International Journal of Engineering & Technology, 7 (4.28) (2018) 577 -581

SPC

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Implementation of the Hard-Decision Low Density Parity Check Codes in A 0.13µm CMOS Process

[1]Daryl P. Pongcol, Roberto B. Madronial, Jr., Olga Joy L. Gerasta,Jefferson A. Hora, [2]J.Banuchandar

[1]Microelectronics Laboratory,
Mindanao State University-Iligan Institute of Technology, Iligan City, Philippines
E-mail: jefferson.hora@g.msuiit.edu.ph
[2]Research Scholar, Anna University, Chennai, India.
**Corresponding author Email: jtbanuchandar@gmail.com*

Abstract

This paper presents a simple and efficient implementation of a Low Density Parity Check (LDPC) error-correcting code, using the hard-decision decoding algorithm in a 0.13 µm TSMC CMOS process. The encoder and decoder modules were simulated by transmitting the correct 8bit code words, and letting it pass through a test bench module that corrupts one or more bits of the channel data, then allowing the decoder to correct the corrupted channel data. The system is able to correct a single bit error with 1 or 2 iterations only. Hence for a clock of 50MHz, the system can detect and correct more than 42 bit errors per 1 KB of data, which is just the goal of this research. Implemented through Verilog HDL using Synopsys Design, the design is simple in a sense that it does not use sophisticated encoding and decoding algorithms and the H- matrix used is a simple ½ code rate (4,8)- regular matrix and thus will result to a small scale and non-congested full-parallel architecture. It only measured 6.29 mm^2 chip area. For a supply voltage of 1.32 V, the total power of only 138.64 µW implied very low power consumption.

Keywords: *Decoder; Error correcting code (ECC); Low density parity check (LDPC); Parity check*

1. Introduction

Networks must be able to transfer data from one device to another with complete accuracy. A system that cannot guarantee that the data received by one device are identical to the data transmitted by another device is essentially useless. The most common causes of errors are the interferences, attenuation, distortion and noise in the channel. These transmission limitations occur at some fixed or unpredictable degree and can be very dangerous to the system. This clearly emphasizes the need for error detection and correction. Reliable systems must have a mechanism for handling such errors [1].

There are many such schemes and algorithms existing for error detection and correction. In each of these schemes, two communication blocks are essential: the channel encoder and channel decoder. These blocks facilitate the error detection and correction process by adding some extra bits called parity bits or redundancy bits prior to transmission (encoding), then reading these new set of data during reception and decode to check if there's an error during that instant of transmission (decoding). The best that the encoders and decoders could do is to detect the error and also locate the position of the bit in error and then correct it. This is done through an error-correcting code (ECC) [2].

Basically these encoder and decoder are digital circuitry that can be embedded on the transmitter and receiver sides of a chip-level communication system. There had been error-correcting codes that existed over the years since the invention of the first ECC called the hamming codes, up to until what is the latest ECC currently being used in most systems these days, the low-density parity check codes (LDPC). LDPC codes were first invented by Robert Gallager in his PhD dissertation on 1960. Due to the computational effort in implementing the encoder and decoder, to an era when vacuum tubes were only just being replaced by the first transistor, and the introduction of the Reed-Solomon code, the code was ignored and forgotten [3]. Till then a lot of state-of-the-art coding gain codes were also introduced, particularly the Turbo Code. These codes can achieve coding gains close to the theoretical limit for channel coding, which is the Shannon Limit, but the implementation cost is high [4]. In 1996, LDPC codes were rediscovered by D. MacKay and R. Neal. They found it to have very good coding gain performance, but the implementation cost is much lower.

This paper had aimed to design and implement, using the synthesizable Verilog HDL on the 130 nm CMOS technology process, the low density parity check codes. The researchers' major considerations in the design and implementation of this ECC will be the speed of the encoding and decoding processes, the design to work on a clock of at least 50MHz given the target library to be 90 nm, memory input/output in 8, 16, 32, or 64 bits, and a capability of correcting 42 bit errors per 1KB of information.

2. Codes Basics

Although the goal of error checking is to correct errors, most of the time, it is first necessary to detect errors. Error detection (without correction) is simpler than error correction and is the first step in the error correction process [1].

