International Journal of Computing and Applications | ISSN: 0973-5704
Volume 13, Number 1, (January-June 2018), pp 171-176

IMPLEMENTATION OF MEMORY EFFICIENT SEMI-PARALLEL LDPC DECODER USING ITERATIVE MESSAGE PASSING ALGORITHM

Shruti Bamnote[1], Sandeep Kakde[2] and Atish Khobragade[3]

[1-2]Y C College of Engineering, Nagpur, India. Email: [1]shrutibamnote0101@@gmail.com, [2]sandip.kakde@gmail.com
[3]Rajiv Gandhi College of Engg., Nagpur, India. Email: atish_khobragade@rediffmail.com

Abstract: LDPC decoding for wireless application is an intensive operation which is done in a computational manner .It requires millions of messages to be passed to achieve acceptable performance. This paper gives an overview regarding implementation of a LDPC decoder using Iterative Message Passing Algorithm, implemented in HDL Verilog and synthesized in CMOS 0.180 m technology. The implementation utilizes block-level Semi parallelism in order to achieve less memory, high throughput and error-correcting performance, at the cost of large silicon area. Newer silicon technologies allow for higher frequencies; the architectural implications of this are explored. The Variable Node's are each connected to two RAM's, one holding the LLR values and one holding the sum of the incoming messages from the Check Node's. Each Variable Node holds adders to combine incoming messages and create the outgoing message, and registers to hold the I/O data and the messages. Here Single-port RAM's were used, requiring 2 cycles for each message transmission, but using less silicon and allowing more flexibility in terms of folding logic or multi-cycle paths, compared to dual-port RAM's.

Keywords: LDPC codes; Memory Efficient; Belief Propogation algorithm; Check Node, Variable Node, iterative decoding.

1. INTRODUCTION

Low-Density Parity Check (LDPC) coding is a form of error coding which was introduced by R. Gallager, that can achieve performance close to the Shannon limit, exceeding the performance of Turbo codes [3]. The coding scheme was introduced in the early 1960's and it has gained favor recently because of its excellent performance and lack of patent rights. Many recent standards include optional or mandatory LDPC coding methods, and among these is the second generation Digital Video Broadcasting standard for satellite applications [3]. This application is creative by low latency requirements, so the standard employs tough coding over codeword's 64,800 bits long. Although the standard was designed for low complexity in hardware, the length of the codeword's makes this the most computationally intensive of LDPC codes described in current standards. There are different message Passing Algorithms are available for decoding the messages. Belief Propagation is one of the important algorithms used for memory efficient implementation of LDPC Decoder.

2. BELIEF PROPOGATION ALGORITHM

Belief-propagation (BP) is the most commonly and widely used algorithm for decoding LDPC codeword's, and is the one used in this project. This algorithm is formed around the idea of iterative message passing along all the edges of a Tanner graph, as illustrated in Figure 1.

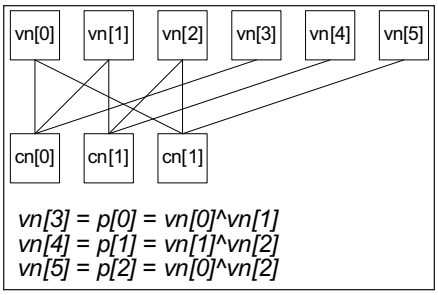

Figure 1: Simple Tanner Graph

In the BP algorithm, parity bits and likelihood values are conceded as messages from all variable

nodes to all connected check nodes. The initial likelihood values are derived from the channel quality and the Euclidian distance between the received symbols and the nearest constellation points. Messages are conceded as log-likelihood ratios (LLR's), since representing probability ratios in this form allows for simpler arithmetic. The parity equations all estimate to zero, so the check nodes determine the expected parity bits at each connected VN, based on the signs of each of the other connected nodes. The expected parity bits are passed to each connected variable node, along with likelihood values. The VN's in turn use these values to update the VN parity bits and likelihood values, and the cycle begins again. In this way, the messages tend to strengthen bits which are in agreement with parity equations and correct bits which are in error.

