S-72.333 Postgraduate Course in Radiocommunications
Fall 2000

Source and Channel Encoder and Decoder Modeling

Markku Pukkila
Nokia Research Center
markku.pukkila@nokia.com

HUT 12.12.2000

Contents

I. Introduction

This report is concentrating on source and channel coding issues and especially their modeling in simulation platforms. Source encoding and decoding issues are first described. A/D conversion is the key operation in the transmitter side, so an analog signal is sampled, quantized and then encoded into groups of k bits. In the receiver there is D/A conversion, so the discrete signal is decoded and filtered back to analog signal. Both uniform and nonuniform quantization methods are described as well as differential quantization, which is beneficial for correlated sequence. Also one method to encode discrete information is presented. The section concludes to some general remarks about how to simulate source coding systems.

Channel coding is used to achieve more reliable transmission by adding redundancy in the information bits. The most common channel encoding techniques are block and convolutional codes, although so called iterative turbo codes have recently got a lot of interest among researchers. The block encoders are processing a block of data at a time and calculating the parity check bits for this block after which they read the next input bits to replace the current block. Convolutional encoders instead introduce memory into the code by using the previous time instants as well to calculate the current encoder output. Convolutional decoders can be based for example on MLSE or MAP principles, the former solving minimum sequence error and the latter minimum symbol error. For instance in Full Rate GSM Speech both block and convolutional encoding are used, but on the other hand, the end of each burst is kept uncoded as it is not so critical for human's ear. Another methods to improve link performance is to use interleaving, nonbinary codes (more than 2 different symbols available) or concatenate at least two codes serially or parallel.

The coded system can be divided into encoder/decoder pair and a discrete channel between them. Typically more samples are needed to represent the channel part in a simulation. Sometimes it is enough to calculate bounds for a codec and one can so avoid long and complex simulations. At least with large SNR values one can get reliable bounds.

II. Source Coding [1]

In Fig. 1 a general source encoding and decoding system is presented. Analog information source is sampled and quantized and finally represented with binary (or M-ary) symbols $\underline{b}(kT_s)$. In the receiver side the received symbols $\hat{\underline{b}}(kT_s)$ are decoded and filtered back to analog signal. So there is an analog-to-digital (A/D) conversion in the transmitter and digital-to-analog (D/A) conversion in the receiver. Typical values in the case of speech are:

- 8000 analog samples per second
- each sample quantized into 2^k levels, $k \in [6,10]$
- each quantized sample encoded by a group of k binary values
- total bit rate between 48000 and 80000 bits per second
- decoder maps groups of k bits into 2^k levels and filters analog signal from that

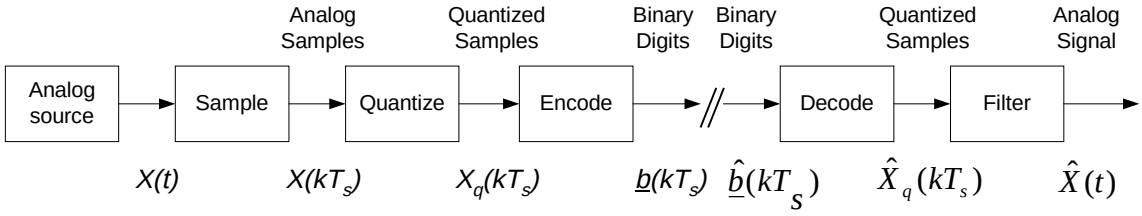

Figure 1. Block diagram for source coding system.

The *quantization* operation typically transforms an analog signal X into a discrete signal X_q. This operation introduces some quantization noise X_q-X into the system and the system designer tries to minimize the mean squared error $E\{(X-X_q)^2\}$. If the samples $X(kT_s)$ are uncorrelated, then they can be quantized independently, but otherwise some data processing is needed to remove correlation between samples before quantization.

A. Quantization

Classical *uniform* quantization divides the input voltage into equal intervals, thus ith interval corresponds to range $i\Delta\pm(\Delta/2)$. When input voltage falls into this range, quantizer gives an output integer i. *Nonuniform* quantizer divides the input range into unequal intervals. The input signal is first processed by a *compressor* and after that with a uniform quantizer. So called μ-law for a compressor gives $x_c=v_c/V$ as a function of $x=v/V$ as follows

$$x_c = (\operatorname{sgn} x)\frac{\log(1+\mu|x|)}{\log(1+\mu)} \tag{1}$$

where v_c is the compressor output, v the input, V the maximum input and μ is typically 255. So called A-law is given by

$$\begin{aligned} x_c &= (\operatorname{sgn} x)\frac{A|x|}{1+\ln A}\ ,\ 0 \le |x| \le A^{-1} \\ &= (\operatorname{sgn} x)\frac{1+\ln A|x|}{1+\ln A}\ ,\ \ A^{-1} \le |x| \le 1 \end{aligned} \quad . \tag{2}$$

The source decoder estimates x_c by performing simply inverse mapping from the integer i as follows

$$\hat{x}_c = (\operatorname{sgn} x)(\Delta/2 + i\Delta), \quad \Delta = 2^{-m},\ i = 0,1,...,2^m - 1 \quad . \tag{3}$$

Finally, an estimate on the original input is given by the *expander* as follows

$$\hat{x} = \operatorname{sgn}(\hat{x}_c) g^{-1}(\hat{x}_c) \quad , \tag{4}$$

where (sgn x)g(x) describes the used compressor law (1) or (2).

