IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2023.DOI

Reordered Exponential Golomb Error Correction Code for Universal Near-Capacity Joint Source-Channel Coding

ALEXANDER HAMILTON, MOHAMMED EL-HAJJAR, and ROBERT G. MAUNDER
Department of Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ, U.K
Corresponding author: Robert G. Maunder (e-mail: rm@ecs.soton.ac.uk)

ABSTRACT Joint Source-Channel Coding (JSCC) is a powerful technique that allows for the efficient transmission of information by simultaneously considering the characteristics of both the source and the channel. The recently proposed Exponential Golomb Error Correction (ExpGEC) and Rice Error Correction (REC) codes provide generalized JSCC schemes for the near capacity coding of symbols drawn from large or infinite alphabets. Yet these require impractical decoding structures, with large buffers and inflexible system design, this was mitigated by the introduction of the Reordered Elias Gamma Error Correction (REGEC) which itself had limited flexibility with regards to source distribution. In this paper, we propose a novel Reordered Exponential Golomb Error Correction (RExpGEC) coding scheme, which is a JSCC technique designed for flexible and practical near-capacity performance. The proposed RExpGEC encoder and decoder are presented and its performance is analysed using Extrinsic Information Transfer (EXIT) charts. The flexibility of the RExpGEC is shown via the novel trellis encoder and decoder design. Finally, the Symbol Error Rate (SER) performance of RExpGEC code is compared when integrated into the novel RExpGEC-URC-QPSK scheme against other comparable JSCC and Separate Source Channel Coding (SSCC) benchmarkers. Specifically the RExpGEC-URC-QPSK scheme is compared against the REGEC-URC-QPSK scheme, and a serial concatenation of the Exponential Golomb and Convolution Code, which becomes the novel Exp-CC-URC-QPSK scheme. Our simulation results demonstrate the performance gains and flexibility of the proposed RExpGEC-URC-QPSK scheme against the benchmarkers in providing reliable and efficient communications. Specifically, the RExpGEC-URC-QPSK scheme outperforms the SSCC in a uncorrelated Rayleigh fading channel by 2 to 3.6 dB (dependent on source distribution). Furthermore, the RExpGEC-URC-QPSK scheme consistently operates within 2.5 dB of channel capacity when measuring E_b/N_0, whilst providing flexibility in SNR performance when compared to the REGEC-URC-QPSK scheme. These performance gains come at the cost of complexity, whereby the RExpGEC-URC-QPSK scheme is 3.6 times more complex than Exp-CC-URC-QPSK scheme under certain conditions. This paper highlights the unique capabilities of RExpGEC as a high performance, practical and flexible JSCC technique.

INDEX TERMS Channel Coding, Joint Source-Channel Coding (JSCC), Reordered Exponential Golomb, Source Coding.

I. INTRODUCTION

THE modern world relies heavily on information content and reliable access to information transfer of large data sources. In order to transmit this information over noisy wireless communications links via reliable means, information sources are typically encoded via a process known as source coding in order to compress them. Then these sources are separately encoded to enable redundancy and robustness by channel coding. This process, originally postulated by Shannon in [1] utilizes Separate Source and Channel Coding (SSCC) which can theoretically achieve near capacity operations. In SSCC a separate near-entropy source code, such

as the Lempel-Ziv code [2], Elias Gamma [3], Huffman [4], Golomb [5], Shannon-Fano [6] codes, or many other possible source codes [7]–[10], can be utilised with a separate channel code, such as a turbo code [11], [12], Polar Code [13], LDPC code [14], or any number of channel codes [6], [15]–[19] to theoretically achieve near-capacity performance. However in order to achieve near-capacity performance these near-entropy source codes become infinitely complex, extraordinarily large in block size [20] or require large processing times [21]. For example, the Lempel-Ziv code [2] requires accurate knowledge of the entire source and associated symbol probabilities before a single bit can be transmitted to the receiver. Equally with SSCC, a single bit in error, or a dropped packet can cause the entirety of the information to be lost. Therefore, not only must a robust channel code be utilised, but error checking and higher layer signalling must be employed to ensure the entire message is received.

For the transmission of information, whereby the source distribution is not known apriori, there is strong motivation for the use of universal codes for source coding [22], which provide finite codeword lengths irrespective of source distribution, provided the source is monotonically distributed. This family of codes are known as universal codes, and include Elias Gamma [3], Fibonacci Code [23] and the Exponential Golomb (ExpG) code [24]. These universal codes operate without any knowledge of the source symbol probabilities. However, non-negligible redundancy remains in the encoded bitstreams, which in turn leads to capacity loss when treated as a separate source and channel problem. This capacity loss motivates Joint Source Channel Coding (JSCC) [25] whereby the residual redundancy from the source coding can be utilised to enhance the attainable error correction capability.

State of the art JSCC techniques, such as the Reordered Elias Gamma Error Correction Code (REGEC) [34] have been demonstrated to show near capacity performance both large and infinite cardinality sources. However, they are only designed for a limited range of probability distributions, approximating specific zeta distributed sources which are psuedo-monotonic in nature (where successive symbols have successively lower symbol probabilities). This limitation meant that outside of these source distributions the REGEC code offers poor coding efficiency outside of these source distributions.

The Exponential Golomb Error Correction Code (ExpGEC) code introduced in [35] is parameterizable, whereas the Elias Gamma code used in the REGEC code of [34] is a special case of this with fixed parameters. Therefore, the ExpGEC is more generalized and has greater flexibility for different source distributions. However, the proposed scheme requires a series of buffers to realise, due to the structure of the ExpG code and its variable length nature. Furthermore, *the proposed scheme had high complexity and the ExpGEC could not be represented using a single finite complexity decoder*, as could be done with the REGEC in [36]. A summary of relevant and major contributions in the field of channel coding, source coding, and JSCC is presented in Table 1.

TABLE 1. Relevant and major contributions in source and channel coding

Year	Author	Contribution
1948	Shannon, C. [1]	The foundations of information theory
1952	Huffman, D.A [4]	Huffman source code
1960	Reed, I.S.; Solomon, G. [19]	Reed-Solomon channel code
1962	Gallager, R. [14]	Original LDPC channel code
1967	Viterbi, A.J. [18]	The Viterbi decoding algorithm
1975	Elias, P. [3]	Elias Gamma source code
1977	Massey, J.L [25]	Non-iterative JSCC
1978	Ziv, J.; Lempel, A. [2]	Lempel-Ziv variable rate source code
1979	Rissanen, J [26]	Arithmetic source coding
1980	Stout, Q.F. [27]	The Stout source code
1988	Bernard, M.A [28]	The VLEC JSCC
1993	Berrou, C [11]	The Turbo channel code
1996	Fraenkel, A.S [23]	The Fibonacci channel code
2000	Bauer, R.; Hagenauer, J. [29]	Introduction of iterative decoding for VLECs
2000, 2001	Görtz, N [30], [31]	The introduction of iterative JSCC decoding
2009	Arikan, E. [13]	Polar channel code
2013	Maunder, R.G. [32]	The Unary Error Correction (UEC) JSCC
2013	Wang, T. [33]	The Elias Gamma Error Correction (EGEC) JSCC
2016	Wang, T. [34]	The Reordered Elias Gamma Error Correction (REGEC) JSCC
2016	Brejza, M. [35]	The Exponential Golomb Error Correction (ExpGEC) JSCC
2023	Hamilton, A. (This work)	The Reordered Exponential Golomb Error Correction (RExpGEC) JSCC

In this paper, we propose the Reordered Exponential Golomb Error Correction Code (RExpGEC), which attempts to tackle these problems, and presents a novel highly flexible JSCC that can be used for diverse probability distributions with a large of infinite source cardinality. The proposed RExpGEC is able to be realised in a finite complexity decoder attaining near-capacity performance for a variety of source symbol distributions. We further illustrate the performance and flexibility of the novel RExpGEC by providing simulation results of the proposed scheme in a uncorrelated Rayleigh fading channel in comparison to a SSCC benchmarker.

Given the above background the novel contributions of this paper can be summarized as follows:

1) We propose the novel RExpGEC coding scheme, which is a flexible near-capacity JSCC suitable for any pseudo-monotonic source distributions including diverse zeta-distributed sources.
2) We propose a novel trellis decoder designed for the proposed RExpGEC code which has a low and finite complexity even when symbols are drawn from a large or infinite alphabet.

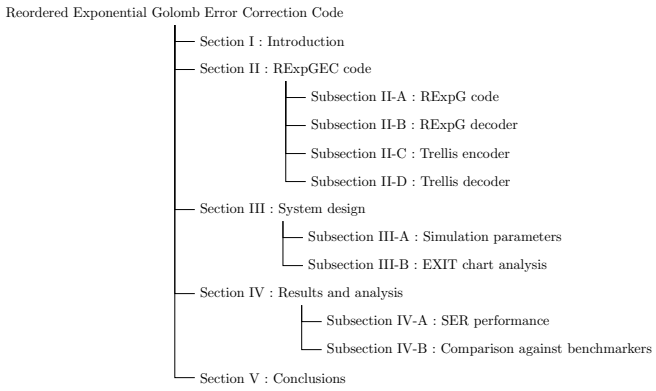

FIGURE 1. Structure of the paper.

3) We propose a novel iterative decoding scheme which exemplifies the concatenation of the RExpGEC with a Unity Rate Convolutional (URC) inner code and QPSK Modulation (RExpGEC-URC-QPSK)
4) We present novel performance analysis of RExpGEC, measured by Extrinsic Information Transfer (EXIT) chart analysis initially developed in [37], and by Symbol Error Rate (SER) performance when concatenated with a URC inner code and QPSK modulation,
5) We compare the Symbol Error Rate (SER) of the RExpGEC-URC-QPSK scheme performance to that of comparable JSCC and SSCC techniques.

Our findings illustrate the exceptional performance and flexibility of RExpGEC in delivering reliable and efficient communication over a wireless channel. Specifically, the RExpGEC outperforms the SSCC in a uncorrelated Rayleigh fading channel by 2 to 3.6 dB (dependent on source distribution), whilst providing a finite and low complexity that is flexible dependent on system design. Furthermore, we show that by increasing the paramaterization of the REGEC with the RExpGEC, the utility for different pseudo-monotonical source distribution improves, due to the ability to closer match the source distribution to the average codeword length and target the source coding performance of the RExpGEC.

This paper emphasizes the unique features of the RExpGEC and its ability to offer near-capacity flexible source and channel coding. A visual structure of the paper can be seen in Figure 1 demonstrating the flow of the discussion within this paper. The rest of the paper is organized as follows. We provide an introduction to the novel RExpGEC code in Section II, where we highlight the novel contributions both around the development of a universal JSCC that can be flexibility adapted to different source distribution and has a finite low complexity decoder even when the source alphabet is large or infinite. In Section III we exemplify a novel concatenation of the RExpGEC with a URC inner code and QPSK modulation, known as the RExpGEC-URC-QPSK scheme, to achieve near-capacity performance. Furthermore, Section III also provides novel contributions presented based on the EXIT chart analysis of the RExpGEC outer and URC inner code to enable the near-capacity design of the inner code and predict SER performance. In Section IV we then compare the SER results against a series of benchmarkers in order to understand the relative and absolute performance of the RExpGEC scheme. Finally, we conclude in Section V.

