IPCTEK

5G NR HIGH RATE LDPC V2.0

IP Product Datasheet

PD004

Version 1.0

May 21, 2022

IPCTEK

Table of Contents

Introduction

The IPCTEK's 5G-NR High Rate LDPC FPGA IP cores implement the LDPC encoder and decoder of the 5G-NR code as described in the 3GPP TS 38.212, section 5.3.2. The encoder and the decoder implement the base graph 1, lift size 384 and high-rate matrix part, code rate (22,24). However, thanks to the universal LDPC frame work, the VHDL code can be easily adapted to support both base graph 1 and base graph 2 and any lift size values.

The LDPC encoder achieves an encoding speed of more than 15.0 Gbps (clock frequency 280 MHz) while minimizing the utilization of LUTs and registers resource.

For the LDPC decoder, layered decoding architecture is implemented. Memory access conflict problem is carefully analyzed and avoided using a bit-accurate simulation software, resulting in excellent performance in terms of A Posteriori Probability (APP) convergence speed and frame error rate (FER). Check node processors are fully pipelined, resulting in a very high-speed performance. The decoding maximum clock frequency reaches 280 MHz at Ultrascale and Ultrascale+ devices. At 280 MHz clock frequency, 6 iterations, the decoding speed is equal to 2.69 Gbps. With the IPCTEK's universal LDPC framework, a very high-performance error correction solution can be implemented within a tiny footprint on the FPGA, which makes the IP cores perfectly suitable for either prototyping applications or industrial products.

Key Features

- Compliant to the 5G-NR LDPC of 3GPP TS 38.212 standard. Support the shortened (8448, 9216) code, base graph 1 (high-rate part), lift size 384.
- Highly portable VHDL code, no Xilinx primitive is used.
- Delivered along with a bit-accurate graphical C++ (Qt) simulation software. Thanks to IPCTEK's universal LDPC frame work, other quasi-cyclic LDPC codes can be simulated and the decoding ROM can be generated for the decoder. The VHDL code is easily adapted to support both base graph 1 & 2 and any lift size values.
- At 280 MHz clock, an encoding speed of more than 15.0 Gbps (information bits) is achieved. The encoder is highly optimized, which consumes less than 3k LUTs and 3k registers.
- State-of-the-art layered decoding method was implemented. The decoder has a very fast convergence speed. Almost all the code words are found after 6 iterations at the waterfall region.
- Normalized (scaled) Min-Sum (NMS) decoding algorithm, which is a good performance-complexity compromise, is used. Another advantage is that the NMS algorithm is not sensible to the channel's LLR scaling.
- At 280 MHz clock, 6 iterations, a decoding speed of 2.69 Gbps (information bits) is achieved. In Ultracale and Ultrascale+ devices, the decoder can work at a clock frequency of more than 280 MHz.
- Optional on-the-fly early stopping technique, increasing the average decoding speed.
- Statistic interface for useful decoding information.
- AXI4 Stream in/out data interfaces for seamless integration.

Acronyms

Acronym	Definition
APP	A Posteriori Probability
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
C2V	Check-to-Variable message
FER	Frame Error Rate
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
Gbps	Gigabits per second
LDPC	Low Density Parity Check
LLR	Log Likelihood Ratio
LSB	Least Significant Bit
Mbps	Megabits per second
MSB	Most Significant Bit
NMS	Normalized Min Sum
V2C	Variable-to-Check message

IP Specification

The decoder's global block diagram is shown in Figure 1.