B. Differential quantization

If the sequence {$X(kT_s)$} is *correlated*, the variance of the difference $X(kT_s)$ - $X((k\text{-}1)T_s)$ is smaller than the original variance of $X(kT_s)$. Therefore it is beneficial to quantize the difference instead of the original sequence to achieve lower MSE. In general a *predictor* uses a linear combination of M previous values as follows

$$\hat{X}(k) = \sum_{i=1}^{M} a_i X(k-i)\,. \tag{5}$$

The predictor coefficients a_i can be solved from

$$R_{XX}(j) = \sum_{i=1}^{M} a_i R_{XX}(j-i) \quad . \tag{6}$$

In Fig. 2 transmitter and receiver with differential quantization are shown.

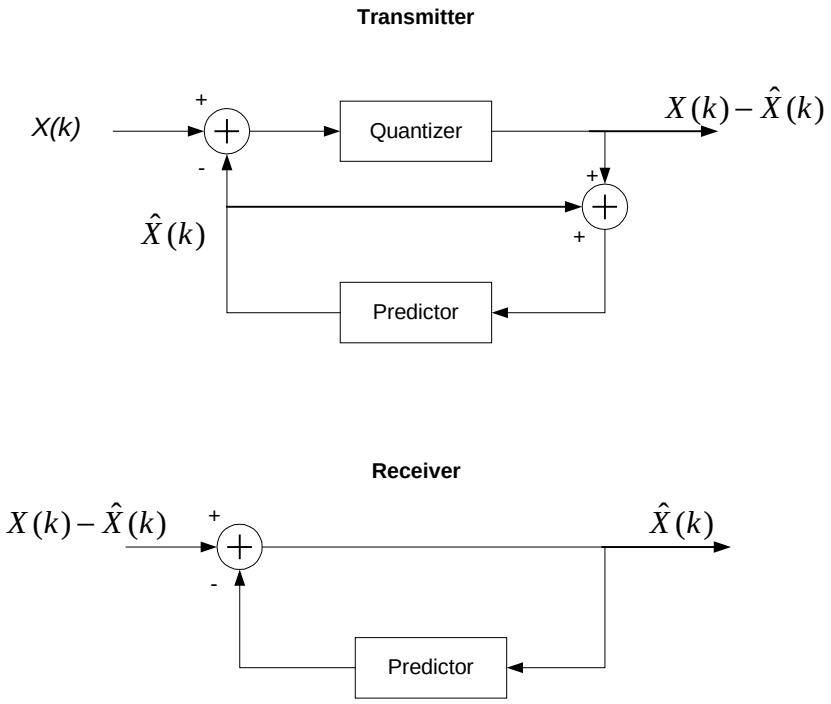

Figure 2. Differential encoding.

C. Encoding discrete information sources

In the case of *discrete information source*, there are only a limited number of possible symbols that could be transmitted. Source encoding is now transforming original information into a binary or M-ary sequence. In order to conserve system bandwidth it is beneficial that encoding utilises as low bit rate as possible and that the encoding is unique, hence the original message can be recovered without errors.

There are many possible algorithms to encode discrete source. Here is given an algorithm that was first proposed by Shannon and which maps *M-ary alphabet* into binary digits. Let us assume that the encoder input consists of blocks of N symbols and there are $q(N)$ possible messages of that length. Let $p_1,p_2,\ldots,p_{q(N)}$ be the probabilities of these messages, and $p_1+p_2+\ldots+p_{q(N)}=1$, $p_1 > p_2 >\ldots> p_{q(N)}$. Then the minimum average number of bits per symbol for these blocks is

$$H_N = \frac{1}{N}\sum_{i=1}^{q(N)} p_i \log_2(1/p_i)\,. \tag{7}$$

Shannon proposes the following encoding for the blocks of N symbols. Let

$$F_i = \sum_{k=1}^{i-1} p_k \quad \text{and} \quad F_1 = 0 \quad . \tag{8}$$

Let also n_i be a following integer

$$\log_2(1/p_i) \le n_i < 1+\log_2(1/p_i) \quad . \tag{9}$$

Then the binary encoding for message m_i is the binary expansion of F_i using up to n_i bits. This algorithm assigns *shorter codes for messages of higher probability*, which decreases the average number of bits per symbol.