II. PROPOSED REXPGEC CODE

In this section, we introduce the novel RExpGEC code design and its constituent components, which are the novel RExpG encoder and the Trellis encoder which enables transition from RExpG to RExpGEC, as well as the corresponding novel trellis decoder which enables a finite, low complexity and flexible decoder design for the RExpGEC.

The considered block diagram of the RExpGEC is shown in Figure 2, where $\mathbf{x}$ represents the sequence of information symbols, $\mathbf{r}$ the sequence of RExpG encoded bits, and $\mathbf{z}$ the sequence of RExpGEC encoded bits. For the purpose of Figure 2 the inner code functionality also constitutes modulation mapping and demapping for transmission over a wireless channel, as these can be seen as part of the generic inner code functionality for the RExpGEC.

Section II-A introduces the RExpG code, its design, and some of it's attractive qualities which enable the novel tree and trellis designs. Section II-B introduces our novel RExpG decoder, which operates on a novel tree representation of the RExpG code. This tree has a finite complexity which enables the design of a finite yet flexible trellis, which is used as the basis for RExpGEC decoder described in Section II-D. Section II-C introduces the novel trellis structure design for the RExpGEC and how it is used to encode RExpGEC codewords. Section II-D introduces the novel trellis decoder for the RExpGEC and how it is used to decode RExpGEC codewords for near-capacity performance.

A. REXPG CODE

The proposed novel Reordered Exponential Golomb (RExpG) code is a universal code which can work with large or infinite source sets, where each symbol in the source sequence $\mathbf{d}$ has a value (d_i) in the range 1 to L (which is the cardinality of the source), where higher symbol values correspond to reducing probabilities. Each symbol is mapped to a RExpG codeword, where longer codewords are used for higher symbol indices. These symbols can be seen in Table 2 The code is parameterised by the k parameter, where $k \in (0, 1, 2, 3, ...)$. This enables the code to have high-efficiency source coding for a variety of different distributed sources [38], as discussed in [35] where the authors discuss the information efficiency of the similar Exponential Golomb code for varying zeta-distributed sources. This ability to closely match a zeta distributed source enables the RExpG to be suited to a variety of different uses, such as video [34], [39], text [40] or imagery [41] which provides the RExpGEC with a wide flexibility.

As shown in Table 2 the bits in each RExpG codeword may be considered to derive from two groups, a unary part

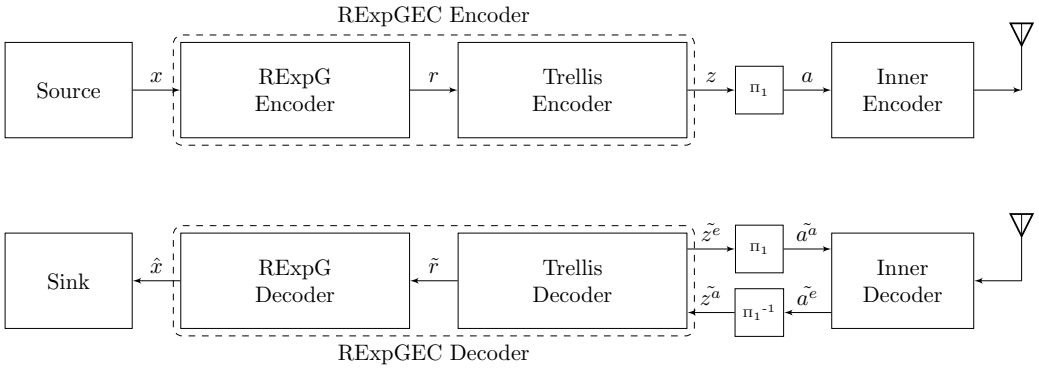

FIGURE 2. Generic Block Diagram of REXpGEC.

Reducing Probability

d_i	REXpG(d_i) k=0 Combi	T	REXpG(d_i) k=1 Combi	T	REXpG(d_i) k=2 Combi	Term	REXpG(d_i) k=3 Combi	Term
1	**1**		**1**	**0**	**1**	**00**	**1**	**000**
2	**001**		**1**	**1**	**1**	**01**	**1**	**001**
3	**011**		**001**	**0**	**1**	**10**	**1**	**010**
4	**00001**		**001**	**1**	**1**	**11**	**1**	**011**
5	**00011**		**011**	**0**	**001**	**00**	**1**	**100**
6	**01001**		**011**	**1**	**001**	**01**	**1**	**101**
7	**01011**		**00001**	**0**	**001**	**10**	**1**	**110**
8	**0000001**		**00001**	**1**	**001**	**11**	**1**	**111**
9	**0000011**		**00011**	**0**	**011**	**00**	**001**	**000**
10	**0001001**		**00011**	**1**	**011**	**01**	**001**	**001**
11	**0001011**		**01001**	**0**	**011**	**10**	**001**	**010**
12	**0100001**		**01001**	**1**	**011**	**11**	**001**	**011**
13	**0100011**		**01011**	**0**	**00001**	**00**	**001**	**100**
14	**0101001**		**01011**	**1**	**00001**	**01**	**001**	**101**
15	**0101011**		**0000001**	**0**	**00001**	**10**	**001**	**110**

Increasing Codeword length

TABLE 2. The decomposition of symbols d_i of the Reordered Exponential Golomb Codewords for $k \in 0, 1, 2, 3$ demonstrating the concatenation of the Mixed and Terminal codes. 'Combi' refers to the combination of interlaced FLC and Unary bits, 'T' and 'Term' both refer to the Terminal bits of the REXpG codeword.

Reducing Probability

d_i	ExpG(d_i) k=0 Unary	FLC	ExpG(d_i) k=1 Unary	FLC	ExpG(d_i) k=2 Unary	FLC	ExpG(d_i) k=3 Unary	FLC
1	**1**		**1**	**0**	**1**	**00**	**1**	**000**
2	**01**	**0**	**1**	**1**	**1**	**01**	**1**	**001**
3	**01**	**1**	**01**	**00**	**1**	**10**	**1**	**010**
4	**001**	**00**	**01**	**01**	**1**	**11**	**1**	**011**
5	**001**	**01**	**01**	**10**	**01**	**000**	**1**	**100**
6	**001**	**10**	**01**	**11**	**01**	**001**	**1**	**101**
7	**001**	**11**	**001**	**000**	**01**	**010**	**1**	**110**
8	**0001**	**000**	**001**	**001**	**01**	**011**	**1**	**111**
9	**0001**	**001**	**001**	**010**	**01**	**100**	**01**	**0000**
10	**0001**	**010**	**001**	**011**	**01**	**101**	**01**	**0001**
11	**0001**	**011**	**001**	**100**	**01**	**110**	**01**	**0010**
12	**0001**	**100**	**001**	**101**	**01**	**111**	**01**	**0011**
13	**0001**	**101**	**001**	**110**	**001**	**0000**	**01**	**0100**
14	**0001**	**110**	**001**	**111**	**001**	**0001**	**01**	**0101**
15	**0001**	**111**	**0001**	**0000**	**001**	**0010**	**01**	**0110**

Increasing Codeword length

TABLE 3. The decomposition of symbols d_i of the ExpG Codewords for k=0, k=1, k=2, and k=3 showing the concatenation of the Fixed Length Code and Unary codes.

and a fixed length part, which are interlaced to create the REXpG codeword, in a manner shown in Figure 3. For the purpose of illustration in Figure 3 the bits which derive from the unary group can be referred to as the unary sub-symbol, and those from a fixed length the Fixed Length Code (FLC) sub-symbol. The bits which derive from the an underpinning Fixed Length Code (FLC), are shown in yellow in Tables 2 and 3 and fig. 3 and referred to as u, whereas the bits which derive from a unary code are shown in blue in Tables 2 and 3 and fig. 3 and are referred to as t. The initial bit of the REXpG codeword is always a unary bit, following which the bits are subsequently interlaced between unary and FLC bits until the final unary bit is reached, this initial section of the codeword can be referred to as the 'combination' section of the codeword. The final k bits corresponding to the k parameter of the REXpG code itself will always be bits derived from the fixed length sub-symbol, where this section of the REXpG codeword can be referred to as the 'terminal' section of the codeword. This interlacing and construction can be observed in Figure 3

The Unary Code, as used in the unary sub-symbol, is defined by a series of all zero-valued bits immediately followed by a single logical one-valued bit, which is subsequently referred to as the *terminal unary bit* which indicates the end of the Unary Code sub-symbol, the total length of the Unary code sub-symbol u is defined as $x(d_i)$.

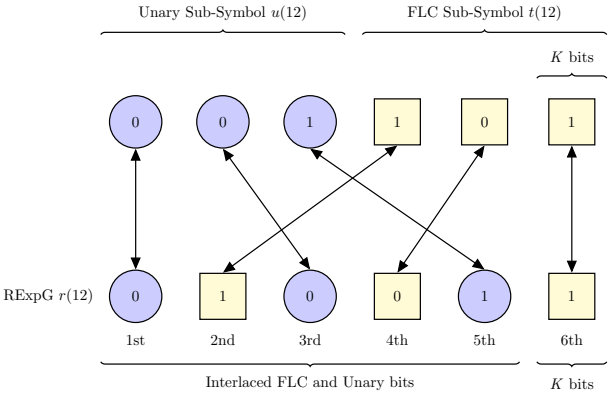

FIGURE 3. Construction of REXpG symbol $d_i = 12$ when $k = 1$.

$$x(d_i) = \lfloor log_2(d_i + 2^k - 1) \rfloor + 1 - k. \quad (1)$$

The Fixed Length Code sub-symbol is a representation of the bits which exist within a RExpG symbol which derive from the fixed length part and can be directly be calculated as the sub-symbol $t(d_i)$. The value of the fixed length sub-symbol is.

$$t(d_i) = d_i - 2^{\lfloor log_2(d_i + 2^k - 1) \rfloor} + 2^k - 1, \quad (2)$$

which is represented in binary form with a length exactly corresponding to $s(d_i)$.

$$s(d_i) = \lfloor log_2(d_i + 2^k - 1) \rfloor. \quad (3)$$

The RExpG codeword has a length as follows in Equation (4), of which $x(d_i)$ of those are Unary bits, with the remainder $s(d_i)$ being FLC bits. This is the sum of the length of the Unary $x(d_i)$ and FLC $s(d_i)$ sub-symbols.

$$l(d_i) = x(d_i) + s(d_i) = 2 \times \lfloor log_2(d_i + 2^k - 1) \rfloor + 1 - k. \quad (4)$$

Based on Equation (4), for the RExpG the *terminal unary bit* is set at a fixed k bit locations from the end of the codeword and preceding this every alternative bit is a Unary codeword. Therefore, there is always a known number of bits to the end of the codeword, which is k bits, following a logical one-value unary bit. This enables a finite complexity decoder to be utilised as will be detailed in Section II-D.

To provide a specific example of a RExpG symbol, we visually describe the process for developing the RExpG symbol for $d_i = 12$ when $k = 1$ in Figure 3, and how a transmitter can convert from symbol d_i to a RExpG codeword $r(d_i)$. This symbol $d = 12$,k$= 1$ will be utilised as an example throughout Section II.