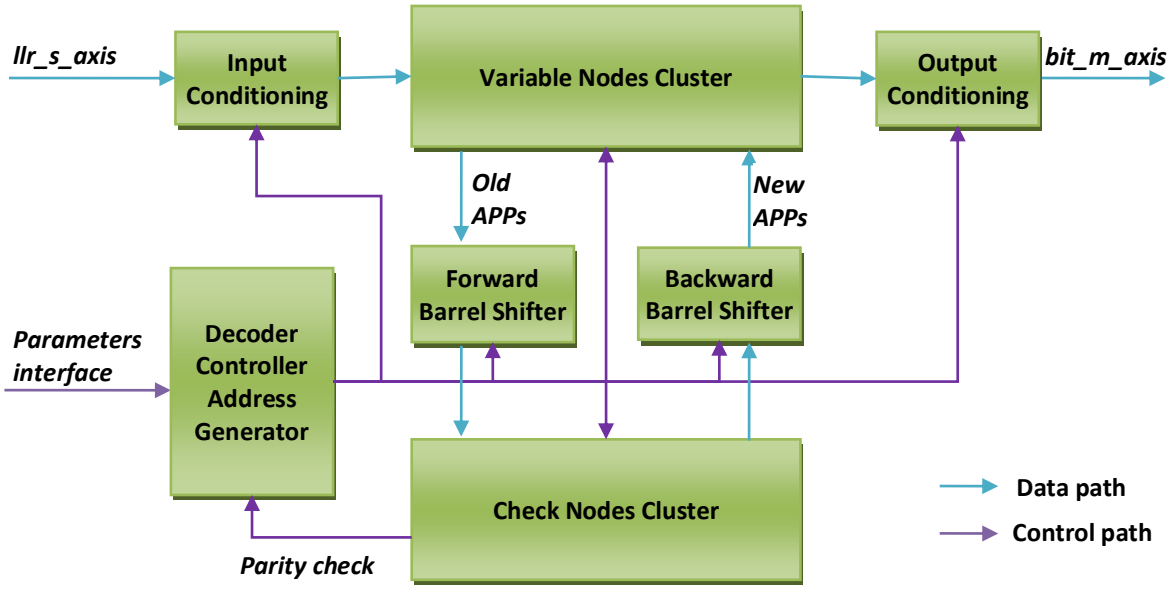

Figure 1 – Decoder architecture block diagram

Parameter	Value
Message passing method	Layered decoding
Decoding algorithm	Normalized Min-Sum
Normalized factor	0.75
Parallelism degree	384
Channel LLR quantization bits	5
Check node quantization bits	6
Variable node (APP) quantization bits	8
Input LLR data bus	384 LLR values per clock
Output decoded bits data bus	384 bits per clock

Table 1 – Decoder key parameters

Input Conditioning

The Input Conditioning module receives channel LLR values from the user's application logic in order to initialize the variable node memory, preparing for the decoding of a new codeword. It also converts a shorten frame of 9216 (384*24) values to a full frame of 9984 (384*26) values for the base code. 768 (384*2) zeros will be inserted at the beginning of a frame.

To optimize the decoding speed, the Input Conditioning module accepts 384 LLR values per clock rising edge. The input ports are compliant to the AXI4 Stream interface, which means the LLR values are consumed when both _tvalid and _tready signal are asserted.

Variable Nodes Cluster

The variable nodes cluster is basically a group of Block RAMs (BRAMs) containing the APP values. Before the decoding process for a new codeword begins, the APP values are initialized with the channel LLR. During the decoding process, these APP values are updated and converged to the correct codeword.

In order to achieve a good decoding speed, the decoder's parallelism degree (the number of check node processors working in parallel) is chosen to be equal to 384. Hence, the APP memory is also organized such that 384 APP values can be accessed every clock rising edge. Simple dual port BRAMs are used where one port is used to write updated APP values and the other port is used to read the old APP values.

