

## Channel Coding

### Error Correcting Coding

#### Introduction

The noise present during transmission may cause errors. It is important to be able to firstly detect if an error has occurred, so that the appropriate action, like asking for a retransmission, can occur. In broadcasting situations retransmission is not possible, and under those circumstances the ability to be able to correct for errors is very important. This is done by adding error correcting data to the transmitted message. By adding this data to the message data, more data is squeezed through the channel and more errors will occur. If the error correction can fix more errors than are created by adding the additional data, then a net reduction in errors occur and there is an overall benefit. As can be seen later, it is possible to add too many error correcting bits to the message, resulting in a net increase in errors and a decrease in channel performance.

There are two basic error correcting techniques:

Firstly Block codes, where a block of data has an appropriate block of error correcting bits added to it. After reception of the whole block at the receiver, any errors in that block are corrected.

Secondly Convolutional codes, where a length of K bits of data are analysed and N error correcting bits are generated. M new data bits are then read into the string of K data bits and another N new error correcting bits are generated. Convolutional codes give a better performance for burst errors, which are common in many data transmission systems.

#### Block Codes

Row and Column Parity

By performing a parity check on both the row and column of a data block, one error can be corrected. As an example consider adding an odd parity bit to a 4x4 block of data.

0	0	1	0	0
0	1	1	0	1
0	1	0	1	1
1	0	1	1	0
0	1	0	1	x

Consider that the red data bit is received in error as shown below. This will then cause a different parity check for both the blue parity bits, thus showing which bit is wrong, to that we can correct the data by inverting that bit, to produce the original data.

0	0	1	0	0
0	0	1	0	1
0	1	0	1	1
1	0	1	1	0
0	1	0	1	x

0	0	1	0	0
0	0	1	0	1
0	1	0	1	1
1	0	1	0	0
0	1	0	1	x

Having two errors as shown, causes 4 parity bits, shown in blue to be wrong. In this situation we know that there have been two errors, but we don't know where. Furthermore if there are two errors on the same row, there will be no row parity errors, but there will be two column parity errors, again we know there are two errors, but we do not know where. If there are 4 errors, in two rows of two and two columns of two, as shown below, then no parity errors are shown.

0	0	1	0	0
0	0	1	1	1
0	1	0	1	1
1	1	1	0	0
0	1	0	1	x

Any errors in the parity bits can also be detected by adding a parity on the row and column parity bits. That is then located at the position shown by X. When no parity is added to the parity bits or when a parity is added to both row and column parity bits, the nice block structure is no longer maintained. Row and Column parity are not as efficient as other block codes but they can be used to very simply illustrate the principle and limitations of error correcting coding.

### **Hamming Codes**

Hamming coding illustrates the general principles of error correcting coding. Consider a data message of  $k$  bits long. In practice Hamming codes are limited to correct one error, though in theory they can be used to correct more than one error, the solution of the simultaneous equations required becomes very difficult. We want to correct one error by adding  $b$  check-bits. The total message is thus  $n=k+b$  bits long. For a binary data system, there will thus be  $s=2^k$  different data messages and a total of  $2^n$  possible number of symbols to be transmitted. If  $E$  errors are to be corrected, the number of allowable words per message is:

$$1+n$$

Where the 1 is the word without errors and there are  $n$  words with one error.

Since we have  $s$  different data messages, and all of those with the allowable errors must fit inside the  $2^n$  possible number of symbols, so that:

$$2^n \geq s * (1+n) = 2^k * (1+n)$$

$$2^{n-k} = 2^b \geq (1+n)$$

Thus for  $b=5$  parity bits, we can have at best  $32-1=31=n$  total number of bits and thus  $31-5=26=k$  data bits. The number of parity-bits to be added for different data blocks and the resulting coding efficiency is shown in the following table:

Data	Parity	Total	$\eta$	$\eta\%$
4	3	7	4/7	57.1
11	4	15	11/15	73.3
26	5	31	26/31	83.9
57	6	63	57/63	90.5
120	7	127	120/127	94.5

The efficiency will thus increase as the length of the data block increases.

