A New Class of Turbo-like Codes with Universally Good Performance and High-Speed Decoding

Keith M. Chugg[1,2], Phunsak Thiennviboon[2], Georgios D. Dimou[2], Paul Gray[2], and Jordan Melzer[1]

[1]Communication Sciences Institute
Electrical Engineering Dept.
University of Southern California
Los Angeles, CA, 90089-2565
{chugg,jmelzer}@usc.edu

[2]TrellisWare Technologies, Inc.
16516 Via Esprillo, Ste. 300
San Diego, CA 92127–1708
{phunsak,gdimou,pgray}
@trellisware.com

Abstract*— Modern turbo-like codes (TLCs), including concatenated convolutional codes and low density parity check (LDPC) codes, have been shown to approach the Shannon limit on the additive white Gaussian noise (AWGN) channel. Many design aspects remain relatively unexplored, however, including TLC design for maximum flexibility, very low error rate performance, and amenability to simple or very high-speed hardware codecs. In this paper we address these design issues by suggesting a new class of TLCs that we call Systematic with Serially Concatenated Parity (S-SCP) codes. One example member of this family is the Generalized (or Systematic) Repeat Accumulate code. We describe two other members of this family that both exhibit good performance over a wide range of block sizes, code rates, modulation, and target error probability. One of these provides error floor performance not previously demonstrated with any other TLC construction and the other is shown to offer very low complexity decoding with good performance. These two codes have been implemented in high-speed hardware codecs and performance curves based on these down to bit error rates below 10^{-10} are provided.

I. Introduction

The introduction of turbo codes [1] and the rediscovery of low density parity check (LDPC) codes [2], [3] has revolutionized coding theory and practice. Initially demonstrated for relatively low code rates, large block sizes, and binary modulation, these modern or Turbo-Like Codes (TLCs) have since been shown to be capable of near-optimal performance for nearly all practical operational scenarios. We use the term operational scenario as a given target block error rate (BLER) or bit error rate (BER), input block size (k), code rate (r), and modulation format.

While good TLCs have been demonstrated for most operational scenarios of interest, no one simple code structure has been shown to provide good performance over the entire range of practical interest. Also, some operational scenarios have been particularly challenging for TLC designers. An important example is the case of very high code rates ($r > 0.9$), moderate block sizes ($k < 4096$), and very low error probabilities (BER $< 10^{-10}$) which is of interest in data storage and high-speed fiber links.

In this paper we describe a new class of TLCs, called Systematic with Serially Concatenated Parity (S-SCP) codes, that provide good performance over a wide range of operational scenarios and are amenable to simple hardware implementation. These codes were not designed to be the best performing codes at a single operational scenario, but rather were designed for this universally good performance, flexibility, and ease of implementation. This is in contrast to much of the recent TLC code design literature which has focused on performance optimization for a given operational scenario. In particular, optimizing the asymptotic signal-to-noise (SNR) threshold has received a great deal of attention (*e.g.*, [4], [5], [6], [7]). However, such optimizations typically yield highly irregular code structures, which complicate implementation and rate-adaptation, especially for high-speed decoding.

Despite the fact that the S-SCP codes proposed have not been highly optimized for a particular operational scenario, the performance achieved is near that of the best point designs. Specifically, the best performance for a particular operational scenarios is readily estimated – providing a “theoretical bound” on achievable performance [8], [9]. The best point designs are typically within approximately 0.5 dB of SNR of this limit. The simple, flexible S-SCP codes presented typically perform within 0.3 dB of the best point design over a wide range of operational scenarios. In the case of high code rates, small block sizes, and low error probabilities, the new code presented provides the best performance known to the authors.

The S-SCP code construction shares some common traits with serially concatenated convolutional codes (SCCCs)[10], hybrid concatenated convolutional codes (HCCCs) [11], and the structured LDPC codes that we refer to as Generalized Repeat Accumulate (GRA) codes [12]. In fact, an effort has been made to combine the best traits of these constructions. The new TLC is a systematic code with parity bits generated using a serial concatenation of an outer code, an interleaver and an inner recursive parity generator as illustrated in Fig. 1(a). Note that a key feature of this code is that the inner parity generator outputs fewer bits than it takes in – this allows the code rate to be adjusted easily by adjusting this ratio.

In this paper we present two example S-SCP codes that were developed by TrellisWare Technologies in 2003 and are commercially available in various hardware codecs. One code, called the FlexiCode by TrellisWare, uses 4-state convolutional codes and provides exceptional error floor performance [13], [14]. Another version is the Flexible LDPC (F-LDPC) code [15] which provides SNR threshold performance similar to that of the FlexiCode, but exhibits error floors typical of well-designed LDPC codes (LDPCCs) – *i.e.*, higher floors than that of the FlexiCode, but still acceptable for many applications. The F-LDPC code uses two-state convolutional codes and can be interpreted as a punctured, structured LDPC code. Using TrellisWare’s high-speed hardware implementations of these codes allows us to present curves with BER down to 10^{-10}.

In Section II we describe the S-SCP family further, its relation to existing TLCs, and the special cases used for the TrellisWare codes. The decoding operations are summarized in Section III. A more detailed comparison of the decoding complexity of the F-LDPC and GRA codes are considered in Section IV. The

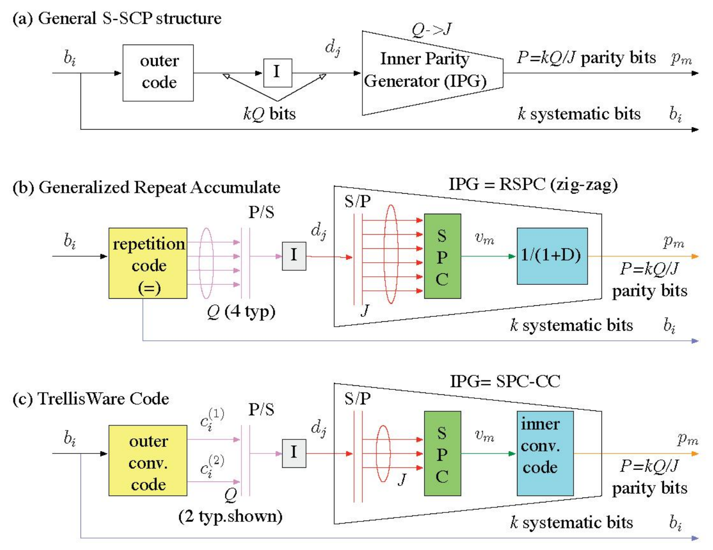

Fig. 1. The S-SCP code structure with two examples. The GRA code is shown for $J = 6$ and the TW code is shown for $J = 3$. The S/P and P/S are serial-parallel and parallel to serial converters, respectively.

code rate (r)	SPC size (J) for TW Codes	SPC size (J) for GRA
1/2	2	4
0.6	3	6
2/3	4	8
3/4	6	12
7/8	14	28
19/20	38	76

TABLE I
VARIOUS OVERALL CODE RATES FOR THE TW AND GRA CODES WITH OUTER CODE $r_o = 1/2$ AND $r_o = 1/4$ ($Q = 4$), RESPECTIVELY.

performance of the FlexiCode and F-LDPC codes is demonstrated in Section V. We provide uniform interleaver analysis of a subclass of S-SCP codes that includes the TrellisWare codes in the Appendix. In particular, the asymptotic interleaver gain properties of the TrellisWare codes are shown to be the same as that of SCCCs, providing justification of the excellent error floor performance.

II. S-SCP CODE STRUCTURE AND PROPERTIES

The S-SCP code structure is shown in Fig. 1(a). The k information bits $\{b_i\}$ are sent to the channel and parity generating concatenated system. The outer code is rate $r_o = 1/Q$ and generates kQ bits that are bit interleaved and input to the inner parity generator (IPG). The ratio of the number of input bits to output bits for the IPG is $r_i = J$. We are most interested in cases where $J > 1$ so that the IPG is not an error correction code since its mapping is not invertible. Note that the rate of the overall code is

$$r = \frac{k}{k + kQ/J} = \frac{J}{J+Q} \tag{1}$$

and this can be controlled for a given Q by varying the parameter J of the inner code. To summarize notation, there are k input bits, kQ interleaved bits, $P = kQ/J$ parity bits, and $n = k + P$ total codeword bits.

