UNIVERSITÀ DEGLI STUDI DI PADOVA

Facoltà di Ingegneria

Corso di laurea triennale in ingegneria dell' informazione

Single Parity Check Product Codes

Laureanda:
Ilaria Savorgnan

Relatore:
Prof. Nevio Benvenuto

Anno Accademico 2009-2010

Ringraziamenti

Desidero innanzitutto ringraziare di cuore i miei genitori, che mi hanno sempre sostenuta con fiducia e hanno condiviso ogni tappa del mio percorso.

Un grazie sincero alle persone che sono per me punti fermi e che mi sono state vicine con costanza e affetto.

Un ringraziamento va anche al Prof. Nevio Benvenuto per la disponibilità e le utili indicazioni.

Contents

Introduction

This work has the aim to introduce and delve the concept of product codes, which were introduced for the first time by Elias in 1954, and represent the first method capable of achieving error-free coding with a nonzero code rate (as the number of dimensions increase to infinity). Specifically, single parity check product codes (that is, a peculiar class of product codes in which the encoder adds one bit to a sequence of n information bits such that the resultant $(n+1)$-bit codeword has an even number of ones) will be analyzed in detail. In Section 1 product codes are introduced, developing the first order and high order checks. In Section 2 a concrete and visual construction of a product code is explained, together with a focus on Shannon's limit to capacity. Section 3 and Section 4 expose the two concept that are the basis of product codes: concatenated coding and iterative decoding; both are fully investigated, in order to have a detailed overview of product codes. Exploiting the knowledge acquired in the previous sections, Section 5 describes single parity check product codes, focuses on encoding, decoding, and performance.

1 An introduction to product codes

Product codes are serially concatenated codes which were introduced by Elias in 1954. The transmitted symbols are divided into so-called *information digits* and *check digits*. The customer who has a message to send supplies the information digits which are tramitted unchanged. Periodically the coder at the transmitter computes some check digits, which are functions of past information digits, and transmits them [1].

Since these coding procedures are derived by an iteration of simple error-correcting and detecting codes, their performance depends on what kind of code is iterated. In 1954, Elias suggested that for a binary channel with a small and symmetric error probability, the best choice among the available procedures is the *Hamming-Golay single-error-correction double-error-detection code* developed by Hamming [2] for the binary case and extended by Golay [3] to the case of symbols selected from an alphabet of M different symbols, where M is any prime number.

1.1 First order check

Consider a noisy binary channel, which transmits either a zero or a one, with a probability (1 - p_0) that the symbol will be received as transmitted, and a probability p_0, that it will be received in error. Error probabilities for successive symbols are assumed to be statistically independent. Let the receiver divide the received symbol sequence into consecutive blocks, each block consisting of N_1 consecutive symbols. Because of the assumed independence of successive transmission errors, the error distribution in the blocks will be binomial: there will be a probability:

$$P(0) = (1 - p_0)^{N_1} \tag{1}$$

that no errors have occurred in a block, and a probability $P(i)$:

$$P(i) = \frac{N_1!}{i!(N_1 - i)!} p_0^i (1 - p_0)^{N_1 - i} \tag{2}$$

that exactly i errors have occurred.

If the expected number of errors per received block, $N_1 p_0$, is small, then the use of a *Hamming error-correction code* will produce an average number of errors per block, $N_1 p_1$, after error correction, which is smaller still. Thus p_1, the average probability of error per position after error correction, will be less than p_0.

The *single-error-correction* check digits of the Hamming code give the location of any single error within the block of N_1 digits, permitting it to be corrected. If more errors have occurred, they give a location which is usually not that of an incorrect digit, so that altering the digit in that location will usually cause one new error, and cannot cause more than one. The *double-error-detection* check digit tells the receiver whether an even or an odd number of errors has occurred. If an even number has occurred and an error location is indicated, the receiver does not make the indicated correction, and thus avoids what is very probably the addition of a new error.

The *single-correction double-detection* code, therefore, will leave error-free blocks alone, will correct single errors, will not alter the number of errors when it is even, and may increase the number by at most one when it is odd and greater than one. This gives for the expected number of errors per block after checking:

$$\sum_{even(i)\geq 2}^{\leq N_1} iP(i) + \sum_{odd(i)\geq 3}^{\leq N_1} (i+1)P(i) \tag{3}$$

$$\leq P(2) + \sum_{i=3}^{\leq N_1} (i+1)P(i) \tag{4}$$

$$\leq \sum_{i=0}^{N} (i+1)P(i) - P(0) - 2P(1) - P(2) \tag{5}$$

$$\leq 1 + N_1 p_0 - P(0) - 2P(1) - P(2). \tag{6}$$

Substituting the binomial error probabilities from (2), expanding and collecting terms, gives, for $N_1 p_0 \leq 3$:

$$N_1 p_1 \leq N_1(N_1 - 1)p_0^2 \tag{7}$$

or:

$$p_1 \leq (N_1 - 1)p_0^2 < N_1 p_0^2 \tag{8}$$

The error probability per position can therefore be reduced by making N_1 sufficiently small. The shortest code of this type requires $N_1 = 4$ and the inequality (8) suggests that a reduction will therefore not be possible if $p_0 \geq \frac{1}{3}$. The fault is in the equation, however, and not the code: for $N_1 = 4$ it is a simple majority-rule code which will always produce an improvement for any $p_0 < \frac{1}{2}$. A Hamming single-correction double-detection code uses C of the N positions in a block for checking purposes and the remaining $N - C$ positions for the customers symbols, where:

$$C = \lceil \log_2(N-1) + 2 \rceil \tag{9}$$

1.2 Higher order checks

After completing the first-order check, the receiver discards the C_1 check digits, leaving only the $(N_1 - C_1)$ checked information digits, with the reduced error probability p_1 per position. (It can be shown that the error probability after checking is the same for all N_1 positions in the block, so that discarding the check digits does not alter the error probability per position for the information digits).

Now some of these checked digits are made use of for further checking, again with a Hamming code. The receiver divides the checked digits into blocks of N_2; the C_2 checked check digits in each block enable it, again, to correct any single error in the block, although multiple errors may be increased by one in number. In order for the checking to reduce the expected number of errors per second-order block, however, it is necessary to select the locations of the N_2 symbols in the block with some care.

The simplest choice would be to take several consecutive first-order blocks of $(N_1 - C_1)$ adjacent checked information digits as a second-order block, but this is guaranteed not to work. For if there are any errors at all left in this group of digits after the first-order checking, there are certainly two or more, and the second-order check cannot correct them. In order for the error probability per place after the second-order check to satisfy the analog of (8), namely:

$$p_j \leq (N_j - 1)p_{j-1}^2 < N_j p_{j-1}^2 \tag{10}$$

It is necessary for the N_2 positions included in the second-order check to have statistically independent errors after the first check has been completed. This will be true if, and only if, each position was in a different block of N_1 adjacent symbols for the first-order check. The simplest way to guarantee this independence is to put each group of $N_1 \times N_2$ successive symbols in a rectangular array, checking each row of N_1 symbols by means of C_1 check digits, and then checking each column of already checked symbols by means of C_2 check digits. The transmitter sends the $(N_1 - C_1)$ information digits in the first row, computes the C_1 check digits and sends them, and proceeds to the next row. This process continues down through row $(N_2 - C_2)$. Then the transmitter computes the C_2 check digits for each column and writes them down in the last C_2 rows. It transmits one row at a time, using the first $(N_1 - C_1)$ of the positions in that row for the second-order check, and the last C_1 digits in the row for a first-order check of the second-order check digits [1].

2 The product codes concept

The concept of product codes is very simple and relatively efficient for building very long block codes by using two or more short block codes. Let us consider two systematic linear block codes $\mathcal{C}^1$ with parameters (n_1, k_1, δ_1) and $\mathcal{C}^2$ with parameters (n_2, k_2, δ_2), where n_i, k_i and δ_i (i= 1,2) stand for codeword length, number of information bits, and minimum Hamming distance, respectively. The product code $\mathcal{P} = \mathcal{C}^1 \otimes \mathcal{C}^2$ is obtained by:

1. placing $(k_1 \times k_2)$ information bits in an array of k_1 rows and k_2 columns;
2. coding the k_1 rows using code $\mathcal{C}^2$;
3. coding the n_2 columns using code $\mathcal{C}^1$;

The parameters of the product code $\mathcal{P}$ are $n = n_1 \times n_2$, $k = k_1 \times k_2$, $\delta = \delta_1 \times \delta_2$, and the code rate R is given by $R = R_1 \times R_2$, where R_i is the code rate of code $\mathcal{C}^i$. Thus, we can build very long block codes with large minimum Hamming distance by combining short codes with small minimum Hamming distance. Given the procedure used to construct the product code, it is clear that the $(n_2 - k_2)$ last columns of the matrix in Fig.1 are codewords of $\mathcal{C}^1$. By using the matrix generator, one can show [4] that the $(n_1 - k_1)$ last rows of matrix $\mathcal{P}$ are codewords of $\mathcal{C}^2$. Hence, all of the rows of matrix $\mathcal{P}$ are codewords of $\mathcal{C}^2$ and all of the columns of matrix $\mathcal{P}$ are codewords of $\mathcal{C}^1$ [5].

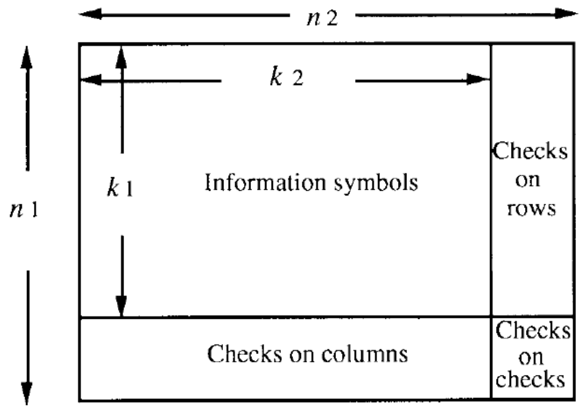


Figure 1: Construction of product code $\mathcal{P} = \mathcal{C}^1 \times \mathcal{C}^2$.

