32nd NATIONAL RADIO SCIENCE CONFERENCE
(NRSC 2015), March 24-26, 2015,
October University for Modern Sciences and Arts, Egypt

Flexible LDPC decoder architecture for (3-6) regular codes

Ahmed M. Sadek[1], Aziza I. Hussein[2]
[1]Faculty of Computers & Information, Fayoum University, Fayoum, Egypt, ams13@fayoum.edu.eg
[2]College of Engineering, Minia University, Minia, Egypt, aziza_hu@yahoo.com

ABSTRACT

Low density parity check (LDPC) codes are a class of block codes with a very high error correcting performance. That performance made them suitable for many modern applications such as Digital Satellite Broadcasting system (DVB-S2), Wireless Local Area Network (IEEE 802.11n) and Metropolitan Area Network (802.16e). The decoding process of these codes is based on an iterative algorithm that requires many computational cycles. The implementation of a high performance flexible decoder that can support multiple codes is still an area of research. This paper presents a modified implementation technique of the normal sum-product decoding algorithm. That modification greatly enhances the performance of the decoding process to achieve high throughput. Furthermore, a flexible, partially parallel architecture that is tailored especially to support that modified implementation is given. The proposed LDPC decoder architecture is simulated on Xilinx ISE Simulator and implemented using VHDL code.

Keywords: *FEC, block codes, LDPC, iterative decoding, sum-product algorithm*

I. INTRODUCTION

Low density parity check codes are first introduced in the 1962 PhD thesis of Gallager [1] and after 35 years, they have been rediscovered by MacKay and Neal [2]. Their remarkable error correcting performance has made them a basis of many modern standards, such as satellite transmission standard DVB-S2, WIMAX standard IEEE 802.16e, and WIFI standard IEEE 802.11n[3],[4],[5].

LDPC codes can be represented by a sparse parity check matrix or using a bipartite graph called Tanner graph[2].

A Tanner graph has two sets of nodes: the first set of nodes is called "variable nodes" and they represent the N-bits of a codeword, the second set of nodes is called "check nodes' and they represent the parity check equations. An edge is drawn between *i*th variable node and *j*th check node if the *i*th code bit is included in the *j*th parity check equation.

LDPC are most commonly decoded iteratively with the message-passing algorithm, also known as sum-product algorithm or belief propagation algorithm, by calculating and exchanging of messages back and forth along the edges of the Tanner graph.

Many implementation methods have been introduced to design efficient, flexible and high performance decoders. Fully parallel decoders, where every check and variable node is implemented in hardware and is routed to each other as described by the Tanner graph, resulted in high performance but with excessive implementation costs [6], [7]. Serial decoders are too slow for almost all applications [8]. Partial parallel decoders, where multiple check nodes and variable nodes processors and storage memory, are a tradeoff between performance and cost [8], [9], [10], [11],[12].

Some decoding architectures rely on the direct implementation of the sum product algorithm and designing the arrangement of processors, memory, interconnection, and permutation networks [8], [9].

Optimization of the sum product algorithm was a subject of investigation and some alternative methods such as Min-Sum algorithm were introduced. It can reduce the hardware complexity of the sum product algorithm at the cost of acceptable performance degradation where the complex computations at the check nodes are approximated with simple comparison and summation operations [13], [14], [15].

This paper introduces a modified implementation of the sum product algorithm over a partially parallel architecture that is flexible to support multiple (3, 6) regular LDPC codes. The proposed architecture uses a combination of unicast and multicast communication between check and variable nodes, and achieves higher bitrates than available implementations.

ISBN 978-1-4799-7723-9/15/$31.00 © 2015 IEEE

The rest of this paper is organized as follows: Section II reviews the sum product algorithm for LDPC decoding. Section III introduces the flexible architecture that supports the modified algorithm implementation. Section IV discusses the simulation and implementation results.

II. LDPC DECODING ALGORITHM

As their name suggests, LDPC codes are block codes with parity-check matrices that contain only a very small number of non-zero entries. The main difference between LDPC and ordinary block codes is how they are decoded. LDPC codes are decoded iteratively using a graphical representation of their parity-check matrix, called a Tanner graph.

