Turbo Encoder & Decoder
MegaCore Function User Guide
SUPPLY OF THIS IP CORE DOES NOT CONVEY A LICENSE NOR IMPLY ANY RIGHT TO USE ANY TURBO CODES PATENTS OWNED BY FRANCE TELECOM, TDF OR GET. PLEASE CONTACT FRANCE TELECOM FOR INFORMATION ABOUT TURBO CODES LICENSING PROGRAM AT THE FOLLOWING ADDRESS:
France Telecom R&D- VAT/TURBOCODES, 38 Rue du General Leclerc 92794 Issy Moulineaux Cedex 9

Copyright © 2004 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera’s standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
Contents

About this User Guide .................................................................................................................. v
  Revision History .................................................................................................................. v
  How to Contact Altera ...................................................................................................... v
  Typographic Conventions .................................................................................................. vi

Chapter 1. About this MegaCore Function
  Release Information ......................................................................................................... 1–1
  Device Family Support ..................................................................................................... 1–1
  Introduction ....................................................................................................................... 1–2
  New in Version 1.6.0 ......................................................................................................... 1–2
  Features .............................................................................................................................. 1–2
  General Description ......................................................................................................... 1–3
    Interleavers ..................................................................................................................... 1–4
    Turbo Encoder ............................................................................................................... 1–4
    Turbo Decoder ............................................................................................................... 1–4
    OpenCore Plus Evaluation ............................................................................................. 1–5
  Performance ...................................................................................................................... 1–5

Chapter 2. Getting Started
  System Requirements ....................................................................................................... 2–1
  Design Flow ....................................................................................................................... 2–1
  Obtain & Install the Turbo Encoder & Decoder MegaCore Function .................................. 2–2
    Download the Turbo Encoder & Decoder MegaCore Function ....................................... 2–2
    Installing the Turbo Encoder & Decoder Files ................................................................ 2–3
    Directory Structure ....................................................................................................... 2–4
  Turbo Encoder & Decoder Walkthrough ......................................................................... 2–4
    Evaluate the BER using the C-Model ............................................................................ 2–5
    Create a New Quartus II Project ..................................................................................... 2–7
    Launch IP Toolbench ...................................................................................................... 2–8
    Step 1: Parameterize ....................................................................................................... 2–9
    Step 2: Set Up Simulation .............................................................................................. 2–12
    Step 3: Generate ........................................................................................................... 2–14
  Simulate your Design ....................................................................................................... 2–16
    IP Functional Simulation Model .................................................................................... 2–17
    Use the ModelSim VHDL Models .................................................................................. 2–18
  Compile the Design .......................................................................................................... 2–20
  Program a Device ............................................................................................................. 2–20
  Set Up Licensing .............................................................................................................. 2–20
  Append the License to Your license.dat File .................................................................... 2–21
Specify the License File in the Quartus II Software ................................................................. 2–21

Chapter 3. Specifications—Encoder
Functional Description .................................................................................................................. 3–1
  OpenCore Plus Time-Out Behavior ......................................................................................... 3–2
Parameters .......................................................................................................................... 3–2
Signals .................................................................................................................................... 3–2
Encoder Operation .............................................................................................................. 3–3
MegaCore Verification ........................................................................................................... 3–6
Performance ....................................................................................................................... 3–7

Chapter 4. Specifications—Decoder
Functional Description .............................................................................................................. 4–1
  Data Format ......................................................................................................................... 4–1
  Memory Requirements ........................................................................................................ 4–2
  OpenCore Plus Time-Out Behavior ....................................................................................... 4–3
Parameters .......................................................................................................................... 4–4
Signals ................................................................................................................................... 4–4
Decoder Operation ................................................................................................................ 4–5
  External Puncturing and Depuncturing ............................................................................... 4–8
MegaCore Verification ........................................................................................................... 4–9
Performance ....................................................................................................................... 4–9
Revision History

The table below displays the revision history for the chapters in this user guide.

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Date</th>
<th>Version</th>
<th>Changes Made</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>June 2004</td>
<td>1.6.0</td>
<td>● Updated device family support table.</td>
</tr>
<tr>
<td></td>
<td>February 2004</td>
<td>1.5.0</td>
<td>● Updated device family support table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>● Added OpenCore® Plus description.</td>
</tr>
<tr>
<td>2</td>
<td>June 2004</td>
<td>1.6.0</td>
<td>● Updated system requirements.</td>
</tr>
<tr>
<td></td>
<td>February 2004</td>
<td>1.5.0</td>
<td>● Updated Quartus® II software information.</td>
</tr>
<tr>
<td>3</td>
<td>February 2004</td>
<td>1.5.0</td>
<td>● Added OpenCore Plus information.</td>
</tr>
<tr>
<td>4</td>
<td>February 2004</td>
<td>1.5.0</td>
<td>● Added OpenCore Plus information.</td>
</tr>
</tbody>
</table>

How to Contact Altera

For technical support or other information about Altera products, go to the Altera world-wide web site at www.altera.com. You can also contact Altera through your local sales representative or any of the sources listed below.

<table>
<thead>
<tr>
<th>Information Type</th>
<th>USA &amp; Canada</th>
<th>All Other Locations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>800-800-EPLD (3753)</td>
<td>+1 408-544-8767</td>
</tr>
<tr>
<td></td>
<td>7:00 a.m. to 5:00 p.m. Pacific Time</td>
<td>7:00 a.m. to 5:00 p.m. (GMT -8:00)</td>
</tr>
<tr>
<td></td>
<td>Pacific Time</td>
<td>Pacific Time</td>
</tr>
<tr>
<td>Product literature</td>
<td><a href="http://www.altera.com">www.altera.com</a></td>
<td><a href="http://www.altera.com">www.altera.com</a></td>
</tr>
<tr>
<td>Altera literature services</td>
<td><a href="mailto:lit_req@altera.com">lit_req@altera.com</a></td>
<td><a href="mailto:lit_req@altera.com">lit_req@altera.com</a></td>
</tr>
<tr>
<td>Non-technical customer service</td>
<td>800-767-3753</td>
<td>+ 1 408-544-7000</td>
</tr>
<tr>
<td></td>
<td>7:00 a.m. to 5:00 p.m. (GMT -8:00)</td>
<td>7:00 a.m. to 5:00 p.m. (GMT -8:00)</td>
</tr>
<tr>
<td></td>
<td>Pacific Time</td>
<td>Pacific Time</td>
</tr>
<tr>
<td>FTP site</td>
<td>ftp.altera.com</td>
<td>ftp.altera.com</td>
</tr>
</tbody>
</table>
This document uses the typographic conventions shown below.