2.1 Construction of a product code

It is possible to combine the use of two or more codes so as to obtain a more powerful code. For example, a single parity check on a vector is capable of detecting all single errors. Now consider information symbols arranged in a rectangular array, with a single over-all parity check on each row and each column. This iteration of a simple parity-check code is capable of correcting all single errors, for if a single error occurs, the row and column in which it occurs are indicated by parity-check failures. In fact this code, which is a linear code, has minimum weight 4, the minimum weight code word having non-zero components at the intersections of two rows and two columns.

An important generalization results if each row of the array is a vector taken from one code and each column a vector from a different code. Product codes can also be generalized to three or higher dimensional arrays. These are all linear codes, and the generator matrix for the product of two codes is combinatorially equivalent to the tensor product of the generator matrices of the two original codes. It should be noted that certain symbols, such as those in the lower right-hand corners (see Fig.1) are checks on check symbols. These can be filled in as checks on rows and will be consistent as checks on columns, or viceversa. If they are filled in as checks on rows according to the parity-check rules for the row code, then each parity-check column is actually a linear combination of the columns that contain information symbols. Each of these has parity symbols added to it to make it a code vector, and therefore the parity-check columns, being linear combinations of code vectors for the column code, are also code vectors for the column code [6].

A multidimensional product code can be constructed in the following way. The data to be transmitted are arranged in a hypercube of dimension d with the length in each dimension defined by $\{k_1, k_2, ...k_d\}$. The i-th dimension is encoded with a systematic $(n_i, k_i, d_{min(i)})$ code, and this is repeated for all $i = 1, ...d$ dimensions. The order of encoding is not important. The resulting d-dimensional product code has block length:

$$N = \prod_{i=1}^{d} n_i \tag{11}$$

and code rate:

$$R = \prod_{i=1}^{d} r_i \tag{12}$$

where $r_i = \frac{k_i}{n_i}$ is the rate of the code in the i-th dimension. The two-dimensional (2-D) code consists of a data block, parity checks on the rows, checks on the columns, and checks on the checks [8]. The *single parity check code* is one of the most popular error detection codes because it is easy to implement. In these codes, the encoder adds one bit to a sequence of n information bits such that the resultant $(n+1)$-bit codeword has an even number of ones. Two or more SPC codes can be used to construct a product code [7]. We will consider only product codes formed from binary, systematic, linear component codes, specifically single parity check component codes which have the same length in each of the dimensions. The code rate and minimum distance are [8]:

$$R = \left(\frac{n-1}{n}\right)^d \tag{13}$$

$$d_{min} = 2^d \tag{14}$$

We shall then introduce the two concepts that are the basis of the single parity check product codes we are going to analyse: *concatenated coding* and *iterative decoding*. Let's first introduce Shannon bound on capacity [9].

2.2 The limits to capacity

In 1948 Claude Shannon was working at Bell Laboratories in the USA on the fundamental information transmission capacity of a communication channel. (In doing so he also rigorously quantified the concept of *information*, and thus founded the discipline of information theory.) He showed that a communication channel is in principle able to transmit information with as few errors as we wish, even if the channel is subject to errors due to noise or interference, provided the capacity of the channel is not exceeded. This capacity depends on the signal-to-noise ratio (SNR), the ratio of the signal power to noise power, as shown in Fig.2.

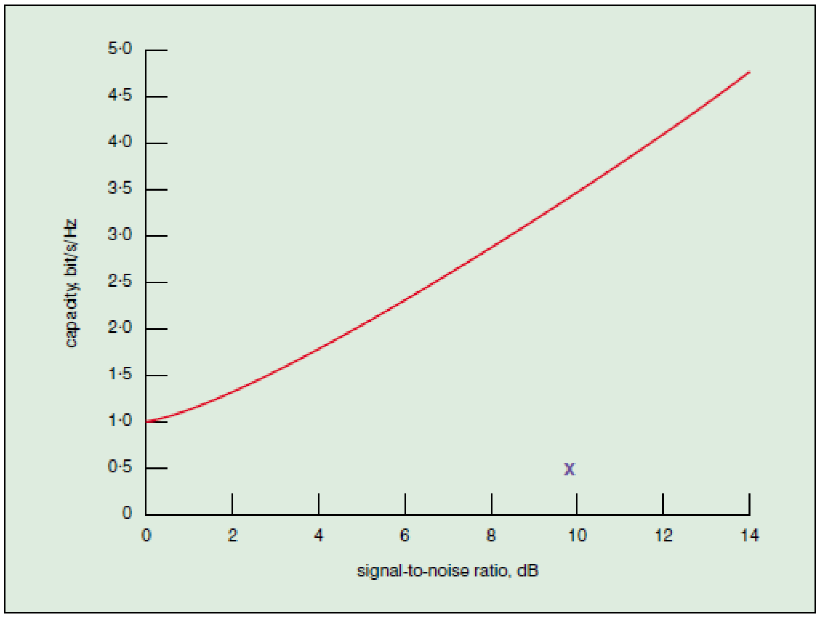


Figure 2: Shannon bound on capacity per unit bandwidth, plotted against signal-to-noise ratio. 'x' indicates the capacity and SNR requirements of BPSK for a BER of 10^{-3}.

Note that the capacity obtainable by conventional means is much less than this capacity limit. For example, the x mark on Fig.1 shows the performance achieved on a radio system with a simple modulation scheme: binary phase-shift keying (BPSK). This is for a bit error ratio (BER) of 0.001, which is low enough for only a few services, such as speech, whereas the Shannon theory promises an arbitrarily low BER. Note that at the same SNR a capacity several times greater could be achieved; or equivalently that the same capacity could be achieved with a signal power many decibels lower. This highlighted the potential gains available and led to the quest for techniques that could achieve this capacity in practice. Shannon did in fact also show in principle how to achieve capacity. The incoming data should be split into blocks containing as many bits as possible (say k bits). Each possible data block is then mapped to another block of n code symbols, called a *codeword*, which is transmitted over the channel. The set of codewords, and their mapping to data blocks, is called a *code*, or more specifically a *forward error correcting (FEC) code*. At the receiver there is a decoder, which must find the codeword that most closely resembles the word it receives, including the effects of noise and interference on the channel. The decoder is more likely to confuse codewords that resemble one another more closely: hence the power of the code to correct errors and overcome noise and interference depends on the degree of resemblance. This is characterised in terms of the minimum number of places in which any two codewords differ, called the *Hamming distance*.

Remarkably, Shannon showed that capacity could be achieved by a completely random code, that is a randomly chosen mapping set of codewords. The drawback is that this performance is approached only as k and n tend to infinity. Since the number of codewords then increases as 2^k, this makes the decoders search for the closest codeword quite impractical, unless the code provides for a simpler search technique.

3 Concatenated coding for product codes

The power of FEC codes increases with length k and approaches the Shannon bound only at very large k, but also the decoding complexity increases very rapidly with k. This suggests that it would be desirable to build a long, complex code out of much shorter *component codes*, which can be decoded much more easily. Concatenation provides a very straightforward means of achieving this. So the *serial and parallel concatenation* of codes is well established as a practical means of achieving excellent performance [10].

3.1 Serial concatenation

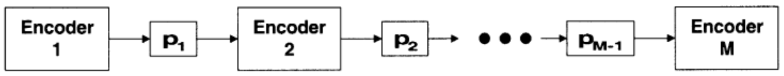


Figure 3: Principle of serial-concatenated codes.

The principle is to feed the output of one encoder (called the *outer encoder*) to the input of another encoder, and so on, as required. The final encoder before the channel is known as the *inner encoder* (Fig.3). The resulting composite code is clearly much more complex than any of the individual codes. This simple scheme suffers from a number of drawbacks, the most significant of which is called *error propagation.* If a decoding error occurs in a codeword, it usually results in a number of data errors. When these are passed on to the next decoder they may overwhelm the ability of that code to correct the errors. The performance of the outer decoder might be improved if these errors were distributed between a number of separate codewords. This can be achieved using an *interleaver — de-interleaver.* The simplest type of interleaver is illustrated in Fig.4.

This simple *interleaver* (sometimes known as a rectangular or block interleaver) consists of a two-dimensional array, into which the data is read along its rows. Once the array is full, the data is read out by columns, thus permuting the order of the data. (Because it performs a permutation, an interleaver is commonly denoted by the Greek letter π, and its corresponding de-interleaver by π^{-1}.) The original order can then be restored by a corresponding *de-interleaver*: an array of the same dimensions in which the data is read in by columns and read out by rows. This *interleaver*

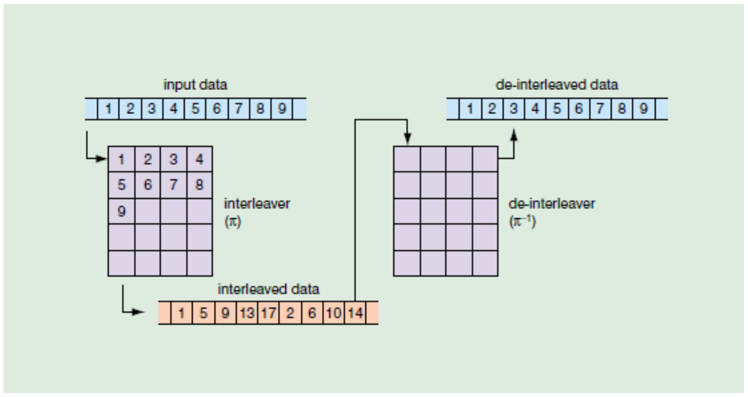


Figure 4: Operation of interleaver and de-interleaver.

may be placed between the outer and inner encoders of a concatenated code that uses two component codes, as for 2-D single parity check product codes as shown in Fig.5. Then, provided the rows of the interleaver are at least as long as the outer codewords, and the columns at least as long as the inner data blocks, each data bit of an inner codeword falls into a different outer codeword. Hence, provided the outer code is able to correct at least one error, it can always cope with single decoding errors in the inner code.