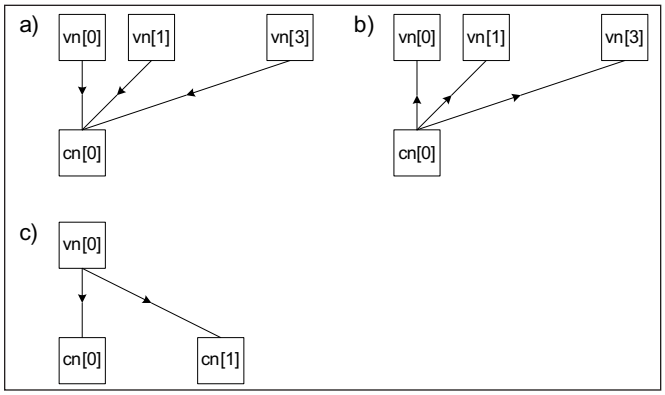

Figure 2: Message Passing. In (a), parity bits and likelihood ratios are passed from the variable nodes to the check nodes. In (b), the received messages are pooled to create messages for the variable nodes, containing most likely parity values and the likelihood of that value being correct, based on the messages received from the other VN's. (c) shows how the variable nodes (in this case, VN[0]) uses the received data to send updated parity values and likelihood ratios in the following iteration.

The ideal BP implementation combines the messages from the variable nodes to create optimal reliability messages. For a parallel implementation, each computation node would have to implement this function. It was shown in [4] that the Min-Sum algorithm can provide alike performance to the ideal implementation with far lower computational complexity. Rather than combining the likelihood values optimally in the check nodes for each edge, the Min-Sum algorithm uses the least likely received message as an approximation of the result for all edges but one. The message created for the edge which transferred the smallest LLR must be calculated separately, since the received result on any edge may not be used in the calculation of the message that will flow back along that same edge. For the edge that transferred the smallest LLR, the next-to-smallest LLR is used as an approximation. The Min-Sum approach allows for simpler arithmetic and reduces storage requirements, since only two LLR values are stored.

It was found in [8] that a non-linear message representation improves the compression of messages while retaining the same performance as a linear representation. The simulations carried out for DVB-S2's LDPC specification could not reproduce this result, with BER results for log-scale messages uniformly inferior to the linear version. Simulations were carried out using logarithmic tables of various lengths, using bases between 1.25 and 2.0, and maximum values the same or twice the magnitude of the linear tables. Another coding scheme was simulated, with the goal of improving the granularity of the messages by removing a redundant signed zero representation. In this encoding scheme, the coded inputs were offset by positive or negative 1, so that +0 represented $\frac{1}{2^b} \times \text{clip threshold}$ and −0 represented a negative number of the same magnitude. This offset-linear scheme retained symmetry around zero and improved the granularity of the messages. Unfortunately, while offset-linear encoding did have better performance under certain EbN0 values, the performance degraded relative to the linear implementation as EbN0 increased. It is intended to deliver a quasi-noise free stream, so the codeword input to the BCH decoder should have an error rate of at most 10^{-4}. By that criterion, offset-linear encoding had poorer performance under all usable conditions, compared to direct linear encoding.

It was found in [4] that clipping the received signal at ±1.25 can improve performance under some circumstances, but can also introduce a noise floor. No effort was made to find the optimal clipping value, as

clipping is implemented outside of the LDPC decoder, in the quantization/LLR function.

3. VLSI ARCHITECTURE

LDPC decoder uses codewords of up to 64,800 bits in length, and with more than 280,000 edges in the Tanner graph. Passing this lots of messages in parallel would be impractical, and would provide far higher performance than what is requisite. For this project, a performance level of 135Mbps was targeted, and it was assumed that the synthesized result would run at 200MHz or faster. Single-port RAM's were used, requiring 2 cycles for each message transmission, but using less silicon and allowing more flexibility in terms of folding logic or multi-cycle paths, compared to dual-port RAM's. Based on these assumptions, the minimum degree of parallelism was calculated:

$$\frac{30\,\frac{\text{iteration}}{\text{codeword}} \times 2 \times 233\,\frac{\text{kmessage}}{\text{iteration}} \times 135\,\text{Mbit/s}}{200\,\frac{\text{Mcycle}}{\text{s}} \times 0.5\,\frac{\text{message}}{\text{cycle}} \times 64\,\frac{\text{kbit}}{\text{codeword}}} \approx 300$$

The standard is written such that bits are arranged in groups of 360 bits, so this is the degree of parallelism that was chosen. Increasing the frequency or using dual-port RAM's could allow for 180*x*, 90*x* or 45*x* parallelism for reduced area, with some increase in the complexity of the control logic.

Central to the design is a shuffle network, which shifts 360 input messages to 360 outputs through a 3-stage pipeline. The shuffle network has a multiplexer at each input, to pick from the check node and the variable node messages. The VN's are each connected to two RAM's, one holding the LLR values and one holding the sum of the incoming messages from the CN's. Each VN holds adders to combine incoming messages and create the outgoing message, and registers to hold the I/O data and the messages.