Two specific examples of S-SCP codes are the Generalized Repeat Accumulated (GRA) [12] and TrellisWare (TW) code structures shown in Figs. 1(b) and (c), respectively. The outer code for the GRA code is a repetition code, typically of rate 1/4 ($Q = 4$). The IPG for the GRA code is a concatenation of a single parity check (SPC) mapper and an accumulator. This combination was called a Zig-Zag code in [16] and may also be accurately termed a Recursive Single Parity Check (RSPC) generator. The TW code uses a convolutional code (CC), typically with rate $r_o = 1/2$. The output bits of the interleaver are then grouped into non-overlapping blocks of size J and their modulo 2 sum is formed and used as the input to drive a recursive rate one convolutional code. This SPC processing has the effect of making the IPG have a "rate" of $J \geq 1$. Note that for these typical values of Q and a given input block size and code rate, the GRA code will have an interleaver size and J value that are twice as large as those of the TW code. Table I provides representative code rates achieved with integer J for the GRA and TW codes.

One of the primary advantages of the S-SCP code structure is the ability to achieve excellent error floor performance, even at high code rates. This can be explained in terms of the interleaver gain properties that result from the uniform interleaver analysis methods developed in [17], [18], [10], [11]. These methods suggest that the BER (P_b) and BLER (P_{cw}) will vary asymptotically with block size as

$$P_b \sim N^{\alpha_{\max}} \qquad\qquad P_{cw} \sim N^{\alpha_{\max}+1} \tag{2}$$

where N is the interleaver size and $\alpha_{\max}$ is the maximum exponent of N in an asymptotic union bound approximation [17], [18], [10], [11]. If the BER (BLER) decays with N, then the code is said to achieve interleaver gain in BER (BLER). For the S-SCP structure with IPG comprising a SPC followed by a recursive CC, we show in the Appendix that

$$\alpha_{\max} = -\left\lfloor \frac{d_{\mathrm{o,min}} + 1}{2} \right\rfloor \tag{3}$$

where $d_{\mathrm{o,min}}$ is the minimum distance of the outer code, assumed to be at least 2. Note that if $d_{\mathrm{o,min}} \geq 3$, then $\alpha_{\max} \leq -2$ and the S-SCP code will achieve interleaver gain in both BER and BLER.

Note that SCCCs, known for their excellent error floor properties, have the same maximum exponent of N as given (3). Designing low complexity SCCCs with high code rate is difficult, however, since for a SCCC the overall code rate is the product of the inner and outer code rates. One can select a recursive, rate one inner code and the overall SCCC code rate is then equal to the outer code rate r_o. For code rates of $7/8$ and larger, it is difficult to achieve $d_{\mathrm{o,min}} \geq 3$ with a four state outer code. Outer convolutional codes with more than four states can be used at a complexity cost. This complexity can be alleviated to some degree by decoding this outer code over the graph of the dual code [19]. This approach, however, does not yield a single simple code structure that can be adapted easily over all rates and block sizes. To achieve reasonably good rate flexibility for SCCCs with 4-state constituent codes, one must use constituent codes with a large number of parallel state transitions and complex, rate-specific puncture patterns. The S-SCP code structure avoids this problem by using an IPG with rate above one. This enables the use of a lower outer code rate so that the minimum distance of 3 or larger can be achieved with low complexity.

The TW F-LDPC code uses a two state outer CC with generator polynomial $G_1(D) = G_2(D) = 1 + D$ and the same IPG as the GRA – *i.e.,* the RSPC generator. Note that this outer code and the $Q = 4$ repetition code both respond to a weight one input with a weight four output. In fact, the performance of the F-LDPC and GRA (with $Q = 4$) are nearly identical. The F-LDPC code and GRA codes, which may both be interpreted as structured LDPCCs, are compared in detail in Section IV. The F-LDPC code has good performance for all block sizes and for rates ranging from $r = 1/2$ to $19/20$. The flooring performance of the F-LDPC code is good – *e.g.,* for $k = 1024$ and $r = 7/8$, the F-LDPC and GRA codes will begin to hit an error flare at a BER of approximately 10^{-7}. This is impressive floor performance considering the low complexity of the constituent codes and would be very difficult to achieve with a SCCC based on 2-state codes. While this performance is acceptable for many applications, operational scenarios calling for lower error rates at these high code rates motivate the TW FlexiCode design, which uses 4-state convolutional codes for the outer code and IPG. Specifically, the outer code of the FlexiCode is non-recursive with generator selected as a mixture of $G_1(D) = 1$, $G_2(D) = 1 + D + D^2$, and $G_3(D) = 1 + D^2$ which is achieved by puncturing a the corresponding rate 1/3 code to rate 1/2 using puncture pattern of length 8 outer trellis stages (*i.e.,* 24 input to 16 output bits). The inner convolutional code is a rate 1 recursive code and uses the parity generator $(1 + D^2)/(1 + D + D^2)$ for overall rate 1/2, and $(1+D)/(1+D+D^2)$ for all other code rates. As will be demonstrated, using 4-state constituent codes improves the floor performance significantly relative to the F-LDPC and GRA codes. In the $k = 1024$, $r = 7/8$ example, the FlexiCode construction achieves a BER of 10^{-10} without significant flooring effects.

A. Other S-SCP Codes

Other S-SCP code designs are possible and may perform slightly better for specific operating scenarios. Noting that the role of the SPC is to adjust the rate of the IPG, a natural case to consider is using a puncture pattern on the outputs of a rate one recursive code instead of the SPC at its input. This is possible, and although the proof is not included for brevity, such codes have the same asymptotic interleaver gain as described above. For the case of the $1/(1 + D)$ inner convolutional code (*i.e.,* an accumulator), the SPC and puncturing are equivalent (see Section IV). For the TW FlexiCode case, the SPC and puncturing are different. Ping was the first to suggest using a SPC to alter rate in place of puncturing [20] and also noted that the potential for complexity reduction. Considering the FlexiCode, there are $2k$ inputs to the IPG, but only $P = 2k/J$ inputs passed from the SPC to the 4-state convolutional code. As will be described in Section III, this means that the iterative decoder need only run the forward-backward algorithm (FBA) (*e.g.,* [21], [22]) on $2k/J$ 4-state trellis sections and $2k$ 2-state trellis sections. If no SPC is used and the 4-state inner code is punctured, $2k$ 4-state trellis sections are processed. Thus, for $J > 2$ using a SPC in place of puncturing can substantially reduce decoder complexity. Furthermore, using the SPC instead of puncturing for the 4-state code typically provides better performance.

Another possible variation is to use irregular designs based on mixing different outer code polynomials. This is achieved to some degree in the FlexiCode design by selecting the outer puncture pattern. Further improvements in threshold can be achieved, however, by using an outer code with rate less than 1/2 based on mixtures of generating polynomials. For example, the F-LDPC code threshold performance can be improved with the following modification. Use an outer code with generators $G_1(D) = 1$ and $G_2(D) = 1 + D$; repeat the first output bit of this code Q_1 times and the second Q_2 times (the F-LDPC code uses $Q_1 = 0$ and $Q_2 = 2$). Time-varying values of Q_1 and Q_2 can be optimized, setting J to the corresponding value required to fix the overall rate, using standard methods of SNR threshold optimization [4], [5]. The best distribution on the values of Q_1 and Q_2 will change with code rate, however, and typically the resulting interleaver will have size larger than $2k$, which results in larger decoder implementation complexity. A similar construction, termed Accumulate Repeat Accumulate codes, have been developed for $r = 1/2$ in [23], [24] with $G_1(D) = 1$ and $G_2(D) = 1/(1 + D)$. Similar optimizations for the GRA code have been performed with similar advantages and drawbacks [12].

