A Flexible Hardware Encoder for Systematic Low-Density Parity-Check Codes

Hemesh Yasotharan, Anthony Chan Carusone
University of Toronto
10 Kings College Road
Toronto, Ontario M5S 3G4
Canada

***Abstract*—This paper proposes a flexible low density parity check encoder. This encoder simplifies the calculations found in other flexible encoders by increasing memory usage, allowing for parallelization and faster encoding. The flexibility of this encoder allows it to be used in emerging multi code applications and standards. To evaluate the encoder, a Verilog description was developed and synthesized on an Altera Stratix platform for the IEEE 802.16e WiMAX standard. The implementation used 11,430 logic elements and operated at a maximum clock frequency of 60 MHz. The throughput ranged from 119 Mbps for rate-$\frac{1}{2}$ codes to 357 Mbps for rate-$\frac{5}{6}$ codes. A speedup of 2.5–6 times is demonstrated compared to the prior art.**

I. Introduction

Low Density Parity Check (LDPC) [1] codes have been shown to perform extremely close to the Shannon limit of channel capacity [2], [3]. The high performance, freedom from patent restrictions, and inherent parallelism of LDPC codes has resulted in the inclusion of these codes in various standards such as IEEE 802.16e [4] and CDMA [5]. In many of these applications, several codes are specified allowing codes of different rate and complexity to be chosen depending on the quality of the channel. A flexible encoder architecture must accommodate various code rates and block lengths. Ideally, the system should be able to change its encoding parameters based on user input. This would allow one encoder to be used for all of the codes in a given standard, or for a wide range of applications [4], [5].

Although many architectures have been reported for flexible high speed LDPC decoders [6], [7], [8], [9], there has been much less work on flexible LDPC encoders. A flexible architecture that uses recursion to calculate parity bits was proposed in [10]. The design methodology is flexible in regards to code rate and length. The architecture itself however is not flexible. In other words, the design needs to be re-synthesized for each different block length and code rate.

The Richardson Urbanke (RU) method of encoding LDPC codes [11] relies on preprocessing the parity matrix before running the encoder. An implementation is presented in [12] that is optimized for code rates of $\frac{1}{2}$. The architecture can be parallelized by instantiating multiple encoders that access the same memory banks in order to encode several codewords in parallel. The RU method involves several arithmetic operations which when implemented on an FPGA need complex controllers to coordinate [12]. A similar architecture is presented in [13], implemented on a DSP platform.

Under the RU method, the parity portion of each message, p, is split into two sub components, p_1 and p_2. Calculation of the first parity portion, p_1, involves two matrix vector multiplication operations [12]. Calculation of p_2 requires that p_1 be known plus an additional matrix multiplication. Since the calculation of p_2 requires p_1, parity bits cannot be generated independently of each other. These multiple matrix calculations constitute a significant impediment to creating a fast flexible encoder.

In this paper, we propose an alternative scheme to encoding LDPC codes that does not rely on the Richardson Urbanke method. The complexity of the RU method is O$(n+g^2)$, where n is the block length and g the 'gap' which can be shown to be on the order of $\sqrt{n}$ [11]. Thus the overall complexity of the RU method is linear. The architecture proposed only uses one vector multiplication. An FPGA implementation takes O(2) to calculate one parity bit and thus has a complexity of O(p), where p is the length of the parity bits. This operation is described in detail in later sections. Our architecture requires less matrix multiplications than the RU method but with increased memory usage. The main features of this architecture are its high throughput and flexibility. Like the RU method, this architecture can be parallelized with many instances thus increasing throughput.

Back substitution is another possible method of encoding [11]. This method relies on a parity matrix in lower triangular form and uses recursion to form the other parity bits as given by

$$P_l = \sum_{j=1}^{n-k} H_{l,j} S_j + \sum_{j=1}^{l-1} H_{l,j+n-k} P_j \qquad (1)$$

This method was used in [10], where they achieved high throughput. However, the disadvantage is that this architecture will take more memory than that proposed. The amount of memory needed for back substitution is $(n-k)k + \frac{k^2}{2}$ bits, which is more than the $(n-k)k$ bits needed for the proposed architecture. In addition, modifying this architecture to support dynamic flexibility would result in a complicated system with more logic elements. Finally, like the RU method, parity bits can not be generated independently from one another, which makes it difficult to parallelize the system.