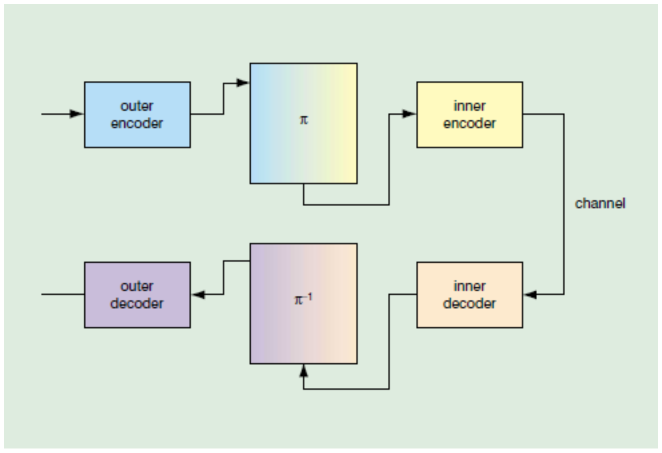


Figure 5: Concatenated encoder and decoder with interleaver.

Usually the block codes used in such a concatenated coding scheme are *systematic*: that is, the k data bits appear in the codeword, along with $n-k$ parity or check bits (1 check bit for SPCPC), which allow the data bits to be corrected if errors occur, making a codeword of length n. Now suppose the *outer code* has data length k_1 and code length n_1, while the *inner code* has data length k_2 and code length n_2, and the *interleaver* has dimension k_2 rows by n_1 columns. Then the parity and data bits may be arranged in an array as shown in Fig.6.

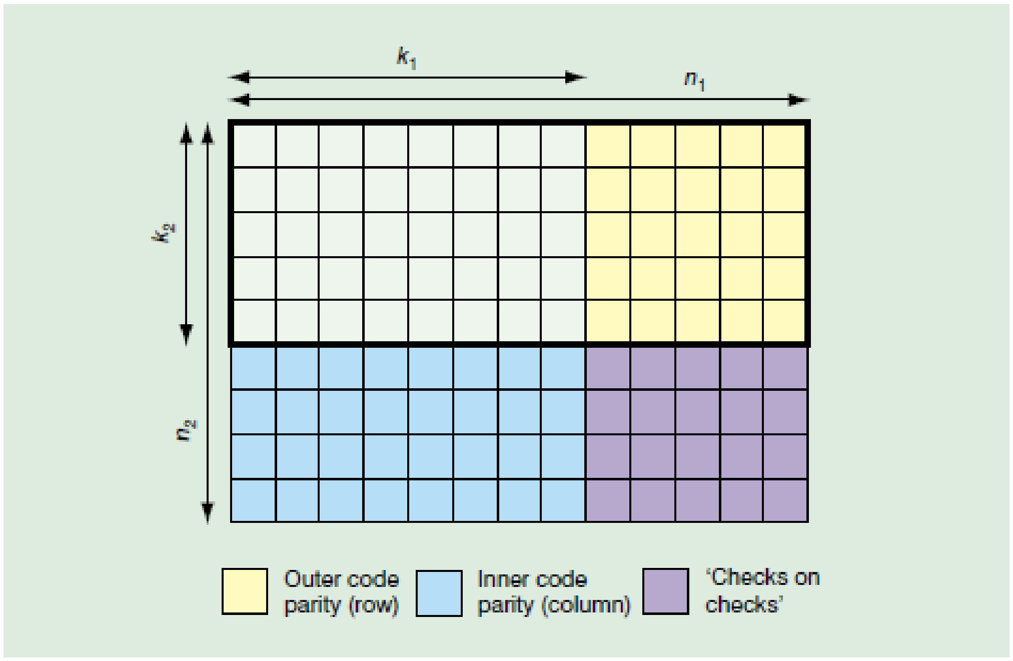


Figure 6: Array for interleaved concatenated code.

Part of this array (within the heavy line) is stored in the interleaver array: the *rows* contain *codewords of the outer code*. The parity of the inner code is then generated by the outer encoder as it encodes the data read out of the interleaver by columns. This includes the section of the array generated by encoding the parity of the outer code in the inner code, marked "checks on checks" in the figure. The *columns* of the array are thus *codewords of the inner code*. Observe that the composite code is much longer, and therefore potentially more powerful, than the component codes: it has data length $k_1 \times k_2$ and overall length $n_1 \times n_2$.

3.2 Parallel concatenation

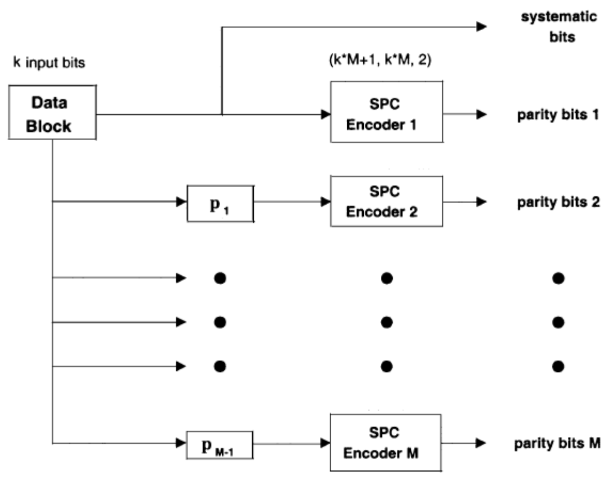


Figure 7: Principle of parallel-concatenated codes.

There is an alternative connection for concatenated codes, called *parallel concatenation* (Fig.7), in which the same data is applied to M encoders in parallel, but with $M-1$ interleavers between them, as shown in Fig.8(a) for $M=2$:

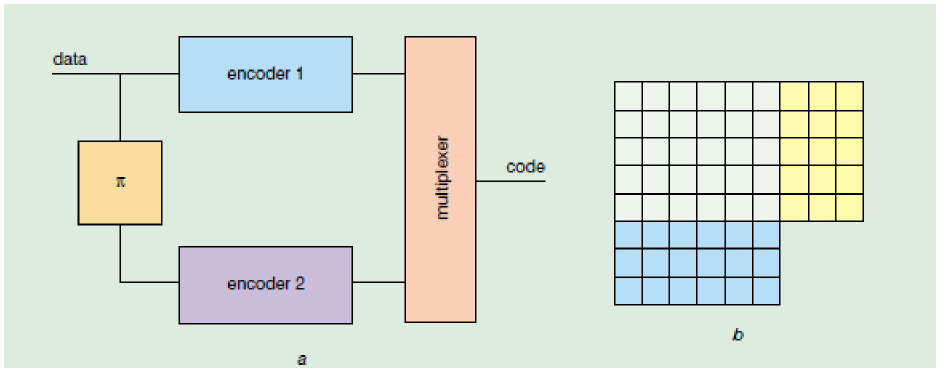


Figure 8: Parallel concatenation: (a) encoder structure; (b) code array.

If systematic block codes and a rectangular interleaver are used, as in Section 3.1, but the systematic component of the second code output is not transmitted (since it is duplicated), then the code array is as shown in Fig.8(b). It is essentially the same as in Fig.6, except that the "checks on checks" are not present [9].

4 Iterative decoding for product codes

In Section 2.1 we introduced two concepts that are the basis of single parity check product codes: *concatenated coding* and *iterative decoding*.

Concatenated coding has been analysed in Section 3. In this section we introduce the concept of *iterative decoding*. Then, in Section 5, it will be shown how iterative decoding can be used to realize a parallel decoder structure for single parity check product codes.

The conventional decoding technique for *product codes* is that shown in Fig.5: the inner code is decoded first, then the outer. However, this may not always be as effective as we might hope [9].

Consider a received codeword array with the pattern of errors shown by the Os in Fig.9. Suppose that both component codes are capable of correcting single errors only. As mentioned above, if there are more errors than this the decoder may actually introduce further errors into the decoded word. For the pattern shown this is the case for two of the column codewords, and errors might be added as indicated by X. When this is applied to the outer (row) decoder some of the original errors may be corrected (indicated by a cross through the O), but yet more errors may be inserted (marked with +). However, the original pattern would have been decoded correctly had it been applied to the row decoder first, since none of the rows contains more than one error.

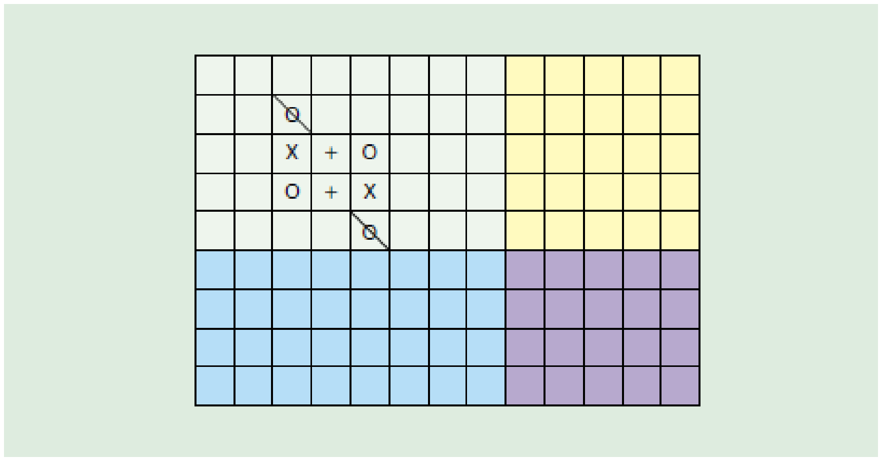


Figure 9: Pattern of received errors (O) in codeword array, with errors introduced by inner (column) decoder (X) and outer (row) decoder (+).

