Turbo Encoder/Decoder MegaCore Function User Guide

Version 1.0
April 2000

About this User Guide

User Guide

This user guide provides comprehensive information about the Altera® turbo encoder/decoder MegaCore™ function.

For the most up-to-date information about Altera IP products, go to the Altera world-wide web site at **http://www.altera.com/IPMegaStore**.

How to Contact Altera

For additional information about Altera products, consult the sources shown in Table 1.

Table 1. How to Contact Altera

Information Type	Access	USA & Canada	All Other Locations
Technical support	Telephone hotline	(800) 800-EPLD (6:00 a.m. to 6:00 p.m. Pacific Time)	(408) 544-7000 *(1)* (7:30 a.m. to 5:30 p.m. Pacific Time)
	Fax	(408) 544-6401	(408) 544-6401 *(1)*
	Electronic mail	**telecom@altera.com**	**telecom@altera.com**
	FTP site	**ftp.altera.com**	**ftp.altera.com**
Altera Literature Services	Telephone hotline	(888) 3-ALTERA *(1)*	(408) 544-7144 *(1)*
	Fax (please specify Altera Literature)	(408) 935-0515	(408) 935-0515
	Electronic mail	**lit_req@altera.com** *(1)*	**lit_req@altera.com** *(1)*
Non-technical customer service	Telephone hotline	(800) SOS-EPLD	(408) 544-7000 (7:30 a.m. to 5:30 p.m. Pacific Time)
	Fax	(408) 544-7606	(408) 544-7606
General product information	Telephone	(408) 544-7104	(408) 544-7104 *(1)*
	World-wide web site	**http://www.altera.com/IPMegaStore**	**http://www.altera.com/IPMegaStore**

Note:
(1) You can also contact your local Altera sales office or sales representative.

Typographic Conventions

The ***Turbo Encoder/Decoder MegaCore Function User Guide*** uses the typographic conventions shown in Table 2.

Table 2. Conventions

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: **Save As** dialog box.
bold type	External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: $\mathbf{f_{MAX}}$, **\maxplus2** directory, **d:** drive, **chiptrip.gdf** file.
Bold italic type	Book titles are shown in bold italic type with initial capital letters. Example: ***1999 Device Data Book***.
Italic Type with Initial Capital Letters	Document titles are shown in italic type with initial capital letters. Example: *AN 75 (High-Speed Board Design)*.
Italic type	Internal timing parameters and variables are shown in italic type. Examples: t_{PIA}, $n + 1$. Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: <*file name*>, <*project name*>**.pof** file.
Initial Capital Letters	Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu.
"Subheading Title"	References to sections within a document and titles of Quartus and MAX+PLUS II Help topics are shown in quotation marks. Example: "Configuring a FLEX 10K or FLEX 8000 Device with the BitBlaster™ Download Cable."
`Courier type`	Reserved signal and port names are shown in uppercase Courier type. Examples: `DATA1`, `TDI`, `INPUT`. User-defined signal and port names are shown in lowercase Courier type. Examples: `my_data`, `ram_input`. Anything that must be typed exactly as it appears is shown in Courier type. For example: `c:\max2work\tutorial\chiptrip.gdf`. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword `SUBDESIGN`), as well as logic function names (e.g., `TRI`) are shown in Courier.
1., 2., 3., and a., b., c.,...	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.
■	Bullets are used in a list of items when the sequence of the items is not important.
✓	The checkmark indicates a procedure that consists of one step only.
☞	The hand points to information that requires special attention.
↵	The angled arrow indicates you should press the Enter key.
	The feet direct you to more information on a particular topic.

Contents

Notes:

Specifications—Encoder

Contents

User Guide

Notes:

Specifications—Encoder

User Guide

Features

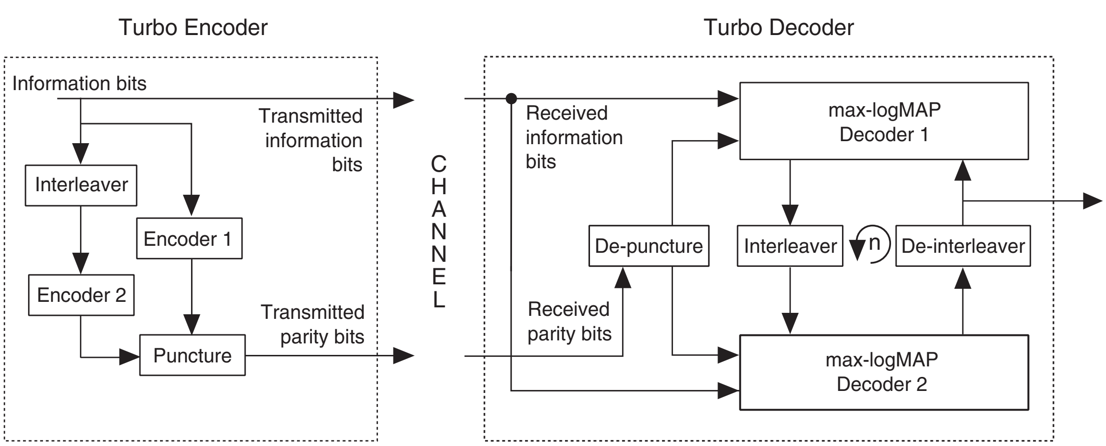
- Compliant with the *3rd Generation Partnership Project* (3GPP); *Technical Specification Group Radio Access Network; Multiplexing and Channel Coding (FDD) (3G TS 25.212 version 3.1.0)*
- Dramatically shortens design cycles
- Optimized for the Altera® APEX™ 20K architecture
- OpenCore™ feature allows you to instantiate and simulate designs in the Quartus™ software before licensing

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'.

The basic block diagram of a turbo encoder/decoder is shown in Figure 1.

Figure 1. Turbo Encoder/Decoder Block Diagram



The Turbo Encoder

The turbo encoder MegaCore™ function uses a stream-driven implementation and 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 a pseudo random interleaver into encoder 2. The encoded bit streams can be punctured to save bandwidth.

Interleavers

Interleaving is the process of reordering a binary sequence in a systematic way. Convolutional codes are designed to combat random independent errors. However, 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 320 and 5,114. 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.

Functional Description

Figure 2 shows the turbo encoder interface.

Figure 2. The Turbo Encoder Interface

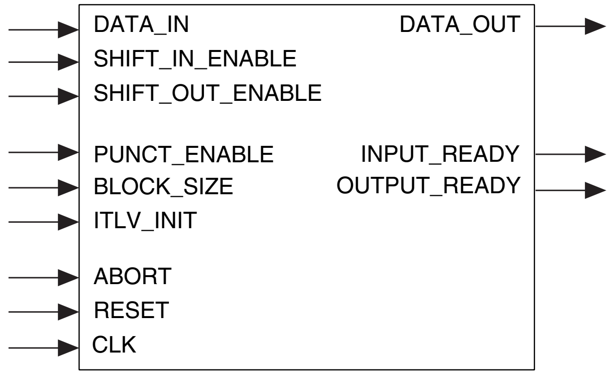


Table 1 shows the interface signal definitions.

Table 1. Interface Signals

Signal Name	Width	Description
CLK	1	Clock.
RESET	1	Asynchronous reset.
ABORT	1	Synchronous reset.
BLOCK_SIZE	13	Block size. Between 320 and 5,114
ITLV_INIT	1	Interleaver initialization. Must be asserted high for one or more clock cycles after reset, or after changing BLOCK_SIZE. *(1)*
INPUT_READY	1	Indicates that the encoder is ready to accept input data.
OUTPUT_READY	1	Indicates that the encoder is ready to output data.
SHIFT_IN_ENABLE	1	When high, DATA_IN will be shifted into the turbo encoder on the next rising clock edge.
SHIFT_OUT_ENABLE	1	When high, the next output will be made available on DATA_OUT after the next rising clock edge.
DATA_IN	1	Data input.
DATA_OUT	1	Data output.
PUNCT_ENABLE	1	Puncture rate. When 0, no puncturing, rate = 1/3; when 1, rate = 1/2.

Note:

(1) ITLV_INIT can be asserted only when INPUT_READY is high, and before data is shifted-in. If you wish to reinitialize the interleaver at any other time, ABORT must be asserted. This returns the encoder to a state where it can respond to ITLV_INIT.