<table>
<thead>
<tr>
<th>Visual Cue</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold Type with Initial Capital Letters</strong></td>
<td>Command names, dialog box titles, checkbox options, and dialog box option names are shown in bold, initial capital letters. Example: <strong>Save As</strong> dialog box.</td>
</tr>
<tr>
<td><strong>bold type</strong></td>
<td>External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: f&lt;sub&gt;MAX&lt;/sub&gt;, \qdesigns directory, d: drive, chiptrip.gdf file.</td>
</tr>
<tr>
<td><strong>Italic Type with Initial Capital Letters</strong></td>
<td>Document titles are shown in italic type with initial capital letters. Example: <strong>AN 75: High-Speed Board Design</strong>.</td>
</tr>
<tr>
<td><strong>Italic type</strong></td>
<td>Internal timing parameters and variables are shown in italic type. Examples: t&lt;sub&gt;PIA&lt;/sub&gt;, n + 1.</td>
</tr>
<tr>
<td>Initial Capital Letters</td>
<td>Keyboard keys and menu names are shown in initial capital letters. Examples: Delete key, the Options menu.</td>
</tr>
<tr>
<td>“Subheading Title”</td>
<td>References to sections within a document and titles of on-line help topics are shown in quotation marks. Example: “Typographic Conventions.”</td>
</tr>
<tr>
<td>Courier type</td>
<td>Signal and port names are shown in lowercase Courier type. Examples: datal, tdi, input. <strong>Active-low signals are denoted by suffix n, e.g., resetn.</strong></td>
</tr>
<tr>
<td>Anything that must be typed exactly as it appears is shown in Courier type. For example: c:\qdesigns\tutorial\chiptrip.gdf. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword <strong>SUBDESIGN</strong>), as well as logic function names (e.g., <strong>TRI</strong>) are shown in Courier.</td>
<td></td>
</tr>
<tr>
<td>1., 2., 3., and a., b., c., etc.</td>
<td>Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.</td>
</tr>
<tr>
<td>• -</td>
<td>Bullets are used in a list of items when the sequence of the items is not important.</td>
</tr>
<tr>
<td>✔</td>
<td>The checkmark indicates a procedure that consists of one step only.</td>
</tr>
<tr>
<td>▴</td>
<td>The hand points to information that requires special attention.</td>
</tr>
<tr>
<td>!</td>
<td>The caution indicates required information that needs special consideration and understanding and should be read prior to starting or continuing with the procedure or process.</td>
</tr>
<tr>
<td>!</td>
<td>The warning indicates information that should be read prior to starting or continuing the procedure or processes.</td>
</tr>
<tr>
<td>←</td>
<td>The angled arrow indicates you should press the Enter key.</td>
</tr>
<tr>
<td>←</td>
<td>The feet direct you to more information on a particular topic.</td>
</tr>
</tbody>
</table>
Chapter 1. About this MegaCore Function

Release Information

Table 1–1 provides information about this release of the Turbo Encoder and Decoder MegaCore® function.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1.6.0</td>
</tr>
<tr>
<td>Release Date</td>
<td>June 2004</td>
</tr>
<tr>
<td>Ordering Codes</td>
<td>IP-TURBO/ENC (Encoder)</td>
</tr>
<tr>
<td></td>
<td>IP-TURBO/DEC (Decoder)</td>
</tr>
<tr>
<td>Product IDs</td>
<td>0019 (Encoder)</td>
</tr>
<tr>
<td></td>
<td>0020 (Decoder)</td>
</tr>
<tr>
<td>Vendor ID</td>
<td>6AF7</td>
</tr>
</tbody>
</table>

Device Family Support

MegaCore® functions provide either full or preliminary support for target Altera® device families, as described below:

- **Full support** means the MegaCore function meets all functional and timing requirements for the device family and may be used in production designs.
- **Preliminary support** means the MegaCore function meets all functional requirements, but may still be undergoing timing analysis for the device family; it may be used in production designs with caution.

Table 1–2 shows the level of support offered by the Turbo Encoder and Decoder MegaCore function to each of the Altera device families.

<table>
<thead>
<tr>
<th>Device Family</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stratix® II</td>
<td>Full</td>
</tr>
<tr>
<td>Stratix GX</td>
<td>Full</td>
</tr>
<tr>
<td>Stratix</td>
<td>Full</td>
</tr>
<tr>
<td>Cyclone™ II</td>
<td>Full</td>
</tr>
<tr>
<td>Cyclone</td>
<td>Full</td>
</tr>
</tbody>
</table>
**Introduction**

The Altera Turbo Encoder MegaCore function dramatically shortens design cycles. The Turbo Decoder MegaCore function is a high-performance logarithmic “maximum a posteriori” (max-logMAP) decoder for maximum error correction.

**New in Version 1.6.0**

- Support for Cyclone II devices

**Features**

- Compliant with the 3rd Generation Partnership Project (3GPP); Technical Specification Group Radio Access Network; Multiplexing and Channel Coding (FDD) (3G TS 25.212 version 3.2.0)
- Encoder supports selectable double buffering of data
- Encoder supports parallel outputs
- Encoder supports high-speed downlink packet access (HSDPA) data rates
- High-performance max-logMAP decoder for maximum error correction
- Decoder data rates in excess of 2 megabits per second (Mbps)
- 3GPP-compliant “mother” interleaver
- Decoder interleaver block sizes from 40 to 5,114 bits—block size can change between each block
- Soft values (logarithmic likelihood) from 3 to 8 bits
- Optional two memory banks for maximum decoder throughput
- IP functional simulation models for use in Altera-supported VHDL and Verilog HDL simulators
- Support for OpenCore® Plus evaluation

---

**Table 1–2. Device Family Support (Part 2 of 2)**

<table>
<thead>
<tr>
<th>Device Family</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mercury™</td>
<td>Full</td>
</tr>
<tr>
<td>Excalibur™</td>
<td>Full</td>
</tr>
<tr>
<td>HardCopy®</td>
<td>Full</td>
</tr>
<tr>
<td>APEX™ II</td>
<td>Full</td>
</tr>
<tr>
<td>APEX 20KE &amp; APEX 20KC</td>
<td>Full</td>
</tr>
<tr>
<td>APEX 20K</td>
<td>Full</td>
</tr>
<tr>
<td>Other device families</td>
<td>No support</td>
</tr>
</tbody>
</table>
General Description
A significant amount of research has been carried out to make efficient use of available bandwidth. This research has led to the development of sophisticated coding schemes. The last major step was the introduction of the Viterbi and Reed-Solomon decoders. Since then, a new method has emerged. A combination of iteratively run “soft in/soft out” decoders with simple component codes and an interleaver have allowed the gap to the theoretical limit (Shannon limit) to be narrowed even further. This process is referred to as “turbo coding” or “iterative decoding”.

Figure 1–1 shows the basic block diagram of a turbo encoder and decoder.

**Figure 1–1. Turbo Encoder & Decoder Block Diagram**

**Note to Figure 1–1:**
(1) Although the illustration shows two max-logMAP decoders, they are physically implemented as one max-logMAP decoder.
General Description

Interleavers

Interleaving is the process of reordering a binary sequence in a systematic way. Convolutional codes are designed to combat random independent errors. For channels with memory, such as fading channels, this method is not optimal. Errors typically come in bursts rather than being randomly distributed. Interleaving can be used to disperse the burst errors, making them easier to correct.

The turbo encoder interleaver, as defined by 3GPP, is a 3-stage interleaver with a block size between 40 and 5,114 bits. The input sequence is first written row by row into a matrix. The rows are then algebraically interleaved, based on sets of prime integers. Each row is then interleaved with a predefined pattern. The output sequence is generated by reading out the matrix, column by column. The output sequence is pruned in the cases where the input sequence does not exactly fill the matrix.

Turbo Encoder

The turbo encoder feeds the incoming information bits through to the output. In addition, it encodes them using encoder 1, a recursive convolutional encoder. It also feeds the information bits via an interleaver into encoder 2. If required, the encoded bit streams can be punctured by external logic. The datapath through the encoder may be double buffered, to allow a new data block to be shifted in whilst encoding and shifting out the previous block.

Turbo Decoder

After depuncturing the received data stream, the information bits and parity 1 bits are fed into decoder 1. The equalizer (not shown in Figure 1–1) delivers soft information on the received data stream, i.e., it delivers probabilities of the received values. These probabilities can be interpreted as containing the received bit value and the confidence value, which indicates how likely it is that this bit is correct. Decoder 1 then evaluates these probabilities and combines them with the parity 1 probabilities. This refines the soft information so the confidence of individual bit correctness is maximized. The refined probabilities are fed into decoder 2 with the information bits and the parity 2 bits, again producing enhanced soft information. After a predefined number of iterations (typically three to six), the decoding process is completed, and the soft decision values are available at the output.

When data is exchanged between the two decoders, the soft values are reordered to match the interleaving structure. This reordering is done with the interleaver and deinterleaver between the decoders. When the second decoder has finished, the next iteration is started. Again,
About this MegaCore Function

decoder 1 is activated using the soft information from the previous decoding as well as the information bits and parity 1 bits, and the second decoder is activated.

The decoding makes use of the “maximum a posteriori” (MAP) algorithm; an extremely computationally intensive algorithm. There has been much work in both improving the efficiency and researching the use of alternatives. The only viable alternative is the soft output Viterbi algorithm (SOVA), which is similar to the algorithms used in some channel equalizers. However, SOVA is sub-optimum and is currently not supported by the Altera Turbo Encoder and Decoder MegaCore function.