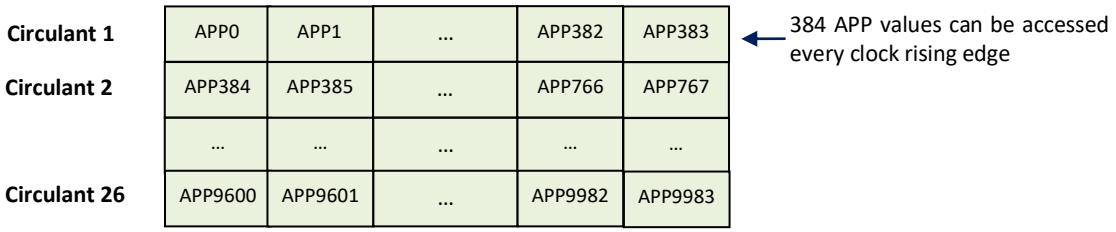

Figure 2 – APP memory organization

Note: As a convention, the LLR values accepted by the decoder is defined as follow.

$$LLR_y = \frac{P(b=0|y)}{P(b=1|y)}$$

Check Nodes Cluster

The check nodes cluster includes a group of 384 check node processors working in parallel and a memory containing check-to-variable messages used during the decoding process. From old APP values and old check-to-variable messages, the check node processors calculate new values according to the Normalized Min-Sum algorithm. The normalized factor is equal to 0.75, which is a good complexity/performance compromise. During that the min value is searched, variable-to-check messages are stored in a FIFO. When the min value is found, the FIFO is read in order to calculate new APP values. The check node processor architecture is illustrated in Figure 3.

While updating the newly calculated APP values, the check node processors also calculate parity check equations using APP values' sign. This information is used on-the-fly by the decoder controller for early stopping if necessary.

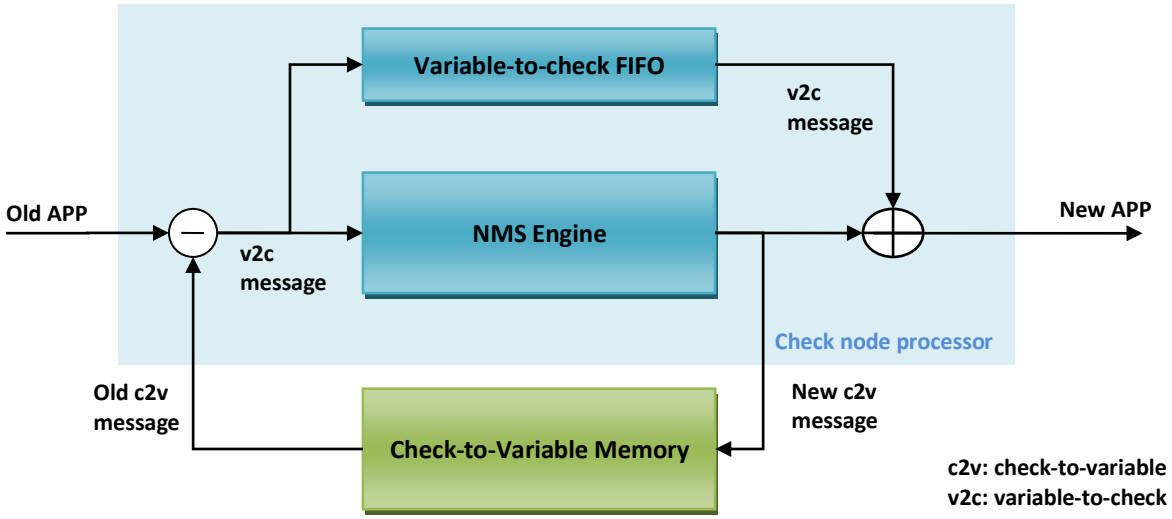

Figure 3 – Check node processor architecture

The NMS engine is fully pipelined, which means the new check-to-variable messages calculation and the new APPs updating are processed in parallel. As a result, it can process one APP value every clock rising edge, maximizing the decoding speed.

Barrel Shifters

A forward Barrel Shifter is responsible for arranging APP values so that the check node processors receive correct data to process. When new APP values are available to be updated, a backward Barrel Shifter is used to shift them back to the initial order before writing into the APP BRAMs. The Barrel Shifters can process one shift operation every clock rising edge.

