A Low-Complexity Hybrid LDPC Code Encoder for IEEE 802.3an (10GBase-T) Ethernet

Aaron E. Cohen, Member, IEEE, and Keshab K. Parhi, Fellow, IEEE

Abstract—This paper presents a novel hybrid encoding method for encoding of low-density parity-check (LDPC) codes. The design approach is applied to design 10-Gigabit Ethernet transceivers over copper cables. For a specified encoding speed, the proposed method requires substantially lower complexity in terms of area and storage. Furthermore, this method is generic and can be adapted easily for other LDPC codes. One major advantage of this design is that it does not require column swapping and it maintains compatibility with optimized LDPC decoders. For a 10 Gigabit Ethernet transceiver which is compliant with the IEEE 802.3an standard, the proposed sequential (5-Parallel) hybrid architecture has the following implementation properties: critical path: $[\log_2(324)+1]T_{XOR} + T_{AND}$, number of XOR gates: 11,056, number of AND gates: 1620, and ROM storage: 104,976 bits (which can be minimized to 52,488 bits using additional hardware). This method achieves comparable critical path, and requires 74% gate area, 10% ROM storage as compared with a similar 10-Gigabit sequential (5-Parallel) LDPC encoder design using only the G matrix multiplication method. Additionally the proposed method accesses fewer bits per cycle than the G matrix method which reduces power consumption by about 82%.

Index Terms—LDPC, Encoder, 10GBase-T, 802.3an, Hybrid

I. INTRODUCTION

Low density parity check (LDPC) codes are one of the most powerful error correcting codes available to designers today. These were originally proposed by Gallager in 1963 [1]. However more recently they were rediscovered by MacKay, Neal, and Wiberg [2], [3].

LDPC codes have the unique property of being able to achieve near Shannon limit channel capacity which make them extremely efficient in terms of bandwidth utilization. One of their main drawbacks is that they exhibit a higher computation complexity for encoding and decoding, and hence they require more complicated architectures in terms of area, critical path, and power consumption.

For their efficiency, LDPC codes have been adopted into the latest standards for 802.3an, 802.11n, and 802.16e [4]–[6]. While 802.3an uses a fixed LDPC code, 802.11n and 802.16e were designed for multiple rates and hence both require the use of variable rate encoders and decoders for efficient implementation.

Copyright © 2008 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purpose must be obtained from the IEEE by sending a request to pubs-permissions@ieee.org

Manuscript received June 26, 2008; revised October 10, 2008.

This work was supported by National Science Foundation under contract 0441632.

The authors are with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455 USA (email: {cohene82, parhi}@umn.edu).

Previous research on LDPC encoding has focused on complicated methods which work in a variety of cases such as the Richardson-Urbanke (RU) LDPC encoding method [7]. Further research in LDPC encoding has taken advantage of certain properties that exist in certain LDPC codes. These methods include iterative graph-based message-passing encoders [8], and the more trivial designs such as the Quasi-Cyclic LDPC encoding method [9], [10] and structured LDPC encoders [11] which only apply to specific LDPC codes. Research in high performance soft-decision LDPC decoding architecture design originated with approximating the Belief Propagation (BP) algorithm with the fixed point Sum-Product algorithm (SPA) or the Min-Sum algorithm (MSA) [12]. The MSA has been modified further to achieve faster convergence through the use of a scalar factor [13] or a correction factor [14] or both. Due to the numerous methods for implementing LDPC decoders and encoders, comparing architecture designs focuses on comparing bit error rate (BER) performance, throughput, and power dissipation for a given technology. The majority of implementations target Field Programmable Gate Array (FPGA) technology or Application Specific Integrated Circuit (ASIC) technology [15]–[22].

In [18], the authors present a sequential LDPC decoder for IEEE 802.3an (10GBase-T ethernet) that achieves 16GB/s throughput with 4 bit quantization and 8 iterations per frame in 90 nm technology with an area footprint of 9.8 mm². A broadcasting technique was introduced in [19] to mitigate routing congestion for LDPC decoders for 10GBase-T. Throughput was subsequently increased by 60% with their broadcasting technique. Recently a 10.5 Watt chip for 10GBase-T in 0.13 μm has been implemented [20]. The Shannon limit is approximately 3 dB less than the performance achieved with their LDPC decoder.

After ratification of the IEEE 802.16e standard (WiMax) [6], multimode soft decision LDPC decoder design has become increasingly important. These newer LDPC decoder designs must be reconfigurable in terms of allowing different rates (k/n) or modes depending on the channel encoding. In [21], the authors developed a multimode soft decision LDPC decoder consuming only 52mW with an area footprint of 8.29 mm². This decoder is limited to Quasi-Cyclic codes only and is not compatible with the LDPC code from the IEEE 802.3an standard [4]. In [22], the authors present a fully compliant multimode LDPC decoder for IEEE 802.16e. This architecture was able to achieve 109 MHz operating frequency with 90 nm technology and dissipates 186 mW with a 1.0 volt supply.

One of the main reasons to optimize LDPC encoding is to minimize the area and storage complexity of encoding. Reducing area and storage allows for lower power consumption for
the LDPC encoder design. In general, the time complexity has less importance because soft-decision LDPC decoding requires significantly more time to complete than LDPC encoding.

