5G LDPC Intel® FPGA IP User Guide

Updated for Intel® Quartus® Prime Design Suite: **21.1**

IP Version: **21.1.0**

Online Version
Send Feedback

UG-20057

ID: **683107**
Version: **2021.03.29**

Contents

1. About the 5G LDPC Intel® FPGA IP

Low-density parity-check (LDPC) codes are linear error correcting codes that help you to transmit and receive messages over noisy channels. The 5G LDPC Intel® FPGA IP implements LDPC codes compliant with the 3rd Generation Partnership Project (3GPP) 5G specification for integration in your wireless design.

LDPC codes replace Turbo codes, popular in 3G and 4G wireless cellular communications. LDPC codes offer better spectral efficiency and support the high throughput for 5G new radio (NR).

Related Information

- Introduction to Intel FPGA IP Cores
  Provides general information about all Intel FPGA IP cores, including parameterizing, generating, upgrading, and simulating IP cores.
- Creating Version-Independent IP and Qsys Simulation Scripts
  Create simulation scripts that do not require manual updates for software or IP version upgrades.
- Project Management Best Practices
  Guidelines for efficient management and portability of your project and IP files.
- 3GPP New Radio Specification
  The final equivalents are Release 15, 3GPP Technical Specification Group RAN 1, NR:
  - (1) Multiplexing and channel coding, 3GPP TS 38.212 (v15.3.0)
  - (2) Physical layer procedures for data, 3GPP TS 38.214 (v15.3.0)

1.1. 5G LDPC Intel FPGA IP Features

The 5G LDPC IP offers the following features:

- 3GPP 5G LDPC specification compliant
- For the decoder:
  - Improved block error rate (BLER) performance
  - Improved power efficiency of IP
  - Per-block modifiable code block length, code rate, base graph, and maximum number of iterations
  - Configurable input precision
  - Layered decoder scheduling architecture to double the speed of convergence compared to non-layered architecture
  - Early termination based on syndrome check on each iteration
  - Single or dual decoders
- For the encoder: per-block modifiable code block length and code rate
- No external memory requirement
- MATLAB and C++ models for performance simulation and RTL test vector generation
- Verilog HDL testbench option
- Avalon®streaming input and output interfaces

Related Information

- 3GPP New Radio LDPC Specification
- Avalon Streaming Interface Specifications

1.2. 5G LDPC Intel FPGA IP Device Family Support

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

- 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 core 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 meets all functional and timing requirements for the device family. You can use it in production designs.

Table 1. 5G LDPC IP Device Family Support

Device Family	Support
Intel Agilex™	Advance
Intel Arria® 10	Final
Intel Stratix® 10	Final
Other device families	No support

1.3. 5G LDPC IP Release Information

Table 2. Release Information

Item	Description
Version	21.1.0
Release Date	2021.05.04
Ordering Code	IP-5G-LDPC

Intel verifies that the current version of the Intel Quartus® Prime software compiles the previous public version of each IP. Intel does not verify that the Intel Quartus Prime software compiles IP versions older than the previous version. The *Intel FPGA IP Release Notes* lists any exceptions.

Related Information

Intel FPGA IP Release Notes

1.4. 5G LDPC IP Performance and Resource Utilization

Intel generated the resource utilization and performance for the encoder and the decoder by compiling the designs with Intel Quartus Prime software v21.1. Only use these approximate results for early estimation of FPGA resources (e.g. adaptive logic modules (ALMs)) that a project requires.

Decoder

Table 3. Resource Utilization and Maximum Frequency for Intel Agilex Devices

Targeting AGFB014R24B2I2V device. The f_{MAX} is the five-seeds average f_{MAX} reduced 15% for margins.

IN_WIDTH	NUM_DECODER	MAX_LF_DECODER0	MAX_LF_DECODER1	ALMs	M20K Memory Blocks	f_{MAX} (MHz)
6	1	384	-	58,720	762	548
6	1	192	-	30,082	402	572
6	1	128	-	19,898	281	574
6	1	96	-	15,604	221	574
6	2	-	192	88,696	1165	556
6	2	-	128	78,828	1044	549
6	2	-	96	74,520	984	550
5	1	384	-	53,588	677	561

continued...

IN_WIDTH	NUM_DECODER	MAX_LF_DECODER0	MAX_LF_DECODER1	ALMs	M20K Memory Blocks	f_{MAX} (MHz)
5	1	192	-	27,420	367	572
5	1	128	-	18,314	258	577
5	1	96	-	14,134	203	576
5	2	-	192	80,494	1045	555
5	2	-	128	71,751	936	541
5	2	-	96	67,565	881	547

Table 4. Resource Utilization and Maximum Frequency for Intel Arria 10 Devices

Targeting 10AT115S1F45E1SG device. The f_{MAX} is the five-seeds average f_{MAX} reduced 15% for margins.

IN_WIDTH	NUM_DECODER	MAX_LF_DECODER0	MAX_LF_DECODER1	ALMs	M20K Memory Blocks	f_{MAX} (MHz)
6	1	384	-	47,261	762	306
6	1	192	-	24,698	402	339
6	1	128	-	17,015	281	348
6	1	96	-	13,381	221	360
6	2	-	192	72,690	1165	290
6	2	-	128	64,925	1044	294
6	2	-	96	61,238	984	296
5	1	384	-	41,906	677	312
5	1	192	-	21,654	367	339
5	1	128	-	15,036	258	353
5	1	96	-	11,842	203	353
5	2	-	192	63,292	1045	305
5	2	-	128	56,714	936	313
5	2	-	96	53,658	881	318

Table 5. Resource Utilization and Maximum Frequency for Intel Stratix 10 Devices

Targeting 1SG280HU2F50E2VG device. The f_{MAX} is the five-seeds average f_{MAX} reduced 15% for margins.

IN_WIDTH	NUM_DECODER	MAX_LF_DECODER0	MAX_LF_DECODER1	ALMs	M20K Memory Blocks	f_{MAX} (MHz)
6	1	384	-	62,090	762	384
6	1	192	-	31,889	402	413
6	1	128	-	21,772	281	409
6	1	96	-	16,838	221	419
6	2	-	192	93,439	1165	365
6	2	-	128	83,526	1044	365
6	2	-	96	79,157	984	374
5	1	384	-	56,806	677	369

continued...

IN_WIDTH	NUM_DECODER	MAX_LF_DECODER0	MAX_LF_DECODER1	ALMs	M20K Memory Blocks	f_{MAX} (MHz)
5	1	192	-	29,469	367	412
5	1	128	-	20,063	258	411
5	1	96	-	15,690	203	433
5	2	-	192	84,685	1045	378
5	2	-	128	75,905	936	383
5	2	-	96	71,549	881	373

Table 6. Resource Utilization and Maximum Frequency for Intel Stratix 10-L Devices

Targeting 1SG280HU2F50E2LG device. The f_{MAX} is the five-seeds average f_{MAX} reduced 15% for margins.

IN_WIDTH	NUM_DECODER	MAX_LF_DECODER0	MAX_LF_DECODER1	ALMs	M20K Memory Blocks	f_{MAX} (MHz)
6	1	384	-	62,297	762	379
6	1	192	-	31,849	402	394
6	1	128	-	21,715	281	403
6	1	96	-	16,812	221	405
6	2	-	192	93,144	1165	359
6	2	-	128	83,298	1044	360
6	2	-	96	78,968	984	368
5	1	384	-	56,768	677	378
5	1	192	-	28,754	367	403
5	1	128	-	20,038	258	401
5	1	96	-	15,684	203	399
5	2	-	192	85,326	1045	367
5	2	-	128	76,386	936	381
5	2	-	96	71,937	881	367

Encoder

Intel generated the data for the following devices:

- Intel Agilex AGFA014R24A2E2VR0
- Intel Arria 10 10AT115S1F45E1SG
- Intel Stratix 10 1SG280HU2F50E2VG
- Intel Stratix 10-L 1SG280HU2F50E2LG

Table 7. Resource Utilization and Maximum Frequency

The f_{MAX} is the five-seeds average f_{MAX} reduced 15% for margins.

Device	ALMs	M20K Memory Blocks	f_{MAX} (MHz)
AGFB014R24B2I2V	17,629	23	554
10AT115S1F45E1SG	16,876	23	323
1SG280HU2F50E2VG	17,617	23	442
1SG280HU2F50E2LG	17,585	23	382

2. Getting Started with the 5G LDPC Intel FPGA IP

Related Information

- Introduction to Intel FPGA IP
- IP Catalog and Parameter Editor
  The IP Catalog displays the IP available for your project.
- Generating Intel FPGA IP
  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.