The rest of paper is organized as follows. Section II briefly introduces LDPC codes and covers some properties of systematic LDPC codes that are used in the WiMAX standard. Section III proposes an encoder architecture which takes advantage of these properties and explains the benefits and tradeoffs of the design. Section IV presents the synthesis results and provides comparison with other architectures. Finally, the paper is concluded in Section V.

II. Systematic Low-Density Parity-Check Codes

LDPC codes are block codes defined as the null space of a sparse parity matrix H. The generator matrix, G, is related to the parity matrix as follows:

$$\begin{aligned} HG^T &= 0 \\ G^T &= null(H) \\ G &= [null(H)]^T \end{aligned} \tag{2}$$

Since all valid codewords, x, satisfy

$$Hx = 0 \tag{3}$$

where x is a column vector. A codeword can be formed from a message, s, by the following formula:

$$x = G^T s \tag{4}$$

For code words of length n, encoding k information bits requires a Generator matrix, G, of size k by n. The code rate, r, of the LDPC code is defined as the ratio of the number of message bits, k, to the total length of the encoded codeword, n:

$$r = \frac{k}{n} \tag{5}$$

In general, the generator matrix G is dense and the calculation of the final code word requires considerable resources. However, systematic LDPC codes have a G matrix with a unique form as described further down. In [3], it was stated that all regular LDPC parity matrices can be transformed to give systematic codes through Gaussian elimination and reordering of columns in the parity check matrix. Furthermore, all LDPC codewords specified in the IEEE 802.16e WiMAX standards are systematic [4]. This means that all codewords, x, comprise the message bits, s, and the parity bits p: $x^T = [s^T p^T]$. Correspondingly, the generator matrix is of the form:

$$G^T = \begin{bmatrix} I_k \\ P^T \end{bmatrix} \tag{6}$$

Where I_k denotes a $k \times k$ identity matrix and P^T, a submatrix of size $n-k$ by k. An example of such a generator matrix is depicted graphically in Figure 1, based on the IEEE 802.16e standard. Thus to encode a systematic LDPC code, only the parity bits need to be calculated as follows:

$$p = P^T s \tag{7}$$

The final codeword is a combination of the original message and the calculated parity portion.

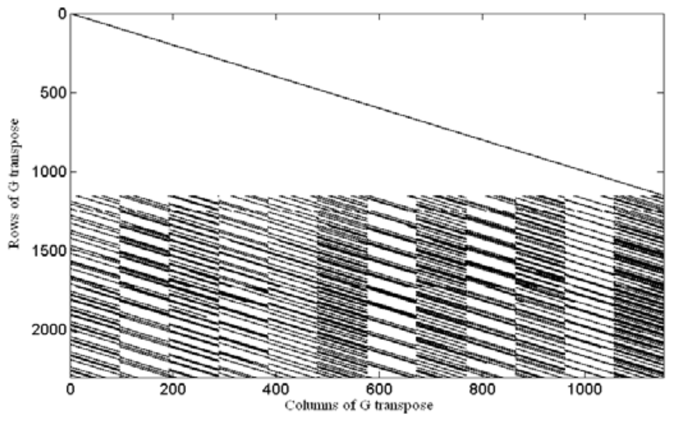

Fig. 1. Generator G^T matrix based on an IEEE 802.16e specified parity matrix obtained by the procedure described in [3]. Non-zero entries of the matrix are identified by dark areas in the plot.