Address Generator

The Address Generator is basically a ROM which contains all necessary information for the decoding process. Several important pieces of information are: APP BRAM read/write address, BRAM read/write enable and Barrel Shifters' shift values.

It is well-known in the literature that APP memory conflict problem can arise in a layered decoder. For one layer, a check node processor reads certain APP values (the connected variable nodes with the check nodes in this layer) for processing, which takes certain time to finish before updating new APP values. For the next layer, it is possible that the processor reads the same APP values which have not been updated during the previous layer. Hence, an APP value is read 2 times for 2 different layers before being updated. In such a case, the decoding performance is significantly degraded. In the IPCTEK's LDPC framework, we have a bit-accurate C++ simulation model which carefully analyzes such a memory conflict problem. We solve the problem in two steps. Firstly, decoding layer order is designed such that the memory conflict problem can be avoided, or at least reduced to the minimum. Secondly, stall cycles (periods of time at which the decoder does nothing) are appropriately added to

completely solve the problem. Because stall cycles reduce the decoding speed, the two above steps can be repeated many times in order to maximize the decoding speed.

Decoder Controller

The controller is basically a Finite State Machine (FSM) which controls the whole decoding process. Before decoding a new codeword, the FSM registers the early stopping option and the maximum number of iterations at the parameter interface. The parameters are considered to be valid during that the slave axis interface is sinking a new codeword.

Output Conditioning

When the decoding process is finished, the Output Conditioning module reads APP values in the BRAMs, extracts the values' sign then sources the decoded bits to the master axis interface.

It is noted that while the decoded bits are outputted, the LLR values for a new codeword can be inputted at the same time in order to maximize the decoding speed. The Output Conditioning module outputs 384 bits per clock rising edge. The output ports are compliant to the AXI4 Stream interface, which means the decoded bits are consumed when both _tvalid and _tready signal are asserted.

Performance

Convention: the latency is defined as the time delay between the first input valid data and the first output valid data.

Encoder Latency

The encoder latency is equal to 156 clock periods. At 280 MHz clock frequency, this is equivalent to 0.56 µs.

Decoder Latency

The decoder latency is equal to 78 + 133*N (clock periods) where N is the number of iterations.

- One-iteration case (minimum): 211 clock periods. This corresponds to 0.75 µs at 280 MHz processing clock.
- Ten-iteration case (typical): 1408 clock periods. This corresponds to 5.03 µs at 280 MHz processing clock.

Throughput

Encoder throughput

The encoder outputs 9216 encoded bits, which corresponds to 8448 information bits, every 153 clock periods. Hence, for example with 280 MHz processing clock, the encoder throughput is equal to 16.86 Gbps encoded bits, or 15.46 Gbps information bits.

Decoder throughput

The decoder processing time of one code word is equal to 80 + 133*N (clock periods) where N is the number of iterations. The decoder throughput is hence dependent on the processing clock frequency and the number of iterations. With 280 MHz processing clock, the net throughput (information bits) is illustrated in the table below.

N	Net decoder throughput (Gbps)	Comments
1	11.1	
2	6.83	
3	4.93	
4	3.86	
5	3.17	
6	**2.69**	**Typical for good speed-FER compromise**
7	2.33	
8	2.06	
9	1.85	
10	**1.67**	**Typical for best FER performance**

Table 2 – Decoder throughput in function of number of iterations. Clock frequency 280 MHz.

Maximum operation frequency

Both the encoder and the decoder are carefully designed with pipeline registers so that in general they can work at a very high clock frequency. In Ultrascale and Ultrascale+ Xilinx devices, we have successfully implemented the encoder and the decoder which work at more than 280 MHz. Please refer to the reference example design for more details.