Using the Turbo Encoder

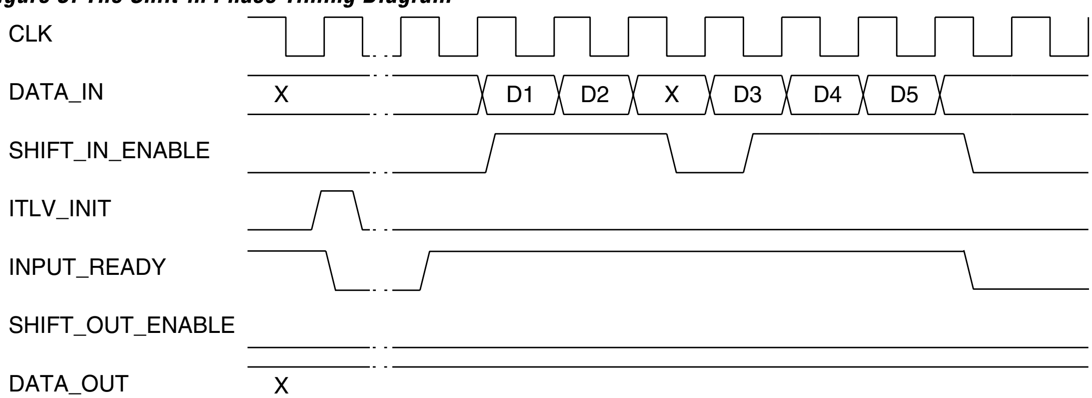
The turbo encoder is a slave device that is operated using two control input signals: SHIFT_IN_ENABLE and SHIFT_OUT_ENABLE. Being a slave device, it is up to the host to provide the control signals at the correct time with regard to the encoder's state. Two status signals are used to indicate the encoders state: INPUT_READY and OUTPUT_READY.

The process of encoding a block of data can be broken down into four phases.

The Shift-In Phase

Figure 3 shows the shift-in phase timing diagram. Shifting-in can only commence if INPUT_READY is high. ITLV_INIT is asserted high for one clock cycle to initialize the interleaver (necessary only if BLOCK_SIZE is changed and after RESET). The data at DATA_IN must be valid when SHIFT_IN_ENABLE is high, as it will be registered on the next rising clock edge. After a rising clock edge, DATA_IN may be changed without asserting SHIFT_IN_ENABLE low. 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.

Figure 3. The Shift-in Phase Timing Diagram

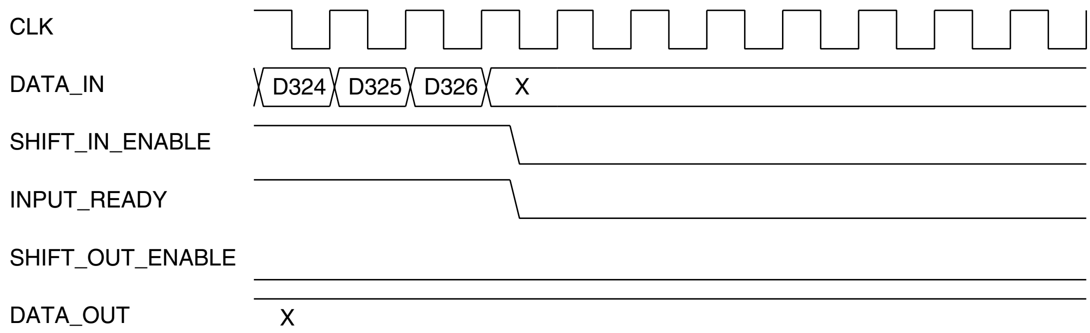


The Active Phase

Figure 4 shows the active phase timing diagram. The encoder will go into the active phase automatically when it detects that the required number of bits have been shifted in. The host should not try to shift more data in, any attempt to do so is be ignored by the encoder.

During the active phase the encoder interleaves and encodes the information, generates the tail bits, and prepares itself to produce punctured output data. The active phase lasts for more than `BLOCK_SIZE` clock cycles. When the active phase is over, the encoder will automatically enter the shift-out phase.

Figure 4. The Active Phase Timing Diagram



The Shift-Out Phase

Figure 5 shows the shift-out phase 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.

Trellis Termination of Turbo Code

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 3.1.0).* 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. The tail bits emerge from the encoder in the following sequence:

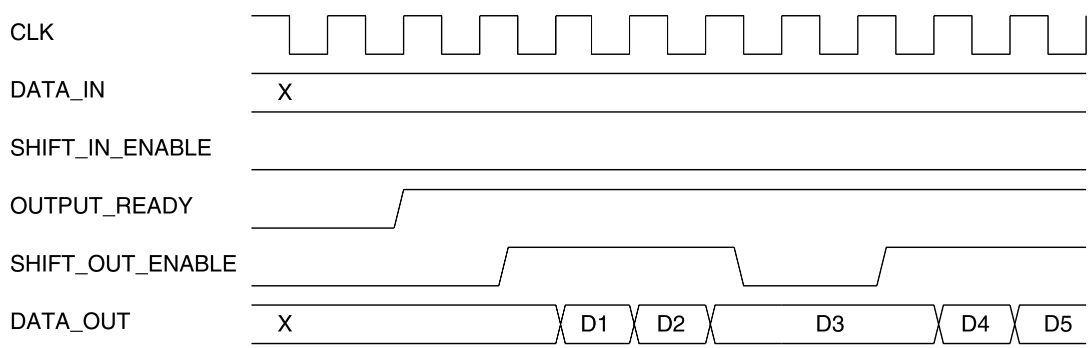
X(t) Y(t) X(t+1) Y(t+1) X(t+2) Y(t+2) X′(t) Y′(t) X′(t+1) Y′(t+1) X′(t+2) Y′(t+2) where:

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

☞ The tail is not effected by puncturing, for a given set of information input bits, i.e. it is identical for punctured and non-punctured encoded outputs.

During the normal phase, the information and parity sequences emerge from the encoder as shown in the following sequences:
X(t) Y(t) Y'(t) X(t+1) Y(t+1) Y'(t+1) ...(unpunctured);
X(t) Y(t) X(t+1) Y'(t+1) X(t+2) Y(t+2) ...(punctured).

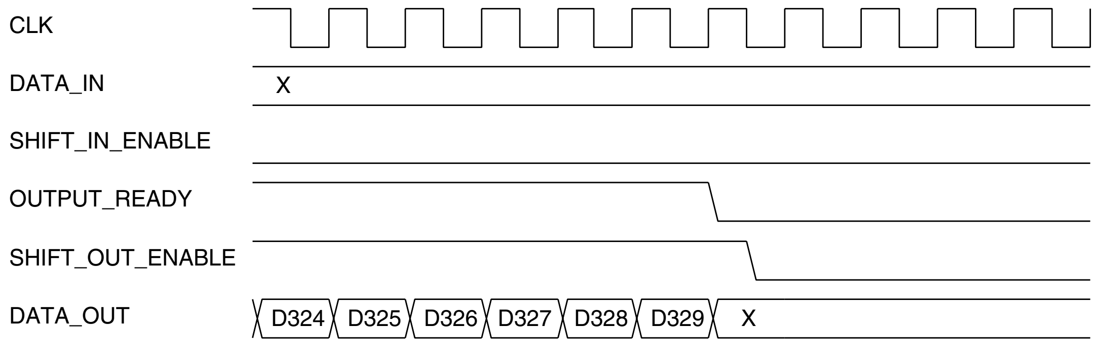
Figure 5. The Shift-Out Phase Timing Diagram



The Finished Phase

In the finished phase, the encoder prepares itself for the next block of data by clearing all registers. `OUTPUT_READY` is asserted low after the final data has been shifted out by the host system (see Figure 6). One clock cycle later the encoder will return to the shift-in phase, ready to accept the next block. `OUTPUT_READY` will remain low until the next block of data is ready to be shifted-out.

Figure 6. The Finished Phase Timing Diagram



Performance

Table 2 shows the function's performance with various devices.

Table 2. Encoder Performance

Device	Frequency (MHz)
APEX 20K100 -2	38
APEX 20K200 -2	34
APEX 20KE100 -2	50
APEX 20KE200 -2	48
APEX 20K100 -1	51
APEX 20KE100 -1	60

Size

The amount of logic needed for the turbo encoder MegaCore function is 2,892 logic elements (LEs). The function uses 10 embedded system blocks (ESBs).