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

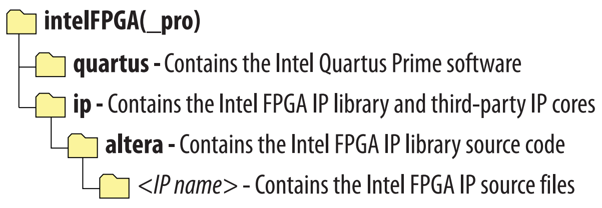

intelFPGA(_pro)
quartus - Contains the Intel Quartus Prime software
ip - Contains the Intel FPGA IP library and third-party IP cores
altera - Contains the Intel FPGA IP library source code
<IP name> - Contains the Intel FPGA IP source files

Table 8. 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

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.

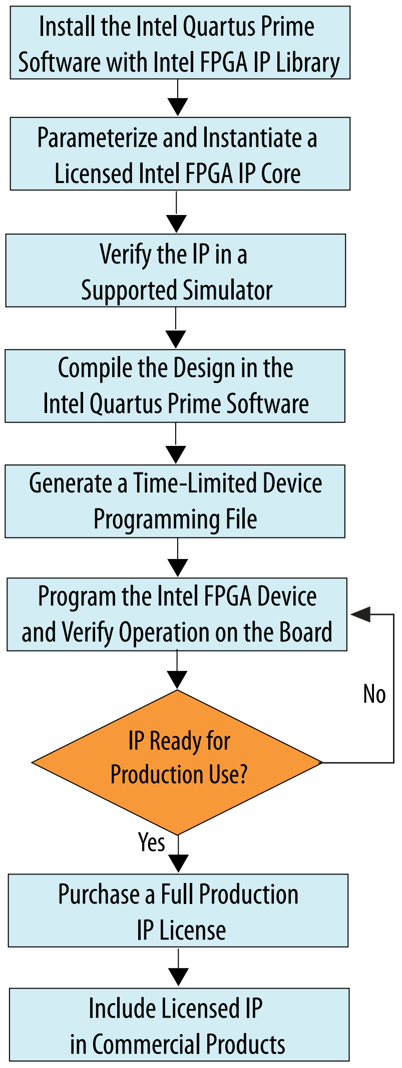
Intel FPGA IP Evaluation Mode supports the following operation modes:

- **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


Install the Intel Quartus Prime Software with Intel FPGA IP Library
Parameterize and Instantiate a Licensed Intel FPGA IP Core
Verify the IP in a Supported Simulator
Compile the Design in the Intel Quartus Prime Software
Generate a Time-Limited Device Programming File
Program the Intel FPGA Device and Verify Operation on the Board
IP Ready for Production Use?
No
Yes
Purchase a Full Production IP License
Include Licensed IP in Commercial Products

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. 5G LDPC IP IP Timeout Behavior

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

For IP, 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, for the encoders `cw` goes low and `rst_n` goes low; for decoders `source_data` goes low, `reset_n` goes low.

Related Information

AN 320: OpenCore Plus Evaluation of Megafunctions

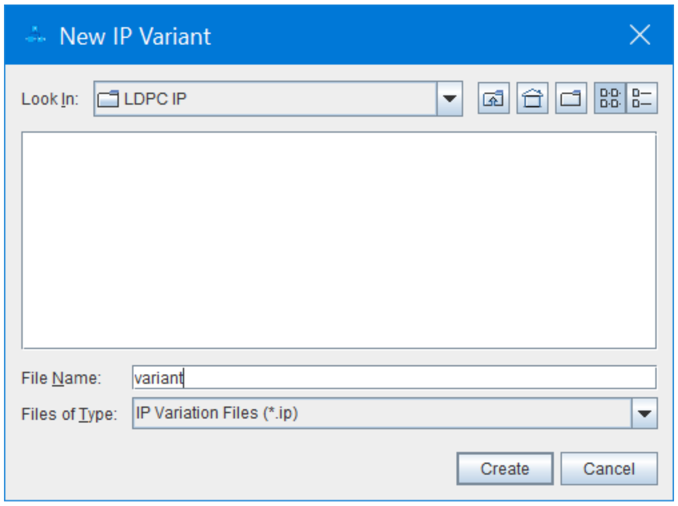
3. Designing with the 5G LDPC Intel FPGA IP

3.1. Generating a 5G LDPC Intel FPGA IP

To include the IP in a design, generate the IP in the Intel Quartus Prime software. Or optionally, you can generate a design example that includes the generated 5G LDPC IP, a C++ model, a MATLAB model, simulation scripts, and test data.

1. Create a New Intel Quartus Prime project
2. Open IP Catalog.
3. Select **DSP ➤ Error Detection and Correction ➤ 5G LDPC Intel FPGA IP** and click **Add**
4. Enter a name for your IP variant and click **Create**.

Figure 3. IP Variant File Name



New IP Variant

Look In: LDPC IP

File Name: variant

Files of Type: IP Variation Files (*.ip)

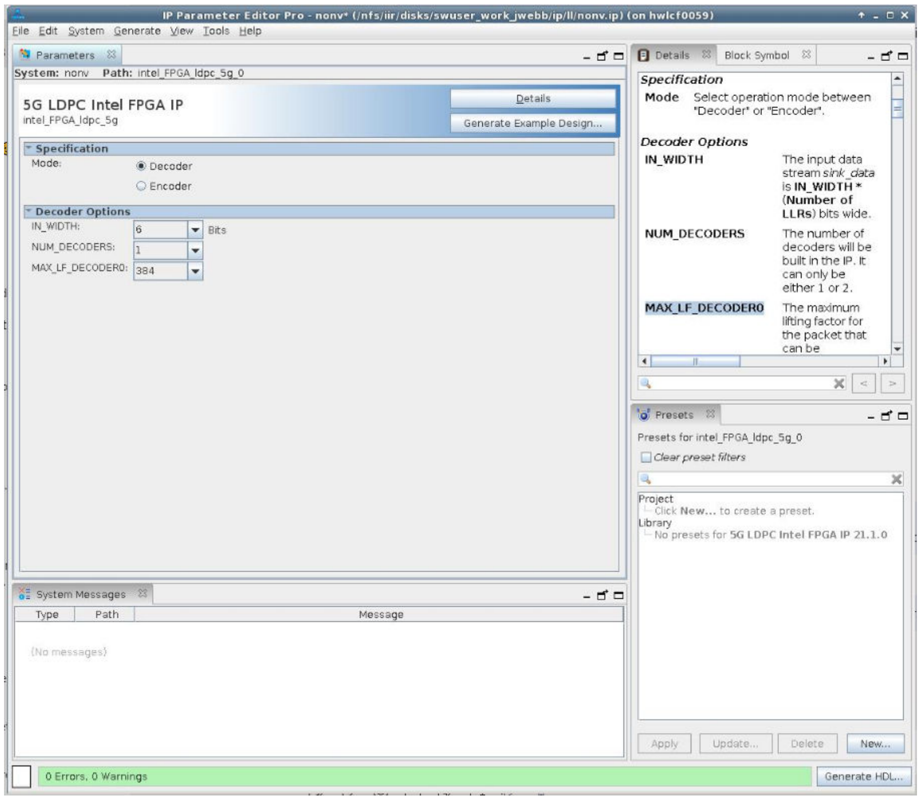
The name is for both the top-level RTL module and the corresponding `.ip` file.

The parameter editor for this IP appears.

5. Choose your parameters.

   You can choose **Encoder** or **Decoder** and select decoder parameters.

Figure 4. 5G LDPC Parameter Editor



IP Parameter Editor Pro - nonv* (/nfs/iir/disks/swuser_work_jwebb/ip/ll/nonv.ip) (on hwlcf0059)

System: nonv Path: intel_FPGA_ldpc_5g_0

5G LDPC Intel FPGA IP
intel_FPGA_ldpc_5g

Specification
Mode: [✓ Decoder] Encoder

Decoder Options
IN_WIDTH: 6 Bits
NUM_DECODERS: 1
MAX_LF_DECODER0: 384

Details

Specification

Mode	Select operation mode between "Decoder" or "Encoder".

Decoder Options

IN_WIDTH	The input data stream *sink_data* is **IN_WIDTH * (Number of LLRs)** bits wide.
NUM_DECODERS	The number of decoders will be built in the IP. It can only be either 1 or 2.
MAX_LF_DECODER0	The maximum lifting factor for the packet that can be

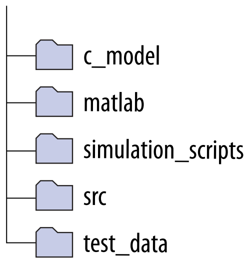
Presets for intel_FPGA_ldpc_5g_0
Clear preset filters
Project
Click **New...** to create a preset.
Library
No presets for **5G LDPC Intel FPGA IP 21.1.0**

System Messages
(No messages)

0 Errors, 0 Warnings

6. For an optional design example, click **Generate Example Design**.
   The software creates files for MATLAB, C, or RTL simulations in the target directory. The software does not generate a hardware example.