This paper is organized as follows. First an introduction to LDPC and LDPC encoding is provided in Section II. Next the proposed novel low-complexity Hybrid LDPC code encoder is presented in Section III. Section IV provides an overview of architectural optimizations to increase the performance on our low-complexity Hybrid LDPC code encoder. Afterwards we perform a comparison in terms of area and critical path for a field programmable gate array (FPGA) and an application specific integrated circuit (ASIC) assuming our goal is 10 Gigabit Ethernet LDPC encoding (as specified by the IEEE 802.3an standard [4]) in Section V. Then the performances of various LDPC encoder approaches are compared and the advantages of the proposed encoders are demonstrated through these target implementations. Finally a conclusion and future research directions are discussed in Section VI.

II. BACKGROUND

The LDPC block codes are very important error correcting codes that have proven to have excellent performance (i.e., near Shannon limit performance). As in the case of block codes, we define a generator matrix (G) and a parity check matrix (H). In order to achieve a systematic LDPC code G must be in the following form:

\[
G = \left[ \begin{array}{c} I_k \ P \end{array} \right],
\]

where \(I_k\) is an identity matrix and \(P\) defines the parity bits [23].

In some cases, a code may be specified by only the H matrix and it becomes necessary to solve for the G matrix. If the H matrix is in an arbitrary format, it must be converted into echelon canonical form shown below

\[
H = \left[ \begin{array}{c} -P^T \ I_{n-k} \end{array} \right],
\]

where \(I_{n-k}\) is an identity matrix and \(P\) defines the parity bits [23].

This conversion can be accomplished with the assistance of a computer program. Afterwards, the G matrix can be observed by inspection.

Typically, encoding consists of using the G matrix to compute the parity bits and decoding consists of using the H matrix and soft-decision decoding.

The complexity of the LDPC encoding algorithm is much lower than that of various soft decision LDPC decoding algorithms. Prior research includes developing generic encoder design methods such as the G matrix multiplication method and the Richardson-Urbanke (RU) method [7]. For applications with no constraints on types of LDPC codes, more efficient methodologies such as the quasi-cyclic encoder architectures [9] and alternative encoders based on permuted H matrices, which do not require inversions [24], can be used. For now we will focus on constrained encoder design because we are interested in the IEEE 802.3an 10 Gigabit Ethernet standard [4] and its published (2048, 1723) LDPC code which is based on Reed-Solomon codes [25]. Therefore, when simple encoder architectures are not available, two generic encoding methodologies can be applied to implement the LDPC encoder. These are:

1. The (G) matrix multiplication method, and
2. The Richardson-Urbanke (RU) method [7].

A. LDPC Encoding with (G) Matrix Multiplication

The encoding method which uses the G matrix is simply a matrix multiplication. Given systematic bits (s), one can derive the codeword (c) by multiplying by the matrix (G) as follows:

\[
c = s \times G = \left( \begin{array}{c} s \\ p_1 \\ p_2 \end{array} \right).
\]

Then a typical LDPC encoder implementation based on the matrix G multiplication consists of performing the matrix multiplication in a sequential manner [26] such that one only performs the \(p_1\) and \(p_2\) computations as shown in Fig. 1.

\[
G \text{ Matrix ROM (GROM)} \quad k < L 
\]

Fig. 1. Sequential LDPC Encoder based on G Matrix Multiplication

The sequential LDPC encoder based on G matrix multiplication requires a ROM to store the portion of the G matrix used to compute \(p_1\) and \(p_2\). The field multiplication block is composed of AND gates for \(GF(2)\) operations. The field summation block is composed of XOR gates for \(GF(2)\) operations. The AND gates perform the bitwise row multiplication. Then each XOR tree adds up the results to compute one parity bit. The AND and XOR logic gates can be replicated to generate \(L\)-parallel designs (i.e, designs which output \(L\) parity bits in one clock cycle).

B. LDPC Encoding with the Richardson Urbanke (RU) Method [7]

The Richardson-Urbanke (RU) encoding method was first published in [7]. This method uses a unique trick to reformulate the H matrix into a special H matrix which requires the following operations: back-substitution for multiplication by \(T^{-1}\) rather than a true inversion, linear computations, and multiplication by smaller dense matrices. Therefore, this method can reduce the complexity over the G matrix multiplication method. Next the RU method will be described.

First let’s assume that the H matrix can be converted into the approximate lower triangular form as shown in Fig. 2 and below.

\[
H = \left( \begin{array}{ccc} A & B & T \\ C & D & E \end{array} \right),
\]

where \(T\) is lower triangular and

\[
\Phi = -ET^{-1}B + D \quad \text{is non-singular in} \ GF(2).
\]
leads to a long critical path. Thus, it is desirable to explore
the critical path determined by the max

\[ \text{encoder architecture. There are three primary paths with the} \]

substitution or two larger and denser matrix multiplications.

Assuming we are using pre-computation we can perform
row swapping but this may suffer from a large
necessary to swap columns to generate a non-singular matrix.