Specifications—Decoder

Contents

User Guide

Notes:

Specifications—Decoder

Features

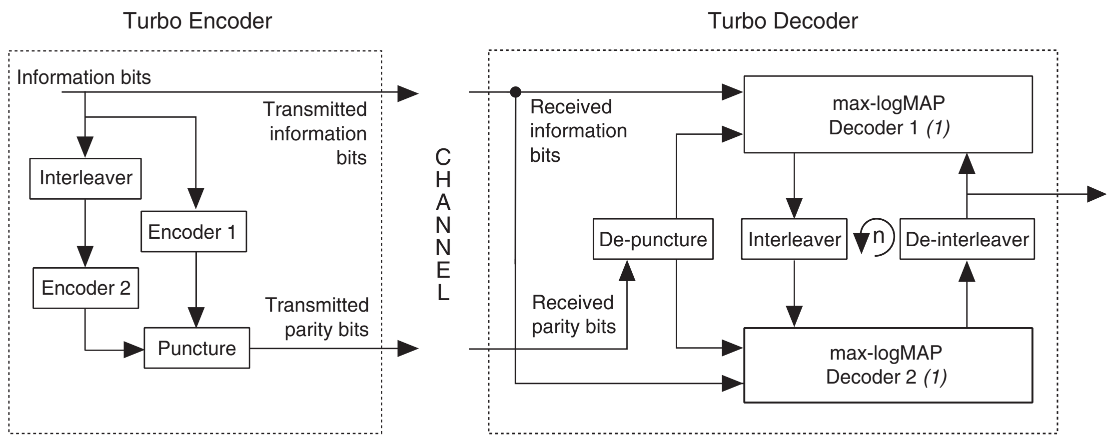
- Compliant with *3rd Generation Partnership Project* (3GPP); *Technical Specification Group Radio Access Network; Multiplexing and Channel Coding (FDD) (3G TS 25.212 version 3.1.0)*
- High-performance max-logMAP (logarithmic 'maximum a posteriori') decoder for maximum error correction
- Data rates in excess of 2 megabits per second (Mbps)
- Includes 3GPP-compliant 'mother' interleaver
- Interleaver block sizes from 320 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 throughput
- Optimized for the Altera® APEX™ 20K and APEX 20KE architectures
- MegaWizard™ Plug-In for easy parameterization

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 shows the basic block diagram of a turbo encoder/decoder.

Figure 1. Turbo Encoder/Decoder Block Diagram



Note:

(1) Although the illustration shows two max-logMAP decoders, they are physically implemented as one max-logMAP decoder.

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.

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. The encoded bit streams can be punctured to save bandwidth.

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) 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.

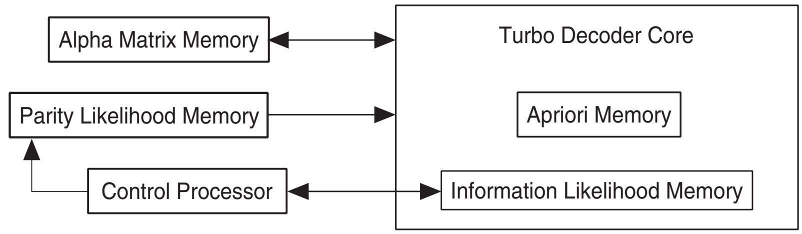
When data is exchanged between the two decoders, the soft values are reordered to match the interleaving structure. This reordering is done with the interleaver and deinterleaver between the decoders. When the second decoder has finished, the next iteration is started. Again, decoder 1 is activated using 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/decoder MegaCore™ function.

Functional Description

Figure 2 shows an example system that contains the turbo decoder MegaCore function. In this example, the processor controls the operation of the core 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

Figure 2. An Example System



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 1 shows the meanings of 3-bit values.

Table 1. The Meanings of 3-bit Values

Value	Meaning
011	Maximum likelihood of a '0'
010	Medium likelihood of a '0'
001	Low likelihood of a '0'
000	Equal probability of a '1' and a '0'
111	Low likelihood of a '1'
110	Medium likelihood of a '1'
101	Maximum likelihood of a '1'
100	Not used

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 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 apriori 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 apriori 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 core. 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.

Interface Definition

This section describes the turbo decoder MegaCore function's interfaces. Table 2 shows the function's parameters.

Table 2. Parameters

Name	Range	Description
SOFTBITS	3 to 8	The number of bits (*N*) used for the log likelihood values for information and parity bits.
TMEMACCA	2 to 6	The number of pipeline stages outside the core for read access to the alpha matrix memory.
TMEMACCP	2 to 6	The number of pipeline stages outside the core for read access to the parity likelihood memory.
BANKSWAP	0 or 1	Indicates whether two banks of information memory are to be included in the core to achieve maximum performance.

Table 3 shows the general signals.

Table 3. General Signals

Name	Type	Description
CLK	Input	System clock.
RESET	Input	Global asynchronous reset.

Table 4 shows the configuration and control interface signal definitions.

Table 4. Configuration and Control Interface Signals

Name	Type	Width	Description
ABORT	Input		Aborts any active operation and returns core to reset state.
START	Input		Instructs core to begin decoding a block of data.
LD_INT	Input		Instructs interleaver to re-initialize with new block size.
ACTIVE	Output		Indicates core is busy processing data or initializing the interleaver.
PUNCTURE	Input		When asserted, the core will set alternate Parity 1 and Parity 2 values to 0.
BLOCK_SIZE	Input	13	Block size.
ITERATIONS	Input	4	Number of iterations.

Table 5 shows the information memory interface signal definitions.

Table 5. Information Memory Interface Signals

Name	Type	Width	Description
INFO_DATA_IN	Input	*N*	Memory data input.
INFO_ADDR	Input	13	Memory address.
INFO_RD	Input		Memory read strobe.
INFO_WR	Input		Memory write strobe.
INFO_DATA_OUT	Output	*N*	Memory data output.

Table 6 shows the parity likelihood memory interface signal definitions.

Table 6. Parity Likelihood Memory Interface Signals

Name	Type	Width	Description
PARITY_DATA_IN	Input	*N*	Memory data input.
PARITY_ADDR	Output	14	Memory address.
PARITY_RD	Output		Information memory read strobe.

Table 7 shows the alpha matrix memory interface signal definitions.

Table 7. Alpha Memory Interface Signals

Name	Direction	Width	Description
ALPHA_DATA_IN	Input	$8 \times (N - 1)$	Data from memory.
ALPHA_DATA_OUT	Output	$8 \times (N - 1)$	Data to memory.
ALPHA_ADDR	Output	13	Memory address.
ALPHA_RD	Output		Memory read strobe.
ALPHA_WR	Output		Memory write strobe.

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 core by asserting the RESET signal. The RESET signal must be de-asserted synchronously with respect to CLK to avoid metastability issues.

2. Configure the core 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 3).

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

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

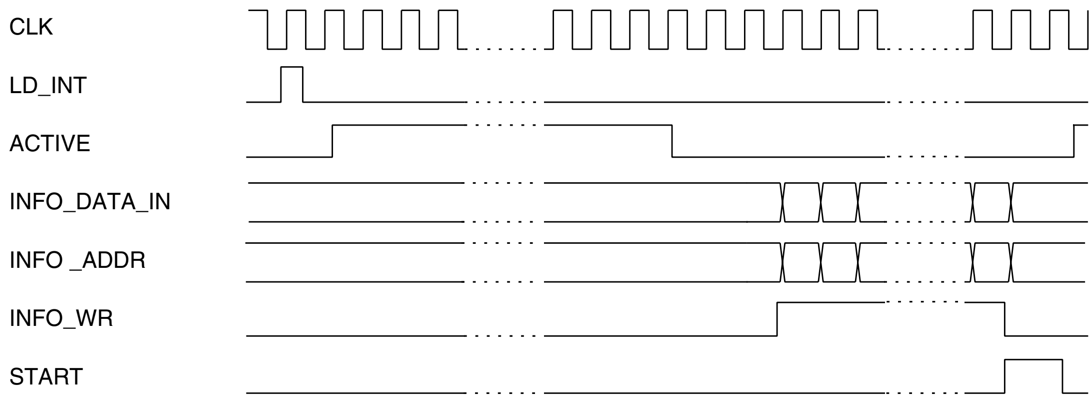
6. Wait until ACTIVE goes inactive.