D. Simulation of source coding

We can observe that *idealized A/D operation*, uniform or nonuniform quantization, is a simple memoryless transformation. So it is enough to read input value and calculate the corresponding output. Real A/D converters have many different properties, but very often idealized models will do. In the receiver end an inverse D/A conversion is performed. Usually first the binary sequence is mapped into quantized digital sequence and after that the sequence is filtered into an analog signal.

Hence, as we have presented source encoding and decoding steps, we can now assume that encoded binary sequence consists of ’0’ and ’1’ with equal probabilities or M-ary sequence with each symbol equally probably. As we have this assumption, we can *replace the actual source encoder with a (pseudo) random binary sequence*, if we are not particularly interested in source encoding/decoding, sampling or quantization operations.

III. Channel Coding [1]

The famous Shannon capacity sets the maximum transmission rate C, where information can be transmitted with arbitrarily high reliability. For example, in additive white Gaussian noise channel this capacity is given by

$$C = B\log_2(1+S/N) \tag{10}$$

where B is the channel bandwidth, S and N are signal and noise powers, respectively. One method to approach Shannon limit is to use error control coding in the system, since it enables far more reliable transmission, although more energy per information bit is needed. The general system model is given in Fig. 3, where the source data is *encoded* in the transmitter side and the equalised signal is *decoded* in the receiver.

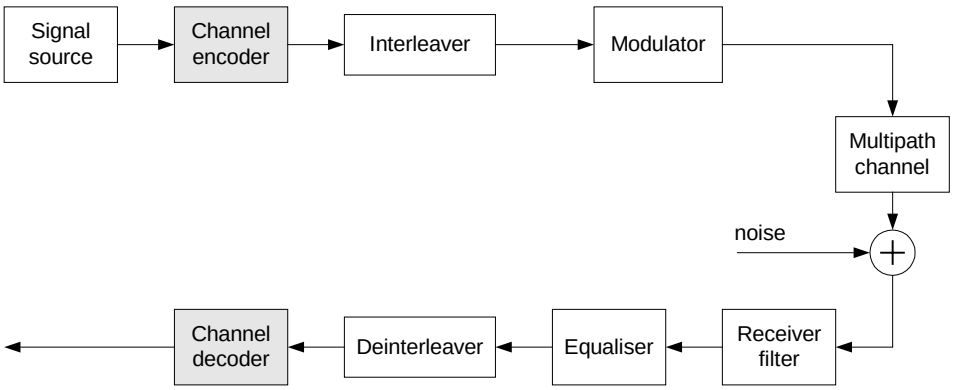

Figure 3. Block diagram for a system utilising channel coding.

Channel coding methods are divided into two different classes, which are *block and convolutional codes*. Block coding is more straightforward to implement, as it is comprised of a block of information bits and some check (parity) bits dependent on the information part. The convolutional coding uses a continuous data stream, which is processed by a specific finite state machine. In the recent years so called *turbo codes* have been a popular research topic, since they actually can reach very close to Shannon limit. Turbo codes are based on parallel or serial concatenation of basic (convolutional) codes and they were introduced by Berrou et al. 1993 [2].

The benefit of channel coding is usually given as *coding gain*, which is the difference in SNR between uncoded and coded systems at a certain BER level. Unfortunately, this gain is often very difficult to calculate analytically (at least in realistic situations), thus the performance is usually found out by computer simulations.

A. Block Codes

In general, a block channel encoder takes a block of *k* information bits in and *adds n-k parity bits* in the end of block, thus providing as output *n*-bit block, which is called a *codeword*. These codewords should be carefully designed, so that the channel decoder can recover the original message, although some transmission errors may have occurred. Hence, the decoder can *detect and correct transmission errors*, which improves the average bit error rate in the receiver. This kind of coding methods are called *forward error correction* (FEC) schemes. To design a specific FEC scheme, one has to choose values *n* and *k*, develop a good encoding method to construct parity bits and select a decoding algorithm.

Of course when *n* is large compared to *k*, the code makes it possible to detect and correct several transmission errors within a data block. *The decoder* just compares the received codeword to all possible words and *selects the best match*. If d_{min} is the minimum Hamming distance between any pair of words, the code can recover all words having *t* or less errors, where

$$t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor \quad . \tag{11}$$

The code designer tries to maximise *minimum distance* $d_{\min}$ but also the *code efficiency R = k/n*. Large *n* values would be beneficial in design, but unfortunately they lead to complex implementation of encoder and decoder, and therefore rather small values are usually favoured.