**Example:**

Design a Hamming code to correct one error for a 4 bit data block. From the table and formula 3 parity bits are required to correct one error. We generate 3 parity equations, each of those equations checking the parity of different data and parity bits. This results in 3 simultaneous equations that are solved using modulo-2 algebra. To keep the solutions to these simultaneous equations simple, each of the three equations is arranged to only look at one parity-bit, as shown below.

Digit	D1	D2	D3	D4	D5	D6	D7
Parity Mask P1		1	1	1	1		
Parity Mask P2	1		1	1		1	
Parity Mask P3	1	1		1			1
PM Decimal	3	5	6	7	4	2	1

In this table the parity masks are shown, if there is a 1 in the table then that digit is included in the parity calculation. We must make sure that the check masks for each of the digits D1 to D7 is different and that each digit has at least one parity check applied to it. (Hence PM decimal = 0 is not used). D1 to D4 are the data bits and D5 to D7 are the error correcting bits. From this table it can be seen that the three parity equations are:

$$\begin{aligned} P1 &= D2 + D3 + D4 + D5 \\ P2 &= D1 + D3 + D4 + D6 \\ P3 &= D1 + D2 + D4 + D7 \end{aligned}$$

Where P1 is 0 or 1 to give an even number of ones for even parity and an odd number of ones for odd parity. These three modulo-2 algebra equations are now solved for D5, D6 and D7. Modulo-2 algebra is similar to conventional binary algebra except that  $1+1=0$ . By arranging the parity masks such that the error correcting bits only have one parity check applied to them, the solution of those equations is trivial, so that:

$$\begin{aligned} D5 &= D2 + D3 + D4 \\ D6 &= D1 + D3 + D4 \\ D7 &= D1 + D2 + D4 \end{aligned} \quad \text{for even parity}$$

and

$$D5 = D2 + D3 + D4 + 1$$

$$D6 = D1 + D3 + D4 + 1$$

$$D7 = D1 + D2 + D4 + 1$$

for odd parity

At the receiver the same parity is calculated for the received signal. If the received parity and the calculated parity is different, then there is an error. The difference in the received and calculated parity bits give a precise location of the error so that that data bit can be changed to correct the data. If the data transmitted is 1101 then the error correcting bits for even parity are as follows:

Digit	D1	D2	D3	D4	D5	D6	D7
Parity Mask P1		1	1	1	1		
Parity Mask P2	1		1	1		1	
Parity Mask P3	1	1		1			1
PM Decimal	3	5	6	7	4	2	1
Data + Parity	1	1	0	1	0	0	1

The transmitted signal is thus 1101001. If 1111001 is received, where the blue bit is an error, then the error can be corrected as follows:

Digit	D1	D2	D3	D4	D5	D6	D7
Parity Mask P1		1	1	1	1		
Parity Mask P2	1		1	1		1	
Parity Mask P3	1	1		1			1
PM Decimal	3	5	6	7	4	2	1
TX Data + Parity	1	1	0	1	0	0	1
RX Data + Parity	1	1	1	1	0	0	1
Parity for RX data					1	1	1
Difference Decimal					1	1	0

Calculating the parity for the received signal gives 111 for bits D5 to D7. This is different from the values received, so that there is an error. The difference in the received and calculated parity is 110 or 6, so that the error is in the data bit that has the 110 = 6 parity mask. Thus bit D3 was in error. Hamming coding will correct for errors in both the data and parity bits, as can be shown below, where 1101001 is sent and 1101011 is received.

Digit	D1	D2	D3	D4	D5	D6	D7
Parity Mask P1		1	1	1	1		
Parity Mask P2	1		1	1		1	
Parity Mask P3	1	1		1			1
PM Decimal	3	5	6	7	4	2	1
TX Data + Parity	1	1	0	1	0	0	1
RX Data + Parity	1	1	0	1	0	1	1
Parity for RX data					0	0	1
Difference Decimal					0	1	0

## Multiple Error Correcting Algebraic Codes

Hamming codes with one bit error correction are very easy to implement and can easily be done using CPLD's. There are situations where more than one error is to be corrected. It is very difficult to do that using Hamming codes but other code techniques exist for this.