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 shows the turbo decoder timing diagram.

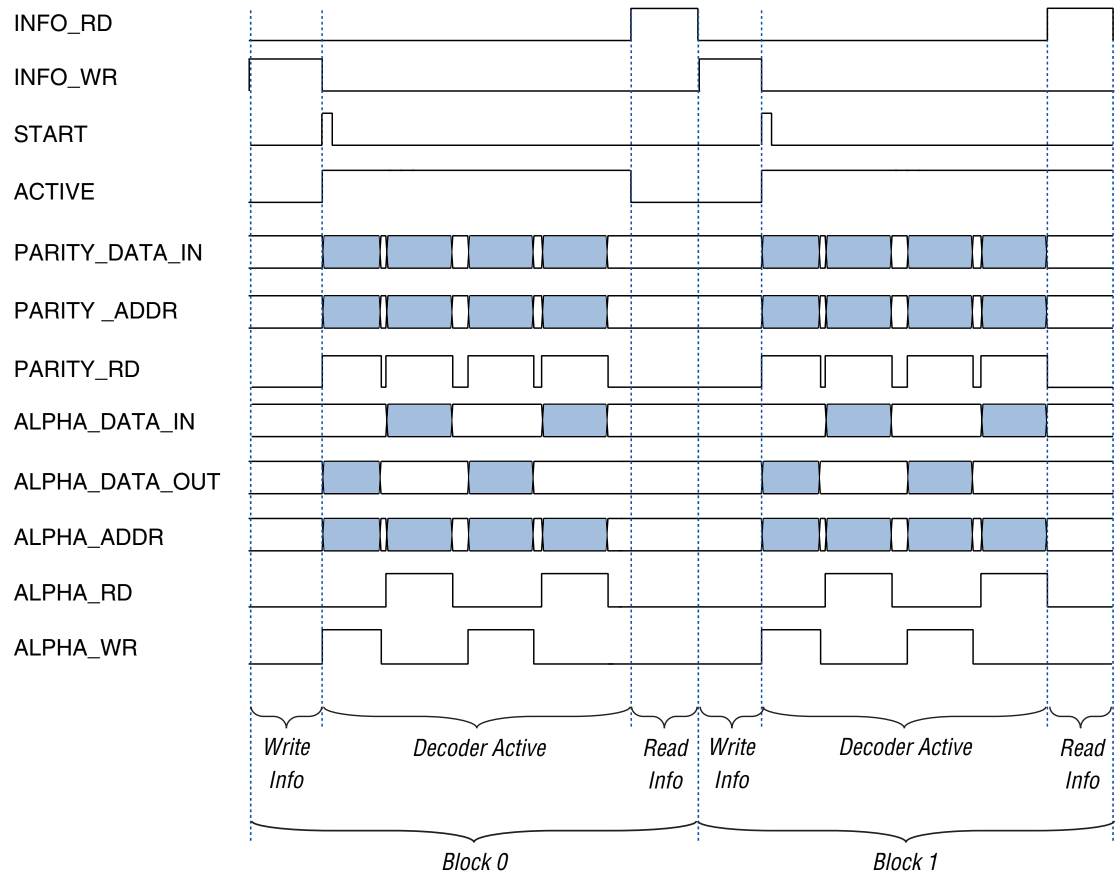
Figure 3. Initializing the Turbo Decoder (1)



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.

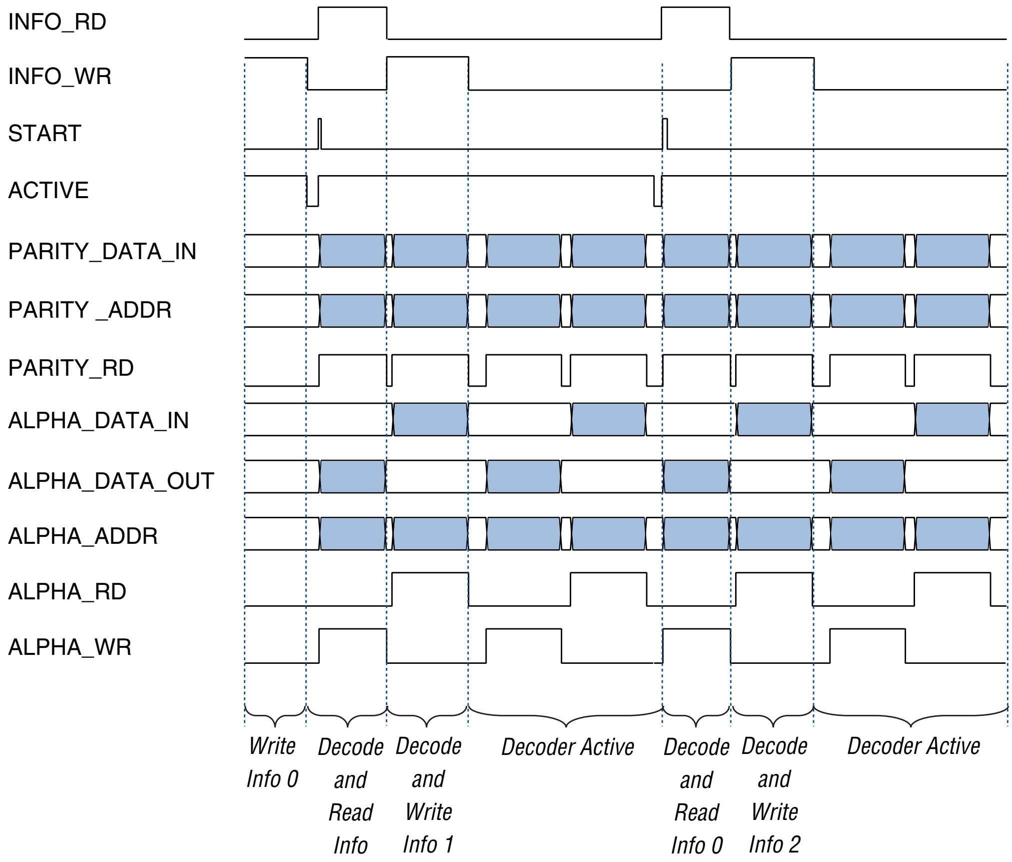
Figure 4. Turbo Decoder Timing Diagram—without Bank Swapping (1)



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.

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 core is decoding the current block (see Figure 5). If you use bank swapping, implement two banks of parity likelihood memory outside the core.

Figure 5. Turbo Decoder Timing Diagram—with Bank Swapping (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.

Performance

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. The maximum clock frequency of the turbo decoder is about 50 MHz, but this depends on the parameters selected. The performance of the turbo decoder is determined by the number of iterations required.

For example, with five iterations, 20 clock cycles will be required per sample. Allowing an overhead of 5 cycles per sample gives 25 cycles per bit. At a clock rate of 50 MHz, that provides a bit rate of 2 Mbps.

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

Size

The amount of logic needed for the turbo decoder MegaCore function does not vary greatly; about 5,000 to 7,500 logic elements (LEs), which fits on an Altera EP20K200 device. However, the memory configuration influences the chosen device. The internal RAM takes up four embedded system blocks (ESBs). The size of the other memories varies with the number of bits used to represent the soft decision (logarithmic likelihood) values. Table 8 defines the various memory sizes, where N is the number of bits used to represent the logarithmic likelihood values:

Table 8. Memory Sizes

Memory Name	Size	Size with Bank Swapping
Information Likelihood	$5K \times N$	$2 \times 5K \times N$
Parity Likelihood	$1K \times N$	$2 \times 10K \times N$
Apriori	$5K \times N$	$5K \times N$
Alpha Matrix	$5K \times 8(N - 1)$	$5K \times 8(N - 1)$

The information likelihood and apriori memories must be on-chip and are included with the turbo decoder. The parity likelihood and alpha matrix memories are outside the core, 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, by avoiding the need to read the results and write the new data between blocks.