One simple example of generating a (7,4) binary block code is given as a shift register in Fig.4 The upper row is filled with input signal and then the parity bits in the lower row can be calculated. Finally, all 7 bits are available at the output.

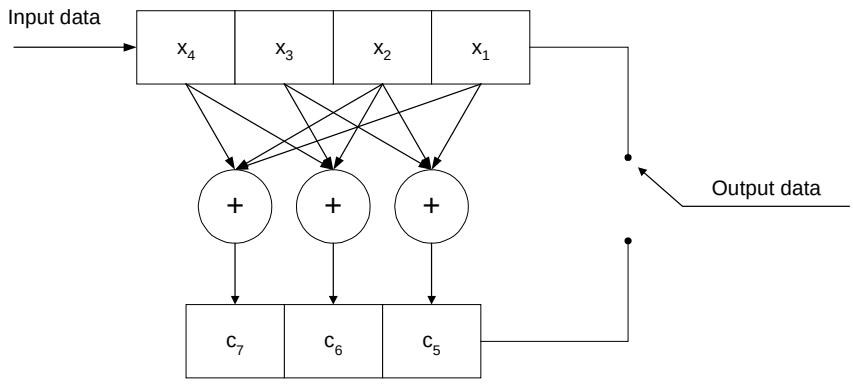

Figure 4. Shift register for generating binary (7,4) code. [3]

Channel decoders for block codes are much more complex than encoders due to transmission errors. A simple comparison between all possible code words and the received word becomes impractical as the block length grows. Therefore different computationally efficient but still

reasonably well performing decoding algorithms are developed. Part of the decoding methods are *nonalgebraic procedures*, while some others exploit *algebraic structure* of the code. Decoding performance improves further, if the decoder can use *soft values* as input. A soft value tells the decision bit, but also *reliability information* on that. For example, value 0.8 can denote strong probability of transmitting 1 and –0.9 strong probability of sending 0.

B. Convolutional Codes

In contrast to block codes, convolutional codes are operating with *continuous stream of input data* instead of blocks. However, they are also based on the *redundancy*, which distinguish different codes from each other and enables decoder to detect the correct transmitted sequence. The convolutional decoders itself differ very much from block decoders, but they are still characterised by *a bit rate R=k/n* and a distance measure between codes called *minimum free distance* d_f, which tells how far apart all possible sequences lie from each other.

So convolutional codes introduce some *memory* in the transmitted sequence, which is easy to find from the Fig. 5 below. This encoder takes k=2 bits in at the time and outputs n=3 each time. What is more, the same inputs are used to provide two sets of outputs; so called *constraint length* L is also 2. Typically, k and n are between 1 to 8, and R is then between 1/3 (at least in data services) and 7/8 and L is from 2 to 10. The convolutional codes are usually expressed with *polynomials* that tell which inputs are added in each output. For instance, generator polynomials for the given (2,3)-code would be $1+D^2+D^3$, $1+D+D^3$ and $1+D^2$, where D means one bit delay variable.

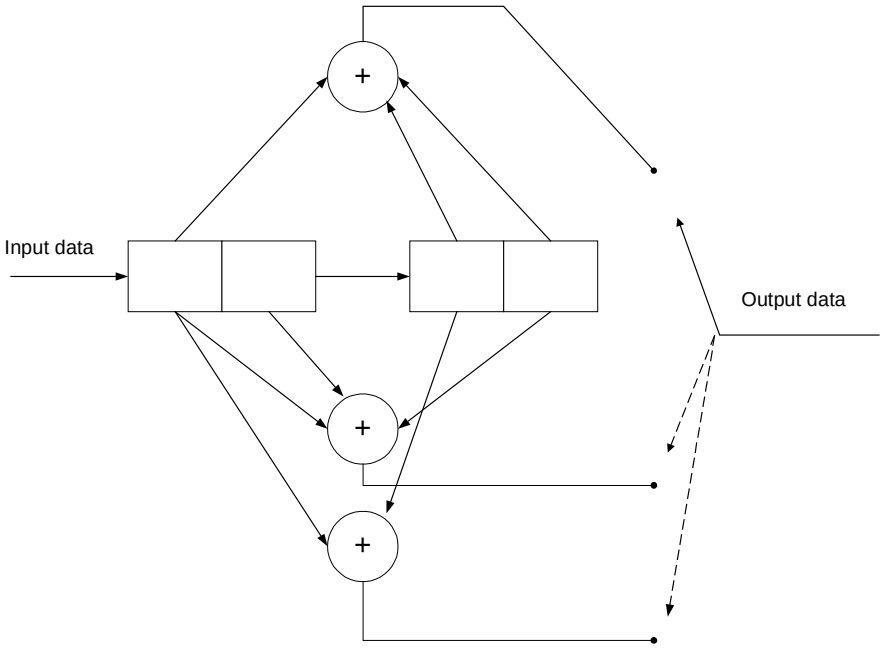

Figure 5. A 2/3-rate convolutional encoder with L=2, k=2, n=3. [3]

Alternative ways to describe encoder structure is to use *state or trellis diagrams*[3]. State diagram shows the internal state of the encoder memory and its possible movements as time goes by. Trellis diagram is more thorough, since it actually shows all the possible paths (movements), as it presents each time instant on the right of the current time instant. In that way, the whole trellis is saved when time goes on. Trellis diagram is especially handy, if one likes study what happens in the decoding of the sequence.

