Turbo Intel® FPGA IP User Guide

Updated for Intel® Quartus® Prime Design Suite: **20.4**

IP Version: **20.4.0**

Subscribe

Send Feedback

UG-TURBO | 2021.09.30

Latest document on the web: **PDF** | **HTML**

Contents

1. About the Turbo Intel® FPGA IP

The Turbo Intel® FPGA IP implements Turbo codes that is compliant with 3rd Generation Partnership Project (3GPP) LTE/LET-A and UMTS specification for integration in your wireless design. Forward-error correction (FEC) channel codes commonly improve the energy efficiency of wireless communication systems.

Turbo codes are suitable for 3G and 4G mobile communications and satellite communications to help transmitting and receiving messages over noisy channels. You can also use Turbo codes in other applications that require reliable information transfer over bandwidth- or latency-constrained communication links in the presence of data corrupting noise.

Related Information

- 3GPP TS 36.212 Version 15.2.1 Release 15
- 3GPP TS 25.212 Version 14.0.0 Release 14

1.1. Turbo Intel FPGA IP Features

The Turbo Intel FPGA IP offers the following features:

- General features:
  - 3GPP LTE compliant with support for block sizes from 40 to 6,144.
  - 3GPP UMTS compliant with support for block sizes from 40 to 5,114.
  - C/MATLAB bit-accurate models for performance simulation or RTL test vector generation.
- Decoder features:
  - Run time parameters for interleaver size and number of iterations.
  - Early termination with cyclical redundancy check (CRC).
  - Compile time parameters for the number of parallel engines, input precision.
  - Uses MaxLogMAP decoding algorithm.
  - Double-buffering for reduced latency real-time applications, which allows the decoder to receive data while processing the previous data block at compile time.
  - No external memory required.
- Encoder features:
  - Run-time selectable interleaver block sizes.
  - Compile time parameters for standard (LTE or UMTS) and parallel encoding engines (1, 4 or 8) for high throughput and low latency.
  - Code rate 1/3 only. Use external rate matching for other code rates.
  - Double-buffering allows the encoder to receive data while processing the previous data block at compile time.

1.2. Turbo Intel FPGA IP Device Family Support

Intel offers the following device support levels for Intel FPGA IP cores:

- Advance support—the IP is available for simulation and compilation for this device family. FPGA programming file (`.pof`) support is not available for Quartus Prime Pro Stratix 10 Edition Beta software and as such IP timing closure cannot be guaranteed. Timing models include initial engineering estimates of delays based on early post-layout information. The timing models are subject to change as silicon testing improves the correlation between the actual silicon and the timing models. You can use this IP core for system architecture and resource utilization studies, simulation, pinout, system latency assessments, basic timing assessments (pipeline budgeting), and I/O transfer strategy (data-path width, burst depth, I/O standards tradeoffs).
- Preliminary support—Intel verifies the IP with preliminary timing models for this device family. The IP core meets all functional requirements, but might still be undergoing timing analysis for the device family. You can use it in production designs with caution.
- Final support—Intel verifies the IP with final timing models for this device family. The IP core meets all functional and timing requirements for the device family. You can use it in production designs.

Table 1. Turbo IP Device Family Support

Device Family	Support
Intel Agilex™	Advance
Intel Stratix® 10	Final
Intel Arria® 10	Final
Intel Cyclone® 10	Final
Stratix V	Final
Arria V	Final
Cyclone V	Final
Stratix IV GT	Final
Stratix IV GX/E	Final
Cyclone IV	Final
Arria II GX	Final
Arria II GZ	Final
Intel MAX® 10 FPGA	Final
Other device families	No support

1.3. Turbo IP Core Performance and Resource Utilization

Table 2. Performance and Resource Utilization

Typical expected performance for a Turbo IP Core using the Quartus Prime software with the Intel Agilex (AGFA014R24A2E2VR0), Intel Stratix 10 (1SG280HU2F50E2VG), and Intel Arria 10 (10AT115S1F45E1SG) devices.

Device	Speed Grade	Parameters				ALM	Memory		fMAX(1) (MHz)
		Codec Type	Standard	Input Bits	Engines		M20K	M10K	
Intel Agilex	2	Encoder	LTE	-	1	529	8	-	425
				-	4	900	10	-	425
				-	8	1.4K	19	-	425
				-	-	529	8	-	425
			UMTS	-	1	1.1K	9	-	432
		Decoder	LTE	6	8	12.8K	27	-	414
				8	8	15.2K	32	-	411
				6	16	23.4K	34	-	383
				8	16	28K	40	-	365
			UMTS	6	2	4.9K	80	-	303
				8	2	5.5K	90	-	295
				6	4	7.2K	92	-	304
				8	4	8.1K	97	-	286
Intel Stratix 10	2	Encoder	LTE	-	1	528	8	-	405
				-	4	987	10	-	411
				-	8	1.4K	19	-	381
			UMTS	-	1	1.1K	9	-	330
				-	1	1.1K	9	-	315
		Decoder	LTE	6	8	12.7K	27	-	313
				8	8	15.2K	32	-	313
				6	16	23.2K	34	-	292
				8	16	27.8K	40	-	271
			UMTS	6	2	4.8K	80	-	239
				8	2	5.5K	90	-	230
				6	4	7.1K	92	-	239
				8	4	8.1K	97	-	225

continued...

(1) The table shows f_{MAX} frequency reduced by 15%. The actual average is f_{MAX} x 1.15.

Device	Speed Grade	Parameters				ALM	Memory		fMAX(1) (MHz)
		Codec Type	Standard	Input Bits	Engines		M20K	M10K	
Intel Arria 10	1	Encoder	LTE	-	1	476	2	-	425
				-	4	902	10	-	371
				-	8	1.4K	10	-	371
			UMTS	-	1	1K	4	-	304
		Decoder	LTE	6	8	11K	27	-	323
				8	8	13K	32	-	331
				6	16	19.9K	34	-	311
				8	16	23.8K	40	-	298
			UMTS	6	2	4.7K	71	-	228
				8	2	5.5K	81	-	213
				6	4	7.4K	73	-	225
				8	4	8.4K	78	-	208

1.4. Turbo Code Licensing Disclaimer

France Telecommunication, for itself and certain other parties, claims certain intellectual property rights covering Turbo Codes technology, and has decided to license these rights under a licensing program called the Turbo Codes Licensing Program. Supply of this IP core does not convey a license nor imply any right to use any Turbo Codes patents owned by France Telecommunication, TDF or GET. For information about the Turbo Codes Licensing Program, contact France Telecommunication at the following address:

France Telecommunication R&D

VAT/TURBOCODES

38, rue du Général Leclerc

92794 Issy Moulineaux

Cedex 9

France

1.5. Release Information

Intel FPGA IP versions match the Intel Quartus® Prime Design Suite software versions until v19.1. Starting in Intel Quartus Prime Design Suite software version 19.2, Intel FPGA IP has a new versioning scheme.

(1) The table shows f_{MAX} frequency reduced by 15%. The actual average is f_{MAX} x 1.15.

The Intel FPGA IP version (X.Y.Z) number can change with each Intel Quartus Prime software version. A change in:

- X indicates a major revision of the IP. If you update the Intel Quartus Prime software, you must regenerate the IP.
- Y indicates the IP includes new features. Regenerate your IP to include these new features.
- Z indicates the IP includes minor changes. Regenerate your IP to include these changes.

Table 3. Turbo Intel FPGA IP Release Information

Item	Description
Intel Quartus Prime Version	20.4
IP Version	20.4.0
Release Date	2020.12.14
Ordering Code	IP-TURBO (IPR-TURBO)

Related Information

- Turbo IP Core Release Notes
- Errata for Turbo IP core in the Knowledge Base

2. Getting Started with the Turbo Intel FPGA IP

2.1. Installing and Licensing Intel FPGA IP Cores

The Intel Quartus Prime software installation includes the Intel FPGA IP library. This library provides many useful IP cores for your production use without the need for an additional license. Some Intel FPGA IP cores require purchase of a separate license for production use. The Intel FPGA IP Evaluation Mode allows you to evaluate these licensed Intel FPGA IP cores in simulation and hardware, before deciding to purchase a full production IP core license. You only need to purchase a full production license for licensed Intel IP cores after you complete hardware testing and are ready to use the IP in production.