### BCH codes

(Bose, Chandhuri and Hocquenhem) are algebraic codes similar to the Hamming code. However the BCH codes are designed to be decoded using combinational logic. The BCH codes are not as efficient as the ideal Hamming code, but the hardware can easily be realised. The number of data bits and the total sequence length for detecting and correcting  $t$  errors is given by the following equations:

$$\begin{aligned} \text{Block Length:} & & n &= 2^m - 1 \\ \text{Message Bits:} & & k &\geq n - mt \\ \text{Number of detectable errors:} & & t & \end{aligned}$$

and are shown in the following table:

Integer m	3	4	4	4	5	5	5	5	5
Word Length n	7	15	15	15	31	31	31	31	31
Data length k	4	11	7	5	26	21	16	11	6
Error Corr. Bits b	3	4	8	10	5	10	15	20	25
Number of errors t	1	1	2	3	1	2	3	4	5
Efficiency $\eta\%$	57	73	47	33	84	68	52	35	19

Integer m	6	6	6	6	6	6	6	6	6
Word Length n	63	63	63	63	63	63	63	63	63
Data length k	57	51	45	39	36	30	24	18	7
Error Corr. Bits b	6	12	18	24	27	33	39	45	56
Number of errors t	1	2	3	4	5	6	7	8	9
Efficiency $\eta\%$	90	81	71	62	57	48	38	29	11

Note that for  $m=6$ ,  $t=5$  the number of message bits that can be used is greater than the 33 that is expected from the formula. BCH codes are used in dial up modems to correct for transmission errors.

### Reed Solomon codes

Reed Solomon (RS) codes are a subclass of non-binary BCH codes and were discovered before BCH codes. RS codes are extremely efficient and are used in many applications. RS codes are very good for removing burst errors. As a result they are used on Audio CD's.

For RS codes the number of data bits and the total sequence length for detecting and correcting  $t$  errors is given by the following equations:

$$\begin{aligned} \text{Block length:} & & n &= 2^m - 1 \text{ symbols} \\ \text{Message length:} & & k & \text{ symbols} \\ \text{Number of Parity bits:} & & n - k &= 2t \text{ symbols} \\ \text{Number of detectable errors:} & & t & \end{aligned}$$

A popular value is  $m=8$ , resulting in a block length of 255 symbols and RS codes with  $m=8$  are very powerful in correcting errors.

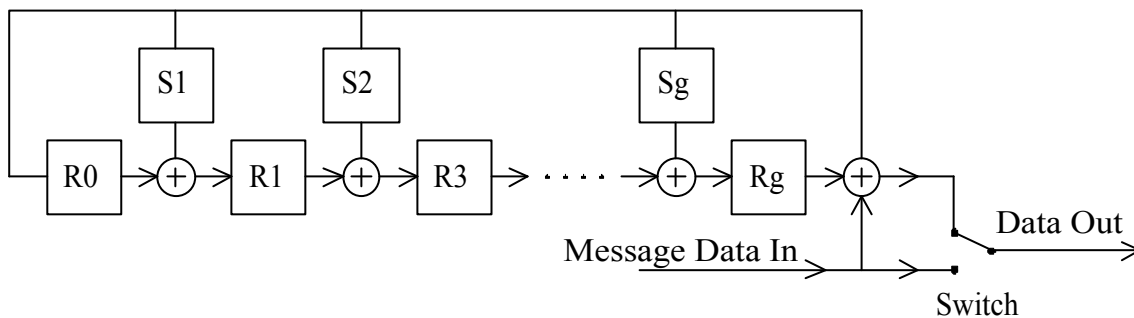
A Reed Solomon code is normally denoted as  $RS(n,k)$  to show the block length and message length.  $RS(7,4)$  will thus correspond to the previous Hamming code example. For Audio CD's Sklar[1] has a good description of how RS codes are used. The symbols are 8 bits, rather than binary as we have been considering till now. Data is interleaved and then 24 bytes have 4 error correcting bytes added to them. That new data stream is then interleaved again and a further 4 error correcting words are added to every 28 byte block to produce 32 byte long blocks. Finally the resulting data is interleaved again. In decoding the errors are corrected and if needed interpolation between successive samples is done to minimise the effect of any uncorrectable errors. This results in a very good burst error correction capabilities and typical scratches are undetectable on most disks.