The decoding problem of convolutional codes (and closely related equalisation problems) is widely researched area. The two most common approaches (which are also very closely related) are called the *maximum-likelihood sequence estimation* MLSE (usually implemented by the Viterbi algorithm) and *maximum a posteriori* algorithm MAP. These are briefly introduced in the following, more information is found for instance from [5] about MLSE and from [4] about MAP.

The MLSE problem is to find the sequence of transmitted information symbols $\hat{\mathbf{u}}$, which maximises the likelihood function as follows

$$\hat{\mathbf{u}} = \arg\max_{\mathbf{u}} \; p\left(\hat{\mathbf{c}}\middle|\mathbf{u}\right) \;, \tag{12}$$

where $\hat{\mathbf{c}}$ is the received (encoded) signal from the equaliser. An equivalent problem is to maximise the log-likelihood function log $p(\mathbf{c}|\mathbf{u})$, which is solved by minimising the path metric

$$PM = \left\|\hat{\mathbf{c}} - \mathbf{U}\Psi\right\|^2 = \sum_k \left(\hat{c}_k - t_k\right)^2 \tag{13}$$

where the reconstructed encoder output assuming structure Ψ is given as

$$t_k = \sum_{m=0}^{M-1}\sum_{l=0}^{L} u_{m,k-l}\, g_{m,l} \;. \tag{14}$$

In MLSE-VA the trellis is spanned by all the L symbols in (14) and each transition is comprised of M elements, which are describing different encoder outputs. Hence, MLSE finds minimum sequence error.

The objective of the MAP algorithm is to minimise the bit error probability for each bit by estimating a posteriori probabilities (APP) of states and transitions of the Markov source from the received signal sequence. The MAP algorithm introduced by Chang and Hancock [4] uses the information of the whole received sequence to estimate a single bit probability. In other words, the MAP algorithm selects the bit $u_k \in \{-1,+1\}$ at time instant k, which maximises the following APP

$$\hat{u}_k = \arg\max_{u_k} \left[p\left(u_k\middle|\hat{\mathbf{c}}\right)\right] \;. \tag{15}$$

The optimum MAP algorithm saves multiplicative transition probabilities in the trellis, which is computationally difficult. Therefore in practical implementations the path metric is calculated in the

log-domain, which enables cumulative calculations. Since MAP requires both forward and backward recursions, it is around two times more complex than MLSE-VA. The main advantage of the MAP algorithm is more reliable soft information, since it is optimised for symbolwise decoding.

C. Example (GSM Speech)

To illustrate the two channel coding techniques the coding scheme of GSM Full Rate Speech is presented in Fig. 6 [6]. The most important beginning of the speech block is protected by block coding, so there is 3 parity bits for 50 first bits and 4 parity bits for the next 132 bits. Then these 189 bits are protected by ½-rate convolutional code. Finally, 78 last bits are added to the end as such, because the end is not so important for human's ear.