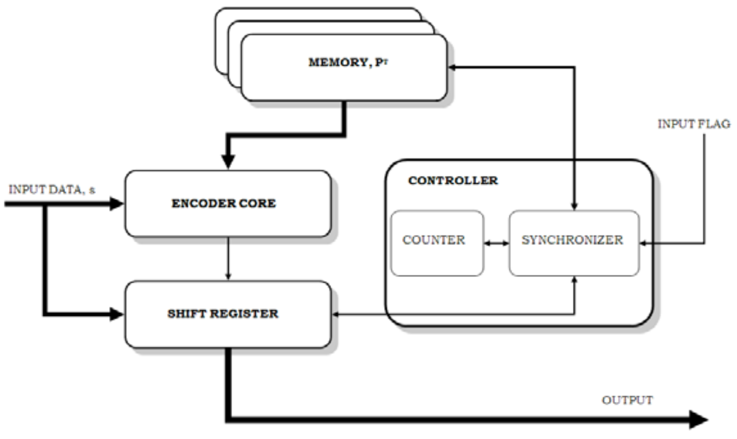

Fig. 2. A schematic of the proposed encoder architecture

III. Proposed Encoder Architecture

In this section, we describe the proposed flexible encoder architecture. The contents of the parity sub-matrix, P^T, code rate and block size of the LDPC code are stored in memory. The parity sub-matrix, code length, and rate can be changed by simply re-writing the contents of the memory.

A. Controller

Controllers are used to coordinate the retrieval of message and parity check bits from memory, parity calculations, and codeword storage. They are governed by two inputs: the rate and the block size of the codeword. The block size indicates the number of bits in the codeword. The code rate, r, of the LDPC code informs the controller on the number of parity bits, p, that are present and need to be calculated in a codeword of length n.

$$p = n(1 - r) \tag{8}$$

The overall encoder architecture is shown in Figure 2. The parity sub-matrix, P^T, is stored in memory. Upon the entry of a new message into the system, controllers fetch the memory contents and pass them to parity calculators. Every clock cycle

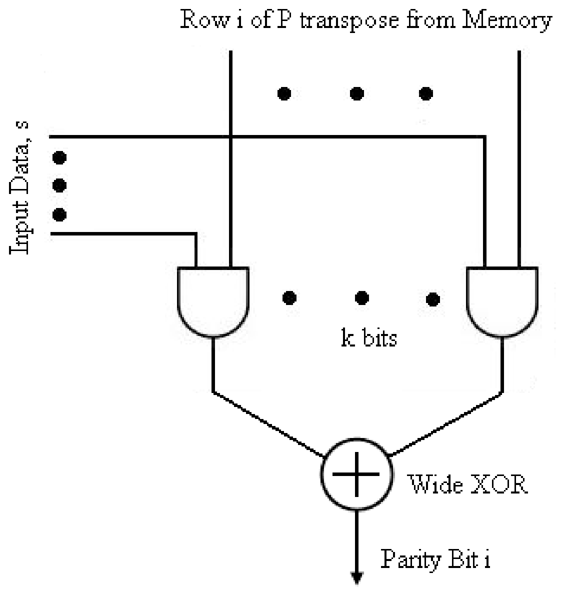

Fig. 3. A diagram of the encoder core which performs vector multiplication. Its inputs are the input message vector and a row of the parity submatrix. The output is a parity bit of the codeword.

results in a new row from the parity sub-matrix being read and the associated parity bit calculated. The results of each parity calculation are then returned to memory. This process is repeated until all parity bits are computed. The final codeword is a combination of the input message and the calculated parity portion.

B. Encoder Core

Each parity bit is the result of binary vector multiplication of the message and one row of the parity submatrix, P^T. As can be seen in Figure 3, this requires binary multiplication (bit wise AND) between the input vector and a row vector i of the parity submatrix. The results of this AND operation are then reduced to one bit through binary summation (XOR). This bit is parity bit i of the parity portion of the codeword. Therefore one parity bit is generated per clock cycle, provided that the clock period is longer than the logic delay of the AND and XOR operations. This can be viewed as a constant time operation to generate one parity bit. Therefore, in order to generate all parity bits the complexity would be O(p), where p is the length of the parity portion. Thus, the overall complexity is linear in time.