OpenCore Plus Evaluation

With Altera’s free OpenCore Plus evaluation feature, you can perform the following actions:

■ Simulate the behavior of a MegaCore function within your system
■ Verify the functionality of your design, as well as evaluate its size and speed quickly and easily
■ Generate time-limited device programming files for designs that include MegaCore function
■ Program a device and verify your design in hardware

You only need to purchase a license for the MegaCore function when you are completely satisfied with its functionality and performance, and want to take your design to production.

For more information on OpenCore Plus hardware evaluation using turbo, see “OpenCore Plus Time-Out Behavior” on page 3–2, “OpenCore Plus Time-Out Behavior” on page 4–3 and AN 320: OpenCore Plus Evaluation of Megafunctions.

Performance

For the encoder performance, see “Performance” on page 3–7; for the decoder performance, see “Performance” on page 4–9.
Chapter 2. Getting Started

System Requirements

The instructions in this section require the following hardware and software:

■ A PC running the Windows NT/2000/XP, Red Hat Linux 7.3 or 8.0, or Red Hat Enterprise Linux 3.0 operating system; or a Sun workstation running the Solaris 7 or 8 operating system
■ Quartus® II software version 4.1 or higher

Design Flow

To evaluate the Turbo Encoder and Decoder MegaCore® function using the OpenCore® Plus feature, the design flow involves the following steps:

1. Obtain and install the Turbo Encoder and Decoder MegaCore function.

2. Evaluate the bit-error rate (BER) using the C model.

3. Create a custom variation of the Turbo Encoder and Decoder MegaCore function using IP Toolbench.

IP Toolbench is a toolbar from which you can quickly and easily view documentation, specify parameters, and generate all of the files necessary for integrating the parameterized MegaCore function into your design. You can launch IP Toolbench from within the Quartus II software.

4. Implement the rest of your design using the design entry method of your choice.

5. Use the IP Toolbench-generated IP functional simulation model or the precompiled ModelSim VHDL model to verify the operation of your design.

For more information on IP functional simulation models, refer to the white paper Using IP Functional Simulation Models to Verify Your System Design.

6. Use the Quartus II software to compile your design.
Obtain & Install the Turbo Encoder & Decoder MegaCore Function

You may also generate an OpenCore Plus time-limited programming file, which you can use to verify the operation of your design in hardware.

7. Purchase a license for the MegaCore function.

Once you have purchased a license for the Turbo Encoder and Decoder MegaCore function, the design flow involves the following additional steps:

1. Set up licensing.

2. Generate a programming file for the Altera® device(s) on your board.

3. Program the Altera device(s) with the completed design.

4. Perform design verification.

Obtain & Install the Turbo Encoder & Decoder MegaCore Function

Before you can start using Altera MegaCore functions, you must obtain the MegaCore files and install them on your computer. Altera MegaCore functions can be installed from the MegaCore IP Library CD-ROM either during or after Quartus II installation, or downloaded individually from the Altera web site and installed separately.

The following instructions describe the process of downloading and installing the Turbo Encoder and Decoder MegaCore function. If you have already installed the Turbo Encoder and Decoder MegaCore function from the MegaCore IP Library CD-ROM, skip to “Directory Structure” on page 2–4.

Download the Turbo Encoder & Decoder MegaCore Function

If you have Internet access, you can download MegaCore functions from Altera’s web site at www.altera.com. Follow the instructions below to obtain the Turbo Encoder and Decoder MegaCore function via the Internet. If you do not have Internet access, contact your local Altera representative to obtain the MegaCore IP Library CD-ROM.

1. Point your web browser to www.altera.com/ipmegastore.

2. Type Turbo in the IP MegaSearch box.

3. Click Go.
Getting Started

4. Choose **Turbo Encoder Function** or **Turbo Decoder Function** from the search results page. The product description web page displays.

5. Click **Download Free Evaluation** on the top right of the product description web page.

6. Fill out the registration form and click **Submit Request**.

7. Read the Altera MegaCore license agreement, turn on the **I have read the license agreement** check box, and click **Proceed to Download Page**.

8. Follow the instructions on the Turbo Encoder and Decoder MegaCore function download and installation page to download the function and save it to your hard disk.

**Installing the Turbo Encoder & Decoder Files**

The following instructions describe how you install the Turbo Encoder and Decoder MegaCore function on computers running the Windows, Linux, or Solaris operating systems.

**Windows**

To install the Turbo Encoder or Decoder MegaCore function on a PC running the Windows operating system, follow these steps:

1. Choose **Run** (Start menu).

2. Type `<path name>\turbo_codec-v1.6.0.exe`, where `<path name>` is the location of the downloaded MegaCore function.

3. Click **OK**. The **Turbo Installation** dialog box appears. Follow the on-screen instructions to finish installation.

**Solaris & Linux**

To install the Turbo Encoder and Decoder MegaCore function on a computer running the Solaris or Linux operating systems, follow these steps:

1. Decompress the package by typing the following command:

   ```
gzip -d turbo_codec-v1.6.0.tar.gz
   ```

2. Extract the package by typing the following command:
turbo_codec-v1.6.0.tar

Directory Structure

Figure 2–1 shows the directory structure for the Turbo Encoder and Decoder MegaCore function.

Figure 2–1. Turbo Encoder & Decoder Directory Structure

This walkthrough explains how to create a Turbo Encoder or Decoder MegaCore function using the Altera turbo encoder and decoder IP Toolbench and the Quartus II software. As you go through the wizard, each step is described in detail. When you are finished generating a turbo encoder or decoder, you can incorporate it into your overall project.

This walkthrough involves the following steps:

- “Evaluate the BER using the C-Model” on page 2–5
- “Create a New Quartus II Project” on page 2–7
- “Launch IP Toolbench” on page 2–8
- “Step 1: Parameterize” on page 2–9
“Step 2: Set Up Simulation” on page 2–12
“Step 3: Generate” on page 2–14

Evaluate the BER using the C-Model

The C-model is a program that allows you to enter your choice of parameters and view the number of errors, the BER and the frame error rate. To use the C-model, perform the following steps:

1. Open the Command Prompt. Change the directory to <pathname>\turbo_code\c_model, where <pathname> is the location of the function.
2. Type BER_simulator for the C-model with no-puncturing; BER_simulator_punct for the C-model with puncturing, and type in the parameters and required values as shown:

   -c <channel>  -n <signal-to-noise ratio>  -i <number of iterations>  
   -f <number of frames>  -l <block length>  -w <number of softbits>r

   For example:

   ber_simulator -c 0 -n 1 -i 5 -f 100 -l 40 -w 5r

   Table 2–1 shows the C-model parameters and valid values.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Valid Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel</td>
<td>0 or 1 (1)</td>
</tr>
<tr>
<td>Signal to noise ratio</td>
<td>0 to 20</td>
</tr>
<tr>
<td>Number of iterations</td>
<td>1 to 16</td>
</tr>
<tr>
<td>Number of soft bits</td>
<td>1 to 16</td>
</tr>
<tr>
<td>Block size</td>
<td>40 to 5,114</td>
</tr>
<tr>
<td>Number of frames</td>
<td>Any positive integer &gt; 20</td>
</tr>
</tbody>
</table>

   Note to Table 2–1:
   (1) Channel value 0 selects “additive white Gaussian noise”; 1 selects Rayleigh Fading.

3. If the parameters are entered incorrectly, the incorrect syntax error message appears and the C-model will illustrate the correct syntax.
4. The C-model outputs are in the format shown:
BER Graphs

Alterna provides a database of combinations of parameters with the BERs that they produce. You can access this database using MATLAB. The following steps explain how to show the turbo decoder BER graphs in MATLAB.

For more information on MATLAB, refer to the Math Works website at www.mathworks.com.

1. Open the MATLAB software. At the command prompt change to the \turbo_codec\c_model directory.
2. At the command prompt type BER_graphs. Select your chosen axis variables in the menus, change the values with the slider bars, and click Update to view the BER graph. Figure 2–2 shows an example.

