International Journal of Electronics

ISSN: 0020-7217 (Print) 1362-3060 (Online) Journal homepage: www.tandfonline.com/journals/tetn20

FPGA implementation of low complexity LDPC iterative decoder

Shivani Verma & Sanjay Sharma

To cite this article: Shivani Verma & Sanjay Sharma (2016) FPGA implementation of low complexity LDPC iterative decoder, International Journal of Electronics, 103:7, 1112-1126, DOI: 10.1080/00207217.2015.1087052

To link to this article: https://doi.org/10.1080/00207217.2015.1087052

INTERNATIONAL JOURNAL OF ELECTRONICS, 2016
VOL. 103, NO. 7, 1112–1126
http://dx.doi.org/10.1080/00207217.2015.1087052

Taylor & Francis
Taylor & Francis Group

FPGA implementation of low complexity LDPC iterative decoder

Shivani Verma and Sanjay Sharma

Electronics and Communication Engineering Department, Thapar University, Patiala, India

ABSTRACT

Low-density parity-check (LDPC) codes, proposed by Gallager, emerged as a class of codes which can yield very good performance on the additive white Gaussian noise channel as well as on the binary symmetric channel. LDPC codes have gained lots of importance due to their capacity achieving property and excellent performance in the noisy channel. Belief propagation (BP) algorithm and its approximations, most notably min-sum, are popular iterative decoding algorithms used for LDPC and turbo codes. The trade-off between the hardware complexity and the decoding throughput is a critical factor in the implementation of the practical decoder. This article presents introduction to LDPC codes and its various decoding algorithms followed by realisation of LDPC decoder by using simplified message passing algorithm and partially parallel decoder architecture. Simplified message passing algorithm has been proposed for trade-off between low decoding complexity and decoder performance. It greatly reduces the routing and check node complexity of the decoder. Partially parallel decoder architecture possesses high speed and reduced complexity. The improved design of the decoder possesses a maximum symbol throughput of 92.95 Mbps and a maximum of 18 decoding iterations. The article presents implementation of 9216 bits, rate-1/2, (3, 6) LDPC decoder on Xilinx XC3D3400A device from Spartan-3A DSP family.

ARTICLE HISTORY
Received 2 August 2013
Accepted 31 May 2015

KEYWORDS
LDPC; iterative decoding algorithms; parity-check matrix; check-node unit; SMPA

1. Introduction

Low-density parity-check (LDPC) codes are error correction codes that allow communication on the noisy channels near the Shannon capacity limits. LDPC codes play a major role for the reliable communication system. LDPC codes were first introduced by Robert Gallager in 1963 (Gallager, 1963). LDPC codes were ignored for 30 years because of their high computational complexity for hardware technology at that time. Then LDPC codes were rediscovered by MacKay and Neal in 1990 (MacKay, 1999).

The LDPC matrix is represented in (m, n) form, where m and n are number of ones in each column and row, respectively, and else are all zeros. The code rate of the (m, n) matrix is $1 - m/n$. LDPC codes can also be represented with the connected set of graph called tanner graph. Tanner graph directly maps the parity-check matrix into nodes which is easily interpretable. In this graph, variable nodes are equal to the number of columns and check nodes are equal to the number of rows in the matrix.

Tanner graph is called as a bipartite graph as the variable and check nodes can be partitioned into two sets and the only edges of the graph are between the variable nodes and the check nodes (Figure 1). Variable nodes and check nodes are connected to each other by undirected wire that is why it is also called as undirected bipartite graph. In LDPC, a random parity check matrix gives

CONTACT Sanjay Sharma sanjay.sharma@thapar.edu

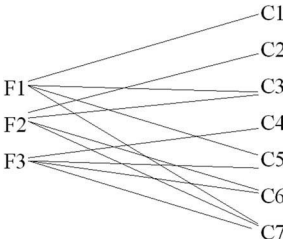

Figure 1. Tanner graph representation of LDPC parity-check matrix.

better performance in noisy channel but the hardware implementation of random matrix is complex and throughput is low. But when Mackay and Neal re-introduced the LDPC code, they proved that optimally designed LDPC code had capability to perform near Shannon limit (MacKay, 1999).

Large numbers of LDPC algorithms have been introduced with varying complexity and performance. A new merged-schedule merge-passing algorithm has been proposed by Mansour and Shanbhag (2003) that reduces the memory overhead of the current algorithm for low to moderate-throughput decoders. Moreover, a parallel soft-input-soft-output (SISO) message update mechanism has been proposed by Mansour and Shanbhag (2003) that implements the recursions of the Bahl–Cocke–Jelinek–Raviv algorithm in terms of simple "max-quartet" operations that do not require look-up tables and incur negligible loss in performance compared with the ideal case. However, trade-off between decoder complexity and decoder performance (like throughput, number of iterations, BER) is still a problem. The bit error rate (BER) performance of LDPC code down to 10^{-10} can be reached within 2 h using the FPGA platform. For practical implementation complexity and power consumption, 2 channel iterations and 2 TPC decoder iterations are employed (Lingyan, Hongwei, Keirn, & Kumar, 2006). Various log-likelihood-ratio-based belief-propagation (LLR-BP) decoding algorithms and their reduced-complexity derivatives for LDPC codes have been presented by Chen, Dholakia, Eleftheriou, Fossorier, and Hu (2005). Iterative log belief-propagation (BP) algorithm, based on soft-decision decoding, has high decoding capability which showed high decoding complexity. In contrast, Bit-Flip algorithm, based on hard-decision decoding, has low decoding complexity yielded poor performance. So for trade-off between complexity and performance, simplified message passing algorithm (SMPA) was introduced (Chandrasetty & Aziz, 2011). Efficient implementations of the sum-product algorithm were presented by Xiao-Yu, Eleftheriou, Arnold, and Dholakia (2001) for decoding LDPC codes using log-likelihood ratios (LLR) as messages between symbol and parity-check nodes. Decoder architecture is another important factor that determines the complexity and throughput of the decoder. Fully parallel architecture has high throughput but has very high decoder complexity. As the code length increases, complexity also increases. So complexity limits the maximum code length to be decoded. Serial decoder architecture not only has low decoder complexity but also has low throughput. Its throughput decreases with increase in the code length. So, partially parallel decoder was introduced (Zhang & Parhi, 2001a,b) to trade-off between throughput and complexity. Boutillon, Conde-Canencia, and Al Ghouwayel (2013) presented the architecture, performance and implementation results of a serial GF (64)-LDPC decoder based on a reduced-complexity version of the Extended Min-Sum algorithm. The implemented decoder presents performance at less than 0.7 dB from the belief propagation algorithm for different code lengths and rates. Masera, Quaglio, and Vacca (2007) presented a novel formulation of the decoding algorithm that strongly simplifies internal communication requirements and enables the development of decoders supporting generally defined LDPC codes. An efficient programmable architecture coupled with a scalable and dynamic transport network for storing and routing messages has been proposed by Mansour and Shanbhag (2003). Simulation results demonstrate that the proposed architecture attains a

throughput of 1.92 Gb/s for a frame length of 2304 bits, and achieves savings of 89.13% and 69.83% in power consumption and silicon area over state-of-the-art, with a reduction of 60.5% in interconnect length.