2.1. Error Correcting Codes

The most common and least expensive scheme for error detection is the *parity check*. Here, a *parity bit* is added to every data unit so that the total number of 1's in the data unit (including the parity bit) becomes even (or odd for odd-parity). This scheme is the most basic and the oldest but more modern and sophisticated algorithms still make use of the parity check in their steps [5].

Use of simple parity check allows detection of single-bit errors in a received message. Correction of these errors requires more information, since the position of the corrupted bit must be identified if it is to be corrected. If a corrupted bit can be detected, it can be corrected by simply complementing its value. Correction is not possible with one parity bit only since any bit error in any position produces exactly the same information, i.e., error. If more bits are included in a message, and if those bits can be arranged such that different corrupted bits produce different error results, then corrupted bits could be identified [6].

Forward error-correction coding (also called 'channel coding') is a type of digital signal processing that improves reliability of the data by introducing a known structure into the data sequence prior to transmission. This structure enables the receiving system to detect and possibly correct errors caused by corruption from the channel and the receiver. In a communication system that employs forward error-correction coding, the digital information source sends a data sequence to an encoder. The encoder inserts redundant (or parity) bits, thereby outputting a longer sequence of code bits, called a 'code word'. These code words can then be transmitted to a receiver, which uses a suitable decoder to extract the original data sequence [6]. Figure 1 illustrates the discussion.

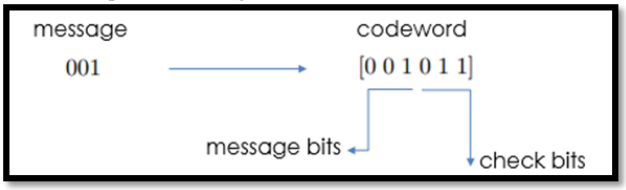


Fig. 1. The Message, check, and code word bits in an FEC encoded message

2.2. LDPC Codes

As their name suggests, LDPC codes are linear block codes with parity-check matrices that contain only a very small number of non-zero entries. An LDPC code parity-check matrix is called (w_c, w_r)-regular if each code bit is contained in a fixed number, wc, of parity checks and each parity-check equation contains a fixed number, w_r, of code bits.

LDPC codes are often represented in graphical form by a *Tanner Graph*. The Tanner graph consists of two sets of vertices: n vertices for the codeword bits (called *bit* or *variable nodes*), and m vertices for the parity-check equations (called *check nodes*). So for the given example of an H-matrix, the corresponding Tanner Graph is as shown in Figure 2.

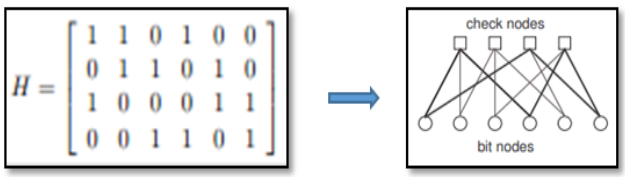


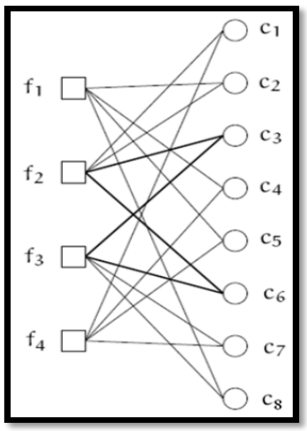
Fig. 2. The relationship with H-matrix and the tanner graph

2.3. The Hard-Decision Algorithm

In [7], Leiner uses a (4, 8) linear block code to illustrate the hard decision decoder and corresponding tanner graph, as shown in Figure 3 (a) and (b), respectively.

$$\mathbf{H} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Fig. 3. (a) A (4, 8) linear block code



(b) Corresponding tanner graph

By the help of an example presented in [8] {4}, the researchers will illustrate how the Hard-Decision decoding algorithm operates. An error free codeword of H is c = [1 0 0 1 0 1 0 1] T. Suppose we receive y = [1 0 0 1 0 1 0 1] T. So c_2 is flipped [9]. The algorithm, as illustrated in [8], is as follows:

Step 1. In the first step, all variable nodes send a message to their connected check nodes. In this case, the message is the bit they believe to be correct for them. For example, message node v_2 receives a 1 ($y_2 = 1$), so it sends a message containing 1 to check nodes c_1 and c_2.

