

The Altera® 3GPP LTE Turbo Reference Design demonstrates using Turbo codes for encoding with trellis termination support, and forward error correction (FEC) decoding with early termination support. The reference design is suitable for 3GPP long term evolution (LTE) channel card or baseband modem applications compatible with the *3GPP Technical Specification* (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 LTE, which require throughputs in the range from two to several hundred Mbps.

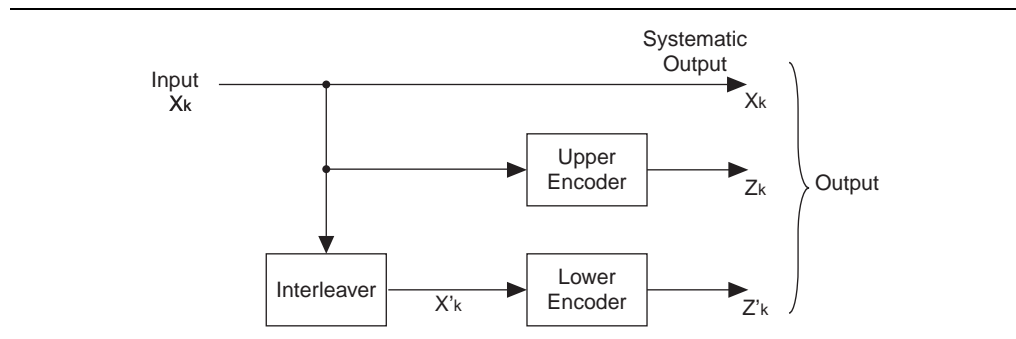
## Turbo Encoder

The 3GPP LTE Turbo encoding specified in the 3GPP LTE 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 LTE compliant.
- All 3GPP LTE 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).

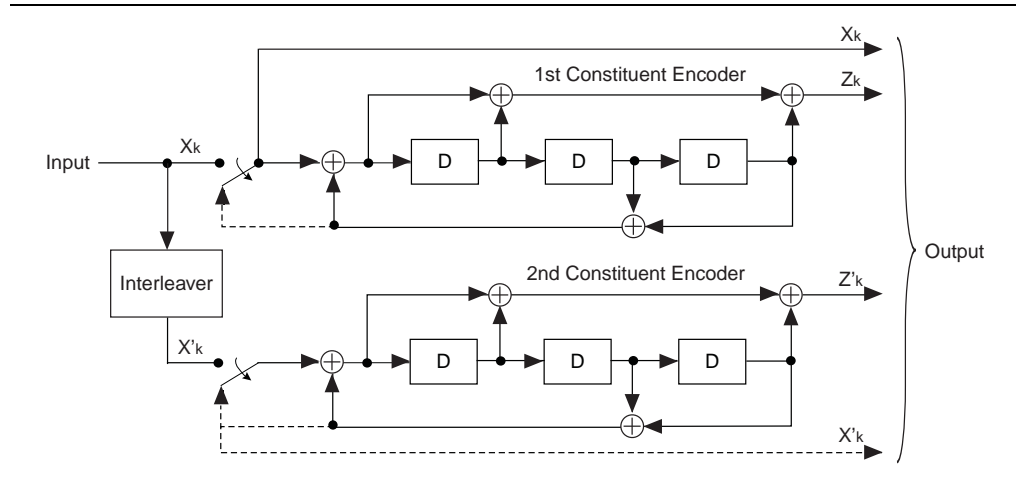
Trellis termination is performed by taking the tail bits from the shift register feedback after all information bits are encoded. The tail bits are padded after the encoding of information bits.

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:

$$X'_i = X_{\pi(i)}, i = 0, 1, \dots, K-1$$

Where the relationship between the output index  $i$  and the input  $\pi(i)$  index satisfies the following quadratic form:

$$\pi(i) = (f_1 i + f_2 i^2) \text{mod} K$$

The parameters  $f_1$  and  $f_2$  depend on the block size  $K$ . [Table 1](#) shows the interleaver parameters specified in the *3GPP Technical Specification*.

