics.C_HAS_BFP = 0
generics.C_HAS_ROUNDING = 0
input = rand(1024,1) * 2 - 1 + j*(rand(1024,1) * 2 - 1)
scaling_sch = [2 2 2 2 2]
[output, blkexp, overflow] = xfft_v5_0_bitacc_mex(generics, 10, input,
scaling_sch, 1)
```

Notes:

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. Input data is an array of complex floating-point data, but the FFT core being modelled 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 built-in quantize function.

   Type `help quantizer/quantize` or `doc quantize` on the MATLAB command line for more information.
4. Output data 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), output data is all in the range -1.0 ≤ data < +1.0, the precision is C_INPUT_WIDTH bits, and the data has a sign bit, then the binary point, then (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.fffffff, 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 ± 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.fffffff, where s is the sign bit, i is an integer bit, and f is a fractional bit.
5. 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.

6. 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).
7. 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.

Modelling Multichannel FFTs

The FFT v5.0 C model and FFT v5.0 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 modelled simply by running a single channel model several times on each channel's input data.

For the FFT v5.0 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.