Then the two equations are given by:

\[ \text{As}^T + Bp_1^T + Tp_2^T = 0, \]

and

\[(ET^{-1}A + C)s^T + (ET^{-1}B + D)p_1^T + 0 = 0.\]

The two equations above allow one to solve for the parity bits
using only the supplied systematic bits such that:

\[ p_1^T = \Phi^{-1}(ET^{-1}A + C)s^T \]

\[ p_2^T = T^{-1}(As^T + Bp_1^T). \]

Assuming we are using pre-computation we can perform
the \( p_1 \) calculation with one dense matrix multiplication or
several smaller sparse multiplications and one smaller dense
multiplication. Similarly the \( p_2 \) calculation can be computed
by two small sparse matrix multiplications and one back
substitution or two larger and denser matrix multiplications.

Fig. 3 contains the final computational flow for the RU
encoder architecture. There are three primary paths with the
critical path determined by the max\{ \( [A, \oplus, -T^{-1}], [A, \]
\( T^{-1}, -E, \oplus, -\Phi^{-1}, B, \oplus, -T^{-1}], [C, \oplus, -\Phi^{-1}, B, \]
\( \oplus, -T^{-1}] \) \}. These three paths contain multiple complex
operations which increases the complexity of the design and
leads to a long critical path. Thus, it is desirable to explore
alternative simpler methods which have a smaller critical path
and significantly less area overhead. Table I contains the area
complexity involved with the encoding process where \( \text{HW}(X) \)
represents the Hamming weight of matrix \( X \).

III. NOVEL HYBRID APPROACH

The last two LDPC encoder designs suffered from two
main problems. First the sequential LDPC encoder required
significant storage overhead which makes it less ideal for
implementation. The RU Method suffered from a long crit-
ical path and odd constraints which could make the LDPC
encoder non-systematic. To overcome these two problems,
we developed a novel low-complexity Hybrid LDPC code
encoder which requires significantly less area and storage and
maintains a systematic encoder form as in Fig. 4. As shown
in Fig. 4, the \( p_1 \) values are generated by the G matrix whereas
the \( p_2 \) values are generated by the RU method. In Fig. 4, the
GROM corresponds to the read only memory which stores the
G matrix coefficients and the TROM corresponds to the
read only memory which stores the T matrix coefficients. This
method will be described next.

The main idea of this method is to compute the \( p_1 \) values
by using the G matrix. Therefore, we do not need the inverse of the
\( \Phi \) matrix as in the RU method. This leads to the Hybrid
method’s primary advantage over the RU method; it does not
require row or column swapping when \( \Phi \) is singular because the
\( p_1 \) values are computed with the G matrix. We propose to
compute the \( p_1 \) values sequentially as in the sequential LDPC
encoder. However, instead of using the G matrix to compute
\( p_2 \) values we use the special property \( p_2^T = T^{-1}(As^T + Bp_1^T) \)
from the RU method to compute \( p_2 \) using a parallel
TABLE I

<table>
<thead>
<tr>
<th>Operation</th>
<th>Requirements</th>
<th>AREA (XORs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Ax^T$</td>
<td>(m-g)×(n-m) sparse mlt</td>
<td>HW(A)−(m-g)</td>
</tr>
<tr>
<td>$T^{-1}[Ax^T]$</td>
<td>(m-g)×(m-g) back sub</td>
<td>HW(T)−(m-g)</td>
</tr>
<tr>
<td>$-E[T^{-1}Ax^T]$</td>
<td>(g)×(m-g) sparse mlt</td>
<td>HW(E)−(g)</td>
</tr>
<tr>
<td>$Cs^T$</td>
<td>(n-m)×(g) sparse mlt</td>
<td>HW(C)−(n-m)</td>
</tr>
<tr>
<td>$[-ET^{-1}As^T] + [Cs^T]$</td>
<td>(n-m) bit addition</td>
<td>(n-m) g^2</td>
</tr>
<tr>
<td>$-\Phi^{-1}[-ET^{-1}As^T + Cs^T]$</td>
<td>(g)×(g) dense mlt</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Requirements</th>
<th>AREA (XORs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Ax^T$</td>
<td>(m-g)×(m-g) sparse mlt</td>
<td>HW(A)−(m-g)</td>
</tr>
<tr>
<td>$Bp^T_2$</td>
<td>(m-g)×(g) sparse mlt</td>
<td>HW(B)−(m-g)</td>
</tr>
<tr>
<td>$As^T + Bp^T_1$</td>
<td>(m-g) bit addition</td>
<td>HW(C)−(n-m)</td>
</tr>
<tr>
<td>$-T^{-1}[As^T + Bp^T_1]$</td>
<td>(m-g)×(m-g) back sub</td>
<td>HW(T)−(m-g)</td>
</tr>
</tbody>
</table>

Fig. 4. Sequential Hybrid LDPC Encoder

sparse matrix multiplication $([A B] \times (s \ p_1) = As^T + Bp^T_1)$ and a sequential back substitution (or matrix multiplication) computation. This has two advantages. First this method has a shorter critical path when the back substitution is implemented in an L-parallel fashion with small L. Second the memory storage requirements can be significantly reduced because T is lower triangular and much smaller than the parity check equations (P) from the G matrix. Therefore this effectively cuts the memory storage in half when using back substitution versus matrix multiplication. Also, if T is singular then no inverse exists and matrix multiplication by $T^{-1}$ cannot be implemented but back substitution is still a valid alternative to obtain the $p_2$ values.