Finally, we note that the S-SCP code structure is similar to that of the Hybrid Concatenated Convolutional Codes (HCCCs) described in [11] with a parallel branch comprising just the systematic bits. The difference between the S-SCP code structure shown in Fig. 1 and the HCCCs discussed in [11] is that the former uses an inner parity generator that is not a code (*i.e.,* not invertible) while the HCCCs use an inner code. While this may seem to be a minor difference, as discussed above, this is the key property allowing the S-SCP codes to achieve high code rates and maintain interleaver gain which the HCCCs cannot achieve.

III. Iterative Decoding of the S-SCP Codes

The S-SCP codes are decoded iteratively using the standard rules of iterative decoding (*e.g.,* see [25], [26], [27], [22]). This may be viewed either as iterative message-passing on a graphical model of the code containing cycles or, equivalently, as a set of soft-in/soft-out (SISO) decoder modules updating and exchanging soft-decision messages. We focus on the TW code structure here and discuss some of the details of the decoding in this section.

The iterative decoder for the TW code structure of Fig. 1(c) is shown in Fig. 2(a). The iterative decoder accepts soft-decision metrics from the channel. We consider an additive white Gaussian noise (AWGN) channel with model,

$$z_t = \sqrt{E_c}(-1)^{x_t} + w_t, \qquad t = 0, 1, \ldots n \tag{4}$$

where n is the output block size, x_t is the sequence of coded bits (*i.e.,* $\{x_t\} = \{b_i\} \cup \{p_m\}$) and w_t is a realization of AWGN with variance $N_0/2$. Therefore the incoming channel messages (or metrics) for the channel bits are $\mathrm{MI}[x_t] = \frac{4\sqrt{E_c}}{N_0} z_t$, which are the bit-level negative log-likelihood ratios (NLLRs). The channel metrics corresponding to the systematic bits are input to the outer code SISO. The activation schedule for the iterative decoder in Fig. 2(a) is: outer-SISO, interleaver, SPC-SISO inward,

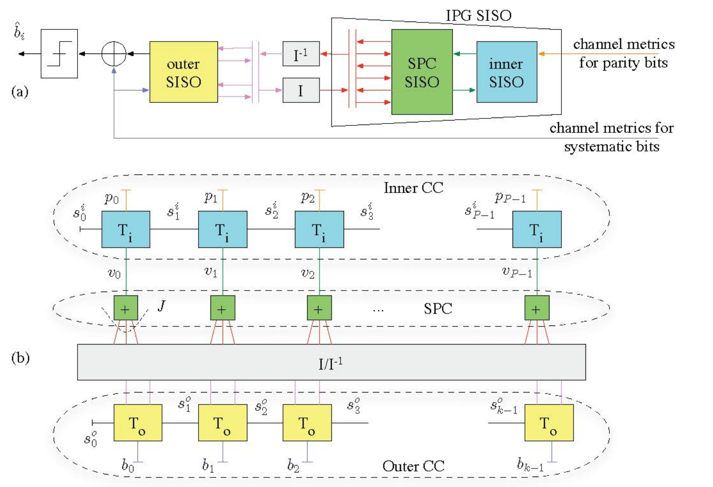

Fig. 2. The iterative decoder for the TW code structure of Fig. 1(c) is shown in (a) with the corresponding graphical model in (b) (shown for $J = 3$). The constraints marked with $+$ are even parity constraints; those marked with T$_\text{o}$ and T$_\text{i}$ are the outer and inner CC trellis section constraints, respectively.

inner SISO, SPC-SISO outward, and deinterleaver; which defines one iteration. The inner and outer SISOs can be based on the standard forward-backward algorithm (FBA) (*e.g.*, [21], [22]). The SPC-SISO is discussed in detail below. At each activation of the outer SISO, hard decisions on the information bits can be obtained by adding the systematic channel metrics and the extrinsic messages on b_i provided by the outer SISO – *i.e.*, $\hat{b}_i = 1$ is $\mathrm{MI}[b_i] + \mathrm{MO}[b_i] < 0$ and $\hat{b}_i = 0$, otherwise.

The iterative decoding described above is equivalent to running the standard message-passing rules on the graphical model of the TW code structure shown in Fig. 2(b).[1] The message activation schedule is left and right on the bottom subgraph (outer SISO FBA), up through the interleaver permutation, up through the SPC nodes (SPC-SISO inward), left and right on the top subgraph (inner SISO FBA), downward through the SPC nodes (SPC-SISO outward), and through the deinterleaver permutation.

When processing messages in the negative-log (metric) domain as described above, one can use either min-sum processing or min*-sum processing where $\min^*(x, y) = \min(x, y) - \ln[1 + \exp(-|x-y|]$ (*e.g.*, [22]). The SPC-SISO processing is as follows. Consider one SPC block that enforces the constraint

$$a_0 + a_1 + a_2 \cdots + a_J = 0 \tag{5}$$

and takes in inputs messages for each[2] denoted by $\mathrm{MI}[a_i]$ for $i = 0, 1, \ldots J$. The outgoing messages are

$$\mathrm{MO}[a_i] = g(\mathrm{MI}[a_0], \ldots \mathrm{MI}[a_{i-1}], \mathrm{MI}[a_{i+1}], \ldots \mathrm{MI}[a_J]) \tag{6}$$

[1]This convention for graphical models is similar to the normal graphs described in [28] and the explicit index diagrams in [22]. In particular, variables are represented as edges and constraints as vertices. Equality constraints are shown as circles and the message updates are the same as the "variable nodes" in other graphical modeling conventions.

[2]All messages for binary variables are assumed to be in normalized or NLLR form. Specifically, a positive (negative) metric indicates a belief that the variable is a 0 (1) and larger magnitude indicates higher confidence in this belief.

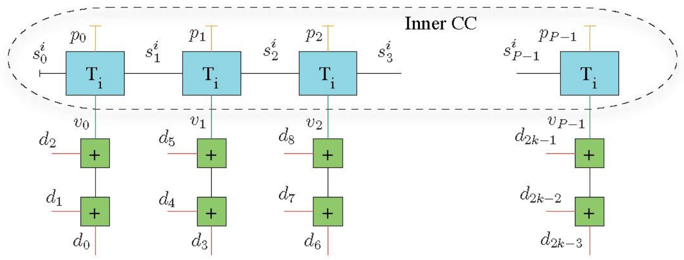

Fig. 3. An alternative graphical model for the IPG of Fig. 2(b) that illustrates how the SPC-SISO processing is equivalent to the FBA running on short accumulator trellises (shown for $J = 3$).

and the function $g(x, y)$ is defined by

$$g(x, y) = \min(x, y) - \min(0, x + y) \tag{7a}$$

$$= \min(|x|, |y|)\mathrm{sign}(x)\mathrm{sign}(y) \qquad \text{(min-sum)} \tag{7b}$$

$$g(x, y) = \min{}^*(x, y) - \min{}^*(0, x + y) \tag{7c}$$

$$\begin{aligned} &= \min(|x|, |y|)\mathrm{sign}(x)\mathrm{sign}(y) \\ &\quad - \ln\left[\frac{1 + \exp(-|x - y|)}{1 + \exp(-|x + y|)}\right] \qquad \text{(min}^*\text{-sum)} \end{aligned} \tag{7d}$$

and in both cases $g(.)$ can be computed for multiple arguments using $g(x, y, z) = g(g(x, y), z)$. Thus, the inward SPC-SISO processing is computing $g(\cdot)$ of for blocks of J messages read from interleaver to produce the message on v_m. The outward SPC-SISO processing task is to compute $g(\cdot)$ for the message on v_m produced by the inner SISO and the $J - 1$ messages for every subset of size $J - 1$ of the d_j variables involved in the SPC block. This SPC processing can also be interpreted using that fact that a SPC constraint is equivalent to an accumulator encoder started and terminated in the zero state. For example, consider the graph in Fig. 3 that is an alternative representation of the IPG shown in Fig. 2(b). This may be viewed as following directly from the encoding constraint

$$v_m = d_{mJ} + d_{mJ+1} + d_{mJ+2} \cdots + d_{mJ+J-1} \tag{8}$$

and the fact that v_m is the input to the inner convolutional code. With this model, it is clear that the SPC processing can be viewed as running an FBA-based SISO on an accumulator trellis. In particular, messages are passed upward along the SPC model in Fig. 3 (the forward recursion of the FBA) and the final state metric for each SPC block becomes part of the trellis transition metrics for the inner code SISO. Then output messages from the inner code SISO are used to initialize the backward recursion of the FBA for each SPC block. Note that this interpretation makes it clear that the SISO processing corresponding to the IPG is equivalent to running the FBA on $2k$ accumulator trellis sections and $P = 2k/J$ inner code trellis sections.