The Intel Quartus Prime software installs IP cores in the following locations by default:

Figure 1. IP Core Installation Path

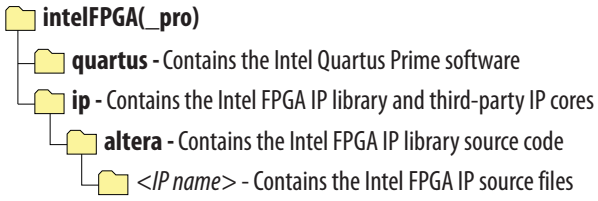


Table 4. IP Core Installation Locations

Location	Software	Platform
`<drive>:\intelFPGA_pro\quartus\ip\altera`	Intel Quartus Prime Pro Edition	Windows*
`<drive>:\intelFPGA\quartus\ip\altera`	Intel Quartus Prime Standard Edition	Windows
`<home directory>:/intelFPGA_pro/quartus/ip/altera`	Intel Quartus Prime Pro Edition	Linux*
`<home directory>:/intelFPGA/quartus/ip/altera`	Intel Quartus Prime Standard Edition	Linux

Note: The Intel Quartus Prime software does not support spaces in the installation path.

2.1.1. Intel FPGA IP Evaluation Mode

The free Intel FPGA IP Evaluation Mode allows you to evaluate licensed Intel FPGA IP cores in simulation and hardware before purchase. Intel FPGA IP Evaluation Mode supports the following evaluations without additional license:

- Simulate the behavior of a licensed Intel FPGA IP core in your system.
- Verify the functionality, size, and speed of the IP core quickly and easily.
- Generate time-limited device programming files for designs that include IP cores.
- Program a device with your IP core and verify your design in hardware.

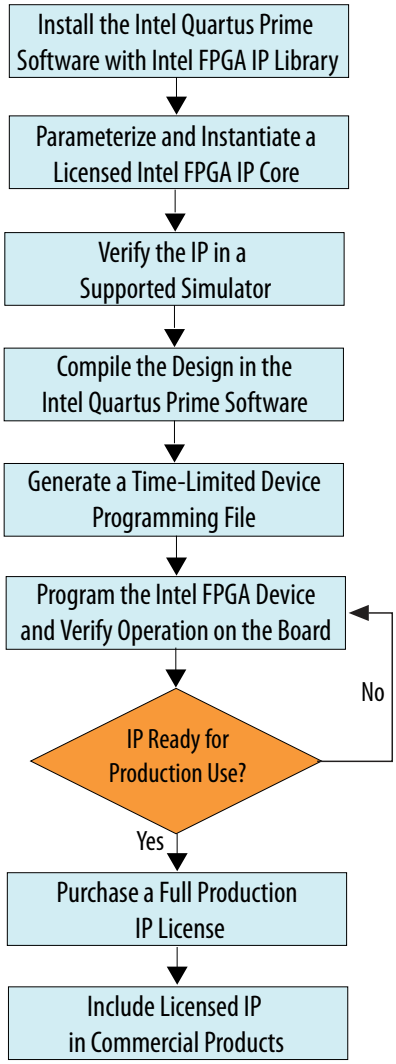
Intel FPGA IP Evaluation Mode supports the following operation modes:

- **Tethered**—Allows running the design containing the licensed Intel FPGA IP indefinitely with a connection between your board and the host computer. Tethered mode requires a serial joint test action group (JTAG) cable connected between the JTAG port on your board and the host computer, which is running the Intel Quartus Prime Programmer for the duration of the hardware evaluation period. The Programmer only requires a minimum installation of the Intel Quartus Prime software, and requires no Intel Quartus Prime license. The host computer controls the evaluation time by sending a periodic signal to the device via the JTAG port. If all licensed IP cores in the design support tethered mode, the evaluation time runs until any IP core evaluation expires. If all of the IP cores support unlimited evaluation time, the device does not time-out.
- **Untethered**—Allows running the design containing the licensed IP for a limited time. The IP core reverts to untethered mode if the device disconnects from the host computer running the Intel Quartus Prime software. The IP core also reverts to untethered mode if any other licensed IP core in the design does not support tethered mode.

When the evaluation time expires for any licensed Intel FPGA IP in the design, the design stops functioning. All IP cores that use the Intel FPGA IP Evaluation Mode time out simultaneously when any IP core in the design times out. When the evaluation time expires, you must reprogram the FPGA device before continuing hardware verification. To extend use of the IP core for production, purchase a full production license for the IP core.

You must purchase the license and generate a full production license key before you can generate an unrestricted device programming file. During Intel FPGA IP Evaluation Mode, the Compiler only generates a time-limited device programming file (<*project name*>`_time_limited.sof`) that expires at the time limit.

Figure 2. Intel FPGA IP Evaluation Mode Flow



Note: Refer to each IP core's user guide for parameterization steps and implementation details.

Intel licenses IP cores on a per-seat, perpetual basis. The license fee includes first-year maintenance and support. You must renew the maintenance contract to receive updates, bug fixes, and technical support beyond the first year. You must purchase a full production license for Intel FPGA IP cores that require a production license, before generating programming files that you may use for an unlimited time. During Intel FPGA IP Evaluation Mode, the Compiler only generates a time-limited device programming file (`<project name>_time_limited.sof`) that expires at the time limit. To obtain your production license keys, visit the Self-Service Licensing Center.

The Intel FPGA Software License Agreements govern the installation and use of licensed IP cores, the Intel Quartus Prime design software, and all unlicensed IP cores.

Related Information

- Intel FPGA Licensing Support Center
- Introduction to Intel FPGA Software Installation and Licensing

2.1.2. Turbo IP Timeout Behavior

All IP cores in a device time out simultaneously when the most restrictive evaluation time is reached. If a design has more than one IP core, the time-out behavior of the other IP cores may mask the time-out behavior of a specific IP core.

For IP cores, the untethered time-out is 1 hour; the tethered time-out value is indefinite. Your design stops working after the hardware evaluation time expires. The Quartus Prime software uses Intel FPGA IP Evaluation Mode Files (`.ocp`) in your project directory to identify your use of the Intel FPGA IP Evaluation Mode evaluation program. After you activate the feature, do not delete these files..When the evaluation time expires, the data output port `reset_n` goes low, which keeps the IP core permanently in its reset state.

Related Information

AN 320: OpenCore Plus Evaluation of Megafunctions

2.2. IP Catalog and Parameter Editor

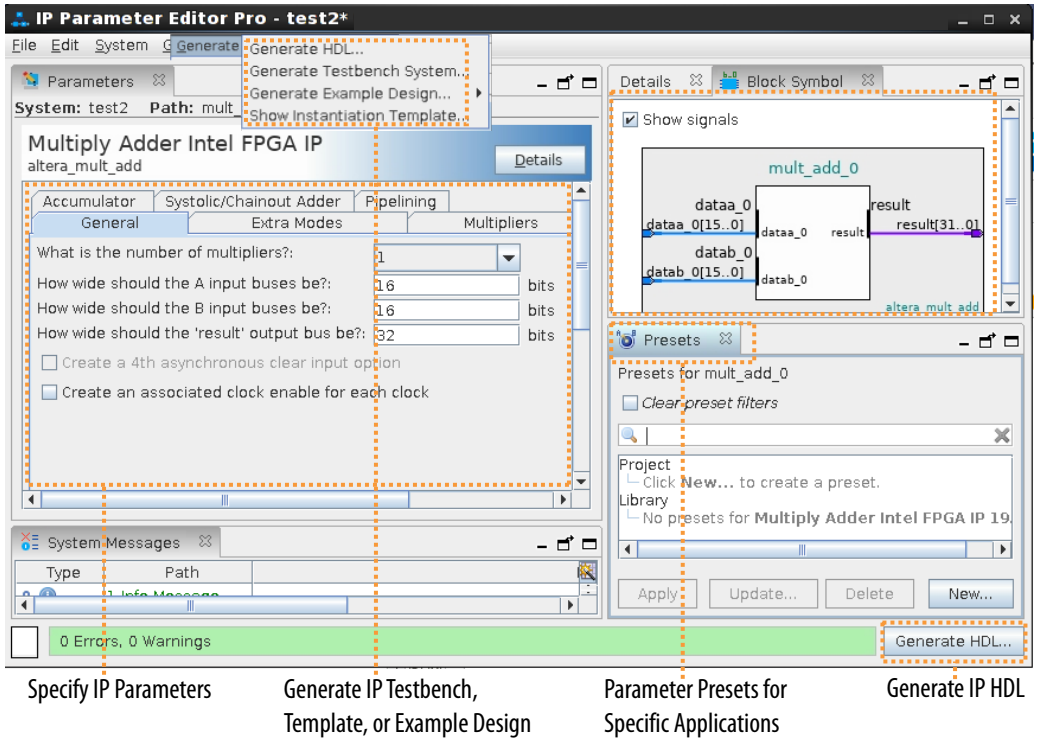
The IP Catalog displays the IP cores available for your project, including Intel FPGA IP and other IP that you add to the IP Catalog search path. Use the following features of the IP Catalog to locate and customize an IP core:

- Filter IP Catalog to **Show IP for active device family** or **Show IP for all device families**. If you have no project open, select the **Device Family** in IP Catalog.
- Type in the Search field to locate any full or partial IP core name in IP Catalog.
- Right-click an IP core name in IP Catalog to display details about supported devices, to open the IP core's installation folder, and for links to IP documentation.
- Click **Search for Partner IP** to access partner IP information on the web.

The parameter editor prompts you to specify an IP variation name, optional ports, and output file generation options. The parameter editor generates a top-level Intel Quartus Prime IP file (`.ip`) for an IP variation in Intel Quartus Prime Pro Edition projects. This file represents the IP variation in the project, and stores parameterization information.[2]

[2] The parameter editor generates a top-level Quartus IP file (`.qip`) for an IP variation in Intel Quartus Prime Standard Edition projects.

Figure 3. Example IP Parameter Editor



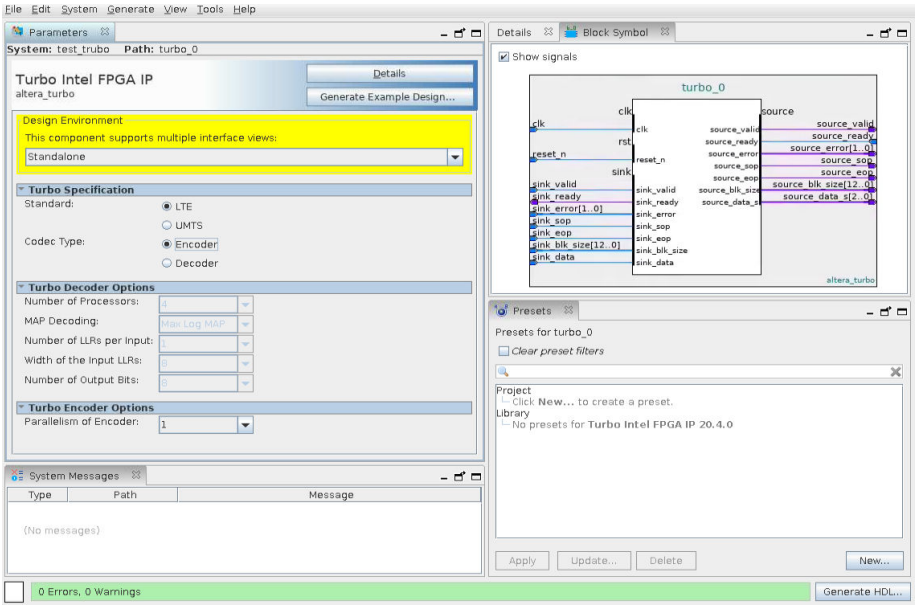
2.3. Specifying the IP Core Parameters and Options

Quickly configure Intel FPGA IP cores in the Intel Quartus Prime parameter editor. Double-click any component in the IP Catalog to launch the parameter editor. The parameter editor allows you to define a custom variation of the IP core. The parameter editor generates the IP variation synthesis and optional simulation files, and adds the `.ip` file representing the variation to your project automatically.

Follow these steps to locate, instantiate, and customize an IP core in the parameter editor:

1. Create or open an Intel Quartus Prime project (`.qpf`) to contain the instantiated IP variation.
2. In the IP Catalog (**Tools ➤ IP Catalog**), locate and double-click the name of the IP core to customize. To locate a specific component, type some or all of the component's name in the IP Catalog search box. The New IP Variation window appears.
3. Specify a top-level name for your custom IP variation. Do not include spaces in IP variation names or paths. The parameter editor saves the IP variation settings in a file named <*your_ip*>`.ip`. Click **OK**. The parameter editor appears.

Figure 4. IP Parameter Editor (Intel Quartus Prime Pro Edition)



4. Set the parameter values in the parameter editor and view the block diagram for the component. The **System Messages** tab at the bottom displays any errors in IP parameters:
   - Optionally, select preset parameter values if provided for your IP core. Presets specify initial parameter values for specific applications.
   - Specify parameters defining the IP core functionality, port configurations, and device-specific features.
   - Specify options for processing the IP core files in other EDA tools.

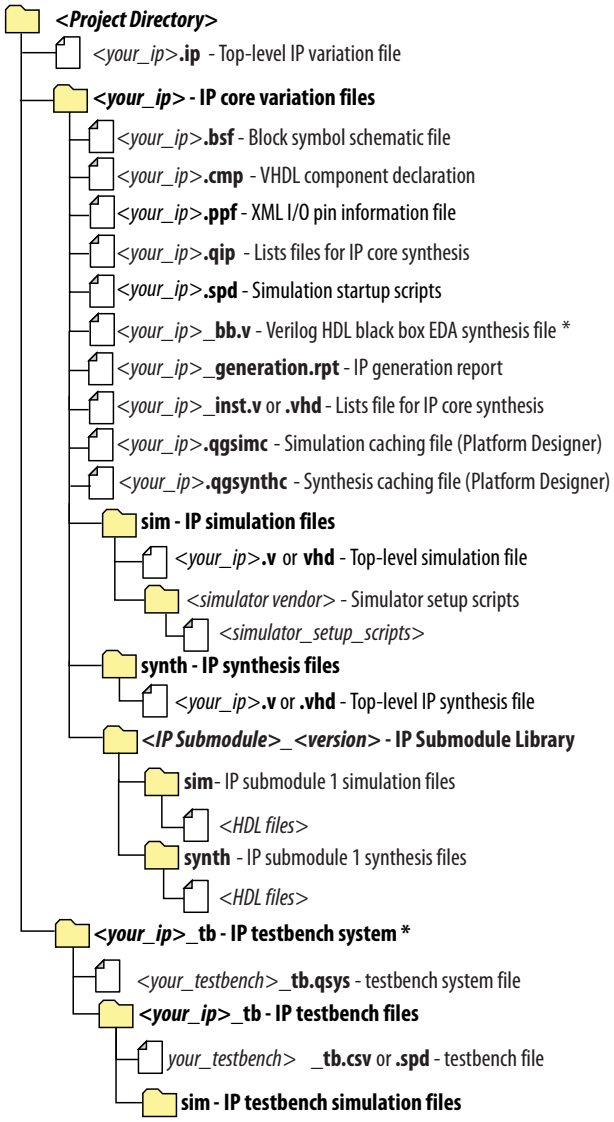
   Note: Refer to your IP core user guide for information about specific IP core parameters.
5. Click **Generate HDL**. The **Generation** dialog box appears.
6. Specify output file generation options, and then click **Generate**. The synthesis and simulation files generate according to your specifications.
7. To generate a simulation testbench, click **Generate ➤ Generate Testbench System**. Specify testbench generation options, and then click **Generate**.
8. To generate an HDL instantiation template that you can copy and paste into your text editor, click **Generate ➤ Show Instantiation Template**.
9. Click **Finish**. Click **Yes** if prompted to add files representing the IP variation to your project.
10. After generating and instantiating your IP variation, make appropriate pin assignments to connect ports.

Note: Some IP cores generate different HDL implementations according to the IP core parameters. The underlying RTL of these IP cores contains a unique hash code that prevents module name collisions between different variations of the IP core. This unique code remains consistent, given the same IP settings and software version during IP generation. This unique code can change if you edit the IP core's parameters or upgrade the IP core version. To avoid dependency on these unique codes in your simulation environment, refer to *Generating a Combined Simulator Setup Script*.