2. Message-passing iterative decoding algorithms

Message-passing decoding iterative algorithms involve exchange of messages between the check nodes and variable nodes discretely on the time axis. The algorithm involves exchange of messages between the variable nodes and the check nodes and this process of exchange goes on till a certain number of iterations are done. Following steps illustrate the exchange sequence of messages:

(1) Each variable node sends a message belonging to some message alphabet to the check node. Initially, it can be the random received value of the variable node.
(2) Each check node processes the messages it receives from its adjacent variable nodes, and sends back a suitable message to each of its neighbouring variable nodes.
(3) Next, each variable node produces new messages from the messages received from the check nodes and its own received value to send to its neighbouring check nodes.

The above sequence continues for many time steps, till a certain number of iterations are reached. The algorithm ensures that only extrinsic information is exchanged between the two nodes i.e., message transmitted along a particular edge is not correlated to the message just received along that edge. The transmitted bit can be estimated as +1 or −1 depending on the sign of the message. The reliability of the estimate made can be determined from the absolute value of the message. For the case of erasure, when the value is 0, the sign can be +1 or −1 with equal probability. (Richardson & Urbanke, 2001).

Iterative decoding algorithms can be categorised into hard-decision and soft-decision algorithms. The message alphabet can be binary (i.e. can have value 0 or 1), or infinite (e.g., set of real numbers). The algorithm is referred to as hard-decision if the message alphabet is binary; otherwise, it is called soft-decision. The computational complexity of iterative decoding algorithms is directly related to the alphabet size of messages.

2.1. *Hard-decision algorithms*

Hard decision algorithms are very simple to implement. In these algorithms, only binary values are used throughout the decoding process. For example, Gallager's algorithm (Gallager, 1962) is a hard-decision decoder in the set of majority based algorithms (Boutillon et al., 2013) with alphabet A = {−1; 1}. Bit Flipping algorithm (BF) (Gallager, 1962) is another example of hard-decision algorithms. The BF algorithm defines a flipping function for counting the number of unsatisfied syndrome bits (check nodes) in which each variable node participates. Each variable node has a binary buffer to store a hard decision value; the content of this buffer will be flipped if the corresponding output of the flipping function is more than a certain threshold. Decoding continues till all the check node equations are satisfied or until the maximum number of iterations is reached.

2.2. *Soft-decision algorithms*

Soft-decision decoding algorithms, in contrast to hard-decision algorithms, have a continuous alphabet. Examples of soft-decision algorithms are belief propagation (BP) (Chen et al., 2005; MacKay, 1999;Richardson & Urbanke, 2001) and Weighted Bit Flipping algorithms (WBF).

The complexity of BP algorithm is very high but it is the most optimal LDPC decoding algorithm from the error rate point of view. Min-sum algorithm (Fossorier, 2001; Fossorier, Mihaljevic, & Imai, 1999; Madi, Mansouri, & Ahaitouf, 2012; Xiao & Banihashemi, 2004) is a reduced complexity approximation of BP. Xiao-Yu et al., 2001 gave a review of reduced complexity algorithms. The

family of WBF algorithms shows a good trade-off between performance and complexity than BP. The WBF family consists of modified weighted bit flipping (MWBF) (Zhang & Fossorier, 2004) and improved modified weighted bit flipping (IMWBF) (Jiang, Zhao, Shi, & Chen, 2005). Unlike BF, these algorithms involve switching of only one bit per iteration.

2.2.1. *Belief propagation algorithm*

Let v_i, where $i = 1,\ldots, N$, and c_j, where $j = 1, \ldots, M$, denote the set of variable nodes and the set of check nodes, respectively. Let m_{ab} denote the message sent from node a to node b.

Initialisation: Let y_i represent received values from additive white Gaussian noise channel. Then variable node messages are given by

$$m_{vcij} = \frac{2}{\sigma^2} y_i \tag{1}$$

Check node operation:

$$m_{vcij} = 2\ \tanh^{-1}\left(\prod_{vk \in \frac{N(cj)}{vk}} \tanh\left(\frac{m_{vckj}}{2}\right)\right) \tag{2}$$

Variable node operation:

$$m_{vcij} = m_{v_i} + \sum_{Ck \in M(vi)/cj} m_{cvki} \tag{3}$$

where $m_{v_i} = \frac{2}{\sigma^2} yi$.

Hard decision:

$$x_{v_i} = m_{v_i} + \sum_{Ck \in M(vi)} m_{cvki} \tag{4}$$

Quantisation is done such that $z_i = 0$ if $x_{v_i} = 0$, otherwise $z_i = 1$. Decoding is considered as complete if condition $zH^T = 0$ is satisfied. If the condition fails go to step 2. Decoding failure is declared if a code word cannot be found within some specified maximum iterations.

Implementation of an iterative algorithm can be done in probability, likelihood ratio (LR) or log-likelihood ratio (LLR) domain. Practically, the most widely used domain is LLR domain as it involves less complex and more robust message-passing algorithms.

2.2.2. *Log-belief propagation decoding algorithm*

BP decoding algorithm implementation is very complex due to the use of various multiplication operations. The complex multiplication operations can be reduced to simple addition operations by implementing BP decoding algorithm in logarithmic domain (Zhang & Parhi, 2003). For explaining log-BP algorithm let us consider m variable nodes and n check nodes. Let a be a subset of m, i.e. a set of variable nodes which are adjacent to a particular check node b and b be a subset of n, i.e. a set of check nodes which are adjacent to a particular variable node a.