Figure 5. Design Example Directory Structure



7. Click **Generate HDL**.

Intel Quartus Prime generates the RTL and the files necessary to instantiate the IP in your design and synthesize it.

Related Information

5G LDPC IP Decoder Parameters on page 27

3.2. Simulating the 5G LDPC IP RTL

Verify that the RTL behaves the same as these models.

Before simulating, generate a 5G LDPC design example.

1. For Synopsys VCS, in the directory `/simulation_scripts/synopsys/vcsmx`:
   a. Execute `source vcsmx_setup.sh`.
      After the compilation finishes, it will run the simulation and finish in 100 ps as the default set in the `vcsmx_setup.sh`.
   b. Execute `./simv`
      VCS starts to simulate. At the end of the simulation, a script compares the decoder output with the expected output (decoder only) and you see `Simulation passed`.
2. For Mentor ModelSim, in the directory `/simulation_scripts/mentor`:
   a. Execute `vsim`.
   b. In the GUI, execute `source msim_setup.tcl`.
   c. Execute `ld` to compile the Intel Quartus Prime simulation library, IP design, and testbench files
   d. Execute `run -all` to start the simulation.

   At the end of the simulation, a script compares the decoder output with the expected output (decoder only) and you see `Simulation passed`.

   .
3. For Cadence NCSim, in the directory `/simulation_scripts/cadence`, execute `source ncsim_setup.sh`.
   At the end of the simulation, a script compares the decoder output with the expected output (decoder only) and you see `Simulation passed`.

Related Information

- Running the 5G LDPC IP C++ Models on page 36
- Running the 5G LDPC Decoder and Encoder MATLAB Model in the Design Example on page 37

3.3. 5G LDPC Simulation Results

Encoder Simulation Results

Figure 6. Input Message to the Encoder

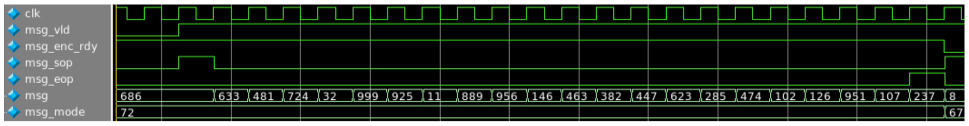


Table 9. Decimal Interpretations of the Bit Groups in the Input Message to the Encoder

686 is the first word and 237 is the last word in the input message.

msg_mode	msg_vld	msg	msg_sop	msg_eop
72	HIGH	686	HIGH	LOW
72	HIGH	633	LOW	LOW
72	HIGH	481	LOW	LOW
72	HIGH	724	LOW	LOW
72	HIGH	32	LOW	LOW
72	HIGH	999	LOW	LOW
72	HIGH	925	LOW	LOW
72	HIGH	11	LOW	LOW
72	HIGH	899	LOW	LOW
72	HIGH	956	LOW	LOW
72	HIGH	146	LOW	LOW
72	HIGH	463	LOW	LOW
72	HIGH	382	LOW	LOW
72	HIGH	447	LOW	LOW
72	HIGH	623	LOW	LOW
72	HIGH	285	LOW	LOW
72	HIGH	474	LOW	LOW
72	HIGH	102	LOW	LOW
72	HIGH	126	LOW	LOW
72	HIGH	951	LOW	LOW
72	HIGH	107	LOW	LOW
72	HIGH	237	LOW	HIGH
67	LOW			LOW

Figure 7. Data and Control Signals at the Start of the Encoder Codeword Output

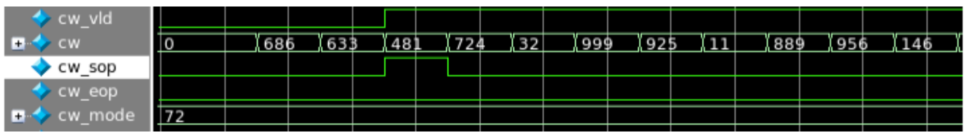


Table 10. Data and Control Signals at the Start of the Encoder Codeword Output

The codeword consists of the initial message with 2*Z starting bits punctured out, and of a block of parity bits appended after the message. The table shows that the first two data words 686 and 633 are punctured out, by deasserting `cw_vld` and also by asserting the SOP signal. 481 is the first data word in the codeword.

cw_vld	cw_mode	cw	parity bits	cw_sop	cw_eop
LOW	72	686		LOW	LOW
LOW	72	633		LOW	LOW
HIGH	72	481		HIGH	LOW
HIGH	72	724		LOW	LOW
HIGH	72	32		LOW	LOW
HIGH	72	999		LOW	LOW
HIGH	72	925		LOW	LOW
HIGH	72	11		LOW	LOW
HIGH	72	899		LOW	LOW
HIGH	72	956		LOW	LOW
HIGH	72	146		LOW	LOW

Figure 8. End of the Encoder Codeword Output

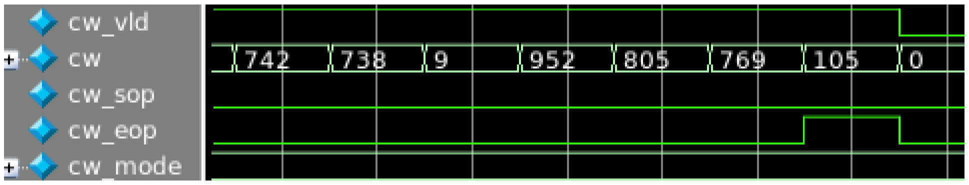


Table 11. Data and Control Signals at the End of the Encoder Codeword

cw_vld	cw_mode	cw	parity bits	cw_sop	cw_eop
HIGH	72	742	parity	LOW	LOW
HIGH	72	738	Parity	LOW	LOW
HIGH	72	9	parity	LOW	LOW
HIGH	72	952	parity	LOW	LOW
HIGH	72	805	parity	LOW	LOW
HIGH	72	769	parity	LOW	LOW
HIGH	72	105	parity	LOW	HIGH
LOW	Don’t care	Don’t care	Don’t care	LOW	LOW

Decoder Simulation Results

Figure 9. Codeword LLR Array Input to the Decoder

- `sink_valid` remains asserted for the duration of the LLR codeword while input data is valid. The upstream component can deassert it as needed.
- `sink_sop` is asserted only for the first data block in. It gets stretched for more than one clock cycle by the backpressure from the decoder, because the decoder deasserts `sink_ready`
- `sink_eop` pulses high for one clock cycle on the last valid input data block. It also can be stretched by the backpressure from the decoder's sink.

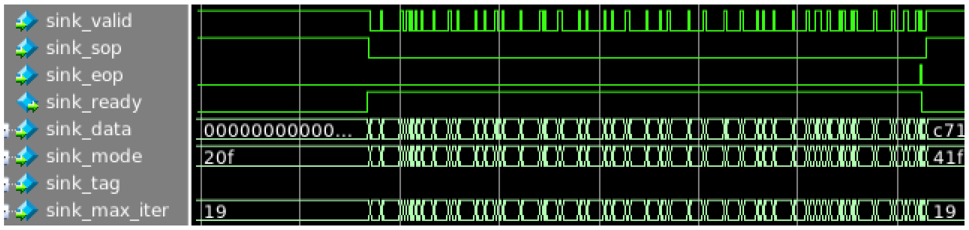
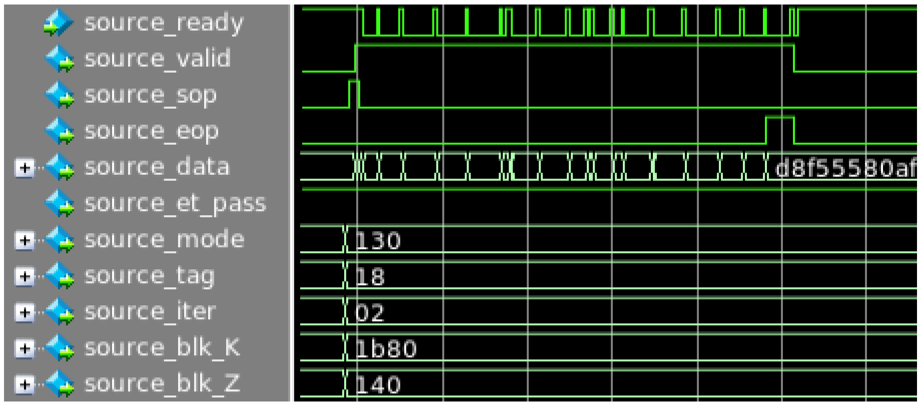


Figure 10. Decoded Message Output from the Decoder

- `source_valid` and `source_sop` are asserted about the same time. `source_sop` is asserted only for the first message block out
- The decoder may assert `source_sop` earlier and keep it high for a number of clock cycles – only the data block output during the last clock cycle just before `source_sop` is deasserted (and with `source_valid` asserted) is the actual beginning of the message
- `source_valid` remains asserted for the duration of the output message
- `source_sop` may get stretched for more than one clock cycle by the backpressure from the testbench, i.e. because the testbench deasserts `source_ready` downstream
- `sink_eop` is asserted for the last valid message block output. It also can be stretched by the backpressure from the sink downstream.