BER/FER performance

Below is the BER/FER performance measured on the FPGA target, 10 iterations max, early stopping off. The decoder gives an excellent BER/FER performance without error floor. At waterfall region, most of the codewords are found after 6 iterations. This result is achieved with the reference example design where an encoder, a decoder and an AWGN channel are implemented on the Xilinx evaluation board KCU105. The processing clock is equal to 280 MHz. It is noted that the users can reproduce this result with an evaluation example design available upon request.

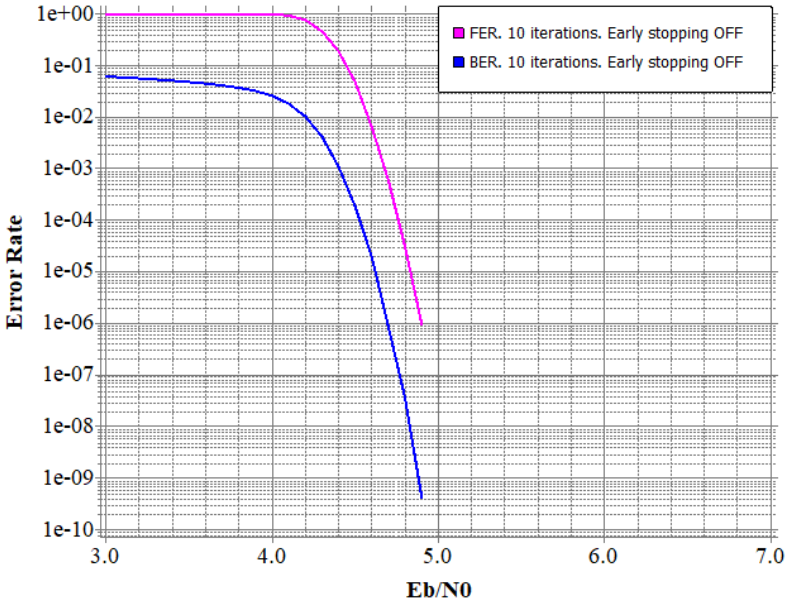

Figure 4 – BER/FER performance. 10 iterations max. Early Stopping off

Optimal number of iterations

Because the number of maximum iterations has a direct impact on the decoder throughput, it is interesting to analyze how the BER and FER depend on this parameter. The figure below illustrates the FER performance in function of the maximum number of iterations. The early stopping option is disabled. We remark that from 2 to 6 iterations, the FER is significantly improved when the number of iterations increases. This can be explained by the fact that at the waterfall region most of the codewords are found after 6 iterations as shown in Table 3. Beyond 10 iterations, no significant improvement is observed.

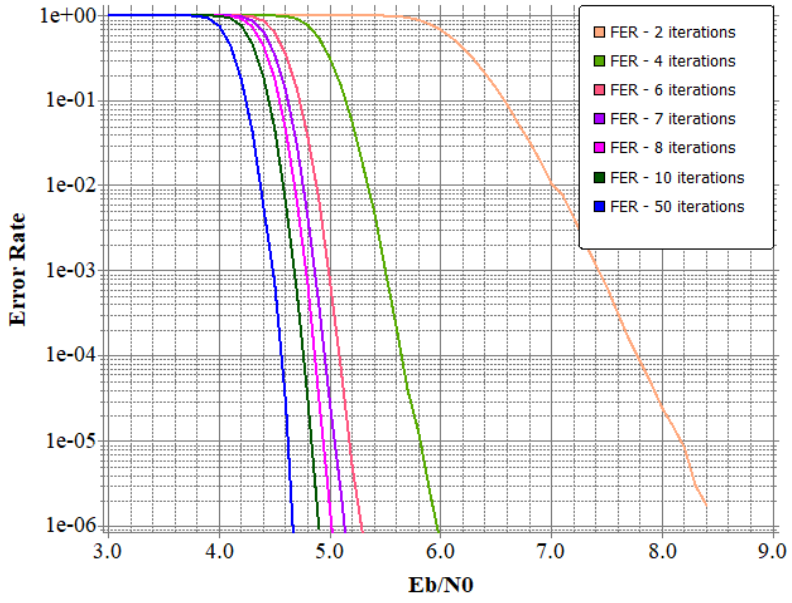

Figure 5 – FER performance in function of number of iterations. Early Stopping off