Typical Configurations

A typical configuration uses 5 bits to represent the soft decision values. The parity memory is on-chip, and the alpha matrix memory off-chip. The information likelihood and apriori memories are 5K × 5, and occupy 13 ESBs each. The parity memory occupies 26 ESBs. The total on-chip memory requirement is 58 ESBs, and an Altera EP20K300E is a suitable device. If the parity memory is implemented off-chip, the total ESB count is 32 ESBs, and the decoder would fit in an Altera EP20K200E device. Table 9 indicates further example configurations.

Table 9. Configuration Examples

SOFTBITS	BANKSWAP	Alpha Memory	Parity Memory	ESB Count	Suitable Device
3	0	On-chip	On-chip	75	EP20K400
3	1	On-chip	On-chip	98	EP20K400
4	0	Off-chip	On-chip	44	EP20K200
4	1	Off-chip	On-chip	74	EP20K400
5	0	Off-chip	Off-chip	30	EP20K200
5	1	Off-chip	Off-chip	43	EP20K200
5	0	Off-chip	On-chip	55	EP20K300E
5	1	Off-chip	On-chip	93	EP20K400
5	0	On-chip	On-chip	135	EP20K600E
8	0	Off-chip	Off-chip	44	EP20K200
8	1	Off-chip	On-chip	64	EP20K300E
8	0	Off-chip	Off-chip	84	EP20K400
8	1	Off-chip	On-chip	144	EP20K600E

Notes:

Getting Started

Contents

User Guide

Notes:

Getting Started

User Guide

This section describes how to obtain the turbo encoder/decoder MegaCore™ function, explains how to install it on your PC, and walks you through the process of implementing the function in a design. You can test-drive MegaCore functions using the Altera® OpenCore™ feature to simulate the functions within your custom logic. When you are ready to generate programming or configuration files, you should license the function through the Altera web site or through your local Altera sales representative. The functions are optimized for Altera APEX™ 20K and APEX 20KE devices, greatly enhancing your productivity by allowing you to focus efforts on the custom logic in the system.

This walk-through involves the following steps:

1. Downloading and installing the turbo encoder/decoder MegaCore function.
2. Generating a custom MegaCore function.
3. Implementing your system using VHDL, or Verilog HDL.
4. Compiling your design.
5. Licensing the turbo encoder/decoder MegaCore function and configuring the devices.

The instructions assume that:

- You are using a PC.
- You are familiar with the Quartus™ software.
- The Quartus software version 2000.02 (or higher) is installed in the default location.
- You are using the OpenCore feature to test-drive the turbo encoder/decoder MegaCore function or you have licensed the function.

Downloading & Installing the Function

Before you can start using Altera MegaCore functions, you must obtain the MegaCore files and install them on your PC. The following instructions describe this process.

Obtaining MegaCore Functions

If you have Internet access, you can download the turbo encoder/decoder MegaCore function from the Altera web site at **http://www.altera.com**. Follow the instructions below to obtain the turbo encoder/decoder MegaCore function via the Internet. If you do not have Internet access, you can obtain the turbo encoder/decoder MegaCore function from your local Altera representative.

1. Point your web browser at **http://www.altera.com/IPmegastore**.
2. In the IP MegaSearch **keyword** field type `turbo`.
3. Click the appropriate link for your desired MegaCore function.
4. Click the link for the download icon.
5. Follow the online instructions to download the function and save it to your hard disk.

Installing the MegaCore Files

For Windows 95/98 and Windows NT 4.0, follow the instructions below:

1. Click **Run** (Start menu).
2. Type *<path name>**<filename>*, where *<path name>* is the location of the downloaded MegaCore function and *<filename>* is the filename of the function. Click **OK**.
3. The **MegaCore Installer** dialog box appears. Follow the online instructions to finish installation.
4. After you have finished installing the MegaCore files, you must specify the MegaCore function's library folder (**\turbo_codec\lib**) as a user library in the Quartus and MAX+PLUS II software. Search for "User Libraries" in Quartus Help for instructions on how to add a library.

MegaCore Folder Structure

Altera MegaCore function files are organized into several folders; the top-level directory is **\turbo_codec** (see Table 1).

The MegaCore folder structure may contain several MegaCore products. Additionally, Altera updates MegaCore files from time-to-time. Therefore, Altera recommends that you do not save your project-specific files in the MegaCore folder structure.

Table 1. Turbo MegaCore Folders

Folder	Description
\turbo_codec	Contains all files for the turbo encoder/decoder MegaCore function.
\doc	Contains documentation for the function.
\lib	Contains encrypted lower-level design files. After installing the MegaCore function, you should set a user library in the Quartus or MAX+PLUS II software that points to this folder. This library allows you to access all the necessary MegaCore files.
\c_model	Contains the file to run the C-model, and the files for the bit error rate (BER) graphs.
\sim_lib\sim_model	Contains the **vhdl\modelsim** folder
\vhdl\modelsim	Contains precomplied libraries for the Model*Sim* simulator.
\sim_lib\reference_design	Contains the reference design file.
\sim_lib\testbench	Contains the files for the testbench.

Generating a Custom Turbo Function

This section describes the design flow using the Altera turbo encoder/decoder MegaCore function and the Quartus development system. Altera provides a MegaWizard™ Plug-In with the turbo encoder/decoder MegaCore function. The MegaWizard Plug-In Manager, which you can use within the Quartus software or as a stand-alone application, lets you create or modify design files to meet the needs of your application. You can then instantiate the custom megafunction in your design file.

You can use the Altera OpenCore feature to compile and simulate the MegaCore functions in the Quartus software, allowing you to evaluate the functions before deciding to license them.

To create a custom version of the turbo encoder/decoder MegaCore function, follow these steps:

☞ Before you can use the MegaWizard™ Plug-In, your PC must have Java Run Time Environment version 1.2.2 installed. This file can be obtained from the Java web site at **http://java.sun.com/products/jdk/1.2/jre/**

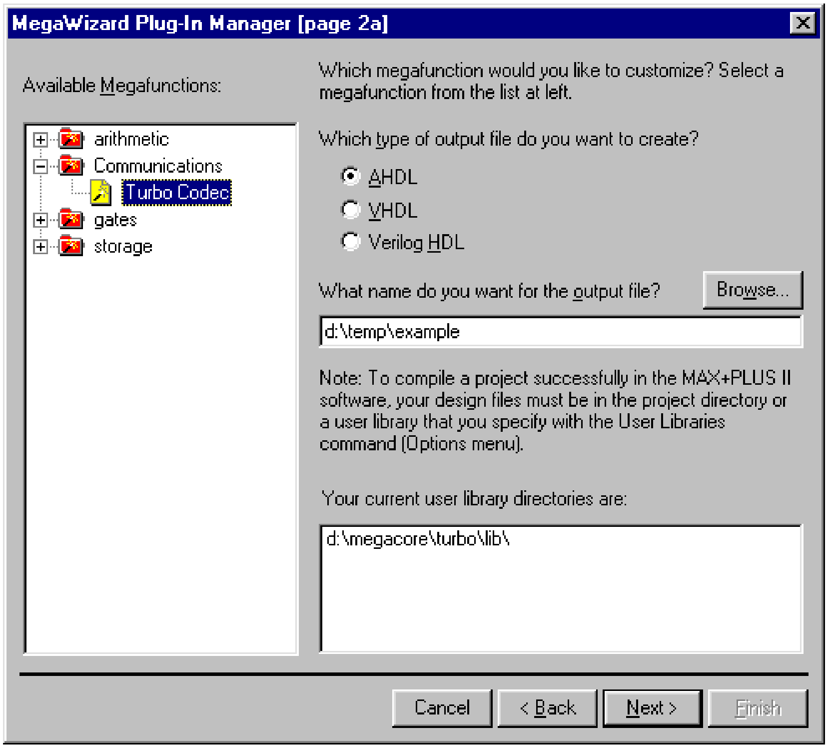
1. Start the MegaWizard Plug-In Manager by choosing the **MegaWizard Plug-In Manager** command (Tools menu in the Quartus software). The **MegaWizard Plug-In Manager** dialog box is displayed.