IV. Comparison of the F-LDPC and GRA Codes

For a more detailed comparison of the F-LDPC and GRA codes, we first show how each may be decoded using the same basic operations. We then show that these may be viewed as structured LDPCCs to compare their decoding complexity with that of other LDPC code constructions. First, the graphical models

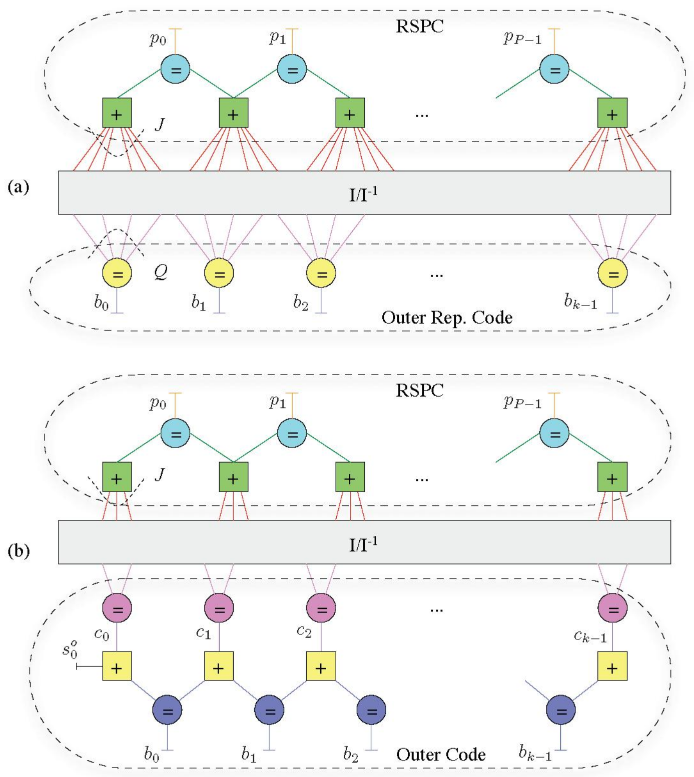

Fig. 4. Graphical models for (a) the GRA code and (b) the F-LDPC code.

for the GRA and F-LDPC codes are illustrated in Fig. 4. Note that the graphical model of the F-LDPC code is a special case of that in Fig. 2(b) which accounts for the specific code generators. Specifically, since

$$c_i = c_i^{(1)} = c_i^{(2)} = b_i + b_{i-1} \tag{9}$$

the state of the outer code is a visible variable – *i.e.,* $s_i^o = b_{i-1}$. Similarly, the state of the accumulator is $s_m^i = p_{m-1}$ which is also a visible variable. The RSPC encoding constraint is

$$p_m = p_{m-1} + v_m = p_{m-1} + d_{mJ} + d_{mJ+1} \cdots + d_{mJ+J-1} \tag{10}$$

The relation in (10) implies that a RSPC generator is equivalent to regular $(J-1)$ of J puncturing at the output of a standard accumulator (*i.e.,* with no SPC pre-processing). This also follows from inspection of Fig. 3 when the inner convolutional code is an accumulator. As a result, the IPG-SISO can be implemented by running a single accumulator-SISO over one long block of size kQ.

The FBA-based SISO processing for an accumulator is very simple to describe in terms of additions and the g-functions in (7). Consider an accumulator with input bits a_j and output bits x_j – *i.e.,* $x_j = x_{j-1} + a_j$ for $j = 0, \ldots P-1$. Denote the input messages for these variables as $\mathrm{MI}[a_j]$ and $\mathrm{MI}[x_j]$, respectively. The FBA message updates can be shown to be

$$\mathrm{F}_{j+1} = g(\mathrm{F}_j, \mathrm{MI}[a_j]) + \mathrm{MI}[x_j] \tag{11a}$$
$$\mathrm{B}_j = g(\mathrm{B}_{j+1} + \mathrm{MI}[x_j], \mathrm{MI}[a_j]) \tag{11b}$$
$$\mathrm{MO}[a_j] = g(\mathrm{F}_j, \mathrm{B}_{j+1} + \mathrm{MI}[x_j]) \tag{11c}$$
$$\mathrm{MO}[x_j] = \mathrm{F}_{j+1} + \mathrm{B}_{j+1} - \mathrm{MI}[x_j] \tag{11d}$$

where F_j and B_j are the forward and backward state metrics for state $s_j = x_{j-1}$, and $\mathrm{MO}[\cdot]$ denotes the outgoing messages. The values of F_0 and B_P are initialized according to the accumulator initialization/termination information available.

It is notable that each trellis stage in an accumulator trellis requires 3 $g(\cdot)$ operations and 4 add operations (*i.e.,* the sum in (11b) and (11c) are the same). If there is no input soft information on x_j ($\mathrm{MI}[x_j] = 0$ for all j) and no output information on x_j is to be computed, then the number of $g(\cdot)$ operations is still 3, but all of the add operations are eliminated. This corresponds to the processing in (6) which is also the check node processing in an LDPC decoder. If there is soft input information on x_j, but no corresponding soft output information is required, then the number of add operations per trellis section is reduced from 4 to 2.

We now describe how the accumulator SISO processing in (11) can be used to implement the GRA and F-LDPC decoding. First, consider the RSPC generator that comprises the SPC and accumulator used in both codes. Using (10) we can establish the correspondence of d_j to a_j in (11) and p_m to x_{mJ} with all other x_j variables punctured. It follows that the $\mathrm{MI}[a_j]$ in (11) are the messages on d_j read from the interleaver, and $\mathrm{MI}[x_j]$ is set to the channel metric for p_m when $j = mJ$ and zero for other values of j. Note that $\mathrm{MO}[x_j]$ is not required.

Next consider the outer code SISO for the F-LDPC. Since the $g(D) = 1 + D$ encoder is the inverse mapping of the accumulator, the corresponding SISO operation can be implemented with the accumulator SISO with the roles of input and output variables reversed. In particular, it is straightforward to show that associating a_i and x_i from (11) with c_i and b_i from (9), the corresponding $g(D) = 1 + D$ FBA-SISO is implemented. Note that prior to running this FBA-based SISO, the metrics for c_i must be computed as the sum of the messages for $c_i^{(1)}$ and $c_i^{(2)}$ coming from the deinterleaver. Similarly, computing the updated messages for $c_i^{(1)}$ and $c_i^{(2)}$ to be passed to the IPG-SISO requires more additions corresponding to the equality constraint in Fig. 4(b).

The outer SISO processing for the GRA decoder corresponds to activating each equality constraint in Fig. 4(a). For $Q = 4$, this requires 8 adds per node using the standard equality constraint (variable) node processing.

In summary, the F-LDPC decoder must process k trellis sections for the outer SISO and $2k$ sections for the IPG, or $3k$ total accumulator-equivalent trellis sections per iteration. The GRA decoder must process kQ trellis sections (all for the IPG). Since, for similar performance, $Q = 4$ is required, the GRA decoder performs approximately 33% more $g(\cdot)$ operations than the corresponding F-LDPC decoder. The number of additions required for each GRA decoder iteration is $8k$ for the outer SISO processing and $2P$ for the IPG. The F-LDPC decoder requires

$6k$ adds for the outer SISO and $2P$ adds for the IPG SISO processing. Thus, the GRA decoder requires $2k$ more adds per iteration, which is 25% more than the F-LDPC decoder for rates of $1/2$ and above.