Eb/N0 (dB)	Average number of iterations
<= 3.7	50
3.8	49.2
3.9	49.7
4.0	42.4
4.1	35.6
4.2	24.2
4.3	13.1
4.4	10.0
4.5	8.9
4.6	7.8
4.7	7.1
4.8	6.6
4.9	6.2
5.0	5.8
6.0	4.2
7.0	3.7
8.0	3.1
>= 9.0	3.0

Table 3 – Average number of iterations in function of Eb/N0. Max number of iterations 50. Early stopping ON

Early stopping impact

Early stopping is a technique which early stops the decoder before reaching the maximum number of iterations when it estimates that a correct codeword is found. This option is interesting in low BER/FER region as it boosts the average decoder throughput. However, if the early stopping technique is designed without care, it may introduce an error floor in the BER/FER curve. In the IPCTEK's LDPC decoder, we implement a simple yet effective early stopping technique which does not degrade the BER/FER performance, while boosting the average decoding speed as illustrated in the figure below.

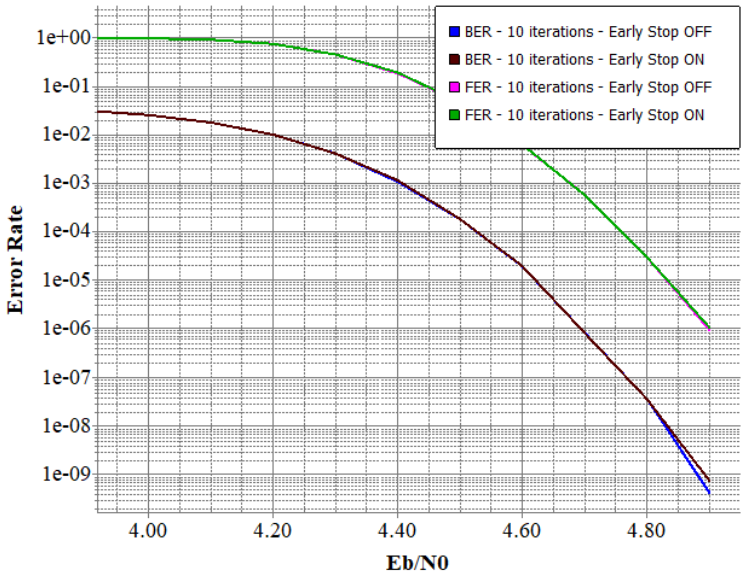

Figure 6 – Early Stopping ON versus Early Stopping OFF. 10 iterations max

Resource Utilization

Below is the resource utilization after synthesis on the Xilinx KCU105 board target.

Encoder

Parameters	Utilization
LUT utilization	2276
Register utilization	2401
Block RAM Tile utilization	6

Table 4 – Encoder resource utilization

Decoder

Parameters	Utilization
LUT utilization	69007
Register utilization	79198
Block RAM Tile utilization	75.5

Table 5 – Decoder resource utilization

Port Descriptions

This section describes the IP ports' functionality.

