LogiCORE IP 3GPP LTE Turbo Encoder v1.0 Bit-Accurate C Model

User Guide

UG490 (v1.0) April 25, 2008

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/25/08	1.0	Initial Xilinx release.

Table of Contents

Schedule of Tables

Chapter 1: Introduction

Chapter 2: User Instructions

Chapter 3: LTE TCC Bit-Accurate C Model

About This Guide

This user guide provides information about the Xilinx® LogiCORE™ IP 3GPP LTE Turbo Encoder v1.0 bit-accurate C model for 32-bit and 64-bit Linux platforms and 32-bit and 64-bit Windows platforms.

Guide Contents

This manual contains the following chapters:

- Chapter 1, "Introduction" contains an overview of the 3GPP LTE Turbo Encoder v1.0 bit-accurate C model.
- Chapter 2, "User Instructions" provides information on the C model contents and installation.
- Chapter 3, "LTE TCC Bit-Accurate C Model" contains information on using the interface and compiling with the C model.

Additional Resources

To find additional documentation, see the Xilinx web site at:

www.xilinx.com/literature.

To search the Answer Database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx website at:

www.xilinx.com/support.

Conventions

This document uses the following conventions. An example illustrates each convention.

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 that you enter in a syntactical statement	**`ngdbuild`** *design_name*

Convention	Meaning or Use	Example
Helvetica bold	Commands that you select from a menu	**File → Open**
	Keyboard shortcuts	**Ctrl+C**
Italic font	Variables in a syntax statement for which you must supply values	**ngdbuild** *design_name*
	References to other manuals	See the *Development System Reference Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **bus[7:0]**, they are required.	**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 . . .	Repetitive material that has been omitted	IOB #1: Name = QOUT' IOB #2: Name = CLKIN' . . .
Horizontal ellipsis ...	Repetitive material that has been omitted	**allow block** *block_name loc1 loc2 ... locn;*

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section "Additional Resources" for details. Refer to "Title Formats" in Chapter 1 for details.
Red text	Cross-reference link to a location in another document	See Figure 2-5 in the *Virtex-II Platform FPGA User Guide.*
Blue, underlined text	Hyperlink to a website (URL)	Go to http://www.xilinx.com for the latest speed files.

XILINX®

Chapter 1

Introduction

The Xilinx® LogiCORE™ IP 3GPP LTE Turbo Encoder v1.0 core has a bit accurate C model designed for system modeling. This allows the user to model the core performance. For purposes of abbreviation, 3GPP LTE Turbo Encoder v1.0 is used interchangeably with LTE-TCC throughout this document.

Features

- Bit-accurate to 3GPP LTE Turbo Encoder v1.0 core
- Available for 32-bit and 64-bit Linux platforms
- Available for 32-bit and 64-bit Windows platforms
- Designed for integration into a larger system model
- Example C++ code provided showing how to use the C model functions

Overview

This user guide provides information about the Xilinx LogiCORE™ IP 3GPP LTE Turbo Encoder v1.0 bit-accurate C model for 32-bit and 64-bit Linux, and 32-bit and 64-bit Windows platforms.

The model consists of a set of C functions that reside in a shared library. Example C code is provided to demonstrate how these functions form the interface to the C model. Full details of this interface are given in "LTE TCC Bit-Accurate C Model" in Chapter 3.

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

Additional Core Resources

For detailed information on the LTE-TCC core, see the following documents:

- *3GPP LTE Turbo Encoder v1.0 Product Specification* (DS674)
- 3GPP LTE Turbo Encoder v1.0 Release Notes

Technical Support

For technical support, go to www.xilinx.com/support.

Xilinx provides technical support for use of this product as described in *3GPP LTE Turbo Encoder v1.0 Bit-Accurate C Model User Guide* (UG489) and the *3GPP LTE Turbo Encoder v1.0 Product Specification* (DS674). 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 LTE-TCC core, C model, and the accompanying documentation.

Core and C Model

For comments or suggestions about the LTE-TCC core or C model, please submit a WebCase: www.xilinx.com/support/clearexpress/websupport.htm

Be sure to include the following information:

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

Documents

For comments or suggestions about the LTE-TCC documentation, please submit a WebCase: 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

Unzipping the LTE-TCC model ZIP file produces the directory structure and files shown in Table 2-1.

