# Encoders for Block-Circulant LDPC Codes

Kenneth Andrews, Sam Dolinar, and Jeremy Thorpe
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA, USA
Email: {andrews, sam, jeremy}@shannon.jpl.nasa.gov

***Abstract*— In this paper, we present two encoding methods for block-circulant LDPC codes. The first is an iterative encoding method based on the erasure decoding algorithm, and the computations required are well organized due to the block-circulant structure of the parity check matrix. The second method uses block-circulant generator matrices, and the encoders are very similar to those for recursive convolutional codes. Some encoders of the second type have been implemented in a small Field Programmable Gate Array (FPGA) and operate at 100 Msymbols/second.**

## I. Introduction

Recently, block-circulant LDPC codes have been found that provide both excellent error correction performance and well structured decoder architectures. Constructions have been presented by Lin *et al* [1], [2], Tanner *et al* [3], [4], Milenkovic *et al* [5], the authors [6], and others. In this paper, we explore some encoder designs for these codes, and discuss a hardware encoder implementation.

We define a circulant as a square binary matrix where each row is constructed from the previous row by a single right cyclic shift; we do not require that each row has Hamming weight 1. An $rT \times nT$ parity check matrix $H$ can be constructed by concatenating $r \times n$ sparse circulants of size $T \times T$. The density of each circulant matrix is indicated by the corresponding value in an $r \times n$ base matrix $H_{\rm base}$. The Tanner graph corresponding to this matrix is called a protograph [7]. Entries greater than 1 in the base matrix correspond to multiple edges in the protograph. Base matrices can be expanded into block-circulant LDPC codes by replacing each entry in $H_{\rm base}$ with a circulant containing rows of the specified Hamming weight; the resulting codes are quasicyclic. Alternatively, they can be expanded into less structured codes by replacing each entry with a sum of arbitrary permutation matrices.

Protographs for our AR3A and AR4A codes of rate 1/2 are shown in Figures 1 and 2, and we use these as examples throughout the paper. Squares are parity check nodes and circles are variable nodes, where the solid circles represent transmitted symbols and the open ones are punctured. These designs were derived from a three step encoding procedure: accumulate, repeat-by-3 (or 4), and accumulate [8]; hence their names. Each protograph describes a $3 \times 5$ block-circulant parity check matrix, and the number of parallel edges shows the degree of the corresponding circulant.

In practice, these protographs cannot be directly expanded into block-circulant codes without introducing low weight codewords, regardless of the choice of circulants. A practical solution is to expand the protographs twice, first with small permutation matrices, such as of size $4 \times 4$ or $8 \times 8$, and then with circulants to build the full code. The result is a parity check matrix such as the one shown in Figure 3 for a very small AR4A code, where each nonzero entry in the matrix is represented by a dot. This code was constructed by putting the AR4A protograph variable nodes in the order $(4, 2, 1, 5, 3)$ and check nodes in order (A, B, C) as demarcated by the solid lines, expanding with $4 \times 4$ permutations, and then expanding with $16 \times 16$ circulants. The resulting $12 \times 20$ block-circulant structure is emphasized by dotted lines.

In Section II, we examine iterative encoders, similar to those of Richardson and Urbanke in [9], that also take advantage of the block-circulant structure of the parity check matrix. In Section III, we examine the existence and construction of systematic block-circulant generator matrices. In Section IV, we develop simple hardware circuits for encoders using block-circulant generator matrices, and describe an FPGA implementation. Concluding remarks are in Section V.

## II. Iterative Encoders

An encoder for any $(N, K)$ LDPC code can be built from an erasure correcting decoder. A set of $K$ linearly independent variable nodes are selected as the systematic symbols, and these are initialized with the $K$ information bits to be encoded. If there are no stopping sets, then the remaining $N - K$ parity symbols are computed iteratively with the standard erasure

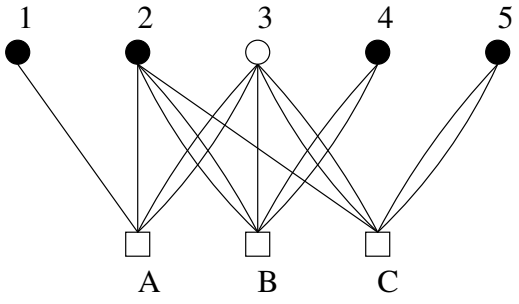


Fig. 1. The AR3A protograph

[1]This work was funded by the IND Technology Program and performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