The following steps are required to design the Hybrid encoder. First use row swapping to reformulate the H matrix into approximate lower triangular form as shown in Fig. 2 and below.

$$H = \begin{pmatrix} A & B & T \\ C & D & E \end{pmatrix},$$  \hspace{1cm} (8)

where T is lower triangular of the form

$$T = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ t_{2,1} & 1 & \cdots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ t_{m-g,1} & \cdots & t_{m-g,m-g-1} & 1 \end{pmatrix},$$  \hspace{1cm} (9)

With T in lower triangular form of size $(m-g) \times (m-g)$ we do not need an inverse of T to exist to solve for the $p_2$ bits because back substitution is valid. However it is important to note that back substitution is limited by its linear increase in critical path as L increases where L is the level of parallelism. An example of back substitution for a $4 \times 4$ T matrix is illustrated below.

Example Back Substitution: Finding $y_i$’s from $T$ and $x_i$’s

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ t_{2,1} & 1 & 0 & 0 \\ t_{3,1} & t_{3,2} & 1 & 0 \\ t_{4,1} & t_{4,2} & t_{4,3} & 1 \end{pmatrix} \times \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}. \hspace{1cm} (10)$$

Finding $y_1$ is trivial but it is important to note that subsequent $y_i$’s after $y_1$ require the results from the previous $y_i$’s. Below we have provided the equations required for solving the $y_i$’s given the T matrix above.

$$y_1 = x_1 \hspace{1cm} (11)$$
$$y_2 = x_2 \oplus t_{2,1}y_1 \hspace{1cm} (12)$$
$$y_3 = x_3 \oplus t_{3,1}y_1 \oplus t_{3,2}y_2 \hspace{1cm} (13)$$
$$y_4 = x_4 \oplus t_{4,1}y_1 \oplus t_{4,2}y_2 \oplus t_{4,3}y_3. \hspace{1cm} (14)$$

As can be seen above, increasing parallelism leads to a longer critical path. In this case, the critical path is

$$x_1 \rightarrow y_1 \rightarrow y_2 \rightarrow y_3 \rightarrow y_4 \rightarrow .$$

We can write the equations for the Hybrid encoder by using the G matrix to compute the $p_1$ values (which works even when $\Phi$ is singular) and using the Richardson-Urbanke method to perform the $p_2$ calculations as follows:

$$p_1^T = s \ast G(:, k + 1 : n - (m - g)),$$  \hspace{1cm} (15)

where columns $k + 1$ to $n - (m - g)$ are used and

$$p_2^T = -T^{-1}(As^T + Bp_1^T). \hspace{1cm} (16)$$

Fig. 5 contains a block diagram based on these equations for the low-complexity Hybrid LDPC code encoder. In Fig. 5, the GROM corresponds to the read only memory which stores the G matrix coefficients, the TRROM corresponds to the read only memory which stores the T matrix coefficients, the
MROM corresponds to the read only memory which stores the coefficients for the loop unrolling technique which will be described later, the SR block corresponds to the shift register for storing previous $y_i$ values for the next iteration for an L-parallel design, the $\sum$ blocks correspond to XOR trees for generating the intermediate results of the current $y_i$ computations, the $[A \ B]$ block corresponds to the matrix multiplication specified by the RU method, and the G matrix block corresponds to the matrix multiplication to generate the $p_1$ values.

Finally, the constraints imposed on our method are that the G matrix is known and that the H matrix can be converted into approximate lower triangular form with a relatively large value of $(m-g)$ and sparse matrices A and B. Unlike the RU method we do not require the inverse of a matrix which is not in lower triangular form and therefore we do not need to perform column swapping to find a non-singular matrix. Maintaining column compatibility with the decoder is essential in preventing additional latency and unwarranted storage.

In the case of the RS (2048, 1723) LDPC code ($m-g = 324$ and $A$, $B$ are sparse) we need to compute one ($2048 - 1723 = 324 = 1$) parity bit for $p_1$ (i.e. the GROM is not required) which can be accomplished in $\log_2(854)T_{XOR}$ time by a fully parallel implementation. Pipelining cutsets need to be used before the back substitution architecture to help minimize the critical path. Similarly, the sparse matrix multiplication lends itself well to a fully parallel implementation. It may be beneficial to add a pipelining cutset to the sparse matrix multiplication computation. In this way, the matrix multiplication can be performed in parallel with the $p_1$ computation in order to minimize circuit glitches. For this application $T$ is singular and therefore we must utilize back substitution in the final step of the $p_2$ computation. Finally the majority of the engineering work for this LDPC encoder design will revolve around 1.) optimizing the sequential back substitution architecture involved in the final step of the $p_2$ computation and 2.) in efficient structures for implementing the read only memory blocks.

