MODULE 4

CHAPTER 1: ERROR CONTROL CODING

STRUCTURE

- Rationale for Coding
- Discrete memory less channel
- Shannon’s theorem on channel capacity Revisited
- Types of errors
- Types of codes
- Example of Error Control Coding
- Block codes
- Minimum Distance Considerations
- Standard Array and Syndrome Decoding
- Hamming Codes

OBJECTIVE

- To analyze different types of erros
- To develop procedures for designing efficient coding schemes for controlling various types of errors in digital communication system.
- To Study various methods of detecting and/or correcting error and compare different coding schemes.

INTRODUCTION

The earlier chapters have given you enough background of Information theory and Source encoding. In this chapter you will be introduced to another important signal - processing operation, namely, "**Channel Encoding**", which is used to provide 'reliable' transmission of information over the channel. In particular, we present, in this and subsequent chapters, a survey of 'Error control coding' techniques that rely on the systematic addition of 'Redundant' symbols to the transmitted information so as to facilitate two basic objectives at the receiver: 'Error-detection' and 'Error correction'. We begin with some preliminary discussions highlighting the role of error control coding.

4.1 RATIONALE FOR CODING:

The main task required in digital communication is to construct 'cost effective systems' for transmitting information from a sender (one end of the system) at a rate and a level of reliability that are acceptable to a user (the other end of the system). The two key parameters available are transmitted signal power and channel band width. These two parameters along with power spectral density of noise determine the signal energy per bit to noise power density ratio, $\mathbf{E_b/N_0}$ and this ratio, as seen in chapter 4, uniquely determines the bit error for a particular scheme and we would like to transmit information at a rate $\mathbf{R_{Max} = 1.443\ S/N}$. Practical considerations restrict the limit on $\mathbf{E_b/N}_0$ that we can assign. Accordingly, we often arrive at modulation schemes that cannot provide acceptable data quality (i.e. low enough error performance). For a fixed $\mathbf{E_b/N}_0$, the only practical alternative available for changing data quality from problematic to acceptable is to use "coding".

Another practical motivation for the use of coding is to reduce the required $\mathbf{E_b/N}_0$ for a fixed error rate. This reduction, in turn, may be exploited to reduce the required signal power or reduce the hardware costs (example: by requiring a smaller antenna size).

The coding methods discussed in chapter 2 deals with minimizing the average word length of the codes with an objective of achieving the lower bound viz. **H(S) / log r**, accordingly, coding is termed "entropy coding". However, such source codes cannot be adopted for direct transmission over the channel. We shall consider the coding for a source having four symbols with probabilities $\mathbf{p(s_1) = 1/2,\ p(s_2) = 1/4,\ p(s_3) = p(s_4) = 1/8}$. The resultant binary code using Huffman's procedure is:

s_1………	**0**	s_3……	**1 1 0**
s_2………	**10**	s_4……	**1 1 1**

Clearly, the code efficiency is **100%** and **L = 1.75 bints/sym = H(S)**. The sequence $\mathbf{s_3s_4s_1}$ will then correspond to **1101110**. Suppose a one-bit error occurs so that the received sequence is **0101110**. This will be decoded as "$\mathbf{s_1s_2s_4s_1}$", which is altogether different than the transmitted sequence. Thus although the coding provides **100%** efficiency in the light of Shannon's theorem, it suffers a major disadvantage. Another disadvantage of a '**variable length**' code lies in the fact that output data rates measured over short time periods will fluctuate widely. To avoid this problem, buffers of large length will be needed both at the encoder and at the decoder to store the variable rate bit stream if a fixed output rate is to be maintained.

Some of the above difficulties can be resolved by using codes with "**fixed length**". For example, if the codes for the example cited are modified as **000**, **100**, **110**, and **111**. Observe that even if there is a one-bit error, it affects only one "**block**" and that the output data rate will not fluctuate. The encoder/decoder structure using '**fixed length**' code words will be very simple compared to the complexity of those for the variable length codes.

Here after, we shall mean by "**Block codes**", the fixed length codes only. Since as discussed above, single bit errors lead to '**single block errors**', we can devise means to detect and correct these errors at the receiver. Notice that the price to be paid for the efficient handling and easy manipulations of the codes is reduced efficiency and hence increased redundancy.