Table 2-1: **LTE-TCC Bit-Accurate C Model Directory Structure and Files**

File	Description
tcc_encoder_3gpplte_v1_0_bitacc_cmodel.h	Model header file
run_bitacc_cmodel.c	Example code calling the model
doc/	Documentation folder
README.txt	Release notes
tcc_encoder_3gpplte_bitacc_cmodel_ug490.pdf	This file
lin/	Model for 32-bit Linux(1)
libIp_tcc_encoder_3gpplte_v1_0_bitacc_cmodel.so	Model shared object library
libSTL.so	STL library, used by model
libstlport.so.5.1	Portability library, used by model
lin64/	Model for 64-bit Linux(1)
libIp_tcc_encoder_3gpplte_v1_0_bitacc_cmodel.so	Model shared object library
libSTL.so	STL library, used by model
libstlport.so.5.1	Portability library, used by model
nt/	Model for 32-bit Windows(2)
libIp_tcc_encoder_3gpplte_v1_0_bitacc_cmodel.dll	Model dynamically linked library

Table 2-1: **LTE-TCC Bit-Accurate C Model Directory Structure and Files *(Cont'd)***

File	Description
libIp_tcc_encoder_3gpplte_v1_0_bitacc_cmodel.lib	Model .lib file for compiling
stlport.5.1.dll	STL library, used by model
nt64/	Model for 64-bit Windows(3)
libIp_tcc_encoder_3gpplte_v1_0_bitacc_cmodel.dll	Model dynamically linked library
libIp_tcc_encoder_3gpplte_v1_0_bitacc_cmodel.lib	Model .lib file for compiling
stlport.5.1.dll	STL library, used by model

Notes:

1. Files is these folders are compiled with GCC v4.1.1.
2. Files in this folder are compiled with Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 13.10.3077 for 80x86.
3. Files in this folder are compiled with Microsoft (R) C/C++ Optimizing Compiler Version 14.00.40607.85 for AMD64.

Installation

On Linux, ensure that the directory in which the files libIp_tcc_encoder_3gpplte_v1_0_bitacc_cmodel.so, libstlport.so.5.1, and libSTL.so are located is on your $LD_LIBRARY_PATH environment variable, or is the directory in which you will run your executable that calls the LTE-TCC C model.

On Windows, ensure that the directory in which the files libIp_tcc_encoder_3gpplte_v1_0_bitacc_cmodel.dll and stlport.5.1.dll are located is either on your $PATH environment variable, or is the directory in which you will run your executable that calls the LTE-TCC C model.

XILINX®

Chapter 3

LTE TCC Bit-Accurate C Model

LTE-TCC C Model Interface

The Application Programming Interface (API) of the C model is defined in the header file `tcc_encoder_3gpplte_v1_0_bitacc_cmodel.h`. The interface consists of four functions, and four structures supporting those functions.

Structures

The interface consists of the following structures.

Generics

The `xilinx_ip_tcc_encoder_3gpplte_v1_0_generics` structure specifies the generics that should apply to the modeled core. The structure contains fields for the core generics however currently none of these generics will affect bit accuracy of the model. The generics are detailed in Table 3-1.

Table 3-1: **Generics Structure**

Member	Type	Description
C_FAMILY	char*	Device family.
C_ELABORATION_DIR	char*	Core elaboration directory.
C_HAS_RFD_IN	int	Output flow control port present.
C_HAS_CE	int	Clock enable port present.
C_HAS_ND	int	Input flow control port present.
C_HAS_SCLR	int	Synchronous clear port present.
C_HAS_ACLR	int	Asynchronous clear port present.

State

The `xilinx_ip_tcc_encoder_3gpplte_v1_0_state` structure defines the internal state of the C model. Because the structure is solely for internal use by the C model, the layout of the structure is not defined. User modification of the state structure may lead to undefined behavior.

Input

The `xilinx_ip_tcc_encoder_3gpplte_v1_0_inputs` structure is used to specify input data for the C model. See Table 3-2.

Table 3-2: **Input Structure**

Member	Type	Description
`din`	unsigned char*	Pointer to an array of bytes which hold the data input bits to be encoded.
`din_size`	int	Size of data to be encoded, i.e., block size.

The `din` array is used to pass the bit inputs into the C model. The `din_size` field defines the size of the `din` array and should be equal to the code block size.

Allocation of the arrays is the responsibility of the user. They may be allocated statically or dynamically. If allocated dynamically, then the user remains responsible for de-allocation. Note that the arrays maybe larger than that specified by `din_size` allowing the arrays to be pre-allocated for the largest code block (6144 bits). The C model only uses the first `din_size` elements of each array.

Output

The `xilinx_ip_tcc_encoder_3gpplte_v1_0_outputs` structure is used to specify output data from the C model. See Table 3-3.

Table 3-3: **Output Structure**

Member	Type	Description
`rsc1_sys`	unsigned char*	Pointer to an array of bytes that holds the non-interleaved systematic data.
`rsc1_par`	unsigned char*	Pointer to an array of bytes that holds the encoded non-interleaved parity data.
`rsc2_par`	unsigned char*	Pointer to an array of bytes that holds the encoded interleaved parity data.
`max_dout_size`	int	Allocated size of the arrays `rsc1_sys`, `rsc1_par`, and `rsc2_par`.
`dout_size`	int	Number of bits in the arrays `rsc1_sys`, `rsc1_par`, and `rsc2_par` returned by the C model.