Figure 2–2. BER Graph
The results for this model are not accurate for low BERs, as only one million samples are used for each parameter combination. Also the BER has a floor of $10^{-8}$; preventing the graphs going on to $-\infty$. A BER of $10^{-8}$ can be interpreted as the decoder decoding all bits correctly.

Create a New Quartus II Project

Before you begin, you must create a new Quartus II project. With the New Project wizard, you specify the working directory for the project, assign the project name, and designate the name of the top-level design entity. You will also specify the Turbo Encoder and Decoder MegaCore function user library. To create a new project, perform the following steps:

1. Choose Programs > Altera > Quartus II <version> (Windows Start menu) to run the Quartus II software. You can also use the Quartus II Web Edition software.

2. Choose New Project Wizard (File menu).

3. Click Next in the introduction (the introduction will not display if you turned it off previously).

4. Specify the working directory for your project. This walkthrough uses the directory d:\temp.

5. Specify the name of the project. This walkthrough uses example.

6. Click Next.

7. Click User Library Pathnames.

8. Type <path>\turbo_codec-v1.6.0\lib\ into the Library name box, where <path> is the directory in which you installed the Turbo Encoder and Decoder MegaCore function. The default installation directory is c:\megacore.

9. Click Add.

10. Click OK.

11. Click Next.

12. Choose the device family that you wish to target in the Family list.

13. Click Finish.
You have finished creating your new Quartus II project.

**Launch IP Toolbench**

To launch IP Toolbench, follow these steps:

1. Start the MegaWizard® Plug-In Manager by choosing **MegaWizard Plug-In Manager** (Tools menu). The **MegaWizard Plug-In Manager** dialog box is displayed.

   Refer to the Quartus II Help for more information on how to use the MegaWizard Plug-In Manager.

2. Specify that you want to create a new custom megafunction variation and click **Next**.

3. Select **Turbo Codec v1.6.0** in the **DSP > Error detection/correction** directory.

4. Choose the output file type for your design; the wizard supports AHDL, VHDL, and Verilog HDL.

5. Specify a name for the output file, `<directory name>\<variation name>`. **Figure 2–3** shows the wizard after you have made these settings.
6. Click Next to launch IP Toolbench.

**Step 1: Parameterize**

To parameterize your MegaCore function, follow these steps:

1. Click Step 1: Parameterize in IP Toolbench (see Figure 2–4).
2. Read the message and click **OK**.

3. Select either decoder or encoder and choose your parameters (see **Figure 2–5**).
4. Click Next.

5. The order code for your megafuction displays. Click Finish (see Figure 2–6).
Step 2: Set Up Simulation

An IP functional simulation model is a cycle-accurate VHDL or Verilog HDL model file produced by the Quartus II software (version 3.0 or higher). It allows for fast functional simulation of IP using industry-standard VHDL and Verilog HDL simulators.

You may only use these simulation model output files for simulation purposes and expressly not for synthesis or any other purposes. Using these models for synthesis creates a non-functional design.

To generate an IP functional simulation model for your MegaCore function, follow these steps:
1. Click **Step 2: Set Up Simulation** in IP Toolbench (see **Figure 2–7**).

**Figure 2–7. Set Up Simulation**

![Set Up Simulation](image)

2. Turn on **Generate Simulation Model** (see **Figure 2–8**).

**Figure 2–8. Generate Simulation Model**

![Generate Simulation Model](image)

An IP Functional Simulation Model is a cycle-accurate VHDL or Verilog model produced by the Quartus® II Software. It allows for fast functional simulations of IP using industry-standard VHDL and Verilog simulators.

You may only use these simulation model outputs for simulation purposes and expressly not for synthesis or any other purposes. Using these models for synthesis will create a non-functional design.

3. Choose the language in the **Language** list.
4. Click OK.

Generating a simulation model for the turbo decoder may take some time.

**Step 3: Generate**

To generate your MegaCore function, follow these steps:

1. Click **Step 3: Generate** in IP Toolbench (see Figure 2–9).

**Figure 2–9. IP Toolbench—Generate**

2. The generation report lists the design files that IP Toolbench creates (see Figure 2–10). Click **Exit**.
Table 2–2 describes the IP Toolbench-generated files.

<table>
<thead>
<tr>
<th>Extension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.vhd, .v, or .tdf</td>
<td>A MegaCore function variation file, which defines a VHDL, Verilog HDL or AHDL top-level description of the custom MegaCore function. Instantiate the entity defined by this file inside of your design. Include this file when compiling your design in the Quartus II software.</td>
</tr>
<tr>
<td>.cmp</td>
<td>A VHDL component declaration file for the MegaCore function variation. Add the contents of this file to any VHDL architecture that instantiates the MegaCore function.</td>
</tr>
<tr>
<td>.inc</td>
<td>An AHDL include declaration file for the MegaCore function variation. Include this file with any AHDL architecture that instantiates the MegaCore function.</td>
</tr>
<tr>
<td>_bb.v</td>
<td>A Verilog HDL black-box file for the MegaCore function variation. Use this file when using a third-party EDA tool to synthesize your design.</td>
</tr>
<tr>
<td>.bsf</td>
<td>A Quartus II symbol file for the MegaCore function variation. You can use this file in the Quartus II block diagram editor.</td>
</tr>
</tbody>
</table>
Simulate your Design

You can now integrate your custom variation into your design and simulate and compile.

Simulate your Design

You can simulate your design using the IP Toolbench-generated VHDL or Verilog HDL functional simulation models, or the precompiled ModelSim® VHDL models.

For more information on IP functional simulation models, refer to the white paper Using IP Functional Simulation Models to Verify Your System Design.

Altera supplies a VHDL reference design for the decoder (see Figure 2–11). The reference design is supplied as source code—it instantiates the decoder, it is synthesizable, and it can be used as a basis for your design.

---

**Table 2–2. IP Toolbench-Generated Files (Part 2 of 2)**

<table>
<thead>
<tr>
<th>Extension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.html</td>
<td>The MegaCore function report file.</td>
</tr>
<tr>
<td>.vo or .vho</td>
<td>A VHDL or Verilog HDL IP functional simulation model.</td>
</tr>
<tr>
<td>_inst.vhd or _inst.v</td>
<td>A VHDL or Verilog HDL sample instantiation file.</td>
</tr>
</tbody>
</table>

---

Altera also supplies a VHDL system testbench (see Figure 2–12), which you can use to simulate the functionality of the Turbo Decoder and Encoder MegaCore function. You can simulate the testbench using IP functional simulation models or the precompiled ModelSim VHDL models.
You can use the IP functional simulation model with any Altera-supported VHDL or Verilog HDL simulator. To use the IP functional simulation model that you created in “Step 2: Set Up Simulation” on page 2–12 in the Altera-provided reference design and testbench with the ModelSim simulator, follow these steps:

1. Launch the ModelSim-Altera simulator.

   If you are not using the ModelSim-Altera simulator, refer to the white paper Using IP Functional Simulation Models to Verify Your System Design.

2. Change the directory to your Quartus II project directory by typing `cd <project directory>`.

3. Create a library called work, by typing the following commands:

   ```
   vlib work
   vmap work work
   ```

4. Compile your IP functional simulation models, `<encoder variation name>.vho` and `<decoder variation name>.vho`.

   The testbench requires both an encoder and decoder.

5. Compile the following testbench support files, which are in the `\sim_lib\testbench` directory:
Simulate your Design

- `auktde_tdec_refdesign_ctrl.vhd`
- `aukte_tenc_refdesign_ctrl.vhd`
- `awgn_bpsk_channel.vhd`

6. When you generate your MegaCore function, IP Toolbench creates a VHDL testbench, which is parameterized to match your MegaCore function. Compile the following VHDL testbench files from your Quartus II project directory.