IV. ARCHITECTURAL OPTIMIZATIONS

In the previous section, we determined that designing an LDPC code encoder for 10GBase-T requires utilizing back substitution and read only memory (ROM) architectures. Both of these architectural characteristics can contribute to long critical path delays. In the case of back substitution the critical path grows linearly with the degree of parallelism whereas in the case of the ROM architectures the critical path is related to the ROM size and structure. In this section we will develop a method of performing balancing on the back substitution operation to shorten the critical path and we will develop fast architectures for implementing the ROM structures.

A. BACK SUBSTITUTION

A typical 5-parallel back substitution design would normally be limited to a critical path or more specifically a loop bound $T_O = (\log_2(324) + 4)T_{XOR} + 4T_{AND}$. Therefore, without optimizing the critical path of the 5-parallel back substitution computation the critical path is longer than the G matrix multiplication. However, with advanced techniques and a small increase in storage overhead we can reduce the critical path below the G matrix multiplication method. First let’s consider the back substitution example for a 2-parallel design as

$$
\begin{bmatrix}
1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4
\end{bmatrix}
= 
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4
\end{bmatrix}
\tag{17}
$$

Now considering $y_i$ and $y_{i+1}$ are computed in the same cycle then it is important to remove the dependence on $y_i$ from the $y_{i+1}$ computation (marked by parentheses $(.)$) in order to minimize the critical path. By conditionally adding the equations for $y_i$ and $y_{i+1}$ together we can generate new equations for $y_{i+1}$ which do not rely on $y_i$ but instead require slightly more computation on the right hand side of the equation as shown below

$$
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4
\end{bmatrix}
= 
\begin{bmatrix}
x_1 \\
x_1 + x_2 \\
x_3 \\
x_3 + x_4
\end{bmatrix}
\tag{18}
$$

$$
\begin{bmatrix}
x_1 \\
x_1 + x_2 \\
x_3 \\
x_3 + x_4
\end{bmatrix}
= 
\begin{bmatrix}
1 & 0 \\
1 & 1 \\
1 & 0 \\
1 & 1
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4
\end{bmatrix}
\tag{19}
$$

This new modification has removed the dependence but requires the additional storage of 1 bit (marked by braces above) per cycle. This technique is a modified version of look-ahead adapted for our specific application. A typical hardware implementation, Fig. 6, would consist of performing the previous $y_i$ computation and the $x_i$ computation in parallel followed by an exclusive or gate to generate the current $y_i$'s.
of the parallel load operation which occurs in the beginning before the computation cycles. Hence this method is the fastest available implementation method for the read only memories because it has the least capacitance on the output leading to the computation blocks in the hybrid LDPC encoder. This method has the following disadvantage: higher power consumption. The numerous read and write operations cause the system to have higher power consumption than an alternative based on the ROM implementation.

Comparison: The best method for comparing these two techniques is to compare their power consumption. Power consumption on a wire with a capacitance can be computed using the following formula

\[ P = C_L V_{DD}^2 f, \]  

where \( C_L \) is the capacitance load, \( V_{DD} \) is the supply voltage, and \( f \) is the frequency.

For the shift register implementation the number of switching nodes is related to the Hamming weight of the matrix. For the lower triangular \( T \) matrix this corresponds to a Hamming weight of 1457. The position of the ones dictates how many times it will cause a transition. Therefore the power consumption of the shift register implementation of the ROM can be computed by the following formula

\[ P_{SR} = \sum_{i=1}^{HW(MATRIX)} (maxcycles - i_{cycles}) \times P_{FF}, \]  

where \( HW(MATRIX) \) represents the Hamming weight of the matrix, \( i_{cycles} \) represents the number of cycles for the \( i \)’th bit, \( maxcycles \) represents the number of cycles for the \( L \)-parallel design, and \( P_{FF} \) represents the standard power equation.

For the PLA implementation the number of switching nodes is related to the number of bits read and the Hamming weight of each entry. Therefore the power consumption of the PLA implementation of the ROM can be computed by the following formula

\[
P_{PLA1} = \sum_{i=1}^{n} (m - HW(row_i)) \times P_{BUS} \\
= (m \times n - HW(MATRIX)) \times P_{BUS},
\]

where \( HW(row_i) \) represents the Hamming weight of the \( i \)’th row, \( m, n \) represent the size of the matrix, and \( P \) represents the standard power equation. Alternatively, if the ROM stores only the information for the location of the ones then

\[
P_{PLA2} = \sum_{i=1}^{n} (HW(row_i)) \times P_{BUS} \\
= HW(MATRIX) \times P_{BUS}.
\]

We compared the power consumption of these two methods using the \( T \) matrix from the low-complexity Hybrid LDPC code encoder for 10GBase-T Ethernet and obtained the results shown in Table II. As can be seen from this table the shift register method achieves lower power consumption when compared to storing the location of the zeros. Although its power factor appears higher in the sequential design it benefits
tremendously once parallelism is applied while the PLA-like method based on precharging does not. Ideally, the second method (PLA2) for storing the location of the ones should be applied when using low Hamming weight matrices. This second method achieves better power consumption but requires a charging and evaluation phase plus an additional inversion to generate the correct output and hence will increase the critical path. In either PLA case, implementing a static CMOS design is not practical due to the complexity of the pull-up (PLA2) or pull-down (PLA1) network.