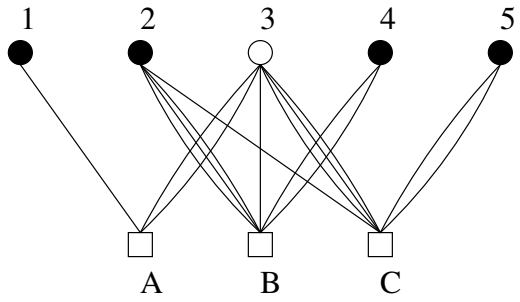


Fig. 2. The AR4A protograph

correcting algorithm. Because the known symbol positions are known *a priori*, the existence of stopping sets is also known. This method is equivalent to Richardson and Urbanke's low-complexity encoding algorithm [9] when their variable $g = 0$.

If $H$ has full rank $R = N - K$, and this iterative encoding method succeeds, then each of the $N - K$ parity check equations is solved exactly once to determine one of the $N-K$ unknown parity symbols. For a check equation with $d$ terms, $d - 2$ exclusive-OR operations are required. Thus, iterative encoding requires exactly $E - 2R$ exclusive-OR operations, where $E$ is the number of nonzero elements in $H$. For an arbitrary LDPC code, the scheduling of these computations can be complex; for block-circulant codes, they can be performed in well organized groups of $T$ operations. The amount of memory required in such a decoder varies depending on the code structure; it is sufficient to store all $N$ code symbols.

We illustrate these ideas with the AR3A and AR4A code examples. When the rows and columns of the AR4A base matrix are reordered as (B, A, C) and (4, 2, 3, 1, 5), we get,

$$H_{\text{base}} = \begin{bmatrix} 2 & 3 & 1 & 0 & 0 \\ 0 & 0 & 2 & 1 & 0 \\ 0 & 1 & 3 & 0 & 2 \end{bmatrix}.$$

Iterative encoding begins by applying the $kT = 2T$ information symbols to the first two columns in the base matrix. The first row of $T$ check equations can be solved in parallel to determine the third column of code symbols, and then the next row can be solved to determine the fourth column. The 2 in the lower right corner means that each remaining check equation has two unknowns, and iterative encoding is halted by the stopping set. However, note that this parity check matrix is not full rank: the sum of the first $T$ and last $T$ rows of $H$ is the all-zero vector, independent of the circulants chosen. This means that one of the remaining $T$ undetermined code symbols can be assigned an additional information bit, and iterative encoding now completes successfully, operating (in a permuted order) as an accumulator of length $T$.

The AR3A code shows somewhat different behavior. With the same row and column ordering, the AR3A base matrix is

$$H_{\text{base}} = \begin{bmatrix} 2 & 2 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 1 & 2 & 0 & 2 \end{bmatrix}.$$

Foreseeing the problematic 2 in the lower right corner, we can construct one redundant check equation by summing the last $T$ rows of $H$ to get the length $N = 5T$ vector, $h = \begin{bmatrix} 0_T & 1_T & 0_T & 0_T & 0_T \end{bmatrix}$, where $0_T$ and $1_T$ represent strings of $T$ zeros and $T$ ones, respectively. This check equation shows that the first $2T$ variable nodes are not linearly independent, and cannot all be assigned information bits. Instead, we assign information bits to the first $2T - 1$ and to the very last variable node. Iterative encoding begins with the constructed check equation $h$, which computes the $2T$'th code symbol as the parity of the preceding $T - 1$ symbols. Iterative encoding then proceeds to completion exactly as for the AR4A code.