Note that if the output of the outer decoder were reapplied to the inner decoder it would detect that some errors remained, since the columns would not be codewords of the inner code. (A codeword of a single error correcting code must contain either no errors or at least three.) This in fact is the basis of the *iterative decoder*: to reapply the decoded word not just to the inner code, but also to the outer, and repeat as many times as necessary. However, it is clear from the foregoing argument that this would be in danger of simply generating further errors. One further ingredient is required for the iterative decoder: this ingredient is *Soft-in, soft-out* (SISO) decoding [9].

4.1 Soft-in, soft-out (SISO) decoding

The performance of a decoder is significantly enhanced if, in addition to the *hard decision* made by the demodulator on the current symbol, some additional *soft information* on the reliability of that decision is passed to the decoder. For example, if the received signal is close to a decision threshold (say between 0 and 1) in the demodulator, then that decision has low reliability, and the decoder should be able to change it when searching for the most probable codeword. In the decoder of a concatenated code the output of one decoder provides the input to the next. Thus to make full use of soft-decision decoding requires a component decoder that generates soft information as well as making use of it. This is the *SISO decoder*.

Soft information usually takes the form of a *log-likelihood ratio* for each data bit. The *likelihood ratio* is the ratio of the probability that a given bit is 1 to the probability that it is 0. If we take the logarithm of this, then its sign corresponds to the most probable hard decision on the bit (if it is positive, 1 is most likely; if negative, then 0).

$$\Lambda(b) = \ln\left(\frac{P(b=1|Y)}{P(b=0|Y)}\right) \tag{15}$$

Where b represents the transmitted data bits.

The absolute magnitude is a measure of our certainty about this decision. Subsequent decoders can then make use of this reliability information. It is likely that decoding errors will result in a smaller reliability measure than correct decoding. In the example this may enable the outer (row) decoder to correctly decode some of the errors resulting from the incorrect inner decoding. If not it may reduce the likelihood ratio of some, and a subsequent reapplication of the column decoder may correct more of the errors, and so on [9].

Thus we can regard the *log-likelihood ratio* as a measure of the total information we have about a particular bit. In fact this information comes from several separate sources. Some comes from the received data bit itself: this is known as the *intrinsic information.*

Information is also extracted by the two decoders from the other received bits of the row and the column codeword. When decoding one of these codes, the information from the other code is regarded as *extrinsic information.* It is this information that needs to be passed between decoders, since the intrinsic information is already available to the next decoder, and to pass it on would only dilute the extrinsic information. The most convenient representation for the concept of *extrinsic information* is as a log-likelihood ratio, in which case extrinsic information is computed as the difference between two log-likelihood ratios. In effect, extrinsic information is the incremental information gained by exploiting the dependencies that exist between a message bit of interest and incoming raw data bits processed by the decoder [12].

5 Single Parity Check Product Codes (SPCPC)

5.1 Encoding

Consider *parallel* SPCPC which have the same length in every dimension, as seen before the i-th component code can be defined as:

$$(n_i, k_i, \delta_i) = (kD + 1, kD, 2) \tag{16}$$

where:

- n_i stands for codeword length
- k_i stands for number of information bits
- δ_i stands for minimum Hamming distance
- D stands for dimension

i = 1,2....D

For a 2D code, it consists of *data block*, *parity checks on the rows*, *parity checks on the columns*, and *parity on parity checks*.

Therefore the *code rate* can be given as:

$$R = \left(\frac{n-1}{n}\right)^D \tag{17}$$

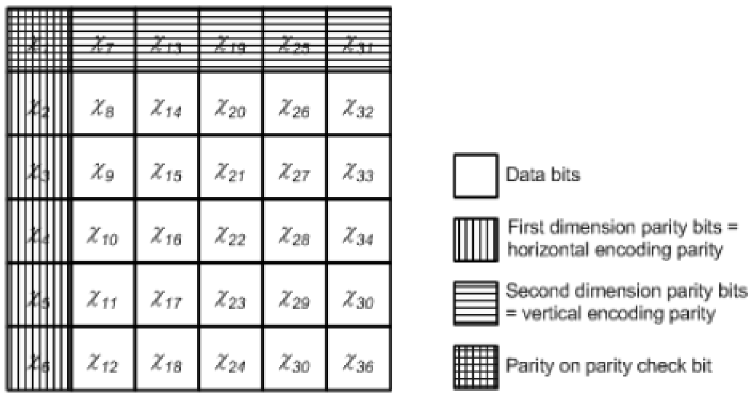
For a *2D-single parity check product code*, the *encoding* is performed by generating the even parity check bit for every rows and columns of the block code as illustrated in Fig.10.

This code consists of the data block, the parity checks on row and column also parity on parity check bits for $n_1 = n_2 = n = 6$.

The data is encoded using *dimensional based reading order (DBRO)*, to obtain several distinct of the codeword sequences. The first and second possible codeword sequences, X_1 and X_2 are obtained from *DBRO algorithm* which is given as:

$$X_l = \mathcal{X}(e_1, e_2)|e_1 = 1 + (l + n_1 - 1) mod n_1; e_2 = \left\lceil \frac{l}{n_1} \right\rceil \tag{18}$$

where the l-th bit of the 2D-SPCPC codeword sequence is the bit at (e_1, e_2) in two dimensional coordinate of codeword block $\mathcal{X}$ for l = 1,2..N.


Data bits
First dimension parity bits = horizontal encoding parity
Second dimension parity bits = vertical encoding parity
Parity on parity check bit

(a)

$$\mathbf{x}_1 = [\chi_1\ \chi_2\ \chi_3\ \chi_4\ \chi_5\ \chi_6\ \chi_7\ \chi_8\ \chi_9\ \chi_{10}\ \chi_{11}\ \chi_{12}\ \chi_{13}\ \chi_{14}\ \chi_{15}\ \chi_{16}\ \chi_{17}\ \chi_{18} \cdots$$
$$\chi_{19}\ \chi_{20}\ \chi_{21}\ \chi_{22}\ \chi_{23}\ \chi_{24}\ \chi_{25}\ \chi_{26}\ \chi_{27}\ \chi_{28}\ \chi_{29}\ \chi_{30}\ \chi_{31}\ \chi_{32}\ \chi_{33}\ \chi_{34}\ \chi_{35}\ \chi_{36}]$$

$$\mathbf{x}_2 = [\chi_1\ \chi_7\ \chi_{13}\ \chi_{19}\ \chi_{25}\ \chi_{31}\ \chi_2\ \chi_8\ \chi_{14}\ \chi_{20}\ \chi_{26}\ \chi_{32}\ \chi_3\ \chi_9\ \chi_{15}\ \chi_{21}\ \chi_{27}\ \chi_{33} \cdots$$
$$\chi_4\ \chi_{10}\ \chi_{16}\ \chi_{22}\ \chi_{28}\ \chi_{34}\ \chi_5\ \chi_{11}\ \chi_{17}\ \chi_{23}\ \chi_{29}\ \chi_{35}\ \chi_6\ \chi_{12}\ \chi_{18}\ \chi_{24}\ \chi_{30}\ \chi_{36}]$$

(b)

Figure 10: 2D-SPCPC Codeword, (a) two dimensions codeword block, (b) corresponding two possible codeword sequences.

N is the length of the codeword sequence and $\lceil x \rceil$ is ceil function that defines the smallest integer greater than x.

The detail of the *SPCPC encoder* block is illustrated in Fig.11.

The scramble information bits are divided into a data frame with length K for:

$$K = \prod_{d=1}^{D} k_d \tag{19}$$

and k_d are the length of component encoder input at dimension d-th.

In Fig.10, a *data block size* is $k_1 \times k_2$.

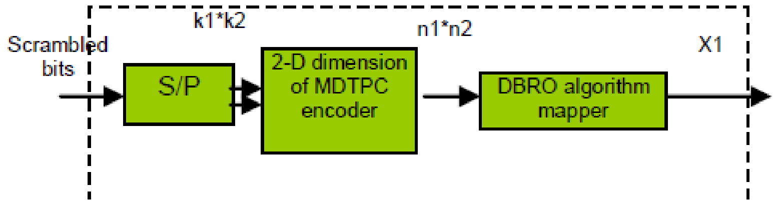


Figure 11: Detail of SPCPC encoder block. Turbo product code (TPC) consists of the product of two systematic block codes separated by uniform interleaver.

The 2D data block is encoded with identical SPC component codes of $(n_2, k_2, 2)$ and the *resulting codeword* is $n_1 \times n_2$. Then, using (17), the possible codeword sequence is selected [13].

5.2 Decoding

The decoding process of product codes is based on a suboptimal iterative processing in which each component decoder takes advantage of the extrinsic information produced by the other decoder at the previous step [14].

The parallel decoder structure proposed here is based on [11] with an extension of using weighting extrinsic information.

We then start from the general decoding structure in Fig.12, where each component soft decoder accepts the soft information from the demodulation process. This soft information is combined with a priori information from all other component decoders to generate *extrinsic information.* All soft information is properly interleaved and deinterleaved before being fed forward or backward.