![Table II](image)

<table>
<thead>
<tr>
<th>Method</th>
<th>Power Consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>106, 104P_{FF}</td>
</tr>
<tr>
<td>PLA1</td>
<td>103, 519P_{BUS}</td>
</tr>
<tr>
<td>PLA2</td>
<td>1, 457P_{BUS}</td>
</tr>
</tbody>
</table>

V. PERFORMANCE ANALYSIS AND COMPARISON

The previous sections have focused on the development of methodologies for implementing LDPC code encoders and architectural techniques to improve the encoders; however a detailed comparison has not been made. This section explores the design characteristics for the different LDPC code encoders. First, this section will compare our novel hybrid LDPC encoder in terms of implementation on a field programmable gate array (FPGA). Afterwards, this section will analyze our novel hybrid LDPC encoder in terms of implementation on an application specific integrated circuit (ASIC). Finally, this section will compare the FPGA and ASIC LDPC encoder implementations with other published encoder designs.

A. FPGA Implementation Analysis and Comparison

We took our design from the previous sections (Hybrid/Seq) and implemented it in Verilog HDL. This design consists of the ROM implemented as a ROM by using the Xilinx ROM block, TROM implemented as shift registers due to FPGA design constraints on memory implementations, and the $p_1$ computation computed immediately in a 5-Parallel hybrid LDPC code encoder design. This method was compared with the fully parallel Hybrid/RU method. The RU method requires several matrix multiplications to compute $p_1$ but we have instead used the Hybrid method which makes use of the $G$ matrix to compute $p_1$. By simplifying the matrix multiplications from the RU method into one equation the $G$ matrix equation can be derived but it technically is not using the RU method anymore because then it computes $p_1$ using the $G$ matrix. We believe a practical implementation would take this simplification with or without the understanding that it is now using the $G$ matrix to minimize latency. Therefore, we are calling this method the fully parallel Hybrid/RU method because it makes use of the $G$ matrix but it does not make use of the folding techniques. Additionally we implemented the $G$ matrix method in VHDL and a fully parallel pipelined Hybrid/RU method with the $G$ matrix for calculating $p_1$ in Verilog for comparison. The Hybrid/Seq design is compared to the fully parallel Hybrid/RU design we implemented, the fully parallel design published on OpenCores [30], and a 5-parallel $G$ matrix multiplication method which we implemented. We utilized Xilinx ISE 9.1i (www.xilinx.com) to generate the statistics for a Virtex 4 FPGA. This allows us to analyze our method compared to alternative methods and generate useful statistics on the hardware requirements. The results in Table III were obtained by synthesizing only the LDPC encoder without serial to parallel conversions which are required for implementation due to the limited pin count. The timing analysis was performed with Xilinx Timing Analyzer on the implemented design with the serial to parallel conversions. These results indicate that the Hybrid/RU methodology generates a more efficient encoder design.

The OpenCores design did not fit on the chip. It was a factor of 2 larger than our design. Additionally, the $G$ matrix methods ran for several days before running out of memory; therefore, they did not complete.

Fig. 7 includes a picture of the Hybrid LDPC code encoder synthesized to the Xilinx Virtex 4 (xc4vlx25-12sf363) with serial to parallel converters. The OpenCores design used more resources than is available in the FPGA; therefore, it has been omitted. Additionally, the $G$ matrix method did not complete the synthesis phase of the design.

![Fig. 7](image)

B. ASIC Implementation Analysis and Comparison

In order to achieve a fair comparison for application specific integrated circuit (ASIC) implementation we took a target application (RS (2048, 1723) LDPC code for 10 Gigabit Ethernet) with a total cycle limitation of 80 cycles which requires at the minimum a 5-parallel LDPC encoder and derived the following architectural requirements listed in Table IV. Technically, the Hybrid method can be implemented in a 3-parallel fashion without pipelining or a 4-parallel fashion with pipelining due to the relative sparseness of the $T$ matrix (the first 86 rows contain a single one) but our comparison is for the un-optimized general case. In Table IV, the Hamming weights of the matrices were used in computing the critical path and gate count, the matrix size was used in computing the ROM storage, and the number of bits for pipelining was used in computing MEM for the LDPC encoders. For reference the values for implementing the fully parallel Hybrid/RU method are listed. This method was optimized to use the $G$ matrix for computing the $p_1$ value but is identical to the parallel RU method after solving the equation for $p_1$. 
TABLE III
FPGA IMPLEMENTATION COMPARISON

<table>
<thead>
<tr>
<th>Method</th>
<th>Parallel</th>
<th>Slices</th>
<th>FFs</th>
<th>LUTs</th>
<th>Timing Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>G matrix (Ours)</td>
<td>Fully</td>
<td>5</td>
<td>10752</td>
<td>(21504)</td>
<td>Xilinx WebPack ISE 9.1i ran out of memory</td>
</tr>
<tr>
<td>G matrix (Ours)</td>
<td>5</td>
<td>816</td>
<td>324</td>
<td>(1722)</td>
<td>Xilinx WebPack ISE 9.1i ran out of memory</td>
</tr>
<tr>
<td>OpenCores.org [30]</td>
<td>Fully</td>
<td>18757</td>
<td>(174%)</td>
<td>0</td>
<td>32679 (151%)</td>
</tr>
<tr>
<td>Hybrid/RU</td>
<td>4525</td>
<td>(42%)</td>
<td>6336</td>
<td>(29%)</td>
<td>6333 (29%)</td>
</tr>
<tr>
<td>Hybrid/Seq</td>
<td>8816</td>
<td>(81%)</td>
<td>11911</td>
<td>(55%)</td>
<td>15618 (72%)</td>
</tr>
</tbody>
</table>