Step 2. In the second step, every check nodes calculate a response to their connected variable nodes using the messages they received from step 1. This is done by having a parity check or an XOR operation to each of the 4 bits received, and if the reduction XOR result is 1, the check node sends to its connected variable nodes the opposite of what it receives, else if the reduction XOR result is 0, the check nodes send the same message it gets. Table 1 illustrates the steps in 1 and 2.

Table 1. Check Nodes Activities for Hard-Decision Decoder

C-node	Received/ sent				
C0	Received: Sent:	1 from V1 0 to V1	1 from V3 0 to V3	0 from V4 1 to V4	1 from V7 0 to V7
C1	Received: Sent:	1 from V0 0 to V0	1 from V1 0 to V1	0 from V2 1 to V2	1 from V5 0 to V5
C2	Received: Sent:	0 from V2 0 to V2	1 from V5 1 to V5	0 from V6 0 to V6	1 from V7 1 to V7
C3	Received: Sent:	1 from V0 1 to V0	1 from V3 1 to V3	0 from V4 0 to V4	0 from V6 0 to V6

What this table illustrates, is that if a check node receives and odd number of ones, it will send a flipped 4-bit value, otherwise if it receives an even number of ones, it will send the same 4-bit number back. So for example in c_0 and c_1 in the table, they both received 1101 from the v-nodes, thus it will send 0010 to their connected v-nodes. C_2 and c_3 received 0101 and 1100 respectively, which both contains even number of ones. Thus, it sends the same 0101 and 1100 respectively.

Step 3. In this step, the variable nodes use messages they get from the check nodes to decide if the bit at their position is a 0 or a 1 by majority rule. The variable nodes then send this hard-decision to

their connected check nodes. Table 2 illustrates this step. To make it clear, let us look at variable node v_2. It receives 2 0's from check nodes c_1 and c_2. Together with what it already has $y_2 = 1$, it decides that its real value is 0. It then sends this information back to check nodes v_1 and v_2.

Table 2. Variable Nodes Decisions for Hard-Decision Decoder

v-node	yi received	messages from check nodes		decision
v0	1	c1-->0	c3-->1	1
v1	**1**	c0-->0	c1-->0	**0**
v2	0	c1-->1	c2-->0	0
v3	1	c0-->0	c3-->1	1
v4	0	c0-->1	c3-->0	0
v5	1	c1-->0	c2-->1	1
v6	0	c2-->0	c3-->0	0
v7	1	c0-->0	c2-->1	1

Step 4. Repeat 2 until either exit at step 2 or a certain number of iterations has been passed. In this example, the algorithm terminates right after the first iteration as all parity check equations have been satisfied. V_2 is corrected to 0.

3. Experimental Results

As shown in Figure 4, we decided to design the system given the LDPC H matrix, G matrix and the corresponding Tanner graph. The corresponding distinct messages and the resulting code words were also presented at the bottom of Figure 5.

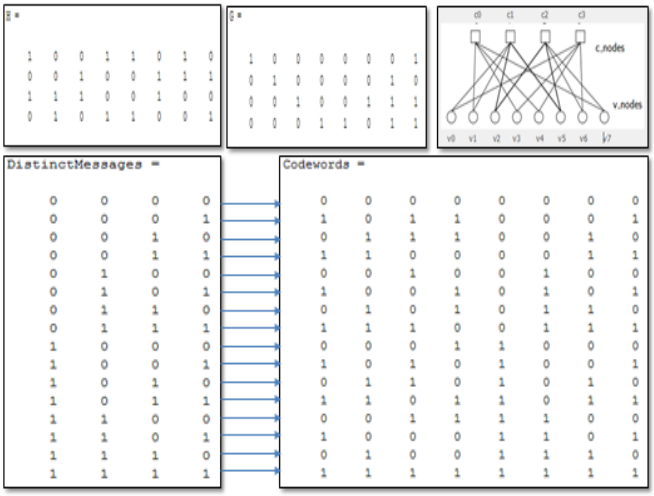
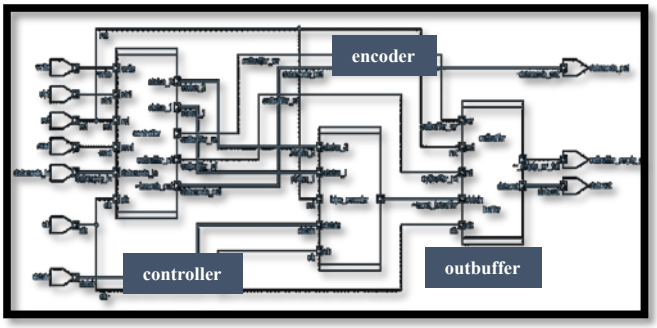


Fig. 4. H and G Matrix and the corresponding tanner graph with the respective distinct messages and code words.