(1) Compute likelihood ratio of each variable node; $LR = \ln \frac{P[x_a = 0]}{P[x_a = 1]}$

(2) Compute intrinsic message, i.e.

$$\mathbf{VC}_{ba} = \overline{LR} \ln \frac{(\exp(-|LR|) + 1}{-(\exp(-|LR|) + 1} \tag{5}$$

where $\overline{LR} = \begin{cases} +1 & \forall\ LR \geq 0 \\ -1 & \forall\ LR \leq 0 \end{cases}$.

(3) Compute check-to-variable message for each check node b, i.e.

$$\mathbf{CV}_{ba} = \ln \frac{\exp(-\beta)+1}{-\exp(-|\beta|)+1} \prod_{j \in n(b)/a} \overline{\mathbf{VC}_{bj}} \tag{6}$$

where $\beta = \sum_{j \in N(b)/a} |\mathbf{VC}_{bj}|$.

(4) Updating the likelihood ratio value. The updated value of likelihood ratio is given by

$$ULR = LR + \sum_{k \in M(a)/b} |\mathbf{CV}_{ka}|$$

(5) The *ULR* value is used to compute finally the variable-to-check message.

$$\mathbf{VC}_{ba} = \overline{ULR} \ln \frac{(\exp(-|ULR|)+1}{-(\exp(-|ULR|)+1} \tag{7}$$

(6) Finally, LLR can be computed as $LLR = LR + \sum_{b \in M(a)} \mathbf{CV}_{ba}$.

Decision is made on the basis of sign of LLR. If LLR comes out to be positive, bit is assumed to be zero whereas it is taken as one if LLR comes out to be negative or zero.

3. Proposed algorithm

This article proposes a simplified message passing algorithm. The different computation steps of the proposed algorithm are listed below:

(1) Compute likelihood ratio of each variable node; $LR = \ln \frac{P[x_a = 0]}{P[x_a = 1]}$.

(2) Compute intrinsic message i.e. $\mathbf{VC}_{ba}$ = MSB (*LR*) where, MSB (*LR*) denotes most significant bit of *LR*, i.e. sign bit.

(3) Compute check-to-variable message for each check node *b*, i.e.

$$\mathbf{CV}_{ba} = \mathrm{VC}_{b,1} \text{ xor } \mathrm{VC}_{b,2} \text{ xor } \ldots \text{ xor } \mathrm{VC}_{b,j} \quad \text{where } j \in N(b)/a$$

(4) Updating the likelihood ratio value. The updated value of likelihood ratio is given by

$$ULR = LR + \sum_{k \in M(a)/b} Y_{ka}$$

where $\mathrm{Y}_{ba} = \begin{cases} +g & \text{if} \quad \mathrm{CV}_{ba} > 0 \\ -g & \text{if} \quad \mathrm{CV}_{ba} < 0 \end{cases}$

(5) The *ULR* value is used to compute finally the variable-to-check message:

$$\mathbf{VC}_{\mathrm{ba}} = MSB(ULR)$$

(6) Finally, LLR can be computed as $LLR = LR + \sum_{b \in M(a)} \mathbf{CV}_{ba}$.

Decision is made on the basis of sign of LLR. If LLR comes out to be positive, bit is assumed to be zero whereas it is taken as one if LLR comes out to be negative or zero. The proposed algorithm provides trade-off between low decoding complexity and decoder performance.

4. Construction of parity-check matrix

The (3, 6) parity-check matrix **P** can be constructed by using two regular and fixed matrices and one random matrix. (Li, Elassal, & Bayoumi, 2004; Zhang, & Parhi, 2001a,b; Zhang & Parhi, 2002). Say **M**1 and **M**2 are two regular and fixed matrices (Figures 2 and 3), each of same order, say ($xy \times xy^2$). Matrix **M**1 is composed of a number of identity matrices, each of order ($x \times x$) whereas matrix **M**2 is composed of a number of "shifted" identity matrices $\mathbf{I}_s$, again of the order ($x \times x$). The "shifted"

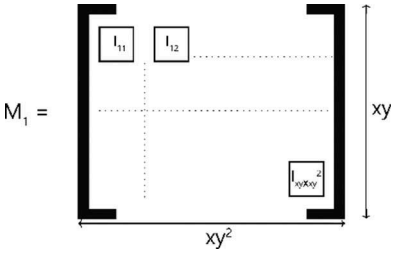

Figure 2. Regular and fixed matrix **M**1.

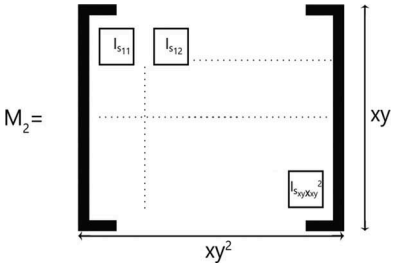

Figure 3. Regular and fixed matrix M2.

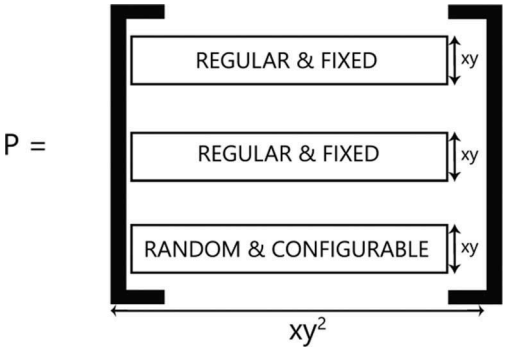

Figure 4. Parity-check matrix P.

identity matrix $\mathbf{I}_s$ may be obtained by shifting the identity matrix towards right by a particular number of times, which can be calculated using "mod" operator.

Say, matrix **M**3, is a random matrix. Its dimensions are of the same order as that of regular and fixed matrices i.e. $(xy \times xy^2)$. Matrix **M**3 is random in the sense that the placement of ones in the matrix and the change in error positions in subsequent iterations are totally random. Thus **P** can be considered as is $\mathbf{P} = [\mathbf{M1}^T, \mathbf{M2}^T, \mathbf{M3}^T]^T$ (Figure 4).

5. Partial-parallel decoder architecture

This article presents the design of partial-parallel decoder architecture having high speed and low complexity (Figure 5). The architecture has been designed for (3, 6) parity-check matrix. (Parhi, 1999; Zhang & Parhi, 2001a,b).