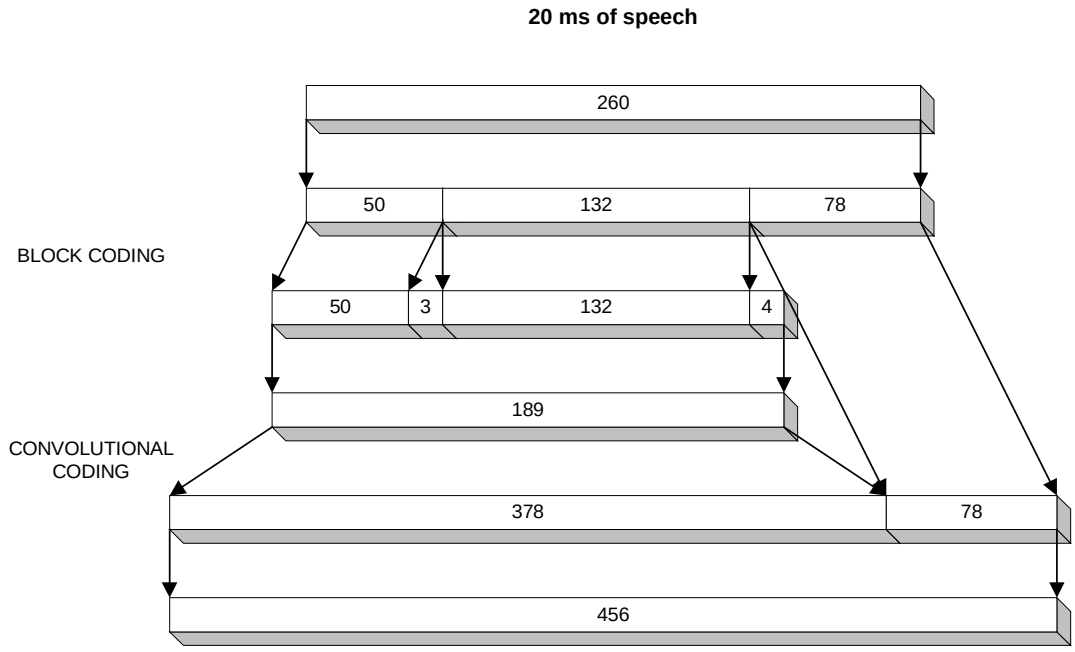

Figure 6. Coding scheme for GSM Full Rate Speech.

D. Interleaving, nonbinary codes, concatenation

In the presence of bursty error patterns, like in fading channels or interference, the normal error control methods are not sufficient enough to recover transmission errors. There are three different ways to improve coding results in this kind of situation; *interleaving, nonbinary codes and concatenation of several codes*. A new research topic has also been *iterative decoding and equalisation*, which can provide significant gains in some cases.

Interleaving is a simple method to break bursty errors far apart from each other. *The bits of a radio block are simply reordered*, which enables decoder to detect and correct transmission errors,

because errors are not any more consecutively. Like in Fig. 3 is presented, interleaving is done right after encoding and deinterleaving just before decoding. *Block interleaver* structure for (15,7) code with m=5 and d_{min} = 5 is shown in Fig. 7, although *convolutional structures* are also possible. So channel coding system with this kind of block interleaving, where bits are read in row after row and read out column after column (m bits in each column) can cope with up to

$$t = m\left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor \tag{16}$$

errors. The cost of interleaving is the inevitable *delay and storage requirements* both in the transmitter and receiver side.

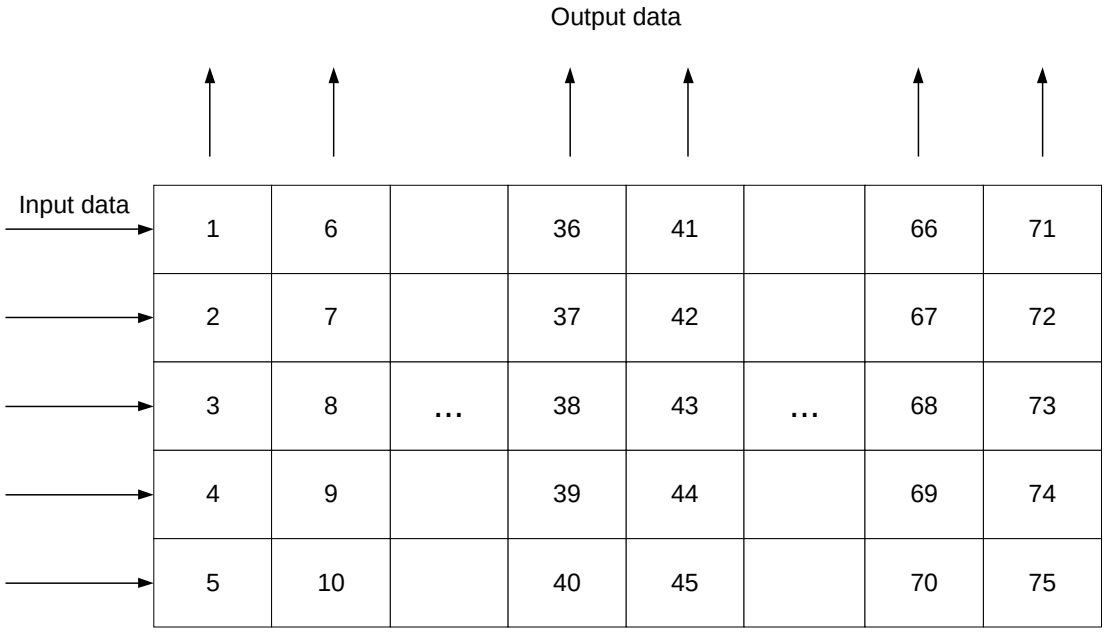

Figure 7. A block interleaver structure for a (15,7) code with m = 5 and d_{min} = 5.

