

LDPC Decoder for DVB-S2

Introduction:

The LDPC Decoder core provides designers an LDPC Decoder block used in DVB-S2 systems.

LDPC codes are the codes that offer error detection and correction capabilities close to maximum theoretical limits. LDPC codes were discovered by Gallager in 1962. But they were not given much attention for decades as the technology was not matured enough for its efficient implementation. With the success of iterative 'turbo codes' MacKay and Neal reintroduced LDPC codes in 1995.

LDPC codes have easily parallelizable decoding algorithms. The parallelizability is 'adjustable' providing an option to choose between throughput and complexity.

LDPC decoder along with BCH decoder forms a Forward Error Correction (FEC) block, which is used in DVB-S2. During encoding, the information data is added with redundant data called parity data. This parity data is useful in detecting the errors that are introduced during the transmission of information data through channel.

DVB-S2 is the second-generation for satellite broad-band applications, developed by Digital Video Broadcasting (DVB) Project in 2003. This is the first standard that uses LDPC mechanism for error detection and correction. It's a single and very flexible standard that covers variety of satellite broadcasting applications. This system also have it's applications in interactive services, professional applications such as digital TV contribution, news gathering, data content distribution and Internet trunking.

Features:

- High speed DVB-S2 LDPC decoder
- Compliant with DVB-S2 standard
- Supports both normal(64800) and short(16400) Frame lengths
- Programmable Iteration Length
- Supports all code rates (1/4, 1/3, 2/5, 1/2, 3/5, 2/3, 3/4, 4/5, 5/6,8/9,9/10).
- Easy to integrate with other modules.
- Fully synchronous design with single clock.
- Available for both FPGA and ASIC versions.

Description:

The LDPC Decoder core is compliant to DVB_S2 standard.

Figure 1 shows the functionality of the LDPC Decoder unit. The encoded data after modulation is transmitted through a channel. At the receiver end, the data from the channel is demodulated to generate Log Likelihood Ratio (LLR) values. This LLR values gives a probability value indicating the amount of certainty that a transmitted bit is either 0 or 1. For example, a very high negative LLR value indicates a very good 1 value. A low negative LLR value indicates that the transmitted bit can be 1. Similarly, a very high positive LLR value indicates a very good 0 value and a low positive LLR value indicates that the transmitted bit can be 0.

The core contains Bit Node units and Check Node Units. The transmitted LLR channel values are fed as input to the LDPC Decoder. Bit node and Check node units communicate with each other to detect and correct the errors present in the transmitted data.

Functionality :

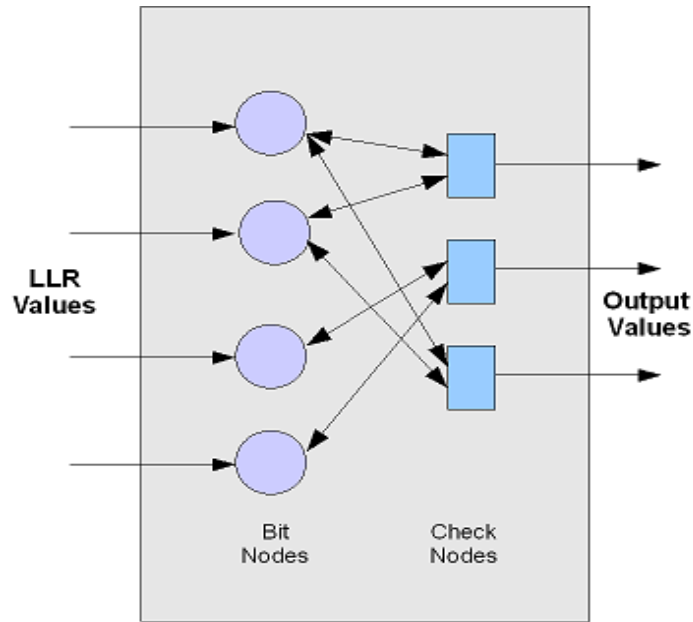


Figure : Functional Diagram

Parameter Table :

Parameter	Type	Description
max_itr_length	Integer	Represents the maximum length of iterations
input_wl	Integer	Represents the word length of each LLR Value

Table 1: LDPC Decoder parameter table.