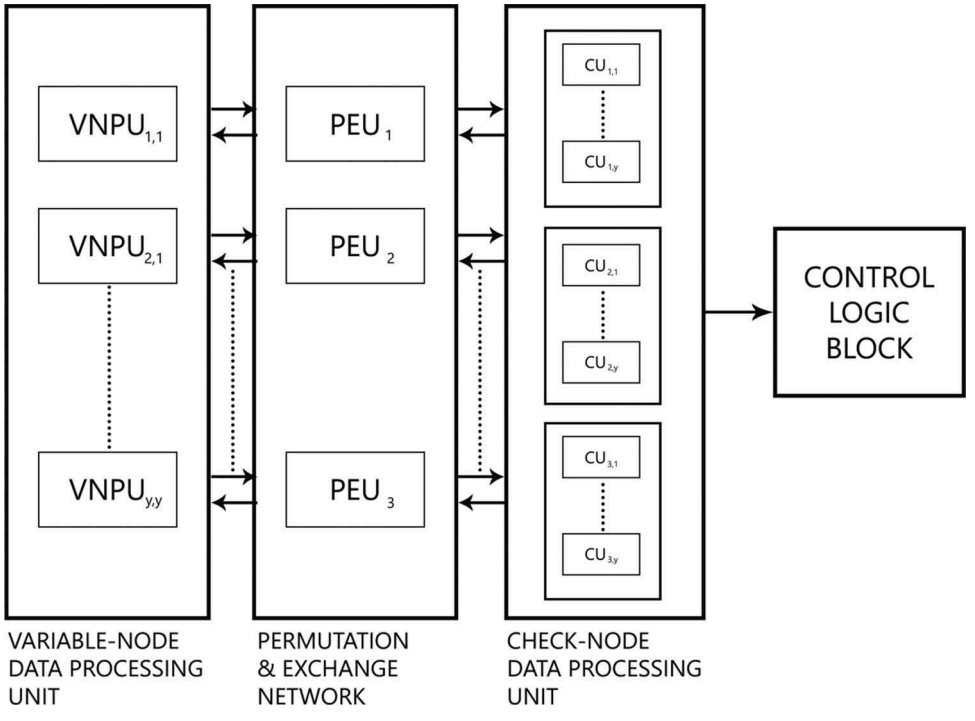

Figure 5. Principal partial-parallel decoder architecture.

The architecture presents implementation of:

(1) Variable node data processing unit (VNDPU): Each VNDPU has 36 variable node processing units (VNPUs). Each VNPU further consists of RAM memory blocks: three extrinsic RAM (E-RAM) blocks each of size (256 × 1), one Intrinsic RAM (I-RAM) block of size (256 × 8) and one decision RAM (D-RAM) block of size (256 × 1), in addition to a computation unit for performing variable node computations (Figure 6). Each E-RAM block has x memory locations and any memory location out of these x available locations stores the extrinsic information exchanged between a particular variable node and its neighbouring check node. This is great reduction size of RAM as given by Lingyan et al. (2006).

(2) Check node data processing unit (CNDPU): Each check node data processing unit (CNDPU) has 3 check units (CUs) for performing check node computations. Each CU further contains 6 check nodes.

(3) Permutation and exchange network (PEN): The PEN consists of 3 permutation and exchange units which are used for permutation and exchange of messages between CNDPU and VNDPU. Exchange of information between variable node and check node is done through bidirectional permutation and exchange network. During variable-to-check message, permutation is done and during check-to-variable message, reverse permutation is done.

(4) Control block: All the CNUs communicate the parity check results to a control block which will determine if all the parity check equations specified by the parity-check matrix have been fulfilled.

In the each decoding iteration, the proposed decoder works in two modes: VNP (variable node processing) and CNP (check node processing) mode. The decoder processes each iteration in $(2x + 4)$ cycles of processing time, with each processing mode occupying half of the processing cycles.

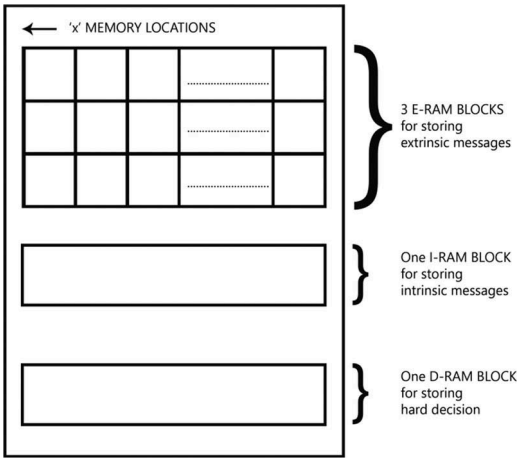

Figure 6. Memory blocks in VNDPU.

5.1. *Variable node data processing unit (VNDPU)*

VNDPU consists of 36 VNPUs in the proposed architecture. Each VNPU performs three operations:

(1) Accessing 1-bit check-to-variable messages from the E-RAM block;
(2) Computation of 1-bit hard decision, 1-bit variable-to-check messages and updated value of LLR from the stored 5-bit intrinsic message and the check-to-variable message;
(3) Storing the computed values in their respective memory blocks i.e. hard decision in the D-RAM block, updated LLR in the I-RAM block and the computed variable-to-check messages in the E-RAM block.

Thus, as input VNPU has three 1-bit check-to-variable messages, VNPU also has 5-bit intrinsic message during initialisation and 8-bit intrinsic message during decoding iteration as input. After processing and the required computations, VNPU outputs three 1-bit variable-to-check messages, one 1-bit hard decision value and one 8-bit intrinsic message. The variable-to-check messages are stored in the E-RAM, hard-decision value is stored in the D-RAM and intrinsic message is stored in the I-RAM block.

5.1.1. *Architecture of VNU*

The architecture shown in the Figure 7 realises the variable node unit of SMPA algorithm. All the input and output messages are in 2's complement format. In this architecture, there are six 8-bit binary adders and MSB of intrinsic output i.e. updated LLR is used as hard-decision bit.

The architecture also associates one address decoder with each E-RAM block. The address decoders (AD1, AD2 and AD3) provide the read address for variable-to-check messages and write address for check-to-variable messages. The three address decoders along with the three permutation and exchange units (AD1 with PEU1; AD2 with PEU2; AD3 with PEU3) jointly specify the complete connectivity between the variable nodes and the check nodes. All the three address decoders are implemented as mod-8 binary counters, counting from 0 to (2^8–1), and are initialised to value '0' while the decoder is performing variable node computations. While in check node computation mode, the three decoders are initialised to different values. AD1 is initialised to value '0'; AD2 is initialised to value $((m-1)\cdot n) \bmod x$ for a particular VNPU_{mn}; AD3 is randomly initialised as it is used for realisation of random matrix M3. AD3 can be randomly initialised to a value p if for a given value of m but for two different values of n, the value of p is also different. Another constraint that needs to be checked for p is that for one value of n but for two different values of m, the difference between two p values is given by $((m_1 - m_2) \bmod x)$.