Related Information

C++ and MATLAB Software Models on page 35

4. 5G LDPC Intel FPGA IP Functional Description

The 5G LDPC Intel FPGA IP comprises an encoder and a decoder.

5G LDPC Decoder on page 19

5G LDPC Encoder on page 27

Avalon Streaming Interfaces in DSP Intel FPGA IP on page 31

4.1. 5G LDPC Decoder

The 5G LDPC Decoder supports all modes specified in the 3GPP New Radio specification. The decoder can be either a single decoder or dual decoders.

Input packets arbitration for dual decoders:

- If the incoming packet has small Z, the IP sends it to the second decoder, if it is available. If the second decoder is full and not available to process the small packet, the IP sends the small packet to the first decoder, if it is available. If none of the two decoders are available for the small packet, the IP back pressures to the upstream
- If the incoming packet has large Z, the IP sends it to the first decoder, if it is available. If the first decoder is full and not available to process this packet, the IP back pressures to the upstream. Even if the upstream has the subsequent packet that has small Z and the second decoder is available to process it, the packet doesn't arrive at the IP until the first decoder becomes available to process the previous big packet

The order of output decoded packets is in the order that the IP receives the packets, although the IP processes the packets out of order.

4.1.1. 5G LDPC Decoder Signals

Figure 11. Decoder Signals

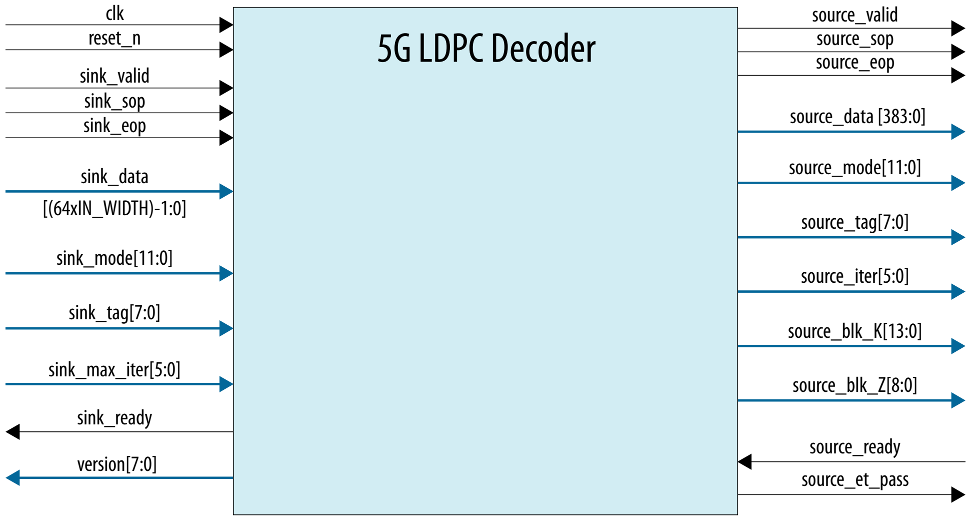


Table 12. Decoder Interface Signals

Signal names beginning with `sink_` are in the input interface to the decoder IP; signal names beginning with `source_` are in the output interface from the decoder IP (except for `sink_ready` and `source _ready`). Both interfaces comply with the Avalon-ST specification with `readyLatency=0`.

Name	Direction	Description
`clk`	Input	Clocks the 5G LDPC decoder IP signals and internal transitions.
`reset_n`	Input	Active low, synchronous reset signal for 5G LDPC decoder. Asserting this signal for one clock cycle is sufficient to ensure the reset process initiates.
`version[7:0]`	Output	Version number.For IP version 21.1.0, version = 0x0C).

Name	Direction	Description
`sink_valid`	Input	Qualifies the `sink_data` signal. When `sink_valid` is not asserted, the IP stops processing input until you reassert the `sink_valid` signal.
`sink_sop`	Input	Marks the start of an incoming packet.
`sink_eop`	Input	Marks the end of an incoming packet.
`sink_ready`	Output	Indicates that the decoder is ready to receive data on the current clock cycle. The IP can backpressure incoming data by deasserting this signal. The `readyLatency` for this signal is 0: the IP can read valid input data in the same clock cycle in which it raises this signal. Refer to the *Avalon Interface Specifications* for the description of this Avalon streaming interface property.

continued...

Name	Direction	Description
sink_data[(64*IN_WIDTH)-1:0]	Input	Data input. The IP processes this input only while it asserts sink_ready and you assert the sink_valid signal. The default value of **IN_WIDTH** is 6.
sink_mode[11:0]	Input	Input block mode. The value specifies the lifting size, base graph, and code rate. Refer to Input and Output Block Mode Meaning in 5G LDPC Decoder Data Formats on page 23. This signal must be valid for the current information block when the upstream source asserts sink_sop and sink_valid.
sink_tag[7:0]	Input	Block tag. An optional tag that accompanies the block from input to output. You can use this tag to identify the correspondence of the input and output block. This signal must be valid for the current information block when the upstream source asserts sink_sop and sink_valid.
sink_max_iter[5:0]	Input	Specifies the maximum number of decoding iterations. The maximum value is 63. This signal must be valid for the current information block when the upstream source asserts sink_sop and sink_valid.

Name	Direction	Description
source_valid	Output	The IP asserts this signal when source_data holds valid data.
source_sop	Output	The IP asserts this signal to mark the start of a packet.
source_eop	Output	The IP asserts this signal to mark the end of a packet.
source_ready	Input	Indicates that the design downstream of the IP is ready to receive data. The design can backpressure the IP by deasserting this signal. The readyLatency for this signal is 0: the downstream design receives data the IP core drives on source_data in the same clock cycle in which the design asserts this signal. Refer to the *Avalon Interface Specifications* for the description of this Avalon- streaming interface property.
source_data[383:0]	Output	Data output. When the IP sends valid data on this bus, it asserts the source_valid signal. If source_ready is deasserted, the decoder holds the data constant until source_ready is asserted. If **NUM_DECODERS** = 1, and **MAX_LF_DECODER0** is not 384, • source_data[383:192] is 0 if **MAX_LF_DECODER0** is 192, • source_data[383:128] is 0 if **MAX_LF_DECODER0** is 128 • source_data[383:96] is 0 if **MAX_LF_DECODER0** is 96 Otherwise, the IP uses the full bus width.
source_mode[11:0]	Output	The value specifies the lifting size, base graph, and code rate. Refer to *Input and Output Block Mode Meaning* in 5G LDPC Decoder Data Formats on page 23. This signal is valid when source_sop and source_valid are both asserted.

continued...

Name	Direction	Description
source_tag[7:0]	Output	Block tag. An optional tag that goes with the block from input to output. You can use this tag to establish the correspondence of the input and output block. This signal is valid when source_sop and source_valid are both asserted.
source_blk_K[13:0]	Output	Transmission block size K, in bits. K is the size of the output block from the decoder IP. This signal is valid when source_sop and source_valid are both asserted.
source_blk_Z[8:0]	Output	The lifting size of the output block. For the output block, only the source_blk_Z least significant bits of source_data[383:0] are valid. This signal is valid when source_sop and source_valid are both asserted.
source_iter[5:0]	Output	Actual number of decode iterations by the IP to process the current output block. The count starts at 0 and might not be an accurate count of full iterations, because the decoding process might stop at any point if the decoder meets early termination criteria. The count cannot exceed the value communicated for the current transmission block via sink_max_iter minus one. This signal is valid when source_sop and source_valid are both asserted.
source_et_pass	Output	Indicates whether the current information block meets early termination criteria. 0 indicates that the block does not meet early termination criteria; 1 indicates that the block meets early termination criteria. The decoder can also assert the signal on the last iteration. This signal is valid when source_sop and source_valid are both asserted.

Figure 12. Example 5G LDPC Decoder Input Timing Diagram

This timing diagram illustrates an example transaction on the decoder input interface. Refer to the *Avalon Interface Specifications* for information about the required behavior of the ready, valid, sop, eop, and data signals on this Avalon-ST interface.