## III. Encoders Using Block-Circulant Generator Matrices

LDPC code constructions using circulant matrices invite some algebraic analysis. We begin by noting some properties of a single circulant. The set of circulant matrices of size $T \times T$, here denoted $\mathcal{C}_T$, forms a ring under the standard operations of modulo-2 matrix addition and matrix multiplication. The zero element is the all-zero matrix, and the identity is the $T \times T$ identity matrix (both of which are circulants). For a circulant $M$ of size $T \times T$, we can associate with it a polynomial $m(x)$ of degree $\leq T - 1$ in the indeterminate $x$. We do this by taking the symbols in the first row of $M$ as the coefficients of ascending powers of $x$. These polynomials also form a ring, under the operations of polynomial addition, and polynomial multiplication modulo $x^T + 1$. For example,

$$M = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \in \mathcal{C}_T \text{ corresponds to}$$
$$m(x) = 0 + 1x + 0x^2 + 0x^3 = x \in GF_2[x]/(x^4 + 1).$$

This mapping from $M$ to $m(x)$ defines a ring isomorphism, and maps the all-zero circulant to the zero polynomial, and the identity circulant to the polynomial 1. It also maps the matrix transpose $M^T$ to $m(x^{-1})$.

The ring $GF_2[x]/(x^T + 1)$ has zero-divisors: for example, $(1+x^i)(1+x+x^2+\cdots+x^{T-1}) = 0$ for any $i \neq 0$. By the ring isomorphism, the corresponding circulants multiply to zero. In matrix language, neither circulant is full rank. This fact works in reverse, so we conclude that $M$ is full rank if and only if $m(x)$ is not a zero divisor in the ring $GF_2[x]/(x^T+1)$. Rank-deficient circulant matrices take a few qualitatively different forms. If $T$ is divisible by the integer $p$, then the circulant corresponding to $1 + x^p + x^{2p} + \cdots + x^{T-p}$ contains repeated rows. If $m(1) = 0$, i.e. $m(x)$ has an even number of terms, then $(1 + x + \cdots + x^{T-1})m(x) = 0$ and the circulant is rank deficient. Neither is required; the circulant $1 + x + x^3$ with $T = 7$ only has rank 4, for example.

Next, we consider square block matrices composed of circulants. Let $S$ of size $rT \times rT$ be composed of $r \times r$ circulants. Continuing to make use of our ring isomorphism, let $s$ be an $r \times r$ matrix of polynomials corresponding to the

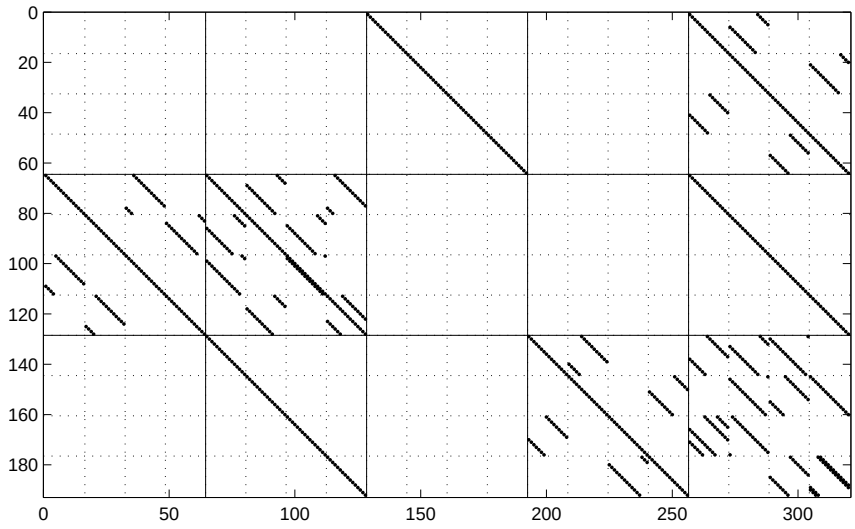


Fig. 3. A Block-Circulant Parity Check Matrix Built From the AR4A Protograph

circulants in $S$. It is not hard to show that $S$ is invertible if and only if $\det(s)$ is not a zero divisor. Moreover, when $S$ is invertible, there exists an $r \times r$ polynomial matrix $w$ such that $ws = I_r$, the $r \times r$ identity (polynomial) matrix.