2.3.1. IP Core Generation Output (Intel Quartus Prime Pro Edition)

The Intel Quartus Prime software generates the following output file structure for individual IP cores that are not part of a Platform Designer system.

Figure 5. Individual IP Core Generation Output (Intel Quartus Prime Pro Edition)



* If supported and enabled for your IP core variation.

Table 5. Output Files of Intel FPGA IP Generation

File Name	Description
<*your_ip*>.ip	Top-level IP variation file that contains the parameterization of an IP core in your project. If the IP variation is part of a Platform Designer system, the parameter editor also generates a .qsys file.
<*your_ip*>.cmp	The VHDL Component Declaration (.cmp) file is a text file that contains local generic and port definitions that you use in VHDL design files.
<*your_ip*>_generation.rpt	IP or Platform Designer generation log file. Displays a summary of the messages during IP generation.
<*your_ip*>.qgsimc (Platform Designer systems only)	Simulation caching file that compares the .qsys and .ip files with the current parameterization of the Platform Designer system and IP core. This comparison determines if Platform Designer can skip regeneration of the HDL.
<*your_ip*>.qgsynth (Platform Designer systems only)	Synthesis caching file that compares the .qsys and .ip files with the current parameterization of the Platform Designer system and IP core. This comparison determines if Platform Designer can skip regeneration of the HDL.
<*your_ip*>.qip	Contains all information to integrate and compile the IP component.
<*your_ip*>.csv	Contains information about the upgrade status of the IP component.
<*your_ip*>.bsf	A symbol representation of the IP variation for use in Block Diagram Files (.bdf).
<*your_ip*>.spd	Input file that ip-make-simscript requires to generate simulation scripts. The .spd file contains a list of files you generate for simulation, along with information about memories that you initialize.
<*your_ip*>.ppf	The Pin Planner File (.ppf) stores the port and node assignments for IP components you create for use with the Pin Planner.
<*your_ip*>_bb.v	Use the Verilog blackbox (_bb.v) file as an empty module declaration for use as a blackbox.
<*your_ip*>_inst.v or _inst.vhd	HDL example instantiation template. Copy and paste the contents of this file into your HDL file to instantiate the IP variation.
<*your_ip*>.regmap	If the IP contains register information, the Intel Quartus Prime software generates the .regmap file. The .regmap file describes the register map information of master and slave interfaces. This file complements the .sopcinfo file by providing more detailed register information about the system. This file enables register display views and user customizable statistics in System Console.
<*your_ip*>.svd	Allows HPS System Debug tools to view the register maps of peripherals that connect to HPS within a Platform Designer system. During synthesis, the Intel Quartus Prime software stores the .svd files for slave interface visible to the System Console masters in the .sof file in the debug session. System Console reads this section, which Platform Designer queries for register map information. For system slaves, Platform Designer accesses the registers by name.
<*your_ip*>.v <*your_ip*>.vhd	HDL files that instantiate each submodule or child IP core for synthesis or simulation.
mentor/	Contains a msim_setup.tcl script to set up and run a simulation with a supported Siemens EDA simulator, such as the ModelSim simulator.
aldec/	Contains a Riviera-PRO* script rivierapro_setup.tcl to setup and run a simulation.
/synopsys/vcs /synopsys/vcsmx	Contains a shell script vcs_setup.sh to set up and run a VCS* simulation. Contains a shell script vcsmx_setup.sh and synopsys_sim.setup file to set up and run a VCS MX simulation.

continued...

File Name	Description
/cadence	Contains a shell script ncsim_setup.sh and other setup files to set up and run an NCSim simulation.
/xcelium	Contains an Xcelium* Parallel simulator shell script xcelium_setup.sh and other setup files to set up and run a simulation.
/submodules	Contains HDL files for the IP core submodule.
<IP submodule>/	Platform Designer generates /synth and /sim sub-directories for each IP submodule directory that Platform Designer generates.

2.4. Simulating Intel FPGA IP Cores

The Intel Quartus Prime software supports IP core RTL simulation in specific EDA simulators. IP generation optionally creates simulation files, including the functional simulation model, any testbench (or example design), and vendor-specific simulator setup scripts for each IP core. You can use the functional simulation model and any testbench or example design for simulation. IP generation output may also include scripts to compile and run any testbench. The scripts list all models or libraries you require to simulate your IP core.

The Intel Quartus Prime software provides integration with many simulators and supports multiple simulation flows, including your own scripted and custom simulation flows. Whichever flow you choose, IP core simulation involves the following steps:

1. Generate IP HDL, testbench (or example design), and simulator setup script files.
2. Set up your simulator environment and any simulation scripts.
3. Compile simulation model libraries.
4. Run your simulator.

2.5. Simulating the Turbo IP with the RTL Simulator

Before simulating, generate the Turbo IP design example from the IP parameter editor.

- To run the simulation with Synopsys VCS simulator:
  1. Run `vcsmx_setup.sh` from <*example_design_directory*>`\simulation_scripts\synopsys\vcsmx\` directory by typing the following commands:

```
>> source vcsmx_setup.sh
>> simv
```

- To run the simulation with Cadence NCSim® simulator:
  1. Run `ncsim_setup.sh` from <*example_design_directory*>`\simulation_scripts\cadence\` directory by typing the following command:

```
sh ./ncsim_setup.sh USER_DEFINED_ELAB_OPTIONS='"-timescale 1ps/1ps"'
USER_DEFINED_SIM_OPTIONS='"-input \"@run; exit\""'
```

- To run the simulation with Xcelium simulator:
  1. Run `xcelium_setup.sh` from <*example_design_directory*>`\simulation_scripts\xcelium\` directory by typing the following command:

```
sh ./xcelium_setup.sh USER_DEFINED_ELAB_OPTIONS='"-timescale 1ps/1ps"'
USER_DEFINED_SIM_OPTIONS='"-input \"@run; exit\""'
```

- To run the simulation with the ModelSim or Questa® simulator:
  1. Run `msim_setup.tcl` from <*example_design_directory*>`\simulation_scripts\mentor\` directory by typing the following commands:

```
do msim_setup.tcl
ld
run -all
```

- To run the simulation with Aldec® simulator:
  1. Run `rivierapro_setup.tcl` from <*example_design_directory*>`\simulation_scripts\aldec\` directory by typing the following commands:

```
do rivierapro_setup.tcl
ld
run -all
```

2.6. Simulating the Turbo IP with the C-Model

Before simulating, generate the Turbo IP design example from the IP parameter editor.

Table 6. Turbo C-Model versus Turbo IP Related Parameters

Turbo C-model	Turbo IP	Value
K	Block size *K*, supports LTE standard	-
CW	sink_data (input codeword Ncb)	-
llr_width	**Width of the input LLRs** supports 5,6,7,8.	8
early_ter	Early termination, always set to 1.	1
crc24b	CRC_type, • 0: CRC24A • 1: CRC24B	-
max_subiter	sink_max_iter supports up to 31 (5 bits).	-
nb_eng	**Number of Processors** supports 2, 4, 8, 16, 32.	16
Iter_used	source_iter	Output
crc_pass	CRC_pass	Output
decoded_bits	source_data_s	Output

2.6.1. Encoder Simulation

1. Go to the <*example_design_directory*>\c\ directory.
2. Compile the C code:

   For LTE, type the following command:

   ```
   >> gcc -lm main_lte_enc.c -o run_enc
   ```

   For UMTS, type the following command:

   ```
   >> gcc -lm main_umts_enc.c -o run_enc
   ```

3. Run the executable without arguments.

   ```
   >>./run_enc
   ```

   Note: The executable reads /test_data/ctc_encoder_input.txt and ./test_data/ctc_encoder_input_info.txt as inputs. The executable generates /test_data/ctc_encoder_output_gold.txt as output. The RTL simulation uses the same input .txt files. The output .txt file provides a golden output, which may be used to check the correctness of the output from RTL simulations.