In general, whatever be the scheme adopted for transmission of digital/analog information, the probability of error is a function of signal-to-noise power ratio at the input of a receiver and the data rate. However, the constraints like maximum signal power and bandwidth of the channel (mainly the Governmental regulations on public channels) etc, make it impossible to arrive at a signaling scheme which will yield an acceptable probability of error for a given application. The answer to this problem is then the use of '**error control coding**', also known as '**channel coding**'. In brief, **"error control coding is the calculated addition of redundancy"**. The block diagram of a typical data transmission system is shown in Fig. 4.1

The information source can be either a person or a machine (a digital computer). The source output, which is to be communicated to the destination, can be either a continuous wave form or a sequence of discrete symbols. The '**source encoder**' transforms the source output into a sequence of binary digits, the information sequence **u**. If the source output happens to be continuous, this involves **A-D** conversion as well. The source encoder is ideally designed such that (i) the number of bints per unit time (bit rate, $\mathbf{r_b}$) required to represent the source output is minimized (ii) the source output can be uniquely reconstructed from the information sequence **u**.

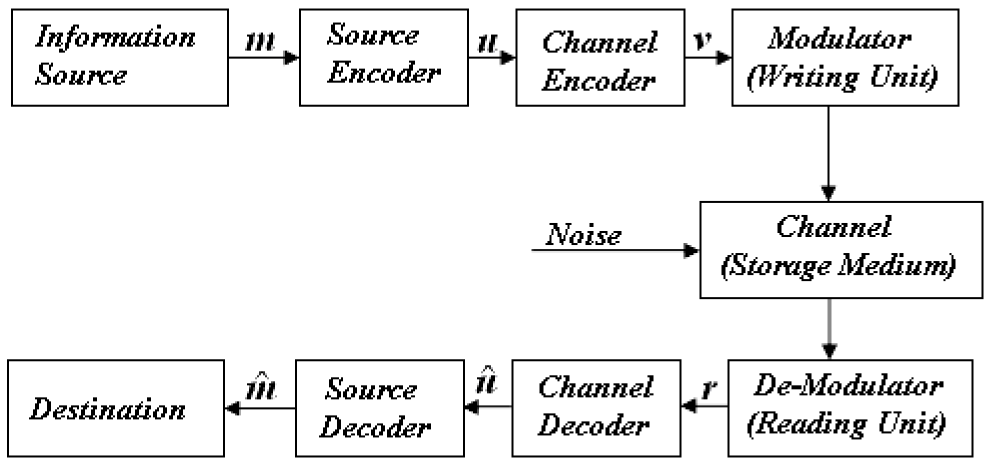


Fig 4.1: Block diagram of a typical data transmission

The '**Channel encoder**' transforms **u** to the encoded sequence **v**, in general, a binary sequence, although non-binary codes can also be used for some applications. As discrete symbols are not suited for transmission over a physical channel, the code sequences are transformed to waveforms of specified durations. These waveforms, as they enter the channel get corrupted by noise. Typical channels include telephone lines, High frequency radio links, Telemetry links, Microwave links, and Satellite links and so on. Core and semiconductor memories, Tapes, Drums, disks, optical memory and so on are typical storage mediums. The switching impulse noise, thermal noise, cross talk and lightning are some examples of noise disturbance over a physical channel. A surface defect on a magnetic tape is a source of disturbance. The demodulator processes each received waveform and produces an output, which may be either continuous or discrete – the sequence **r**. The channel decoder transforms **r** into a binary sequence, $\hat{\boldsymbol{u}}$ which gives the estimate of **u**, and ideally should be the replica of **u**. The source decoder then transforms $\hat{\boldsymbol{u}}$ into an estimate of source output and delivers this to the destination.

Error control for data integrity may be exercised by means of '**forward error correction**' (**FEC**) where in the decoder performs **error correction** operation on the received information according to the schemes devised for the purpose. There is however another major approach known as '**Automatic Repeat Request**' (**ARQ**), in which a re-transmission of the ambiguous information is effected, is also used for solving error control problems. In **ARQ**, error correction is not done at all. The redundancy introduced is used only for '**error detection**' and upon detection, the receiver requests a repeat transmission which necessitates the use of a return path (feed back channel).

In summary, channel coding refers to a class of signal transformations designed to improve performance of communication systems by enabling the transmitted signals to better withstand the effect of various channel impairments such as noise, fading and jamming. Main objective of error control coding is to reduce the probability of error or reduce the $\mathbf{E_b/N_0}$ at the cost of expending more bandwidth than would otherwise be necessary. Channel coding is a very popular way of providing performance improvement. Use of **VLSI** technology has made it possible to provide as much as

8 – dB performance improvement through coding, at much lesser cost than through other methods such as high power transmitters or larger Antennas.