An LDPC code parity-check matrix is called (w_c ,w_r)-regular if each code bit is contained in a fixed number ,w_c, of parity check equation and each parity-check equation contains a fixed number, w_r , of code bits [16].

Equation (1) shows a regular parity-check matrix with w_c = 2 and w_r = 3, and Fig. 1 shows its Tanner graph representation.

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \quad (1)$$

In general, LDPC decoding algorithms are called message passing algorithms as their operation can be described by the passing of messages along the edges of a Tanner graph. Algorithms are named depending on the type of message that is passed over the edges or on the type of calculations performed at the nodes [16].

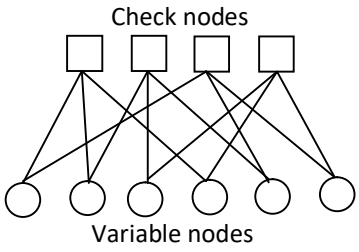

Fig. 1: Tanner graph representation

In bit flipping algorithm, binary bits are passed over the edges, while in Belief propagation algorithm, the messages are probabilities that represent a level of belief about the value of each codeword bit. It is often to represent probabilities as log likelihood ratio "LLR", and in that case, belief propagation is called sum-product algorithm as the operations at the nodes become sum and product operations.

In sum product algorithm, the message from check node j to variable node i, $E_{i,j}$, is the LLR of the probability that bit i causes parity check equation j to be satisfied.

$$E_{j,i} = 2\tanh^{-1}\left(\prod_{\alpha \in V[j], \alpha \neq i} \tanh\left(\frac{M_{j,\alpha}}{2}\right)\right) \quad (2)$$

Where $V[j]$ is the set of variable nodes connected to check node j, and $M_{j,\alpha}$ are their LLR values [14].

Each variable node has an initial LLR value and receives LLRs from each connected check node. The total LLR of each bit is the sum of these LLRs:

$$L_i = r_i + \sum_{\alpha \in C[j]} \left(E_{\alpha,i}\right) \quad (3)$$

ISBN 978-1-4799-7723-9/15/$31.00 © 2015 IEEE

Where *C[j]* is the set of check nodes connected to variable node *i*.
The messages sent from variable nodes to check nodes are the full LLR without the component $E_{j,I}$ which was just received from check node *j*, and hence the message sent from each variable node to all connected check nodes is:

$$M_{j,i} = \sum_{\alpha \in C[j], \alpha \neq j} E_{\alpha,j} + r_i \tag{4}$$

In the ordinary decoder implementation, each check node computes equation (2) and generates multiple messages that are sent as unicasts to multiple variable nodes, and every variable node calculates equation (4) and also generates multiple unicast messages to be sent to all connected check nodes. Using unicasts is the most common implementation method as in [6] – [11].
Another implementation was introduced in [12] and suggested the following: a modified variable (bit) node processor named (MVNP) that delivers one output message calculated as defined by equation (3), and in the same way, a modified check node processor named (MCNP) that generates a single output message calculated according to a modified version of equation (2). That algorithm was named *Low Traffic Belief Propagation (LTBP).* The goal of the processors is to reduce the number of messages to be exchanged between variable node processors and check node processors. That architecture needs to regenerate the original messages (from previous iteration), $E_{j,i}$ and $M_{j,i}$ locally at each node processor

III. THE PROPOSED FLEXIBLE LDPC DECODER

A. Proposed Algorithm Modification

In this work, we suggest an implementation that depends on a combination of unicast and multicast strategies. The reason behind the idea is that implementations that use unicast messages, keeps node processors and the operations they have to perform simple. The other implementation which relied on multicast messages, has greatly reduced the number of messages to be exchanged, but introduced more complexity at the node processors and resulted in more processing cycles.
This work suggests the following:- variable node processors, which calculate equation (4), use multicasts as the operation to be performed is a simple addition. Using multicast will not pose any increase in complexity but will reduce the number of messages greatly. On the other hand, check node processors, which calculate equation (2), will work ordinarily with unicasts as their operation is already complex and cannot afford extra processing cycles.

