

The Altera® 3GPP UMTS Turbo Reference Design demonstrates using Turbo codes for encoding with trellis termination support, and forward error correction (FEC) in a 3GPP universal mobile telecommunications system (UMTS) design suitable for channel card or baseband modem applications (reference 1).

Turbo codes were first proposed by Berrou (and others) in 1993 (reference 2). Since its introduction, turbo code has become the coding technique of choice in many communication and storage systems due to its near Shannon limit error correction capability. These applications include, 3GPP, consultative committee for space application (CCSDS) telemetry channel coding, worldwide interoperability for microwave access (WiMAX), and 3GPP UMTS, which require throughputs in the range from two to several hundred Mbps.

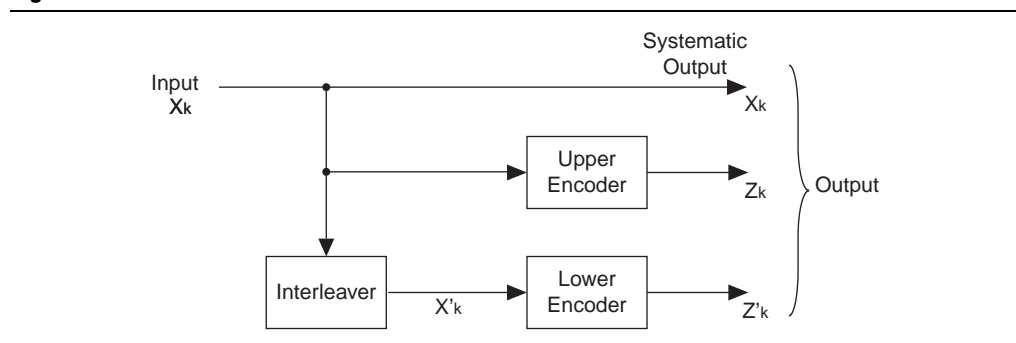
Turbo Encoder

The 3GPP UMTS Turbo encoding specified in the 3GPP UMTS specification uses parallel concatenated convolutional code (PCCC). An information sequence is encoded by a convolutional encoder, and an interleaved version of the information sequence is encoded by another convolutional encoder.

Turbo Encoder Architecture

The Turbo encoder is implemented with two 8-state constituent encoders and one Turbo code internal interleaver (Figure 1).

Figure 1. Turbo Encoder Architecture



The Turbo encoder supports the following features:

- 3GPP UMTS compliant with support for block sizes from 40 to 5,114.
- All 3GPP UMTS interleaver block sizes are selectable at run time.
- Code rate 1/3 only. Other code rates can be achieved by external rate matching.
- Double-buffering allows the encoder to receive data while processing the previous data block.
- C/MATLAB bit-accurate models for RTL test vector generation.

- You can generate VHDL or Verilog HDL testbenches using the MegaWizard® Plug-In Manager.
- Avalon® Streaming (Avalon-ST) interface.
- OpenCore Plus evaluation license.

Transfer Function

The transfer function of the 8-state constituent code for PCCC is:

$$G(D) = \left[1, \frac{g_1(D)}{g_0(D)} \right]$$

where $g_0(D) = 1 + D^2 + D^3$ and $g_1(D) = 1 + D + D^3$.

The initial values of the shift registers of the 8-state constituent encoders are all zeros when starting to encode the input bits.

The output from the turbo coder is:

$$X_0, Z_0, Z'_0, X_1, Z_1, Z'_1, \dots, X_{K-1}, Z_{K-1}, Z'_{K-1}$$

Where:

- Bits X_0, X_1, \dots, X_{K-1} are input to both the first 8-state constituent encoder and the internal interleaver (K is the number of bits).
- Bits Z_0, Z_1, \dots, Z_{K-1} and $Z'_0, Z'_1, \dots, Z'_{K-1}$ are output from the first and second 8-state constituent encoders.
- The bits output from the internal interleaver (and input to the second 8-state constituent encoder) are $X'_0, X'_1, \dots, X'_{K-1}$.

Trellis Termination

Figure 2 on page 3 shows the structure of a rate 1/3 Turbo encoder with trellis termination (shown by the dotted lines).

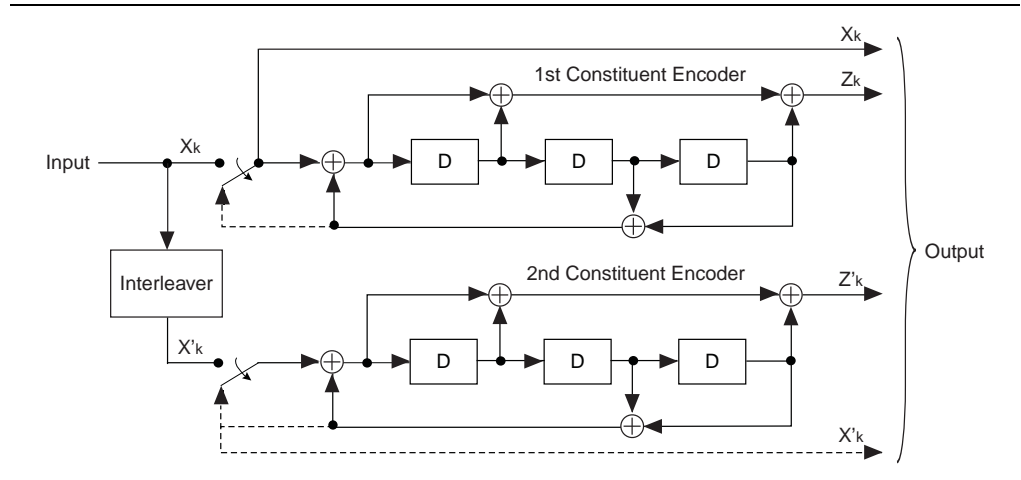
Trellis termination is performed by taking the tail bits from the shift register feedback after all information bits are encoded. The tail bits are padded after the encoding of information bits.