TABLE IV
ASIC IMPLEMENTATION COMPARISON

<table>
<thead>
<tr>
<th></th>
<th>Parallel</th>
<th>Sequential</th>
<th>Percentage Decrease</th>
</tr>
</thead>
<tbody>
<tr>
<td>XOR</td>
<td>23,397</td>
<td>8,610</td>
<td>–28.4%</td>
</tr>
<tr>
<td>AND</td>
<td>0</td>
<td>8,615</td>
<td>81.2%</td>
</tr>
<tr>
<td>OR</td>
<td>0</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>ROM</td>
<td>0</td>
<td>559,975</td>
<td>81.3%</td>
</tr>
<tr>
<td>MEM Critical Path</td>
<td>4096 bits</td>
<td>log(<em>2)(854)T(</em>{XOR})</td>
<td>~ 9.4%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>log(<em>2)(1722)T(</em>{XOR})+T(_{AND})</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>log(<em>2)(854)T(</em>{XOR})</td>
<td>(log(<em>2)(324)+1)T(</em>{XOR})+T(_{AND})</td>
</tr>
</tbody>
</table>

One of the main points to observe from Table IV is the significant reduction in storage overhead (10% of Sequential G) for the Hybrid Method by only storing the lower triangular T matrix. This is an extremely important result because instead of accessing (reading) 5*1723 bits per cycle we only read 5*324 bits per cycle. Therefore, one can expect a 82% power savings (18% of prior power consumption) simply by reducing the ROM size. Further power savings can be achieved through pipelining [26]. Overall the hybrid LDPC code encoder achieves a decrease in all aspects except XOR logic gate counts compared with the G matrix method. Another interesting point is that the total number of gates (XOR, AND, and OR gates) has been decreased to 74% of sequential G matrix method. Finally the critical path is similar for both the sequential G and hybrid encoder methods with the hybrid encoder method achieving a slightly shorter critical path.

As for the ROM implementations, we are suggesting using shift registers for speed due to the sequential access required for the encoder design. While this will increase the power consumption it will minimize the critical path by allowing a static CMOS implementation of the ROM. If the critical path is not constrained then a dynamic CMOS implementation of a PLA network is possible for the ROM.

Generating meaningful ASIC implementation results will require a proper design flow for a 65-90nm standard cell library.

C. Comparison to Published Works

The encoder design for Fewer et al., requires 614 slices with 11 BlockRAMs for FPGA implementation or 17-k ASIC gates [15]. They reported a maximum clock frequency of 278MHz. Our design targets a different FPGA therefore the implementation results are not comparable. The 802.3an design uses a rate = 1723/2048 ≈ 0.84 LDPC code therefore the performance of the Fewer et al architecture should be closer to their rate 0.75 design given as 3.34 Gbps throughput. Analyzing their design it appears to require 575 cycles per codeword. This is significantly slower than our sequential design which only requires 65 + 2 = 67 cycles. Their estimate on the ASIC gates is higher than our sequential design but lower than the parallel design. Additionally, their simulation does not include results for BER less than 10⁻⁶ which makes it difficult to compare their code’s performance with the 10GBase-T LDPC code.

The encoder design for Lee et al., requires 9513 edges, 1065 slices, 18 Block RAMs, and obtained a latency of 0.235 ms for a rate 2/3 code with block length of 2000 bits on a Xilinx Virtex-II XC2V4000-6 FPGA [16]. This design is highly modular and reconfigurable and it can support Irregular and Regular LDPC codes using the RU methodology. Analyzing their design it appears to require (125 MHz * 0.235 ms/codeword) ≈ 29375 cycles per codeword. While this paper targeted a previous generation FPGA, this design requires a significant number of cycles and therefore is incompatible with the high speed implementation requirement for 10GBase-T Ethernet.

In [17], Quasi-cyclic LDPC codes were considered. This allowed the researchers to achieve performance over 10 Gbit/sec. This paper did not include detailed synthesis results but it is similar to the Fewer et al code discussed previously.

VI. Conclusion

This paper has presented a low-complexity Hybrid LDPC code encoder for IEEE 802.3an (10GBase-T) Ethernet. Unlike the prior LDPC encoder methodologies our method is generic and will work with any LDPC code that can be reformulated into an approximate lower triangular form through row (or the less preferred column) swapping.