Figure 5(a) shows the main LDPC encoder module and its pinouts. Table 3 shows the details of those pinouts. To help discuss the operation of the main encoder module, first is to look at its internal architecture, shown at Figure 6(b). Like most implemented LDPC encoder modules, each of them consist of a controller, a buffer and a block that does the encoding.

(a)



(b)

Fig. 5. Main LDPC encoder; (a) Module (b) Internal

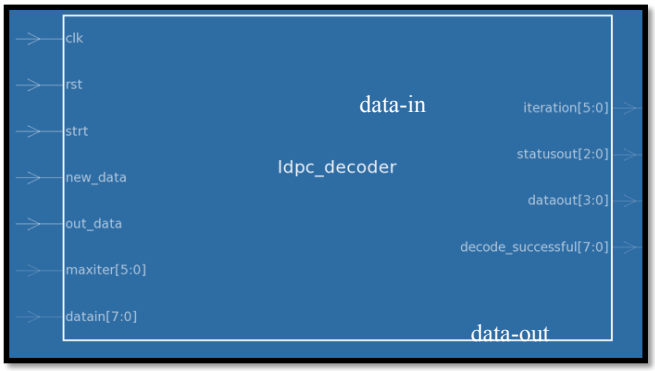
Table 3: Implemented Encoder Port Details

Port	Type	Description	Size (bit)
clk	Input	Master clock of the encoder	1
rst	Input	Master reset of the encoder	1
strt	Input	Master enable of the encoder	1
read	Input	Read signal – Encoder only reads when strt and rd signals are both 1	1
write	Input	Write signal – Encoded output is stored yet on the buffer and is only outputted when this signal is on.	1
outbuffer_empty_or_full	Output	Notifies when the buffer front register hits zero, indicating the buffer is full or already empty	1
datain	Input	Input message	4
dataout	Output	Decoded codeword	8
datamode_in/datamode_out	Input/Output	No Connection/Logic – Port not used	3

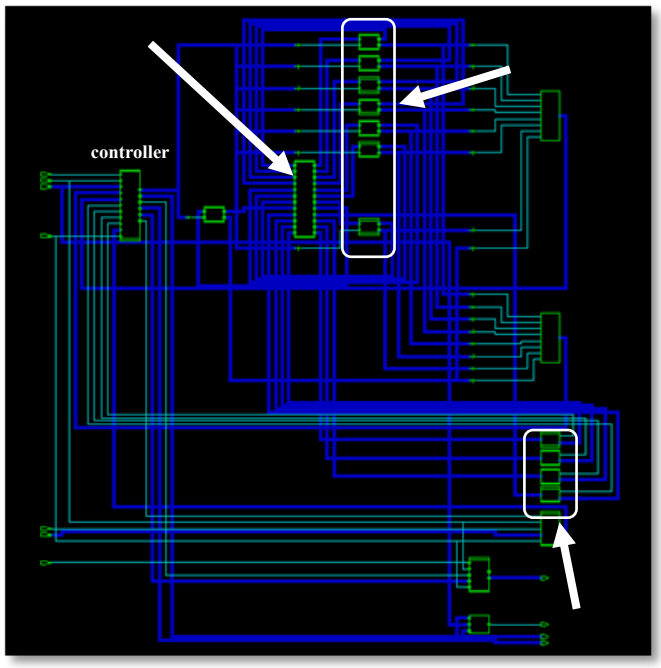
The controller takes care of control signals like the clk, rst, strt, read, and write, then they're corresponding output ports status_1, status_2 and the outbuffer_rd and outbuffer_wr. The ldpc_encoder then takes the status signals as long as the input datain signal and then it outputs the signal directly to the outbuffer. The outbuffer module then holds the output data from the ldpc_encoder module and buffs the data stream. The outbuffer will only output the signal only if the outbuffer_wr signal is high.