As can be observed in Figure 3, initially the sub-symbol $u(12)$ is generated with a length of 3, as defined by Equation (1), which generates the sub-symbol $0, 0, 1$ (as indicated by the blue bits), and the corresponding sub-symbol $t(12)$ with the integer value 5 as defined by Equation (2) , which in binary form with a length defined by Equation (3) is $1, 0, 1$ (as indicated by the yellow bits). Following this via the RExpG generation process these sub-symbols produce the RExpG codeword $0, 1, 0, 0, 1, 0$. Therefore the bits in location 2,4 and 6 originate from the Unary sub-symbol, whereas bits in location 1,3 and 5 originate from the FLC sub-symbol.

Table 2 shows the first 15 codewords for the RExpG code for $k \in 0, 1, 2, 3$. The colours represent the same colours as those used in Figure 3, which will be the same colours used throughout the remainder of this paper to represent bits which are derived from FLC and unary components.

Let us now compare our novel RExpG code with the ExpG code of [35], for which the first 15 codewords are shown in Table 3 and elaborate on the similarities and differences between the two codes. Whilst the structure of the RExpG code is generated from interlaced unary and FLC sub-symbols as described above, the structure of ExpG is composed simply of two concatenated sub-symbols $\mathbf{u}(\mathbf{d_i})$ and $\mathbf{t}(\mathbf{d_i})$, which represent a sub-symbol of Unary Code bits and a sub-symbols of Fixed Length Code bits respectively. As observed in Table 3 the location of the *terminal Unary bit* of each ExpG symbol $\mathbf{d}(\mathbf{d_i})$ in a sequence of symbols $\mathbf{D}$ is at an unknown location, as the *terminal unary bit* is located at a variable location from the end of each codeword, and preceded by an unknown number of logical zero-valued bits. This property motivates the design of the RExpG code, where in contrast, the *terminal unary bit* is at a known distance from the end of the codeword. As we will show in the next sections, this known property of the RExpG can be exploited for designing a finite complexity decoder, rather than the infinite complexity decoder that is required for the ExpG.

The total rate R_o of the RExpGEC when compared with the original information source is a function of both the coding rate of the scheme, the modulation order, as well as the information efficiency (η). The information efficiency of different P_1 and k distributions of the RExpG can be seen in Figure 4 where different P_1 and k distributions are shown. These difference η parameters, combined with the coding rate and modulation order, produce a Continuous-Input Continuous-Output Memoryless Channel (CCMC) capacity for the energy required per bit of information (based on the entropy of the information source) which differs according to the source distribution. For a monotonic source with P_1 of 0.6 and the k parameter of 0, this offers the highest information efficiency or η, which matches with the P_1 value chosen for the EXIT analysis presented in Figures 11 to 13 in Section III-C and simulations presented in Figure 15 in Section IV. However, for a different P_1 value, such as 0.3, other values of k offer better performance, and this can be observed in the results shown in Figures 16 and 17 of Section IV.

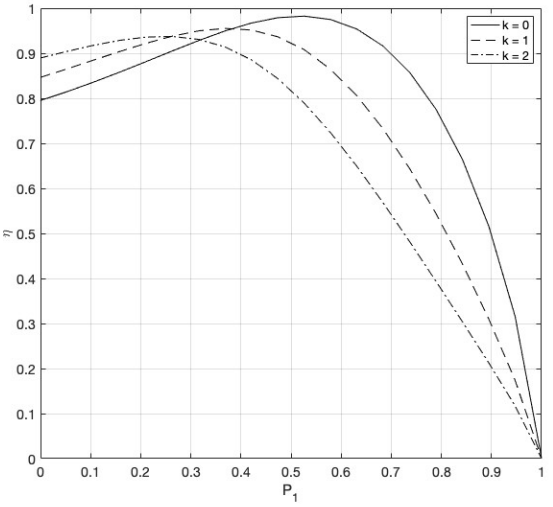

FIGURE 4. Information Efficiency (η_0) of RExpG with different Zeta distributions of infinite cardinality.

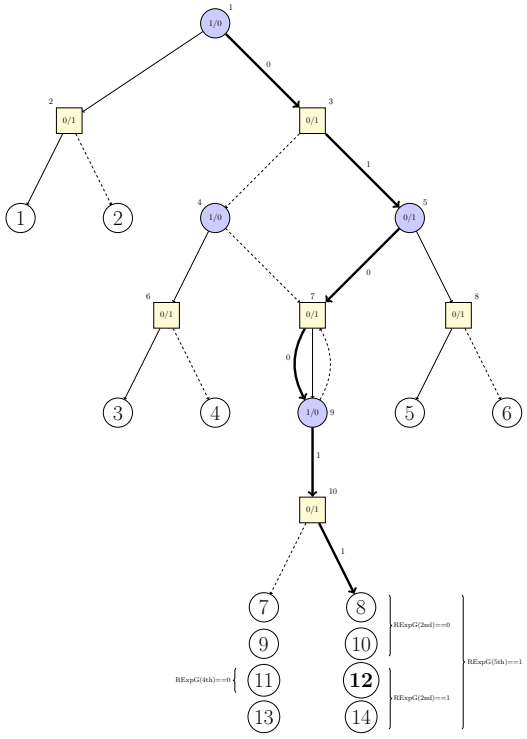

FIGURE 5. RExpG Tree Decoder with a tree depth of 1, and an example of decoding the RExpG symbol with values RExpG k=1, d_i=12.

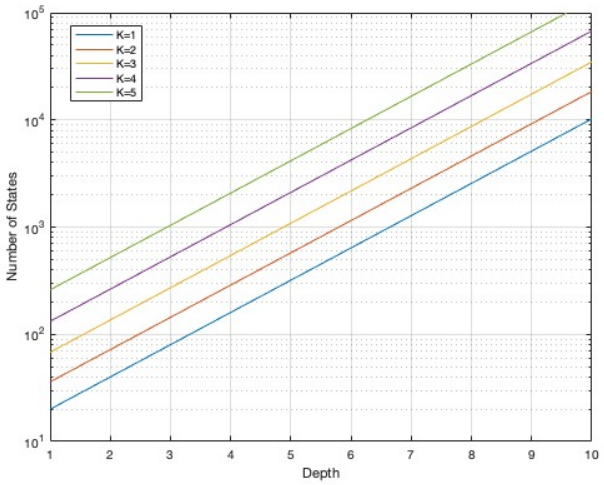

FIGURE 6. Number of States of the RExpG Tree decoder as function of both depth and k.

B. REXPG DECODER

In this section we introduce a method to decode a sequence of RExpG codewords via the RExpG tree decoder, which is illustrated in Figure 5.

As discussed in Section II-A, a key motivation for the RExpG (and associated RExpGEC) is to enable the serial decoding of a sequence of symbols $\mathbf{D}$ and to enable a finite yet flexible decoder design. In order to enable the RExpGEC to have a finite complexity decoder, one must have a finite sized binary tree, which can be achieved with the RExpG code and is described below.

In Figure 5 the tree is represented visually, where each line represents a transition of the RExpG codeword whereby a dashed line represents a 0 bit valued transition and a solid line represents a 1 bit valued transition, and each coloured node represents either a FLC derived state (yellow square) or a unary derived state (blue circle), which are uniquely numbered. The white circles represent the leaf nodes, which indicate that a symbol has been reached. In the case of the tree in Figure 5 which represents a depth of 1, there are 8 unique leaf nodes (also referred to as terminal nodes), with the final 2 leaf nodes representing all symbols above 7 depending on the *holding pattern*. A RExpG tree with a larger depth will have more unique leaf nodes and utilise the *holding pattern* less, as more symbols are likely to have unique paths through the tree.

Because of the design of the RExpG code, a tree can be designed in such a way with a *holding pattern*, which constitute a single FLC and Unary *holding state* with a circular transition as well as an exit and entry transition, as exemplified in Figure 5 between states '7' and '9'. This is considered until the conditions of the Unary sub-symbol becoming a logical 1-valued bit can escape to a known FLC sub-tree, which is constant, as there are always a constant k FLC bits, following the *terminal unary bit*. This can be observed in Figure 5, which shows the ability to enable a holding pattern at the *terminal unary bit*, which in turn enables a finite complexity tree to be achieved for the RExpG code.

Furthermore, where the value of k used in the tree in Figure 5 is 1, the k parameter dictates the number of states in each FLC branch, which comes from unary states, when a logical one-valued bit is identified, indicating a *terminal unary bit*. This is exemplified in Figure 5, where for states $'2', '6', '8'$ and $'10'$ these are the initial stages of an FLC branch. In this case, with the k paramater having a value of 1, there are 2 terminal nodes. Correspondingly, if k was 0 these nodes themselves would be only one terminal node, and if k was 2 there would be 2 FLC 'stages' and 4 terminal nodes.

The number of Unary branches which are before the holding pattern and therefore total number of states of the RExpG tree decoder can be defined by the *depth* parameter, which enables the system designer to vary the complexity of the system design. This scaling of complexity when increasing the k and $depth$ parameter, for the RExpG Tree decoder can be observed in Figure 6, where the number of states can be seen increasing exponentially with depth. This increase in depth enables more unique leaf nodes and unique paths through the tree, which in turn could enable greater entropy to be achieved and potentially achieve mild performance gains.

Furthermore, Figure 5 also offers an example of how the the tree decoder can be utilised to decode the symbol d = 12,

if we trace each received bit down the binary tree, which is represented by the bold trace through the tree itself. Note, this is the same received codeword as we have shown being generated in Figure 3.

Explicitly, when the first bit received is a logical 0, it is impossible for this to be a *terminal unary bit* and the trace proceeds to the right and not into the terminal FLC branch. At this point a logical one-valued bit is received for the 2nd transition, and this must be stored in memory via a counter (as it is retrieved once the decoder leaves the holding pattern). Then, at the next Unary bit, another zero-valued bit is received, and as the $depth$ of this decoder is fixed at 1, the decoder enters the holding pattern. Here the 4th transition is stored in memory as a 0, and the 5th transition represents the *terminal unary bit* as a logical one-valued bit, so the decoder leaves the holding pattern and into a terminal FLC branch. As the bits for the second and fourth transition were stored in memory via the use of a counter, they enable the receiver to identify that it was symbol 12 which was transmitted by observation of a matrix which can be used to identify which symbol was transmitted. This matrix will have a width of 2^k and a depth corresponding to length of each symbol. The 'depth' of the tree, in this case where k=1, depth=1 and the codeword is 6 bits in length, this matrix has 4 rows. Specifically it is possible to identify the symbol transmitted corresponds to d = 12 as the second and fourth transition from the counter $'1, 0'$, with the second transition being the most significant bit. The second transition defines whether the symbol at the terminal node exists in the lower half (if it is a 1), or the upper half (if it is a 0) of all possible states. Then, the fourth transition further down selects within this half and if the codeword is longer, then the sixth, eighth and every even transition until the *terminal unary bit* can downselect until a single row is identified. The column can then be identified by the transitions in the terminal FLC tree. In this case, a 1-valued bit was transmitted which corresponds to the right-most column, and as such it is able to be identified that the symbol in location (2,3) was transmitted, corresponding to symbol d = 12. This is shown by the brackets in Figure 5 whereby symbol 12 was able to be identified.