**Table 1.** Turbo Code Internal Interleaver Parameters (Part 1 of 2)

$i$	$K_i$	$f_1$	$f_2$	$i$	$K_i$	$f_1$	$f_2$	$i$	$K_i$	$f_1$	$f_2$	$i$	$K_i$	$f_1$	$f_2$
<b>1</b>	40	3	10	<b>48</b>	416	25	52	<b>95</b>	1120	67	140	<b>142</b>	3200	111	240
<b>2</b>	48	7	12	<b>49</b>	424	51	106	<b>96</b>	1152	35	72	<b>143</b>	3264	443	204
<b>3</b>	56	19	42	<b>50</b>	432	47	72	<b>97</b>	1184	19	74	<b>144</b>	3328	51	104
<b>4</b>	64	7	16	<b>51</b>	440	91	110	<b>98</b>	1216	39	76	<b>145</b>	3392	51	212
<b>5</b>	72	7	18	<b>52</b>	448	29	168	<b>99</b>	1248	19	78	<b>146</b>	3456	451	192
<b>6</b>	80	11	20	<b>53</b>	456	29	114	<b>100</b>	1280	199	240	<b>147</b>	3520	257	220
<b>7</b>	88	5	22	<b>54</b>	464	247	58	<b>101</b>	1312	21	82	<b>148</b>	3584	57	336
<b>8</b>	96	11	24	<b>55</b>	472	29	118	<b>102</b>	1344	211	252	<b>149</b>	3648	313	228
<b>9</b>	104	7	26	<b>56</b>	480	89	180	<b>103</b>	1376	21	86	<b>150</b>	3712	271	232
<b>10</b>	112	41	84	<b>57</b>	488	91	122	<b>104</b>	1408	43	88	<b>151</b>	3776	179	236
<b>11</b>	120	103	90	<b>58</b>	496	157	62	<b>105</b>	1440	149	60	<b>152</b>	3840	331	120
<b>12</b>	128	15	32	<b>59</b>	504	55	84	<b>106</b>	1472	45	92	<b>153</b>	3904	363	244
<b>13</b>	136	9	34	<b>60</b>	512	31	64	<b>107</b>	1504	49	846	<b>154</b>	3968	375	248

**Table 1.** Turbo Code Internal Interleaver Parameters (Part 2 of 2)

<i>i</i>	$K_1$	$f_1$	$f_2$	<i>i</i>	$K_1$	$f_1$	$f_2$	<i>i</i>	$K_1$	$f_1$	$f_2$	<i>i</i>	$K_1$	$f_1$	$f_2$
14	144	17	108	61	528	17	66	108	1536	71	48	155	4032	127	168
15	152	9	38	62	544	35	68	109	1568	13	28	156	4096	31	64
16	160	21	120	63	560	227	420	110	1600	17	80	157	4160	33	130
17	168	101	84	64	576	65	96	111	1632	25	102	158	4224	43	264
18	176	21	44	65	592	19	74	112	1664	183	104	159	4288	33	134
19	184	57	46	66	608	37	76	113	1696	55	954	160	4352	477	408
20	192	23	48	67	624	41	234	114	1728	127	96	161	4416	35	138
21	200	13	50	68	640	39	80	115	1760	27	110	162	4480	233	280
22	208	27	52	69	656	185	82	116	1792	29	112	163	4544	357	142
23	216	11	36	70	672	43	252	117	1824	29	114	164	4608	337	480
24	224	27	56	71	688	21	86	118	1856	57	116	165	4672	37	146
25	232	85	58	72	704	155	44	119	1888	45	354	166	4736	71	444
26	240	29	60	73	720	79	120	120	1920	31	120	167	4800	71	120
27	248	33	62	74	736	139	92	121	1952	59	610	168	4864	37	152
28	256	15	32	75	752	23	94	122	1984	185	124	169	4928	39	462
29	264	17	198	76	768	217	48	123	2016	113	420	170	4992	127	234
30	272	33	68	77	784	25	98	124	2048	31	64	171	5056	39	158
31	280	103	210	78	800	17	80	125	2112	17	66	172	5120	39	80
32	288	19	36	79	816	127	102	126	2176	171	136	173	5184	31	96
33	296	19	74	80	832	25	52	127	2240	209	420	174	5248	113	902
34	304	37	76	81	848	239	106	128	2304	253	216	175	5312	41	166
35	312	19	78	82	864	17	48	129	2368	367	444	176	5376	251	336
36	320	21	120	83	880	137	110	130	2432	265	456	177	5440	43	170
37	328	21	82	84	896	215	112	131	2496	181	468	178	5504	21	86
38	336	115	84	85	912	29	114	132	2560	39	80	179	5568	43	174
39	344	193	86	86	928	15	58	133	2624	27	164	180	5632	45	176
40	352	21	44	87	944	147	118	134	2688	127	504	181	5696	45	178
41	360	133	90	88	960	29	60	135	2752	143	172	182	5760	161	120
42	368	81	46	89	976	59	122	136	2816	43	88	183	5824	89	182
43	376	45	94	90	992	65	124	137	2880	29	300	184	5888	323	184
44	384	23	48	91	1008	55	84	138	2944	45	92	185	5952	47	186
45	392	243	98	92	1024	31	64	139	3008	157	188	186	6016	23	94
46	400	151	40	93	1056	17	66	140	3072	47	96	187	6080	47	190
47	408	155	102	94	1088	171	204	141	3136	13	28	188	6144	263	480