Figure 6(a) shows the Main_LDPC_Decoder module and its pinouts. Pinouts are then explained in detail at Table 4. Likewise, consider first the internal architecture of the main decoder module, shown in Figure 7(b). Recall that the operation of the decoder makes use of the Tanner Graph in [7], and thus hardware implementation would also look more or less the same. A tanner graph in turn consists also of the variable nodes and the check nodes, here in the standard architecture in the left, it is named VNU (variable node unit) and CNU (check node unit), respectively.

What makes the VCS schematic messy is because the modules are not arranged accordingly during compile, and the v-nodes and c-nodes are scattered all around. But basically, this shows the internal architecture of the decoder which consists of an input and an output buffer, a controller, v-nodes, c-nodes, and an interconnect module that takes care of the interconnection of the tanner graph.



(a)



(b)

Fig. 6. Main LDPC decoder: (a) Module (b) Internal

Table 4. Implemented Encoder Port Details

Port	Type	Description	Size (bit)
clk,rst,strt	Input	Master clock, reset, and enable of the decoder	1
new_data	Input	Read signal – Decoder only reads when start and this signal is 1	1
out_data	Input	Write signal – Decoded output is stored yet on the buffer and is only outputted when this signal is on.	1
maxiter	Output	Specifies maximum iteration	6
datain	Input	Input codeword	8
dataout	Output	Decoded codeword	4
iteration	Output	Outputs the real time iteration of the hard-decision decoding	6
statusout	Output	Outputs the current state of the decoder in real time.	3

Next is to look at the ldpc_encoder module. The goal here is to perform the matrix multiplication. We recall the proposed encoder H and G matrices as shown in Figure 4. The G matrix is derived from H matrix where G = identity matrix form of H.

Recall that in forming the code words, c, the input data, p, must be multiplied by the generator matrix G. The code words is then re-arranged so that the last four bits becomes the first 4 bits to keep track of the column permutations. So if we will take a look at the values 0001 and 0010 the corresponding code words which are the result of the matrix multiplication and the rearrangement as shown in Figure 4 are 10110001 and 01110010, respectively. This is particular data is shown in the simulation timing diagram in Figure 7 below.

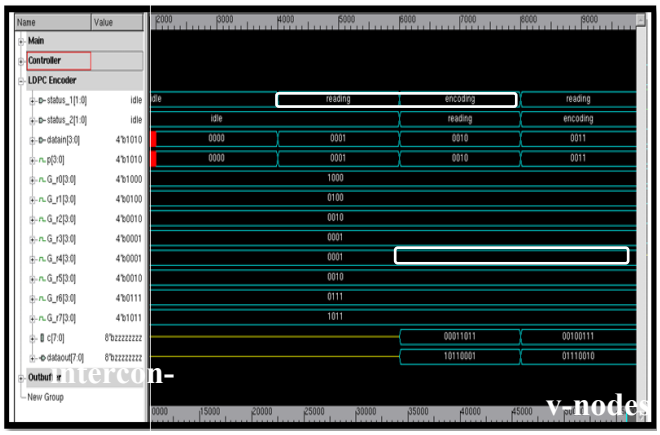


Fig. 7. LDPC_encoder module timing diagram

Looking at the timing diagram of the decoder where in three valid codewords of the architecture are being tested, which are 10010101 and 10110001 and 01110010. Note that the first codeword is the one that was used in the detailed explanation of the algorithm in section 1.7.2.5 while the two code words are those being used in the example with the encoder. Two cases are shown here, the case where both the code words are being flipped at bit positions 7, 0, and 1, respectively, and so it becomes 11010101, 10110000, and 01110000. The second case is simulated where the code words are received without an error. Figure 8 shows the main LDPC decoder timing diagram.