When nonbinary codes are used, there are more than 2 symbols available to be selected. Usually there are q=2^k different symbols, which means that k information bits are mapped into one of those q symbols. The most popular class of nonbinary codes are *Reed-Solomon (RS) codes*, which have the following parameters for (N,K) code

- $N = q - 1 = 2^k - 1$
- $K < N$
- $D_{min} = N - K + 1.$

Hence, an (N,K) RS code can correct up to (N-K)/2 symbol errors per block. RS codes are used a lot due to their *good distance properties and efficient decoding algorithms*. Also bursty errors lead to a small number of symbol errors, which often are easy to correct as long as there are less than (N-K)/2 symbol errors to be corrected.

Concatenated code means *a combination of two shorter codes* that are either serially or parallel combined, in Fig.8 serially concatenated system is presented. Often the outer code is nonbinary and inner is binary, like the inner is (*n,k*) binary convolutional code and outer is (*N,K*) RS code. The inner decoder then outputs soft or hard outputs onto *k*-bit symbols of the outer code. The minimum distance for the concatenated code is $d_{min}D_{min}$, where d_{min} and D_{min} are the minimum distances of the inner and outer codes. The bit rate is Kk/Nn. Also interleaver can be used between outer and inner coders to handle bursty errors.

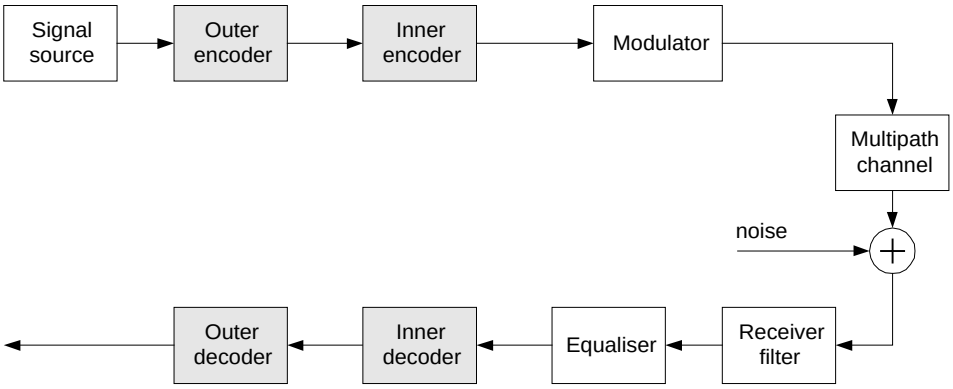

Figure 8. Block diagram of a system with serially concatenated code.

E. Simulation of coded system

Coded communication link can be divided into two independent parts; *the encoder/decoder pair (codec) and the discrete channel between them*. By this way we achieve an efficient and flexible simulation platform. Typical reason for this separation is that we need 4-16 samples per symbol to represent the discrete channel, but *for codec only one sample per symbol is enough*. And if we like to try another codec with the same discrete channel, it is enough to simulate it with the symbol rate, which provides computational savings compared to sampling rate. Also this separation provides useful system insight for the system engineer.

The explicit simulation of a codec is not always needed, since one can approximate or calculate *bounds* for the codec performance. There are two clear advantages, if one can avoid simulation. Decoding algorithms are often very *complex* and therefore simulations would take a lot of time. Another thing is that the target bit error probability is set to very low level (like 10^{-6} or even lower). Hence, to achieve confidence in the estimated error rate, we need to simulate at least $10/p$ symbols, where p is BER of the system. So *the simulation time will be very long*. Due to these reasons codec simulation times are often reduced by statistical means or even calculated analytically. We can derive quite good bounds for large SNR values and simulate shorter runs for smaller SNR values. Sometimes an explicit simulation of a codec is required, if for instance some parameter influence on the decoder is studied.

IV. Summary

In this report we have considered source and channel coding methods and how to simulate them. Source encoding starts with sampling an analog audio signal, then quantization and encoding into bit groups. In the receiver the digital signal is decoded and filtered back to analog form. In uniform quantization the input signal is divided into equal intervals, while in nonuniform methods unequal intervals are used. If correlated sequence is processed, it is useful to exploit differential quantization. Shannon's proposal for encoding discrete information is described. That method allocates shorter codes for messages of higher probability, thus the average number of bits per symbol decreases. Usually it is enough to simulate idealized and very simple A/D converter, although real converter has many imperfections. With this assumption, we can actually replace source encoding with (pseudo) random binary sequence in the simulation.

Channel coding is used to achieve more reliable transmission by adding parity check bits after information bits. Typical channel encoding techniques are block and convolutional codes. The block encoders are processing a block of data at a time and calculating the parity check bits for this block. A code designer tries to maximise the minimum Hamming distance between any pair of code words, but also transmit as little redundancy bits as possible. Convolutional encoders introduce memory into the code by using the previous time instants as well to calculate the current encoder output. Convolutional decoders are often based on MLSE principle, which finds the minimum sequence error, or on MAP principle, which solves the minimum symbol error. For instance in Full Rate GSM Speech both block and convolutional encoding are used, but on the other hand, the end of each burst is kept uncoded as it is not so critical for human's ear. Another methods to improve link performance is to use interleaving, nonbinary codes (more than 2 different symbols available) or concatenate at least two codes serially or parallel.