Flexibility is supported by a variable counter in the controller. The number of parity bits are related to the code rate and block length. Upon the arrival of a new message, the number of parity bits needed to be processed are computed by the controller. Thus, if there is a change in coding schemes between two messages, the controller will be able to respond and encode correctly. The encoder is designed for the maximum expected block length, n_{max}. Codewords which are smaller than n_{max} bits are padded at the front with 0's. This padding doesn't affect the calculation of the parity bit.

Since one parity bit is generated per clock cycle, the predicted throughput (TP) is the block length (n) divided by the number of parity bits and the clock period.

$$\begin{aligned} TP &= \frac{\text{length}}{\text{time}} \\ &= \frac{n}{n(1-r)(\frac{1}{f})} \\ &= \frac{nf}{n(1-r)} \\ &= \frac{f}{(1-r)} \end{aligned} \tag{9}$$

IV. Results

In order to evaluate the proposed architecture, a Verilog description was synthesized and tested on an Altera Stratix EP1S80F1508C6. The code specified in the IEEE 802.16e WiMAX standard [4] was used. It has a maximum block length of 2304 bits. Upon testing, the highest clock frequency that would support encoding of data at all code rates was 60 MHz. The critical path delay was dominated by the 1920-input XOR required for parity calculation, shown in Figure 3. The total encoder, including all control logic occupies 11,430 logic elements and 3,911,680 memory bits. This truly flexible implementation requires no change to the encoder size or structure when switching to different code rates and/or block lengths.

The memory requirements are steep due to the need to keep the system flexible and accommodate all the various block lengths and rates in the WiMAX standards. The amount of memory bits can be reduced by optimizing the encoder for a particular code rate and block size. However, doing so would violate the flexibility requirement.

The Gaussian elimination required to put G into systematic form (5) was performed just once offline in Matlab for each code in the WiMAX standard. The parity sub-matrices and random input vectors were then loaded into the encoder memory. The encoder output was compared with that produced by a software encoder implemented in Matlab. Thousands of vectors were tested over various block lengths and code rates. All tests demonstrated correct encoding functionality and the ability to reconfigure the encoder for different codes dynamically by only rewriting the memory contents.

In Table I, a comparison between this encoder architecture and previously reported LDPC hardware encoders implemented on FPGA or DSP platforms, is presented. As can be seen in the table, our encoder shows excellent performance compared to similar FPGA implementations. It should be noted that not all of the previous works cited provided sufficient information to perform a thorough comparison. A speedup of 2.5–6 times is provided compared to the implementation of the RU encoding method for comparable codes on a Xilinx Virtex-II XCV4000-6 FPGA

TABLE I
COMPARISON OF LDPC ENCODERS

	Technology	Frequency (MHz)	Data Length (bits)	Code Rate	Logic Elements	Memory bits	Throughput (bps)
[14]	FPGA	30.72	Fixed (1800)	Fixed ($\frac{1}{2}$)	-	-	15M
[15]	FPGA	64	Fixed (1536)	Fixed ($\frac{1}{2}$)	-	-	31M
[10]	FPGA	140 - 195	Fixed (2304) Fixed (2304) Fixed (576)	Fixed ($\frac{1}{2}$) Fixed ($\frac{3}{4}$) Fixed ($\frac{5}{6}$)	10,339 12,727 4,295	-	3.35G 5.17G 6.28G
[13]	DSP	-	Variable	Variable	-	-	2.6M @ Rate $\frac{1}{2}$, length 2304
[12]	FPGA	Various (80-170)	Variable	Variable	-	19 block RAMs	30M - 50M 44M @ Rate $\frac{1}{2}$, length 2000 33M @ Rate $\frac{2}{3}$, length 2000
This work	FPGA	60	Variable	Variable	11,430	3,911,680	115M -360M 119.7M @ Rate $\frac{1}{2}$, length 2304 179.3M @ Rate $\frac{2}{3}$, length 2304 238.8M @ Rate $\frac{3}{4}$, length 2304 357.2M @ Rate $\frac{5}{6}$, length 2304

[12]. The cost of the speed up is an increase in the amount of memory needed. With more bits stored in memory, less calculations need to be performed. In comparison, the RU method requires more matrix calculations but relies on less memory. Similarly, the work of [10] uses back substitution. This method requires more memory than the design presented, but it achieves higher throughput. However, the design in [10] is not flexible in regards to code rate or block size.