- `auktde_umts_turbo_decoder_chip.vhd`
- `auktde_turbo_tb.vhd`

7. Type `vsim work.auktde_turbo_tb`

8. Suppress the simulation warning options, choose Simulate Options > Suppress Warnings (Simulate menu). Turn on Synopsys and IEEE.

9. Type `run -all`

To change any of the testbench parameters such as iterations, clock period, block size, etc., edit the file `auktde_turbo_tb.vhd` with a text editor.

Use the ModelSim VHDL Models

To use the ModelSim VHDL models in the Altera-provided reference design and testbench with the ModelSim simulator, follow these steps:

1. Launch the ModelSim-Altera simulator.

2. Change the directory to your Quartus II project directory by typing `cd <project directory>`.

3. Create a library called work, by typing the following commands:

   ```vlib work```
   ```vmap work work```

4. Create a logical map called turbolib to the directory containing the precompiled library by typing the following command in the ModelSim simulator.

   ```vmap turbolib <Drive>:/<Turbo MegaCore Path>/sim_lib/modelsim/vhdl/turbolib```
You can also use the ModelSim graphical user interface (GUI) to create the logical map. Refer to the ModelSim online help for details.

5. Repeat step 4 for the memlib and itlvlib libraries by replacing turbolib with memlib and itlvlib.

Depending on the version of ModelSim that you use, you may need to refresh after you have created logical maps to your libraries. To refresh, type in the following command for each of the libraries: turbolib, memlib, and rtlvlib:

```
vcom -refresh -work <library name>```

6. Compile the IP Toolbench-generated design files, <encoder variation name>.vhd and <decoder variation name>.vhd.

The testbench requires both an encoder and decoder.

7. Compile the following testbench support files, which are in the \sim_lib\testbench directory:

- auktd_tdec_refdesign_ctrl.vhd
- aukte_tenc_refdesign_ctrl.vhd
- awgn_bpsk_channel.vhd

8. When you generate your MegaCore function, IP Toolbench creates a VHDL testbench, which is parameterized to match your MegaCore function. Compile the following VHDL testbench files from your Quartus II project directory.

- auktd_umts_turbo_decoder_chip.vhd
- auktde_turbo_tb.vhd

9. Type `vsim work.auktde_turbo_tb`

10. Suppress the simulation warning options, choose Simulate Options > Suppress Warnings (Simulate menu). Turn on Synopsys and IEEE.

11. Type `run -all`

To change any of the testbench parameters such as iterations, clock period, block size, etc., edit the file `auktde_turbo_tb.vhd` with a text editor.
Compile the Design

You can use the Quartus II software to compile your design. Refer to Quartus II Help for instructions on performing compilation.

Program a Device

After you have compiled your design, program your targeted Altera device, and verify your design in hardware.

With Altera’s free OpenCore Plus evaluation feature, you can evaluate the Turbo Encoder and Decoder MegaCore function before you purchase a license. OpenCore Plus evaluation allows you to generate an IP functional simulation model, and produce a time-limited programming file.

For more information on IP functional simulation models, refer to the white paper Using IP Functional Simulation Models to Verify Your System Design.

You can simulate the Turbo Encoder and Decoder MegaCore function in your design, and perform time-limited evaluation of your design in hardware.


You need to purchase a license for the MegaCore function only when you are completely satisfied with its functionality and performance, and want to take your design to production.

Set Up Licensing

When you are satisfied with the MegaCore function you can purchase a license.

After you purchase a license for Turbo Encoder and Decoder MegaCore function, you can request a license file from the Altera web site at www.altera.com/licensing and install it on your computer. When you request a license file, Altera e-mails you a license.dat file. If you do not have Internet access, contact your local Altera representative.

To install your license, you can either append the license to your license.dat file or you can specify the MegaCore function’s license.dat file in the Quartus II software.
Before you set up licensing for the Turbo Encoder and Decoder MegaCore function, you must already have the Quartus II software installed on your computer, with licensing set up.

Append the License to Your license.dat File

To append the license, perform the following steps:

1. Close the following software if it is running on your computer:
   - Quartus II software
   - MAX+PLUS® II software
   - LeonardoSpectrum™ synthesis tool
   - Synplify software
   - ModelSim simulator

2. Open the Turbo Encoder and Decoder MegaCore function license file in a text editor. The file should contain one FEATURE line, spanning several lines.

3. Open your Quartus II license.dat file in a text editor.

4. Copy the FEATURE line from the Turbo Encoder and Decoder MegaCore function license file and paste it into a new line in the Quartus II license file.

   Do not delete any FEATURE lines from the Quartus II license file.

5. Save the Quartus II license file as a text file.

   When using editors such as Microsoft Word or Notepad, ensure that the file does not have extra extensions appended to it after you save (e.g., license.dat.txt or license.dat.doc). Verify the filename at a command prompt.

Specify the License File in the Quartus II Software

To specify the MegaCore function’s license file, perform the following steps:

   Altera recommends that you give the file a unique name, e.g., <MegaCore name>_license.dat.

1. Run the Quartus II software.
2. Choose License Setup (Tools menu). The Options dialog box opens to the License Setup page.

3. In the License file box, add a semicolon to the end of the existing license path and filename.

4. Type the path and filename of the MegaCore function license file after the semicolon.

   ![Caution](image) Do not include any spaces either around the semicolon or in the path/filename.

5. Click OK to save your changes.
Figure 3–1 shows the turbo encoder interface; Figure 3–2 shows the turbo encoder block diagram.

**Figure 3–1. The Turbo Encoder Interface**

- DATA_IN → DATA_OUT
- SHIFT_IN_ENABLE → PARITY 1
- SHIFT_OUT_ENABLE → PARITY 2
- BLOCK_SIZE → DATA_VALID
- ITLV_INIT → INPUT_READY
- RESET → OUTPUT_READY
- CLK → OUTPUT_BUSY

**Figure 3–2. Block Diagram**

Turbo Encoder

Information bits → Double Buffer → Transmitted information bits

Transmitted parity 1 bits

Interleaver → Encoder 1 → Transmitted parity 1 bits

Transmitted parity 2 bits

Encoder 2
OpenCore Plus Time-Out Behavior

OpenCore® Plus hardware evaluation can support the following two modes of operation:

- **Untethered**—the design runs for a limited time
- **Tethered**—requires a connection between your board and the host computer. If tethered mode is supported by all megafunctions in a design, the device can operate for a longer time or indefinitely

All megafunctions in a device time out simultaneously when the most restrictive evaluation time is reached. If there is more than one megafunction in a design, a specific megafunction’s time-out behavior may be masked by the time-out behavior of the other megafunctions.

For MegaCore® functions, the untethered timeout is 1 hour; the tethered timeout value is indefinite.

Your design stops working after the hardware evaluation time expires and the DATA_OUT, PARITY1 and PARITY2 signals go low.

For more information on OpenCore Plus hardware evaluation, see “OpenCore Plus Evaluation” on page 1–5 and AN 320: OpenCore Plus Evaluation of Megafunctions.

Parameters

Table 3–1 shows the Turbo encoder MegaCore function’s parameters.

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BANKSWAP</td>
<td>0 or 1</td>
<td>The number of banks of information memory. 0 = single memory bank; 1 = 2 banks of information memory, for maximum performance.</td>
</tr>
</tbody>
</table>

Signals

Table 3–2 shows the interface signal definitions.

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK</td>
<td>1</td>
<td>Clock.</td>
</tr>
<tr>
<td>RESET</td>
<td>1</td>
<td>Asynchronous reset.</td>
</tr>
<tr>
<td>BLOCK_SIZE</td>
<td>13</td>
<td>Block size. Between 40 and 5,114</td>
</tr>
</tbody>
</table>
Specifications—Encoder

The turbo encoder includes double buffered storage for the input of information bits. Double buffering allows a new block of data to be shifted in whilst the previous block is encoded and shifted out.

### Encoder Operation

The turbo encoder is a slave device that is operated using two control input signals: \texttt{SHIFT\_IN\_ENABLE} and \texttt{SHIFT\_OUT\_ENABLE}. Because the Turbo Encoder is a slave device, it is up to the host to provide the control signals at the correct time with regard to the encoder’s state. Three status signals are used to indicate the encoder’s state: \texttt{INPUT\_READY}, \texttt{OUTPUT\_READY}, and \texttt{OUTPUT\_BUSY}.