We will briefly discuss in this chapter the channel encoder and decoder strategies, our major interest being in the design and implementation of the channel '**encoder/decoder**' pair to achieve fast transmission of information over a noisy channel, reliable communication of information and reduction of the implementation cost of the equipment.

4.2 Discrete memory less channel:

Referring to the block diagram in Fig. 4.2 the channel is said to be memory less if the **de-modulator** (**Detector)** output in a given interval depends only on the signal transmitted in the interval, and not on any previous transmission. Under this condition, we may model (describe) the combination of the modulator – channel – and the demodulator as a "**Discrete memory less channel**". Such a channel is completely described by the set of transition probabilities $\mathbf{p\ (y_j \mid x_k)}$ where $\mathbf{x_k}$ is the modulator input symbol.

The simplest channel results from the use of binary symbols (both as input and output). When binary coding us used the modulator has only '**0**'`s and '**1**'`s as inputs. Similarly, the inputs to the demodulator also consists of '**0**'`s and '**1**'`s provided **binary quantization** is used. If so we say a '**Hard decision**' is made on the demodulator output so as to identify which symbol was actually transmitted. In this case we have a '**Binary symmetric channel**' (**BSC**). The **BSC** when derived from an additive white Gaussian noise (**AWGN**) channel is completely described by the transition probability '**p**'. The majority of coded digital communication systems employ binary coding with hard-decision decoding due to simplicity of implementation offered by such an approach.

The use of hard-decisions prior to decoding causes an irreversible loss of information in the receiver. To overcome this problem "**soft-decision**" coding is used. This can be done by including a **multilevel quantizer** at the demodulator output as shown in Fig. 4.2(a) for the case of binary **PSK** signals. The input-output characteristics and the channel transitions are shown in Fig. 4.2(b) and Fig. 4.2(c) respectively. Here the input to the demodulator has only two symbols '**0**'`s and '**1**'`s. However, the demodulator output has '**Q**' symbols. Such a channel is called a "**Binary input-Q-ary output DMC**". The form of channel transitions and hence the performance of the demodulator, depends on the location of representation levels of the quantizer, which inturn depends on the signal level and variance of noise. Therefore, the demodulator must incorporate automatic gain control, if an effective multilevel quantizer is to be realized. Further the soft-decision decoding offers significant improvement in performance over hard-decision decoding.

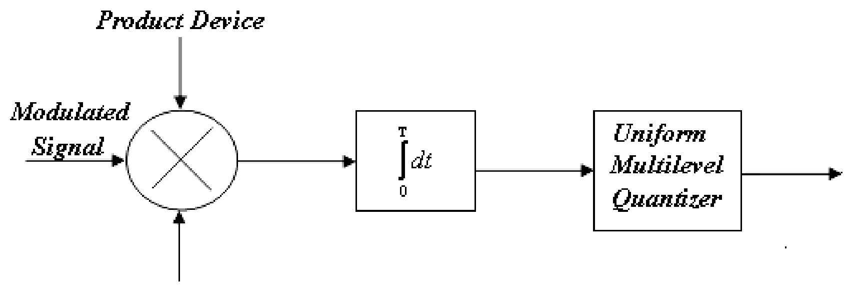
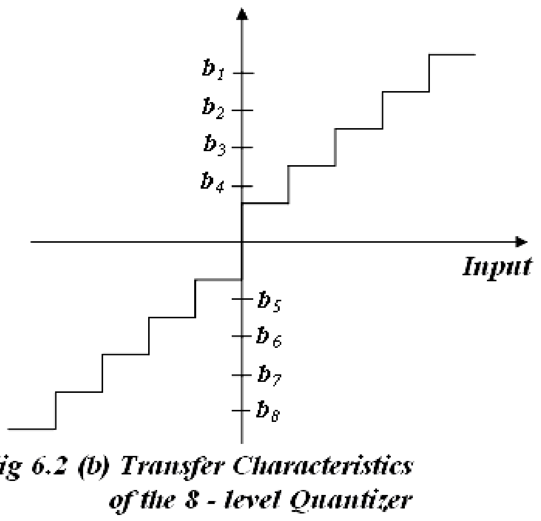