A. High-Level Hardware Architecture Discussion

These high-level computation measures do translate into hardware complexity, but the F-LDPC code also compares favorably to the GRA code for other factors affecting implementation. To consider these, let us characterize a hardware decoder implementation in terms of the rate of decoded (information) bits per logic clock frequency (bps/logic-Hz). If no significant parallel processing is done, then a trellis stage can be processed every clock cycle. For the F-LDPC, this means that each iteration will take approximately $3k$ clock cycles. If I iterations are run, the throughput is roughly $1/(3I)$ bps/logic-Hz. For example, for $I = 10$ iterations, a throughput of $1/30$ bps/logic-Hz can be obtained implying that a design with a 100 MHz clock will provide roughly 3.3 Mbps. The corresponding GRA decoder ($Q = 4$) will achieve a throughput of about $1/(4I)$ bps/Hz, or 2.5 Mbps in the same example scenario. In both of these low-speed decoders, however, the memory circuitry will dominate hardware area for even moderate block sizes (*e.g.*, $k = 2048$). A segment-based architecture [29], [30], [31] for the FBA-based SISO implementation reduces the memory requirements of the forward and backward state metrics to a negligible amount and memory is dominated by storing the channel metrics and the messages passed through the interleaver/deinterleaver. Since the interleave size is $2k$ and $4k$ for the F-LDPC and GRA codes, respectively, the memory savings are substantial. Assuming the lowest rate supported is $r = 1/2$, there are $2k$ channels messages and one such block should be buffered during decoding. This means that there are $4k$ channel metrics to store so that a total of roughly $8k = 4k + 4k$ metrics are to be stored for the GRA decoder while the F-LDPC decoder must store roughly $6k = 2k + 4k$ metrics. Thus, for rates of 1/2 and above, the GRA decoder would require approximately 33% more memory resources, with this figure approaching 50% as the code rate increases.

For a high-speed decoder, parallel processing architectures must be employed to improve throughput. If D degrees of parallelism are used, then the throughput in bps/logic-Hz will increase roughly by this same proportion with the F-LDPC design maintaining the same proportional advantage. As an example of the state of the art, the TW F-LDPC has been implemented in a Xilinx XC2V8000 FPGA and achieves 300 Mbps with a 100 MHz clock, or 3 bps/logic-Hz, with 10 iterations. As D becomes large to achieve such speeds, memory design and routing becomes a primary challenge. Qualitatively, regularly structured code designs simplify this task.

As an example of the effects of irregular designs for SNR threshold optimization, consider an optimized $r = 1/2$ irregular GRA from [12, Table 3]. This code has a mixture of outer repetition code Q values from $\{3, 11, 12, 46, 48\}$ with an average of $\overline{Q} = 19.75$. This provides an improvement of approximately 0.5 dB in SNR threshold as the block size tends toward infinity. For a block size of $k = 1000$, the gain is approximately 0.25 dB. This comes at a cost of approximately a factor of $19.74/4 \approx 5$ in memory, computational complexity, and throughput relative to the $Q = 4$ GRA decoder. Relative to the F-LDPC decoder, this is a factor $19.74/3 \approx 6.6$. Furthermore, the irregular repetition pattern complicates memory access for high-speed implementations and will vary for different code rates. This motivates our approach of designing a simple, regularly structured code that achieves good performance and is amenable to simple/fast hardware implementation.

B. GRA and F-LDPC Codes as LDPCCs

We conclude this section by demonstrating that both the GRA and F-LDPC codes can be interpreted as LDPCCs. In fact, virtually any code that is constructed using only binary variables, equality constraints, SPC constraints, and permutations can be viewed as a LDPCC. The GRA code is equivalent to the semi-random LDPC in [32], the linear-time encodeable LDPC in [33], and the stair-case structured LDPC in [34]. With a block diagonal constraint on the interleaver, this is also the parallel concatenated zig-zag code in [16]. To see this note that (10) implies that

$$[\,\mathbf{S} \mid \mathbf{H}_b\,] \begin{pmatrix} \mathbf{p} \\ \mathbf{b} \end{pmatrix} = \mathbf{0} \tag{12}$$

$$\mathbf{S} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & & 0 \\ 0 & 1 & 1 & & 0 \\ \vdots & & & \ddots & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \quad (P \times P) \tag{13}$$

and $\mathbf{H}_b$ is a $(k \times k)$ matrix with column weight Q and row weight J. The vectors $\mathbf{p}$ $(P \times 1)$ and $\mathbf{b}$ $(k \times 1)$ are vectors of the parity and systematic code bits, respectively. This can be seen graphically from Fig. 4(a) by pulling all variable (equality constraint) nodes to the bottom of the graph.

A similar approach can be used to show that the F-LDPC code is an LDPC with some variables punctured (or hidden) from the channel. Specifically, the graph of Fig. 4(b) can be arranged so that all variable (equality constraint) nodes are at the bottom of the graph and all check nodes (SPC constraints) are at the top. Note that the variable nodes are either b_i, c_i, or p_m. Of these, only c_i are not visible to the channel. It is straightforward to write a check equation of the form

$$\begin{bmatrix} \mathbf{S} & \mathbf{V} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{G} \end{bmatrix} \begin{pmatrix} \mathbf{p} \\ \mathbf{c} \\ \mathbf{b} \end{pmatrix} = \mathbf{0} \tag{14}$$

where $\mathbf{c}$ is the $(k \times 1)$ vector of bits c_i, $\mathbf{S}$ is the dual-diagonal matrix in (13), $\mathbf{G}$ is $(k \times k)$ and also dual-diagonal, and the matrix $\mathbf{V}$ accounts for the repetition by 2 at the outer code, interleaving and the SPC (see [15] for details).

Thus, both the GRA code and the F-LDPC code could be decoded using the variable-check node flooding schedule usually assumed in the LDPC literature. This is not desirable, however, for either hardware of software implementation. For software, this converges at a much slower rate than the decoder schedule previously described. For hardware, this flooding schedule is impractical for memory design and routing. Expressing the codes in this form does allow for high-level comparisons with LDPCCs in the literature. In particular, each check message update is equivalent to running the accumulator FBA-SISO over a number of trellis sections equal to the degree of the check. Thus, the number of accumulator trellis sections processed by for each iteration is roughly the number of edges in the LDPC code Tanner graph. For a regular (3,6) $r = 1/2$ LDPC code, this is $6k$ trellis sections. For highly optimized irregular LDPCCs, this is considerably higher – *e.g.,* the LDPC code with best SNR threshold in [7, Table 1] has approximately $13k$ trellis sections to process per iteration. In either case, the GRA code and, in particular, the F-LDPC code compare favorably in terms of complexity – *e.g.,* the complexity of the regular and irregular LDPC decoders are approximately 100% and 433% greater than that of the F-LDPC decoder for each iteration.

V. Performance Characteristics

In this section we first briefly describe the theoretical limits for modulation-constrained, finite block size, finite BLER performance that serves as a guideline for good modern code performance. We then present representative performance results for TW's FlexiCode and F-LDPC codes. Further performance results for these codes are available in [14] and [15], respectively, and available upon request.

A. Theoretical Guideline for Good TLC Performance

Channel capacity is the ultimate measure of achievable performance. For the modulation constrained, AWGN channel, capacity is often well approximated by the upper bound of the mutual information rate with a uniform a-priori distribution over the signal set. This is the so-called Symmetric Information Rate (SIR) and is readily computed as a function of the SNR [35], [9]. For a given rate (spectral efficiency) this implies a minimum required value of E_s/N_0 to operate, where E_s is the energy per modulation symbol. This value, denoted by $\left(\frac{E_s}{N_0}\right)_{\text{min,SIR}}$, can be interpreted as the minimum value of E_s/N_0 required to achieve zero BLER with infinite block size at the specified rate and modulation.