## 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 3 shows the ordering for a block of size  $K$ .

**Table 2.** Turbo Encoder Output Data Ordering

Output Data	source_data		
	2	1	0
0	$Z'_0$	$Z_0$	$X_0$
1	$Z'_1$	$Z_1$	$X_1$
...	...	...	...
$K-1$	$Z'_{K-1}$	$Z_{K-1}$	$X_{K-1}$
$K$	$X_{K+1}$	$Z_K$	$X_K$
$K+1$	$Z_{K+2}$	$X_{K+2}$	$Z_{K+1}$
$K+2$	$X'_{K+1}$	$Z'_K$	$X'_K$
$K+3$	$Z'_{K+2}$	$X'_{K+2}$	$Z'_{K+1}$

## 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  $K = 6144$ ,  $D = 6144 + 14 = 6158$
- 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,  $L = 0.22 \mu\text{s}$  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  $244.44 \text{ Mbps}$  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/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/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_LTE_ENCODER_RTL',  
                                     [832 832 832 40 48 56 6144]);
```

This command generates test vectors for feeding the encoder with inputs of block size 832 three times and then 40, 48, 56 and 6144 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/test`. You can modify the parameter `dump_dir` in the **Scenarios\_LTE\_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**

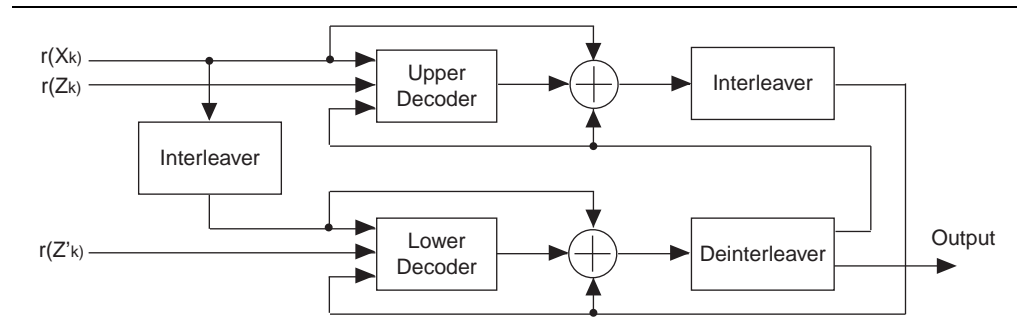
After RTL simulation, another file **ctc\_encoder\_output.txt** is created which should match the contents of **ctc\_encoder\_output\_gold.txt**.



For information on how to start RTL simulation, refer to [“Simulate the Design” on page 22](#).

## Turbo Decoder

[Figure 3 on page 7](#) 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, which 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 LTE compliant.
- Run time parameters for interleaver size and number of iterations.
- Early termination support with cyclical redundancy check (CRC).
- Compile time parameters for the number of parallel engines, choice of decoding algorithm, input precision and output size.
- 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.
- You can generate VHDL or Verilog HDL testbenches using the MegaWizard Plug-In Manager.
- Avalon Streaming (Avalon-ST) interface.
- OpenCore Plus evaluation license.

 For an example of a design that uses a Turbo decoder in a LTE basestation, refer to [AN534: LTE PUSCH Rate Dematcher Reference Design](#).

## Decoding Algorithms

The following 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 that 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.

## Input Data Format

Table 3 shows the input data ordering required for the Turbo decoder.

**Table 3.** Turbo Decoder Input Data Ordering

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	$X'_{K+1}$	$Z'_K$	$X'_K$
K+3	$Z'_{K+2}$	$X'_{K+2}$	$Z'_{K+1}$

**Note to Table 3:**  
(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 1, divided by the probability of this bit being a 0, 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 4 shows the 4-bit mapping values.

**Table 4.** 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 number of output bits (*OUT\_WIDTH\_g*) can be specified to be 1 or 8 bits wide. For 1 bit, the ordering is:

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

Table 5 shows the output data ordering for 8 bits.