B. Proposed Architecture

To implement the suggested algorithm, we proposed an architecture that supports multicasting at the variable node processor and unicasting at the check node processor.
Fig. 2 shows an overall block diagram of the decoder architecture with one variable node *VN* and one check node *CN*.

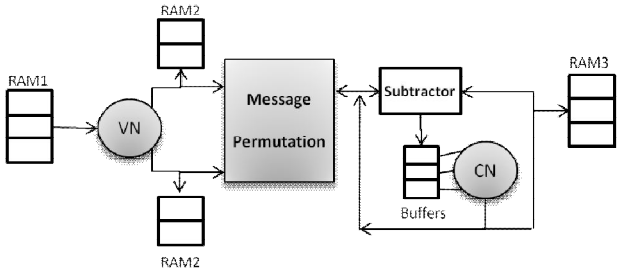

Fig. 2: Architecture of LDPC decoder

Each variable node processor is connected to a set of RAM units. One unit holds the initial LLRs to be processed by this processor (RAM1). Other units hold the values of messages communicated over the connected edges to this processor; two RAM2 units for two connected edges.

ISBN 978-1-4799-7723-9/15/$31.00 © 2015 IEEE

On the other hand, each check node processor also has a set of RAM units. One unit holds the previous iteration values of the edges connected to this processor (RAM3). Another unit works as a buffer that holds the difference between the multicasted message from the VN processor and the previous iteration calculated value. The proposed decoder does not check for a valid codeword, because the time to perform a check would be as long as the check nodes half iteration. So, performing one check on every iteration would dramatically increase the processing time. Therefore, the decoder implements a fixed ten iterations mechanism.

1) *Variable node processor*: The variable node processor is composed of an adder with number of inputs equal to the number of edges connected to the VN processor plus one input for the initial LLR value of the currently processing bit (3 edges in this case). Many adder implementations are available, but it was found that the carry look-ahead adder is the best compromise between performance and complexity [8], so a tree adder was built with carry look-ahead adders. The processor loads three values from three connected RAM units through bidirectional ports. At the same time, the initial LLR value is loaded from the connected codeword RAM unit. The result of the addition is latched so that it can be written back to the edges RAM units through the bidirectional ports. Fig. 3 shows a block diagram of the variable node processor.

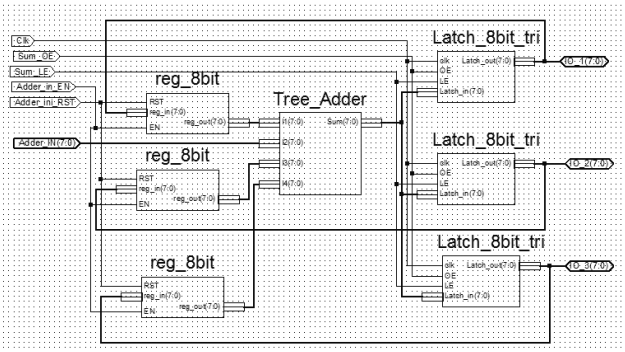

Fig. 3: block diagram of the VN processor

2) *Check node processor*: The check node processor is supposed to calculate the multiplication and the hyperbolic trigonometric function of equation (2). By introducing a new function $\varphi(x)$ where:

$$\varphi(x) = -\ln\left(\tanh\left|\frac{x}{2}\right|\right) \tag{5}$$

Equation (2) can be transformed to the following expression [8]:

$$E_{j,i} = \varphi\left(\sum_{\alpha\in v[j],\alpha\neq i}\varphi(M_{i,\alpha})\right) \tag{6}$$

D. Hayes[8] has investigated the various implementation methods of the nonlinear function $\varphi(x)$, and lookup table (LUT) was the best compromise between performance and resources requirements. So, a lookup table with 256 entries was built to implement the function φ (x).