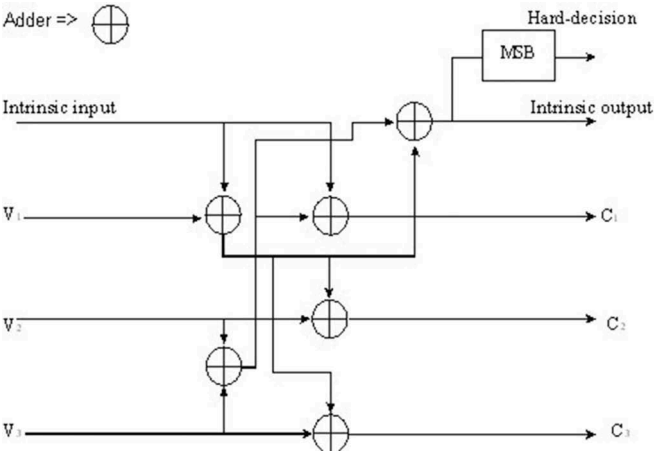

Figure 7. Architecture of VNU.

5.2. *Permutation and exchange network*

Permutation and Exchange network provides connectivity between the variable nodes and the check nodes as specified by 3 sub-matrices **M**1, **M**2 and **M**3. PEN in the proposed architecture consists of 3 permutation and exchange units (PEUs) which provide for permutation, reverse permutation and exchange of data between variable nodes and check nodes.

(1) PEU1 connects $VNPU_{mn}$ to $CU_{1,z}$, thus connecting six VNPU elements with the CU having the same *x*-index.
(2) PEU2 connects $VNPU_{mn}$ to $CU_{2,z}$, thus connecting six VNPU elements with the CU having the same *y*-index.
(3) PEU3 connects $VNPU_{mn}$ to $CU_{3,z}$. This connection is required to be of random type so that randomness of matrix **M**3 is maintained. PEU3 permutes (or reverse permutes) data in two stages as shown in Figure 8.

5.3. *Check node data processing unit (CNDPU)*

The CNDPU performs certain operations to convert variable-to-check extrinsic message to check-to-variable message as listed below:

(1) Three single-bit variable-to-check messages are accessed from the E-RAM blocks.
(2) The accessed data is passed through a permutation and exchange unit of permutation and exchange network.
(3) Computations are performed on the permuted data and variable-to-check extrinsic messages are generated from the check-to-variable messages. In addition, parity check is also performed on the hard decision bits and the results are communicated to the control block that checks if all the parity check conditions specified by the parity-check matrix **P** have been satisfied or not. If the conditions are satisfied, decoding process can be stopped.
(4) The computed data from CNDPU is again passed through permutation and exchange unit of PEN for reverse permutation process (Figure 9).
(5) The resultant message after reverse permutation is written back to the *i*th memory location of E-RAM, from where the message was initially fetched.

5.3.1. *Architecture of CU*

Each CU perform operation of 1 check node i.e. computation of check-to-variable messages. Figure 10 shows the architecture of CU. As shown in the figure, the complexity of CU is less than

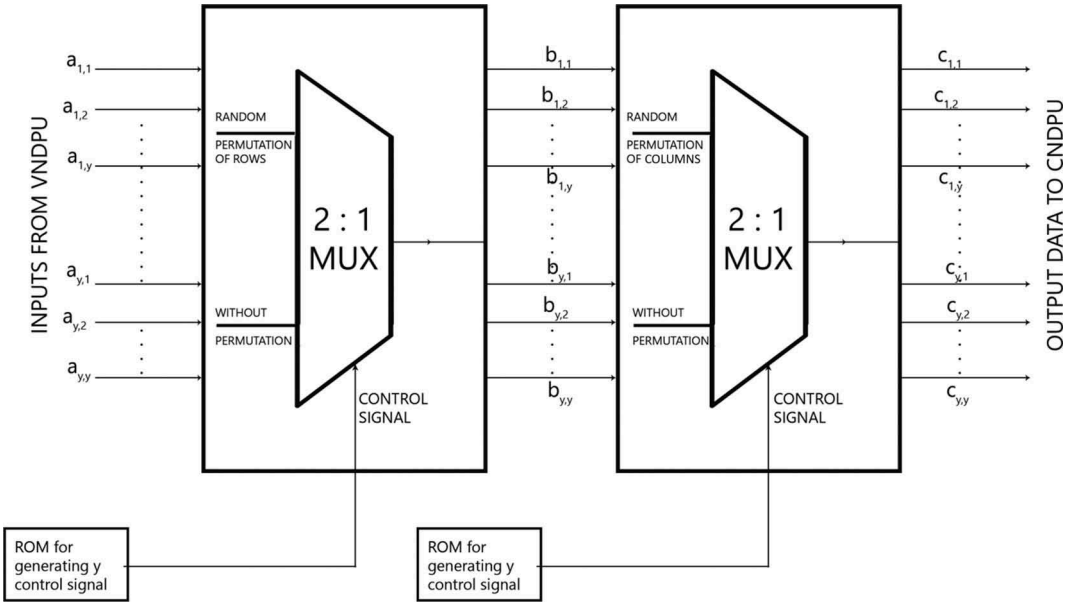

Figure 8. Block diagram of PEU3.

architectural complexity of Log-BP algorithm. It consists of only '12' XOR gates. Input to CU is k i.e. 6-bit variable-to-check message coming from '6' VNPUs and the output is also of 6-bits sending to various VNPUs. This architecture realises the "check node computation" of the SMPA algorithm completely (Zhang, Wang, & Parhi, 2001).

The complete processing of data through VNDPU, PEN and CNDPU is shown in Figure 11.

6. Increasing throughput of the proposed decoder

The throughput of the proposed architecture can be increased by using pipelining process. In the proposed scheme, pipelining has been used at two levels: at the sub-unit level and at the decoder level. At the sub-unit level, pipelining scheme is used to pipeline the operations on CNDPU and VNDPU. At the decoder level, pipelining of processing of consecutive frames is done so as to increase the overall throughput.