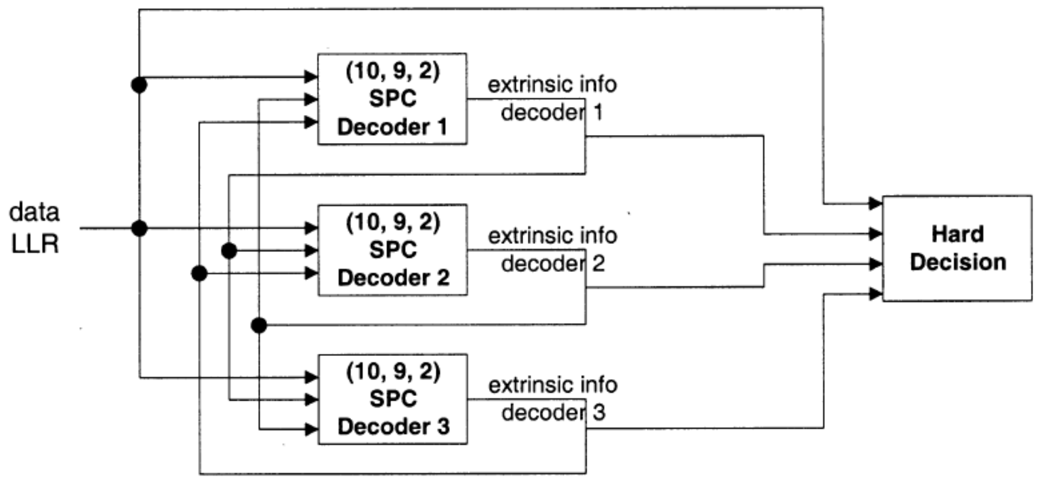


Figure 12: Decoder structure for a 3-PC-SPC code.

Then, we extend the decoder structure in Fig.12 by using *weighting extrinsic information* as illustrated in Fig.13 [13].

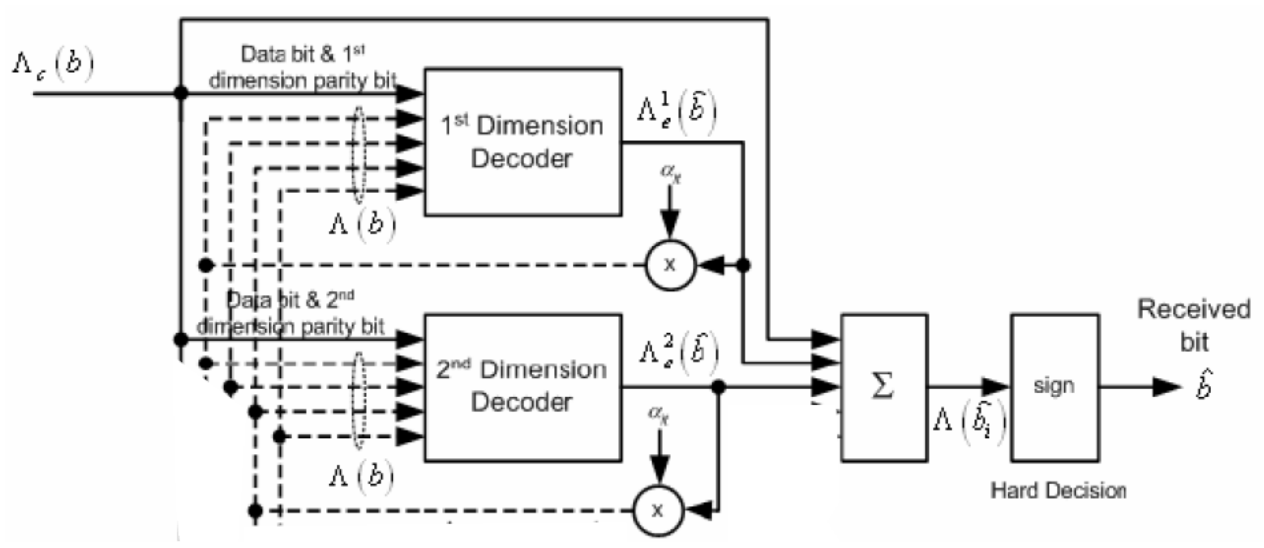


Figure 13: 2D-SPCPC parallel decoder with weighting extrinsic information.

The iterative decoding algorithm for SPC Product Codes is described below:

- *Initialization*: Calculate the channel log-likelihood ratios for all received symbols. Set the extrinsic information values Λ_e to zero for all bits in the product code and every dimension.
- *Decode each dimension*: Calculate the extrinsic information, Λ_e, for all bits in every SPC component code, over all dimensions. Only the extrinsic information is passed between the decoders in each dimension.
- *Repetition*: The decoding cycle, or iteration, is complete once all dimensions have been decoded. Repeat this decoding process for as long as required [8].

Then, for a 2-D SPC product codes, the decoding process starts by calculating the *log-likelihood ratio (LLR)* for each received bit as:

$$\begin{aligned}\Lambda(b) &= \ln\left(\frac{P(b=1|Y)}{P(b=0|Y)}\right)\\ &= \ln\left(\frac{p(Y|b=1)P(b=1)}{p(Y|b=0)P(b=0)}\right)\\ &= \ln\left(\frac{p(Y|b=1)}{p(Y|b=0)}\right)+\ln\left(\frac{P(b=1)}{P(b=0)}\right) \qquad (20)\end{aligned}$$

Where b represents the transmitted data bits.

At beginning the second term, which represent the *a-priori information*, is ignored for the assumption that all bits are equally likely.

The first term in (20) yields the so called *soft channel output LLR* or *channel output metric* $\Lambda_c(b)$:

$$\begin{aligned}\Lambda_c(b) &= \ln\left(\frac{p(Y|b=1)}{p(Y|b=0)}\right)\\ &= \ln\left(\frac{\frac{1}{\sigma\sqrt{2\pi}}e^{\left(\frac{(Y-HS_1)}{2\sigma^2}\right)}}{\frac{1}{\sigma\sqrt{2\pi}}e^{\left(\frac{(Y-HS_0)}{2\sigma^2}\right)}}\right)\\ &= \frac{1}{2\sigma^2}\left[(Y-HS_1)^2-(Y-HS_0)^2\right] \qquad (21)\end{aligned}$$

Where S_1 is the hypothesis representation of $b = 1$ and S_0 is the hypothesis representation of $b = 0$.

The Λ_c bits consist of LLR data bits that are passed to all component decoders and LLR parity bits are passed to the corresponding decoder.

We compute *extrinsic information*, Λ_e for the l-th data bit b_l using log likelihood algebra which is given as:

$$\Lambda_e(\hat{b_l}) = 2(-1)^{n_i}\operatorname{arctan}\left(\frac{\tanh(\Lambda(p))}{2}\prod_{j=1,j\neq l}^{n_i-1}\frac{\tanh(\Lambda(b_j))}{2}\right) \qquad (22)$$

where $\Lambda(p)$ is the LLR for the parity bit, and $\Lambda(b_j)$ is the LLR for the j-th data bit.

The *soft detected bits*, $\Lambda(\hat{b_l})$ is computed as:

$$\Lambda(\hat{b_l}) = \Lambda_c(\hat{b_l}) + \Lambda_e(\hat{b_l}) \tag{23}$$

The soft detected bits from all component decoder are summed up. Since the soft-detected bits have yielded from decoder, the first decoding iteration has been done. The received bits are obtained by applying *hard-decision detector* to the *soft detected bit*. For the next iteration, the *extrinsic information* from all component decoders is fed back to the input of all component decoders. The extrinsic information is used as a priori probability of detected bit. The decoding process will be terminated until a defined iteration [13].

The detail of the *SPCPC decoder* block is illustrated in Fig.14.

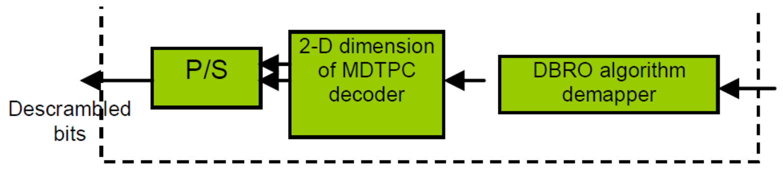


Figure 14: Detail of SPCPC decoder block.

5.3 Performance

The concatenated single parity check product codes have very good performance while the coding-decoding complexity is not high. It can be shown that the probability of bit error for a class of SPC product codes can be driven to zero as the block length tends to infinity within 2 dB of capacity on the additive white Gaussian noise (AWGN) channel [15].

In order to investigate performance on the AWGN channel, we first investigate the performance of single parity check product codes over the binary-symmetric channel (BSC) as an extension of the classic paper by Elias [1]. This approach has the advantage that numerical simulation can be used to drive the asymptotic analysis, and hence iterative decoding can be considered. The key to this analysis is the relationship between the probability of bit error before and after the decoding of each subcode. The overall code is analyzed by calculating the change in the average probability of bit error associated with the decoding of each subcode. In general, this type of analysis is quite simple provided the subcodes are independent. The simplest class of codes satisfying this criterion are product codes. In this case, the subcodes are the component codes of the product code, and each dimension forms a set of similar nonintersecting subcodes. For example, in a two-dimensional product code, the component codes form row and column codes within an array and the support of all the row codes form a nonintersecting set, likewise, the support of the column codes form a nonintersecting set. However, the natural construction of the product code ensures that the support of every row code intersects exactly once with the support of each column code (and thus the support of every column code also intersects exactly once with the support of every row code). Thus, product codes have the advantage that the probability of bit error can be recursively calculated as each subcode is decoded (in a particular order) for every subcode in the code. We will initially focus on the BSC in order to obtain the asymptotic analysis which will allow iterative decoding on the AWGN channel, specifically using numerical simulation to obtain an asymptotic result. The analysis of SPC product codes over the BSC can be cast in a form very similar to that for the extended Hamming "iterated codes", as studied by Elias [1] (see Section 1).