## Cyclic Codes

Binary Cyclic codes are a subclass of linear block codes. At the start of the data block, the shift register is clear and its content is 0 and the switch is set to the bottom position to transmit the data directly. At the transmitter, the message data is firstly transmitted directly and at the same time put into a feedback register, with the particular feedback selections  $S_x$  as either 1 or 0. This feedback is then exclusive OR gated with the shift register contents. At the end of the data block, the parity bits contained in the register are then transmitted. At the receiver the same hardware calculates the parity bits as the message data is received. By then comparing the received parity bits transmitted at the end of the message data block any errors can be determined.

The feedback code is a polynomial:  $G(x) = 1 + s_1X + s_2X^2 + s_3X^3 + s_4X^4 + \dots + s_gX^g$

Where  $S_1$  to  $S_g$  are either 0 or 1, depending on the polynomial required. When the polynomial corresponds to that of a maximum length pseudo random sequence, the cyclic code is called a Pseudo Noise (PN) code.



The above generator block diagram can also be used for the generation of Hamming codes, BCH codes and RS codes. For details of suitable polynomials see Haykin Ch10.4 or Sklar section 5.6.2.

## ***Interleaving***

In some applications like an Audio CD, a scratch causes a burst of errors. Block error correcting codes are susceptible to burst errors, since that will typically cause too many errors to be corrected. For a scratch on a CD, if all those errors are within one block for the error correcting coding then that block of data has too many errors. If the data is spread out over different blocks, then a burst error is spread over several blocks at the error correction stage and it is more likely that all the errors can be corrected. The principle is illustrated in the diagram below. In practice the spreading is many times more than what is shown here as it is desirable to have a wide spread of the data. There is a compromise in that the data spreading will cause a delay. In some applications like a CD player, that is not significant, however in other applications like mobile phones, the delay needs to be minimised. Interleaving is used in many practical communication systems in conjunction with error correcting coding.

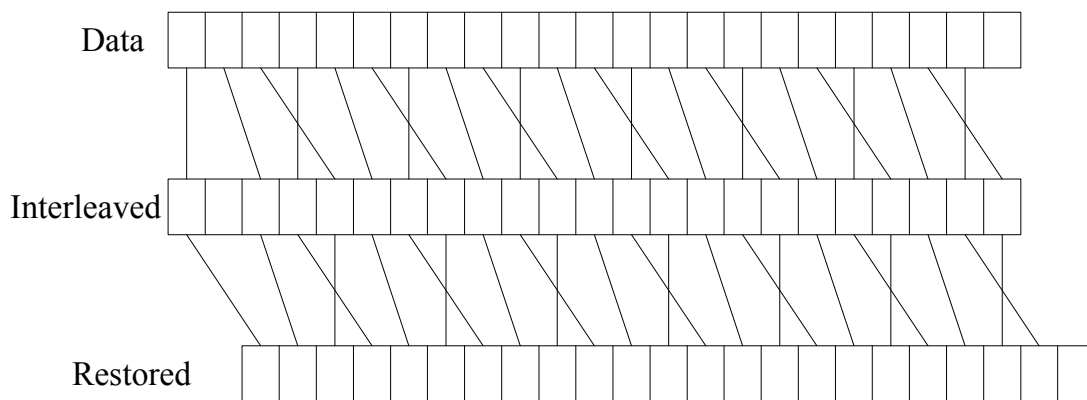


Illustration of interleaving

## ***Convolutional Codes***

[More to follow here](#)