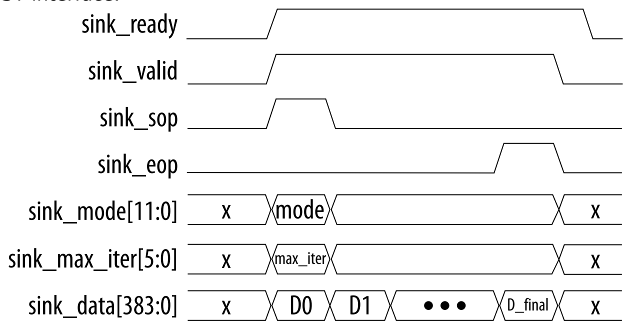
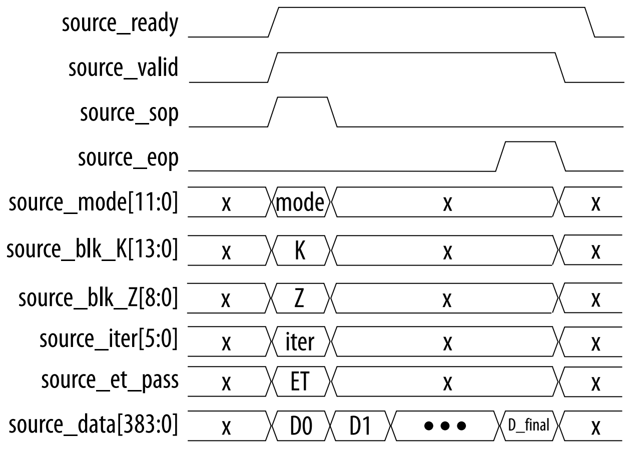


Figure 13. Example 5G LDPC Decoder Output Timing Diagram

This timing diagram illustrates an example transaction on the decoder output interface. Refer to the *Avalon Interface Specifications* for information about the required behavior of the ready, valid, sop, eop, and data signals on this Avalon-ST interface.



Related Information

Avalon Interface Specifications
Detailed information about Avalon-ST interfaces and their signal behavior requirements, including the definition of `readyLatency`.

4.1.2. 5G LDPC Decoder Data Formats

Input Data Format

An input code block of length *N* consists of *N* log-likelihood ratio codes (LLRs): $L_0, L_1, L_2, \ldots, L_{N-1}$. Each LLR is IN_WIDTH bits wide.

IN_WIDTH is 5 or 6. The decoder input on a single active edge of the clock is 64 LLRs. The resulting width of the input data bus to the decoder is 64*`IN_WIDTH`.

The decoder IP accepts punctured LLRs. You can assume that the IP punctures the first 2*Z bits of the original information block during encoding, where *Z* is the lifting size. The IP functionally prepends 2*Z implicit LLRs with value `IN_WIDTH'b0` to the input stream of LLRs.

The number of clock cycles that the IP requires to receive the LLR values is $(n_b - 2)$ * ceil(Z / 64), where n_b is the number of columns in the parity check matrix. The value of n_b depends on the base graph, the information block size K, and the code rate.

Table 13. Example: Decoder Input Data Format for Base Graph 1, Z=384, and Code Rate 8/9

In this example, n_b=27. Therefore, the IP requires (27-2)*ceil(384/64) = 150 clock cycles to receive the input data.

sink_data[64*IN_WIDTH-1:0] = sink_data[383:0]	clock cycle				
	0	1	...	148	149
`sink_data[5:0]`	L_0	L_{64}	...	L_{9472}	L_{9536}
`sink_data[11:6]`	L_1	L_{65}	...	L_{9473}	L_{9537}
...	...	...	...	...	...
`sink_data[383:378]`	L_{63}	L_{127}	...	L_{9535}	L_{9599}

Table 14. Example 2: Decoder Input Data Format for Base Graph 2, Z=208, and Code Rate=1/2

In this example, n_b=22. Therefore, the IP requires (22-2)*ceil(208/64) = 80 clock cycles to receive the input data, where cells with X indicate the bits to ignore. Each 208 LLR symbols take four clock cycles to receive by the IP.

sink_data[64*IN_WIDTH-1:0]	clock cycle										
	0	1	2	3	4	5	6	7	...	78	79
sink_data[1*IN_WIDTH-1:0*IN_WIDTH]	L_0	L_{64}	L_{128}	L_{192}	L_{208}	L_{272}	L_{336}	L_{400}	...	L_{4080}	L_{4144}
...	...	...	...	...	...	...	...	...	...	...	...
sink_data[16*IN_WIDTH-1:15*IN_WIDTH]	L_{15}	L_{79}	L_{143}	L_{207}	L_{223}	L_{287}	L_{351}	L_{415}	...	L_{4095}	L_{4159}
sink_data[17*IN_WIDTH-1:16*IN_WIDTH]	L_{16}	L_{80}	L_{144}	X	L_{224}	L_{288}	L_{352}	X	...	L_{4096}	X
...	...	...	...	X	...	...	...	X	...	...	X
sink_data[64*IN_WIDTH-1:63*IN_WIDTH]	L_{63}	L_{127}	L_{191}	X	L_{271}	L_{335}	L_{399}	X	...	L_{4143}	X

Decoder Input LLR Symbol Format

The log-likelihood value is the logarithm of the probability that the received bit is a 0, divided by the probability that this bit is a 1. 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 uses asymmetrical numeric range for LLRs including the most negative two's complement value for the chosen number of bits.

Table 15. LLR Meaning

LLR (IN_WIDTH=6)	Meaning	Resulting Hard Decision
011111	Most likelihood of a 0	0
...	...	...
000001	Lowest likelihood of a 0	0
000000	Equal probability of a 0 or 1	1 (the convention implemented by the decoder)
111111	Lowest likelihood of a 1	1
...	...	...
100000	Most likelihood of a 1	1

Decoder Input Control and Output Status Signal Formats

The input mode signal, `sink_mode[11:0]`, and the output mode signal, `source_mode[11:0]`, have the same fields. The required input values on `sink_mode` are the expected output values on `source_mode`.

- `sink_mode[11]` encodes the base graph selection. The value of 0 indicates base graph 1 (BG1), and the value of 1 indicates base graph 2 (BG2).
- `sink_mode[10:8]` is code rate selection. Refer to Table 17 on page 26.
- `sink_mode[7]` is enable reduced-syndrome . 0: disable; 1: enable.
- `sink_mode[6]` is disable early termination . 0: enable; 1: disable.
- `sink_mode[5:0]` is Z selection. Refer to Table 16 on page 25.

Table 16. Z Selection sink_mode[5:0]

sink_mode[5:0]	Z	sink_mode[5:0]	Z
0	2	26	48
1	3	27	52
2	4	28	56
3	5	29	60
4	6	30	64
5	7	31	72
6	8	32	80
7	9	33	88
8	10	34	96
9	11	35	104
10	12	36	112
11	13	37	120
12	14	38	128
13	15	39	144
14	16	40	160
15	18	41	176
16	20	42	192
17	22	43	208
18	24	44	224
19	26	45	240
20	28	46	256
21	30	47	288
22	32	48	320
23	36	49	352
24	40	50	384
25	44	X	X

Table 17. Code Rate Selection (sink_mode[10:8])

The numbers are in terms of circulant matrix blocks.

sink_mode[10:8]	Code Rate	Base Graph 1		Base Graph 2	
		Number of Rows in Parity Check Matrix (m_b)	Number of Columns in Parity Check Matrix (n_b)	Number of Rows in Parity Check Matrix (m_b)	Number of Columns in Parity Check matrix (n_b)
000	1/5	X	X	42	52
001	1/3	46	68	22	32
010	2/5	35	57	17	27
011	1/2	24	46	12	22
100	2/3	13	35	7	17
101	22/30 (~3/4)	10	32	X	X
110	22/27 (~5/6)	7	29	X	X
111	22/25 (~8/9)	5	27	X	X

Decoder Output Data Format

The width of the `source_data[383:0]` signal is 384 bits, where only the Z LSBs are valid. You can ignore the rest of the MSBs.The number of output block bits is 22*Z for BG1 and 10*Z for BG2. Hence, the IP requires 22 clock cycles for BG1 and 10 clock cycles for BG2 to output the data.