The CN's are each connected to a single, wide RAM holding the two smallest of the incoming LLR's during the current iteration, the signs of all incoming messages, the locations of the minimum values, and the parity result of all the incoming messages. The message to the VN's are produced by reading one of the two min-LLR values, along with the expected sign value for a particular edge.

The control module reads a ROM to fetch a shift value for the shuffle network and a write/read address for the check nodes. In the forward direction, for message passing from VN's to CN's, these values are used directly. In the reverse direction, the shift value is negated to allow messages to flow along the same edge in both directions. The control module also needs to compensate for delays in the shuffle network, VN's and CN's. To define an LDPC code which could be efficiently implemented in hardware, but a regular edge pattern would have provided poor performance. To create a pseudo-irregular edge pattern, a scrambling factor was introduced, spreading edges throughout the codeword. This scrambling factor, q, is proportional to the number of check nodes, and is defined as $q = \frac{n-k}{360}$. In the standard, the edges are defined in groups of 360 variable nodes, with each edge starting at a $vn[m]$, where m is the relative position within the 360-node group, and ending in check node $x + (m \times q) \bmod (n-k)$, where x is a base parity location, defined in appendices of the standard. For instance, for the 1/4 code rate, the code parameters (n, k) are (64800, 16200), and $q = \frac{64800 - 16200}{360} = 135$.

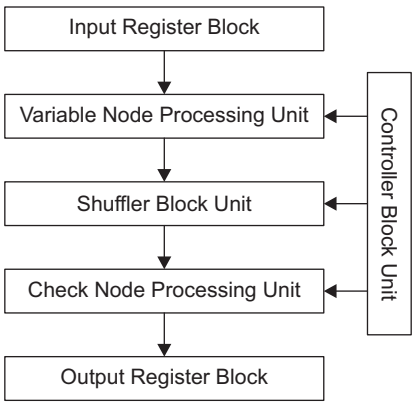

Figure 3: Top-Level Block Diagram

To allow parallel access to all 360 variable nodes in a group, the VN RAM's are arranged as shown below:

The first edge defined in the standard for $vn[0]$ is 23606. The corresponding edge for $vn[1]$ is 23606 + (1 × 135) mod ($n - k$) = 23741. For this example, to allow simultaneous message passing from: {$vn[0]$, $vn[1]$

... *vn*[359]}, the following CN memory locations must be made available:

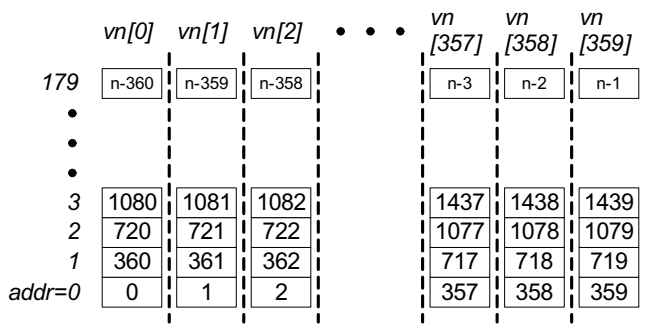

Figure 4: Variable Node Memory Organization

{*cn*[23606], *cn*[23741] ... *cn*[(23606 + 135 * 359) mod($n - k$)]}

The need to access $\{x, x+q, x+2q ...\}$ suggests the following memory organization in the Check Nodes (CN's):