These iterated codes are simply product codes in which the length of the component code doubles with every dimension encoded. Hence:

$$N_i = 2N_{i-1} \tag{24}$$

where N_i is the length of the component code in the i-th dimension.

An extended Hamming product code can always correct a single error in each component code, but an SPC product code cannot correct any errors within a single component code over the BSC. The solution to this problem is to consider two consecutive dimensions of the SPC product code as a *Super Component (SC) code*, so that two-dimensional SPC product codes are the component codes of the overall SPC product code (the product code constructed from two-dimensional SPC product codes as component codes is also an SPC product code).

This SC code has minimum distance four and an extremely simple decoding algorithm which can be used to correct a single error (over the BSC). We will introduces the analysis of SPC product codes by first considering the asymptotic performance of extended Hamming iterated codes [1] on the binary symmetric channel; then, we will extend this analysis to SPC product codes.

5.3.1 Asymptotic performance of extended Hamming iterated codes on the binary symmetric channel

The extended Hamming codes have parameters $(2^m, 2^m - m - 1, 4)$ for some $m \geq 2$, and the overall code rate is simply the product of the component codes rates. Then, we find that the *asymptotic code rate* R is given by:

$$R = \prod_{j=m}^{\infty} \frac{2^j - j - 1}{2^j} = \prod_{j=m}^{\infty} \left(1 - \frac{j+1}{2^j}\right) \tag{25}$$

It can be shown that this code rate tends to a constant greater than zero [1]. Furthermore, R depends on the parameter m, where $m = \log_2 N_1$ and N_1 is the length of the component code in the first dimension.

It has been shown by Elias [1] that the probability of bit error $\mathcal{P}_i$ after decoding the extended Hamming code in the i-th dimension of a product code on the BSC is bounded by:

$$\mathcal{P}_i \leq N_i \mathcal{P}_{i-1}^2 \tag{26}$$

where $\mathcal{P}_{i-1}$ is the probability of bit error before decoding the i-th dimension and, initially, $\mathcal{P}_0 = \mathcal{P}$ is the crossover probability of the channel. The relationship given by (26) requires that bits decoded in the i-th dimension are indipendent with respect to the probability of bit error in the $(i-1)$-th dimension. The simplest code structure which satisfies this requirement is the product code.

Given that the length of the component codes double in each dimension and in view of (26), the optimal decoding order is the same as the order of encoding (shortest to largest block length). This is because the shortest codes will have the most success in reducing the probability of bit error [15].

Following Elias, we can recursively calculate the probability of bit error after decoding k dimensions using (24) and (26) to give:

$$\begin{aligned} \mathcal{P}_k &\leq \left(N_1 2^{k-1}\right)^{2^0} \ldots \left(N_1 2^{k-i}\right)^{2^{j-1}} \ldots \left(N_1 2^0\right)^{2^{k-1}} \mathcal{P}_0^{2^k} \\ &= \frac{1}{N_1} \left(2N_1 \mathcal{P}_0\right)^{2^k} 2^{-(k+1)} \end{aligned} \tag{27}$$

The right side of this expression approaches zero as k increases, provided the crossover probability $\mathcal{P}$ satisfies:

$$\mathcal{P} \leq \frac{1}{2N_1} \tag{28}$$

5.3.2 Asymptotic performance of single parity check product codes on the binary symmetric channel

We will extend the analysis in the previous subsection to single parity check product codes. Consider SPC product codes on the BSC as the number of dimensions tends to infinity. Note that an SPC component code cannot correct any errors on the BSC thus we need to consider the SC codes. In order to maximize the rate of the SC code, for a given block length, the SPC codes in each of the two dimensions are chosen to have the same length n_i. Therefore, $N_i = n_i^2$ and the *SC code rate* is given by:

$$R_i = \left(\frac{(n_i - 1)}{n_i} \right)^2 \tag{29}$$

Furthermore, in keeping with Elias' iterated code philosophy, the length of the SC code needs to approximately double in each dimension. N_i is the length of the SC code in the i-th dimension and the block length satisfies the recursion:

$$N_i = \left(\left\langle \sqrt{2N_{i-1}} \right\rangle \right)^2 \tag{30}$$

where $\langle x \rangle$ denotes the integer closest to x. Hence, $N_i \approx 2N_{i-1}$ and the approximation approaches equality as i increases.

The following algorithm is used to decode these SC codes:

- Find all rows and columns within the two-dimensional SPC product code such that the parity-check equations are unsatisfied.
- If, and only if, one row and one column equations are unsatisfied, flip the bit at the intersection of this row and column; otherwise, leave the received word unchanged.

Clearly, this algorithm has very low complexity and will correct all weight-one error patterns. More importantly, this algorithm will detect (and leave the received word unchanged), all other error patterns which are at more than Hamming distance one from any codeword. Thus, it will detect all even-weight error patterns, assuming the pattern is not a codeword. This is due to the fact that an SC code contains only even-weight codewords (hence, any even-weight error pattern which is not a codeword must be at least distance two from a codeword and, therefore, can be detected). The error detection properties of this algorithm are very advantageous. It will be shown that, unlike the extended Hamming decoder in [1], this decoding algorithm will detect the majority of weight three error patterns and, therefore, it will not incorrectly decode these received words (which would add an extra bit error).

To begin with, we decode the SC code without using this extra information. Note that in this case, the performance of the decoder is exactly the same as the extended Hamming decoder since it will attempt to correct all odd-weight error patterns and leave all even-weight error patterns [1]. Consequently, this decoder will correct all odd-weight error patterns (greater than weight one) to a wrong codeword and hence are assumed to add an extra bit error. Therefore, the expected number of bit errors remaining after decoding a single dimension is bounded by:

$$N_1\mathcal{P}_1 \leq \sum_{even(i)\geq 2}^{N_1} i\binom{N_1}{i}\mathcal{P}^i\mathcal{Q}^{N_1-i} + \sum_{odd(i)\geq 3}^{N_1}(i+1)\binom{N_1}{i}\mathcal{P}^i\mathcal{Q}^{N_1-i} \tag{31}$$

where $\mathcal{P}$ is the probability that any codeword bit is in error and $\mathcal{Q} = 1-\mathcal{P}$. We can write (31) in closed form, as shown in the following:

$$\begin{aligned}
N_1\mathcal{P}_1 &\leq \sum_{i=2}^{N_1} i\binom{N_1}{i}\mathcal{P}^i\mathcal{Q}^{N_1-i} + \sum_{odd(i)\geq 3}^{N_1}\binom{N_1}{i}\mathcal{P}^i\mathcal{Q}^{N_1-i} \\
&= \sum_{i=2}^{N_1} N_1\binom{N_1-1}{i-1}\mathcal{P}^i\mathcal{Q}^{N_1-i} + \left[\sum_{odd(i)\geq 1}^{N_1}\binom{N_1}{i}\mathcal{P}^i\mathcal{Q}^{N_1-i} - N_1\mathcal{P}\mathcal{Q}^{N_1-1}\right] \\
&= N_1\sum_{i=1}^{N_1-1}\binom{N_1-1}{i}\mathcal{P}^{i+1}\mathcal{Q}^{N_1-1-i} + \frac{1}{2}\left[(\mathcal{P}+\mathcal{Q})^{N_1} - (-\mathcal{P}+\mathcal{Q})^{N_1}\right] - N_1\mathcal{P}\mathcal{Q}^{N_1-1} \\
&= N_1\mathcal{P}\left[(\mathcal{P}+\mathcal{Q})^{N_1-1} - \mathcal{Q}^{N_1-1}\right] + \frac{1}{2}\left[(\mathcal{P}+\mathcal{Q})^{N_1} - (-\mathcal{P}+\mathcal{Q})^{N_1}\right] - N_1\mathcal{P}\mathcal{Q}^{N_1-1} \\
&= N_1\mathcal{P}\left[1 - 2\mathcal{Q}^{N_1-1}\right] + \frac{1}{2}\left[1-(1-2\mathcal{P})^{N_1}\right] \qquad (32)
\end{aligned}$$

This analysis extends directly to any dimension of the PC where $\mathcal{P}_i = \mathcal{P}_1, N_i = N_1, \mathcal{P}_{i-1} = \mathcal{P}$, provided the bit errors in the subcode to be decoded are independent. Note that using the bound $(1-\mathcal{P})^n > 1 - n\mathcal{P}$ reduces the analysis to the result given by Elias (26).

The relationship between the probability of bit error before and after decoding an SC code can be improved by considering the number of detectable weight-three error patterns. When a weight-three error pattern is detected, no decoding is attempted and hence no extra error is incurred, unlike the bound (32). All but $4\binom{n_1}{2}^2$ of the weight-three error patterns are detectable. This is due to the fact that all $\binom{n_1}{2}^2$ of the weight-four codewords are square patterns, as shown in Fig.15, and hence the only weight-three error patterns which are at distance one from the weight-four codewords are those obtained by removing a single vertex from these codewords [15].

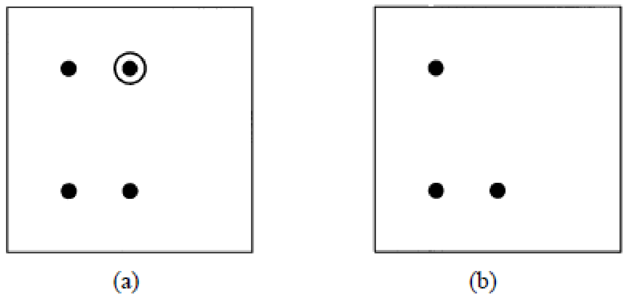


Figure 15: Relationship between weight-four codewords (a) and undetectable weight-three error patterns (b). The dots represent ones in the two-dimensional SPC Product codeword. Note that removing a vertex from a codeword creates an undetectable weight-three error pattern.