Figure 3–3 shows the shift-in timing diagram. Shifting data in can only commence if \texttt{INPUT\_READY} is high. \texttt{ITLV\_INIT} is asserted high for one clock cycle to initialize the interleaver (necessary only if \texttt{BLOCK\_SIZE} is changed and after \texttt{RESET}). The data at \texttt{DATA\_IN} must be valid when \texttt{SHIFT\_IN\_ENABLE} is high, as it will be registered on the next rising clock edge. After a rising clock edge, \texttt{DATA\_IN} may be changed without asserting \texttt{SHIFT\_IN\_ENABLE} low. \texttt{SHIFT\_IN\_ENABLE} may be asserted low at any time to insert a pause in the input stream if data cannot be made available before the rising clock edge.

#### Table 3–2. Interface Signals (Part 2 of 2)

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITLV_INIT</td>
<td>1</td>
<td>Interleaver initialization. Must be asserted high for one or more clock</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cycles after reset, or after changing \texttt{BLOCK_SIZE}. (1)</td>
</tr>
<tr>
<td>INPUT_READY</td>
<td>1</td>
<td>Indicates that the encoder is ready to accept input data.</td>
</tr>
<tr>
<td>OUTPUT_READY</td>
<td>1</td>
<td>Indicates that the encoder is ready to output data.</td>
</tr>
<tr>
<td>SHIFT_IN_ENABLE</td>
<td>1</td>
<td>When high, \texttt{DATA_IN} will be shifted into the turbo encoder on the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>next rising clock edge.</td>
</tr>
<tr>
<td>SHIFT_OUT_ENABLE</td>
<td>1</td>
<td>When high, the next output will be made available on \texttt{DATA_OUT} after</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the next rising clock edge.</td>
</tr>
<tr>
<td>DATA_IN</td>
<td>1</td>
<td>Data input.</td>
</tr>
<tr>
<td>DATA_OUT</td>
<td>1</td>
<td>Data output.</td>
</tr>
<tr>
<td>PARITY1</td>
<td>1</td>
<td>Parity 1 output.</td>
</tr>
<tr>
<td>PARITY2</td>
<td>1</td>
<td>Parity 2 output.</td>
</tr>
<tr>
<td>OUTPUT_BUSY</td>
<td>1</td>
<td>Indicates that the encoder still has some internal data.</td>
</tr>
</tbody>
</table>

Note to Table 3–2:

(1) \texttt{ITLV\_INIT} should only be asserted when \texttt{OUTPUT\_BUSY} is low, and before data is shifted-in.
Encoder Operation

**Figure 3–3. Shift-in Timing Diagram**

<table>
<thead>
<tr>
<th>CLK</th>
<th>DATA_IN</th>
<th>SHIFT_IN_ENABLE</th>
<th>ITLV_INIT</th>
<th>INPUT_READY</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>D1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>D2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>D3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>D4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>D5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The encoder asserts INPUT READY low when it detects that the required number of bits have been shifted in. The host should not try to shift more data in, as any attempt to do so will be ignored by the encoder.

The INPUT READY signal is asserted high when the encoder can accept a new block of data to shift in. If double buffering is selected, a block can be shifted in while the previous block is being encoded and shifted out.

**Figure 3–4** shows the shift-out timing diagram. After OUTPUT READY has gone from low to high, the encoder is ready to shift-out the data using SHIFT_OUT_ENABLE. The output is registered and DATA_OUT is not valid until after the next rising clock edge. SHIFT_OUT_ENABLE can be asserted low at any time to create a pause in the output stream. When all of the punctured data has been read from the encoder, it enters the finished phase.

**Figure 3–4. Shift-Out Timing Diagram**

<table>
<thead>
<tr>
<th>CLK</th>
<th>OUTPUT_READY</th>
<th>SHIFT_OUT_ENABLE</th>
<th>DATA_OUT</th>
<th>PARITY1</th>
<th>PARITY2</th>
<th>OUTPUT_BUSY</th>
<th>DATA_VALID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>P11</td>
<td>P21</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>D1</td>
<td>P12</td>
<td>P22</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>D2</td>
<td>P13</td>
<td>P23</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>D3</td>
<td>X</td>
<td>P24</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The **DATA_VALID** output is asserted low by the encoder, whenever the data and parity outputs are invalid because of pruning. This situation occurs with some block sizes.

The **OUTPUT_BUSY** signal indicates that the encoder has some data remaining to be shifted out, even though **OUTPUT_READY** may be asserted low. This situation occurs if a block of data has just been shifted out and a second block is available in the encoder but not quite ready to be shifted out. The interleaver must not be re-initialized until **OUTPUT_BUSY** is asserted low, which indicates that the encoder has finished encoding and shifting out all data.

The trellis termination of the turbo code is described in *3rd Generation Partnership Project (3GPP); Technical Specification Group Radio Access Network; Multiplexing and Channel Coding (FDD) (3G TS 25.212 version 3.2.0).* The relevant section is repeated here.

Tail bits are added after the encoding of information bits. The trellis is terminated by taking the tail bits from the shift register feedback after all of the information bits have been encoded. The first three tail bits are used to terminate the first convolutional encoder; the last three tail bits to terminate the second convolutional encoder. Each tail bit is followed by a parity bit from the convolutional encoder being terminated. After all data bits have been shifted out, **DATA_VALID** asserts low for three clock cycles and the encoder generates the tail bits. The **SHIFT_OUT_ENABLE** signal is ignored during this time. The **DATA_VALID** signal then asserts high for four clock cycles and the tail bits are output (see Table 3–1). The **SHIFT_OUT_ENABLE** signal may be used to insert a pause in the tail bits output, as for the data output (see Table 3–3).

<table>
<thead>
<tr>
<th>Table 3–3. Tail Bits Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Clock</strong></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

where:

X is the tail bit from the first convolutional encoder
Y is the parity bit from the first convolutional encoder
X' is the tail bit from the second convolutional encoder
Y' is the parity bit from the second convolutional encoder
Figure 3–5 shows the tail bits timing diagram.

**Figure 3–5. Tail Bits Output**

<table>
<thead>
<tr>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK</td>
<td>Clock</td>
</tr>
<tr>
<td>OUTPUT_READY</td>
<td>Output ready</td>
</tr>
<tr>
<td>SHIFT_OUT_ENABLE</td>
<td>Shift output enable</td>
</tr>
<tr>
<td>DATA_OUT</td>
<td>Data output</td>
</tr>
<tr>
<td>PARITY1</td>
<td>Parity 1</td>
</tr>
<tr>
<td>PARITY2</td>
<td>Parity 2</td>
</tr>
<tr>
<td>DATA_VALID</td>
<td>Data valid</td>
</tr>
</tbody>
</table>

For the encoder, a counter is required that counts the parity bits the turbo encoder produces. The count values can be decoded to flag which bits to puncture. You must then remove the flagged bits from the data stream before transmission.

For the decoder, depuncturing must be implemented. Write the value ‘0’ into the memory locations where data is missing from the input data stream.

**MegaCore Verification**

The MegaCore verification includes an automated regression test suite, which is described in the following paragraphs.

Encoder and decoder are tested separately. In both cases, the VHDL implementation is compared against a bit-accurate C model, which has been verified independently and is used as golden reference. A perl script runs a number of simulations, where the C model and the VHDL model are exercised with the same stimuli, and their results are compared. Any difference is considered as failure of the regression test.

The regression test covers a wide range of parameter combinations. Each individual parameter and the block size input port are exercised with every possible value, with the other parameters fixed. Random data are used as data sequence. In addition, all parameters are varied randomly to test at least some parameter combinations that are not covered by the tests described above.
Table 3–4 shows the typical performance using the Quartus® II version 4.0 software available in Stratix™ II, Stratix, and Cyclone™ devices.