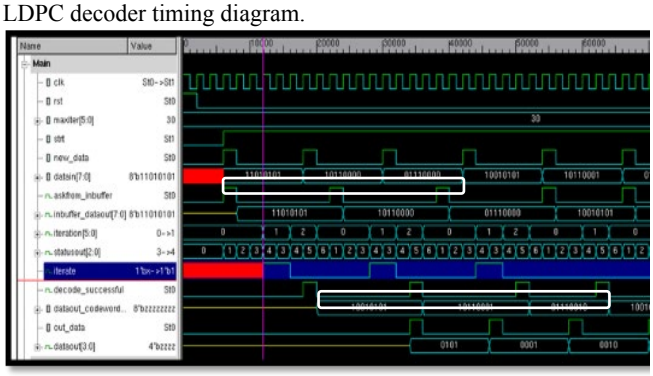


Fig. 9. Main LDPC decoder module timing diagram

It is shown from here the important signals that operate to perform the hard-decision decoding. In each of the erroneous code words, the system iterates up to the second iteration while the cases where the code words are not corrupted, it only goes up to the first checking of the parity check matrix. Likewise, the system needs buffers both at the input and in the output since the nature of the operation is not linear in time, it depends on the number of iteration a frame is decoding. But if the reader will look at the timing diagram in a bird's eye view, it can be seen that the decoder successfully corrected the corrupted frames.

Synthesis of the Verilog code, along with running the corresponding IC Compiler routines necessary to generate the prototype, show that the total area of the designed LDPC decoder is 6288.762798 um^2 which is small enough for a typical LDPC decoder area size using 130nm Logic CMOS Technology. The total power of the designed chip is 138.6393 uW which comprised of the dynamic power which is 112.9294 uW and the cell leakage power which is 25.7009 uW. Since the dynamic power is small, once the device operates, it will only consume less power and it will not heat up easily. And also since the Cell Leakage power is small compared to the dynamic power, it indicates a good design because the power loss is smaller than the dynamic power. The slack time for the input, output and combinational path LDPC Decoder is positive which indicates that the design is good in terms of timing because the data arrives in the input path, output

path and combinational path before the clock changes from logic high to logic low or from logic low to logic high.

4. Conclusion

The designed Low-Density Parity Check Error Correcting Codes was analyzed and has been successfully implemented in integrated circuit. The fundamentals of LDPC encoder and LDPC decoder blocks and special were also discussed in detail in different chapters of this paper. LDPC encoder and LDPC decoder modules were successfully implemented in the integrated circuit with 50 MHz clock using 130nm logic CMOS Technology. Simulated results were presented and all simulations were based on ideal and actual behaviors.

Furthermore, the actual simulation show that the LDPC Error Correcting Code was working and was effective since the digital data that has been assumed to have errors after being encoded in LDPC encoder was corrected after passing through the LDPC decoder. The designed LDPC Decoder can correct at least 1 error on every 8 bit data stream which complies our target of 42 error in 1K bytes.

Acknowledgement

A special thanks and sincerest gratitude to the following research funding institution: USAID –STRIDE and DOST – PCIEERD for providing the licensed tools used for this work.

References

[1] M. Mansour and N. R. Shanbhag (2003, High-throughput LDPC decoders. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 11:976–996.

[2] P. Radosavljevic, A. de Baynast, and J.R. Cavallaro (2005), Optimized Message Passing Schedules for LDPC Decoding, *Proceedings of the conference IEEE 39th on Signals, Systems and Computers*, pages 591–595.

[3] IEEE 802.11 Wireless LANs WWiSE Proposal: High Throughput extension to the 802.11 Standard. IEEE 11-04-0886-00-000n.

[4] L. W. A. Blad and O. Gustafsson, “An early decision decoding algorithm for LDPC codes using dynamic thresholds,” in Proc. Eur. Conf. Circuit Theory and Design , Aug. 2005, pp. III/285–III/288.

[5] M. M. Mansour and N. R. Shanbhag, “A 640-Mb/s 2048-bit programmable LDPC decoder chip,” IEEE J. Solid-State Circuits , vol. 41, no. 3, pp. 684–698, Mar. 2006.

[6] M. Karkooti and J. R. Cavallaro. Semi-parallel reconfigurable architectures for real-time LDPC decoding. In IEEE International Conference on Information Technology: Coding and Computing ITCC 2004, April 2004.

[7] M. M. Mansour and N. R. Shanbhag (2003), “High-throughput LDPC decoders,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 11, no. 6, pp. 976–996, Dec. 2003.

[8] R. G. Gallager, Low-Density Parity-Check Codes, Cambridge, MA: *MIT Press*, 1963.

[9] J. Banuchandar and D. Uthirapathi, "Single photon transistor," International Conference on Information Communication and Embedded Systems (ICICES2014), Chennai, 2014, pp. 1-5. doi: 10.1109/ICICES.2014.7034132