In comparison, in a scheme such as the ExpGEC as proposed in [35] a tree decoder would produce an infinite complexity tree, as it would be unable to generate a holding pattern as their is a variable number of bits seceding any *terminal unary bit*, which itself cannot be easily identified. This concern is visually demonstrated in Figure 7, where a branch extending infinitely to the right can be observed beyond the third Unary bit. To receive additional Unary bits within the ExpG codeword, the receiver needs to expand the FLC tree sub-structure, which increases the receiver complexity exponentially. This expansion could continue up to infinity, depending on the number of Unary bits the receiver intends to receive.

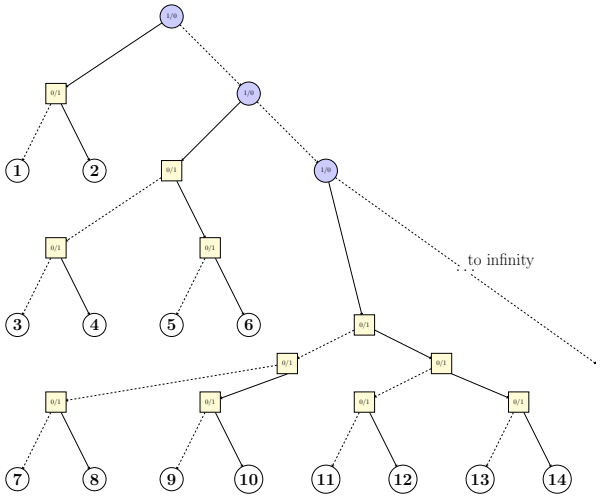

FIGURE 7. Tree Decoder for ExpG Decoder, k=1.

In Sections II-C and II-D we introduce the finite but variable trellis used in the encoding and decoding of RExpGEC, based upon the tree discussed in Section II-B.

C. REXPGEC TRELLIS ENCODER

In this section, we introduce the novel Trellis encoder as shown in Figure 2, which in addition to the RExpG encoder creates the overall RExpGEC encoder. The RExpGEC is the JSCC generated from the base RExpG code, discussed in Section II-A, via the RExpGEC trellis encoder which applies redundancy encoding according to the trellis design and codebook.

The Trellis encoder can be observed within Figure 2 as a component of the overall REXpGEC encoder functionality, with the input $\mathbf{r}$ and output $\mathbf{z}$. The overall trellis structure can be observed in Figure 9 and for each bit in vector $\mathbf{r}$ we progress through one stage of the trellis depending on the bit-value of the transition. The transitions that make up each individual trellis stage can be observed in Figure 8 where the colour mappings of the nodes match those in Figures 3, 5 and 7 to show FLC and Unary nodes with transitions between them according to the bit values in $\mathbf{r}$. Specifically, a dashed transition represents a transition occuring due to a 0 valued bit in the vector $\mathbf{r}$, and a solid transition represents a transition corresponding to a 1 valued bit in $\mathbf{r}$. Subsequently these transitions are referred to as $r_i = 0$ or $r_i = 1$.

The trellis is designed based upon the novel RExpG tree, which is discussed in Section II-B, which enables a finite complexity yet flexible design to be realised for any pseudo-monotonically distributed source.

A single stage of the RExpGEC trellis, as shown in Figure 8, is composed of a series of states represented as a column (m') along with a corresponding following series of states (m) and the possible transitions (m|m') between those states are indicated by a dashed line where a 0 valued bit transition occurs and a solid line when a 1 valued bit transition occurs, as mentioned above. These transitions themselves have direct mapping to the RExpG tree of Figure 5. For

example, node 1 of Figure 5 directly relates to state 1 of Figure 8 with transitions mapped accordingly to state 14 and 3 of Figure 8 which are directly related to nodes 2 and 3 of Figure 5. The transitions from (m|m') from each node are determined by the bit value of **r**, where each transition is encoded onto a codeword, as will be described later in this section, in order to form the RExpGEC codeword associated with the vector **z** of Figure 2.

The transitions can be defined according to some simple rules as follows:

- logical 1 valued RExpG transitions, where $r_i = 1$, which transit from a node n in the upper half of the trellis, complement logical 1 valued RExpG transition from node $n+1$, where both cross the trellis
- logical 0 valued RExpG transitions, where $r_i = 0$, which transit from a node n in the upper half of the trellis, complement logical 0 valued RExpG transition from node $n+1$, where both transitions stay in their original half of the trellis.
- logical 1 valued RExpG transitions of the vector **r**, which transit from a node where the state is a *Terminal FLC State* will immediately enter state 1 or 2, the *start/finish states*

These transitions allow different states to be transitioned into. In order to categorise the states there are four possible types of states, which we explicitly refer to in the trellis as the following:

1) **Start/Finish States** These are the states entered at the start and end of each RExpGEC codeword, which can be entered from a Unary or a FLC transition (dependent on k), and always leave on a Unary transition. The state is between the end of one codeword and the beginning of the next
2) **Transitory States** These are the states through which the codeword may pass for longer codewords, and they can be entered from a Unary or a FLC transition and exited on the opposite transition type to that which it was entered upon. If the exit transition is a $r_i = 1$ valued UEC transition then the remaining transitions of the RExpGEC codeword will be FLC states, as it is a *terminal unary bit* and the trellis immediately enters either the Terminal FLC states or the *start/finish states* depending upon the value of the k parameter.
3) **Holding States** These are the states which represent the Unary and FLC states which a codeword can enter for longer RExpGEC codewords. Depending on the $depth$ parameter, these holding states (which form a holding pattern) can have a higher or lower chance of being entered. If a $r_i = 1$ valued transition occurs when in the Unary holding state, it is a *terminal unary bit* and the trellis immediately enters either the Terminal FLC states or the *start/finish states* depending upon the value of the k parameter.
4) **Terminal FLC States** These states only exist where the k parameter of the RExpGEC is greater than 0, and they directly correspond to the FLC 'leaf nodes' of the RExpG tree. Once the trellis enters these states either the codeword will immediately enter the *start/finish states*, or further Terminal FLC States will be entered, dependent on the value of k.

We consider the complexity of the trellis decoder to be moderate in situations where the 'depth' parameter is low. Specifically, in the case where k=1 and depth=1 the trellis requires 140 add, compare, and select operations per iteration.

To provide an example of a specific path through the trellis, we use the RExpG codeword (d=12) = (0,1,0,0,1,1) to represent the vector **r**, with state 1 being the initial state. We take the bit in the first location denoted by $r_1 = 0$, then the dashed line representing a 0 logical bit leads to state 3 (a transitory state), $r_2 = 1$ causes the transition to state 6 (a transitory state), the next location $r_3 = 0$ would take the encoder to state 10 (a holding state), $r_4 = 0$ to state 12 (a holding state), $r_5 = 1$ to state 19 (a terminal FLC state) and $r_6 = 1$ to state 2 (a start/finish state). This path is denoted as the bold path in Figure 9 where several concatenated trellis stages are shown.

In order to enable a variable coding rate **r** for the RExpGEC, each transition within the trellis is assigned an output codeword of integer length $1/R$. In the case of the example shown in Figure 8, $R = 0.5$ and the output of each transition is of size 2. Therefore, the symbol of $d = 12$,k$= 1$ which is a 6 bit vector as the RExpG codeword (as shown in Figures 3 and 5) becomes a 12 bit length vector **z** as a RExpGEC codeword, of which the logical bit-values are dependent on the codebook assigned to each RExpGEC transition.

The codebook utilised for these transition must obey some rules in order to maintain maximum hamming distance between transitions when transmitted over the channel, and reduce the probability of trellis following the orthogonal path when received. These rules can be summarized as follows:

- logical $r_i = 1$ valued RExpG transitions from a node n in the upper half of the trellis, have a RExpGEC codeword output that is complement to the logical 0 valued RExpG transition from node $n+1$.
- logical $r_i = 0$ valued RExpG transitions from a node n in the upper half of the trellis, have a RExpGEC codeword output that is complement to the logical 1 valued RExpG transition from node $n+1$.
- logical 1 and logical 0 valued RExpG transitions from the same state must have RExpGEC codeword outputs that are orthogonal to each other.

Utilising these rules, a randomised codebook was generated to support Figure 8 that produced an output vector for RExpGEC(d=12) of (1,0,1,0,0,1,1,1,0,1,1,1), which is based on the path through the trellis discussed earlier in this section.

The trellis is mirrored and symmetrical with an upper and lower half. A state of location n in the upper half has a corresponding state $n+1$ in the lower half. For example, both state 1 and 2 of Figure 8 relate to node 1 of Figure 5. The

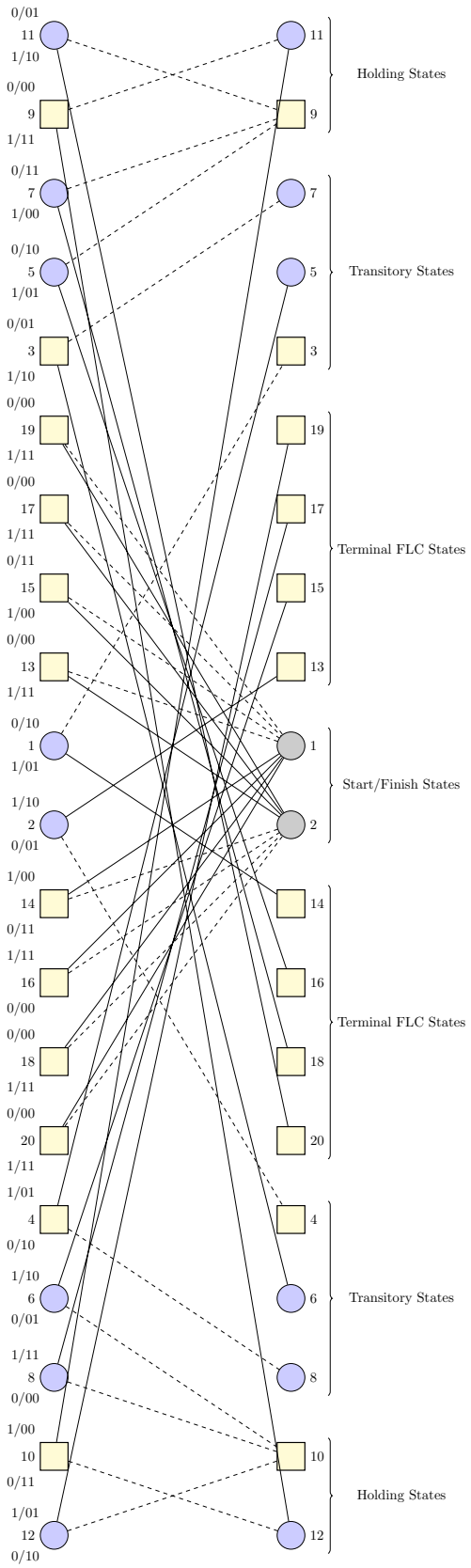

FIGURE 8. Trellis Stage for RExpGEC, depth = 1, k=1.