Fig. 4.2 (a) - Reciever



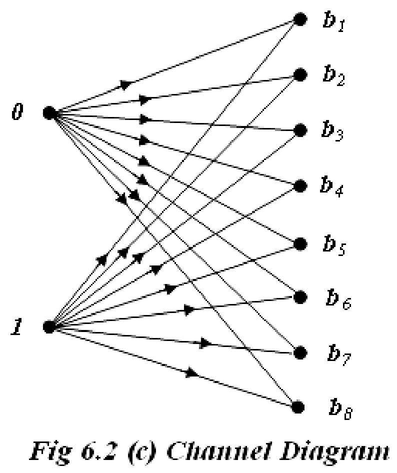


Fig 4.2 (b) Transfer characteristics (c) Channel Diagram

4.3 Shannon's theorem on channel capacity Revisited:

The "**Shannon's theorem on channel capacity**" is re-stated here and call it the "**Coding Theorem**".

"It is possible in principle, to devise a means where by a communication system will transmit information with an arbitrarily small probability of error, provided the information rate **R** (**=r I(X,Y)** where **r**-is the symbol rate) is less than or equal to a rate '**C**' called the 'channel capacity". The technique used to achieve this goal is called "**Coding**". For the special case of a **BSC**, the theorem tells us that if the code rate, $\mathbf{R_c}$ (defined later) is less than the channel capacity, then it is possible to find a code that achieves error free transmission over the channel. Conversely, it is not possible to find such a code if the code rate $\mathbf{R_c}$ is greater than **C**.

The channel coding theorem thus specifies the channel capacity as a "**Fundamental limit**" on the rate at which reliable transmission (error-free transmission) can take place over a **DMC**. Clearly, the issue that matters is not the signal to noise ratio (**SNR**), so long as it is large enough, but how the input is encoded.

The most un-satisfactory feature of Shannon's theorem is that it stresses only about the "existence of good codes". But it does not tell us how to find them. So, we are still faced with the task of finding a good code that ensures error-free transmission. The error-control coding techniques presented in this and subsequent chapters provide different methods of achieving this important system requirement.

4.4 Types of errors:

The errors that arise in a communication system can be viewed as '**independent errors**' and '**burst errors**'. The first type of error is usually encountered by the '**Gaussian noise**', which is the chief concern in the design and evaluation of modulators and demodulators for data transmission. The possible sources are the thermal noise and shot noise of the transmitting and receiving equipment, thermal noise in the channel and the radiations picked up by the receiving antenna. Further, in majority situations, the power spectral density of the Gaussian noise at the receiver input is white. The transmission errors introduced by this noise are such that the error during a particular signaling interval does not affect the performance of the system during the subsequent intervals. The discrete channel, in this case, can be modeled by a Binary symmetric channel. These transmission errors due to Gaussian noise are referred to as '**independent errors**' (**or random errors**).

The second type of error is encountered due to the '**impulse noise**', which is characterized by long quiet intervals followed by high amplitude **noise bursts** (As in switching and lightning). A noise burst usually affects more than one symbol and there will be dependence of errors in successive transmitted symbols. Thus errors occur in bursts

4.5 Types of codes:

There are mainly two types of error control coding schemes – **Block codes** and **convolutional codes**, which can take care of either type of errors mentioned above.

In a block code, the information sequence is divided into message blocks of **k** bits each, represented by a binary **k**-tuple, $\mathbf{u} = (\mathbf{u_1}, \mathbf{u_2} \ldots \mathbf{u_k})$ and each block is called a message. The symbol **u**, here, is used to denote a **k** – bit message rather than the entire information sequence. The encoder then transforms **u** into an **n**-tuple $\mathbf{v} = (\mathbf{v_1}, \mathbf{v_2} \ldots \mathbf{v_n})$. Here **v** represents an encoded block rather than the entire encoded sequence. The blocks are independent of each other.

The encoder of a convolutional code also accepts **k**-bit blocks of the information sequence **u** and produces an **n**-symbol block **v**. Here **u** and **v** are used to denote sequences of blocks rather than a single block. Further each encoded block depends not only on the present **k**-bit message block but also on **m**-pervious blocks. Hence the encoder has a memory of order '**m**'. Since the encoder has memory, implementation requires sequential logic circuits.