To account for the finite block size and non-zero error rate, either the random coding bound (RCB) or the sphere-packing bound (SPB) for the constellation constrained AWGN channel can be considered. These are upper and lower bounds on the performance of the best finite block-length signaling scheme, respectively. The RCB can be computed for the constellation constrained channel with some effort [9], but computation of the SPB is difficult. A SNR penalty can be applied to the minimum SNR implied by capacity to approximate the minimum E_s/N_0 required by the SPB for the modulation unconstrained AWGN channel [8]. It has also been be found empirically that applying this same penalty term to $(E_s/N_0)_{\text{min,SIR}}$ for the modulation constrained channel yields a finite block-size performance bound in close agreement with the RCB over a wide range of operational scenarios [9]. Thus, a good performance guideline for *finite* block-size, *finite* BLER, modulation constrained channels is

$$\left(\frac{E_s}{N_0}\right)_{\text{min,finite,dB}} = \left(\frac{E_s}{N_0}\right)_{\text{min,SIR,dB}} + \Delta_{\text{dB}} \tag{15a}$$

$$\Delta_{\text{dB}} = \sqrt{\frac{20\eta\,(2^\eta+1)\,[10\log_{10}(1/P_{\text{cw}})]}{k\ln(10)\,(2^\eta-1)}} \tag{15b}$$

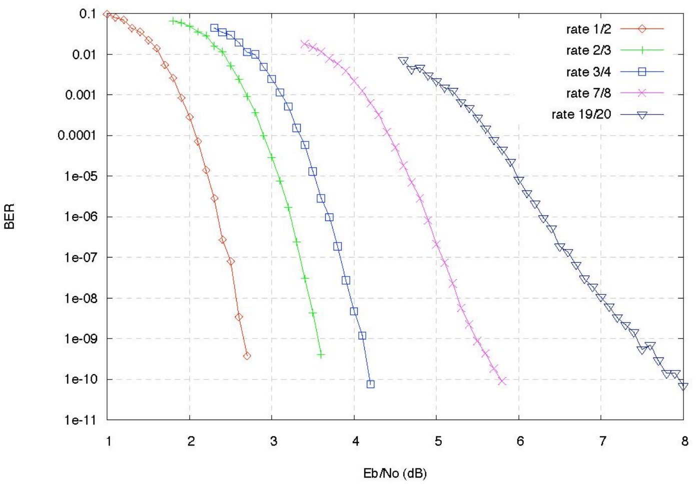

Fig. 5. TW FlexiCode ASIC performance for $k = 1024$ and various rates.

where k is the input block size, η is the operating spectral efficiency in bits per 2-dimensional channel use, and P_{cw} is the operating BLER.

A survey of the literature reveals that the best TLCs designed for a single operational scenario operate within 0.5 dB of E_s/N_0 of this guideline. The TW codes operate approximately 1 dB or closer to this guideline over a large range of operational scenarios as illustrated in the following.

B. FlexiCode Performance Examples

Performance for the FlexiCode is based on measurements performed on the FlexiCode ASIC codec. This code supports input block sizes from 128 bits to 16384 bits in 32 bit increments, eight code rates from rate 1/2 to rate 19/20 of the form $J/(J+2)$ for integer J, and provides internal support for the following modulation schemes: binary phase shift keying (BPSK), quadrature phase shift keying (QPSK), eight phase shift keying (8PSK), sixteen quadrature amplitude modulation (16QAM), and sixteen amplitude-phase shift keying (16APSK).

Performance of the FlexiCode ASIC is shown in Figs. 5 and 6 for $k = 1024$ and $k = 4096$, respectively. The ability of this S-SCP code construction to achieve very low BER performance, even with small block size and high code rate is unique. These results were generated using a 125 MHz logic clock and and a throughput of 54 Mbps (information bits) or 0.432 bps/logic-Hz. Due to pipeline overhead, the number of iterations performed is a function of the block size and, to a lesser extent, the code rate. For $k = 1024$, this corresponds to 17, 16, 16, 15, and 12 iterations for rates 1/2, 2/3, 3/4, 7/8, and 19/20. The same values for $k = 4096$ are 28, 28, 28, 26, and 26. For $k = 16384$, all rates can provide 31 iterations at this throughput. The decoding converges more quickly for higher code rates, so even the $k = 1024$ results contain most attainable iteration gain, with the rate 1/2 case still having a potential for an additional 0.1 dB gain with further iteration. Decreasing the number of iterations proportionally to achieve a throughput of 155 Mbps at the same clock rate (*i.e.,* 1.25 bps/log-Hz) causes approximately 0.7 dB in performance degradation for $k = 1024$ and approximately 0.25 dB degradation for $k = 4096$.

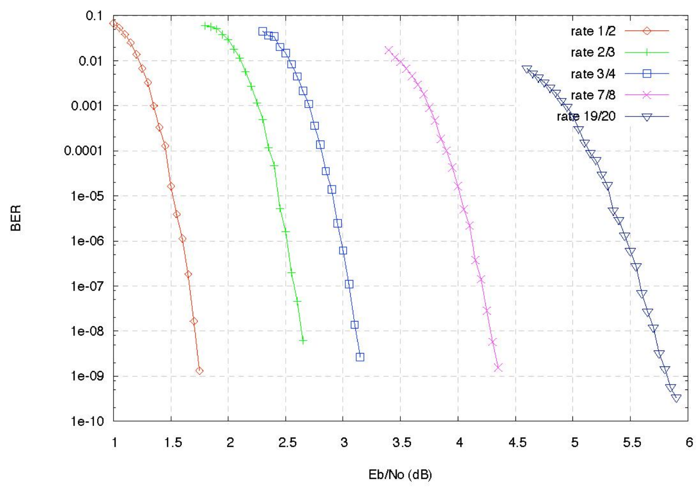

Fig. 6. TW FlexiCode ASIC performance for $k = 4096$ and various rates.

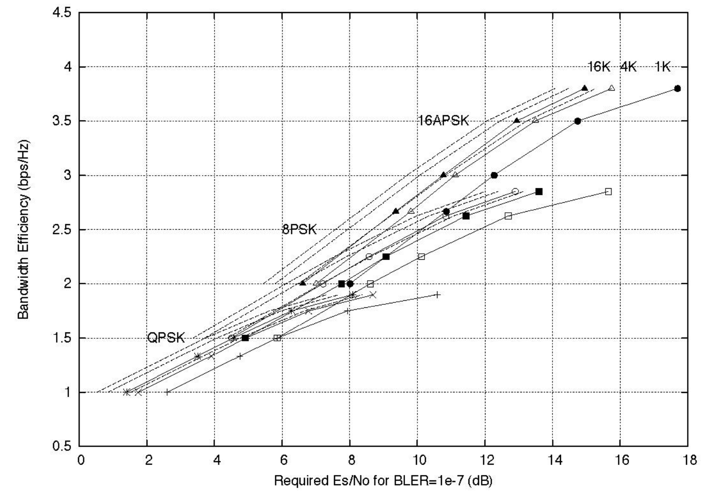

Fig. 7. TW FlexiCode ASIC performance for various block size (k), rates and modulation and BLER of 10^{-7}.

Fig. 7 compares the FlexiCode ASIC performance at 54 Mbps to the theoretical (approximate) bound given in (15) for k of 16384, 4096, and 1024. Note that over a wide range of rates and modulations, the performance is within 0.5 to 1 dB of the finite block size bound for $k = 16384$. This is impressive when considering that the best point design for an one of these results is typically 0.5 dB from the bound and that these results include all hardware implementation effects. Also, note that the BLER of 10^{-7} is quite low in comparison to many results in the literature. The performance of the smaller block sizes is further than 1 dB from the theoretical guideline. For these smaller block sizes, code rates, and very low BLER target, these are the best results known to the authors.[3]

C. *F-LDPC Code Performance Examples*

As mentioned previously, the F-LDPC code performs similarly to the FlexiCode without the extraordinarily low error floors. This performance can be well-approximated by computing the bound in (15) and adding a 1 dB margin. To illustrate this we consider a system with $k = 8000$ and a target BLER of 0.01. Further refinement in the F-LDPC code rate can be achieved by further puncturing the parity bits. We consider a regular puncture pattern of length 16 and the fraction of parity bits maintained from each block of 16 parity bits is $q = 16/16, 15/16, \ldots 8/16$. This yields an overall rate of $r = J/(J = 2q)$ and provides 45 code rates from rate 1/2 to 32/33. The coded bits (systematic and parity) are shuffled via a simple relative prime permutation and Gray-mapped onto quadrature amplitude modulation constellations with size $M = 2$, 4, 16, 64, and 256. In the decoding process, the noisy matched-filter samples are processed to provide the F-LDPC decoder with binary NLLRs and no further modulation processing is performed. The results are shown in Fig. 8 where each point represents a simulation result for a different rate and the theoretical performance limits are also shown. The F-LDPC code obviously provides sufficient rate flexibility with good performance at this block size. The TW hardware cores for the F-LDPC code have similar block size flexibility to that of the FlexiCode ASIC and provide qualitatively similar performance to that shown in Fig. 8 for other block sizes.