<table>
<thead>
<tr>
<th>Device</th>
<th>Logic Elements (LEs)</th>
<th>Memory (Bits)</th>
<th>$f_{\text{MAX}}$ (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stratix II (1), (2)</td>
<td>1,669</td>
<td>17,152</td>
<td>157</td>
</tr>
<tr>
<td>Stratix (3)</td>
<td>2,057</td>
<td>17,152</td>
<td>101</td>
</tr>
<tr>
<td>Cyclone (4)</td>
<td>2,057</td>
<td>17,152</td>
<td>101</td>
</tr>
</tbody>
</table>

**Note to Table 3–4:**
(1) The Quartus II software reports the number of adaptive look-up tables (ALUTs) that the design uses in Stratix II devices. The LE count is based on this number of ALUTs.
(2) EP2S15F484C3 device.
(3) EP1S10F484C5 device.
(4) EP1C3T100C6 device.
Figure 4–1 shows an example system that contains the turbo decoder MegaCore® function. In this example, the processor controls the operation of the MegaCore function by setting up the parameters and initiating the decode of each block. The processor writes the information and parity data into the appropriate memory, and reads the decoded information from the information likelihood memory. All writing and reading is in the form of logarithmic likelihood values.

**Data Format**

The turbo decoder requires all data to be in the log-likelihood format. The equalizer has to provide soft information, also parity 1 and parity 2 bit sequences according to the following equation:

\[
L(x) = \log \frac{P(x = 0)}{P(x = 1)}
\]

The log-likelihood value is the logarithm of the probability that the received bit is a ‘0’, divided by the probability of this bit being a ‘1’, and is represented as a two’s complement number. A value of zero indicates equal probability of a ‘1’ and a ‘0’, which should be used for de-
puncturing. The most negative two’s complement number is unused so that the representation is balanced. As an example, Table 4–1 shows the meanings of 3-bit values.

Table 4–1. The Meanings of 3-bit Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>011</td>
<td>Maximum likelihood of a ‘0’</td>
</tr>
<tr>
<td>010</td>
<td>Medium likelihood of a ‘0’</td>
</tr>
<tr>
<td>001</td>
<td>Low likelihood of a ‘0’</td>
</tr>
<tr>
<td>000</td>
<td>Equal probability of a ‘1’ and a ‘0’</td>
</tr>
<tr>
<td>111</td>
<td>Low likelihood of a ‘1’</td>
</tr>
<tr>
<td>110</td>
<td>Medium likelihood of a ‘1’</td>
</tr>
<tr>
<td>101</td>
<td>Maximum likelihood of a ‘1’</td>
</tr>
<tr>
<td>100</td>
<td>Not used</td>
</tr>
</tbody>
</table>

The output data is provided in the same log-likelihood format. If only hard information is required, the sign bit can be considered as a hard decision bit.

Memory Requirements

The logarithmic maximum a posteriori (max-logMAP) decoder requires access to the described memories.

The information-likelihood memory stores the logarithmic likelihood values for the information bits. This data is written into the memory before decoding begins, and the corrected values are read back from this memory after decoding.

The parity likelihood memories store the logarithmic likelihood values for the parity 1 and parity 2 bits. This data is written into the memory before decoding begins.

The a priori memory stores correction values that are passed between iterations of the turbo decoder. It is not externally accessible.

The alpha matrix memory is used as intermediate storage by the max-logMAP decoders. It is not externally accessible.

The information likelihood memory and the a priori memory are included in the turbo decoder MegaCore function and are always implemented as on-chip memory. The parity likelihood memories and alpha matrix memory can be implemented on-chip or off-chip. They are
not included in the turbo decoder, and you must connect them to the turbo decoder. The turbo decoder requires single-cycle access to each of these memories, although access can be pipelined. Synchronous SRAM is recommended for off-chip memories. The alpha matrix and parity likelihood memories must always be separate from each other.

The turbo decoder supports the use of two banks of information memory, which allows one bank to be written and read by the processor, while the data in the other bank is processed. The use of two banks of information memory minimizes the delay between decoding each block and achieves the maximum throughput. When using this option, two banks of parity-likelihood memory should also be implemented outside the MegaCore function. The address space for parity 1 likelihood memory is from 0 to 8,191 \((2^{13} - 1)\), addresses 000..00 to 011..11; for parity 2 from 8,192 to 16,383 \((2^{14} - 1)\), addresses 100..00 to 111..11. If maximum block size is used the address space for parity 1 likelihood memory is from 0 to 5,116; parity 2 from 8,192 to 13308.

**OpenCore Plus Time-Out Behavior**

OpenCore® Plus hardware evaluation can support the following two modes of operation:

- **Untethered**—the design runs for a limited time
- **Tethered**—requires a connection between your board and the host computer. If tethered mode is supported by all megafunctions in a design, the device can operate for a longer time or indefinitely

All megafunctions in a device time out simultaneously when the most restrictive evaluation time is reached. If there is more than one megafunction in a design, a specific megafunction’s time-out behavior may be masked by the time-out behavior of the other megafunctions.

For MegaCore functions, the untethered timeout is 1 hour; the tethered timeout value is indefinite.

Your design stops working after the hardware evaluation time expires and the **INFO_DATA_OUT** signal goes low

For more information on OpenCore Plus hardware evaluation, see “OpenCore Plus Evaluation” on page 1–5 and *AN 320: OpenCore Plus Evaluation of Megafunctions*. 
Parameters

Table 4–2 shows the parameters.

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOFTBITS</td>
<td>3 to 8</td>
<td>The number of bits (N) used for the log likelihood values for information and parity bits.</td>
</tr>
<tr>
<td>TMEMACCA</td>
<td>2 to 6</td>
<td>The number of pipeline stages outside the MegaCore function for read access to the alpha matrix memory.</td>
</tr>
<tr>
<td>TMEMACCP</td>
<td>2 to 6</td>
<td>The number of pipeline stages outside the MegaCore function for read access to the parity likelihood memory.</td>
</tr>
<tr>
<td>BANKSWAP</td>
<td>0 or 1</td>
<td>Indicates whether two banks of information memory are to be included in the MegaCore function to achieve maximum performance.</td>
</tr>
</tbody>
</table>

Signals

Table 4–3 shows the general signals.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK</td>
<td>Input</td>
<td>System clock.</td>
</tr>
<tr>
<td>RESET</td>
<td>Input</td>
<td>Global asynchronous reset.</td>
</tr>
</tbody>
</table>

Table 4–4 shows the configuration and control interface signal definitions.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>START</td>
<td>Input</td>
<td></td>
<td>Instructs MegaCore function to begin decoding a block of data.</td>
</tr>
<tr>
<td>LD_INT</td>
<td>Input</td>
<td></td>
<td>Instructs interleaver to re-initialize with new block size.</td>
</tr>
<tr>
<td>ACTIVE</td>
<td>Output</td>
<td></td>
<td>Indicates MegaCore function is busy processing data or initializing the interleaver.</td>
</tr>
<tr>
<td>PUNCTURE</td>
<td>Input</td>
<td></td>
<td>When asserted, the MegaCore function will set alternate Parity 1 and Parity 2 values to 0.</td>
</tr>
<tr>
<td>BLOCK_SIZE</td>
<td>Input</td>
<td>13</td>
<td>Block size.</td>
</tr>
<tr>
<td>ITERATIONS</td>
<td>Input</td>
<td>4</td>
<td>Number of iterations.</td>
</tr>
</tbody>
</table>
Table 4–5 shows the information memory interface signal definitions.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFO_DATA_IN</td>
<td>Input</td>
<td>N</td>
<td>Memory data input.</td>
</tr>
<tr>
<td>INFO_ADDR</td>
<td>Input</td>
<td>13</td>
<td>Memory address.</td>
</tr>
<tr>
<td>INFO_RD</td>
<td>Input</td>
<td></td>
<td>Memory read strobe.</td>
</tr>
<tr>
<td>INFO_WR</td>
<td>Input</td>
<td></td>
<td>Memory write strobe.</td>
</tr>
<tr>
<td>INFO_DATA_OUT</td>
<td>Output</td>
<td>N</td>
<td>Memory data output.</td>
</tr>
</tbody>
</table>

Table 4–6 shows the parity likelihood memory interface signal definitions.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARITY_DATA_IN</td>
<td>Input</td>
<td>N</td>
<td>Memory data input.</td>
</tr>
<tr>
<td>PARITY_ADDR</td>
<td>Output</td>
<td>14</td>
<td>Memory address.</td>
</tr>
<tr>
<td>PARITY_RD</td>
<td>Output</td>
<td></td>
<td>Information memory read strobe.</td>
</tr>
</tbody>
</table>

Table 4–7 shows the alpha matrix memory interface signal definitions.