2.6.2. Decoder Simulation

1. Go to the <*example_design_directory*>\c\ directory.
2. Compile the C code:

   For LTE, type the following command:

   ```
   >> gcc -lm main_lte_dec.c -o run_dec
   ```

For UMTS, type the following command:

```
>> gcc -lm main_umts_dec.c -o run_dec
```

3. Run the executable without arguments.

```
>>./run_dec
```

Note: The executable reads `/test_data/ctc_input_data.txt` and `/test_data/ctc_input_info.txt` as inputs. The executable generates `/test_data/ctc_decoded_output_gold.txt` and `ctc_output_et_info_gold.txt`as output. The RTL simulation uses the same input `.txt` files. The output `.txt` file provides a golden output, which may be used to check the correctness of the output from RTL simulations.

2.7. Simulating the Turbo IP with MATLAB

Verify that the RTL behaves the same as these models.

Before simulating, generate the Turbo IP design example from the IP parameter editor.

1. In MATLAB, run MATLAB script from the <*example_design_directory*>`\matlab` directory.

   For encoder simulation:

   - Run the following script for LTE:

     ```
     main_lte_enc
     ```

   - Run the following script for UMTS:

     ```
     main_umts_enc
     ```

   For decoder simulation:

   - Run the following script for LTE:

     ```
     main_lte_dec
     ```

   - Run the following script for UMTS:

     ```
     main_umts_dec
     ```

   Note: MATLAB model reads and generates the same `.txt` files as C model.

3. Parameter Settings

You can customize the Turbo IP by specifying parameters in the IP parameter editor:

Table 7. Parameters

Parameter	Range	Description
Turbo Specification		
Standard	LTE or UMTS	Select LTE or UMTS.
Codec Type	Encoder, Decoder	Select an encoder or decoder.
Turbo Decoder Options		
Number of Processors	2, 4, 8, 16, 32 *Note:* The UMTS supports only 2 and 4, and LTE supports 2, 4, 8, 16, 32.	Select the number of engines (N_{dec}) that the decoder uses. The output width (W_{out}) varies depending on the number of engines: • 8 bits for 8 engines or less, • 16 bits for 16 engines, • 32 bits for 32 engines *Note:* For the UMTS the output width is always 1.
Number of LLRs per input	1 or 2	The number of data LLR per input symbol (N_{LLR}). For two LLR the input symbols are Z'2 Z2 X2 Z'1 Z1 X1 *Note:* This parameter is not available if you select UMTS standard.
Width of the input LLRs	5, 6, 7, 8	The number of input bits (W_{LLR}) to the decoder. Three of the bits are integer bits and the remaining bits are fractional bits: • 5-bit LLRs have two fractional bits. • 6-bit LLRs have three fractional bits. • 7-bit LLRs have four fractional bits. • 8-bit LLRs have five fractional bits.
Turbo Encoder Options		
Parallelism of Encoder	1, 4, 8	Selects the number of parallel encoding engines (N_{enc}) for LTE turbo encoder only. For UMTS turbo encoder the value of this parameter (N_{enc}) is 1. This parameter is not available in the Intel Quartus Prime Standard Edition software IP variations.

4. Turbo Intel FPGA IP Functional Description

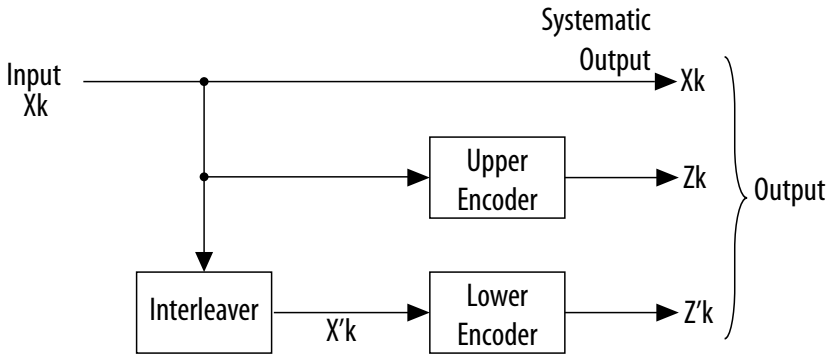
This topic describes the IP core's architecture, interfaces, and signals.

You can parameterize the Turbo IP core as an encoder or a decoder.

4.1. Turbo Encoder

The 3GPP Turbo encoder uses a parallel concatenated convolutional code. A convolutional encoder encodes an information sequence and another convolutional encoder encodes an interleaved version of the information sequence. The Turbo encoder has two 8-state constituent encoders and one Turbo code internal interleaver. The Turbo encoder accepts K bits and outputs 3K+12 bits, having a natural code rate 1/3. The last 12 output bits of every packet are termination bits, which guarantee that the state of the encoder is back to state zero in the end of encoding process.

Figure 6. Turbo Encoder Block Diagram



The output from the turbo coder is:

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

Where:

- Bits $X_0, X_1, \ldots, X_{K-1}$ are input to both the first 8-state constituent encoder and the internal interleaver (K is the number of bits).
- Bits $Z_0, Z_1, \ldots, Z_{K-1}$ and $Z'_0, Z'_1, \ldots, Z'_{K-1}$ 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 $X'_0, X'_1, \ldots, X'_{K-1}$.
- Additionally, encoder outputs 12 termination bits, X_K , X_{K+1}, X_{K+2}, X'_K, X'_{K+1}, X'_{K+2}, Z_K, Z_{K+1}, Z_{K+2}, Z'_K, Z'_{K+1}, Z'_{K+2}.

4.1.1. Turbo Encoder Data Format

The required input data ordering for a block of size K is: X_0, X_1, X_2, ... X_{K-1}. The output data is three bits wide.

Table 8. Turbo Encoder Input Data Ordering for a Block of Size *K*

Output Data	sink_data		
	N_{enc} = 8	N_{enc} = 4	N_{enc} = 1
0	$X_7X_6X_5X_4X_3X_2X_1X_0$	$X_3X_2X_1X_0$	X_0
1	$X_{15}X_{14}X_{13}X_{12}X_{11}X_{10}X_9X_8$	$X_7X_6X_5X_4$	X_1
...	...	...	..
K - 1	-	-	X_{K-1}
K/4-1	-	$X_{K-1}X_{K-2}X_{K-3}X_{K-4}$	-
K/8-1	$X_{K-1}X_{K-2}X_{K-3}X_{K-4}X_{K-5}X_{K-6}X_{K-7}X_{K-8}$	-	-

Table 9. Turbo Encoder Output Data Ordering for a Block of Size *K* and Number of Parallel Encoder = 1

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}

Note: The outputs of the last four clock cycles are corresponding to 12 termination bits.

Table 10. Turbo Encoder Output Data Ordering for a Block of Size *K* and N_{enc} = 4

Note: Parallelism of encoder is only supported in Intel Arria 10, Intel Stratix 10, and Intel Agilex device variations in Intel Quartus Prime Pro Edition software and not supported for Intel Arria 10 device variation in Intel Quartus Prime Standard Edition software.

Output Data	source_data		
	11...8	7...4	3...0
0	$Z'_3Z'_2Z'_1Z'_0$	$Z_3Z_2Z_1Z_0$	$X_3X_2X_1X_0$
1	$Z'_7Z'_6Z'_5Z'_4$	$Z_7Z_6Z_5Z_4$	$X7X_6X_5X_4$
...	...	...	..
K / 4 - 1	$Z'_{K-1}Z'_{K-2}Z'_{K-3}Z'_{K-4}$	$Z_{K-1}Z_{K-2}Z_{K-3}Z_{K-4}$	$X_{K-1}X_{K-2}X_{K-3}X_{K-4}$
K / 4	$Z'_{K+2}X'_{K+1}Z_{K+2}X_{K+1}$	$X'_{K+2}Z'_KX_{K+2}Z_K$	$Z'_{K+1}X'_KZ_{K+1}X_K$

Note: The outputs of the last four clock cycles are corresponding to 12 termination bits.

Table 11. Turbo Encoder Output Data Ordering for a Block of Size *K* and N_{enc} = 8

Note: The value of N_{enc} can be 1, 4, or 8 in the Intel Quartus Prime Pro Edition software and the value is 1 in Intel Quartus Prime Standard Edition software . N_{enc} is only supported in Intel Arria 10, Intel Stratix 10, and Intel Agilex device variations in Intel Quartus Prime Pro Edition software and not supported for Intel Arria 10 device variation in Intel Quartus Prime Standard Edition software.