[3]It is difficult to obtain such results without a high-speed hardware codec.

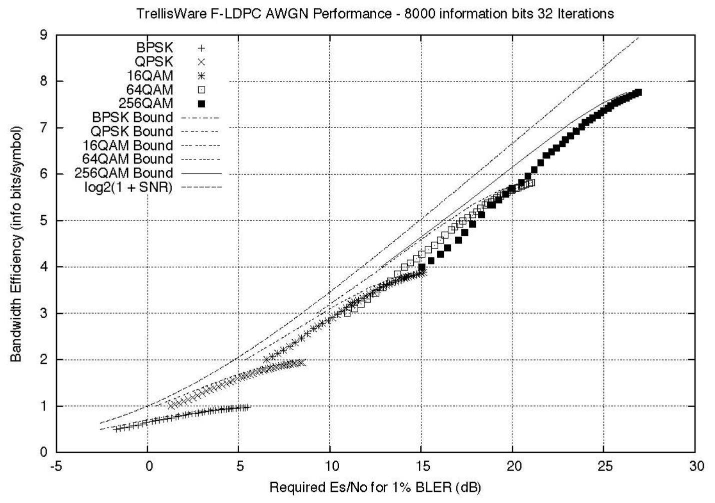

Fig. 8. TW F-LDPC performance (software simulation) for $k = 8000$, BLER$= 10^{-2}$ and various modulations as compared to the theoretical bounds (including the modulation-unconstrained AWGN capacity).

VI. Conclusion

The Systematic with Serial Concatenated Parity code construction described has several good practical and theoretical properties. This family of TLCs which includes the Generalized (or Systematic) Repeat Accumulate code, and two example constructions that have been built in hardware to target various points on the speed-complexity trade-off. The TW FlexiCode provides unique error floor performance despite being based on simple four state convolutional codes. A key idea of this construction is the use of an inner parity generator that is the combination of a single parity check sum, followed by a rate one recursive convolutional code. We have shown that such constructions offer the same desirable asymptotic interleaver gain properties as

Serially Concatenated Convolutional Codes. The F-LDPC code is based on the same construction with two-state convolutional codes. It was demonstrated how this simple and flexible code has significant advantages in terms of complexity relative to the GRA code, which itself is an excellent, flexible modern code design. Specifically, the F-LDPC code provides performance within approximately 0.3 dB of the best known irregular LDPCC designs at the block rates and number of iterations considered for practical systems. This is achieved with complexity that is roughly 5 times less than that of these highly optimized point designs. We also described how one can further optimize these codes for particular operational scenario by decreasing the outer code rate and adding irregularity to the code constraints. Such optimization is expected to offer similar performance advantages (*i.e.,* 0.3 dB) at the expense of additional complexity.

VII. Appendix: Design Rules for S-SCP Codes

Uniform interleaver analysis, as developed in [17], [18], [10], [11] is applied to the TW code construction in Fig. 1(c) – *i.e.,* a S-SCP code with IPG comprising a SPC and CC combination. For a fixed length this can be viewed as a block code, and the union bound on the BER is

$$P_b \leq \sum_{h=h_{\min}}^{N} \sum_{w=1}^{k} \frac{w}{k} A_{w,h}^{C_C} Q\left(\sqrt{2r(w+h)\frac{E_b}{N_0}}\right) \qquad (16)$$

where N is the interleaver size and $A_{w,h}^{C_C}$ is the input-output weight coefficent (IOWC) of the parity branch of the of the S-SCP code. Note that $h_{\min}$ is the minimum output weight of the parity branch only.

The IOWC is dependent on the exact encoders and interleavers used, but asymptotic trends can be determined using the abstract *uniform interleaver* which maps an input word of weight w into all its distinct $\binom{N}{w}$ permutations with probability $1/\binom{N}{w}$, where $\binom{N}{w}$ is the binomial coefficient: $\binom{N}{w} = \frac{N!}{(N-w)!w!}$.

Introducing s as the output weight of the SPC (over all SPC blocks), l as the output weight of the outer convolutional code, and $p_{\rm spc}(s|l)$ as the probability of the output weight of the SPC being s when the input weight is l yields

$$A_{w,h}^{C_C} = \sum_{l=0}^{N} \sum_{s=0}^{l} p_{\rm spc}(s|l) \frac{A_{w,l}^{C_o} \times A_{s,h}^{C_i}}{\binom{N/J}{s}} \qquad (17)$$

where $A_{w,l}^{C_o}$ and $A_{s,h}^{C_i}$ are the IOWCs for the outer and inner CCs, respectively.

The unique step in this uniform interleaver analysis for the TW codes is the consideration of the impact of the SPC (*i.e.,* the rest is similar to the analysis of HCCCs in [11]). Define the reduction in weight caused by the SPC as $\delta = l - s$, and note that, since the output weight is reduced when pairs of input ones fall into a parity check for the same output bit, δ can only take even values. The number of SPC blocks into which one or more the l input ones can fall is at most $s + \frac{\delta}{2}$, this count coming from the case where s parity checks contain a single one and $\frac{\delta}{2}$ contain two ones. The probability that all l inputs fall within a set of $s + \frac{\delta}{2}$ SPC blocks of the N/J SPC blocks times the number of ways to choose those $s + \frac{\delta}{2}$ parity checks from the N/J total parity checks is then an upper bound on $p_{\rm spc}(s|l)$, as it also includes all output weights that are smaller than s. This provides the upper bound

$$\begin{aligned} p_{\rm spc}(s|l) &\leq \left(\frac{s+\frac{\delta}{2}}{N/J}\right)^l \binom{N/J}{s+\frac{\delta}{2}} \qquad (18)\\ &= \left(\frac{s+\frac{l-s}{2}}{N/J}\right)^l \binom{N/J}{s+\frac{l-s}{2}} \\ &\tilde{\leq} (N/J)^{-\frac{l-s}{2}} \frac{(s+\frac{l-s}{2})^l}{(s+\frac{l-s}{2})!} \\ &\propto (N/J)^{-\frac{l-s}{2}} \\ &\propto N^{-\frac{l-s}{2}} \end{aligned}$$

which can be used in (16) to obtain a looser upper bound. Note that a large N approximation for the binomial coefficient has been used.

Computation of the asymptotic IOWCs for the inner and outer convolutional code in (16) follows the standard development in [17], [18], [10], [11] and is omitted for brevity. Substituting these asymptotic approximations for the CC IOWCs and the bound in (18) into (16) yields

$$P_b \tilde{\leq} \sum_{h=h_{\min}^i}^{Nr_o} \sum_{w=w_{\min}}^{N/J} \sum_{l=0}^{N} \sum_{s=0}^{l} \sum_{n^o=1}^{n^o_{\max}} \sum_{n^i=1}^{n^i_{\max}} N^{n^o+n^i-\frac{l+s}{2}-1} B_{w,l,n^o,n^i,h,s} Q\left(\sqrt{2r(w+h)\frac{E_b}{N_0}}\right) \qquad (19)$$

where $B_{w,l,n^o,n^i,h,s}$ is not a function of the interleaver size N. The number of simple error events (which depart and rejoin the all zero trellis path exactly once) for the inner and outer CCs are n^i and n^o, respectively. For a given input weight w and parity branch output weight h, the maximum exponent of N in (19) is

$$\alpha(w,h) = \max_{l,s}\left\{n^o + n^i - \frac{l+s}{2} - 1\right\} \qquad (20)$$

In particular, we are interested in the maximum exponent of N for any possible (w,h) which we denote as $\alpha_{\max}$.