If the code word with **n**-bits is to be transmitted in no more time than is required for the transmission of the **k**-information bits and if $\boldsymbol{\tau_b}$ and $\boldsymbol{\tau_c}$ are the bit durations in the encoded and coded words, i.e. the input and output code words, then it is necessary that

$$\mathbf{n.\tau_c = k.\tau_b}$$

We define the "**rate of the code**" by (also called **rate efficiency**)

$$R_c \triangleq \frac{k}{n}$$

Accordingly, with $f_b = 1/\tau_b$ and $f_c = 1/\tau_c$, we have $\frac{f_b}{f_c} = \frac{\tau_c}{\tau_b} = \frac{k}{n} = R_c$

4.6 Example of Error Control Coding:

Better way to understand the important aspects of error control coding is by way of an example. Suppose that we wish transmit data over a telephone link that has a useable bandwidth of **4 KHZ** and a maximum **SNR** at the out put of **12 dB**, at a rate of **1200 bits/sec** with a probability of error less than 10^{-3}. Further, we have **DPSK** modem that can operate at speeds of **1200**, **1400** and **3600 bits/sec** with error probabilities $2\times(10^{-3})$, $4\times(10^{-3})$ and $8\times(10^{-3})$ respectively. We are asked to design an error control coding scheme that would yield an overall probability of error < 10^{-3}. We have:

C = **16300 bits/sec**, R_c = **1200**, **2400** or **3600 bits/sec.**

[C=$B\log_2\left(1+\frac{S}{N}\right)$. $\frac{S}{N} = 12dB\ or\ 15.85$, B=4KHZ], $p = 2(10^{-3})$, $4(10^{-3})$ and $8(10^{-3})$ respectively. Since $R_c < C$, according to Shannon's theorem, we should be able to transmit data with arbitrarily small probability of error. We shall consider two coding schemes for this problem.

(i) **Error detection**: Single parity check-coding. Consider the (**4, 3**) even parity check code.

Message	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
Parity	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**1**
Codeword	**0000**	**0011**	**0101**	**0110**	**1001**	**1010**	**1100**	**1111**

Parity bit appears at the right most symbol of the codeword.

This code is capable of '**detecting**' all single and triple error patterns. Data comes out of the channel encoder at a rate of **3600 bits/sec** and at this rate the modem has an error probability of $8\times(10^{-3})$. The decoder indicates an error only when parity check fails. This happens for single and triple errors only.

p_d = Probability of error detection.

= $p(X = 1) + p(X = 3)$, where **X** = Random variable of errors.

Using binomial probability law, we have with $p = 8(10^{-3})$:

$$\mathbf{P}(\mathbf{X} = \mathbf{k}) = \binom{n}{k} p^k (1-p)^{n-k}$$

$$p_d = \binom{4}{1} p(1-p)^3 + \binom{4}{3} p^3(1-p), \quad \binom{4}{1} = 4C_1 = 4, \binom{4}{3} = 4C_3 = 4$$

Expanding we get $p_d = 4p - 12p^2 + 16p^3 - 8p^4$

Substituting the value of **p** we get:

$$\mathbf{p_d = 32\times (10^{-3}) - 768\times (10^{-6}) + 8192\times (10^{-9}) - 32768 \times (10^{-12}) = 0.031240326 >> (10^{-3})}$$

However, an error results if the decoder does not indicate any error when an error indeed has occurred. This happens when **two** or **4** errors occur. Hence probability of a detection error = $\mathbf{p_{nd}}$ (probability of no detection) is given by:

$$p_{nd} = P(X = 2) + P(X = 4) = \binom{4}{2} p^2(1-p)^2 + \binom{4}{4} p^4(1-p)^0 = 6p^2 - 12p^3 + 7p^4$$

Substituting the value of **p** we get $\mathbf{p_{nd} = 0.4\times 10^{-3} < 10^{-3}}$

Thus probability of error is less than $\mathbf{10^{-3}}$ as required.

(ii) **Error Correction:** The triplets **000** and **111** are transmitted whenever **0** and **1** are inputted. A majority logic decoding, as shown below, is employed assuming only single errors.

Received Triplet	**000**	**001**	**010**	**100**	**011**	**101**	**110**	**111**
Output message	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**

Probability of decoding error, $\mathbf{p_{de}}$= **P** (two or more bits in error)

$$= \binom{3}{2} \mathbf{p^2 (1-p)} + \binom{3}{3} \mathbf{p^3 (1-p)^0 = 3p^2 - 2p^3}$$

$$=190.464 \times 10^{-6} = 0.19 \times 10^{-3} < p = 10^{-3}$$

Probability of no detection, $\mathbf{p_{nd}}$ =**P** (All 3 bits in error) = $\mathbf{p^3 = 512 \times 10^{-9} << p_{de}}$**!**

In general observe that probability of no detection, $\mathbf{p_{nd}} <<$ probability of decoding error, $\mathbf{p_{de}}$.