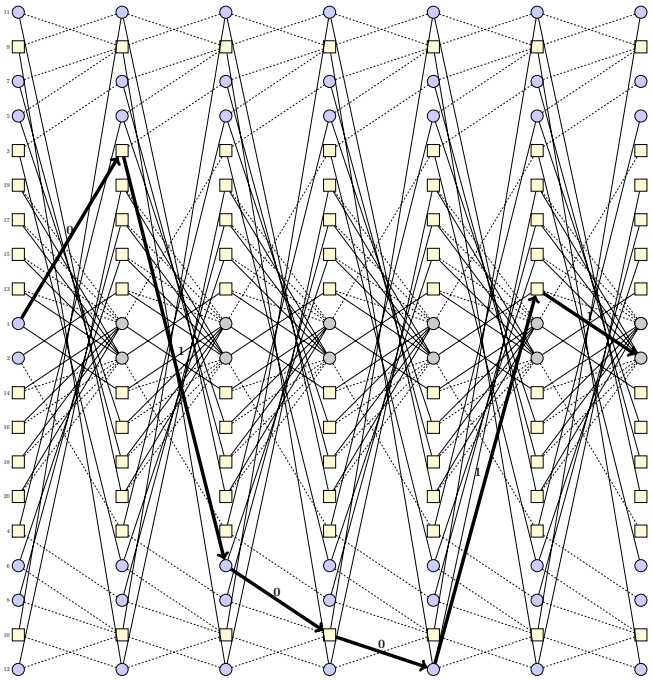

FIGURE 9. Trellis for RExpGEC, depth = 1, k=1.

reason for this mirroring is such that the trellis is designed in a manner whereby every $r_i = 1$ transition (of the RExpG code) causes a transition from the lower half of the trellis to the upper half, which allows the bit values of $\mathbf{z}$ to be equiprobable. If the RExpG was not designed to produce equiprobable bits then this would introduce capacity loss [32], [42].

D. REXPGEC TRELLIS DECODER

The RExpGEC decoder utilises the same trellis as that defined at the encoder. The decoder utilises the trellis to convert apriori LLRs related to encoded bits $\tilde{\mathbf{z}^{\mathbf{a}}}$, which are received from the inner decoder, into extrinsic LLRs relating to the encoded bits $\tilde{\mathbf{z}^{\mathbf{e}}}$. The decoder also outputs aposterior LLRs related to the uncoded bits $\tilde{\mathbf{r}}$ in accordance with the BCJR algorithm [43]. The BCJR algorithm [43] is a forward-backward algorithm used in digital communication for maximum a posteriori (MAP) decoding of any code that can be expressed in a trellis format. The extrinsic LLRs relating to the encoded bits $\tilde{\mathbf{z}^{\mathbf{e}}}$ can be exchanged with an inner decoder to form an iterative receiver. Within this iterative decoder the extrinsic LLRs produced by one component become the apriori LLRs to the other, with the quality of the LLRs typically improving with successive iterations. This enables iterative decoding of a series of received channel LLRs $\tilde{\mathbf{b}}$ to provide strong error correction performance.

Initially the trellis is populated with the received sequence of apriori encoded RExpGEC LLRs $\tilde{\mathbf{z}^{\mathbf{a}}}$, as well as the known probabilities of transitions $P(m|m')$ within the trellis.

The known probabilities of each trellis are calculated of-

fline as a conditional probability from each state to the next. All initial states of the RExpGEC have 2 possible transitions from every state, corresponding to $r_i = 0$ or $r_i = 1$. In order to calculate the probabilities in an empirical manner, prior to transmission and reception, a large discrete series of symbols may be generated apriori (according to the k-parameter), which become a sequence of RExpG codewords, which are then passed into the trellis encoder, whereby each individual transition occurrence is measured and the conditional probability from each node can be calculated. Specifically, this is calculated by measuring how many times the transition corresponding to $r_i = 0$ is used in comparison to $r_i = 1$, such that from each node the $P(r_i = 1)$ transition is calculated as $No(r_i = 1)/[No(r_i = 1) + No(r_i = 0)]$ with the $P(r_i = 0)$ being $1 - P(r_i = 1)$. Following this process, and due to the RExpGEC trellis being used for both encoding and decoding, the transitions of the known conditional probabilities $P(m|m')$ of the RExpGEC trellis transitions can be calculated.

The BCJR [12], [43]–[45] initially calculates the γ values, which are introduced in [43] which represent the probabilities of being in each state, given all the received RExpGEC apriori LLRs$\tilde{\mathbf{z}^{\mathbf{a}}}$ and the apriori known probabilities of each transition $P(m|m')$.

The algorithm then calculates the α values, which are the probabilities of reaching each state of the trellis from the previous set of states. The α values are calculated by the receiver as a function of k, *depth* of the RExpGEC and P_1 of the source and represent the likelihood of each forward transition. This forward recursion starts at the leftmost states and successively calculates α values for each set of states going from left to right [12].

Then, the algorithm calculates the backward β values, which are the probabilities of reaching each state of the trellis, given the previous state. This backward recursion starts from the rightmost states and successively calculates β values for each set of states going to the left [45].

The BCJR algorithm combines the α, β and γ values to obtain the LLRs of each of the RExpG code bits in $\tilde{\mathbf{r}}$ [43]. Following the completion of the BCJR algorithm, a hard decision can be made on all aposterior LLRs of the encoded bits to realise the vector $\hat{\mathbf{x}}$ as per Figure 2. This is done by analysing the sign of LLRs in $\tilde{\mathbf{r}}$, where postive LLRs represent a binary 1 and negative a binary 0. If the codewords are correctly received and the iterative code is able to correctly converge then these bits should directly match the RExpG codewords for the transmitted symbols x, as discussed in Section II-A and in Section II-B.

III. SYSTEM DESIGN

In this section, we present a novel system design which exemplifies the proposed JSCC RExpGEC code. This system concatenates the RExpGEC as introduced in Section II with a Unity Rate Code (URC) [46] and QPSK modulation [47]–[49]. The system is henceforth referred to as the RExpGEC-URC-QPSK scheme. The system design introduces the key aspects of the RExpGEC scheme to achieve a balance between flexibility and performance. The key components of the system are thoroughly discussed in this section to provide a clear understanding of the system, including the constituent code components. This section serves as a cornerstone for the evaluation of the system's performance, which is presented in subsequent sections of the paper.

In Section III-A we introduce the block diagram of the system. Section III-B introduces the system parameters and how they will be used in subsequent analysis.

EXIT chart analysis of the RExpGEC-URC-QPSK scheme is provided in Section III-C, and for comparison with a comparable SSCC in Section IV. Furthermore, comparison with a JSCC benchmarker, the REGEC code of [34] can be made utilising the specific example of k = 0, as the underlying REGEC can be seen as a special case of the RExpGEC where k = 0.

A. SYSTEM MODEL

The RExpGEC-URC-QPSK system is presented in the block diagram in Figure 10, where a vector of symbols $\mathbf{x}$ is generated, which are turned into a vector of RExpG uncoded bits $\mathbf{r}$ and subsequently RExpGEC encoded bits $\mathbf{z}$. These RExpGEC encoded bits are interleaved with Π_1 to form $\mathbf{a}$, then encoded with a URC code to produce encoded RExpGEC-URC bits $\mathbf{y}$. The RExpGEC-URC bits are then interleaved with Π_2 into the vector $\mathbf{b}$ and mapped to QPSK modulation for transmission over a wireless channel. For the receiver the signal is received, demapped according to the QPSK demapper, interleaved encoded LLRs corresponding to the URC encoded signal $\tilde{\mathbf{b}}$ and the interleaver π_2^{-1} are deinterleaved to form $\tilde{y}$. These LLRs then undergo the first operation of the URC decoding to form the extrinsic LLRs from the URC decoder $\tilde{\mathbf{z}^{\mathbf{a}}}$, along with a zero-valued $\tilde{a^a}$ (apriori information from the RExpGEC decoder). Following this the RExpGEC Trellis is populated with $\tilde{\mathbf{z}^{\mathbf{a}}}$ and iterative decoding between the RExpGEC Trellis decoder and URC decoder commences, where the output $\tilde{\mathbf{z}^{\mathbf{e}}}$ becomes the input to the URC of $\tilde{a^a}$ when interleaved with Π_1 and the output of the URC $\tilde{a^e}$ becomes the input to the RExpGEC $\tilde{\mathbf{z}^{\mathbf{e}}}$ when deinterleaved with π_1^{-1}. When the iteration limit is reached or the system has converged a decision is made on the uncoded RExpG LLRs $\tilde{\mathbf{r}}$ and the received symbols $\hat{\mathbf{x}}$ is provided to the Sink.

For the inner code, the URC is chosen in order to harness the iterative performance of turbo-style receivers whilst incurring moderate coding complexity [50]. The URC coding rate has a rate exactly equal to 1, such that the number of bits at the output $\mathbf{y}$ is exactly equal to the bits at the input $\mathbf{a}$, yet it introduces recursion into the bitstream, such that the bits depend upon each other. This then enables iteration and iterative information gain. URC codes have been proposed for a wide variety of diverse applications for such iterative purposes [51]–[54] and are well suited as an inner code for the RExpGEC-URC-QPSK scheme. Therefore, the incorporation of a carefully designed URC enables flexible iterative

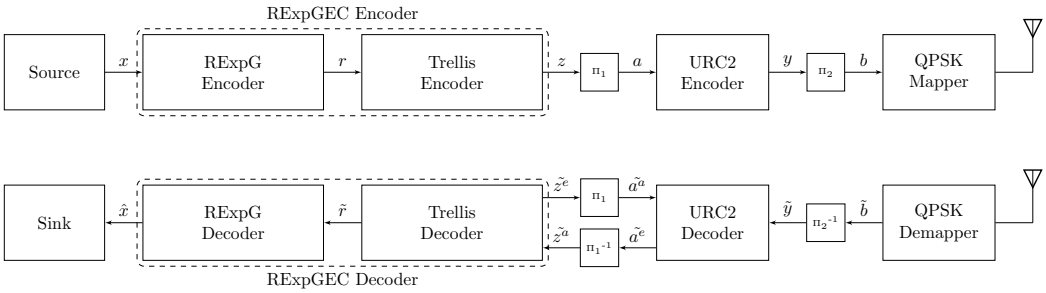

FIGURE 10. Block Diagram for RExpGEC with URC2 and QPSK Mapping.

coding design when used as an inner code for moderate complexity gain and no change to the overall RExpGEC coding rate. The complexity of the RExpGEC-URC-QPSK when the REXpGEC trellis decoder is concatenated with the URC code produces 154 add, compare, and select operations per iteration in the specific case of k=1 and depth=1, however it is acknowledged that this complexity will increase for differing k and depth parameters.

B. SIMULATION PARAMETERS

In order to analyse the performance of the RExpGEC and further enhance the scheme design, a number of simulation parameters require to be chosen for undertaking inital EXIT chart analysis [55], [56] and then in turn Symbol Error Rate (SER) performance, in comparison to the JSCC and SSCC benchmarker.