Now, the heart of the check node processor is composed of an adder with a lookup table on each input and another lookup table on the output. The sign of the output LUT is corrected according to the signs of the inputs. By XOR'ing the MSB of the inputs and if the result is '1', then the result of the output LUT is negated. Fig. 4 shows a block diagram of a check node processor core.

ISBN 978-1-4799-7723-9/15/$31.00 © 2015 IEEE

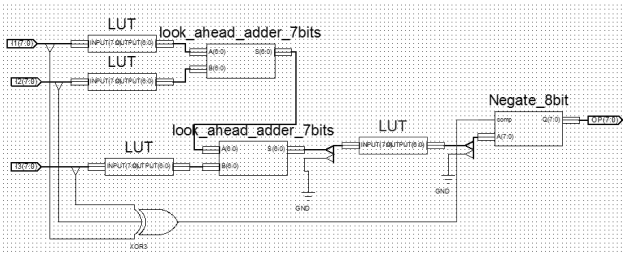

Fig. 4: CN processor core

The values of the edges to be processed by the CN processor are loaded from the RAM units through the permutation network to the minuend input of the subtractor. At the same time, the previous iteration values are loaded from CN's local RAM to the subtrahend input of the subtractor. The output of the subtractor is stored in the CN's local buffer. The CN processor works on the correct edges' values stored in its buffer and calculates equation (6). The result is latched so that it can be written to the local RAM unit, and to the edges RAM units through the permutation network. Fig. 5 shows a block diagram of CN processor.

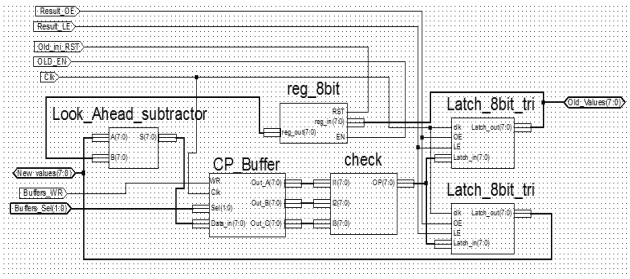

Fig. 5: Block diagram of a CN processor

3) Message permutation network: To support transfer of values from edges RAM units to and from CN processors, a permutation network is required. Many types of permutation networks are available and discussed in [17].

Benes network is found to be a good compromise between scalability and flexibility. It is a non blocking network and it is used to connect every CN processor to any RAM unit at VN nodes. The Benes network traditionally consists of $(2Log_2N-1)$ number of stages where each stage consists of (N/2) 2x2 elementary switches. For this design, each VN processor has multiple RAM units for edges values, 3 for (3, 6) regular codes. Therefore, multiplexers are used between the VN RAM units and the permutation network. Fig. 6 shows a 4x4 permutation network

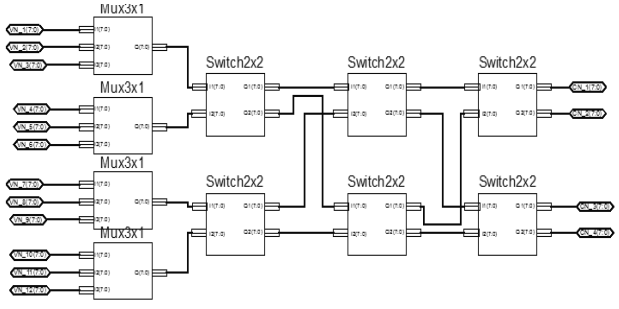

Fig. 6: 4x4 permutation network

The configuration of the permutation network switches is stored in a ROM unit. The control unit is responsible for invoking the permutation ROM to setup the permutation network to connect the CN processors to the appropriate RAM unit. The contents of the permutation ROM unit depend on the parity check matrix of the code. The decoder can support more than one code by loading the proper permutation configuration to ROM of the network.

ISBN 978-1-4799-7723-9/15/$31.00 © 2015 IEEE

IV. RESULTS

To verify the impact of the proposed decoding method on the performance, we simulated the decoding algorithm on a set of (3, 6) regular benchmark codes to compare with [12]. Table (1) shows some details of the selected codes which are available on MacKay's website [17].