Encoder

Port name	Bitwidth	Dir	Description
Common parameters			
clk	1	in	Decoder processing clock. All the interfaces are synchronous to this clock.
rst_n	1	in	Synchronous reset. To reset the IP, this signal must be held low for at least 2 periods of the processing clock clk.
info_part_size_m1_in	8	in	Info bits size minus 1 in number of BRAM entries. For the base graph 1, high-rate code, this value is equal to 21.
codeword_size_m1_in	8	in	Codeword size minus 1 in number of BRAM entries. For the base graph 1, high-rate code, this value is equal to 25.
nb_parity_blocks_m1_in	8	in	Number of blocks of parity bits minus 1. For the base graph 1, high-rate code, this value is equal to 3. This is used by the early stop module to correctly count the number of check nodes.
Information bits slave axis interface (8448-bit frame)			
bit_s_axis_tdata	384	in	Information bits (to-be-encoded bits) from user's application. The bits order is from LSB to MSB. For example, the first information bit is located at s_axis_tdata(0), the second information bit is located at s_axis_tdata(1) etc.
bit_s_axis_tvalid	1	in	Information bits valid, indicating that the data in the s_axis_tdata bus is valid.
bit_s_axis_tready	1	out	Information bits ready. The encoder is ready to accept a new entry only when this signal is asserted.
Encoded bits master axis interface (9216-bit frame)			
encoded_m_axis_tdata	384	out	Encoded bits data bus. The bits order is from LSB to MSB. For example, the first bit is located at m_axis_tdata(0), the second bit is located at m_axis_tdata(1) etc.
encoded_m_axis_tvalid	1	out	Encoded bit data valid, indicating that the data presented in the m_axis_tdata bus is valid.
encoded_m_axis_tready	1	in	Encoded bits data ready. The user's application logic asserts this signal when it is ready to accept a new entry.

Decoder

Port name	Bitwidth	Dir	Description
Common parameters			
CHANNEL_LLR_BITWIDTH			Channel input LLR number of bits. Must be smaller than 8. Default value is 5.
CHANNEL_LLR_AXIS_ BITWIDTH			Channel LLR Axis data width, must be larger than or equal to CHANNEL_LLR_BITWIDTH and must be a multiple of 8. Default value is 8.
C2VMSG_BITWIDTH			Check-to-variable message bit width. Must be larger than CHANNEL_LLR_BITWIDTH and smaller than 8. Default value is 6.
clk	1	in	Decoder processing clock. All the interfaces are synchronous to this clock.
rst_n	1	in	Synchronous reset. To reset the IP, this signal must be held low for at least 2 periods of the processing clock clk.
Decoder parameters			
early_stop_en_in	1	in	Early stopping enable option. '1' Enable. '0' Disable.
nb_iters_max_in	8	in	Maximum number of iterations. Note that this signal and the early_stop_en_in signal can be changed from frame to frame.
input_frame_len_m1_in	8	in	Input shorten codeword length minus 1 in number of axis beats. For base graph 1, high-rate code this is equal to 23.
information_part_size_m1_ in	8	in	Information part size minus 1 in number of BRAM entries. For base graph 1, high-rate code this is equal to 21.
LLR slave axis interface (9216-LLR frame)			
s_axis_tdata	CHANNEL_ LLR_AXIS_ BITWIDTH *384	in	LLR data from user's application. The APP memory is initialized with these LLR values before the decoding process begins. The LLR values are packed from LSB to MSB. For example, the first LLR is packed in s_axis_tdata(7 downto 0), the second LLR is packed in s_axis_tdata(15 downto 8) etc. When the LLR bitwidth is smaller than 8, it is packed in the LSB of the s_axis_tdata bus, the MSB bits are don't care bits. For example, if the LLR bitwidth is equal to 5, s_axis_tdata(4 downto 0) contains valid data, and s_axis_tdata(7 downto 5) are don't care bits. s_axis_tdata(12 downto 8) contains valid data, and s_axis_tdata(15 downto 13) are don't care bits etc.
s_axis_tvalid	1	in	LLR data valid, indicating that the data in the s_axis_tdata bus is valid.
s_axis_tready	1	out	LLR data ready. The decoder is ready to accept a new entry only when this signal is asserted.
Decoded bits master axis interface (8448-bit frame)			
m_axis_tdata	384	out	Decoded bits data bus. The bits order is from LSB to MSB. For example, the first bit is located at m_axis_tdata(0), the second bit is located at m_axis_tdata(1) etc.
m_axis_tvalid	1	out	Decoded bit data valid, indicating that the data presented in the m_axis_tdata bus is valid.
m_axis_tready	1	in	Decoded bits data ready. The user's application logic asserts this signal when it is ready to accept a new entry.
Statistic interface			
stat_data_out	16	out	Statistic data including the number of iterations