The variables which are chosen by the system designer are as follows:

- **k Parameter** The k parameter of the RExpGEC code will be set at different values for analysis of various options, specifically 0,1, and 2, in order to remain consistent with examples shown in Section II. The k parameter controls the length of the FLC bits at the end of the codeword, as shown in Table 2. A higher k value better matches a flatter source distribution with a lower P_1.
- **Coding Rate** The coding rate R_o of the RExpGEC can be altered by varying the bits allocated to each RExpGEC trellis transition as discussed in Section II-C. In order to remain consistent with prior work [34] for comparison the coding rate R_o for the benchmarker will be set at $1/2$.
- **Trellis 'Depth'** The trellis depth will affect the complexity of both the encoder and decoder, which is discussed in Section II-B. In order to remain consistent with examples shown in Section II the trellis depth will be set at a value of 1.
- **URC States** The URC operates on the basis of a trellis that uses the BCJR algorithm and we can control the number of states in it. However for consistence with prior work [34] two-state URC shall be used.
- **Modulation Mapping** The modulation mapping scheme chosen is required to be consitent across all benchmarkers. In order to remain consistent with prior work [34], QPSK shall be used.

The variables which are dependent on the zeta distributed source are:

- **P_1 of Source** P_1 is the probability of the first symbol of any source distribution. Any zeta like (or geometric) distributed source can be accurately characterised by it's size and P_1 value. Various information sources have variable P_1 values, where typically these are in the range of 0.5 - 0.75, as discussed in [35]. In this paper results are presented for a wide range of P_1 values from 0.1 to 0.9.
- **Finite Source Size** A finite source size will require to be chosen for simulation, where in our case a symbol dictionary size of L = 1000 is chosen for the simulations presented in this work, in order to remain consistent with prior work [34].

Parametric analysis on the depth parameter has shown that the truncation gained from having a shorter depth has minimal impact on performance for source dictionaries when L = 1000. Through inspection of the transitions of lower stages in the tree it can be observed that even for large and infinite cardinality sources a transition is rarely observed. As these lower stages are where the gain would be expected, a larger depth has little overall performance gain. However, if a receiver has a large computational power, a system designer may still choose to implement a higher 'depth' parameter to observe good performance from the RExpGEC component code. Furthermore, EXIT chart analysis shows that the EXIT charts presented in Section III-C for a depth of 1 are indistinguishable from those with higher depths.

For simulations we use a fading channel with Rayleigh distribution in our simulations, and present performance results as a function of E_b/N_o (which enables a direct comparison between different source distributions and their respective information entropies).

C. EXIT CHART ANALYSIS

Within this section, we conduct detailed analysis on the iterative nature of the RExpGEC-URC-QPSK scheme and estimate its performance using EXIT chart analysis [47], [57],

[58] to gain an enhanced understanding of the transfer of information between the component codes of the RExpGEC-URC-QPSK scheme. The EXIT charts are presented in Figures 11 to 13. The quality of the information being transferred between the RExpGEC and URC component codes is quantified via the measurement of the mutual information [59] of the LLR values being passed between the component codes and their corresponding bit values from the transmitter. We analyse the mutual information at $\tilde{\mathbf{z}^{\mathbf{a}}}$ referred to as I_a and that of $\tilde{\mathbf{z}^{\mathbf{e}}}$ referred to as I_e, which represents the apriori and extrinsic information to and from the RExpGEC component code. This enables an enhanced understanding of how the mutual information of the LLRs increases in an iterative manner.

In this section, EXIT functions of both the RExpGEC and URC will be presented on the same axes in order to represent how information will iterate between the RExpGEC and URC codes. It is important to note that RExpGEC functions are inverted EXIT functions, as the extrinsic information I_e becomes the apriori information I_a of the URC code.

The transformation of I_a into I_e by the trellis decoder of Figure 10 is characterised by plotting the inverted RExpGEC EXIT function in an EXIT chart [57], as shown in Figures 11 to 13. In this case the inverted RExpGEC EXIT curve reaches the (1, 1) point in the top right corner of the EXIT chart [60]. Since the URC decoder also has an EXIT curve that reaches the (1, 1) point in the top right corner of the EXIT chart [61] as shown in Figures 11 to 13, iterative decoding convergence towards the Maximum Likelihood (ML) performance of the RExpGEC-URC-QPSK scheme is facilitated [62], [63].

The RExpGEC-URC-QPSK scheme may be said to operate near-capacity operation if reliable communication can be maintained at transmission throughputs that approach the Continous-input Continuous-output Memoryless Channel (CCMC) capacity C [49] that is associated with M = 4 QPSK modulation in an uncorrelated Rayleigh fading channel. Previous work on EXIT charts [42] have shown that the proposed scheme will offer near capacity performance if the the URC decoder of Figures 11 to 13 has an EXIT curve with an area beneath it of $A^i = C/[R_o log_2(M)]$ and the area A^o beneath the inverted EXIT curve of the RExpGEC trellis decoder in Figures 11 to 13 approaches the RExpGEC coding rate R_o.

If these two conditions are satisfied, then near-capacity operation will be achieved, when the shape of URC decoder's EXIT curve is closely matched to that of the inverted RExpGEC EXIT curve. This creates a narrow, but marginally open EXIT chart tunnel, which facilitates iterative decoding convergence. This narrow EXIT tunnel can be observed in Figures 11 to 13, where the EXIT tunnels are characterised for various values of k.

The EXIT chart area A_o that is situated below the inverted RExpGEC EXIT curve is given by [33], [34], [42]

$$A_o = \frac{1}{n} \sum_{m'=1}^{r} \sum_{m=1}^{r} P(m|m') log_2(\frac{1}{P(m|m')}), \qquad (5)$$

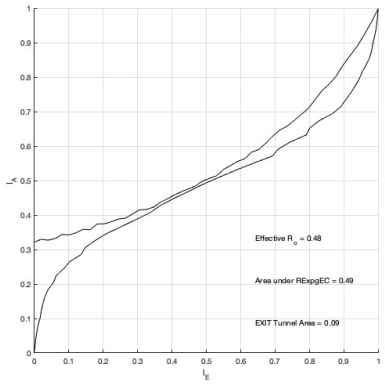

FIGURE 11. EXIT chart demonstrating the EXIT tunnel opening paramaters for $k = 0$ (equivalent to the REGEC of [34] for the inverted RExpGEC EXIT function against the corresponding URC EXIT function at variable E_b/N_o, P_1 = 0.6 L = 1000, SNR = 0.9 dB, CCMC capacity = -0.4 dB.

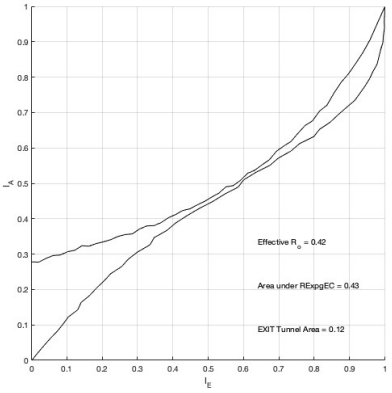

FIGURE 12. EXIT charts demonstrating the EXIT tunnel opening paramaters for $k = 1$ for the inverted RExpGEC EXIT function against the corresponding URC EXIT function at variable E_b/N_o, P_1 = 0.6 L = 1000, SNR = 0.4 dB, CCMC capacity = -0.1 dB.

where $P(m|m')$ represents the probability of a transition within the decoder, and $\mathbf{r}$ is the maximum number of states within the decoder.

In Figures 11 to 13 the EXIT function of the URC and the inverted EXIT function of the RExpGEC are shown for a given SNR when the EXIT tunnel is first presented as open. At this point, the RExpGEC-URC-QPSK scheme should be expected to start to converge [63]. It is important to note that the RExpGEC code with a k paramater of 0, as shown in Figure 11 is a special case of the RExpGEC that enables the RExpGEC to be functionally equivalent to the REGEC code presented in [34]. Therefore, this can be viewed as a benchmarker for a JSCC in comparison to the RExpGEC component code of the RExpGEC-URC-QPSK scheme.

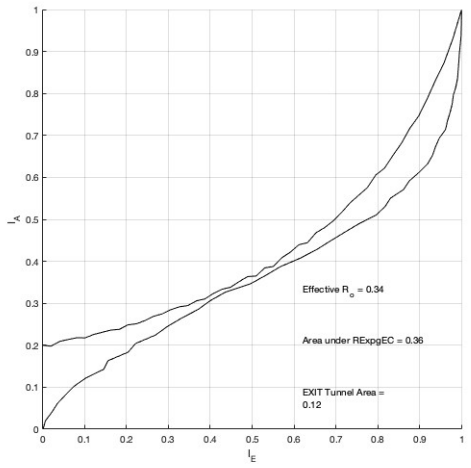

FIGURE 13. EXIT charts demonstrating the EXIT tunnel opening paramaters for $k = 2$ for the inverted RExpGEC EXIT function against the corresponding URC EXIT function at variable E_b/N_o, P_1 = 0.6 L = 1000, SNR = 0.6 dB, CCMC capacity = 0.1 dB.

As shown in Figures 11 to 13, the A_o approaches the effective coding rate R_o of the RExpGEC at the value of $P_1 = 0.6$. This shows that the 2-state URC code choice of [34] also offers near optimal performance in the RExpGEC-URC-QPSK scheme.

The EXIT analysis shows that the URC choice of the two state URC for the RExpGEC-URC-QPSK scheme is a good match for near-capacity performance and demonstrates efficiency in its design. The EXIT charts also provide a strong indication of the potential performance of a ML decoder for the proposed RExpGEC-URC-QPSK scheme which will be further explored via simulation in the following section.

IV. RESULTS AND ANALYSIS

In this section, we characterize the Symbol Error Rate (SER) performance of the RExpGEC-URC-QPSK scheme and compare it with two benchmarker schemes. The first benchmarker scheme is a SSCC scheme which uses the ExpG code concatenated with a convolutional and URC channel codes and then QPSK modulated. The second benchmarker is the REGEC-URC-QPSK scheme from [34], which represents a special case of the RExpGEC-URC-QPSK scheme where k=0 is the only supported parameter value. We will show that our proposed RExpGEC-URC-QPSK scheme offers superior performance in comparison to both the SSCC and JSCC benchmarker both in terms of absolute near-capacity performance, but also in flexibility of design, specifically the ability to match different source distributions more efficiently in comparison to the REGEC-URC-QPSK benchmarker.

In the ExpG-CC-URC-QPSK scheme shown in Fig. 14, the source symbol vector $\mathbf{x}$ is encoded into a ExpG bit vector, which is encoded with a convolutional code [18] interleaved via π_1 and encoded with an URC encoder (of the same code rates as that used in the RExpGEC-URC-QPSK scheme) to produce a vector of ExpG-CC-URC bits $\mathbf{y}$, and then mapped to QPSK modulation for transmission. This enables a direct comparison with the RExpGEC-URC-QPSK scheme as the ExpG-CC-URC-QPSK scheme is the SSCC equivalent.