The first three tail bits terminate the first constituent encoder (upper switch of Figure 2 in lower position) while the second constituent encoder is disabled. The last three tail bits terminate the second constituent encoder (lower switch of Figure 2 in lower position) while the first constituent encoder is disabled.

The transmitted bits for trellis termination are then:

$$X_K, Z_K, X_{K+1}, Z_{K+1}, X_{K+2}, Z_{K+2}, X'_K, Z'_K, X'_{K+1}, Z'_{K+1}, X'_{K+2}, Z'_{K+2}$$

Figure 2. Structure of a Rate 1/3 Turbo Encoder



Internal Interleaver

The bits input to the Turbo code internal interleaver are denoted by X_0, X_1, \dots, X_{K-1} where K is the number of input bits. The bits output from the Turbo code internal interleaver are denoted by $X'_0, X'_1, \dots, X'_{K-1}$.

The relationship between the input and output bits is specified in the *3GPP Technical Specification* (reference 1).

Double-Buffering

The data path is double-buffered to allow a new data block to be shifted in while encoding the previous block. This technique reduces the delay in I/O operation, makes use of the hardware as much as possible and improves the overall throughput.

Input Data Format

The required input data ordering for a block of size K is:

$$X_0, X_1, X_2, \dots, X_{K-1}$$

Output Data Format

The output data is three bits wide. [Table 2](#) shows the ordering for a block of size K .

Table 1. Turbo Encoder Output Data Ordering (Part 1 of 2)

Output Data	source_data		
	2	1	0
0	Z'_0	Z_0	X_0
1	Z'_1	Z_1	X_1
...
$K - 1$	Z'_{K-1}	Z_{K-1}	X_{K-1}
K	X_{K+1}	Z_K	X_K
$K + 1$	Z_{K+2}	X_{K+2}	Z_{K+1}

Table 1. Turbo Encoder Output Data Ordering (Part 2 of 2)

Output Data	source_data		
	2	1	0
$K + 2$	X'_{K+1}	Z'_K	X'_K
$K + 3$	Z'_{K+2}	X'_{K+2}	Z'_{K+1}

Latency Calculation

The encoding delay D is the number of clock cycles consumed to encode an entire block of data. If K is the block size, $D = K + 14$.

The encoding delay does not include the loading delay, which requires the same number of clock cycles as the block size K to load the input data to the input buffer.

For example:

- When $K = 5,114$, $D = 5,114 + 14 = 5,128$
- When $K = 40$, $D = 40 + 14 = 54$

Then the encoding latency (the time taken by the encoder to encode an entire block) can be calculated using the following formula:

$$L = \frac{D}{f_{MAX}} \mu\text{s}$$

Where f_{MAX} is the system clock speed.

For the above examples, $L = 0.21 \mu\text{s}$ and $25.13 \mu\text{s}$ respectively for $f_{MAX} = 245 \text{ MHz}$.

Throughput Calculation

The throughput can be calculated using the following formula:

$$T = \frac{K \times f_{MAX}}{D} \text{ Mbps}$$

For the examples in the previous section, $T = 181.48 \text{ Mbps}$ and 244.33 Mbps respectively.

Test Vector Generation

The following files are needed to perform RTL simulation:

- `ctc_encoder_input.txt`
- `ctc_encoder_input_info.txt`

One test case is provided in `<Turbo install path>/turbo_for_umts/lib/test_files`.

You can use the following procedures to generate your own test vectors using the provided system:

1. Start MATLAB (version 2007b or later)
2. Change the working directory to `<Turbo install path>/turbo_for_umts/cml`.
3. Run the following command:

```
cml_startup
```

4. Make a **test** subdirectory by running the following command:

```
mkdir ../test
```

5. Run the following command:

```
[sim_param, sim_state] = CmlSimulate('Scenarios_UMTS_ENCODER_RTL',
                                     [832 832 832 40 48 56 5114]);
```

This command generates test vectors for feeding the encoder with inputs of block size 832 three times and then 40, 48, 56 and 5,114 as the last block. This matrix can be modified to fit your test needs.

If this command is run successfully, some files are created in your test vector directory `<Turbo install path>/turbo_for_umts/test`. You can modify the parameter `dump_dir` in the `Scenarios_UMTS_ENCODER_RTL.m` file to change this location. Copy these files to your Quartus II project directory (created through the MegaWizard Plug-In Manager).

The software simulation model generates the following three files:

- `ctc_encoder_input.txt`
- `ctc_encoder_input_info.txt`
- `ctc_encoder_output_gold.txt`

After RTL simulation, another file `ctc_encoder_output.txt` is created which should match the contents of `ctc_encoder_output_gold.txt`.

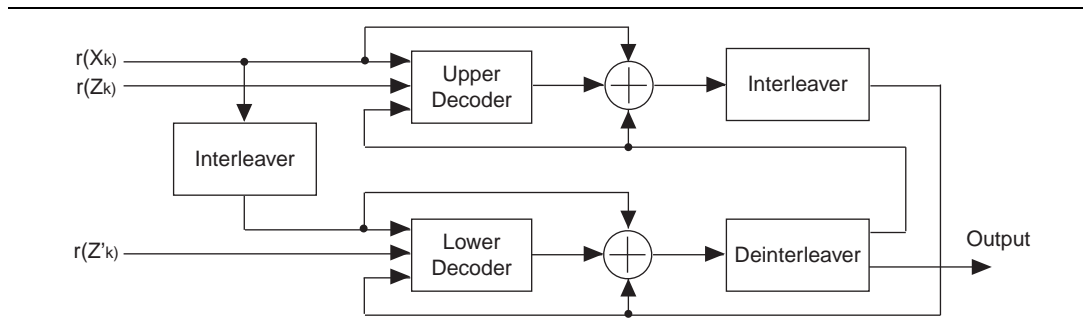


For information on how to start RTL simulation, refer to “[Simulate the Design](#)” on page 18.