Using this extra information, we can calculate the expected number of bit errors after decoding the SC code as:

$$N_1\mathcal{P}_1 \leq N_1\mathcal{P}\left[1-2\mathcal{Q}^{N_1-1}\right]+\frac{1}{2}\left[1-(1-2\mathcal{P})^{N-1}\right]-\left(1-\frac{4\binom{n_1}{2}^2}{\binom{n_1^2}{3}}\right)\binom{N_1}{3}\mathcal{P}^3\mathcal{Q}^{N_1-3} \tag{33}$$

which is always better than (32), especially at high crossover probabilities, $\mathcal{P}$, as shown in Fig.16.

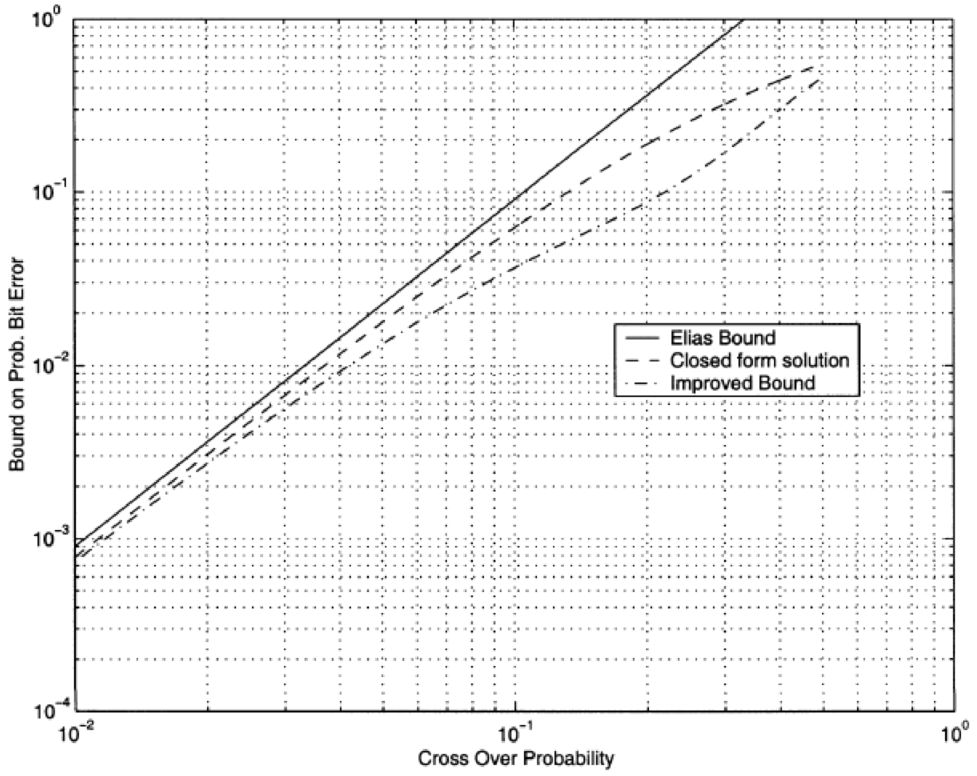


Figure 16: Improvement of (33) compared to the closed-form solution (32) and the bound (26) derived by Elias.

It is possible to use a seminumerical method for improving the bounds on asymptotic performance of an SPC product code, compared to Elias' threshold of $\frac{1}{2N_1}$.

The probability of bit error after decoding the i-th dimension $\mathcal{P}_i$ is calculated recursively using (33) and the probability of bit error after decoding the previous dimension, $\mathcal{P}_{i-1}$. Then $\mathcal{P}_i$ is compared to a threshold $\frac{1}{2N_{i+1}}$, to determine the point at which the asymptotic probability of bit error will tend to zero. This threshold is determined using the following variation of Elias' analysis.

Consider the decoding of an SPCPC in terms of the SC codes. Because the decoding of a single SC code satisfies (26), and assuming the block length satisfies $N_i = 2N_{i-1}$ (which is true for large i), the asymptotic probability of error can be determined. Specifically, we will only consider decoding dimensions j through k (where $1 \leq j \ll k$). Therefore:

$$\begin{aligned}
\mathcal{P}_k &\leq N_k (N_{k-1})^2 (N_{k-2})^4 \ldots (N_j)^{2^{k-j}} \mathcal{P}_{j-1}^{2^{k-j+1}} \\
&= \mathcal{P}_{j-1}^{2^{k-j+1}} \prod_{i=j}^{k} \left(N_j 2^{k-i} \right)^{2^{k-j-1}} \\
&= \frac{1}{N_j} \left(2N_j \mathcal{P}_{j-1} \right)^{2^{k-j+1}} 2^{-(k-j+2)} \qquad (34)
\end{aligned}$$

and, hence, $\mathcal{P}_k \to 0$ as $k \to \infty$ provided:

$$\mathcal{P}_{j-1} \leq \frac{1}{2N_j} \qquad (35)$$

Thus, the probability of bit error can be calculated numerically using (33) for each dimension up to the j-th, at which point the result is compared to the threshold (35) to determine the point at which the overall probability of bit error will tend to zero.

Fig.17 shows the performance of a SPC product code with $n_1 = 8$ and $N_1 = 64$, $N_2 = 121$, $N_3 = 256$, $N_4 = 529$, $N_5 = 1089$, $N_6 = 2209$.

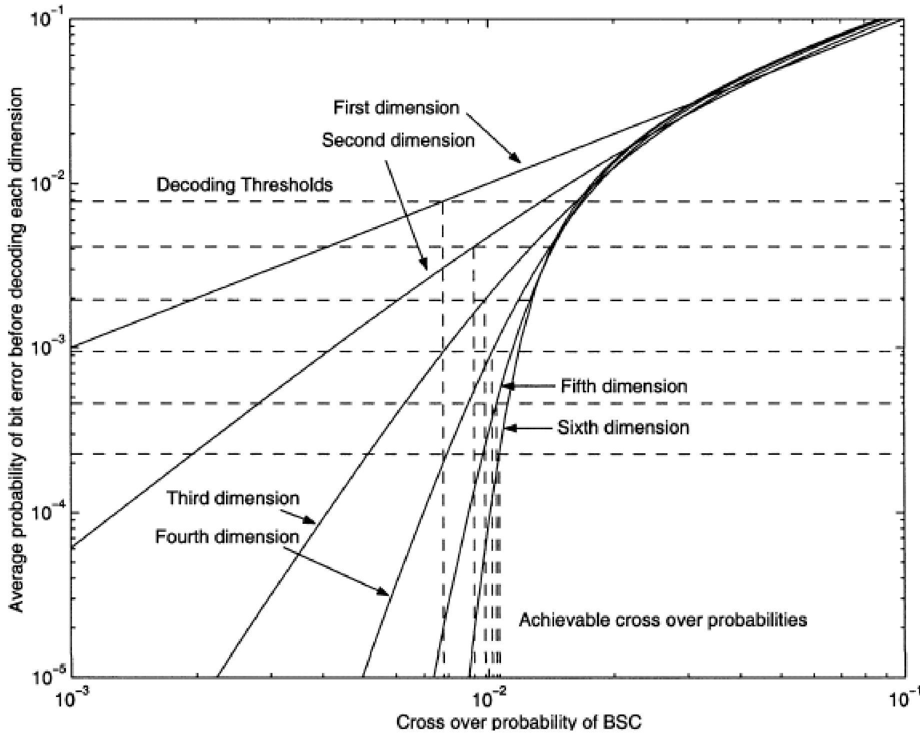


Figure 17: Performance of a SPC Product Code.

The abscissa is the crossover probability of the BSC, which is $\mathcal{P}_0$ and is also represented by the curve labeled *first dimension*. The curve labeled *second dimension* is the probability of bit error after decoding all the SC codes in the first dimension, using (33), and is, therefore, the input probability of bit error into the second dimension. The remaining curves follow the same pattern. Now the most important information about each of these curves is the point at which the probability of error intersects the threshold $\frac{1}{2N_j}$. All bit error probabilities less than or equal to this point will force the asymptotic probability of bit error to zero as the number of dimensions tends to infinity. The intersection is then mapped back to the original crossover probability to determine the maximum $\mathcal{P}$ such that the overall probability of bit error can be forced to zero. Note that this limit improves with every dimension considered (although the improvement diminishes with each dimension). Also note that this method constructs, by default, an upper bound on the performance of the code after decoding the appropriate number of dimensions.

By choosing different values of N_1, the maximum crossover probability such that $\mathcal{P}_k \to 0$ as $k \to \infty$ can be determined over a wide range of code rates. The results are shown in Fig.18 for both the original threshold of Elias (28), and the improved thresholds given in this subsection. These results clearly show that SPC product codes can, asymptotically, allow error-free transmission at nonzero code rates on the BSC [15].