6.1. *Pipelining in VNDPU*

Full-duplex connections are used for realising the connections between the VNPUs and the different memory banks. Thus pipelining technique can be used for increasing the throughput. The address for writing the check-to-variable messages is same as the address from where the variable-to-check messages were initially accessed. Thus, assuming that VNPU takes 3 clock cycles to perform 3 operations, after a delay of 3 clock cycles the write address can be obtained.

6.2. *Pipelining in CNDPU*

As the architecture of CNDPU is not very complex, thus apart from accessing and writing the data on to the *i*th memory location, the permutation, computation and reverse permutation steps can

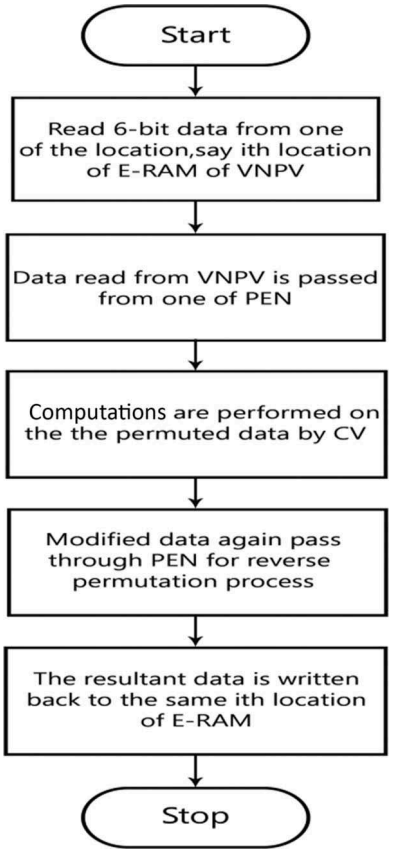

Figure 9. Flow chart showing CNDPU computations.

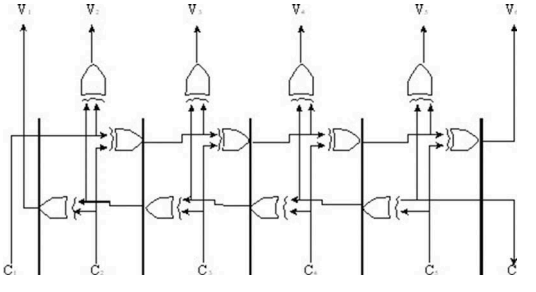

Figure 10. Architecture of CU.

be performed in a single clock cycle. Thus overall 3 clock cycles are used for CNDPU operations. Also, full duplex connections are used for interfacing various CUs of CNDPU with PEN.

6.3. *Pipelining at decoder level*

In the proposed architecture, the processing of consecutive frames is pipelined to achieve higher throughput. While the decoder is reading out 1 frame from the decoder, the next frame is iteratively decoded and the third frame is loaded into the decoder. Thus, the decoder works on

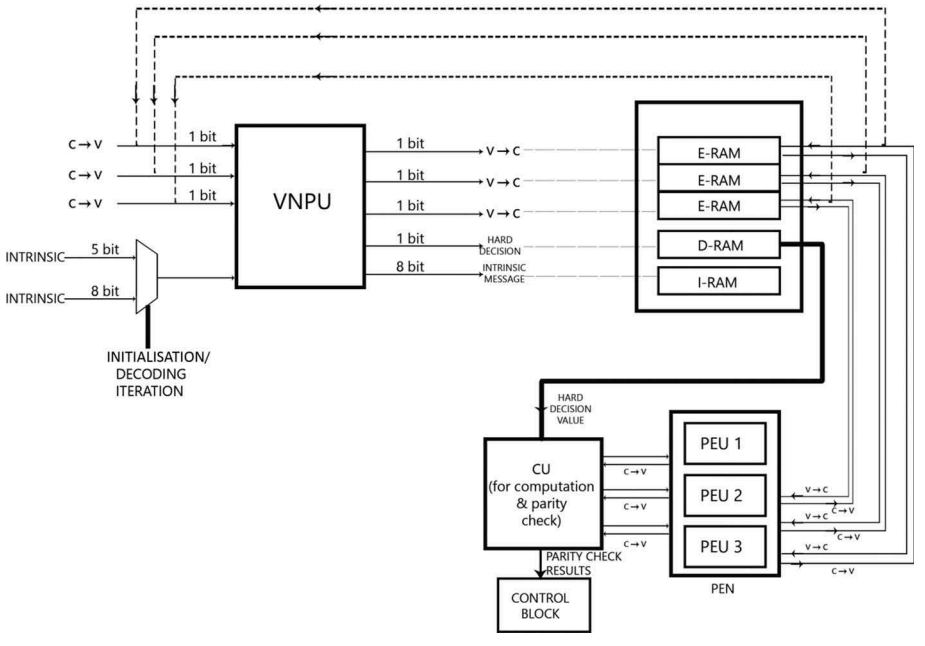

Figure 11. Processing data path through VNDPU, PEN and CNDPU.

3 code frames simultaneously. To realise this pipelining, 2 I-RAMs and 2 D-RAMs are used. The first I-RAM is (256 × 5) RAM because of 5-bit quantisation of LLR i.e. initial value of intrinsic message and the second I-RAM is of (256 × 8) because the value of LLR is updated to 8-bit quantisation for further processing of iterative decoding. Input frame is loaded into the first I-RAM and the second I-RAM is used for storing the intrinsic message during iterative decoding. Output frame is read out from the first D-RAM of decoder and the second D-RAM is used for storing the hard-decision during iterative decoding.

7. Simulation

The decoder gets triggered at the positive edge of the clock. The global clock signal 'clk' is used for synchronous operation. The LLR information is fed through the signal 'din' using 'load' control signal. 'I1' signal gives the memory address of I-RAM in which input LLR information is loaded. Signal 'c' shows the VNU in which LLR information is loaded. The complete process of loading intrinsic information takes around 9474 clock cycles (258 for initialisation and 9216 for input). After decoding the input, 1-bit output is obtained as decoded data using a high 'dataout_ready' control signal. The decoding and retrieval of output from the decoder is synchronous with the clock. Figure 12 shows the block diagram of LDPC decoder along with the control signals. In this architecture, the number of input and output pins used is less compared with architecture proposed by Zhang and Parhi (2002). Output is obtained by using only 1 pin rather than 36 pins. Also, this architecture does not need '14' pins for loading the address in each 'Int' memory of 36 variable nodes (Figure 13).