Output Data	source_data		
	23...16	15...8	7...0
0	$Z'_7Z'_6Z'_5Z'_4Z'_3Z'_2Z'_1Z'_0$	$Z_7Z_6Z_5Z_4Z_3Z_2Z_1Z_0$	$X_7X_6X_5X_4X_3X_2X_1X_0$
1	$Z'_{15}Z'_{14}Z'_{13}Z'_{12}Z'_{11}Z'_{10}Z'_9Z'_8$	$Z_{15}Z_{14}Z_{13}Z_{12}Z_{11}Z_{10}Z_9Z_8$	$X_{15}X_{14}X_{13}X_{12}X_{11}X_{10}X_9X_8$
...	...	...	..
K / 8 - 1	$Z'_{K-1}Z'_{K-2}Z'_{K-3}Z'_{K-4}Z'_{K-5}Z'_{K-6}Z'_{K-7}Z'_{K-8}$	$Z_{K-1}Z_{K-2}Z_{K-3}Z_{K-4}Z_{K-5}Z_{K-6}Z_{K-7}Z_{K-8}$	$X_{K-1}X_{K-2}X_{K-3}X_{K-4}X_{K-5}X_{K-6}X_{K-7}X_{K-8}$
K / 8	4'bx $Z'_{K+2}X'_{K+1}Z_{K+2}X_{K+1}$	4'bx $X'_{K+2}Z'_KX_{K+2}Z_K$	4'bx $Z'_{K+1}X'_KZ_{K+1}X_K$

Note: The outputs of the last clock cycles are corresponding to 12 termination bits. where output bits 23, 22, 21, 20, 15, 14, 13, 12, 7, 6, 5, 4 in the last clock cycle are unused bits.

4.1.2. Turbo Encoder Latency Calculation

The encoding delay *D* is the number of clock cycles the IP core consumes to encode an entire block of data. If *K* is the block size, $D = K/N_{enc} + 14$, where N_{enc}[(3)] is the number of parallel encoder. 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 $K = 6144$, and $N_{enc} = 8$, $D = 6144/8 + 14 = 782$
- When $K = 40$, and $N_{enc} = 1$, $D = 40 + 14 = 54$

You can calculate the encoding latency (the time taken by the encoder to encode an entire block) using the following equation:

$L = D/f$ s

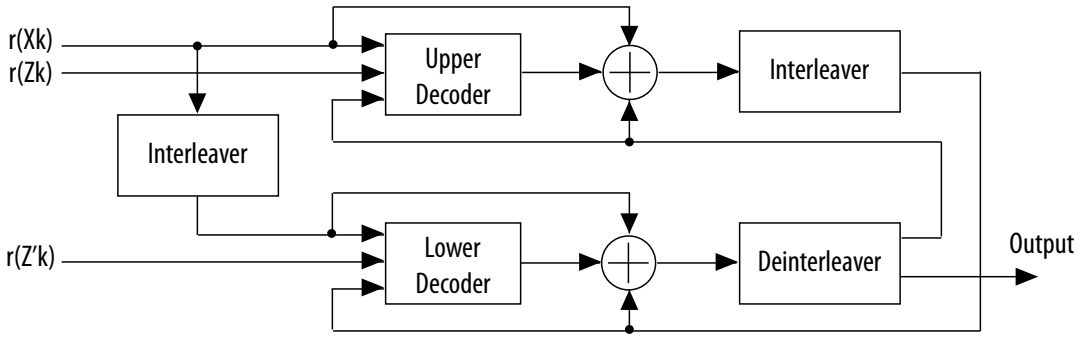
Where f is the system clock speed.

4.2. Turbo Decoder

The 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.

(3) The value of 4 or 8 is only available in Intel Arria 10, Intel Stratix 10, and Intel Agilex device variations in Intel Quartus Prime Pro Edition software.

Figure 7. Turbo Decoder Block Diagram



The Turbo decoder supports the MaxLogMAP decoding algorithm. This algorithm is a simplified version of LogMAP that uses less logic resource and offers slightly reduced BER performance relative to LogMAP.

4.2.1. Turbo Decoder Data Format

Table 12. Turbo Decoder Output Data Ordering for a Block of Size *K*

N is the width of the input LLRs.

Input Data	sink_data		
	3N* - 1 down to *2N	***2N* - 1 down to *N***	***N* - 1 down to 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}

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[P(x=1)/(x=0)]

The log-likelihood value is the logarithm of the probability that the received bit is a 1, divided by the probability that this bit is a 0. It is represented as a two's complement number. A value of zero indicates equal probability of a 1 and a 0, which you should use for depuncturing. The decoder does not use the most negative two's complement number, which means the representation is balanced.

Table 13. Four-bit Mapping Input Values

Input (3 downto 0)	Value
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

The number of output bits can be 1 or 8 bits. For 1 bit, the ordering is: X_0, X_1, X_2, ... X_{K-1}

Table 14. 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}

4.2.2. CRC24A or CRC24B Early Termination

Early termination reduces power consumption and the overall latency, and increases the throughput significantly. It may also increase BER performance of the decoder.

The IP core checks the CRC checksum that the decoder generates after every iteration. Turbo decoding stops as soon as the CRC is successful. turbo decoding does not continue until the maximum number of iterations specified at the input ports. The gains depend on the signal-to-noise ration (SNR) of the received data block, block size, and the maximum number of iterations you specify.

4.2.3. Decoder Latency Calculation

The decoding delay D is the number of clock cycles the IP core consumes to decode an entire block of data.D depends 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 for the worst case latency.

You can calculate the decoding delay D using one of the following equations:

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

where:

- K is the block size
- I is the number of decoding iterations
- N_{dec} is the number of engines specified in the decoder

For LTE,

- $f(K,N_{dec}) = K/N_{dec}$ if K is divisible by N_{dec}.
- $f(K,N_{dec}) = K/16$ if K is not divisible by N_{dec}, but it is divisible by 16.
- $f(K,N_{dec}) = K/8$ in all other conditions.

For UMTS, $f(K,N_{dec}) = \text{ceil}(K/N_{dec})$

For example:

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

You can calculate the decoding latency (the time the decoder takes to decode an entire block to the decoded data is ready for output) using the following equation:

$L = D/f$ s

Where f is the system clock speed.

4.2.4. Error Correction Performance for the Turbo Decoder

Figure 8. BER (bit error rate) Curve of LTE turbo decoder over AWGN channels, where N_{dec}= 8, W_{LLR}= 6, and I= 8

Block sizes (K) are 6144, 4096, 2048, 1024, 512, 256, 128, 64 respectively (from left to right).

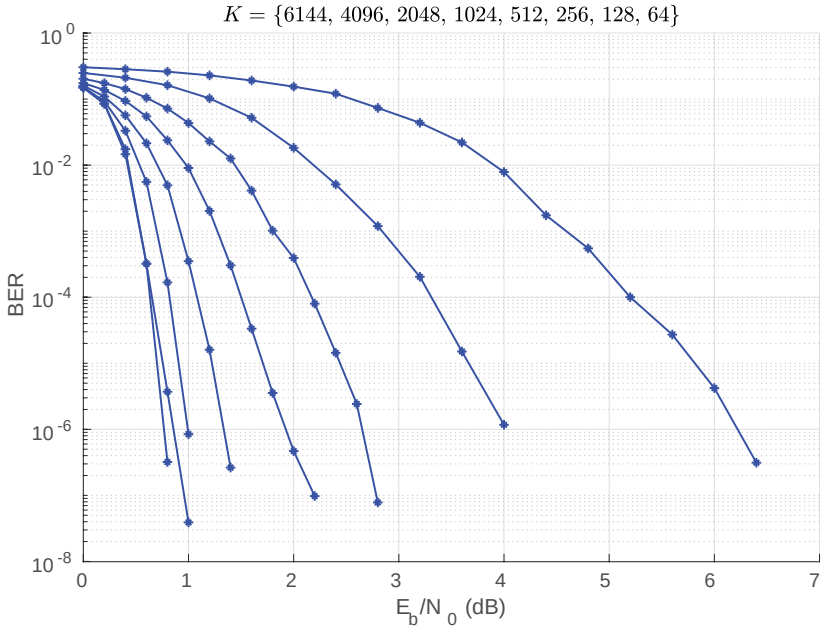


Figure 9. BLER (block error rate) Curve of LTE turbo decoder over AWGN channels, where N_{dec}= 8, W_{LLR}= 6, and I= 8

Block sizes (K) are 6144, 4096, 2048, 1024, 512, 256, 128, 64 respectively (from left to right).