Table (1): Benchmark code with number of edges ϵ

	Code	ϵ
1	(204,102)	612
2	(816,408)	2448
3	(1008,504)	3024
4	(8000,4000)	24000

In table (2), the number of clock cycles needed to communicate messages during one complete iteration (check node iteration and variable node iteration) is estimated for 32 processing elements. Column two displays the theoretical number of cycles needed to process messages for one complete decoder iteration. It is simply the number of code edges divided by the number of processing elements and then multiplied by two for both half iterations.

Table (2): Clock cycles count for messages communications

code	Node Processing	Multicast [12]	Suggested Method	Increase
1	40	90	125	28%
2	154	312	416	25%
3	190	376	512	26.5%
4	1500	2878	4000	28%

Column three is the number of cycles required when using LTBP algorithm that employs multicast [12]. Column four is the number of cycles needed when the suggested method is used. Of course it was predicted that when using a combination of unicast and multicast, the number of clock cycles needed to send and receive messages would increase, up to 28% as seen from table (2). Although the suggested architecture uses more communication cycles, the node processors are simple enough to be make the processing cycles few, resulting in less overall cycles and hence increased throughput as will be shown further on.

To verify the throughput performance of the suggested decoder, two prototype decoders were implemented using Xilinx ISE 14.2 with a combination of VHDL and schematics tools. One of them consists of 4 VN processors and 4 CN processors, while the other consists of 8 VN and 8 CN processors. They were both targeted at Xilinx Virtex5 (XC5VLX20T) development board. Tables (3) and (4) show the device utilization summary report for the implementation with 4 CN/VN processing elements and with 8 CN/VN processing elements respectively.

Table (3): Device utilization of 4 CN/VN PEs

Logic Itilization	Used	Available
Number of slice registers	154	12480
Number of slice LUTs	252	12480
Number of Block RAM	6	26

ISBN 978-1-4799-7723-9/15/$31.00 © 2015 IEEE

Table (4): Device utilization of 8 CN/VN PEs

Logic Utilization	Used	Available
Number of slice registers	287	12480
Number of slice LUTs	469	12480
Number of Block RAM	14	26

As seen from the two tables, the resource usage increased almost linearly with the number of processing elements and that indicates very good scalability.
The contents of the ROM units were calculated offline for the test codes. The maximum clock frequency as determined by the synthesize tool was 257.26 MHz for the 8 CN/VP decoder.
The suggested architecture was able to achieve higher throughput than the architecture presented in [12]. Table 5 presents the throughput achieved for a code of length (1944-972) by the suggested decoder in this work and the one implemented in [12]. When using 8 CN/VN, the suggested decoder could deliver 199 Mbps per iteration compared to 129 Mbps achieved in [12]. If we assume linear performance scalability with the number of processing elements, which is a very reasonable assumption for regular codes, then the suggested decoder would deliver about 796 Mbps with 32 CN/VN processing elements compared to 515 Mbps [12].

Table 5: throughput results

Code	No. of Processing Nodes	Throughput Per Iteration (Mbps)		Increase (%)
		Work of [12]	*This work*	
(1944-972)	8 CN/VN	129	199	54%
	32 CN/VN	515	796	55%

The previous results show that the suggested decoder can deliver up to 55% increase in throughput performance compared to the architecture in [12].

V. CONCLUSION

As modern communications technologies advance, the need for high bandwidth communication increases. Forward error correction is one of the primary processes of any communication system and its performance has a great impact over the whole system. LDPC codes are becoming more and more attractive for modern communication technologies as it can deliver the demands of high bitrates. Decoding of LDPC code was investigated in many directions including various algorithm implementations and variety of processing architectures.

This paper introduces a modified implementation of the usual sum-product decoding algorithm with the goal of minimizing the communication overhead between the processing elements of the decoder, and at the same time, keeping the processing tasks as simple as possible. The paper also introduces a partially parallel decoder architecture described using VHDL to implement the suggested algorithm. The simulation and implementation results showed that the suggested algorithm was able to reduce the number of messages transmitted between the processing elements. It also showed that the decoder was able to deliver high throughput with linear scalability to support even higher bitrates with more processing elements.