The `rsc1_sys, rsc1_par and rsc2_par` arrays are used to receive encoded data from the C model. After a successful encode, each element of each array holds a single encoded data bit. The arrays also contain tail bits. The 12 tail bits are distributed across the three arrays, four extra bits per array, as described in *3GPP LTE Turbo Encoder v1.0 Product Specification* (DS674).

Allocation of the arrays is the responsibility of the user. They may be allocated statically or dynamically. If allocated dynamically, then the user remains responsible for de-allocation. The `max_dout_size` field is used to specify the size of the allocated arrays. When encoding a code block, the C model checks that the output arrays have sufficient space for the encoded data, before updating the `dout_size` field with the actual code block size plus 4 bits to account for the tail bits.

Functions

The interface consists of the following functions.

Default Generics

The `xilinx_ip_tcc_encoder_3gpplte_v1_0_get_default_generics` function is used to create a default `xilinx_ip_tcc_encoder_3gpplte_v1_0_generics` structure:

```
struct xilinx_ip_tcc_encoder_3gpplte_v1_0_generics
  xilinx_ip_tcc_encoder_3gpplte_v1_0_get_default_generics();
```

The function returns a default generic structure which can be used directly as further customization will have no effect on the operation of the model.

Create State

The `xilinx_ip_tcc_encoder_3gpplte_v1_0_create_state` function creates a new state structure based on the given generics:

```
struct xilinx_ip_tcc_encoder_3gpplte_v1_0_state*
  xilinx_ip_tcc_encoder_3gpplte_v1_0_create_state
(
  struct xilinx_ip_tcc_encoder_3gpplte_v1_0_generics generics
);
```

The function returns a pointer to the created state structure. If the state structure cannot be created, then an error message is produced on standard error and the function returns a null pointer.

Simulate

The `xilinx_ip_tcc_encoder_3gpplte_v1_0_bitacc_simulate` function encodes a single code block:

```
int xilinx_ip_tcc_encoder_3gpplte_v1_0_bitacc_simulate
(
  struct xilinx_ip_tcc_encoder_3gpplte_v1_0_state*   state,
  struct xilinx_ip_tcc_encoder_3gpplte_v1_0_inputs   inputs,
  struct xilinx_ip_tcc_encoder_3gpplte_v1_0_outputs* outputs
);
```

On entry, the user must initialize all fields of the `inputs` structure. The `din` array should be filled with input data. Additionally, the `rsc1_sys`, `rsc1_par`, `rsc2_par` and `max_dout_size` fields of the `outputs` structure must be initialized to indicate to the C model the location and size of the output arrays. The `dout_size` field of the `outputs` structure is set by the function on exit.

The function returns zero if the code block was successfully encoded. If the code block could not be encoded, an error message is produced on standard error and the function returns a non-zero error code.

Destroy State

The `xilinx_ip_tcc_encoder_3gpplte_v1_0_destroy_state` function destroys a state structure:

```
void xilinx_ip_tcc_encoder_3gpplte_v1_0_destroy_state
(
  struct xilinx_ip_tcc_encoder_3gpplte_v1_0_state* state
);
```

On return, any memory resources allocated within the state structure are released and the state structure becomes undefined.

Compiling

Compilation of user code requires access to the `tcc_encoder_3gpplte_v1_0_bitacc_cmodel.h` header file. The header file should be copied to a location where it is available to the compiler. Depending on the location chosen, the include search path of the compiler may need to be modified.

Linking

To use the C model the user executable must be linked against the correct libraries for the target platform.

Linux

The executable must be linked against the `libIp_tcc_encoder_3gpplte_v1_0_bitacc_cmodel.so`, `libSTL.so`, and `libstlport.so.5.1` shared object libraries. Files for 32-bit and 64-bit systems are supplied in the `lin` and `lin64` directories, respectively.

Using GCC, linking is typically achieved by adding the following command line options:

```
-L.-lIp_tcc_encoder_3gpplte_v1_0_bitacc_cmodel
```

Note that this assumes the three shared object libraries are in the current directory. If this is not the case, the –L. option should be changed to specify the library search path to use.

Windows

The executable must be linked against the `libIp_tcc_encoder_3gpplte_v1_0_bitacc_cmodel.dll`, `libSTL.dll`, and `libstlport.5.1.dll` dynamic link libraries. Depending on the compiler, the import library `libIp_tcc_encoder_3gpplte_v1_0_bitacc_cmodel.lib` may be required. Files for 32-bit and 64-bit systems are supplied in the `nt` and `nt64` directories, respectively.

Using Microsoft Visual Studio.NET, linking is typically achieved by adding the import library to the Additional Dependencies edit box under the Linker tab of Project Properties.

Example

The `run_bitacc_cmodel.c` file contains example code to show basic operation of the C model.