We are interested in comparing recursive and non-recursive (feed-forward) CCs as the constituent codes. The main property of interest in this context is the minimum input weight to the CC that will yield a simple non-propagating error pattern. For a non-recursive CC, this minimum value is 1 while it is 2 for a recursive CC. This is important for determining the maximum values of n^o and n^i, which in turn determine $\alpha_{\max}$.

First consider the case when the inner CC is non-recursive. The maximum number of simple error patterns for the inner code is $n^i_{\max} = s$, the number of non-zero inputs to the inner CC, since each can cause a simple error pattern. Also, $n^o_{\max}$ is at least $\lfloor \frac{w}{2} \rfloor$ so that

$$\begin{aligned} \alpha(w,h) &= \max_{l,s}\left\{n^o - \frac{l-s}{2} - 1\right\} \qquad (21)\\ &\geq 0 \end{aligned}$$

so there is no guaranteed interleaver gain when the inner CC is non-recursive.

When the inner code is recursive, $n^o_{\max} = \lfloor \frac{s}{2} \rfloor$, yielding

$$\alpha(w, h) = \max_l \left\{ n^o + \left\lfloor \frac{s}{2} \right\rfloor - \frac{l+s}{2} - 1 \right\} \tag{22}$$

Because the difference between s and l is always even, we can break $\frac{l+s}{2}$ into $\lfloor \frac{l+1}{2} \rfloor + \lfloor \frac{s}{2} \rfloor$. Simplifying gives

$$\alpha(w, l) = \max_l \left\{ n^o - \left\lfloor \frac{l+1}{2} \right\rfloor - 1 \right\} \tag{23}$$

Since $n^o_{\max} \leq \frac{l}{d^o_{\text{free}}}$, where d^o_{free} is the free distance of the outer code, we obtain

$$\alpha_{\max} \leq \max_l \left\{ \left\lfloor \frac{l}{d^o_{\text{free}}} \right\rfloor - \left\lfloor \frac{l+1}{2} \right\rfloor - 1 \right\} \tag{24}$$

which for $d^o_{\text{free}} \geq 2$, yields

$$\alpha_{\max} \leq - \left\lfloor \frac{d^o_{\text{free}} + 1}{2} \right\rfloor \tag{25}$$

For large block size $d^o_{\text{free}} = d^o_{\min}$ and this is the result given in (3).

References

[1] C. Berrou, A. Glavieux, and P. Thitmajshima, "Near shannon limit error-correcting coding and decoding: turbo-codes," in *Proc. International Conf. Communications*, Geneva, Switzerland, May 1993, pp. 1064–1070.

[2] R. G. Gallager, "Low density parity check codes," *IEEE Trans. Information Theory*, vol. 8, pp. 21–28, January 1962.

[3] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEE Electronics Letters*, vol. 33, pp. 457–458, March 1997.

[4] S. ten Brink, "Convergence of iterative decoding," *IEE Electronics Letters*, pp. 1117–1119, June 1999.

[5] S. ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," *IEEE Trans. Commununication*, pp. 1727–1737, October 2001.

[6] T.J. Richardson and R.L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Information Theory*, vol. 47, pp. 599–618, Feb. 2001.

[7] T.J. Richardson, M.A. Shokrollahi, and R.L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Information Theory*, vol. 47, no. 2, pp. 619–673, February 2001.

[8] S. Dolinar, D. Divsalar, and F. Pollara, "Code performance as a function of block size," Tech. Rep., JPL-TDA, May 1998, 42–133.

[9] K.M. Chugg and P. Gray, "A simple measure of good modern code peforamnce," (in preparation).

[10] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenation of interleaved codes: performance analysis, design, and iterative decoding," *IEEE Trans. Information Theory*, vol. 44, no. 3, pp. 909–926, May 1998.

[11] D. Divsalar and F. Pollara, "Hybrid concatenated codes and iterative decoding," Tech. Rep., JPL-TDA, August 1997, 42–130.

[12] H. Jin, A. Khandekar, and R. McEliece, "Irregular repeataccumulate codes," in *Turbo Code Conf.*, Brest, France, 2000.

[13] K. M. Chugg, "A new class of turbo-like code with desirable practical properties," in *IEEE Communication Theory Workshop (no proceedings)*, Capri Island, May 2004, (recent results session).

[14] "FlexiCodes: A highly flexible FEC solution," 2004, TrellisWare Technologies White Paper, available at http://www.trellisware.com.

[15] K.M. Chugg, P. Gray, and R. Ward, "Flexible coding for 802.11n MIMO systems," September 2004, IEEE 802.11-04/0953r3, (Berlin, Germany), available at http://www.trellisware.com.

[16] L. Ping, X. Huang, and N. Phamdo, "Zigzag codes and concatenated zigzag codes," *IEEE Trans. Information Theory*, vol. 47, pp. 800–807, Feb. 2001.

[17] S. Benedetto and G. Montorsi, "Design of parallel concatenated convolutional codes," *IEEE Trans. Commununication*, vol. 44, pp. 591–600, May 1996.

[18] S. Benedetto and G. Montorsi, "Unveiling turbo codes: some results on parallel concatenated coding schemes," *IEEE Trans. Information Theory*, vol. 42, no. 2, pp. 408–428, March 1996.

[19] A. Graell, i Amat, S. Benedetto, and G. Montorsi, "Optimal high-rate convolutional codes for partial response channels," in *Proc. Globecom Conf.*, Taipei, Taiwan, 2002, pp. CTS–01–4.

[20] L. Ping, "The SPC technique and low complexity turbo codes," in *Proc. Globecom Conf.*, Rio de Janeiro, Brazil, December 1999, pp. 2592–2596.

[21] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Information Theory*, vol. IT-20, pp. 284–287, March 1974.

[22] K. M. Chugg, A. Anastasopoulos, and X. Chen, *Iterative Detection: Adaptivity, Complexity Reduction, and Applications*, Kluwer Academic Publishers, 2001.

[23] A. Abbasfar, D. Divsalar, and K. Yao, "Accumulate repeat accumulate codes," in *Proc. Globecom Conf.*, Dallas, Texas, December 2004, pp. 509–513.

[24] A. Abbasfar, D. Divsalar, and K. Yao, "Maximum likelihood decoding analysis of accumulate-repeat-accumulate codes," in *Proc. Globecom Conf.*, Dallas, Texas, December 2004, pp. 514–519.

[25] N. Wiberg, *Codes and Decoding on General Graphs*, Ph.D. thesis, Linköping University (Sweden), 1996.

[26] S. M. Aji and R. J. McEliece, "The generalized distributive law," *IEEE Trans. Information Theory*, vol. 46, no. 2, pp. 325–343, March 2000.

[27] F.R. Kschischang, B.J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Information Theory*, vol. 47, pp. 498–519, Feb. 2001.

[28] G. D. Forney, Jr., "Codes on graphs: Normal realizations," *IEEE Trans. Information Theory*, pp. 520–548, February 2001.

[29] J. Dielissen and J. Huisken, "State vector reduction for initialization of sliding windows MAP," in *2nd Internation Symposium on Turbo Codes & Related Topics*, Brest, France, 2000, pp. 387 – 390.

[30] S. Yoon and Y. Bar-Ness, "A parallel MAP algorithm for low latency turbo decoding," *IEEE Communications Letters*, vol. 6, no. 7, pp. 288 – 290, July 2002.

[31] A. Abbasfar and K. Yao, "An efficient and practical architecture for high speed turbo decoders," in *IEEE 58th Vehicular Technology Conference*, October 2003, pp. 337 – 341 Vol.1.

[32] L. Ping, W. K. Leung, and N. Phamdo, "Low density parity check codes with semi-random parity check matrix," *IEE Electron. Lett.*, vol. 35, no. 1, pp. 38–39, Jan. 1999.

[33] K. R. Narayanan, I. Altunbas, and R. Narayanaswami, "Design of serial concatenated msk schemes based on density evolution," *IEEE Trans. Commununication*, vol. 51, pp. 1283 – 1295, Aug. 2003.

[34] D. J. C. MacKay, *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press, 2003.

[35] P. E. McIllree, "Channel capacity calculations for M-ary N-dimensional signal sets," M.S. thesis, The U. South Australia, School of Electronic Eng., Adelaide, February 1995.