References

[1] R. Gallager, “Low-density parity-check codes,” Information Theory, IRE Transactions on, vol. 8, no. 1, pp. 21 –28, 1962
[2] D. MacKay, “Good error-correcting codes based on very sparse matrices,” in Information Theory. 1997. Proceedings., 1997 IEEE Interna-tional Symposium on, 1997

ISBN 978-1-4799-7723-9/15/$31.00 © 2015 IEEE

[3] J. Lorincz and D. Begusic, "Physical layer analysis of emerging IEEE 802.11n WLAN standard," in Advanced Communication Technology 2006. ICACT 2006. The 8th International Conference, vol. 1, 2006, pp. 6 pp. –194.
[4] M. Khan and S. Ghauri, "The WiMAX 802.16e physical layer model,"in Wireless, Mobile and Multimedia Networks, 2008. IET International Conference on, 2008, pp. 117 –120.
[5] A. Morello and V. Mignone, "DVB-S2: The second generation standard for satellite broad-band services," Proceedings of the IEEE, vol. 94, no. 1, pp. 210 –227, 2006
[6] A. J. Blanksby and C. J. Howland,"A 690-mW 1-Gb/s 1024-b, rate ½ low-density parity-check code decoder " IEEE J. Solid-State Circuits, vol. 37, no. 3, pp. 404–412, Mar. 2002
[7] V. A. Chandrasetty, S. M.Aziz, "An area efficient LDPC decoder using a reduced complexity min-sum algorithm ",Integration, the VLSI Journal, Volume 45 Issue 2, pp. 141-148, March, 2012
[8] D. Hayes, "FPGA implementation of a Flexible LDPC decoder",PSc thesis, University of Newcastle, Australia 2008
[9] T. Zhang; K. K. Parhi, An FPGA implementation of (3,6)-regular low-density parity-check code decoder ,Eurasip Journal on Applied Signal Processing. 2003(6):530-542.
[10] Jong-Yeol and H.-J. Ryu, \A 1-gb/s flexible ldpc decoder supporting multiple code rates and block lengths," Consumer Electronics, IEEE Transactions on, vol. 54, pp. 417{424, May 2008.
[11] C. Condo and G. Masera, A Flexible LDPC code decoder with a Network on Chip as underlying interconnect architecture. In Proceedings of CoRR. 2011.
[12] Guido Masera, Federico Quaglio, and Fabrizio Vacca, "Implementation of a Flexible LDPC Decoder" , IEEE transactions on circuits and systems—ii: express briefs, vol. 54, no. 6, june 2007
[13] J. Chen, A. Dholakia, E. Eleftheriou, M.P.C. Fossorier, and X. Hu, "Reduced-complexity decoding of LDPC codes," IEEE Trans. Comms., vol. 53, no. 8, pp. 1288- 1299, Aug. 2005.
[14] J. Zhao, F. Zarkeshvari, and A.H. Banihashemi, "On implementation of min-sum algorithm and its modifications for decoding low-density Parity-check (LDPC) codes," IEEE Trans. Comms., vol. 53, no. 4, pp. 549-554.
[15] Mohammad Rakibul Islam, Dewan Siam Shafiullah, Muhammad Mostafa Amir Faisal, Imran Rahman, "Optimized Min-Sum Decoding Algorithm for Low Density Parity Check Codes",(IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 2, No. 12, 2011
[16] S.J. Johnson, "Introducing Low-density Parity-check codes", Published Internal Technical Report , Department of Electrical and Computer Engineering, University of Newcastle, Australia
[17] Mark Woh, "ARCHITECTURE AND ANALYSIS FOR NEXT GENERATION MOBILE SIGNAL PROCESSING", PhD thesis, dep. of Electrical Engineering, The University of Michigan, 2011

ISBN 978-1-4799-7723-9/15/$31.00 © 2015 IEEE