Table 18. Decoder Output Data Format for Base Graph 1, Z=384

source_data[383:0]	Clock Cycle				
	0	1	...	20	21
`source_data[0]`	b_0	b_{384}	...	b_{7680}	b_{8064}
`source_data[1]`	b_1	b_{385}	...	b_{7681}	b_{8065}
...	...	...	...	...	...
`source_data[383]`	b_{383}	b_{767}	...	b_{8063}	b_{8447}

Table 19. Decoder Output Data Format for Base Graph 2, Z=208

The IP requires 10 clock cycles to output the data.

source_data[383:0]	Clock Cycle				
	0	1	...	8	9
source_data[0]	b_0	b_{208}	...	b_{1664}	b_{1872}
...	...	...	...	...	...
source_data[207]	b_{207}	b_{415}	...	b_{1871}	b_{2079}
source_data[208]	Ignored				
...					
source_data[383]					

4.1.3. 5G LDPC IP Decoder Parameters

Table 20. Decoder Parameters

Parameter	Value	Description
IN_WIDTH	5 or 6 (default value is 6)	The number of bits per input LLR. The width of `sink_data` is (64 * IN_WIDTH).
NUM_DECODERS	1 or 2	The number of decoders in the IP
MAX_LF_DECODER0	96, 128, 192 or 384	The maximum lifting factor for the packet that the first decoder can handle. When **NUM_DECODERS is 1**, all four options are valid When **NUM_DECODERS is 2**, only 384 is valid.
MAX_LF_DECODER1	96, 128 or 192	The maximum lifting factor for the packet that the second decoder can handle. Only for **NUM_DECODERS=2**.

4.2. 5G LDPC Encoder

4.2.1. 5G LDPC Encoder Signals

Figure 14. Encoder Signals

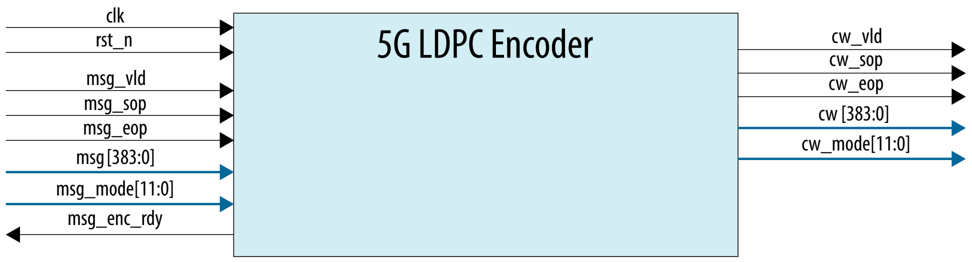


Table 21. Encoder Interface Signals

Signals names beginning with `msg_` are in the input interface to the encoder IP; signal names beginning with `cw_` are in the output interface from the encoder IP (except for `msg_enc_ready`). Both interfaces comply with the Avalon-ST specification. The encoder can backpressure the design by deasserting a ready signal on the input interface, but the design cannot backpressure the encoder.

Name	Direction	Description
`clk`	Input	Clocks the 5G LDPC encoder IP signals and internal transitions.
`rst_n`	Input	Active low, asynchronous reset signal for 5G LDPC encoder IP. Asserting this signal for one full clock cycle is sufficient to ensure the reset process initiates.

Name	Direction	Description
`msg_vld`	Input	Qualifies the `msg` data signal. When `msg_vld` is not asserted, the encoder stops processing input until you reassert the `msg_vld` signal.

continued...

Name	Direction	Description
		If the encoder deasserts the msg_enc_rdy signal, the upstream source must maintain the current value on msg_vld.
msg_sop	Input	Marks the start of an incoming packet.
msg_eop	Input	Marks the end of an incoming packet.
msg_enc_rdy	Output	Indicates that the encoder is ready to receive data on the current clock cycle. The encoder can backpressure incoming data by deasserting this signal. The readyLatency for this signal is 0: the IP can read valid input data in the same clock cycle in which it raises this signal. Refer to the *Avalon Interface Specifications* for the description of this Avalon-ST interface property.
msg[383:0]	Input	Data input. The encoder processes this input only when the upstream source asserts the msg_vld signal and the IP asserts the msg_enc_rdy signal. If the encoder deasserts the msg_enc_rdy signal, the upstream source must maintain the current value on msg.
msg_mode[11:0]	Input	This signal must be valid for the current information block when the upstream source asserts msg_sop. • [5:0] are Z • [8:6] are Code Rate • [9] is Base Graph [0=BG1, 1=BG2] • [11:10] are Kb

Name	Direction	Description
cw_vld	Output	The encoder asserts this signal when cw holds valid data.
cw_sop	Output	The encoder asserts this signal to mark the start of a packet.
cw_eop	Output	The encoder asserts this signal to mark the end of a packet.
cw[383:0]	Output	Data output. When the encoder sends valid data on this bus, it asserts the cw_vld signal.
cw_mode[11:0]	Output	The lifting size of the transmission block. The encoder drives this bus with the value it receives on the input bus msg_mode for the same information block. This signal is valid when cw_sop is asserted.

Figure 15. Example 5G LDPC Encoder Input Timing Diagram

This timing diagram shows an example transaction on the encoder input interface. msg_mode[11:0] = {kb[1:0], bg[0], cr[2:0], z_ind[5:0]}. Refer to the *Avalon Interface Specifications* for information about the required behavior of the ready, valid, sop, eop, and data signals on this Avalon-ST interface.

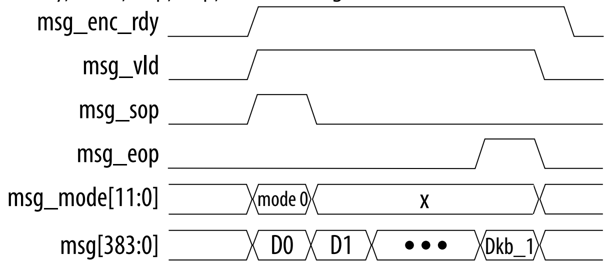
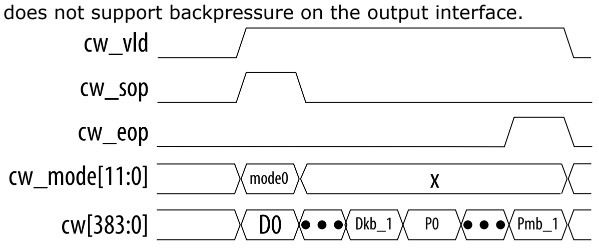


Figure 16. 5G LDPC Encoder Output Timing Diagram

The 5G LDPC encoder does not support backpressure on the output interface.



Related Information

Avalon Interface Specifications
Detailed information about Avalon-ST interfaces and their signal behavior requirements, including the definition of readyLatency.

4.2.2. 5G LDPC Encoder Data Formats

Encoder Input Data Format

The 5G LDPC Encoder IP incoming data on a 384-bit bus (`msg`). The IP clocks in data with K_b clocks, where K_b = 22 for *BG1* and K_b = 6, 8, 9, or 10 for *BG2*. The IP accepts mode information on the clock with `msg_sop` and `msg_vld` both asserted.

The width of the input message bus, `msg[383:0]`, is 384 bits. In each clock cycle, only *Z* LSBs. are valid, you must set the rest of the MSBs to zeros.

Table 22. Encoder Input Data Format for Base Graph 1, Z=64, K_b = 22; message = [M_0, M_1, ..., M_{1407}]

msg[383:0]	clock cycle				
	0	1	...	20	21
msg[0]	M_0	M_{64}	...	M_{1280}	M_{1344}
msg[1]	M_1	M_{65}	...	M_{1281}	M_{1345}
...	...	...	...	...	...
msg[63]	M_{63}	M_{127}	...	M_{1343}	M_{1407}
msg[64]	0	0	...	0	0
...	...	...	...	...	...
msg[383]	0	0	0	0	0

Encoder Input Control Formats (msg_mode[11:0])

Table 23. Z Selection (msg_mode[5:0])

msg_mode[5:0]	Z	msg_mode[5:0]	Z
0	2	26	48
1	3	27	52
2	4	28	56

continued...

msg_mode[5:0]	Z	msg_mode[5:0]	Z
3	5	29	60
4	6	30	64
5	7	31	72
6	8	32	80
7	9	33	88
8	10	34	96
9	11	35	104
10	12	36	112
11	13	37	120
12	14	38	128
13	15	39	144
14	16	40	160
15	18	41	176
16	20	42	192
17	22	43	208
18	24	44	224
19	26	45	240
20	28	46	256
21	30	47	288
22	32	48	320
23	36	49	352
24	40	50	384
25	44	X	X

Table 24. Code Rate, Base Graph, K_b Selection (msg_mode[11:6])

For base graph 2, K_b can be 6, 8, 9, or 10. The encoder automatically inserts the (10 - K_b) * Z filler bits for the encoder input data (the message) before encoding happens. Then the IP removes the filler bits from the encoder output data (the codeword).