Figure 14 shows the simulation result at the end of '18' iterations. This 'int' signal shows the number of iterations at end of clock pulse '19092'.

Figure 15 shows the starting of decoded output at the port 'dout'. This port is synchronous with 'dataout_ready' port. Output is available in the next positive edge of clock cycle when the 'dataout_ready' signal is 'high'.

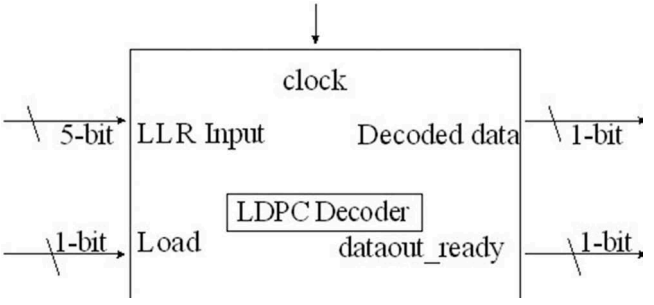

Figure 12. Control signals of LDPC decoder.

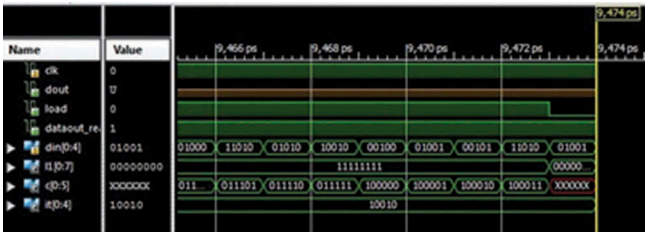

Figure 13. Decoder has loaded all the LLR information.

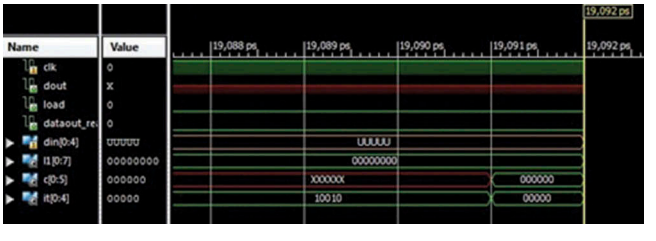

Figure 14. Decoder status after all iterations.

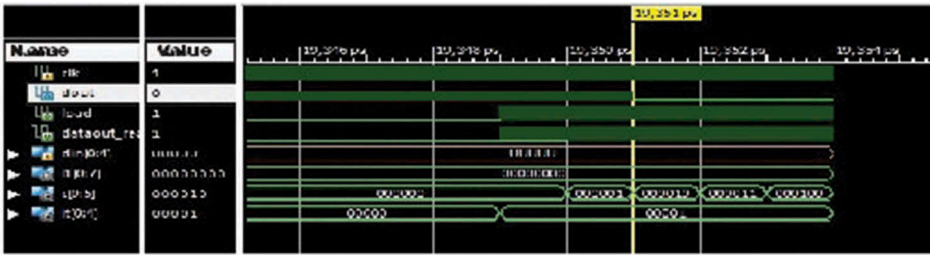

Figure 15. Start of decoded information at output.

8. FPGA implementation

We apply partially parallel architecture for (3, 6)-regular LDPC code and $L = 256$ on Xilinx XC3D3400A device from Spartan-3A DSP family. In this implementation of decoder, we configure:

- First and second intrinsic RAMs i.e. 'Int' RAM is configured as dual port block RAM. The size of first RAM is (256×5) i.e. 1.25 K-bit RAM. The size of second RAM is (256×8) i.e. 2 K-bit RAM.

- First and second decision RAMs i.e. 'Dec' RAM is configured as dual port distributed RAM. The size of first and second decision RAMs is 256 × 1 (i.e. 256 bits RAM).
- Each variable node contains 3e extrinsic RAMs i.e. E1, E2 and E3. These RAMs is configured as distributed RAM. The size of these RAM is 256 × 1 (i.e. 256 bits RAM).

This decoder was implemented using VHDL hardware description language (HDL) and a number of resources utilised by FPGA have been described by synthesis of the VHDL implementation. We used the Xilinx Development System tool suite to place and to route the synthesised implementation for the target XC3D3400A device with the speed option-5. Table 1 shows the statistic of hardware resource utilisation.

Maximum clock frequency suggested by Xilinx Development System tool suite is 96.28 MHz. If we have 'a' decoding iterations for each code frame, then number of clock pulses utilises for decoding the one frame of input will be $(2a\ (L + 2) + (L + 2))$. Here, '2' is added due to number of pipelining stages and extra $(L + 2)$ is due to initialisation process. Maximum symbol throughput will be $(96.28) \cdot k^2.\ L/(2a\ (L + 2) + (L + 2)) = 96.28\ (9216/9546) = 92.95$ Mbps if maximum decoding iteration is '18'. Figure 16 shows placed and route decoder implementation using Xilinx FPGA editor tool.

9. Conclusion

In this work, LDPC decoder has been developed with high decoding throughput and less area consumption and implementation has been done on the Xilinx XC3D3400A FPGA device. The improved decoder provides maximum decoding throughput of 92.95 Mbps in 18 iterations and maximum slice utilisation is 7021. The proposed SMPA algorithm used in this article greatly reduces the resources utilisation. The hardware resource utilisation result obtained by the partially parallel decoder is very less compared with fully parallel decoder architecture.

Table 1. FPGA resource utilisation.

Resources	In Arch. (Lingyan et al., 2006)	In this Arch.
Slices	11,792	7021
Slices reg.	10,105	1558
4 – input LUT	15,933	12,872
IOBs	68	9
Block RAMS	90	72

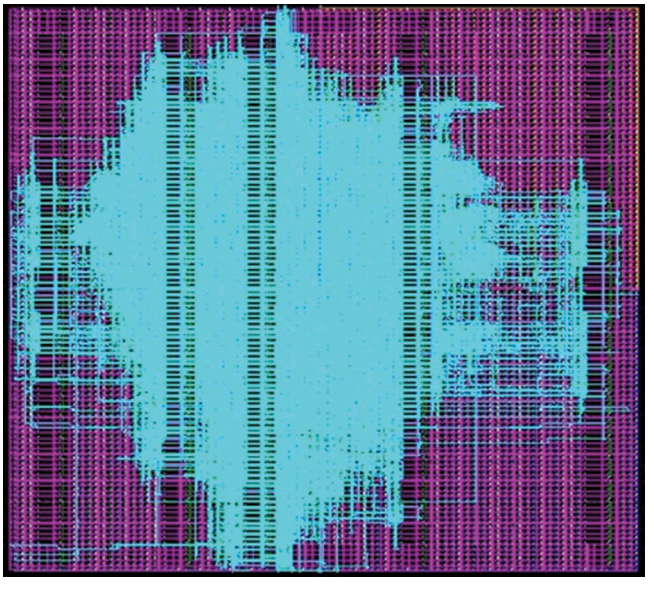

Figure 16. Placed and route decoder implementation.