Our particular interest is in LDPC codes defined by a block matrix $H$ composed of circulants. Let $H$ have size $rT \times nT$, where $r < n$. A quasicyclic code is one for which a "quasicyclic shift" of a codeword is also a codeword. That is, if we partition any codeword $c$ into binary strings of length $T$, and circularly shift each string by the same amount, the resulting vector is also a codeword. It is immediate that any LDPC code defined by a block-circulant $H$ matrix is quasicyclic.

In some cases, such a code has a systematic generator matrix $G$ of size $(n-r)T \times nT$ that is entirely composed of circulants. To show this, we partition $H$ so that $H = [\ Q \quad S\ ]$, where $S$ is square. If $S$ is invertible, the traditional method for constructing $G$ is to find a matrix $W$ such that $WS = I_{rT}$, the $rT \times rT$ identity matrix. Then a systematic generator matrix is $G = [\ I_{(n-r)T} \quad (WQ)^T\ ]$. Algebraically, we can perform the same construction in the isomorphic polynomial ring. After finding the $r \times r$ matrix $w$ such that $ws = I_r$, it is a simple matter to compute $g = [\ I_{n-r} \quad q(x^{-1})w(x^{-1})\ ]$. Replacing the elements of this array with the corresponding circulants, we have a block-circulant generator matrix $G$ for the original code.

Not all block-circulant LDPC codes have block-circulant generator matrices. As a particularly small example, suppose $H$ is described by the single circulant $1 + x + x^3$ with $T = 7$. As noted above, this only has rank 4. One codeword is $[1\ 1\ 1\ 0\ 1\ 0\ 0]$, and because the code is quasi-cyclic (in fact cyclic, because $H$ consists of a single circulant), all cyclic shifts of this codeword are also codewords. However, the circulant corresponding to $1 + x + x^2 + x^4$ only has rank 3, and so cannot be used in its entirety as a generator matrix. In general, if $S$ is not full rank (or equivalently, $\det(s)$ is a zero divisor), then $G$ cannot be quasicyclic.

In the remainder of this section, we return to the AR3A and AR4A codes introduced earlier as practical examples.

The $3T \times 5T$ parity check matrix for AR3A is full rank, and so a generator matrix for this code will have dimension $2T$. We partition $H$ into $[\ Q \quad S\ ]$, where $Q$ contains the columns we wish to make systematic, and $S$ is the square matrix that must be invertible. If we choose $Q$ to include the circulants corresponding to variable nodes 4 and 2 in the protograph, as we did for the iterative encoder, we find that $S$ has rank $rT - 1$, deficient by 1. This misfortune occurs because of the closed loop of degree-2 variable nodes created by protograph nodes 5 and C.

Alternatively, we can choose to make protograph variable nodes 4 and 5 systematic. In this case, $S$ has full rank, and a systematic block-circulant $G$ can be calculated exactly as described. An encoder that performs matrix multiplication by $G$ is particularly suitable for hardware implementation as described in the next section.

As a second example, we look at the AR4A code. For this code, there is no set of $R$ columns that can be selected from $H$ to form an invertible square matrix $S$, because $H$ itself is rank deficient by 1. Perhaps remarkably, these two defects cancel and the method for constructing $G$ can proceed with minor modifications. We select variable nodes 4 and 2 to be systematic, and when $H$ is arranged to put these on the left, it appears as shown in Figure 3. The left two fifths of $H$ is the matrix $Q$, and the remaining square portion on the right is $S$. We solve to find codewords of the form $c_4 = [\ 1 \quad 0 \quad p_1(x) \quad p_5(x) \quad p_3(x)\ ]$, and of the form $c_2 = [\ 0 \quad 1 \quad p_1(x) \quad p_5(x) \quad p_3(x)\ ]$. By expanding these solutions into circulants, we can form a block-circulant