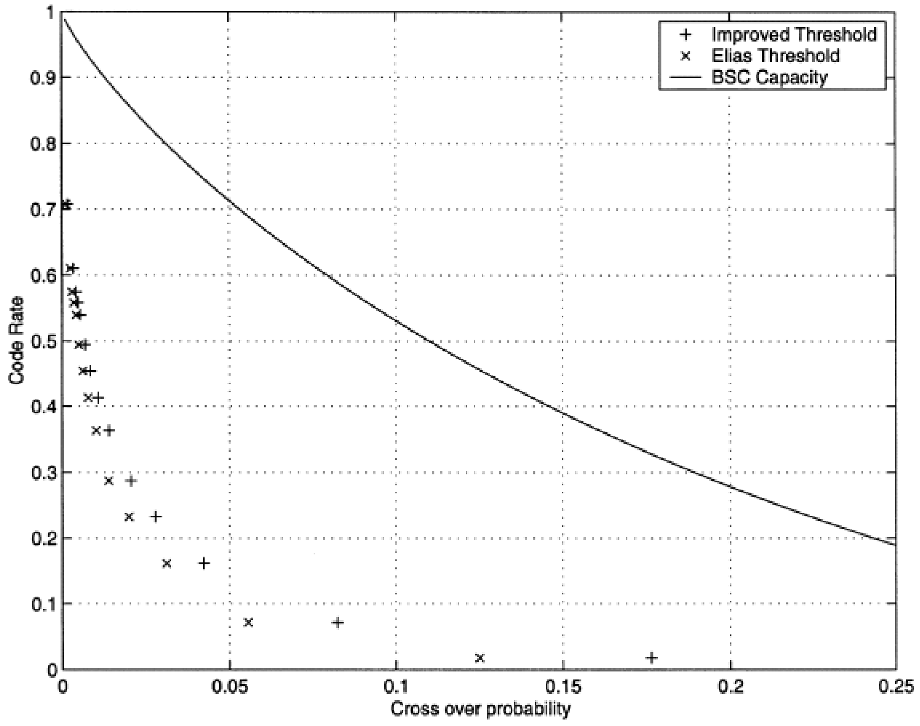


Figure 18: The maximum crossover probability and code rate such that the asymptotic probability of bit error can be forced to zero on the BSC.

5.3.3 Asymptotic performance of single parity check product codes on the AWGN channel

In this subsection we extend the previous results on the BSC to the AWGN channel. This is achieved by iteratively soft-decoding the first few dimensions then hard-decoding the remaining dimensions. Consequently, the asymptotic analysis can be applied provided the soft decoding in the first few dimensions can drive the probability of bit error below a specific threshold, as determined from the previous subsection. The soft-decoding of the first few dimensions will be based upon maximum a posteriori (MAP) decoding of the component SPC codes. The motivation behind this analysis is that the soft-decoding of the first few dimensions will reduce the probability of bit error much faster than the corresponding hard-decoding. Thus, the signal-to-noise ratio (SNR) at which the probability of bit error is less than the threshold (defined in the previous subsection) is the point at which, asymptotically, the probability of bit error can be driven to zero. At most, the first three dimensions of the SC codes will be soft-decoded, which corresponds to six dimensions of the SPC product code.

The use of iterative decoding to decode a dimension of a PC does not affect the independence of bit errors in higher dimensions. To show this is true, the *independence* needs to be viewed from the decoding point of view, specifically looking at the probability of bit error. Initially, the probability that any bit is received in error is independent of any other bit due to the memoryless nature of the channel. Therefore, decoding the PC in a single dimension will introduce dependencies in that dimension but not in higher dimensions. For example, in the two-dimensional case, decoding the rows will not introduce statistical dependency between the columns because the support of a row code intersects with only one element of the support of a column code. In fact, dependencies related to the probability of bit error only occur after a code is decoded (in a given dimension). Hence, repeated decoding of the rows in a two-dimensional code will not introduce any dependencies among the bit errors in the columns.

The results are shown in Fig.19 for iterative soft-decoding of up to the first three dimensions of the SPC product code (in terms of the SC codes). The codes correspond to $n_1 = 4, 5, 6, \ldots 13, 14, 20, 40$. Over a wide range of rates, these codes can asymptotically drive the probability of bit error to zero at SNRs within 2 dB of capacity, and even closer at higher code rates. The improvement in performance is due to *iterative soft-decoding* of the first few dimensions of the SPC product code [15].

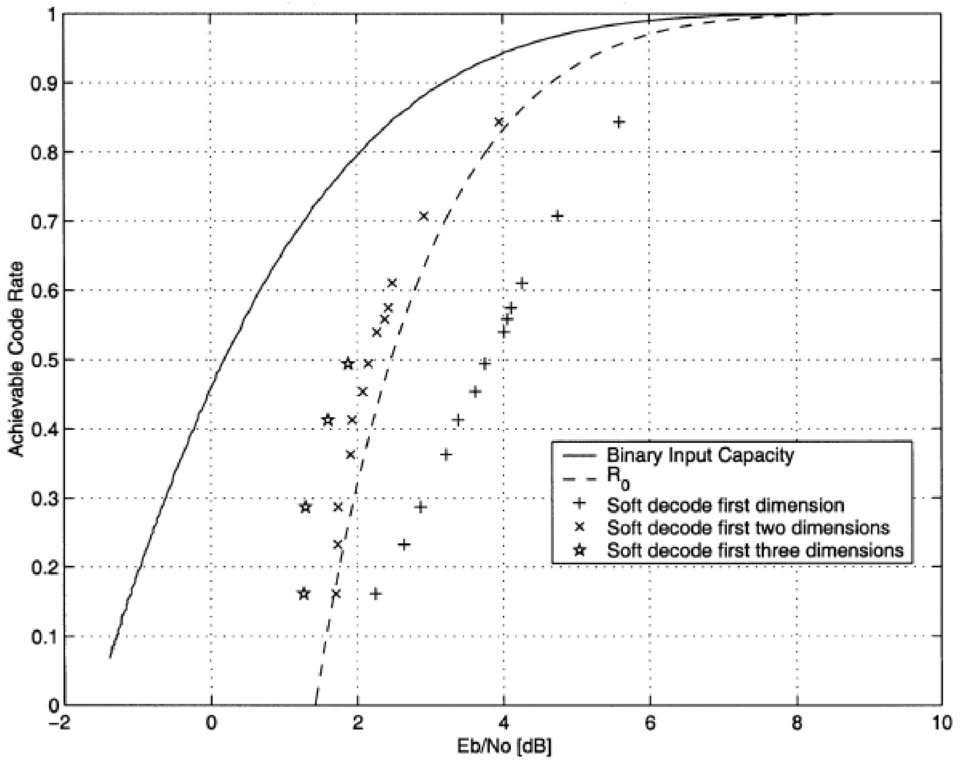


Figure 19: Asymptotic performance, defined as code rate versus $\frac{E_b}{N_0}$ such that $\mathcal{P}_b \to 0$, after iterative soft-decoding up to the first three (six) dimensions of the SPC product code and then hard-decoding the remaining dimensions over the binary input AWGN channel.

6 Conclusion

We have seen that product codes, and specifically single parity check product codes, are a class of codes based on *concatenated coding* and *iterative decoding*. These two fundamental concepts allow product codes to be very efficent and to force the probability of error to zero within 2 dB of capacity on a binary-input AWGN channel. We note that the resulting performance is surprisingly close to capacity on the AWGN channel, given the simplicity of these codes.

References

[1] P. Elias, “Error free coding”, *IRE Trans. Inform. Theory*, vol. IT-4, pp. 29-37, September 1954.

[2] R. W. Hamming, “Error Detecting and Error Correcting Codes”, *Bell System Tech.* vol. 29, pp. 147-161, April 1950.

[3] M. J. E. Golay, “Notes on Digital Coding”, *Proc. I.R.E.* vol. 37, pp. 657, 1949.

[4] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes.* Amsterdam, The Netherlands: North-Holland, vol. 16, pp. 567-580, 1978.

[5] R. M. Pyndiah, “Near-Optimum Decoding of Product Codes: Block Turbo Codes”, *IEEE Trans. on Commmun.*,vol. 46, pp. 1003-1010, August 1998.

[6] W. W. Peterson and E. J. Weldon Jr, *Error Correcting Codes*, 2nd ed. Cambridge, MIT Press, 1972.

[7] H. Xu and F. Takawira, “A New Structure of Single Parity Check Product Codes”, *IEEE Africon*, September 2004

[8] D. M. Rankin and T. A. Gulliver, “Single Parity Check Product Codes”, *IEEE Trans, on Commun.*, vol. 49, no. 8, pp. 1354-1362, August 2001.

[9] A. Burr, “Turbo Codes: the ultimate error control codes?”, *Journal on Electronics and Communication Engineering*, vol. 13, pp. 155-165, August 2001.

[10] M. Rankin, T. A. Gulliver, “Parallel and Serial Concatenated Single Parity Check Product Codes”, *EURASIP Journal on Applied Signal Processing*, pp. 775-783, January 2005.

[11] J. S. K. Tee, D. P. Taylor and P. A. Martin, “Multiple serial and parallel concatenated single parity check codes”, *IEEE Trans. on Commun.*, vol. 51, no. 10, pp. 1666-1675, October 2003.

[12] Y. Isukapalli, S. Rao, “Exploiting the Nature of Extrinsic Information in Iterative Decoding”, Department of Electrical and Computer Engineering, Villanova University, 2003.

[13] N. Ahmad, S. Yusof, N. Fisal, “Single Parity Check Product Code in MB-OFDM Ultra Wideband System”, *IEEE Trans. on Commun.*, Ultra Modern Telecommunications and workshops, 2009. ICUMT ’09. pp. 1-5, October 2009.

[14] G. Colavolpe, G. Ferrari, R. Raheli, “Extrinsic Information in Iterative Decoding: a Unified View”, *IEEE Trans. on Commun.*, vol. 49, no. 12, pp. 2088-2094, December 2001.

[15] M. Rankin, T.A. Gulliver, D.P. Taylor, “Asymptotic Performance of Single Parity Check Product Codes”, *IEEE Trans. on Commun.*, vol. 49, no. 9, pp. 2230-2235, September 2001.