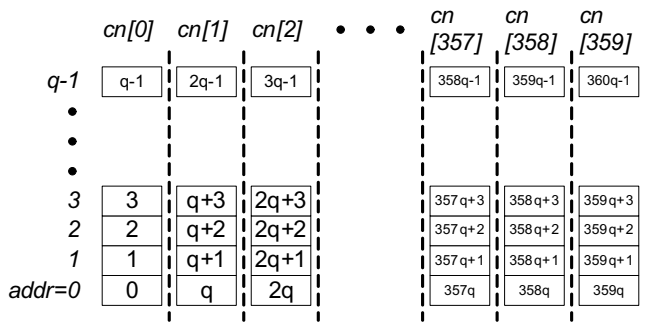

Figure 5: Check Node Memory Organization

This pattern is used through the parity encoding of all the data bits in the codeword, but in the end the parity bits are combined in a different manner. The first parity bit, *vn*[k] is encoded into the check equation cn[1], then *vn*[k + 1] is encoded into cn[2], and so on until the end of the codeword. Storing all messages in the VN's is requisite for the optimal result by summing the received messages, but excluding the message received on the edge being calculated. Storing all the upstream messages would require a large RAM to store up to 30 messages at each location in each VN. To avoid the use of these large RAM's, the CN's store the messages sent in the previous upstream message, and subtract that value from the received downstream message. When neither upstream nor downstream message saturates to the full LLR value, this approach provides identical results to the ideal approach. When one of the messages saturates, some distortion occurs. In particular, for the case when both upstream and downstream messages are saturated, the resulting offset-message received in the CN will be zero, even though the VN sent the maximal LLR. This problem is diminished by limiting the upstream message to one fewer bit than the downstream messages. This approach allows for the use of small RAM's in the VN's, but reduces the performance by reducing the number of effective message bits by one.

The use of layered decoding was considered for this project, but was not appropriate for the projected architecture. Layered decoding can reduce the number of required iterations by updating VN outgoing messages during each iteration and using this updated value to provide improved downstream messages to the CN's. This approach requires shifting the message-passing direction many times during a single iteration to update the VN's with the latest messages. It also implies that the messages are passed in row-order, (in terms of the parity matrix), so that all messages destined for a single check node are passed around the same time. Switching directions has a significant (at least 5 cycle) penalty in the implemented design, increasing the period of each iteration. For instance, if each check node in the Tanner graph were attached to 4 edges, reversing the direction after each complete check-node update would increase each iteration's period by more than 60%. Passing messages in parity matrix row-order, rather than column-order, as is currently implemented, would require a very different memory arrangement than the one proposed, and possibly a number of architectural changes.

4. HDL DESIGN RESULTS

LLR data is first loaded serially into the variable nodes through a chain of registers. The signal, llr_access, turns off message passing and connects the chain of registers, with the signal llr_din_we exchange the data in the RAM for the data in the register chain. For instance, if llr_addr were held to zero and the llr_din_we were driven high, the contents of the chain of registers would be written into address zero, and the contents of each of the RAM's at location zero would

be encumbered into the chain. In this manner, reads and writes from the module can be consummate at once. Data should be written in the format described in Figure 6.

After data has been written into the decoder, the message passing algorithm may begin. This is controlled by the signal "start" which loads parameters, including the mode and the number of iterations, and starts the decoding process. The modes are as follows:

Table 1
Allocation of Memory

Mode number	*Description*	*n*	*k*
0	1/4 normal	64800	16200
1	1/3 normal	64800	21600
2	2/5 normal	64800	25920
3	1/2 normal	64800	32400
4	3/5 normal	64800	38880
5	2/3 normal	64800	43200
6	3/4 normal	64800	48600
7	4/5 normal	64800	51840
8	5/6 normal	64800	54000
9	8/9 normal	64800	57600
10	9/10 normal	64800	58320
11	1/5 short	16200	3240
12	1/3 short	16200	5400
13	2/5 short	16200	6480
14	4/9 short	16200	7200
15	3/5 short	16200	9720
16	2/3 short	16200	10800
17	11/15 short	16200	11880
18	7/9 short	16200	12600
19	37/45 short	16200	13320
20	8/9 short	16200	14400

The system controller manages the swap over of messages between the variable and check nodes by controlling the VN and CN write and read addresses and the shift value of the shuffler.

The controller reads the edge destinations from the ROM and creates addr_cn and addr_vn (the check and variable node addresses), along with the write enable signals based on those edges. The basic state machine operation is as follows:

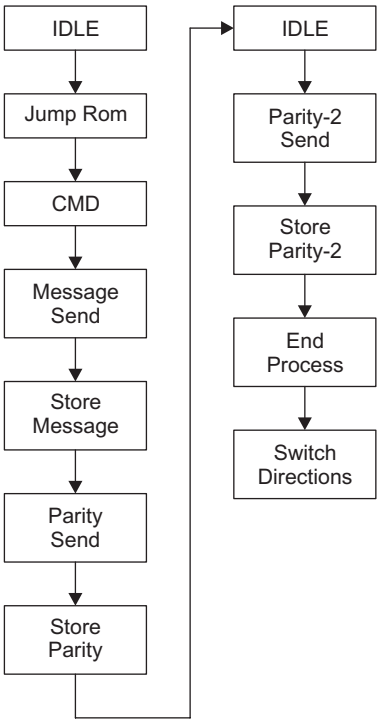

Figure 6: State machine in module iocontrol

The state machine ignores delays in the shuffler, VN's and CN's, and relies on delay registers somewhere else in the module "iocontrol" to bring into line the control signals properly. The shuffler muxes the CN and VN messages into a pipeline of shifters, allowing for any rotation in three cycles. The signal "first_half" controls the multiplexer, using "vn_concat" in the first half of each iteration, and "cn_concat" in the second half.