We have investigated several methods for implementing the read only memory (ROM) blocks such as programmed logic array (PLA) like methods and shift register methods. We found that the shift register method is best for FPGAs and ASICs when there is a higher level of parallelism because it makes use of nearest neighbor communication whereas the NAND-NOR methods based on precharging work best for sequential ASIC designs.
We investigated the implementation characteristics of multiple LDPC code encoders on a field programmable gate array (FPGA). We found that the area requirements for a fully parallel design were between 4 and 6 times larger than a fully parallel Hybrid/RU design and 2 and 3 times larger than an L-parallel design based on our Hybrid LDPC code encoder method. Similarly, we found that the memory requirements for the G matrix multiplication technique exceeded the requirements for our Hybrid LDPC code encoder.

We further investigated the characteristics of multiple LDPC code encoders for application specific integrated circuit (ASIC) implementations. We found that the Hybrid LDPC code encoder achieves significant decreases in logic (74%) and storage (81%) complexity while minimizing the critical path (~10%). Although we were unable to obtain power consumption results for a 65-90nm process we can make use of the analytical power consumption models developed for our ROM to estimate the power consumption reduction to be 82% when using our Hybrid LDPC code encoder.

After analyzing the FPGA and ASIC implementation characteristics, the Hybrid designs were compared with another encoder implementation. It was found that for FPGA and ASIC implementation the 10GBase-T LDPC code requires fewer clock cycles for similar area requirements.

Although more complicated LDPC encoder designs exist, it is not necessary to be constrained to their limitations. By using the best parts from two well known methods on the RS (2048, 1723) LDPC code we have achieved a new Hybrid LDPC encoder which outperforms existing known methods (G matrix multiplication and the Richardson Urbanke method) in terms of storage, area, critical path, and power consumption. We believe this new architecture may be extremely useful to other codes besides the published RS (2048, 1723) LDPC code for 10 Gigabit Ethernet but this requires further study. Some LDPC codes will benefit more from the proposed reformulation while others will benefit less. Exploring specific properties of codes that lead to better reformulations and implementations of encoders with low complexity is a topic that requires further study. Also exploring the effects of row and column swapping on decoder implementation is a topic that requires further study.

ACKNOWLEDGEMENTS

This research was carried out while the author was with Leanics Corporation, Minnesota.

We would like to thank the reviewers and the Associate Editor for their assistance with improving this paper by their valuable feedback on our manuscript.

REFERENCES


Aaron E. Cohen received the B.S. degree in Computer Engineering at the University of Illinois Urbana Champaign in 2002. Afterwards, he attended the University of Minnesota Twin Cities where he received an M.S.E.E. degree in 2004 and Ph.D. in 2007. Currently, he is Project Manager at Leanics Corporation in Minneapolis, MN.

His current research interests include security, voice over Internet protocol (VoIP), design of algorithms, VLSI architectures, and circuits for cryptography and communication systems, with emphasis on error-correcting coding, modular arithmetic, and finite field arithmetic. He is a current member of the IEEE.

In 2003, he worked part time as a Research and Development Intern for Medical Graphics Corporation, which is a medical device company located in Saint Paul, MN.

In 2005, he worked as a Research and Development Intern at Leanics Corporation on 10 Gigabit Ethernet. Beginning in 2006, he became Project Manager and Senior Research and Development Engineer of the SBIR contract N06-086: Tactical Secure Voice/Variable Data Rate Inter Working Function which focuses on developing novel solutions to secure Voice over Internet Protocol (VoIP).

Keshab K. Parhi received his B.Tech., MSEE, and Ph.D. degrees from the Indian Institute of Technology, Kharagpur, the University of Pennsylvania, Philadelphia, and the University of California at Berkeley, in 1982, 1984, and 1988, respectively. He has been with the University of Minnesota, Minneapolis, since 1988, where he is currently Distinguished McKnight University Professor in the Department of Electrical and Computer Engineering. His research addresses VLSI architecture design and implementation of physical layer aspects of broadband communications systems, error control coders and cryptography architectures, high-speed transceivers, and ultra wideband systems. He is also currently working on intelligent classification of biomedical signals and images, for applications such as seizure prediction, lung sound analysis, and diabetic retinopathy screening. He has published over 450 papers, has authored the text book VLSI Digital Signal Processing Systems (Wiley, 1999) and coedited the reference book Digital Signal Processing for Multimedia Systems (Marcel Dekker, 1999). Dr. Parhi is the recipient of numerous awards including the 2004 F.E. Terman award by the American Society of Engineering Education, the 2003 IEEE Kiyo Tomiyasu Technical Field Award, the 2001 IEEE W.R.G. Baker prize paper award, and a Golden Jubilee award from the IEEE Circuits and Systems Society in 1999. He has served on the editorial boards of the IEEE TRANSACTIONS ON CAS, CAS-II, VLSI Systems, Signal Processing, Signal Processing Letters, and Signal Processing Magazine, and served as the Editor-in-Chief of the IEEE Trans. on Circuits and Systems - I (2004-2005 term), and currently serves on the Editorial Board of the Journal of VLSI Signal Processing. He has served as technical program cochair of the 1995 IEEE VLSI Signal Processing workshop and the 1996 ASAP conference, and as the general chair of the 2002 IEEE Workshop on Signal Processing Systems. He was a distinguished lecturer for the IEEE Circuits and Systems society during 1996-1998. He was an elected member of the Board of Governors of the IEEE Circuits and Systems society from 2005 to 2007.