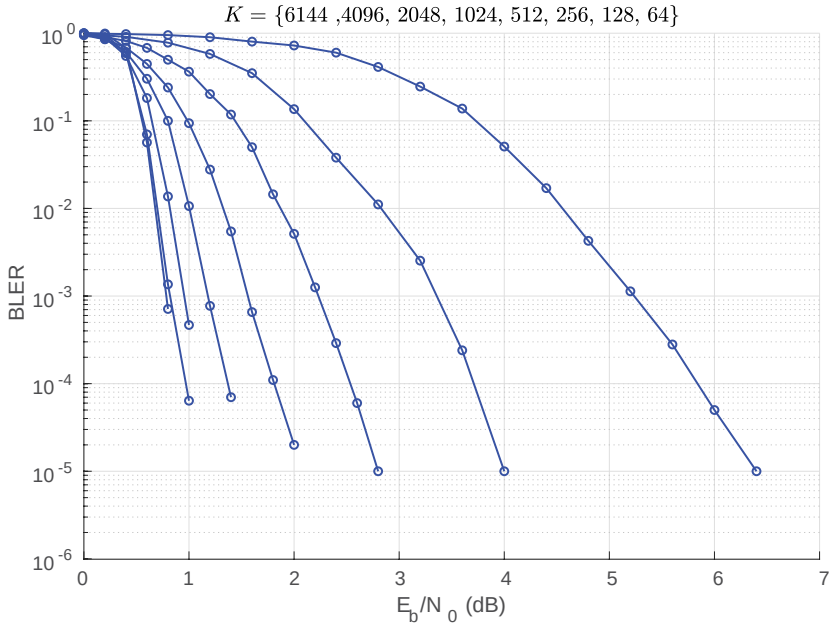


Figure 10. BER Curve of UMTS turbo decoder over AWGN channels, where N_{dec}= 4, W_{LLR}= 6, and I= 8

Block sizes (K) are 5144, 2048, 1024, 512, 256, 128, 64 respectively (from left to right).

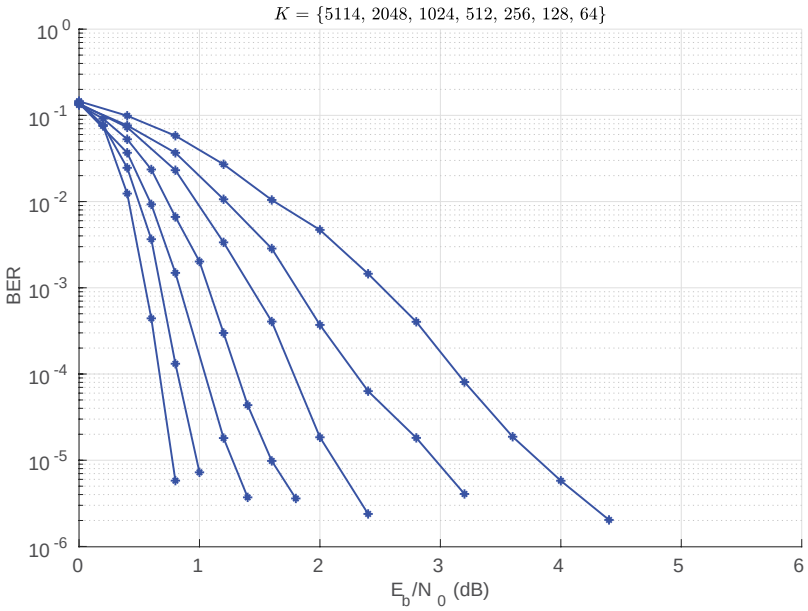
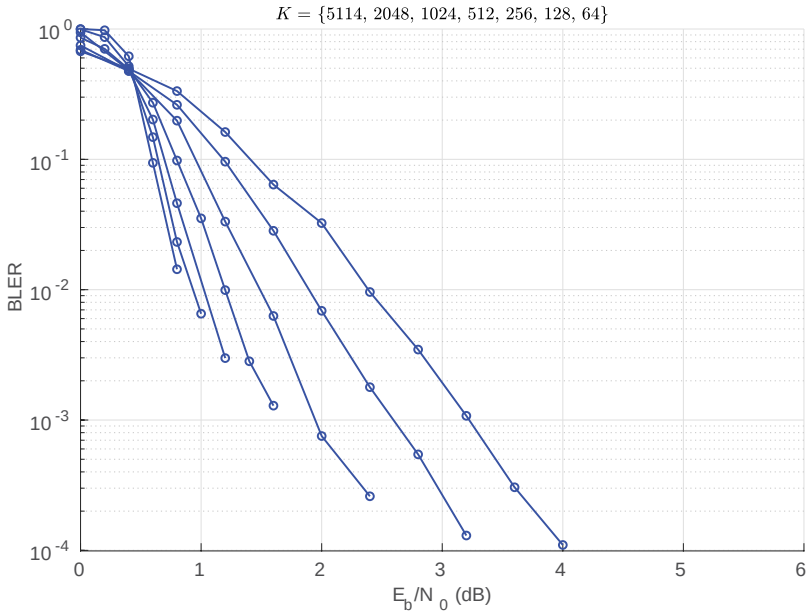


Figure 11. BLER Curve of UMTS turbo decoder over AWGN channels, where N_{dec}= 4, W_{LLR}= 6, and I= 8

Block sizes (K) are 5144, 2048, 1024, 512, 256, 128, 64 respectively (from left to right).



4.3. Turbo IP Core Interfaces and Signals

Signal names beginning with `sink_*` are the input interface to the Turbo IP. Signal names beginning with `source_*` are the output interface from the Turbo IP (excluding `sink_ready` and `source_ready` signals). Both interfaces comply with the Avalon Streaming specification with READY_LATENCY = 0.

Table 15. Turbo Encoder Signals

Signal	Direction	Description
clk	Input	Clock signal that clocks all internal registers.
reset_n	Input	Active low reset signal. The IP core must always be reset before receiving data. If the megafunction is not reset, the Turbo encoder may produce unexpected results because of feedback signals.
sink_blk_size	Input	Specifies the incoming block size. This needs to be held throughout the whole incoming block, from sop to eop.
sink_data	Input	Input data.
sink_eop	Input	Indicates the end of an incoming packet.
sink_sop	Input	Indicates the start of an incoming packet.
sink_valid	Input	Asserted when data at`sink_data` is valid. When you deassert `sink_valid`, the IP core stops processing until you reassert `sink_valid`.
source_ready	Input	Asserted by the downstream module if it cannot accept data.
sink_error	Input	Error signal indicating Avalon-ST protocol violations on input side.
sink_ready	Output	Indicates when the IP core can accept data.
source_blk_size	Output	Specifies the outgoing block size, which is equal to K / N_{enc} + ceil (4 / N_{enc}). Valid at `source_sop`.
source_data_s	Output	Output data.
source_eop	Output	Indicates the end of an outgoing packet.
source_error	Output	Error signal indicating Avalon-ST protocol violations on source side: • 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_sop	Output	Indicates the start of an outgoing packet.
source_valid	Output	Asserted by the IP core when valid data is available to output.

Table 16. Turbo Decoder Signals

Signal	Direction	Description
clk	Input	Clock signal that clocks all internal registers.
reset_n	Input	Active low reset signal. You must always reset the IP core before it receives data. If not reset, the Turbo decoder may produce unexpected results because of feedback signals.

continued...