Code Rate	Base Graph 1 K_b = 22 msg_mode [11:9]=000	Base Graph 2 K_b = 6 msg_mode [11:9]=001	Base Graph 2 K_b = 8 msg_mode [11:9]=011	Base Graph 2 K_b = 9 msg_mode [11:9]=101	Base Graph 2 K_b = 10 msg_mode [11:9]=111
1/5 msg_mode[8:6]=000	X	001000	011000	101000	111000
1/3 msg_mode[8:6]=001	000001	001001	011001	101001	111001
2/5 msg_mode[8:6]=010	000010	001010	011010	101010	111010
1/2 msg_mode[8:6]=011	000011	001011	011011	101011	111011

continued...

Code Rate	Base Graph 1 K_b = 22 msg_mode [11:9]=000	Base Graph 2 K_b = 6 msg_mode [11:9]=001	Base Graph 2 K_b = 8 msg_mode [11:9]=011	Base Graph 2 K_b = 9 msg_mode [11:9]=101	Base Graph 2 K_b = 10 msg_mode [11:9]=111
2/3 msg_mode[8:6]=100	000100	001100	011100	101100	111100
22/30 (~3/4) msg_mode[8:6]=101	000101	X	X	X	X
22/27 (~5/6) msg_mode[8:6]=110	000110	X	X	X	X
22/25 (~8/9) msg_mode[8:6]=111	000111	X	X	X	X

Encoder Output Data Format

The width of the output codeword bus, cw[383:0], is 384 bits. It takes (nb - 2) clock cycles to output the codeword. Each clock cycle, only *Z* LSBs are valid, the rest of the MSBs are 0s.

Table 25. Encoder Output Data Format for Base Graph 1, Z=64, Kb = 22, CR=1/3, codeword = [CW_0, CW_1, ..., CW_{4223}]

cw[383:0]	clock cycle				
	0	1	...	64	65
cw[0]	CW_0	CW_{64}	...	CW_{4096}	CW_{4160}
cw[1]	CW_1	CW_{65}	...	CW_{4097}	CW_{4161}
...	...	...	...	...	...
cw[63]	CW_{63}	CW_{127}	...	CW_{4159}	CW_{4223}
cw[64]	0	0	...	0	0
...	...	...	...	...	...
cw[383]	0	0	0	0	0

4.3. Avalon Streaming Interfaces in DSP Intel FPGA IP

Avalon streaming interfaces define a standard, flexible, and modular protocol for data transfers from a source interface to a sink interface.

The input interface is an Avalon streaming sink and the output interface is an Avalon streaming source. The Avalon streaming interface supports packet transfers with packets interleaved across multiple channels.

Avalon streaming 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. Avalon streaming interfaces can also support more complex protocols for burst and packet transfers with packets interleaved across multiple channels. The Avalon streaming interface inherently synchronizes multichannel designs, which allows you to achieve efficient, time-multiplexed implementations without having to implement complex control logic.

Avalon streaming interfaces support 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 it has congestion on its output.

Related Information

Avalon Interface Specifications

5. Parameter Optimization for the 5G LDPC IP

You should optimize 5G NR performance for BLER (block error rate) performance, throughput, and sizing.

For throughput vs BLER performance, you can limit the maximum number of iterations that the LDPC decoder uses before giving up on decoding the block. BLER improves with increasing the maximum number of iterations. However, you can potentially reduce the throughput if you raise this number too high. At some point, it is better to give up and request retransmission of a packet or to have the network reassign your LDPC parameters for the shared data channel. You can use the C++ or MATLAB software models, to perform this assessment and to determine which set of parameters suits the network best.

The following BLER curves shows vastly different error rate curves, by changing the parameters of the LDPC error correction scheme.

Figure 17. BLER vs. SNR Graphs Obtained with Different 5G LDPC IP Parameters (log-log scale)

The BLER vs SNR curves show the difference in performance introduced by the maximum number of iterations allowed for a particular LDPC mode of the 3GPP 5G NR specification. The smaller that parameter, the worse the performance is for a particular SNR.

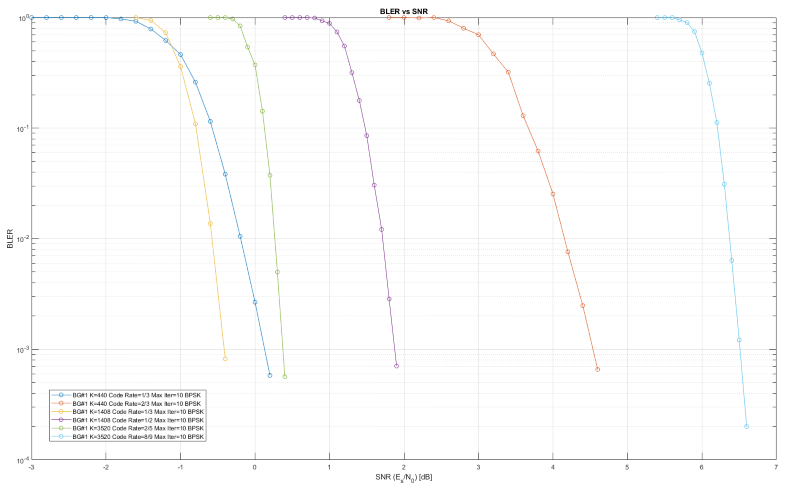


Figure 18. BLER vs. SNR Graphs for a Specific Set of 5G LDPC IP Parameters with Different Iteration Maxima Set for the Decoder (semi-log scale shows the transition dynamic range)

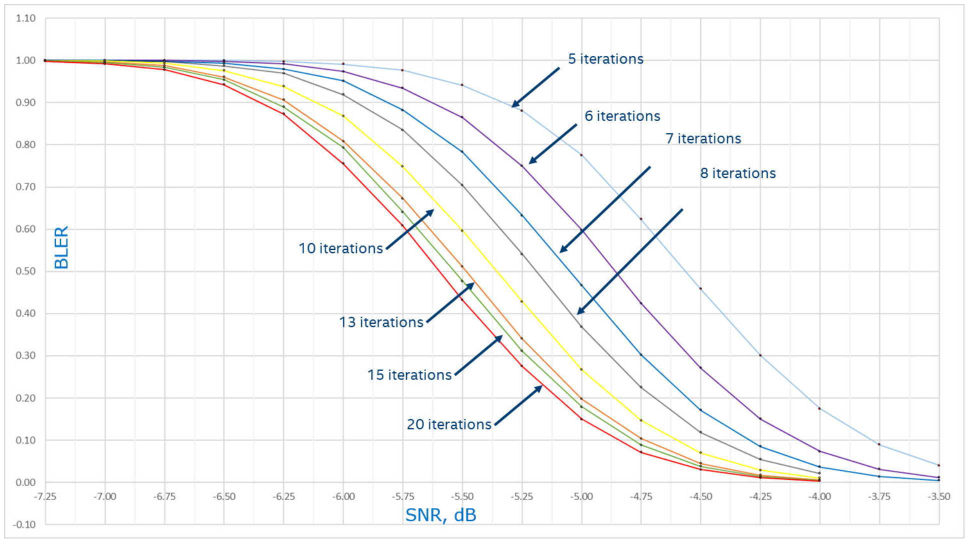
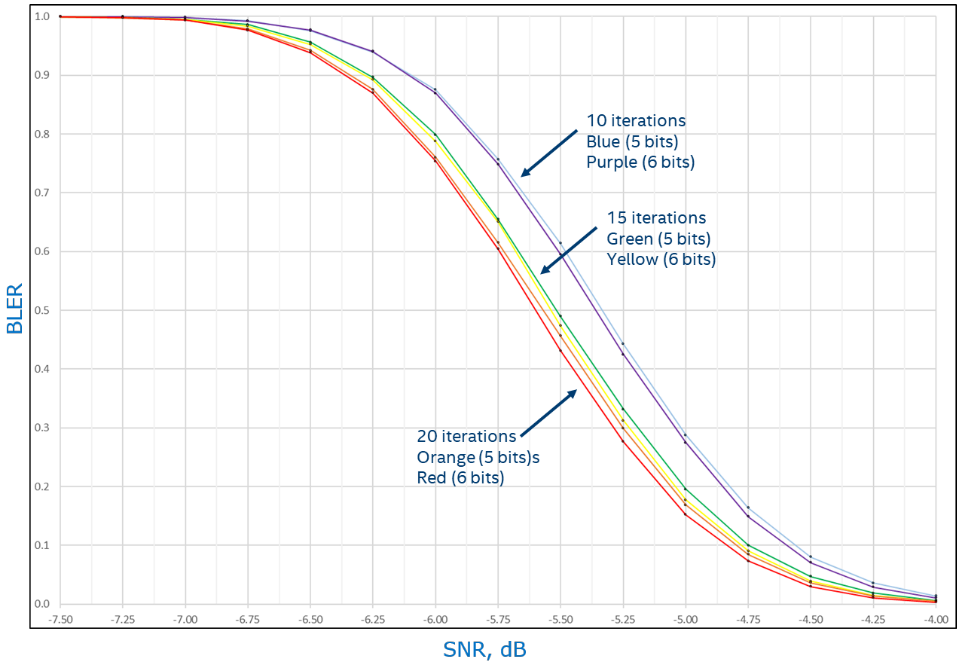


Figure 19. BLER vs SNR Graphs for Several Iteration Maxima with 5G LDPC IP Decoder with 5-Bit and 6-Bit LLRs Respectively (semi-log scale)

The figure shows the difference in BLER by changing the bit precision of input LLR values, i.e. by using 5-bit LLRs vs 6-bit LLRs. The designs with 6-bit LLRs have better BLER performance, but the ones with 5-bit LLRs require fewer FPGA resources and can also operate at a higher maximum frequency



The vertical axis in both figures shows the block rate on the linear scale from 0 (no errors; that BLER is not theoretically attainable) to 1 (the error rate is 100%; the decoder fails to decode every dataword transmitted). The horizontal axis is in logarithmic units (dB) of the ratio between the power of the normalized modulated carrier and the effective power of the additive Gaussian noise in the channel, i.e. its variance.