Turbo Decoder

Figure 3 shows the structure of the Turbo decoder.

Figure 3. Turbo Decoder Architecture



A Turbo decoder consists of two single soft-in soft-out (SISO) decoders that work iteratively. The output of the first (upper decoder) feeds into the second to form a Turbo decoding iteration. Interleaver and deinterleaver blocks re-order data in this process.

The Turbo decoder supports the following features:

- 3GPP UMTS compliant with support for block sizes from 40 to 5,114.

- Run time parameters for interleaver size and number of iterations.
- Compile time parameters for the number of parallel engines, choice of decoding algorithm, and input precision.
- Double-buffering supports reduced latency real-time applications by allowing the decoder to receive data while processing the previous data block.
- No external memory required.
- C/MATLAB bit-accurate models for performance simulation or RTL test vector generation.
- VHDL or Verilog HDL test benches can be generated using the MegaWizard Plug-In Manager.
- Avalon® Streaming (Avalon-ST) interface.
- OpenCore Plus evaluation license.

Decoding Algorithms

Two variants of the Maximum A Posteriori (MAP) decoding algorithm are supported:

- LogMAP: Works on the logarithm domain of MAP and gives good bit error rate (BER) but consumes more logic resources. This option is currently not fully supported. Contact Altera for more information.
- MaxLogMAP: A simplified version of LogMAP which uses less logic resource at a cost of slightly reduced BER performance relative to the LogMAP variant. The MaxLogMAP algorithm implemented in this reference design is a version of MaxLogMAP corrected with a scaling factor, as proposed in reference 3.

Double-Buffering

The data path through the decoder is double-buffered to allow a new data block to be shifted in while decoding and shifting out the previous block. This technique reduces the delay in I/O operation, makes use of the hardware as much as possible and improves the overall throughput.

Input Data Format

Table 2 shows the required input data ordering.

Table 2. Input Data Ordering (Part 1 of 2)

Input Data	sink_data		
	3N-1 downto 2N	2N-1 downto N	N-1 downto 0
0	Z_0	Z_0	X_0
1	Z_1	Z_1	X_1
...
K-1	Z_{K-1}	Z_{K-1}	X_{K-1}
K	X_{K+1}	Z_K	X_K
K+1	Z_{K+2}	X_{K+2}	Z_{K+1}
K+2	X_{K+1}	Z_K	X_K

Table 2. Input Data Ordering (Part 2 of 2)

Input Data	sink_data		
	3N-1 downto 2N	2N-1 downto N	N-1 downto 0
$K+3$	Z_{K+2}	X_{K+2}	Z_{K+1}
Note to Table 2: (1) N represents the number of input bits (IN_WIDTH_g).			

The Turbo decoder requires all data to be in the log-likelihood format. The connected system must provide soft information, including parity 1 and parity 2 bit sequences according to the following equation:

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

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. Table 3 shows the 4-bit mapping values.

Table 3. The Meanings of the Input Values

Input (3 downto 0)	Meaning
0111	Most likelihood of a 1
...	...
0001	Lowest likelihood of a 1
0000	Equal probability of a 0 or 1
1111	Lowest likelihood of a 0
...	...
1001	Most likelihood of a 0
1000	Not used

Output Data Format

The ordering of output bits is:

$$X_0, X_1, X_2, \dots, X_{K-1}$$

Latency Calculation

The decoding delay D is the number of clock cycles consumed to decode an entire block of data. This is dependant on the block size, the number of iterations to perform, and the number of engines available in the decoder.

Contact Altera for a table of decoding delays associated with different block sizes.

Throughput Calculation

Throughput can be calculated using:

$$T = \frac{K \times F_{max}}{D} \text{ Mbps}$$

Where K is the block size, F_{max} is the system clock speed and D is the decoding delay.

Test Vector Generation

The following files are needed to perform RTL simulation:

- `ctc_blksize.txt`
- `ctc_data_input.txt`
- `ctc_iter_data.txt`

One test case is provided in `<Turbo Install path>/turbo_for_umts/lib/test_files`.

You can use the following procedures to generate your own test vectors using the provided system:

1. Start MATLAB (version 2006a or later)
2. Change the working directory to `<Turbo Install path>/turbo_for_umts/cml`.
3. Run the following command:

```
Cml_Startup.
```

4. Make a test subdirectory by running the following command:

```
mkdir ../test
```

5. Check the file `Scenarios_UMTS_RTL.m` for parameters such as *SNR*, and *max_iterations* that you would like to change.



For details of these parameters, refer to the `readme.pdf` file in `<Turbo Install path>/turbo_for_umts/cml/documentation`.

6. Make any required changes and save the file.
7. Run the following command:

```
[sim_param, sim_state] = CmlSimulate('Scenarios_UMTS_RTL', [40 40  
40 56:58 5114]);
```

This command generates test vectors for feeding the decoder with inputs of block size 40 three times and then 56, 57, 58 and 5114 as the last block. This matrix can be modified to fit your test needs.

If the above command is run successfully, a few files are created in `<Turbo install path>/turbo_for_umts/test`. You can modify the parameter *dump_dir* in `Scenarios_UMTS_RTL.m` to change the file location. Copy these files to your Turbo decoder Quartus II project directory (created through the MegaWizard Plug-In Manager).

You should be able to find a file `ctc_decoded_output_gold.txt` in your test vector directory. This file is generated from the software simulation model.