**Table 5.** 8-bit Output Data Ordering

Output Order	source_data							
	7	6	5	4	3	2	1	0
1	$X_7$	$X_6$	$X_5$	$X_4$	$X_3$	$X_2$	$X_1$	$X_0$
2	$X_{15}$	$X_{14}$	$X_{13}$	$X_{12}$	$X_{11}$	$X_{10}$	$X_9$	$X_8$
...	...	...	...	...	...	...	...	...
$K/8$	$X_{K-1}$	$X_{K-2}$	$X_{K-3}$	$X_{K-4}$	$X_{K-5}$	$X_{K-6}$	$X_{K-7}$	$X_{K-8}$

## 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.

The following calculations assume no early termination is taking place, so are the worst case latency.

Let  $K$  be the block size,  $I$  be the number of decoding iterations and  $N$  be the number of engines specified in the decoder. Then the decoding delay  $D$  can be calculated using one of the following formula:

- If  $K < 264$ ,  $D = 26 + (2 \times f(K,N) + 14) \times 2 \times I$
- If  $K \geq 264$ ,  $D = 26 + (f(K,N) + 46) \times 2 \times I$

Where:

$$f(K,N) = \begin{cases} K/N & \text{if } K \text{ is divisible by } N \\ K/8 & \text{if } K \text{ is not divisible by } N \end{cases}$$

For example:

- If  $K = 6144$ ,  $N = 8$ ,  $I = 8$ ,  $D = 26 + (6144/8 + 46) \times 2 \times 8 = 13,050$
- If  $K = 40$ ,  $N = 8$ ,  $I = 8$ ,  $D = 26 + (2 \times 40/8 + 14) \times 2 \times 8 = 410$

The decoding latency (the time the decoder takes to decode an entire block to the decoded data is ready for output) 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,  $L = 65.25 \mu\text{s}$  and  $2.05 \mu\text{s}$  respectively for  $f_{MAX} = 200 \text{ MHz}$ .



These calculations are for running the Turbo decoder for 8 iterations ( $I = 8$ ) and assuming no early termination has occurred.

## 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,  $T = 94.16$  Mbps and  $19.51$  Mbps respectively.



These calculations are for running the Turbo decoder for 8 iterations ( $I = 8$ ) and assuming no early termination has occurred.

## Early Termination Support

This version of the LTE Turbo reference design supports early termination using CRC24A or CRC24B (refer to the *3GPP Technical Specification* for more information).

The CRC checksum generated by the SISO decoder outputs (both lower and the upper decoders in [Figure 3](#)) are checked after every iteration. Rather than continuing until the maximum number of iterations specified at the input ports, the Turbo decoding is terminated as soon as the CRC results with success.

The early termination reduces power consumption, the overall latency and increases the throughput predicted above significantly. Literatures also shows it increases BER performance of the decoder. The gains in any of the above metrics are dependent on the signal-to-noise ration (SNR) ratio of the received data block, block size, and the maximum number of iterations you have specified.

## Test Vector Generation

The following files are needed to perform RTL simulation:

- `ctc_input_info.txt`
- `ctc_input_data.txt`

One test case is provided in `<Turbo Install path>/turbo/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/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_LTE_CRC_ET_RTL.m` for parameters such as *SNR*, *CRC type*, and *max\_iterations* that you would like to change. Make any required changes and save the file.



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

6. Run the following command:

```
[sim_param, sim_state] = CmlSimulate('Scenarios_LTE_CRC_ET_RTL',  
                                     [832 832 832 40 48 56 6144]);
```

This command generates test vectors for feeding the decoder with inputs of block size 832 three times and then 40, 48, 56 and 6144 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/test`. You can modify the parameter `dump_dir` in the `Scenarios_LTE_CRC_ET_RTL.m` file to change this location. Copy these files to your Turbo decoder Quartus II project directory (created through the MegaWizard Plug-In Manager).

The following four files are generated from the software simulation model:

- `ctc_input_info.txt`
- `ctc_input_data.txt`
- `ctc_decoded_output_gold.txt`
- `ctc_output_et_info_gold.txt`

After RTL simulation, another two files `ctc_decoded_output.txt` and `ctc_output_et_info.txt` are created, which should match the contents of `ctc_decoded_output_gold.txt` and `ctc_output_et_info_gold.txt`.



For information on how to start RTL simulation, refer to [“Simulate the Design” on page 22](#).

## 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 LTE Turbo reference design, 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 LTE Turbo reference design has a `READY_LATENCY` value of zero.



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

## Handling Packet Format Errors

The Turbo megafunction has two error ports to communicate data errors in the system:

- `sink_error` is a 2-bit input port to receive the up front error signal.
- `source_error` is a 2-bit output port to indicate that there is an error condition (either caught by the Turbo megafunction or elsewhere in the previous blocks).

The megafunction can handle and recover from the following packet format related errors:

- If an error code is received from the `sink_error` port during the input of a data block, the Turbo megafunction assumes the current data block contains some sort of error and discards the data. Once the error signal is asserted low, the Turbo megafunction expects a fresh start-of-packet (`sink_sop = 1`, `sink_valid = 1`) and ignores the data input until a fresh packet is received.
- If there is a misplaced start-of-packet (`sink_sop`) or end-of-packet (`sink_eop`), an error code is issued depending on the type of error:
  - 01 -> missing start-of-packet
  - 10 -> missing end-of-packet
  - 11 -> unexpected start-of-packet and unexpected end-of-packet
- If the data block size is not supported by the LTE standard, the Turbo megafunction issues an error signal with the value 11 and ignores the rest of the data block until a fresh start of a packet.

Because of the long processing time of a data block and double-buffering at the input and output ports, the errors at the input data are reported as soon as they occur. Therefore, the `source_error` signal might get asserted high at any time during the output of a previous block.

When an error is detected, the error code appears for one clock cycle only.

If there is more than one error related to a particular data block, the Turbo megafunction only displays the error code for the first detected error.