The ExpG-CC-URC-QPSK benchmarker scheme will observe the same coding rate R_o as the RExpGEC-URC-QPSK scheme as introduced in Figure 10. Furthermore, as the ExpG-CC-URC-QPSK scheme is based upon the Exponential Golomb codeword, it can be scaled to different values of k for direct comparison with the RExpGEC-URC-QPSK scheme for the values of k which are simulated. The complexity of the ExpG-CC-URC-QPSK benchmarker scheme is relatively modest in comparison with the 154 add, compare, and select operations per iteration of the RExpGEC-URC-QPSK (where k=1 and depth=1), with a fixed complexity of 42 add, compare, and select operations per iteration. This is due to the fixed decoder structure of the SSCC scheme, where the decoder does not vary with k. Specifically in the case where k=1 and depth=, the RExpGEC-URC-QPSK scheme is 3.6 times more complex than the ExpG-CC-URC-QPSK benchmarker scheme.

In order to evaluate the performance of the RExpGEC-URC-QPSK scheme in comparison with the ExpG-CC-URC-QPSK scheme, simulations were conducted using well motivated values of P_1, k, and L, which are the same as those discussed in Section III-B whereby the performance of both systems can be directly compared. Specifically 100 iterations were applied to all schemes, with early termination criteria applied if the vector $\hat{\mathbf{x}}$ matched the transmitted symbol vector $\mathbf{x}$.

Figure 15 characterizes the performance of the RExpGEC-URC-QPSK scheme introduced in Section III in Figure 10, and provides direct comparison with the benchmarker ExpG-CC-URC-QPSK scheme introduced in Section IV in Figure 14. The scenario shown in Figure 15 is one snapshot of a series of simulation parameters, with other results presented in Figures 16 and 17.

As shown in Figure 15, reliable transmission can be achieved for zeta distributed sources with a P_1 of 0.6 at 1.7 dB for $k = 0$, 1.9 dB for $k = 1$, and 2.2 dB for $k = 2$. This performance constantly outperforms the benchmarker SSCC scheme and is constantly within 2.1 dB of the CCMC capacity for this specific case. Note, CCMC capacity of differing k parameters varies due to the different operating spectral efficiencies with regards to E_b/N_o.

As can be seen in Figure 15 the RExpGEC-URC-QPSK scheme is close to capacity for these specific cases. In figs. 16 and 17, the results from several different parameter values are presented, specifically including results on the variation of P_1 and k, with performance shown in both E_b/N_o as well as channel SNR.

The parametric performance of the RExpGEC-URC-QPSK and ExpG-CC-URC-QPSK schemes with respect to P_1 with performance measured in E_b/N_o and SNR are presented in Figures 16 and 17, respectively. As can be observed in Figure 16 the performance of the RExpGEC-URC-QPSK outperforms the ExpG-CC-URC-QPSK SSCC benchmarker

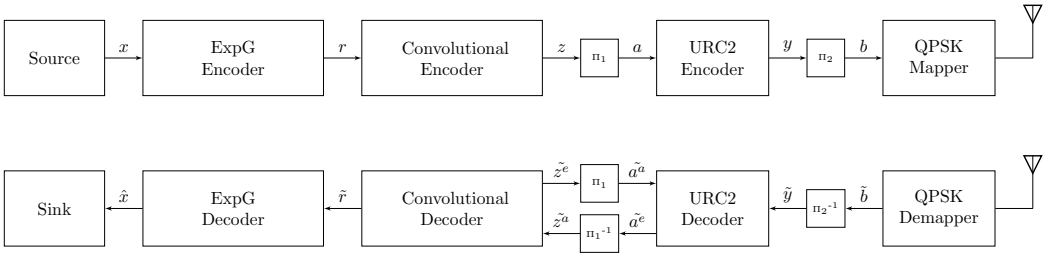

FIGURE 14. ExpG-CC-URC-QPSK Block Diagram.

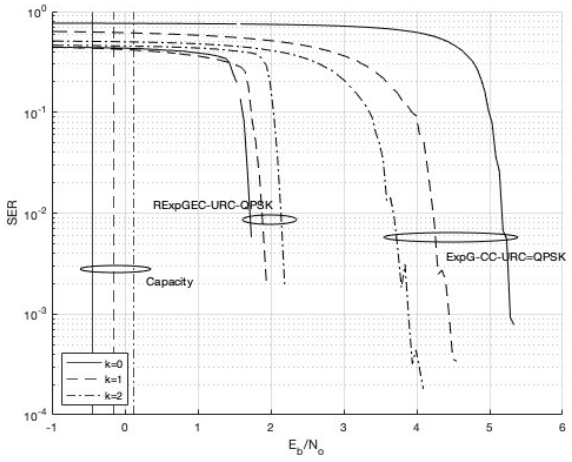

FIGURE 15. Symbol Error rate of the RExpGEC-URC-QPSK and ExpG-CC-URC-QPSK, for variable k parameters with the block length of $\mathbf{x}$ = 10000, P_1 of 0.6, and L = 1000.

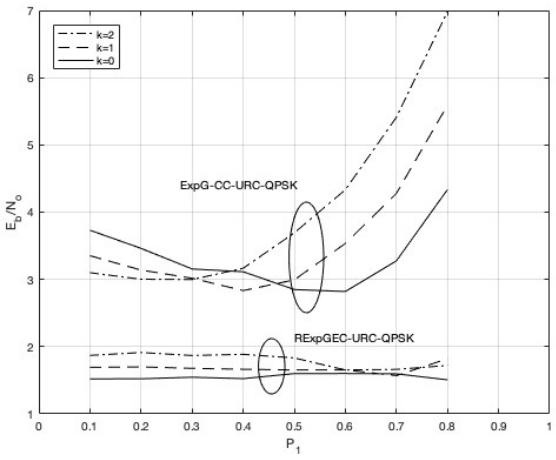

FIGURE 16. E_b/N_o of the RExpGEC-URC-QPSK and ExpG-CC-URC-QPSK, for variable P_1 parameters with the block length of $\mathbf{x}$ = 10000, L = 1000, and k = 0,1,2.

for all cases with regards to E_b/N_o across various P_1, and offers similar performance for all k parameters, which includes the JSCC benchmarker of the REGEC-URC-QPSK. The gap to CCMC capacity does not exceed 2.73 dB for any scenario and in the vast majority of cases is below 2 dB.

In comparison for link-budget constrained deployments where SNR is the important performance metric the results shown in Figure 17 are relevant. In this case not only does the RExpGEC-URC-QPSK still outperform the SSCC benchmarker but dependant on P_1 it can also offer better performance than the REGEC-URC-QPSK JSCC benchmarker. This demonstrates the flexibility of the RExpGEC-URC-QPSK. In all cases, all RExpGEC-URC-QPSK schemes with varying k outperforms all ExpG-CC-URC-QPSK by at least 1 dB.

In these cases, the flexibility of the RExpGEC-URC-QPSK scheme offers significant benefit over the REGEC-URC-QPSK scheme of [34], which can be functionally seen as a special case of the RExpGEC-URC-QPSK scheme where k=0, due to the fact that the system can be designed to utilise a different k parameter, and have better SNR performance. This is particularly useful if the system designed

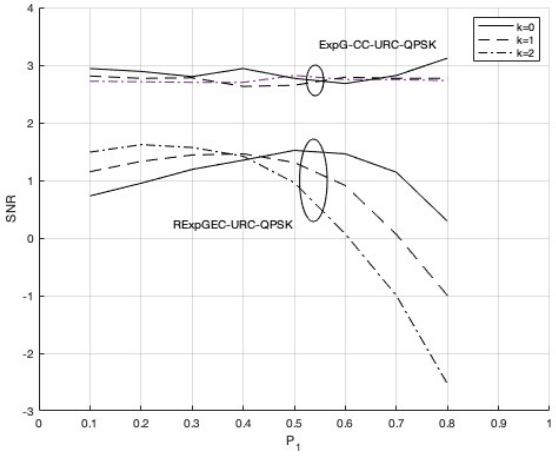

FIGURE 17. SNR of the RExpGEC-URC-QPSK and ExpG-CC-URC-QPSK, for variable P_1 parameters with the block length of $\mathbf{x}$ = 10000, L = 1000, and k = 0,1,2.

is constrained by the link budget, whilst maintaining a tight energy efficiency criteria as the system maintains a similar energy efficiency in terms of energy per bit for all values of k and P_1, whereas the SNR performance varies and the system can be optimised accordingly.

V. CONCLUSIONS

In this paper, we have introduced a novel JSCC code, known as the RExpGEC, which when integrated into the novel RExpGEC-URC-QPSK scheme provides near-capacity transmission of symbol values that are selected from large or infinite monotonic source distributions.

The RExpGEC-URC-QPSK scheme has enhanced flexibility over its JSCC counterpart such as the REGEC-URC-QPSK scheme, whilst maintaining the same performance level for k=0. The RExpGEC-URC-QPSK scheme enables enhanced performance for other values of the k parameter and maintains a gap to the CCMC capacity of 2 dB for all values of P_1 for zeta-like source probability distributions, when QPSK modulation is employed for transmission over an uncorrelated narrowband Rayleigh fading channel.

In some practical scenarios where the source symbols obey particular finite Zeta-like source probability distributions, our RExpGEC-URC-QSPK scheme is shown to offer gains of up to 3.6 dB over SSCC benchmarkers in all cases, when QPSK modulation is employed for transmission over an uncorrelated narrowband Rayleigh fading channel.

These gains are achieved for no-cost with regards to spectral usage and power, without increasing the required transmit-duration, transmit-bandwidth or transmit-energy. However, this is achieved at the cost of complexity of around 3.6 times compared to the SSCC benchmarker when k=1 and depth=1. We consider these performance gains to be significant, since they are achieved within the vicinity of the CCMC capacity, namely within 2 dB. This is achieved by mitigating the capacity loss inherent in SSCC, which is due to the residual redundancy after source coding which is not exploited for error correction. Furthermore the gains are achieved by being able to adjust the k parameter to target different monotonic sources in link-budget constrained scenrios. Since these gains are associated with the improvements offered by the RExpGEC code over the benchmarker SSCC and JSCC codes, similar gains may be expected when combining with any other channel codes.

REFERENCES

[1] Claude Elwood Shannon. A mathematical theory of communication. Bell Syst. Tech. J., 27:623–656, 1948.

[2] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. IEEE transactions on Information Theory, 24(5):530–536, 1978.

[3] Peter Elias. Universal codeword sets and representations of the integers. IEEE transactions on information theory, 21(2):194–203, 1975.

[4] David A Huffman. A method for the construction of minimum-redundancy codes. Proceedings of the IRE, 40(9):1098–1101, 1952.

[5] Solomon Golomb. Run-length encodings (corresp.). IEEE transactions on information theory, 12(3):399–401, 1966.

[6] Robert M Fano. The transmission of information, volume 65. Massachusetts Institute of Technology, Research Laboratory of Electronics . . . , 1949.

[7] J Brian Connell. A huffman-shannon-fano code. Proceedings of the IEEE, 61(7):1046–1047, 1973.

[8] Mathias Wien. High efficiency video coding. Coding Tools and specification, 24, 2015.

[9] Jukka Teuhola. A compression method for clustered bit-vectors. Information processing letters, 7(6):308–311, 1978.

[10] Shimon Even and Michael Rodeh. Economical encoding of commas between strings. Communications of the ACM, 21(4):315–317, 1978.

[11] Claude Berrou, Alain Glavieux, and Punya Thitimajshima. Near shannon limit error-correcting coding and decoding: Turbo-codes. 1. In Proceedings of ICC'93-IEEE International Conference on Communications, volume 2, pages 1064–1070. IEEE, 1993.