After RTL simulation another file `ctc_decoded_output.txt` is created, which should match the contents of `ctc_decoded_output_gold.txt`.



For information on how to start RTL simulation, refer to “[Simulate the Design](#)” on [page 18](#).

Avalon Streaming Interface

The Avalon® Streaming (Avalon-ST) interface is an evolution of the Atlantic™ interface. The Avalon-ST interface defines a standard, flexible, and modular protocol for data transfers from a source interface to a sink interface and simplifies the process of controlling the flow of data in a datapath.

Avalon-ST interface signals can describe traditional streaming interfaces supporting a single stream of data without knowledge of channels or packet boundaries. Such interfaces typically contain data, ready, and valid signals. The Avalon-ST interface can also support more complex protocols for burst and packet transfers with packets interleaved across multiple channels.

The Avalon-ST interface inherently synchronizes multi-channel designs, which allows you to achieve efficient, time-multiplexed implementations without having to implement complex control logic.

The Avalon-ST interface supports backpressure, which is a flow control mechanism, where a sink can signal to a source to stop sending data. The sink typically uses backpressure to stop the flow of data when its FIFO buffers are full or when there is congestion on its output.

When designing a datapath, which includes a 3GPP UMTS Turbo Decoder, you may not need backpressure if you know the downstream components can always receive data. You may achieve a higher clock rate by driving the `source_ready` signal high, and not connecting the `sink_ready` signal.

The Avalon-ST interface used in this 3GPP UMTS Turbo Reference Design has a `READY_LATENCY` value of zero.



For more information on the Avalon-ST interface, refer to the [Avalon Interface Specification](#).

Performance

Resource and error performance figures are available from Altera. Advise the target device, clock speed (F_{max}), input precision and number of iterations if you require figures for any other configuration.

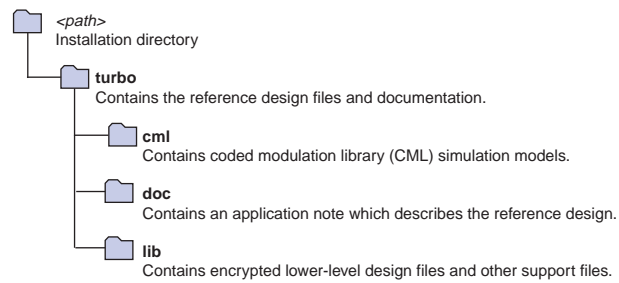
System Requirements

The 3GPP UMTS Turbo Reference Design is supported on Windows XP and requires the Quartus II software v9.0.

Installing the Reference Design

The reference design is available as an InstallShield file from the Altera Wireless business unit.

[Figure 4](#) shows the directory structure for the design files when they have been installed.

Figure 4. Reference Design Directory Structure

OpenCore Plus Evaluation

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

- Simulate the behavior of the 3GPP UMTS Turbo Reference Design 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 megafunctions.
- Program a device and verify your design in hardware.

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

After you purchase a license, you can request a license file from the Altera website at www.altera.com/licensing and install it on your computer. When you request a license file, Altera emails you a **license.dat** file. If you do not have Internet access, contact your local Altera representative.



For more information on OpenCore Plus hardware evaluation, refer to [AN 320: OpenCore Plus Evaluation of Megafunctions](#).

OpenCore Plus Time-Out Behavior

OpenCore[®] Plus hardware evaluation supports the following two operation modes:

- 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 functions in a design time out simultaneously when the most restrictive evaluation time is reached. If there is more than one function in a design, a specific function's time-out behavior may be masked by the time-out behavior of the other functions.

The untethered timeout for the 3GPP UMTS Turbo Reference Design is 1 hour; the tethered timeout value is indefinite.

The `reset_n` signal is forced low when the hardware evaluation time expires, keeping the 3GPP UMTS Turbo Reference Design permanently in its reset state.

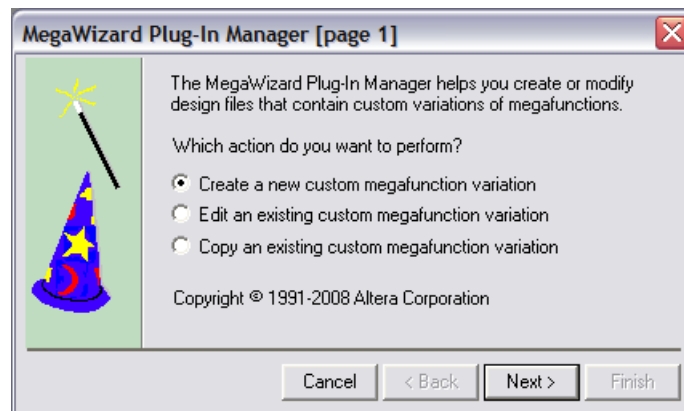
Getting Started

After you have installed the 3GPP UMTS Turbo Reference Design, a **Turbo for UMTS** megafunction is available in the **Error Detection/Correction** section of the MegaWizard Plug-in Manager in the Quartus II software.

The MegaWizard Plug-in Manager flow allows you to parameterize your Turbo decoder, and manually integrate the MegaCore function variation into a Quartus II design. Follow the steps below to use the MegaWizard Plug-in Manager.

1. Create a new project using the **New Project Wizard** available from the File menu in the Quartus II software.
2. Click **MegaWizard Plug-in Manager** on the Tools menu, and select **Create a new custom megafunction variation** (Figure 5).

Figure 5. MegaWizard Plug-In Manager



3. Click **Next**, expand DSP and choose **Turbo for UMTS v2.0** from the Error Detection/Correction section in the **Installed Plug-Ins** list (Figure 6).