It takes a few clock cycles to report the detected error at the `source_error` port.

### Exceptions to the Error Recovery

When the detected error is very close to the boundary of the end-of-packet (that is when there is a missing end-of-packet, or unexpected start-of-packet or end-of-packet) and there is a block following straight after the erroneous block with a different CRC type to the previous block, the Turbo megafunction may not recover from the error immediately. However, the error is reported from the `source_error` port in all circumstances.

## System Requirements

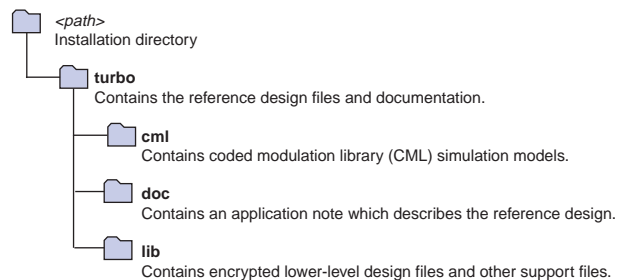
The 3GPP LTE Turbo reference design is supported on Windows XP and Linux workstations, and requires the Quartus II software v9.0.

## Installing the Reference Design

The 3GPP LTE Turbo reference design is available as an InstallShield file from the Altera Wireless business unit.

Figure 4 shows the directory structure for the design files.

**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 LTE 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](http://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 LTE Turbo decoder 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 LTE Turbo decoder reference design permanently in its reset state.

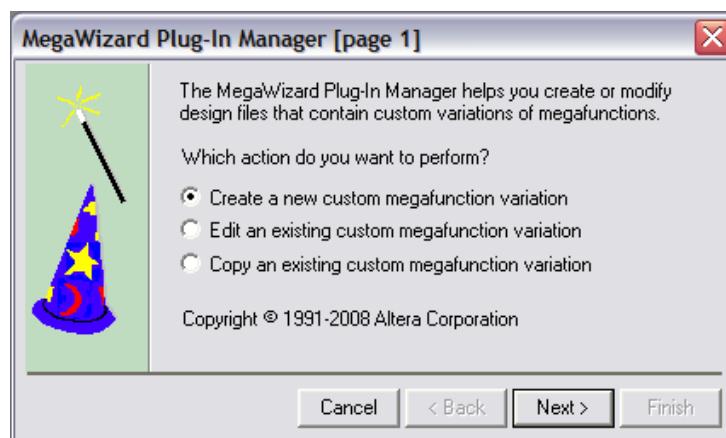
## Getting Started

After you have installed the 3GPP LTE Turbo decoder reference design, a Turbo 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 megafunction variation into a Quartus II design. Perform the following steps 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).

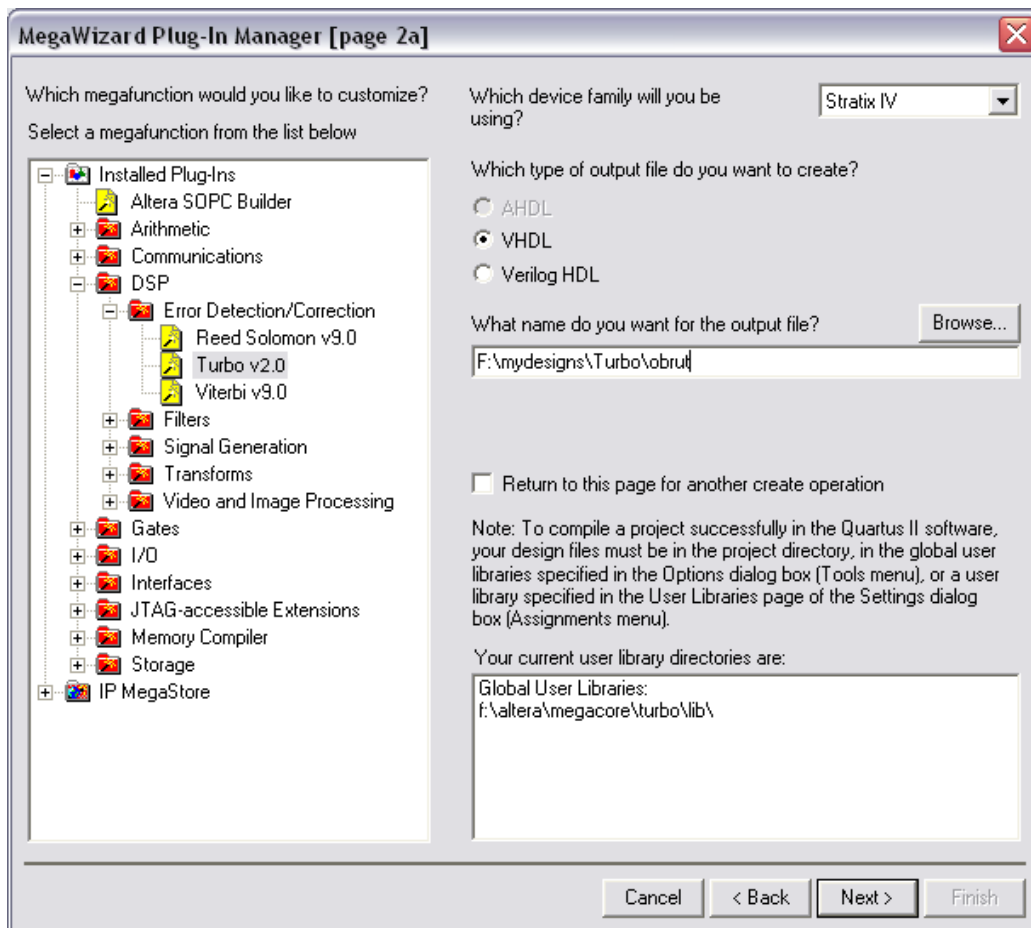
**Figure 5.** MegaWizard Plug-In Manager



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

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

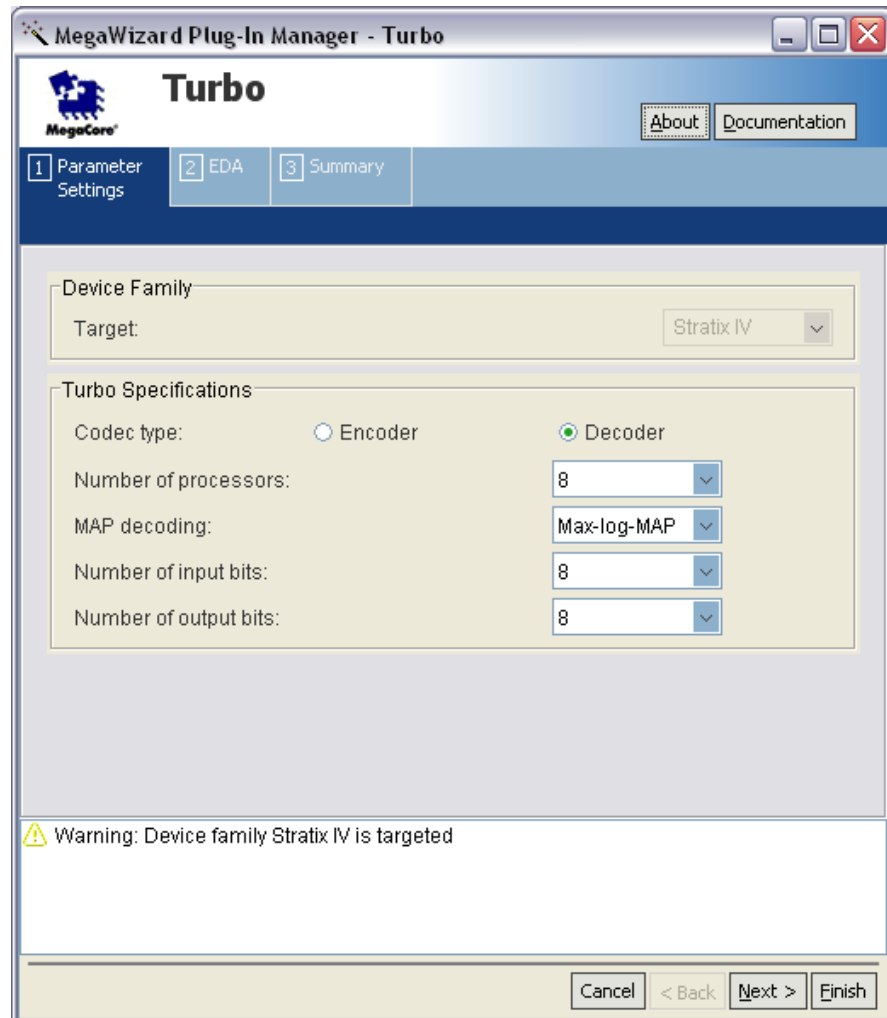
**Figure 6.** Selecting the Turbo Megafunction



4. Verify that the device family is the same as you specified in the **New Project Wizard**.

- 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



- Use the MegaWizard interface to specify the required parameters. Table 6 shows a description of the parameters.

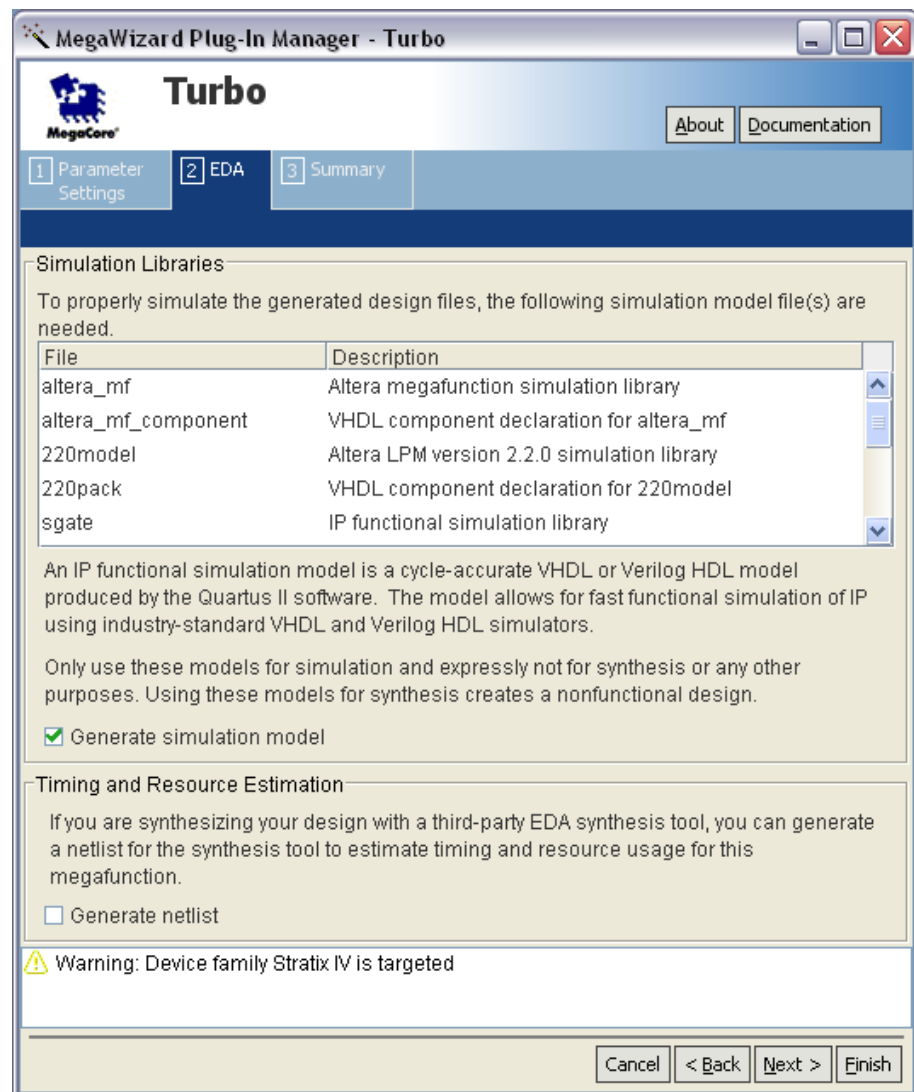
**Table 6.** 3GPP LTE Turbo Parameters (Part 1 of 2)

Parameter	Value	Description
Target	Stratix IV, Stratix III, Stratix II GX, Stratix II, Cyclone III, Arria II GX, Arria GX	Displays the target device family that was specified when you created the Quartus II project.
Codec type	Encoder, Decoder	Select whether to generate an encoder or decoder.
Number of engines	2, 4, 8, 16 (1)	Select the number of engines used by the decoder.
MAP decoding	MaxLogMAP, LogMAP (2)	Select from a list of available decoding algorithms.
Number of input bits	4, 5, 6, 7, 8 (1)	Select the number of input bits to the decoder.

**Table 6.** 3GPP LTE Turbo Parameters (Part 2 of 2)

Parameter	Value	Description
Number of output bits	8 (3)	The number of output bits from the decoder.
<b>Notes to Table 6:</b>		
(1) The reference design has been tested for 2 or 8 engines with 6 or 8 input bits. Contact Altera if you require any other configuration.		