			stat_data_out(7 downto 0) and the codeword found flag stat_data_out(8).
stat_data_valid_out	1	out	Statistic data valid flag. This signal is asserted during one clock period when the decoder finishes one codeword.
Miscellaneous			
ipc_ip_version	16	out	IP version. Should read 0x200.
ipc_ip_demo	1	out	'1' → Demo design, the IP stops after 30 minutes. '0' → Fully functional IP.

Design Guidelines

This section details important guidelines in order to successfully integrate the IP cores into the user FPGA design.

Clocking

For both the encoder and the decoder, there is only one processing clock. The maximum operational clock frequency depends on the FPGA family and speed grade. In general, the modules can work at very high clock frequency (> 200 MHz). For Xilinx Ultrascale and Ultrascale+ devices, 280 MHz processing clock are successfully demonstrated in the example design.

Reset

Upon startup, after the clock is stable, the modules need to be properly reset for a good functioning. The reset ports are synchronous to the clk port. The reset polarity is active low. And the reset pulse needs to be equal to at least 2 clock periods.

Axis basic handshake

All data interfaces used in the encoder and the decoder are compliant to the AXI4 Stream (AXIS) interface. During an AXIS transaction, we have a master which sources the data and a slave which sinks the data. The function of a signal in an AXIS interface can be decoded according to its suffix.

- _tdata: this signal contains the data for the transaction.
- _tvalid: this signal indicates that the data is valid.
- _tready: this signal indicates that the slave is ready to sink the data.
- _tlast: this signal indicates that this data is the last in a frame.
- _tkeep: when the bytewidth of a data word is larger than 1, the tkeep signal indicates which bytes in the word are valid. Each bit in the tkeep signal corresponds to one byte in the data signal. The LSB bit corresponds to the LSB byte and so on.
- _tuser: this signal contains additional information concerning the frame.

In order to successfully integrate a component with an AXIS interface into your design, it is worth taking care of several basic rules.

- The data is only successfully consumed by the slave when both tvalid and tready signals are asserted.
- The master should never wait for the slave's tready signal to be asserted before asserting the tvalid signal. This may result in a deadlock situation in some cases.
- User should assert the tlast signal at the end of a frame, if it exists.

The Figure below shows the basic handshake of an AXIS interface.

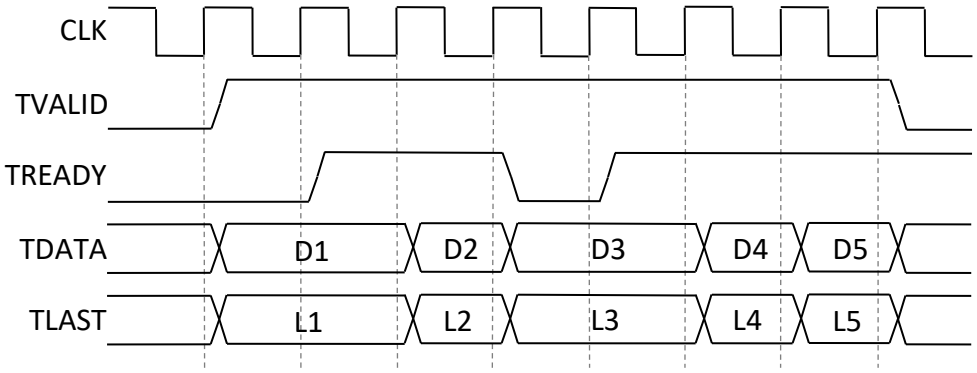

Figure 7- AXI4-Stream basic handshake.

End Of Document.