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

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

3. Select **Turbo Codec** in the **Communications** folder (see Figure 1), and click **Next**.

Figure 1. Selecting the Megafunction



4. Select either decoder or encoder (see Figures 2 and 3). If you select the decoder, select your required parameters. Click **Next.**

3 Getting Started

Figure 2. Selecting the Encoder

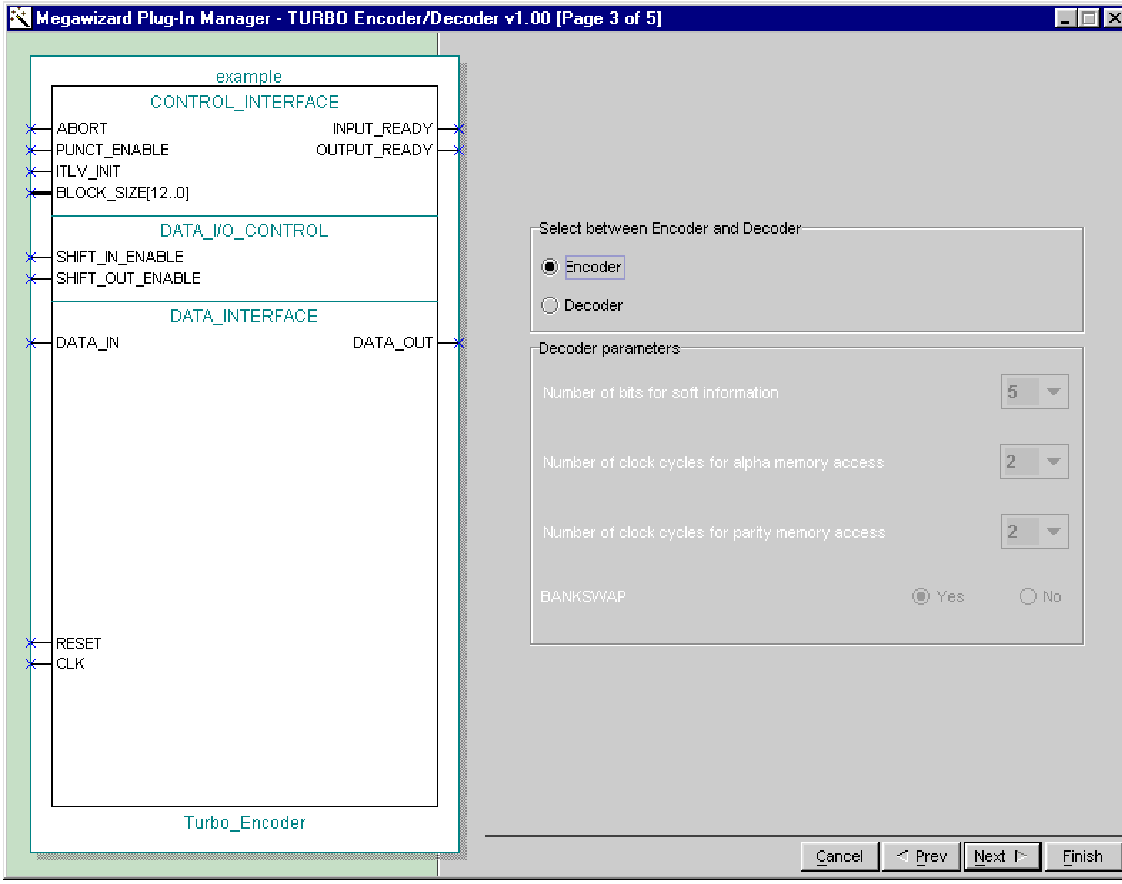
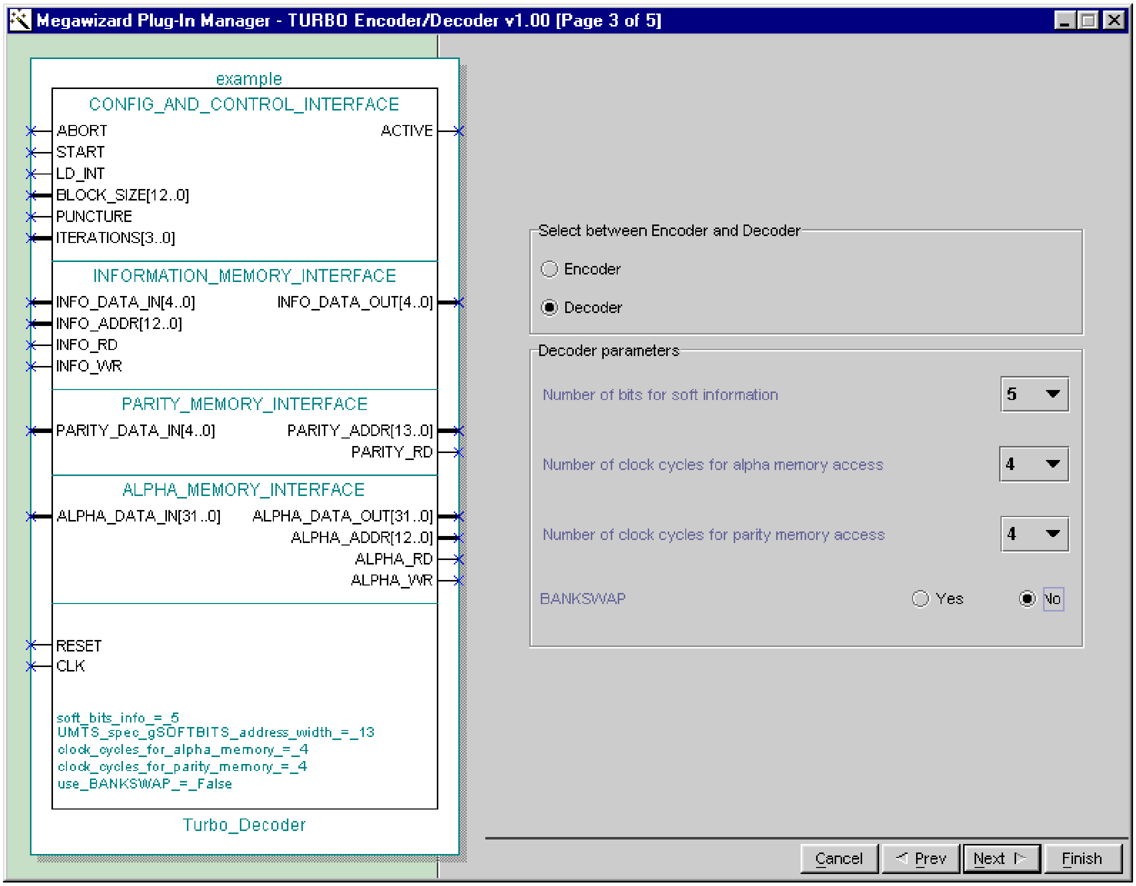
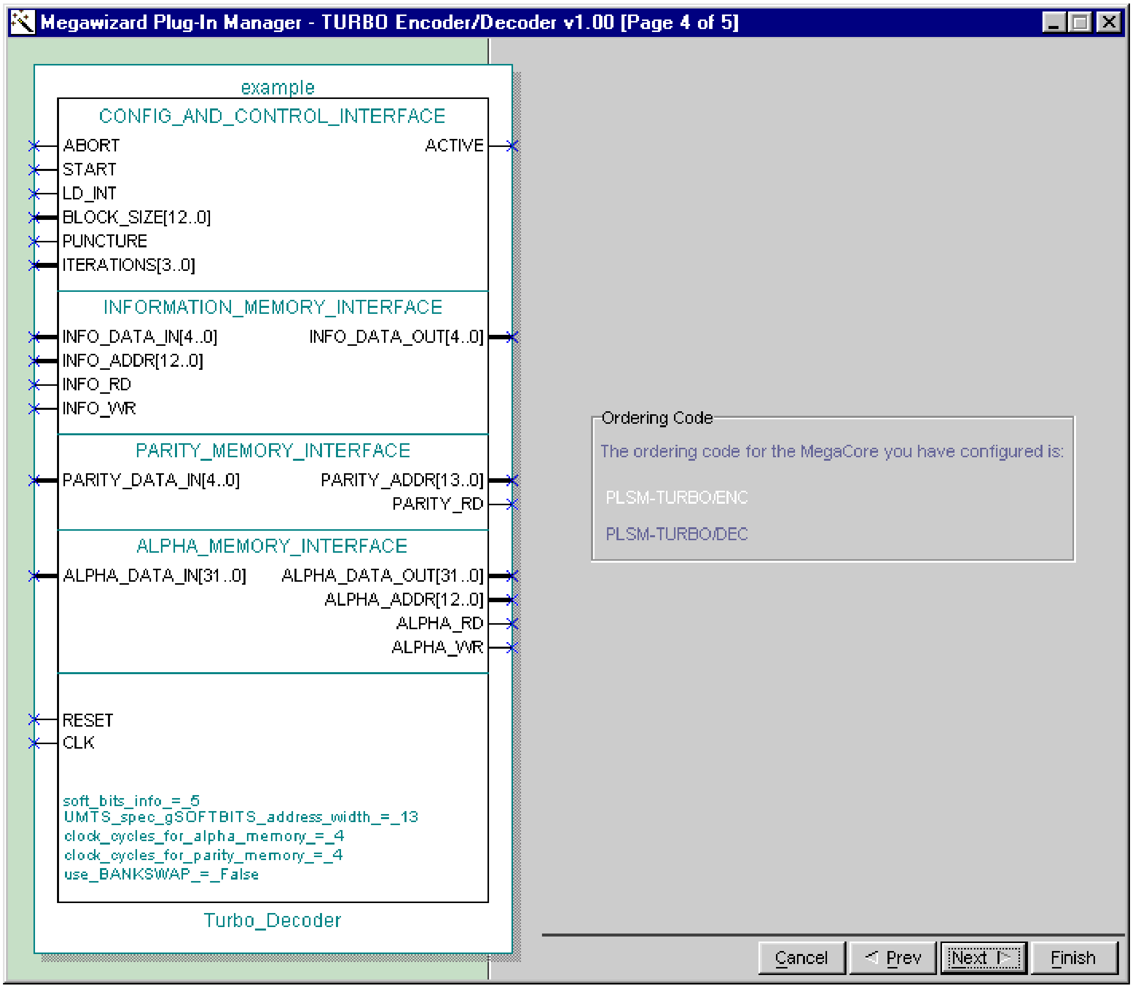