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHA_DATA_IN</td>
<td>Input</td>
<td>8 \times (N – 1)</td>
<td>Data from memory.</td>
</tr>
<tr>
<td>ALPHA_DATA_OUT</td>
<td>Output</td>
<td>8 \times (N – 1)</td>
<td>Data to memory.</td>
</tr>
<tr>
<td>ALPHA_ADDR</td>
<td>Output</td>
<td>13</td>
<td>Memory address.</td>
</tr>
<tr>
<td>ALPHA_RD</td>
<td>Output</td>
<td></td>
<td>Memory read strobe.</td>
</tr>
<tr>
<td>ALPHA_WR</td>
<td>Output</td>
<td></td>
<td>Memory write strobe.</td>
</tr>
</tbody>
</table>

**Decoder Operation**

The turbo decoder MegaCore function operates as a slave device under the control of a processor. The operating sequence is described in the steps that follow.
1. Reset the MegaCore function by asserting the **RESET** signal. The **RESET** signal must be de-asserted synchronously with respect to **CLK** to avoid metastability issues.

2. Configure the MegaCore function by setting the **PUNCTURE**, **ITERATIONS** and **BLOCK_SIZE** configuration signals.

3. Assert **LD_INT** to initialize the interleaver, if **BLOCK_SIZE** has been changed, and wait until **ACTIVE** goes inactive (see Figure 4–2).

4. Write information and parity likelihood values into their respective memories.

5. Assert **START** to begin the decoding of the block.

6. Wait until **ACTIVE** goes inactive.

7. Read the corrected information likelihood values from the information memory.

8. Repeat from step 4 (or from step 2 if the interleaver block size changes).

Figure 4–3 shows the turbo decoder timing diagram.

**Figure 4–2. Initializing the Turbo Decoder**

<table>
<thead>
<tr>
<th>CLK</th>
<th>---------------</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD_INT</td>
<td></td>
</tr>
<tr>
<td>ACTIVE</td>
<td></td>
</tr>
<tr>
<td>INFO_DATA_IN</td>
<td></td>
</tr>
<tr>
<td>INFO_ADDR</td>
<td></td>
</tr>
<tr>
<td>INFO_WR</td>
<td></td>
</tr>
<tr>
<td>START</td>
<td></td>
</tr>
</tbody>
</table>

**Note:**

(1) **BLOCK_SIZE** (not shown) must be valid at least one clock cycle before **LD_INT** is asserted, and then must continue to be valid.
Note: START is asserted for one (or more) clock cycles, while ACTIVE is low. One clock cycle after START is asserted, ACTIVE goes high.

The sequence is slightly different when two banks of information memory are used. Bank swapping automatically takes place every time START is asserted. The corrected information likelihood values from the previous block can be read, and the new information likelihood values for the next block written, while the MegaCore function is decoding the current block (see Figure 4–4). If you use bank swapping, implement two banks of parity likelihood memory outside the MegaCore function.
Decoder Operation

Figure 4–4. Turbo Decoder Timing Diagram—with Bank Swapping Note (1)

Note:
(1) START is asserted for one (or more) clock cycles, while ACTIVE is low. One clock cycle after START is asserted, ACTIVE goes high.

External Puncturing and Depuncturing

The turbo decoder has support for rates 1/2 and 1/3 built in. If other puncturing rates are required, you can implemented them externally.

For the encoder, a counter is required that counts the parity bits the turbo encoder produces. The count values can be decoded to flag which bits to puncture. You must then remove the flagged bits from the data stream before transmission.
The MegaCore verification includes an automated regression test suite, which is described in the following paragraphs.

Encoder and decoder are tested separately. In both cases, the VHDL implementation is compared against a bit-accurate C model, which has been verified independently and is used as golden reference. A perl script runs a number of simulations, where the C model and the VHDL model are exercised with the same stimuli, and their results are compared. Any difference is considered as failure of the regression test.

The regression test covers a wide range of parameter combinations. Each individual parameter and the block size input port are exercised with every possible value, with the other parameters fixed. Random data are used as data sequence. In addition, all parameters are varied randomly to test at least some parameter combinations that are not covered by the tests described above.

The max-logMAP decoder requires two clock cycles to decode each bit, plus a few cycles to fill the pipeline at the start of each decoding block. The max-logMAP decoder needs to operate twice for each iteration of the turbo decoder; once for each set of parity bits. Hence each iteration of the turbo decoder requires 4 clock cycles per information bit.

For example, with five iterations, 20 clock cycles are required per sample. A bit rate of 2 Mbps can be achieved with a clock frequency of 50 MHz.

Higher throughput can be achieved by using several turbo decoders in parallel. The decoding of each block is totally independent of all other blocks.

The amount of logic needed for the turbo decoder MegaCore function varies between about 5,000 and 7,500 logic elements (LEs) and depends on the decoder’s parameters. The size of the memories varies with the number of bits used to represent the soft decision (logarithmic likelihood) values. Table 4–8 defines the various memory sizes, where \( N \) is the number of bits used to represent the logarithmic likelihood values:

<table>
<thead>
<tr>
<th>Memory Name</th>
<th>Size</th>
<th>Size with Bank Swapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information Likelihood</td>
<td>5K ( \times N )</td>
<td>2 ( \times 5K ) ( \times N )</td>
</tr>
<tr>
<td>Parity Likelihood</td>
<td>10K ( \times N )</td>
<td>2 ( \times 10K ) ( \times N )</td>
</tr>
<tr>
<td>A priori</td>
<td>5K ( \times N )</td>
<td>5K ( \times N )</td>
</tr>
<tr>
<td>Alpha Matrix</td>
<td>5K ( \times 8(N - 1) )</td>
<td>5K ( \times 8(N - 1) )</td>
</tr>
</tbody>
</table>
The information likelihood and a priori memories must be on-chip and are included with the turbo decoder. The parity likelihood and alpha matrix memories are outside the MegaCore function, and can be implemented on-chip or off-chip, which allows you to select a configuration best suited to your system. The information likelihood and parity memories may be duplicated to increase throughput (bank swapping), by avoiding the need to read the results and write the new data between blocks.

Table 4–9 shows the typical performance using the Quartus® II version 4.0 software and the default parameters \(\text{SOFTBITS} = 5, \text{TMEMACCA} = 4, \text{TMEMACCP} = 4, \text{BANKSWAP} = 0\) available in Stratix™ II, Stratix, and Cyclone™ devices, including all of the off-MegaCore function memories.

<table>
<thead>
<tr>
<th>Device</th>
<th>LEs</th>
<th>Memory (Bits)</th>
<th>(f_{\text{MAX}}) (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stratix II (1), (2)</td>
<td>4,627</td>
<td>56,064</td>
<td>105</td>
</tr>
<tr>
<td>Stratix (3)</td>
<td>5,160</td>
<td>56,064</td>
<td>84</td>
</tr>
<tr>
<td>Cyclone (4)</td>
<td>5,160</td>
<td>56,064</td>
<td>80</td>
</tr>
</tbody>
</table>

Notes:
1. The Quartus II software reports the number of adaptive look-up tables (ALUTs) that the design uses in Stratix II devices. The LE count is based on this number of ALUTs.
2. EP2S15F484C3 device.
3. EP1S10F484C5 device.
4. EP1C6F256C6 device.