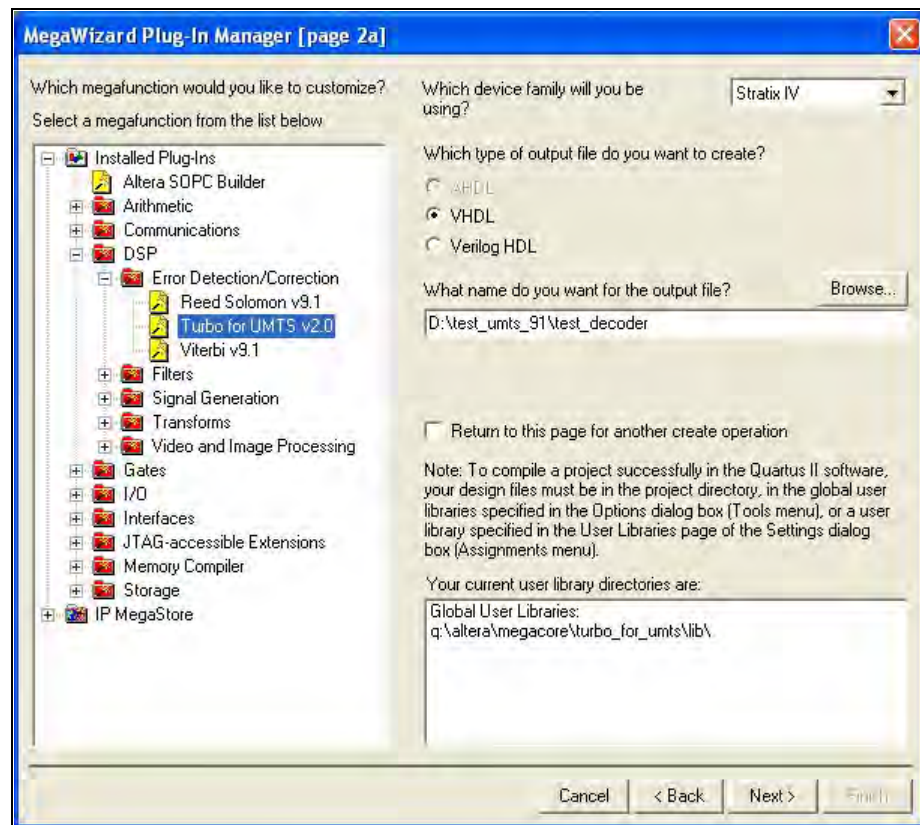
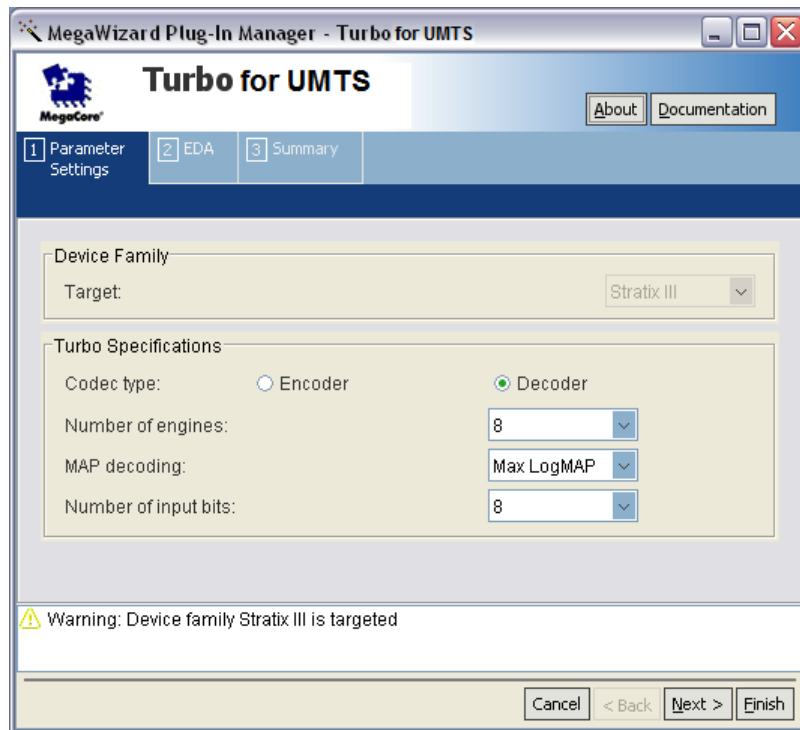
 If **Turbo for UMTS v2.0** does not list in the MegaWizard Plug-In Manager, you may need to add `<Turbo install Path>/turbo_for_ums/lib` to the Quartus II global user libraries (on the Tools menu, click **Options**).

Figure 6. Selecting the Turbo Megafunction

4. Verify that the device family is the same as you specified in the **New Project Wizard**.
5. Specify the top level output file name for your megafunction variation and click **Next** to display the MegaWizard interface **Parameter Settings** page (Figure 7).

Figure 7. MegaWizard Parameter Settings Page

6. Use the MegaWizard interface to specify the required parameters.

Table 4 shows a description of the available parameters.