Figure 3. Selecting the Decoder



5. The MegaWizard Plug-In lists the product order code for your custom megafunction (see Figure 4). You will need this code when you want to license the MegaCore function. Click **Next**.

Figure 4. Order Codes



6. The final screen lists the design files that the wizard creates. Click **Finish**.

Once you have created a custom megafunction, you can integrate it into your system design and compile.

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 bit error rate (BER) and the frame error rate. To use the C-model follow the steps below.

1. Open the MS-DOS Command Prompt software. Change the folder to <*pathname*>**\turbo_codec\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>*↵

   e.g.
   `ber_simulator -c 0 -n 1 -i 5 -f 100 -l 320 -w 5`↵

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

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:

   <number of errors> | *<number of bits transmitted>* |
   <BER> | *<Frame Error Rate>*

Table 2. C-Model Valid Parameter Values

Parameter	Valid Values
Channel	0 or 1 *(1)*
Signal to noise ratio	0 to 20
Number of iterations	1 to 16
Number of soft bits	1 to 16
Block size	320 to 5,114
Number of frames	Any positive integer > 20

Note:
(1) Channel value 0 selects 'additive white Gaussian noise'; 1 selects Rayleigh Fading.

BER Graphs

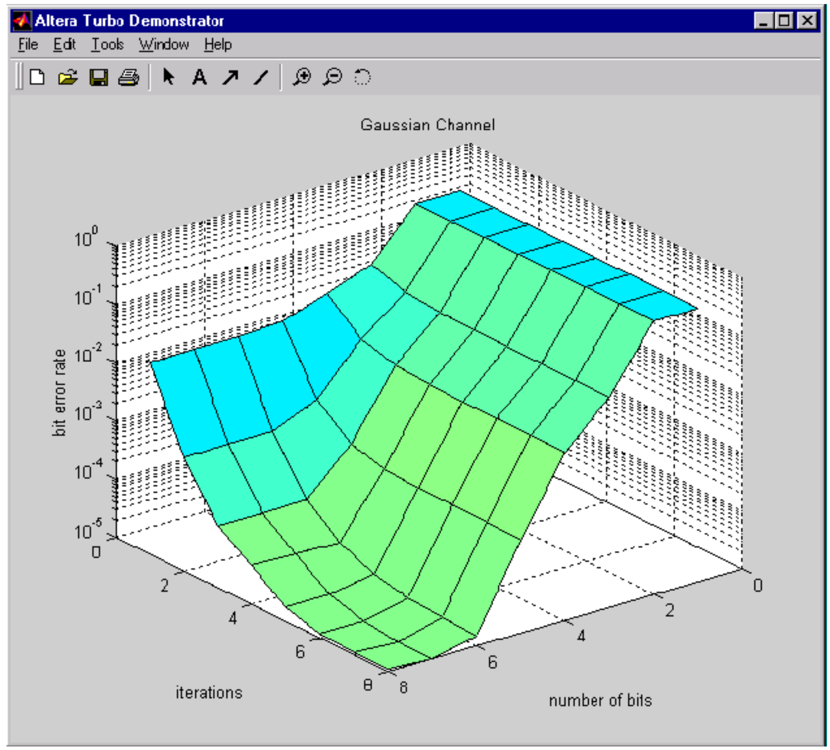
Altera have used this C-model for all combinations of parameters and collated the BERs they produce. These results have been entered into a database. You can access this database using MATLAB. The following steps explain how to show the turbo decoder BER graphs in MATLAB.

For more information on MATLAB, refer to the Math Works website at **http://www.mathworks.co**m.

1. Open the MATLAB software. At the command prompt change the folder to the **\turbo_codec\c_model** folder.

2. At the command prompt type `BER_graphs`. Select your chosen axis variables is the drop-down menus, change the values with the slider bars, and click **Update** to view the BER graph. Figure 5 shows an example.

Figure 5. BER Graph



The results for this model are not accurate for low BERs, as only one million samples were 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

Using the VHDL Model

The turbo decoder MegaCore function is supplied with a VHDL model, a reference design (see Figure 6), and a system testbench (see Figure 7). You can use the reference design to simulate the functionality of the turbo function in your system. The reference design is:

- supplied as source code, instantiating decoder from precompiled libraries;
- synthesizable;
- intended to be used as a basis for your design.

Figure 6. Decoder Reference Design Block Diagram

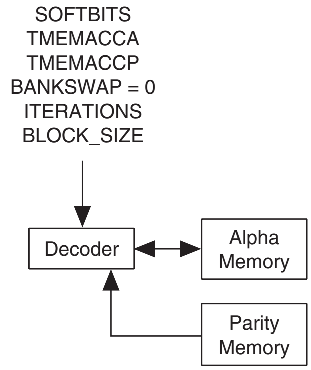
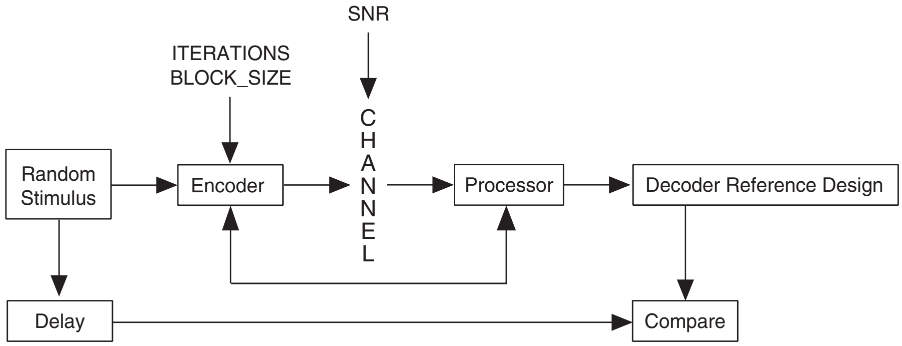


Figure 7. System Testbench Block Diagram



The following instructions describe how to set up your system and how to use the system testbench in the Model*Sim* software.

Setting Up Your System

Four precompiled Model*Sim* libraries are provided with the turbo encoder/decoder MegaCore function:
random, **turbolib**, **memlib**, and **itlvlib**.

These are installed in the folder
turbo_codec\sim_lib\sim_model\vhdl\ModelSim. Follow the steps below to set up logical maps to these libraries.

1. Run the Model*Sim* software and create a logical map called **random** to the folder containing the precompiled library by typing the following command in the Model*Sim* software.