Table 2
Comparison with Existing Work

LDPC Decoder *EDA TOOL:XILINX ISE 14.1* *FPGA FAMILY: Virtex 5*	
Memory Implementation	
Module Name	*Block RAM*
Ref. [5]	36
Proposed Schedule	2

5. CONCLUSION

In this paper, efficient memory utilization was shown by using semi parallel architecture. Reorganizing the memories in the current architecture is a difficult task, since that implies accessing and writing data at

different rates in the VN's and CN's. Another solution might be to combine the VN's and CN's, and to store data for the two functions in a common RAM. Code rates which require large storage in the CN's require less storage in the VN's, and vice-versa, so storing the data together makes better use of RAM. Using the current logic, such a change would require 3-port RAM's with two simultaneous writes to different addresses.

References

[1] Engling Yeo, Payam Pakzad, Borivoje Nikolić, and Venkat Anantharam. "High Throughput Low-Density Parity-Check Decoder Architectures"2001 IEEE.

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

[3] D.J.C. MacKay and R.M. Neal. "Near Shannon limit performance of low density parity check codes." *Electronics Letters*, Volume 32, pages 1645-1646, Aug 1996.

[4] Karkooti, Marjan, and Joseph R. Cavallaro. "Semi-parallel reconfigurable architectures for real-time LDPC decoding." In Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on, Vol. 1, pp. 579-585. IEEE, 2004.

[5] Chen, Yanni, and Dale Hocevar. "A FPGA and ASIC implementation of rate 1/2, 8088-b irregular low density parity check decoder." Global Telecommunications Conference, 2003. GLOBECOM'03. IEEE. Vol. 1. IEEE, 2003.

[6] Shimizu, K., Ishikawa, T., Togawa, N., Ikenaga, T., & Goto, S. (2006, May). A parallel LSI architecture for LDPC decoder improving message-passing schedule. In 2006 IEEE International Symposium on Circuits and Systems (pp. 4-pp). IEEE.

[7] A.H. Banihashemi, J. Zhao and F. Zarkeshvari, "On implementation of min-sum algorithm and its modifications for decoding LDPC codes," IEEE Trans. Comm., Nov 2004.

[8] J. K. S. Lee, J. Thorpe, J. Hawkins, "Memory-Efficient Decoding of LDPC Codes," IEEE Intl. Symposium on Information Theory, Sep 2005.

[9] Kakde, Sandeep, Atish Khobragade, and MD Ekbal Husain. "FPGA Implementation of Decoder Architectures for High Throughput Irregular LDPC Codes." Indian Journal of Science and Technology 9, No. 48 (2016).

[10] "High Throughput Low-Density Parity-Check Decoder Architectures", Engling Yeo, Payam Pakzad, Borivoje Nikolić, and Venkat Anantharam Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720-1770.

[11] Low-Power VLSI Decoder Architectures for LDPC Codes * Mohammad M. Mansour and Naresh R. Shanbhag iCIMS Research Center, ECE Dept. Coordinated Science Laboratory University of Illinois at Urbana-Champaign 1308 W. Main Street, Urbana, IL 61 801.

[12] Kakde, Sandeep, and Atish Khobragade. "VLSI Implementation of a Rate Decoder for Structural LDPC Channel Codes." Procedia Computer Science 79 (2016): 765-771.

[13] Layout-Driven Memory Synthesis for Embedded Systems-on-Chip Luca Benini, Member, IEEE, Luca Macchiarulo, Alberto Macii, and Massimo Poncino, Member, IEEE.

[14] Kakde, Sandeep, and Atish Khobragade. "HDL Implementation of an Efficient Partial Parallel LDPC Decoder Using Soft Bit Flip Algorithm."

[15] European Telecommunications Standards Institute. Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications; ETSI EN 302 307 V1.1.2 (2006-06).

[16] Kakde, Sandeep, and Atish Khobragade. "Performance Analysis of a High-Throughput LDPC Decoder Using Sum Product and Min Sum Algorithm." *Int. J. Com. Dig. Sys* 6, No. 2 (2017).