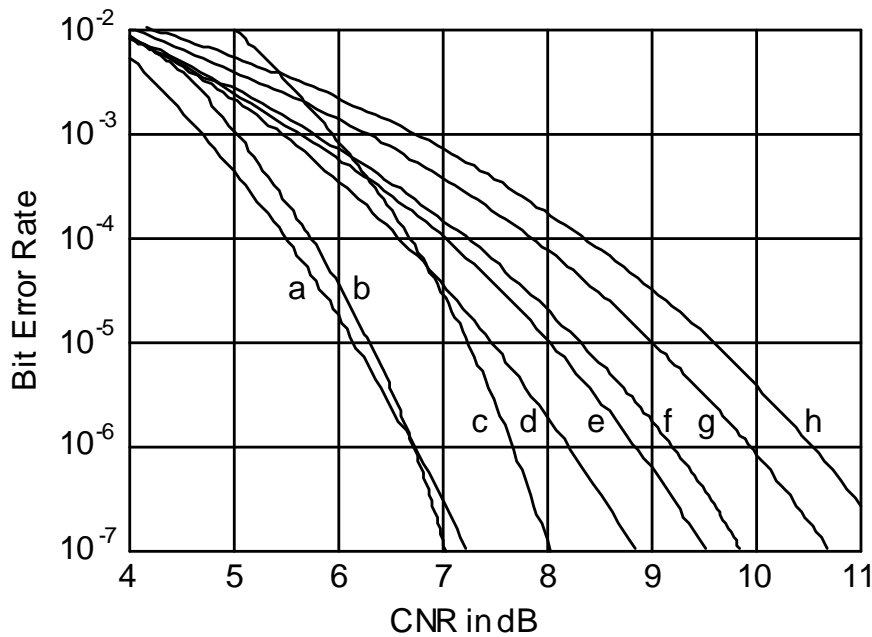


Figure 4. Error Correcting Coding and Channel Capacity [Ref 7, Fig 5.23]  
 a Vitterby half rate,  $K=7$ . b BCH (127,64),  $e = 10$ . c BCH (127,36),  $e = 15$ .  
 d Extended Golay (24,12), f Hamming (15,11),  $e = 1$ . h Uncoded.

### **Introduction to Information Theory**

(Haykin [2] Chapter9) This section gives a very brief introduction to Information theory, and only covers just enough material to deal with the Huffman coding used in Fax machines.

The statement “Examinations are abolished” does not contain much information since it is so unlikely that one would need to verify that it is in fact so and that would immediately raise the question “How are we going to be graded, we will need a mark for this subject”. Similarly the statement “Examinations will be held” also does not contain much information since it is fully expected that this is the case.

When we are asked to use a question to guess a number, the most information is gained if the answer has a 50% probability. For instance to guess a number between 1 and 8, asking if it is greater or equal to 4 results in the most information. The Entropy, (H) being the information content of a source symbol or of a particular message with  $k$  symbols, is given by:

$$H(x) = -\sum_{\text{all } k} p_k \log_2(p_k)$$

Note that logarithms to the base 2 are used here. These can easily be evaluated using a normal calculator by using the expression:

$$\text{Log}_2(A) = \frac{\text{Log}_{10}(A)}{\text{Log}_{10}(2)}$$

The maximum Entropy occurs when each symbol has an equal probability of  $(\frac{1}{k})$  so that:

$$H(x) \leq -\text{Log}_2\left(\frac{1}{k}\right) = \text{Log}_2(k)$$

**Example**

Consider the alphabet of 26 letters A to Z. If they have equal probability, the Entropy is:

$$H = \text{Log}_2(26) = 4.7004 \text{ bits/letter.}$$

In practice 5 bits per letter are required to transmit the alphabet without any coding.

If one assumes that the letters have a probability of:

Probability	Letters
P=0.1	a, e, o, t
P=0.07	h, i, n, r, s
P=0.02	c, d, f, l, m, p, u, y
P=0.01	b, g, j, k, q, x, w, v, z

Then the Entropy is:

$$H = -(4 \times 0.1 \times \text{Log}_2(0.1) + 5 \times 0.07 \times \text{Log}_2(0.07) + 8 \times 0.02 \times \text{Log}_2(0.02) + 9 \times 0.01 \times \text{Log}_2(0.01))$$

$$H = 4.17 \text{ bits / symbol}$$

Doing a fully detailed analysis by considering a long text, like a book and analysing the relative probabilities of all the letters, results in a value of H of approximately 4. Considering inter-symbol dependence, ie U always follows Q, and allowing for us do some intelligent context related guessing of missing or wrong letters shows that the actual Entropy is more like 1.5 bit/letter, resulting in an efficiency of 30%.

Note this also implies that in examination scripts the lecturer can in most instances guess the meaning of illegibly written text, but if less than half the characters can only be deciphered, there are big problems in understanding what is written. There is a tendency amongst mobile phone SMS users to abbreviate and misspell words to reduce the amount of text. This relies on the redundancy in the English text described above.