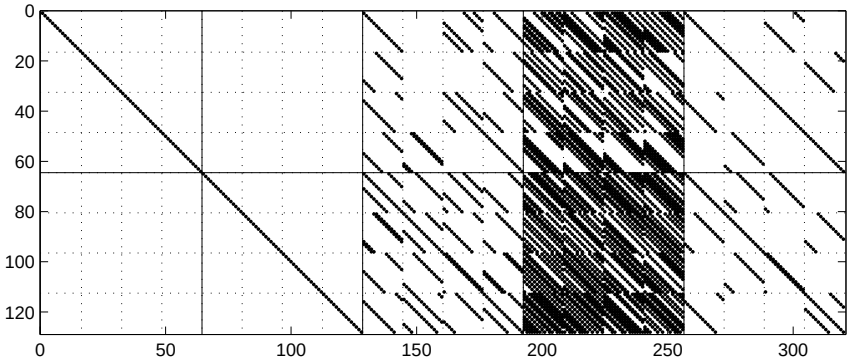


Fig. 4. A Systematic Block-Circulant Generator Matrix for the AR4A Code

"generator" matrix,

$$G = \begin{bmatrix} C_4 \\ C_2 \end{bmatrix}$$

of size $2T \times 5T$. This is one dimension short, and the missing codeword is $c = [\ 0 \quad 0 \quad 0 \quad p(x) \quad 0\ ]$ where $p(x) = 1 + x + x^2 + \cdots + x^{T-1}$. Note that if $c$ were expanded into circulants, the resulting $T \times nT$ matrix has rank 1. For implementation, it is most convenient to use $G$ as the generator matrix and discard this one additional dimension in the code, accepting the miniscule performance loss. The generator matrix $G$, corresponding to the parity check matrix of Figure 3, is shown in Figure 4. Because the last $T$ code symbols are punctured, the rightmost columns of circulants would be deleted from $G$ in implementation. By design, the first two columns of circulants form an identity matrix; the remaining circulants could have been dense by the construction algorithm, but the AR4A protograph structure assures that many remain sparse.

## IV. Hardware Implementation

The systematic block-circulant generator matrices developed in the previous section are particularly amenable to hardware implementation. A hardware encoder can pass the $kT$ message bits to the output as code symbols, while internally performing a multiplication by the (dense) $k \times (n-k)$ matrix in the right hand portion of $G$. The resulting vector serves as the remaining $(n-k)T$ code symbols. A direct implementation of this dense matrix multiplication is shown in Figure 5, as proposed by Lin [2]. The set of $n-k$ cyclic shift registers at the top of the figure, each of length $T$, are loaded with the circulant patterns for the first row of $G$. For each message bit $m_i$ in turn, these registers are cycled once and, if $m_i = 1$, exclusive-ORed with the $n-k$ symbol output register. When each row of circulants is completed, sequences for the next row of circulants in $G$ are loaded into the shift registers.

A further improvement in a hardware encoder is to cyclicly shift the output register, rather than the circulant registers, as shown in Figure 6. In this way, the circulant patterns need not be stored in registers at all, but can be generated as simple combinatorial functions of a symbol counter. This implementation is extremely similar to a set of $n-k$ encoders for recursive convolutional codes, each of constraint length $T$. With the switches set as drawn, the $k$ message bits are fed through the encoder one at a time, and the registers are updated and shifted once per bit. Then the switches are changed and the contents of the registers are sequentially read out as the parity portion of the codeword. This encoder has been implemented in hardware. It requires $n-k$ D-latches, $n-k$ exclusive-OR gates, and a modest amount of additional combinatorial logic. The size $(k = 1024, n = 2048)$ LDPC code fits comfortably in a Xilinx XC3S200 Spartan Field Programmable Gate Array (FPGA), and runs at 100 Msymbols/second. Speed is determined by the maximum clock rate of the FPGA. The maximum supported code size is determined primarily by the number of D-latches required to accumulate the parity, and so scales linearly with $n-k$.

## V. Conclusion

As many research groups have discovered in the last couple years, block-circulant LDPC codes have well structured decoders, and offer excellent error correction performance when designed carefully. Here, we have shown that they possess attractive encoders as well, of a couple different forms.

An iterative encoder is often possible for block-circulant LDPC codes, based on the standard erasure correction algorithm. Due to the circulant structure of the parity check matrix, the computational steps are typically sparse matrix multiplication by a circulant, permutation, and modulo-2 accumulation. The circulant matrix multiplications operate on long strings of sequential bits, so parallel computations are practical and permit fast encoders.