One thing of note is that the throughput of the encoders in [12] and [13] vary with block length. In contrast all codes in our design have the same throughput for a fixed code rate regardless of data length. The only factors that affect throughput are the clock frequency and the code rate.

V. CONCLUSION

An architecture for a flexible LDPC encoder has been presented. Instead of the Richardson-Urbanke encoding methods employed in previous works [12], [13], a direct method is employed where the generator matrix is represented in systematic form [3]. This reduces the computations required for encoding and permits a high-speed flexible hardware encoder to be designed. To verify our design, a flexible encoder was implemented for the IEEE 802.16e WiMAX standard. The FPGA encoder had a throughput between 115Mbps and 360Mbps depending on the block length. The result is the fastest flexible FPGA encoder to date, showing an increase in throughput of 2.5–6 times that of prior art.

REFERENCES

[1] R. Gallagher. *"Low-Density Parity-Check Codes"*. M.I.T. Press, Cambridge, MA, 1963.

[2] S-Y. Chung and G.D.F. Forney. "On the Design of Low-Density Parity-Check Codes within 0.0045 db of the Shannon Limit". *IEEE Communications Letters*, 5(2), February 2001.

[3] D.J.C. MacKay. "Good Error-Correcting Codes Based on Very Sparse Matrices". *IEEE Transactions on Information Theory*, 45(2), March 1999.

[4] IEEE. *"IEEE Std. 802.16e-2005 and IEEE Std.802.16-2004/Cor1-2005"*, 2006.

[5] V. Sorokine, F. R. Kschischang, and S. Pasupathy "Gallager Codes for CDMA Applications-Part I: Generalizations, Constructions, and Performance Bounds". *IEEE Transactions on Communications*, 48(10), pages 1660–1668, Oct. 2000.

[6] Moussa, Baghdadi, Jezequel "Binary de Bruijn Interconnection Network for a Flexible LDPC/Turbo Decoder". *IEEE International Symposium on Circuits and Systems*, pages 97–100, May 2008.

[7] Masera, G., Quaglio, F., Vacca, F. "Implementation of a Flexible LDPC Decoder". *IEEE Transactions on Circuits and Systems II: Express Briefs*, pages 542–546, June 2007.

[8] J.-Y. Lee and H.-J. Ryu "A 1-Gb/s Flexible LDPC Decoder Supporting Multiple Code Rates and Block Lengths". *IEEE Transactions on Consumer Electronics*, pages 417–424, May 2008.

[9] D. Hocevar. "LDPC Code Construction with Flexible Hardware Implementation". *IEEE International Conference on Communications*, vol 4, pages 2708–2712, May 2003.

[10] S. Kopparthi and D. Bruenbacher. "Implementation of a Flexible Encoder for Structured Low-Density Parity-Check Codes". In *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, 2007.

[11] Thomas Richardson and Rudiger Urbanke. "Efficient Encoding of Low-Density Parity-Check Codes". *IEEE Transactions of Information Theory*, 47(2), February 2001.

[12] D.U. Lee and W. Luk. "A Flexible Hardware Encoder for Low Density Parity Check Codes". In *12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 101–111, April 2004.

[13] T. Arslan Z. Khan and S. Macdougall. "A Real Time Programmable Encoder for Low Density Parity Check Code as specified in the IEEE p802.16e/d7 Standard and its Efficient Implementation on a DSP Processor". In *IEEE International SOC Conference*, pages 17–20, September 2006.

[14] Su-Chang Chae and Yun-Ok Park. "Low Complexity Encoding of Regular Low Density Parity Check Codes". In *Vehicular Technology Conference*, pages 1822–1826, Oct. 2003.

[15] Jia-ning Su, Zhou Jiang, Ke Liu, Xiao-yang Zeng, Hao Min. "An Efficient Low Complexity LDPC Encoder Based on LU Factorization with Pivoting". In *6th International Conference on ASIC*, pages 107–110, Oct. 2005.