A transmitted codeword can morph into another valid one, if it is sampled from a noisy channel and translated to LLR values. The LDPC decoder might assert `source_et_pass` and even close before it reaches the maximum number of iterations indicating a successful operation in those cases. However, the decoded dataword does not match the original transmission block. These occurrences are expected. The network should cure such situations by changing the LDPC parameters it is currently using to a set of more robust ones and using the hybrid automatic repeat request (HARQ) mechanism.

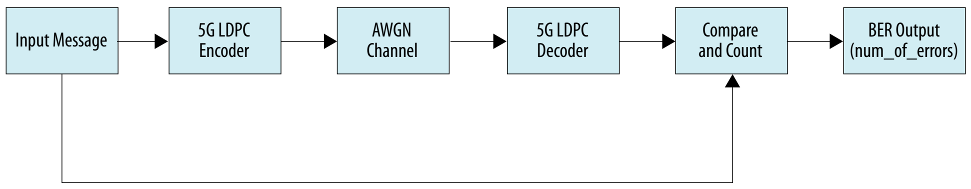
5.1. C++ and MATLAB Software Models

The C++ and MATLAB models represent the functionality of the 5G LDPC IP and can generate BLER vs. SNR graphs for different parameterizations.

The models allow you to:

- Optimize the network performance.
- Shorten the time to determine the channel coding parameters for 5G LDPC such as the base graph, the lifting size, and the code rate.
- Tune the network for better throughput by allowing estimation of the maximum number of iterations needed to decode the majority of codewords under certain channel conditions.

Figure 20. Block Diagram of a Wireless Communication System Emulated with the 5G LDPC IP Software Models



Decoder C++ Model

The file is: `LdpcDecoder_fxp.cpp`.

Function signature:

```
int LdpcDecoder_fxp (
    // Outputs
    vector<int> &BitErrors, // comparison result of input: data vs output:
DecodedBits
    vector<int> &DecodedBits, // decoded bits
    int &DecodedIters, // number of iterations used to decoded
    int &et_pass, // 1: early terminated
    // Inputs
    vector<int> &input, // LLRs
    int ExpFactor, // Z (lifting factor):  2,3,4,5,...,320,352,384
    int max_iter, // number of iterations allowed
    vector<int> &data, // expected data, to be used to compare with DecodedBits
    int BaseGraph, // 0: BG#1, 1: BG#2
    int CodeRate, // BG#1:         1: 1/3, 2: 2/5, 3: 1/2, 4: 2/3, 5: 3/4, 6:
5/6, 7: 8/9
                  // BG#2: 0: 1/5, 1: 1/3, 2: 2/5, 3: 1/2, 4: 2/3
    int in_width = 6, // bit width of input LLRs (2 of that are fractional bits)
    int llr_eq_0_as_negative = 0,// 3: treat LLR == 0 as negative-signed at
internal and output (hard decision)
                                 // 2: treat LLR == 0 as negative-signed at
internal
                                 // 1: treat LLR == 0 as negative-signed at
output (hard decision)
                                   // 0: treat LLR == 0 as positive-signed
      int et_dis = 0, // disable early termination
      int skip_deg1 = 0 // syndrome check: 0: all layers, 1: first 4 layers only
)
```

Note:
- `BitErrors`
  - size = max_iter
  - each element is the number of error comparing input data vs output `DecodedBits` after each iteration, e.g. `BitErrors[1]` is the number of errors after iteration 1
- `DecodedBits`
  - size = `max_iter` * *message length*
  - `DecodedBits[iter` + i * `max_iter]` is the decoded bits after iteration `iter` at bit position `i`, e.g. `DecodedBits[DecodedIters` + i * `max_iter]` is the final decoded bits at bit position `i`

Encoder C++ Model

The file is: `LdpcEncoder.c`

Function signature:

```
void LdpcEncoder (
    // Inputs
    int z, // Lifting factor: 2,3,4,5,...,320,352,384
    int base_graph, // 0: BG#1, 1: BG#2
    int code_rate, // BG#1:         1: 1/3, 2: 2/5, 3: 1/2, 4: 2/3, 5: 3/4, 6:
5/6, 7: 8/9
                   // BG#2: 0: 1/5, 1: 1/3, 2: 2/5, 3: 1/2, 4: 2/3
    int kb, // BG#1: 22, BG#2: 10, 9, 8, 6
    int *msg, // msg array, each element is a symbol (0 or 1) in message
    // Outputs
    int *cw,// codeword array, each element is a symbol (0 or 1) in codeword
    int *cw_size // no. of valid elements in cw_int[]
);
```

5.1.1. Running the 5G LDPC IP C++ Models

The encoder file is: `LdpcEncoder_cTB.c`. The decoder file is `LdpcDecoder_fxp_cppTB.cpp`.

1. For the decoder, run:

```
prompt> g++ LdpcDecoder_fxp_cppTB.cpp
prompt> ./a.out
snr = 1.60
DecodedIters = 7
BitErrors[7] = 0
et_pass = 1
```

2. For the encoder, run:

```
prompt> g++ LdpcEncoder_cTB.c
prompt> ./a.out <testcase>
<testcase> =  0:
  z = 8;
  base_graph = 1; // BG#2
  code_rate = 0; // 1/5
  kb = 6;
  break;
<testcase> =  1:
  z = 36;
  base_graph = 1; // BG#2
  code_rate = 2; // 2/5
  kb = 8;
  break;
<testcase> =  2:
  z = 104;
  base_graph = 0; // BG#1
  code_rate = 7; // 8/9
  kb = 22;
  break;
<testcase> =  3:
  z = 384;
  base_graph = 0; // BG#1
  code_rate = 1; // 1/3
  kb = 22;
```

   For example, type `./a.out 0`, then you see:

```
prompt> tc=0
 prompt> output file: cw.z=8.bg=1.cr=0.kb=6.txt
```

3. To check the result, type:

```
prompt> diff ./cw.z=8.bg=1.cr=0.kb=6.txt ./txt
```

5.1.2. Running the 5G LDPC Decoder and Encoder MATLAB Model in the Design Example

You must compile `.mex` binaries for both the 5G LDPC decoder and the encoder.

1. Change working directory in MATLAB to `<`*`desgin example root`*`>/matlab`
2. Run MATLAB command `mex -setup` to point to the C++ compiler of choice.
3. Run script make: `>> make`

   Note: Tested on gcc/7.2.0 and matlab/2017a.
4. Run an example of encoder output feeds to decoder input: `>> LDPC_example`

6. 5G LDPC IP User Guide Document Archive

If the table does not list an IP version, the user guide for the previous IP version applies.

Table 26. 5G LDPC IP Intel FPGA IPs User Guide Archive

Intel Quartus Prime Version	User Guide
20.1	5G LDPC Intel FPGA IPs User Guide
18.1	5G LDPC Intel FPGA IPs User Guide

7. Document Revision History for the 5G LDPC Intel FPGA IP User Guide

Date	IP Version	Intel Quartus Prime Software Version	Changes
2021.04.05	21.1.0	21.1	• Added dual decoders. • Changed Intel Stratix 10 device support to final.
2020.06.30	20.1.0	20.1	• Corrected IP version number to 20.1.0 • Corrected description of `version[7:0]`. • Added new device to *Performance and Resource Utilization*
2020.06.02	20.1	20.1	• Added support for Intel Agilex devices • Added `version[7:0]` signal to decoder signal table • Deleted *Generated IP Directory Structures* • Updated *Performance and Resource Utilization*
2018.12.21	18.1	-	Initial release.