(2) The LogMAP option is not currently supported.		
(3) Only 8-bit output mode is 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 nonfunctional 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 megafunction 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 7 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 7.** Generated Files (Note 1)

Filename	Description
<variation name>.bsf	Quartus II symbol file for the megafunction variation. You can use this file in the Quartus II block diagram editor.
<variation name>.cmp	A VHDL component declaration file for the megafunction variation. Add the contents of this file to any VHDL architecture that instantiates the megafunction.
<variation name>.html	Generation report file which contains lists of the generated files and ports for the megafunction variation.
<variation name>.qip	Contains Quartus II project information for your megafunction variation.
<variation name>.log	Log file.
<variation name>.vhd, or .v	A megafunction variation file, which defines a VHDL or Verilog HDL top-level description of the custom megafunction. 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 megafunction 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 megafunction 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 7:**

(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 function variation is saved as a HTML file if you turned on the **MegaCore function report file** check box in the MegaWizard **Summary** page.

## Signals

The generation function report also lists the megafunction variation ports.

Table 8 show the Turbo encoder signals.

**Table 8.** 3GPP LTE 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. See parameter <i>K</i> in <a href="#">Table 1 on page 3</a> .
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 <a href="#">“Input Data Format” on page 5</a> 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"> <li>■ 00: No error</li> <li>■ 01: Missing start of packet</li> <li>■ 10: Missing end of packet</li> <li>■ 11: Unexpected end of packet</li> </ul> Other types of errors may also be marked as 11.
source_data	Output	Output data. See <a href="#">Table 2 on page 5</a> 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 9 on page 21](#) show the Turbo decoder signals.


**Table 9.** 3GPP LTE Turbo Decoder 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 decoder may produce unexpected results due to feedback signals.
sink_blk_size	Input	Specifies the incoming block size (See parameter <i>K</i> in Table 1 on page 3).
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 sink_data is valid. When sink_valid is not asserted, processing is stopped until sink_valid is re-asserted.
source_ready	Input	Asserted by the downstream module if it is able to accept data.
sink_data	Input	Input data. (See Table 3 on page 8 for the required 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_max_iter	Input	Specifies the maximum number of half-iterations.
sel_CRC24A	Input	Specifies the type of CRC that you need for the current data block: <ul style="list-style-type: none"> <li>■ 0: CRC24A</li> <li>■ 1: CRC24B</li> </ul>