Encoders composed of linear feedback shift registers are another attractive alternative for block-circulant LDPC codes. These are based on the block-circulant generator matrices that these LDPC codes often possess. Such an encoder requires

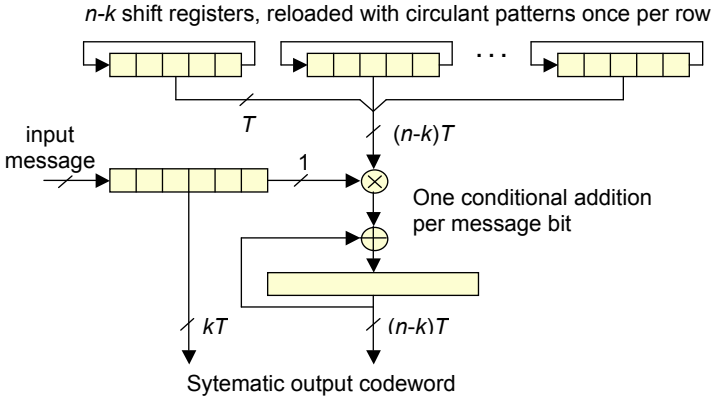


Fig. 5. The Direct Implementation of a Quasicyclic Encoder

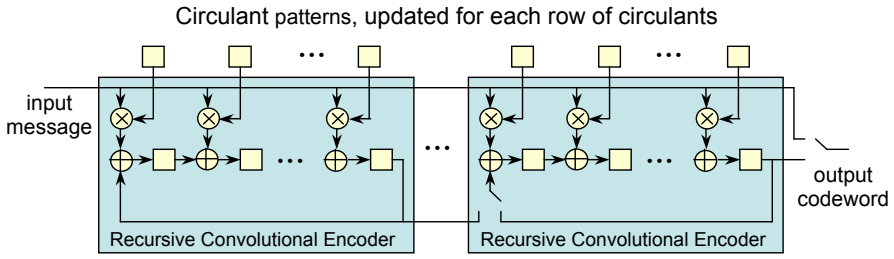


Fig. 6. A Quasicyclic Encoder Using Feedback Shift Registers

remarkably little hardware, and provides a fast, simple, bit-serial architecture. We have implemented these decoders in a small FPGA operating at 100 Msymbols/second.

## REFERENCES


[1] Y. Kou, H. Tang, S. Lin, and K. Abdel-Ghaffar, "On Circulant Low Density Parity Check Codes," in *IEEE International Symposium on Information Theory*, p. 200, June 2002.

[2] S. Lin, "Quasi-Cyclic LDPC Codes." CCSDS working group white paper, Oct. 2003.

[3] R. M. Tanner, "On Graph Constructions for LDPC Codes by Quasi-Cyclic Extension," in *Information, Coding and Mathematics* (M. Blaum, P. Farrell, and H. van Tilborg, eds.), pp. 209–220, Kluwer, June 2002.

[4] A. Sridharan, D. Costello, and R. M. Tanner, "A Construction for Low Density Parity Check Convolutional Codes Based on Quasi-Cyclic Block Codes," in *IEEE International Symposium on Information Theory*, p. 481, June 2002.

[5] O. Milenkovic, I. Djordjevic, and B. Vasic, "Block-Circulant Low-Density Parity-Check Codes for Optical Communication Systems," *IEEE Journal of Selected Topics in Quantum Electronics*, pp. 294–299, Mar. 2004.

[6] J. Thorpe, K. Andrews, and S. Dolinar, "Methodologies for Designing LDPC Codes Using Protographs and Circulants," in *IEEE International Symposium on Information Theory*, p. 238, June 2004.

[7] J. Thorpe, "Low-Density Parity-Check (LDPC) Codes Constructed from Protographs," IPN Progress Report 42-154, JPL, Aug. 2003.

[8] A. Abbasfar, D. Divsalar, and K. Yao, "Accumulate Repeat Accumulate Codes," in *IEEE International Symposium on Information Theory*, (Chicago, Illinois), June 2004.

[9] T. Richardson and R. Urbanke, "Efficient Encoding of Low-Density Parity-Check Codes," *IEEE Transactions on Information Theory*, pp. 638–656, Feb. 2001.