Signal	Direction	Description
CRC_pass	Output	Indicates whether CRC was successful: • 0: Fail • 1: Pass *Note:* This signal is present only in LTE mode.
CRC_type	Output	Indicates the type of CRC that was used for the current data block: • 0: CRC24A • 1: CRC24B *Note:* This signal is present only in LTE mode.
sel_CRC24A	Input	Specifies the type of CRC that you need for the current data block: • 0: CRC24A • 1: CRC24B *Note:* This signal is present only in LTE mode.
sink_blk_size	Input	Specifies the incoming block size. This signal needs to be held throughout the whole incoming block from SOP to EOP.
sink_data	Input	Input data.
sink_eop	Input	Indicates the end of an incoming packet.
sink_error	Input	Error signal indicating Avalon-ST protocol violations on input side.
sink_max_iter	Input	Specifies the maximum number of half-iterations. It must be even numbers for UMTS.
sink_ready	Output	Indicates when the IP core can accept data.
sink_sop	Input	Indicates the start of an incoming packet.
sink_valid	Input	Assert when data at `sink_data` is valid. When `sink_valid` is not asserted, processing stops until you reassert `sink_valid`.
source_blk_id	Output	Specifies the outgoing block ID.
source_blk_size	Output	Specifies the outgoing block size.
source_data_s	Output	Output data.
source_eop	Output	Indicates the end of an outgoing packet.
source_error	Output	Error signal indicating Avalon-ST protocol violations on source side: • 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_iter	Output	Shows the number of half iterations after which the Turbo decoder stops processing the current data block. LTE only.
source_ready	Input	Asserted by the downstream module if it can accept data.
source_sop	Output	Indicates the start of an outgoing packet.
source_valid	Output	Asserted by the IP core when there is valid data to output.

Signals in Platform Designer (Standard) Systems

Platform Designer (Standard) systems instantiate all Turbo IP core signals as part of the Avalon-ST data bus.

Table 17. Turbo Encoder Data Input

Bits	Signal
N_{enc} + 12:N_{enc}	sink_blk_size
N_{enc} -1:0	sink_data

Table 18. Turbo Encoder Data Output

Bits	Signal
3 *N_{enc}+12:3*N_{enc}	source_blk_size
3*N_{enc}-1:0	source_data

Table 19. Turbo Decoder Data Input

IW is the number of bits for input precision.

Bits	Signal
3*W_{LLR}*N_{LLR}+18	Sel_CRC24A (LTE only)
3*W_{LLR}*N_{LLR}+17:3*W_{LLR}*N_{LLR}+13	sink_max_iter
3*W_{LLR}*N_{LLR}+12:3*W_{LLR}*N_{LLR}	sink_blk_size
3*W_{LLR}*N_{LLR}-1:0	sink_data

Table 20. LTE Turbo Decoder Data Output

Bits	Signal
W_{out}+19	CRC_Pass
W_{out}+18	CRC_type
W_{out}+17:W_{out}+13	source_iter
W_{out}+12:W_{out}	source_blk_size
W_{out}-1:0	source_data

Table 21. UMTS Turbo Decoder Data Output

Bits	Signal
13:1	source_blk_size
0	source_data

4.3.1. Packet Format Errors

The Turbo IP interface complies with Avalon Streaming protocol. The Turbo IP is able to detect the following malformed Avalon Streaming sink packets that are mistakenly fed to the IP. If the IP detects an error , the current packet gets dropped and the output port `source_error` outputs the corresponding error code a few cycles later. Then, the IP continues to process the next available packet. Please note that the IP may not be able to detect complex error cases correctly. For instance, two or more of the following cases occur at the same time in a packet. Therefore, it is always recommended to protect the IP against malformed input packets when use in production.

1. Error code "01" indicates missing `sink_sop`.
2. Error code "10" indicates missing `sink_eop`.
3. Error code "11" indicates unexpected `sink_sop` or `sink_eop`.
4. Default "00" indicates no errors.

The IP input port `sink_error` accepts error code provided from the upstream block. However, a non-zero value from `sink_error` does not result in the current packet being dropped by the IP immediately. Instead the IP always detects errors itself. If IP founds an error, the current packet processed as normal and `source_error` is "00". If IP founds an error, the `source_error` signal is a combination of `sink_error` and the error detected by the IP itself, as follows:

1. If `error_found`= 00, `sink_error`= any, then the `source_error`= 00.
2. If `error_found`= 01, `sink_error`= 00 or 01, then the `source_error`= 01
3. If `error_found`= 01, `sink_error`= 10 or 11, then the `source_error`= 11.
4. If `error_found`= 10, `sink_error`= 00 or 10, then the `source_error`= 10.
5. If `error_found`= 10, `sink_error`= 10 or 11, then the `source_error`= 11.
6. If `error_found`= 11, `sink_error`= any, then the `source_error`= 11.

4.4. Turbo Throughput

You can calculate the throughput using the following equation:

$T = K/D$ * frequency (bits per second)

Encoder throughput calculation example:

- When K = 6144, N_{enc} = 8, frequency = 300 MHz, Throughput = 6144/782 x 300M = 2.36 Gbps
- When K = 40, N_{enc} = 1, frequency = 300 MHz, Throughput = 40/56 x 300M = 214.3 Mbps

Decoder throughput calculation example:

- When K = 6144, N_{dec} = 8, I = 8, frequency = 300 MHz, Throughput = 6144/13,050 x 300M = 141.2 Mbps
- When K = 40, N_{dec} = 8, I = 8, frequency = 300 MHz, Throughput = 40/410 x 300M = 29.3 Mbps

A. Turbo IP Core User Guide Document Archives

IP versions are the same as the Intel Quartus Prime Design Suite software versions up to v19.1. From Intel Quartus Prime Design Suite software version 19.2 or later, IP cores have a new IP versioning scheme.

If an IP core version is not listed, the user guide for the previous IP core version applies.

Intel Quartus Prime Version	IP Core Version	User Guide
15.1	-	Turbo IP Core User Guide

6. Document Revision History for Turbo Intel FPGA IP User Guide

Table 22. Turbo C-Model versus Turbo IP Related Parameters

Document Version	Intel Quartus Prime Version	IP Version	Changes
2021.09.30	20.4	20.4	Added Questa simulator.
2021.03.28	20.4	20.4	Added *Turbo C-Model versus Turbo IP Related Parameters* table
2021.03.04	20.4	20.4	Added extra detail to **Width of the input LLRs** parameter.
2020.12.14	20.4	20.4.0	• Implemented Intel rebranding. • Renamed the Turbo IP to Turbo Intel FPGA IP. • Removed instances of DSP Intel FPGA IP throughout the document. • Made following changes in *About the Turbo Intel FPGA IP* chapter: — Added overview about the Turbo Intel FPGA IP. — Updated features in *Turbo Intel FPGA IP Features* section. — Added support for Intel Stratix 10 and Intel Agilex device family. — Updated resource utilization numbers in *Table: Performance and Resource Utilization*. — Updated *Release Information* topic. • Made following changes in *Getting Started with the Turbo Intel FPGA IP* chapter: — Added new topic *Installing and Licensing Intel FPGA IP Cores* and removed *Licensing IP Cores* and *OpenCore Plus IP Evaluation* topics. — Renamed and updated *Generating IP Cores* topic to *Specifying the IP Core Parameters and Options*. — Added following new topics: • *Simulating the Turbo IP with the RTL Simulator* • *Simulating the Turbo IP with the C-Model* • *Simulating the Turbo IP with MATLAB* • Added new chapter *Parameter Settings*.

continued...

Document Version	Intel Quartus Prime Version	IP Version	Changes
			• Added new parameters **Parallelism of Encoder** and **Number of LLRs per input**. • Made following changes in *Turbo Intel FPGA IP Functional Description* chapter: — Updated description in *Turbo Encoder* section. — Updated *Turbo Encoder Data Format* section. — Added examples of latency calculation in *Turbo Encoder Latency Calculation* section. — Modified signal from *source_data* to *source_data_s*. — Added new signal `source_blk_size` in *Table: Turbo Decoder Signals*. — Updated *Turbo Throughput* with new examples. • Added new section *Parameter Optimization for the Turbo Decoder*.

Date	Version	Changes
2017.11.06	17.1	• Added 16 and 32 to **Number of processors** parameter • Added **Number of LLRs per inputt** parameter • Added support for Intel Cyclone 10 devices • Removed product ID and vendor ID.
2016.05.06	16.0	Corrected features list.
2015.11.11	15.1	• Corrected performance table M20K and ALM entries. • Updated decoder block diagram • Removed Arria 10 performance table entry
2015.11.01	15.1	Initial release