Table 4. 3GPP UMTS Turbo Parameters

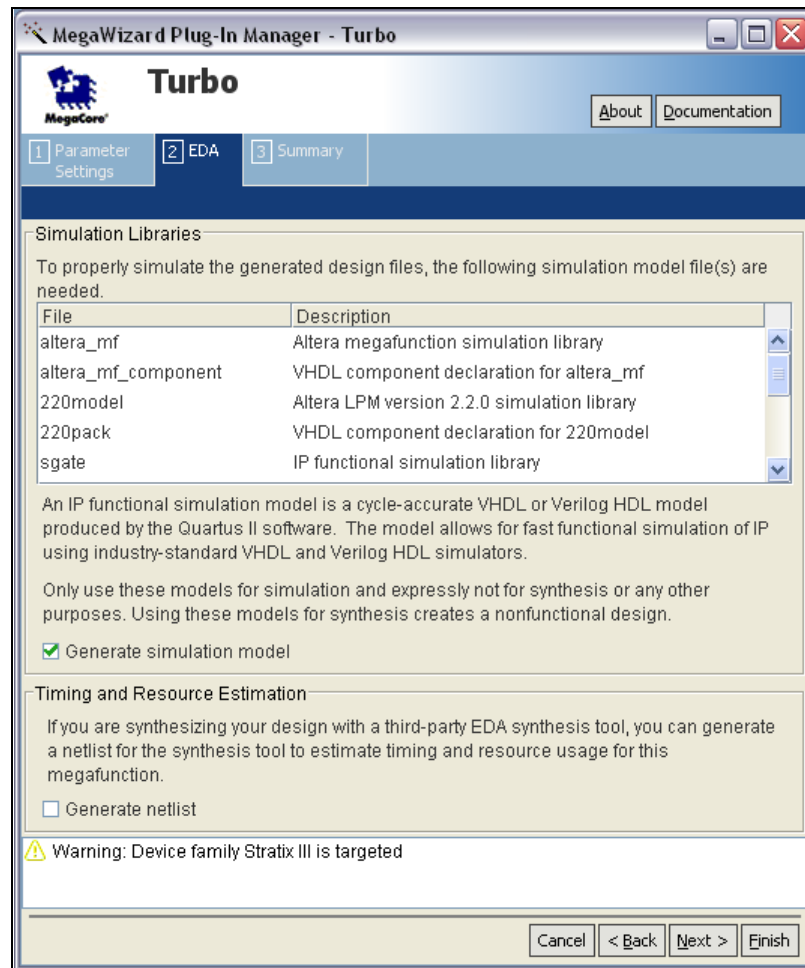
Parameter	Value	Description
Target	Stratix IV, Stratix III, Stratix II GX, Stratix II, Cyclone III, Arria GX	Displays the target device family that was specified when you created the Quartus II project.
Codec type	Encoder, Decoder	You can choose whether to generate an Encoder or Decoder.
Number of engines	2 or 4 (Note 1)	You can choose the number of engines used by the decoder.
MAP decoding	MaxLogMAP, LogMAP (Note 2)	You can choose from a list of available decoding algorithms.
Number of input bits	4, 5, 6, 7, 8 (Note 1)	You can choose the number of input bits to the decoder.

Notes to Table 4:

- (1) The reference design has been tested for four engines with eight input bits. Contact Altera if you require any other configuration.
- (2) The LogMAP option is not currently supported.

7. Click **Next** to complete the parameterization and display the **EDA** page (Figure 8).

Figure 8. MegaWizard EDA Page



8. On the **EDA** page, turn on **Generate Simulation Model**.



An IP functional simulation model is a cycle-accurate VHDL or Verilog HDL model produced by the Quartus II software.

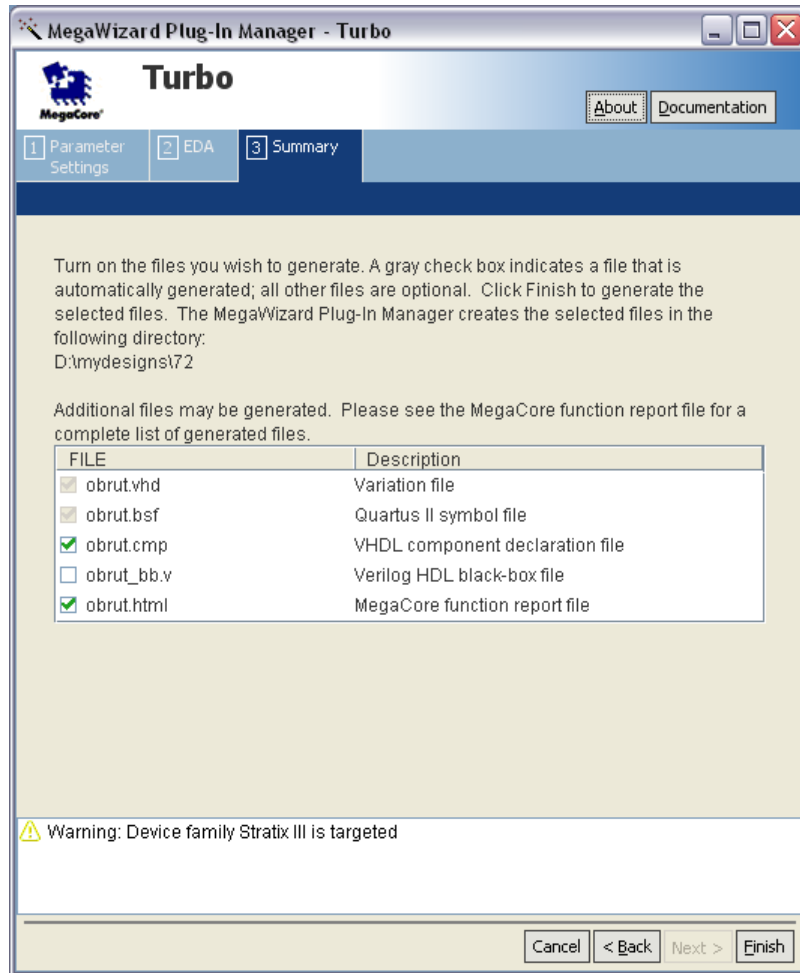


Use the simulation models only for simulation and not for synthesis or any other purposes. Using these models for synthesis creates a non-functional design.