source_blk_id	Output	Specifies the outgoing block ID. This port is a debug port in the testbench and can be left unconnected.
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"> <li>■ 00: No error</li> <li>■ 01: Missing start of packet</li> <li>■ 10: Missing end of packet</li> <li>■ 11: Unexpected end of packet</li> </ul> Other types of errors may also be marked as 11.
source_data	Output	Output data. (See Table 5 on page 9 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.
CRC_pass	Output	Indicates whether CRC was successful: <ul style="list-style-type: none"> <li>■ 0: Fail</li> <li>■ 1: Pass</li> </ul>
CRC_type	Output	Indicates the type of CRC that was used for the current data block: <ul style="list-style-type: none"> <li>■ 0: CRC24A</li> <li>■ 1: CRC24B</li> </ul>
source_iter	Output	Shows the number of half iterations after which the Turbo decoder stopped processing the current data block.

## 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 megafunction.

 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.

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 megafunction variation, but ensure you specify your variation name to match the Quartus II project name.
2. Verify 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**.

 For more information, refer to *Simulating Altera IP Using the Quartus II NativeLink Feature* in volume 3 of the *Quartus II Handbook*.

## Compile the Design and Program a Device

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

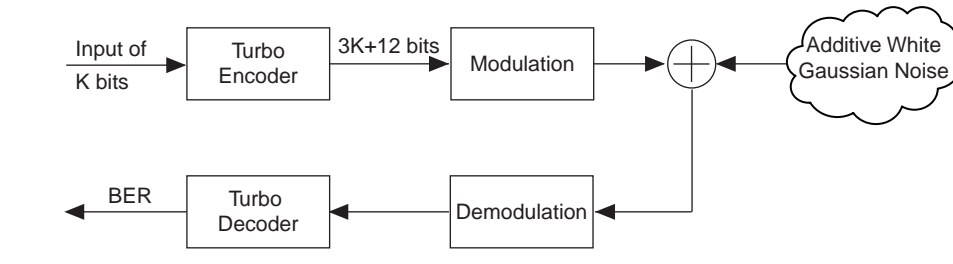
After a successful compilation, you can program the targeted Altera device and verify the design in hardware. Refer to the Quartus II Help for instructions on programming your design.

## Verification Methodology

The following steps describe the verification process that the development of the 3GPP LTE Turbo reference design uses:

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

**Figure 10.** Turbo Decoder 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 testbenches 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 for 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 8)*, TS 36.212 v8.3.0, May 2007.
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. *AN534: LTE PUSCH Rate Dematcher Reference Design*.
5. *Avalon Interface Specifications*.
6. *AN 320: OpenCore Plus Evaluation of Megafunctions*.

## Revision History

Table 10 shows the revision history for the *AN-505: 3GPP LTE Turbo Reference Design* application note.

**Table 10.** AN-505 Revision History

Version	Date	Summary
2.0	January 2010	Updated $f_{MAX}$ for turbo decoding.
1.2	June 2009	Added support for Turbo encoding.
1.1.1	February 2009	Added packet format error handling section.
1.1	September 2008	Added support for early termination with CRC.
1.0	February 2008	First release of this application note.



101 Innovation Drive  
San Jose, CA 95134  
[www.altera.com](http://www.altera.com)  
Technical Support  
[www.altera.com/support](http://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