The relative occurrence of letters of the alphabet does vary with language and even spelling within the same language. In the USA the letter Z is used more often than in the UK.

**Huffman Coding**

Huffman coding allows the length of the words to be varied, the process used it to successively combine the symbols of least probability. The process is best illustrated using an example.

Consider 5 symbols S<sub>0</sub> to S<sub>4</sub>. These symbols have probabilities ranging between 0.4 and 0.1 as shown in the figure below:

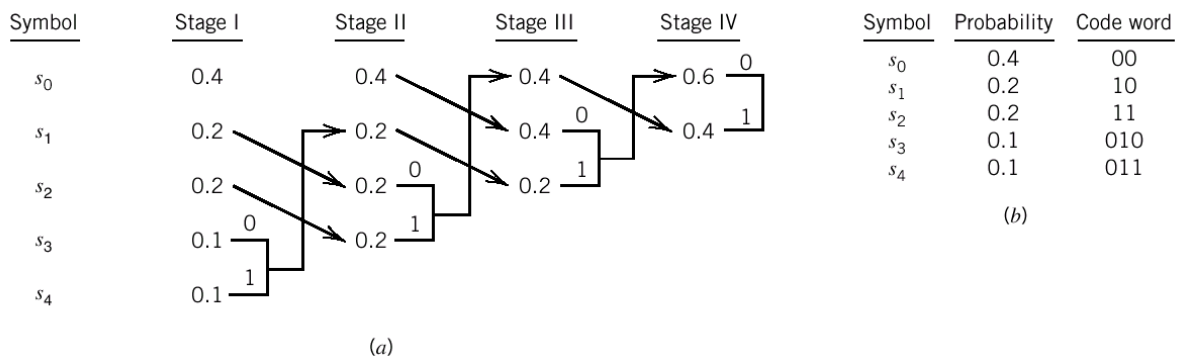


Figure 1 (a)Huffman encoding algorithm and (b) resulting Code words (Fig 9.5 Haykin)

The process is as follows:

Stage 1. Arrange the symbols in an order of descending probability.

Stage 2. Combine the two symbols with the least probability and then arrange the resulting symbols again in an order of descending probability. To minimise the word length of the symbols, any combined groups are put at the top of a group of symbols with the same probability, as shown for  $S_3$  and  $S_4$  combined in Stage 2 above.

Stage 3 etc. Repeat this process until all symbols are combined and a probability of 1 results.

The code for each symbol is now obtained by following this Huffman coding tree, with the least significant bits being in the column corresponding to stage 1 and the most significant bits corresponding to the column of stage 4. The resulting codes are shown in Figure 1(b).

The efficiency of the coding can be obtained as follows:

Symbol	Probability	Coding	$P\log_2(P)$	# of bits	Bits x Prob
$S_0$	0.4	00	-0.52877	2	0.8
$S_1$	0.2	10	-0.46439	2	0.4
$S_2$	0.2	11	-0.46439	2	0.4
$S_3$	0.1	010	-0.33219	3	0.3
$S_4$	0.1	011	-0.33219	3	0.3
Sum	1		2.12193		2.2

If no coding is used 3 bits are required to transmit the 5 symbols, resulting in an efficiency of  $2.12193/3 = 70.73\%$ . For Huffman coding the average symbol length is 2.2 bits per symbol. The efficiency of the code is thus  $2.12193/2.2 = 96.45\%$ . Huffman coding normally results in efficiencies greater than 90%.

More efficient coding techniques are possible, but are beyond the scope of this course. Lempel-Ziv coding for instance uses the inter-symbol interdependence to further reduce the average number of bits per symbol that need to be sent.

Huffman coding is used together with run length coding in fax machines. In a fax machine the pixel is either black or white. On a scan line one always starts with a white pixel and in run length encoding the number of successive white pixels is transmitted as a Huffman coded word. The run of white pixels is followed by a run of black pixels. The number of successive black pixels is again coded using Huffman coding. This is followed by a run of white pixels etc. The end of line symbol denotes the end of the line, and is required to ensure that the system can cope with an error.

#### References:

- [1] Bernard Sklar, "Digital Communications", Prentice Hall, 1988, isbn 0-13-212713-X