User Interface:

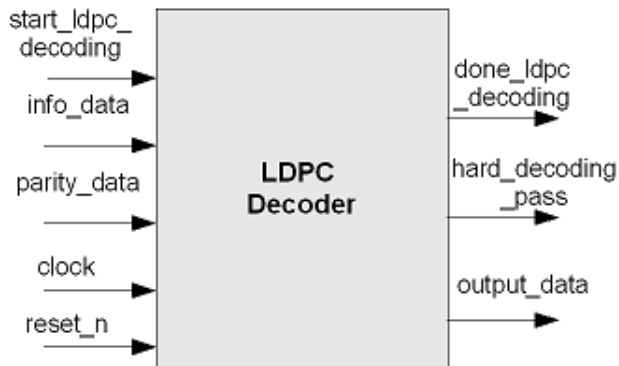


Figure2 : Schematic Block Diagram

Pin Description :

Signal	Direction	Data Width	Description
clock	Input	1	Global synchronous clock
reset_n	Input	1	Active Low Reset
info_data	Input	input_wl * 360	LLR of Info Data
parity_data	Input	input_wl * 360	LLR of Parity Data
start_ldpc_decoding	Input	1	Start signal for decoding
done_ldpc_decoding	Output	1	Signal indicating end of decoding
hard_decoding_pass	Output	1	Pass = 1, fail = 0
output_data	Output	360	Output Data

Table 2: Signal Definition Table

The communication between bit node and check nodes is iterative. This process ends when the decoder converges to a code word or maximum iteration length is reached.

The decoding algorithm has the following four stages.

Initialization:

The channel LLR Values are assigned to the edges that goes out from Bit Node Unit.

Check Node Update:

Check Node Units receives the data from Bit Node edges and process the data according to Min Sum algorithm. The processed data is transmitted to Bit Node Units.

Bit Node Update:

All the edges that goes out from Check node to Bit node are added to compute a sum value. This Sum value is used for Hard Decoding. Bit Node value is updated by subtracting its value from sum value.

Hard Decoding:

Depending on the sign of Sum value, the

transmitted data is determined. Using these values, the parity check equations are computed. If all parity equations are satisfied then the decoder stops, otherwise another Check Node and Bit Node Update is performed.

Figure 2 shows the schematic block diagram of LDPC Decoder. With the assertion of start_ldpc_decoding signal, the decoder core starts receiving data from the pins info_data and parity_data at a rate, 360 LLR values per clock cycle. Once a complete frame of data is received, the decoding process starts.

When the decoding process finishes, the LDPC core asserts done_ldpc_decoding signal indicating the end of decoding. hard_decoding_pass signal indicates the success of ldpc decoding. The decoded data is transmitted out at a rate 360 bits per clock.

Table 1 gives the info about parameterized variables and Table 2 gives the pin description of LDPC decoder.

Performance:

	Resource Utilization on Xilinx Virtex5 device	Throughput at different code rate in Mbits per sec (at 140 MHz frequency)
With 180 Processing Units	42K FF 105K LUT 319 BRAM	100 at code rate 1/2 114 at code rate 1/3 137 at code rate 1/4 96 at code rate 2/3 101 at code rate 2/5 75 at code rate 3/5
With 360 Processing Units	75K FF 194K LUT 151 BRAM	154 at code rate 1/2 174 at code rate 1/3 206 at code rate 1/4 145 at code rate 2/3 155 at code rate 2/5 114 at code rate 3/5

Table 3: LDPC Decoder for DVB-S2 Performance table

Verification:

- Test Vectors are developed using c-model.
- Results are compared with the C model results for different code rates.

Deliverables:

- C-Model, Verilog RTL source code
- Test Benches
- C-Model code for generation of test vectors
- Detailed user documentation