[12] Matthew F Brejza, Liang Li, Robert G Maunder, Bashir M Al-Hashimi, Claude Berrou, and Lajos Hanzo. 20 years of turbo coding and energy-aware design guidelines for energy-constrained wireless applications. IEEE Communications Surveys & Tutorials, 18(1):8–28, 2015.

[13] Erdal Arikan. Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. IEEE Transactions on information Theory, 55(7):3051–3073, 2009.

[14] Robert Gallager. Low-density parity-check codes. IRE Transactions on information theory, 8(1):21–28, 1962.

[15] Lajos Hanzo, Robert G Maunder, Jin Wang, and Lie-Liang Yang. Near-capacity variable-length coding: regular and EXIT-chart-aided irregular designs, volume 20. John Wiley & Sons, 2010.

[16] Stephen B Wicker and Vijay K Bhargava. Reed-Solomon codes and their applications. John Wiley & Sons, 1999.

[17] Samir Kallel and David Haccoun. Generalized type ii hybrid arq scheme using punctured convolutional coding. IEEE transactions on communications, 38(11):1938–1946, 1990.

[18] Jerold Heller and Irwin Jacobs. Viterbi decoding for satellite and space communication. IEEE Transactions on Communication Technology, 19(5):835–848, 1971.

[19] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. Journal of the society for industrial and applied mathematics, 8(2):300–304, 1960.

[20] Ben Krause, Liang Lu, Iain Murray, and Steve Renals. Multiplicative lstm for sequence modelling. arXiv preprint arXiv:1609.07959, 2016.

[21] SR Kodituwakku and US Amarasinghe. Comparison of lossless data compression algorithms for text data. Indian journal of computer science and engineering, 1(4):416–425, 2010.

[22] Jorma Rissanen and Glen Langdon. Universal modeling and coding. IEEE Transactions on Information Theory, 27(1):12–23, 1981.

[23] Aviezri S Fraenkel and Shmuel T Kleinb. Robust universal complete codes for transmission and compression. Discrete Applied Mathematics, 64(1):31–55, 1996.

[24] David Salomon. Data compression: the complete reference. Springer Science & Business Media, 2004.

[25] James L Massey. Joint source and channel coding. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE ELECTRONIC SYSTEMS LAB, 1977.

[26] Jorma Rissanen and Glen G Langdon. Arithmetic coding. IBM Journal of research and development, 23(2):149–162, 1979.

[27] Q Stout. Improved prefix encodings of the natural numbers (corresp.). IEEE Transactions on Information Theory, 26(5):607–609, 1980.

[28] Margaret Bernard and Bhu Dev Sharma. Some combinatorial results on variable length error correcting codes. Ars Combinatoria, 25:181–194, 1988.

[29] Rainer Bauer and Joachim Hagenauer. Symbol-by-symbol map decoding of variable length codes. ITG FACHBERICHT, pages 111–116, 2000.

[30] Norbert Gortz. Iterative source-channel decoding using soft-in/soft-out decoders. In 2000 IEEE International Symposium on Information Theory (Cat. No. 00CH37060), page 173. IEEE, 2000.

[31] N Gortz. On the iterative approximation of optimal joint source-channel decoding. IEEE Journal on Selected Areas in Communications, 19(9):1662–1670, 2001.

[32] Robert G Maunder, Wenbo Zhang, Tao Wang, and Lajos Hanzo. A unary error correction code for the near-capacity joint source and channel coding of symbol values from an infinite set. IEEE Transactions on Communications, 61(5):1977–1987, 2013.

[33] Tao Wang, Wenbo Zhang, Robert G Maunder, and Lajos Hanzo. Near-capacity joint source and channel coding of symbol values from an infinite source set using elias gamma error correction codes. IEEE transactions on communications, 62(1):280–292, 2013.

[34] Tao Wang, Matthew F Brejza, Wenbo Zhang, Robert G Maunder, and Lajos Hanzo. Reordered elias gamma error correction codes for the near-capacity transmission of multimedia information. IEEE Access, 4:5948–5970, 2016.

[35] Matthew F Brejza, Tao Wang, Wenbo Zhang, David Al-Khalili, Robert G Maunder, Bashir M Al-Hashimi, and Lajos Hanzo. Exponential golomb and rice error correction codes for generalized near-capacity joint source and channel coding. IEEE Access, 4:7154–7175, 2016.

[36] Tao Wang. Elias Gamma Error Correction Code. PhD thesis, University of Southampton, 2016.

[37] Stephan Ten Brink. Convergence of iterative decoding. Electronics letters, 35(10):806–808, 1999.

[38] Norman L Johnson, Adrienne W Kemp, and Samuel Kotz. Univariate discrete distributions, volume 444. John Wiley & Sons, 2005.

[39] Gary J Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (hevc) standard. IEEE Transactions on circuits and systems for video technology, 22(12):1649–1668, 2012.

[40] Aria Haghighi and Lucy Vanderwende. Exploring content models for multi-document summarization. In Proceedings of human language technologies: The 2009 annual conference of the North American Chapter of the Association for Computational Linguistics, pages 362–370, 2009.

[41] Jeffery R Price and Majid Rabbani. Biased reconstruction for jpeg decoding. IEEE Signal Processing Letters, 6(12):297–299, 1999.

[42] Alexei Ashikhmin, Gerhard Kramer, and Stephan ten Brink. Extrinsic information transfer functions: model and erasure channel properties. IEEE Transactions on Information Theory, 50(11):2657–2673, 2004.

[43] Lalit Bahl, John Cocke, Frederick Jelinek, and Josef Raviv. Optimal decoding of linear codes for minimizing symbol error rate (corresp.). IEEE Transactions on information theory, 20(2):284–287, 1974.

[44] Shuai Shao, Peter Hailes, Tsang-Yi Wang, Jwo-Yuh Wu, Robert G Maunder, Bashir M Al-Hashimi, and Lajos Hanzo. Survey of turbo, ldpc, and polar decoder asic implementations. IEEE Communications Surveys & Tutorials, 21(3):2309–2333, 2019.

[45] Jason P Woodard and Lajos Hanzo. Comparative study of turbo decoding techniques: An overview. IEEE Transactions on vehicular technology, 49(6):2208–2233, 2000.

[46] Dariush Divsalar, Samuel Dolinar, and Fabrizio Pollara. Serial concatenated trellis coded modulation with rate-1 inner code. In Globecom'00-IEEE. Global Telecommunications Conference. Conference Record (Cat. No. 00CH37137), volume 2, pages 777–782. IEEE, 2000.

[47] S Ten Brink, J Speidel, and R-H Yan. Iterative demapping for qpsk modulation. Electronics letters, 34(15):1459–1460, 1998.

[48] Bernard Sklar et al. Digital communications, volume 2.

[49] John G Proakis and Masoud Salehi. Digital communications, volume 4. McGraw-hill New York, 2001.

[50] Wenbo Zhang, Matthew F Brejza, Tao Wang, Robert G Maunder, and Lajos Hanzo. Irregular trellis for the near-capacity unary error correction coding of symbol values from an infinite set. IEEE Transactions on Communications, 63(12):5073–5088, 2015.

[51] Zunaira Babar, Hung Viet Nguyen, Panagiotis Botsinis, Dimitrios Alanis, Daryus Chandra, Soon Xin Ng, and Lajos Hanzo. Serially concatenated unity-rate codes improve quantum codes without coding-rate reduction. IEEE Communications Letters, 20(10):1916–1919, 2016.

[52] Zunaira Babar, Hung Viet Nguyen, Panagiotis Botsinis, Dimitrios Alanis, Daryus Chandra, Soon Xin Ng, and Lajos Hanzo. Unity-rate codes maximize the normalized throughput of on–off keying visible light communication. IEEE Photonics Technology Letters, 29(3):291–294, 2016.

[53] Zunaira Babar, Mohd Azri Mohd Izhar, Hung Viet Nguyen, Panagiotis Botsinis, Dimitrios Alanis, Daryus Chandra, Soon Xin Ng, Robert G Maunder, and Lajos Hanzo. Unary-coded dimming control improves on-off keying visible light communication. IEEE Transactions on Communications, 66(1):255–264, 2017.

[54] Jie Hu, Mengyuan Li, Kun Yang, Soon Xin Ng, and Kai-Kit Wong. Unary coding controlled simultaneous wireless information and power transfer. IEEE Transactions on Wireless Communications, 19(1):637–649, 2019.

[55] Mohammed El-Hajjar and Lajos Hanzo. Exit charts for system design and analysis. IEEE Communications Surveys Tutorials, 16(1):127–153, 2014.

[56] L. Hanzo, O. Alamri, M. El-Hajjar and N. Wu. Near-Capacity Multi-Functional MIMO Systems: Sphere-Packing, Iterative Detection and Co-operation. John Wiley & Sons - IEEE Press, 2009.

[57] Stephan Ten Brink. Convergence behavior of iteratively decoded parallel concatenated codes. IEEE transactions on communications, 49(10):1727–1737, 2001.

[58] Joachim Hagenauer. The exit chart-introduction to extrinsic information transfer in iterative processing. In Signal Processing Conference, 2004 12th European, pages 1541–1548. IEEE, 2004.

[59] David JC MacKay. Information theory, inference and learning algorithms. Cambridge university press, 2003.

[60] Jörg Kliewer, Norbert Goertz, and Alfred Mertins. Iterative source-channel decoding with markov random field source models. IEEE Transactions on Signal Processing, 54(10):3688–3701, 2006.

[61] Robert G Maunder and Lajos Hanzo. Iterative decoding convergence and termination of serially concatenated codes. IEEE transactions on vehicular technology, 59(1):216–224, 2009.

[62] Dariush Divsalar, Hui Jin, and Robert J McEliece. Coding theorems for" turbo-like" codes. In Proceedings of the annual Allerton Conference on Communication control and Computing, volume 36, pages 201–210. University Of Illinois, 1998.

[63] G David Forney Jr. Convolutional codes ii. maximum-likelihood decoding. Information and control, 25(3):222–266, 1974.

a1.png

ALEXANDER HAMILTON (M'15) is a Senior Standarization Specialist at Nokia, where he is a 3GPP RAN Delegate as well as being a visiting post-graduate researcher at the School of Electronics and Computer Science, University of Southampton, England. His research interests include information theory, joint source/channel coding, iterative decoding, non-linear channels, and modulation techniques. He is a Chartered Engineer with the IET.

a2.png

MOHAMMED EL-HAJJAR (SM'14) is an associate professor at the School of Electronics and Computer Science, University of Southampton, England. He is the recipient of several academic awards and has published a Wiley-IEEE book, more than 80 papers, and numerous patents. His research interests include communications systems and networking design.

a3.png

ROBERT G. MAUNDER (S'03, M'06, SM'12) (rm@ecs.soton.ac.uk) received a B.Eng. in electronic engineering in 2003 and a Ph.D. in telecommunications in 2007 from the University of Southampton. In 2007, he began a lectureship at the University of Southampton. In 2013, he was promoted to associate professor and to professor in 2017. He was awarded Senior Member status of IEEE in 2012, Chartered Engineer status of IET in 2013, and Fellow status of IET in 2017.