Disclosure statement

No potential conflict of interest was reported by the authors.

References

Boutillon, E., Conde-Canencia, L., & Al Ghouwayel, A. (2013). Design of a GF (64)-LDPC decoder based on the EMS algorithm. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *60*(10), 2644–2656. doi:10.1109/TCSI.2013.2279186

Chandrasetty, V. A., & Aziz, S. M. (2011). FPGA Implementation of a LDPC decoder using a reduced complexity message passing algorithm. *Journal of Networks*, *6*(1), 36–45. doi:10.4304/jnw.6.1.36-45

Chen, J., Dholakia, A., Eleftheriou, E., Fossorier, M. P. C., & Hu, X.-Y. (2005). Reduced-complexity decoding of LDPC codes. *IEEE Transactions on Communications*, *53*(8), 1288–1299. doi:10.1109/TCOMM.2005.852852

Fossorier, M. (2001). Iterative reliability-based decoding of low-density parity check codes. *IEEE Journal Select Areas Communicable*, *19*, 908–917. doi:10.1109/49.924874

Fossorier, M., Mihaljevic, M., & Imai, H. (1999). Reduced complexity iterative decoding of low-density parity check codes based on belief propagation. *IEEE Transactions Communicable*, *47*, 673–680. doi:10.1109/26.768759

Gallager, R. (1963). *Low density parity check codes*. Cambridge, MA: Massachusetts Institute of Technology.

Gallager, R. G. (1962, January). Low-density parity check codes. *IRE Transactions on Information Theory*, *39*(1), 37–45.

Jiang, M., Zhao, C., Shi, Z., & Chen, Y. (2005). An improvement on the modified weighted bit-flipping decoding algorithm for LDPC codes. *IEEE Communications Letters*, *9*, 814–816. doi:10.1109/LCOMM.2005.1506712

Li, Y., Elassal, M., & Bayoumi, M. (2004). Power efficient architecture for (3, 6) low density parity check code decoder. In *Circuits and Systems, 2004, ISCAS '04.Proceedings of the 2004 International Symposium*, Los Angles, CA (Vol. 4, pp. IV–81-4). doi:10.1109/ISCAS.2004.1328945

Lingyan, S., Hongwei, S., Keirn, Z., & Kumar, B. V. K. V. (2006). Field programmable gate array (FPGA) for iterative code evaluation. *IEEE Transactions on Magnetics*, *42*(2), 226–231. doi:10.1109/TMAG.2005.861744

MacKay, D. J. C. (1999). Good error-correcting codes based on very sparse matrices. *IEEE Transactions on Information Theory*, *45*, 399–431. doi:10.1109/18.748992

Madi, A. A., Mansouri, A., & Ahaitouf, A. (2012, March). Design, simulation and hardware implementation of low density parity check decoders using min-sum algorithm. *IJCSI International Journal of Computer Science Issues*, *9*(3), 83–91.

Mansour, M. M., & Shanbhag, N. R. (2003). High-throughput LDPC decoders. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *11*(6), 976–996. doi:10.1109/TVLSI.2003.817545

Masera, G., Quaglio, F., & Vacca, F. (2007). Implementation of a flexible LDPC decoder. *IEEE Transactions on Circuits and Systems II: Express Briefs*, *54*(6), 542–546. doi:10.1109/TCSII.2007.894409

Parhi, K. K. (1999). *VLSI digital signal processing systems: Design and implementation*. New York, NY: John Wiley & Sons.

Richardson, T., & Urbanke, R. (2001). The capacity of low-density parity check codes under message passing decoding. *IEEE Transactions Informatics Theory*, *47*, 599–618. doi:10.1109/18.910577

Xiao, H., & Banihashemi, A. H. (2004). Graph-based message-passing schedules for decoding LDPC codes. *IEEE Transactions on Communications*, *52*, 2098–2105. doi:10.1109/TCOMM.2004.838730

Xiao-Yu, H., Eleftheriou, E., Arnold, D. M., & Dholakia, A. A. D. A. (2001). Efficient implementations of the sum-product algorithm for decoding LDPC codes. *Global Telecommunications Conference, 2001. GLOBECOM '01, IEEE*, *2*, 1036–1036.

Zhang, J., & Fossorier, M. (2004). A modified weighted bit-flipping decoding of low-density parity-check codes. *IEEE Communications Letters*, *8*, 165–167. doi:10.1109/LCOMM.2004.825737

Zhang, T., & Parhi, K. K. (2001a, September). VLSI implementation-oriented (3, k)-regular low-density parity-check codes. In *IEEE Workshop on Signal Processing Systems (SiPS)* (pp. 25–36). Antwerp, Belgium. doi:10.1109/SIPS.2001.957328

Zhang, T., & Parhi, K. K. (2001b). Joint code and decoder design for implementation-oriented (3, k)-regular LDPC codes. In *Proceedings of IEEE Asilomar Conference* (pp. 1232–1236). Pacific Grove, CA. doi:10.1109/ACSSC.2001.987687

Zhang, T., & Parhi, K. K. (2002, October 16–18). A 54 MBPS (3, 6)-regular FPGA LDPC decoder. In *Signal Processing Systems, 2002* (SIPS '02), *IEEE Workshop on* (pp. 127–132). doi:10.1109/SIPS.2002.1049697

Zhang, T., & Parhi, K. K. (2003). An FPGA implementation of (3, 6)-regular low-density parity-check code decoder. *EURASIP Journal on Applied Signal Processing, Special Issue on Rapid Prototyping of DSP Systems*, *2003*(6), 530–542. doi:10.1155/S1110865703212105

Zhang, T., Wang, W., & Parhi, K. K. (2001, May). On finite precision implementation of low density parity-check codes decoder. *Proceedings of 2001 IEEE International Symposium on Circuits and Systems* (pp. 202–205). Sydney. doi:10.1109/ISCAS.2001.922207