9. Some third-party synthesis tools can use a netlist that contains only the structure of the megafunction, but not detailed logic, to optimize performance of the design that contains the megafunction. If your synthesis tool supports this feature, turn on **Generate netlist**.

10. Click Next to display the **Summary** page (Figure 9).

Figure 9. Summary Page



11. On the **Summary** tab, select the files you want to generate. A grey checkmark indicates a file that is automatically generated. All other files are optional.
12. Click **Finish** to generate the MegaCore function and supporting files. The generation phase may take several minutes to complete. The generation progress and status is displayed in a report window.



For more information about the MegaWizard Plug-In Manager, refer to the Quartus II Help.

Generated Files

Table 5 describes the generated files and other files that may be in your project directory. The names and types of files vary depending on the variation name and HDL type you specify during parameterization. For example, a different set of files are created based on whether you create your design in Verilog HDL or VHDL.

Table 5. Generated Files (Note 1)

Filename	Description
<variation name>.bsf	Quartus II symbol file for the MegaCore function variation. You can use this file in the Quartus II block diagram editor.
<variation name>.cmp	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.
<variation name>.html	Generation report file which contains lists of the generated files and ports for the MegaCore function variation.
<variation name>.qip	Contains Quartus II project information for your MegaCore function variation.
<variation name>.log	Log file.
<variation name>.vhd, or .v	A MegaCore function variation file, which defines a VHDL or Verilog HDL 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.
<variation name>.vho or .vo	VHDL or Verilog HDL IP functional simulation model.
<variation name>_bb.v	Verilog HDL black-box file for the MegaCore function variation. Use this file when using a third-party EDA tool to synthesize your design.
<variation name>_gb.v	A timing and resource estimation netlist for use in some third-party synthesis tools.
<variation name>_nativelink.tcl	A Tcl script that can assign NativeLink simulation testbench settings to the Quartus II project.
<variation name>_quartus.tcl	A Tcl script that can run compilation in the Quartus II software.
<variation name>_tb.vhd, or .v	A VHDL or Verilog HDL testbench file for the MegaCore function variation. The VHDL file is generated when a VHDL top level has been chosen or the Verilog HDL file when a Verilog HDL top level has been chosen.

Note to Table 5:

(1) The <variation name> prefix is added automatically using the base output file name you specified in the MegaWizard Plug-In Manager.



A generation report file containing a list of the design files and ports defined for your MegaCore function variation is saved as a HTML file, if you turned on the **MegaCore function report file** in the MegaWizard **Summary** page.

Signals

The generation function report also lists the megafunction variation ports.

Table 6 show the Turbo encoder signals.

Table 6. 3GPP UMTS Turbo Encoder Signals

Signal	Direction	Description
clk	Input	Clock signal that clocks all internal registers.
reset_n	Input	Active low reset signal. The megafunction must always be reset before receiving data. If the megafunction is not reset, the Turbo encoder may produce unexpected results due to feedback signals.
sink_blk_size	Input	Specifies the incoming block size.
sink_sop	Input	Marks the start of an incoming packet.
sink_eop	Input	Marks the end of an incoming packet.
sink_valid	Input	Asserted when data at <code>sink_data</code> is valid. When <code>sink_valid</code> is not asserted, processing is stopped until <code>sink_valid</code> is re-asserted.
source_ready	Input	Asserted by the downstream module if it is able to accept data.
sink_data	Input	Input data. See “Input Data Format” on page 3 for the required data ordering.
sink_error	Input	Error signal indicating Avalon Streaming protocol violations on input side. Any non-zero value on the <code>sink_error</code> port causes the Turbo encoder to ignore the current data block. The value received from this port is written to the <code>source_error</code> output port a few cycles later.
source_sop	Output	Marks the start of an outgoing packet.
source_eop	Output	Marks the end of an outgoing packet.
source_valid	Output	Asserted by the megafunction when there is valid data to output.
sink_ready	Output	Indicates when the megafunction is able to accept data.
source_error	Output	Error signal indicating Avalon-ST protocol violations on source side: <ul style="list-style-type: none"> ■ 00: No error ■ 01: Missing start of packet ■ 10: Missing end of packet ■ 11: Unexpected end of packet Other types of errors may also be marked as 11.
source_data	Output	Output data. See Table 1 on page 3 for the data ordering.
source_blk_size	Output	Specifies the outgoing block size. This port is a debug port in the testbench and can be left unconnected.

Table 7 shows the Turbo decoder signals.

Table 7. 3GPP UMTS Turbo Decoder Signals (Part 1 of 2)

Signal	Direction	Description
clk	Input	Clock signal that clocks all internal registers.
reset_n	Input	Active low reset signal. The megafunction must always be reset before receiving data. If the megafunction is not reset, the 3GPP UMTS Turbo Reference Design may produce unexpected results due to feedback signals.
sink_blk_size	Input	Specifies the incoming block size.
sink_sop	Input	Marks the start of an incoming packet.
sink_eop	Input	Marks the end of an incoming packet.
sink_valid	Input	Asserted when data at <code>sink_data</code> is valid. When <code>sink_valid</code> is not asserted, processing is stopped until <code>sink_valid</code> is re-asserted.
source_ready	Input	Asserted by the downstream module if it is able to accept data.