The coded system consists of encoder/decoder pair and a discrete channel between them. Typically several samples per symbol are needed to represent the channel part in a simulation, but only one sample per symbol in the codec. Sometimes it is enough to calculate bounds for a coded system and thereby avoid long and complex simulations. Especially for high SNR values one can calculate reliable bounds and save a lot of simulation time.

V. References

[1] M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, "Simulation of Communication Systems", Plenum Press, 1992, 731 p.

[2] C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes", Proc. ICC'93, Geneva, Switzerland, May 1993, pp. 1064-1070.

[3] J.G. Proakis, "Digital Communications", 3rd edition, McGraw-Hill, 1995, 929 p.

[4] R.W. Chang and J.C. Hancock, "On Receiver Structures for Channels Having Memory", *IEEE Trans. on Inf. Theory*, Vol. 12, No. 3, pp. 463-468, Oct. 1996.

[5] G. D. Forney, Jr., "Maximum-likelihood Sequence Estimation of Digital Sequences in the Presence of Intersymbol Interference", *IEEE Trans. Inf. Theory*, Vol. IT-18, pp. 363-378, May 1972.

[6] ETSI specifications, GSM 05.03, Channel coding, March 1995.

VI.Homework

P4.20. *Discrete Channel Model and Coding.* Parity-check codes are frequently used in data communications. In each byte (8 bits), 7 bits represents information and the eighth is a parity bit; it is "1" if the number of "ones" in the information bits is odd, and it is "0" otherwise. Thus, the number of "ones" in every byte should be even and the receiver detects errors if the number of "ones" is odd. Find the probability of undetected errors for the Gilbert model given below (See ex. 4.3 and 4.4).

Gilbert model. According to Gilbert's model, it consists of stochastic sequential machine (SSM), which has two states: G (good) and B (bad) with transition probabilities

$$q_{11} = 0.997, \quad q_{12} = 0.003, \quad q_{21} = 0.034, \quad q_{22} = 0.966.$$

The model state transition diagram is illustrated in Fig. 9 below. In state G errors do not occur, while in state B the bit-error probability equals 0.16. Thus we have

$$p_{00}(1) = 1, \quad p_{01}(1) = 0, \quad p_{10}(1) = 0, \quad p_{11}(1) = 1$$

$$p_{00}(2) = 0.84, \quad p_{01}(2) = 0.16, \quad p_{10}(2) = 0.16, \quad p_{11}(2) = 0.84.$$

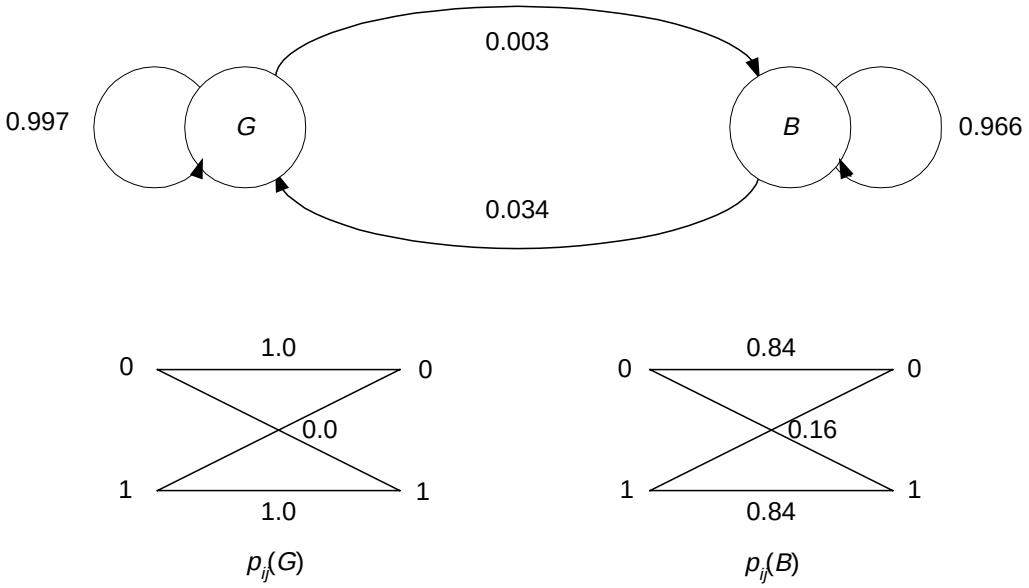

Figure 9. Gilbert model, state and channel transitions.