   ```
   vmap random <Drive:>/<Turbo MegaCore Path>
       /sim_lib/sim_model/vhdl/ModelSim/random↵
   ```

☞ You can also use the Model*Sim* graphical user interface to create the logical map. Refer to the Model*Sim* online help for details.

2. Repeat step 1 for the other libraries but replace every instance of **random** with **turbolib**, then **memlib**, and finally **itlvlib**.

Using the VHDL Testbench

Altera provides a reference design and a system testbench. You must compile these files, before simulating with the Model*Sim* software, by performing the following steps.

1. Select **Simulation** (Options menu). Under the **Suppress Warnings** section in the **Simulation Options** dialog box, check **From Synopsis Packages** and **From IEEE Numeric Std Packages**. Click **OK**

2. Choose **Compile** (Design menu).

3. In the **Compile HDL Source Files** dialog box (see Figure 8), click **Default Options**. The **Compiler Options** dialog box appears (see Figure 9).

Figure 8. Compile HDL Source Files Dialog Box

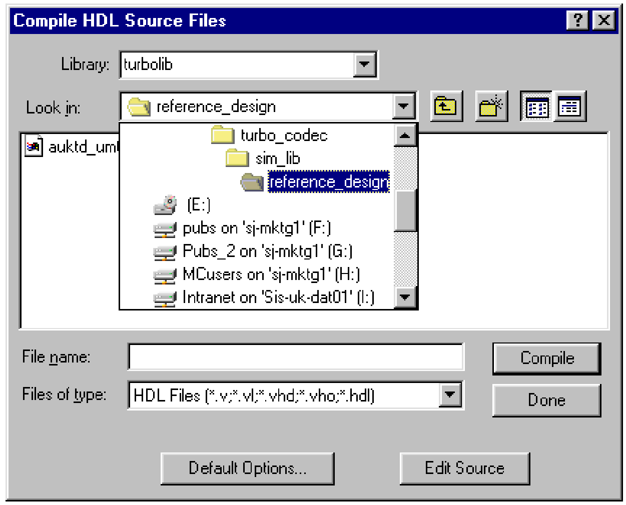
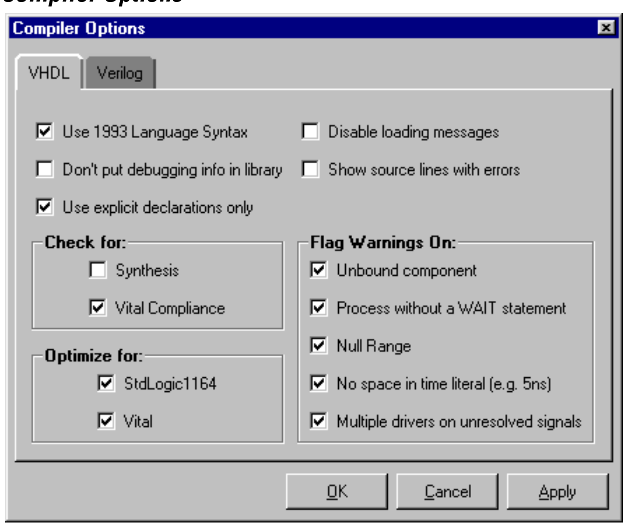


Figure 9. Compiler Options



4. In the **Compiler Options** dialog box, check the **Use 1993 Language Syntax** option and the **Use explicit declarations only** option in the **VHDL** tab. Click **OK**.

5. In the **Compile HDL Source Files** dialog box, select **turbolib** in the **Library** drop-down list. Also select the **turbo_codec\sim_lib\reference_design** folder in the **Look In** drop-down list box.

3 Getting Started

6. Select **auktd_umts_turbo_decoder_chip.vhd** and click **Compile**.

7. Select the **turbo_codec\sim_lib\testbench** folder in the **Look In** drop-down list box.

8. Select **AWGN_BPSK_channel.vhd** and click **Compile**. Repeat for **auktd_tdec_refdesign_ctrl.vhd**, **aukte_tenc_refdesign_ctrl**, and **auktde_turbo_tb**. You must compile **auktde_turbo_tb.vhd** last.

9. Once compilation finishes, click **Done**.

The configuration is included in the file **auktde_turbo_tb.vhd**; the VHDL configuration is defined with the prefix **cfg_**.

The configuration must be loaded specifying the parameter values. See Table 1 on page 11, Table 2 on page 24 and Table 4 on page 25 for a detailed description of the parameters. The `clock_period` parameter can be any valid time period (for example, 30 ns); the `signal2noise_ratio` parameter can be set between 3.0 and –0.5 (dB); the `seed` parameter can be set between 0 and 4,095.

☞ The `seed` parameter selects which random number sequence you require, i.e. there are 4,096 different random number sequences.

You can quickly load the configuration from the command line in the Model*Sim* software. For example:

```
vsim -GgTMEMACCA=2 -GgTMEMACCP=2
    {-Gclock_period=20 ns} -Ggiterations=3
    -Gsignal2noise_ratio=1.0 -Ggblock_size=320
    -GgSOFTBITS=4 -Gseed=100
    TurboLib.cfg_auktde_turbo_tb↵
```

☞ You can also use the Model*Sim* graphical user interface to load the configuration. Refer to the Model*Sim* online help for details.

Altera provide a **.do** file in the **turbo_codec\sim_lib\sim_model\vhdl\ModelSim** folder. This sets-up a timing diagram display window that shows the relevant signals for the testbench. To use the **.do** file, type the following commands.

```
do <Drive:>/<Turbo MegaCore Path>
    sim_lib/sim_model/vhdl/ModelSim/
    testbench_wave_template.do↵
    run 300 us↵
```

Performing Synthesis, Compilation & Post-Routing Simulation

The Quartus software works seamlessly with tools from all EDA vendors, including Cadence, Exemplar Logic, Mentor Graphics, Synopsys, Synplicity, and Viewlogic. After you have licensed the MegaCore function, you can generate EDIF, VHDL, Verilog HDL, and Standard Delay Output Files from the Quartus software and use them with your existing EDA tools to perform functional modeling and post-route simulation of your design.

The following sections describe the design flow to compile and simulate your custom MegaCore design with a third-party EDA tool. To synthesize your design in a third-party EDA tool and perform post-route simulation, perform the following steps:

1. Create your custom design instantiating a turbo encoder/decoder MegaCore function.

2. Synthesize the design using your third-party EDA tool. Your EDA tool should treat the MegaCore instantiation as a black box by either setting attributes or ignoring the instantiation.

3. After compilation, generate a hierarchical netlist file in your third-party EDA tool.

4. Open your netlist file in the Quartus software.

   ☞ Before you compile, you must select Leonardo Spectrum as the design entry tool. To do this, select **EDA Tool Settings** (**Project** Menu) and under the Design entry/synthesis tool drop-down box, select Leonardo Spectrum.

5. Select **Compile mode** (Processing menu).

6. Specify the compiler settings in the **Compiler Settings** dialog box (Processing menu) or use the Compiler Settings wizard.

7. Specify the user libraries for the project and the order in which the Compiler searches the libraries.

8. Specify the input settings for the project under the **EDA Tool Settings** (Project menu).

9. Depending on the type of output file you want, specify Verilog HDL output settings or VHDL output settings in the **General Settings** dialog box (Project menu).

10. Compile your design. The Quartus Compiler synthesizes and performs place-and-route on your design, and generates output and programming files.

11. Import your Quartus-generated output files (**.edo**, **.vho**, **.vo**, or **.sdo**) into your third-party EDA tool for post-route, device-level, and system-level simulation.

12. Turn on the **Verilog Netlist Writer** or **VHDL Netlist Writer** command (Interfaces menu), depending on the type of output file you want to use in your third-party simulator.

13. Compile your design. The Quartus Compiler synthesizes and performs place-and-route on your design, and generates output and programming files.

14. Import your Quartus-generated output files (**.edo**, **.vho**, **.vo**, or **.sdo**) into your third-party EDA tool for post-route, device-level, and system-level simulation.

Configuring a Device

After you have compiled and analyzed your design, you are ready to configure your targeted Altera device. If you are evaluating the MegaCore function with the OpenCore feature, you must license the function before you can generate configuration files.