Table 7. 3GPP UMTS Turbo Decoder Signals (Part 2 of 2)

Signal	Direction	Description
sink_data	Input	Input data (See Table 2 on page 6 for the required input data ordering).
sink_error	Input	Error signal indicating Avalon-ST protocol violations on input side. Any non-zero value on the sink_error port causes the Turbo decoder to ignore the current data block. The value received from this port is written to the source_error output port a few cycles later.
sink_iter	Input	Specifies the number of half-iterations.
source_sop	Output	Marks the start of an outgoing packet.
source_eop	Output	Marks the end of an outgoing packet.
source_valid	Output	Asserted by the MegaCore function when there is valid data to output.
sink_ready	Output	Indicates when the MegaCore function is able to accept data.
source_error	Output	Error signal indicating Avalon-ST protocol violations on source side: <ul style="list-style-type: none"> ■ 00: No error ■ 01: Missing start of packet ■ 10: Missing end of packet ■ 11: Unexpected end of packet Other types of errors may also be marked as 11.
source_data	Output	Output data.
source_blk_size	Output	Specifies the outgoing block size. This port is used as a debug port in the testbench and can be left unconnected.

Simulate the Design

You can simulate your design using the MegaWizard-generated VHDL or Verilog HDL IP functional simulation models and testbench.

The IP functional simulation model is either a `.vo` or `.vho` file, depending on the output language you specified. Compile the `.vo` or `.vho` file in your simulation environment to perform functional simulation of your custom variation of the MegaCore function.



For more information on IP functional simulation models, refer to the Simulating Altera IP in Third-Party Simulation Tools chapter in volume 3 of the *Quartus II Handbook*.

Simulating in Third-Party Simulation Tools Using NativeLink

You can perform a simulation in a third-party simulation tool from within the Quartus II software, using NativeLink.

You can use the Tcl script file `<variation name>_nativelink.tcl` to assign default NativeLink testbench settings to the Quartus II project.

To perform a simulation in the Quartus II software using NativeLink, perform the following steps:

1. Create a custom MegaCore function variation as described earlier in this chapter but ensure you specify your variation name to match the Quartus II project name.

2. Check that the absolute path to your third-party EDA tool is set in the **Options** page under the Tools menu in the Quartus II software.
3. On the Processing menu, point to **Start** and click **Start Analysis & Elaboration**.
4. On the Tools menu, click **Tcl scripts**. Select the `<variation name>_nativelink.tcl` Tcl script and click Run. Check for a message confirming that the Tcl script was successfully loaded.
5. On the Assignments menu, click **Settings**, expand **EDA Tool Settings**, and select **Simulation**. Select a simulator under **Tool name** then in **NativeLink Settings**, select **Compile test bench** and click **Test Benches**. Confirm that appropriate testbench settings have been applied to the Quartus II project.
6. On the Tools menu, point to **EDA Simulation Tool** and click **Run EDA RTL Simulation**.



Refer to *Simulating Altera IP Using the Quartus II NativeLink Feature* in volume 3 of the *Quartus II Handbook* for more information.

Compile the Design and Program a Device

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

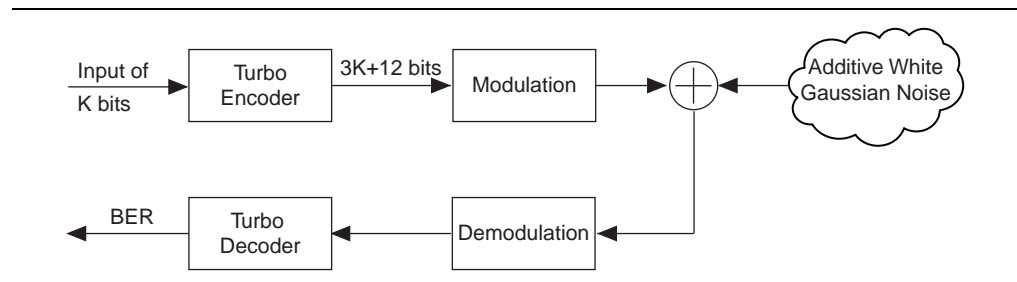
After a successful compilation, you can program the targeted Altera device and verify the design in hardware. Refer to Quartus II Help for instructions on programming your design.

Verification Methodology

The following steps describe the verification process used in the development of the 3GPP UMTS Turbo Reference Design:

1. The floating-point simulation model is plugged into a test vector generator using the simulation flow shown in [Figure 10](#).

Figure 10. Turbo Verification Methodology



2. The BER performance of the floating-point model is compared against a reference BER performance.
3. A fixed-point model is developed and parameters such as the number of bits are adjusted to check that similar BER performance to the reference is achieved.
4. An RTL model is generated in VHDL. The RTL is tested using ModelSim using the same test vector generation suite for the floating-point and fixed-point models.

5. During the development of RTL, the results are always validated with the fixed-point model results. All RTL building blocks have separate test benches to verify that RTL models match the software models.
6. The design uses the Quartus II software as the synthesis and place and route tool during the development of the RTL implementation. The RTL is optimized in terms of resource usage and performance.

References

For more information about Turbo codes and the 3GPP specification, consult the following references:

1. *3GPP Technical Specification: Group Radio Access Network, Evolved Universal Terrestrial Radio Access, Multiplexing and channel coding (Release 6)*, TS 25.212 v3.3.0, June 2000.
2. C. Berrou, A. Glavieux, and P. Thitimajshima, *Near Shannon limit error-correcting coding and decoding: Turbo codes*, in Proceedings of the IEEE International Conference on Communications, 1993, pp. 1064-1070.
3. J. Vogt, A. Finger, *Improving the max-log-MAP turbo decoder*, Electronics Letters, 36(23), 1937-1939, 2000.
4. *Avalon Interface Specifications*.
5. *AN 320: OpenCore Plus Evaluation of Megafunctions*.

Revision History

Table 8 shows the revision history for the AN-526: 3GPP UMTS Turbo Reference Design application note.

Table 8. AN-526 Revision History

Version	Date	Summary
2.0	January 2010	Added support for Turbo encoder.
1.0	October 2008	First release of this application note.



101 Innovation Drive
San Jose, CA 95134
www.altera.com
Technical Support
www.altera.com/support

Copyright © 2010 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.



I.S. EN ISO 9001