The preceding examples illustrate the following aspects of error control coding. Note that in both examples with out error control coding the probability of error =$\mathbf{8\times(10^{-3})}$ of the modem.

1. It is possible to detect and correct errors by adding extra bits-the check bits, to the message sequence. Because of this, not all sequences will constitute bonafied messages.

2. It is not possible to detect and correct all errors.

3. Addition of check bits reduces the effective data rate through the channel.

4. Since probability of no detection is always very much smaller than the decoding error probability, it appears that the error detection schemes, which do not reduce the rate efficiency as the error correcting schemes do, are well suited for our application. Since error detection schemes always go with **ARQ** techniques, and when the speed of communication becomes a major concern, Forward error correction (**FEC**) using error correction schemes would be desirable.

4.7 Block codes:

We shall assume that the output of an information source is a sequence of Binary digits. In 'Block coding' this information sequence is segmented into 'message' blocks of fixed length, say **k**. Each message block, denoted by **u** then consists of **k** information digits. The encoder transforms these **k**-tuples into blocks of code words **v**, each an **n**- tuple 'according to certain rules'. Clearly, corresponding to $\mathbf{2^k}$ information blocks possible, we would then have $\mathbf{2^k}$ code words of length **n** > **k**. This set of $\mathbf{2^k}$ code words is called a "**Block code**". For a block code to be useful these $\mathbf{2^k}$ code words must be distinct, i.e. there should be a one-to-one correspondence between **u** and **v**. **u** and **v** are also referred to as the '**input vector**' and '**code vector**' respectively. Notice that encoding equipment must be capable of storing the $\mathbf{2^k}$ code words of length **n** > **k**. Accordingly, the complexity of the equipment would become prohibitory if **n** and **k** become large unless the code words have a special structural property conducive for storage and mechanization. This structural is the '**linearity**'.

4.7.1 Linear Block Codes:

A block code is said to be linear **(n ,k)** code if and only if the $\mathbf{2^k}$ code words from a **k**-dimensional sub space over a vector space of all **n**-Tuples over the field **GF(2)**.

Fields with $\mathbf{2^m}$ symbols are called 'Galois Fields' (pronounced as Galva fields), GF ($\mathbf{2^m}$).Their arithmetic involves binary additions and subtractions. For two valued variables, (**0, 1**).The modulo – 2 addition and multiplication is defined in Fig 4.3.

		Y	
		0	*1*
X	*0*	*0*	*1*
	1	*1*	*0*

$X \oplus Y$

$\otimes$		*Y*	
		0	*1*
X	*0*	*0*	*0*
	0	*1*	*1*

$X \otimes Y$

Fig 4.3

The binary alphabet (**0, 1**) is called a **field** of two elements (a binary field and is denoted by **GF (2)**. (Notice that ⊕ represents the EX-OR operation and ⊗ represents the AND operation).Further in binary arithmetic, **–X=X** and **X – Y = X ⊕ Y**. similarly for 3-valued variables, modulo – 3 arithmetic can be specified as shown in Fig 6.4. However, for brevity while representing polynomials involving binary addition we use + instead of ⊕ and there shall be no confusion about such usage**.**

Polynomials **f(X)** with **1** or **0** as the co-efficients can be manipulated using the above relations. The arithmetic of **GF(2^m)** can be derived using a polynomial of degree '**m**', with binary co-efficients and using a new variable α called the primitive element, such that **p(α) = 0**.When **p(X)** is irreducible (i.e. it does not have a factor of degree < **m** and >**0**, for example $X^3 + X^2 + 1$, $X^3 + X + 1$, $X^4 + X^3 + 1$, $X^5 + X^2 + 1$ etc. are irreducible polynomials, whereas $f(X)=X^4+X^3+X^2+1$ is not as **f(1)** = **0** and hence has a factor **X+1**) then **p(X)** is said to be a '**primitive polynomial**'.

If v_n represents a vector space of all **n**-tuples, then a subset **S** of v_n is called a subspace if (i) the all Zero vector is in **S** (ii) the sum of any two vectors in **S** is also a vector in **S**. To be more specific, a block code is said to be linear if the following is satisfied. "If v_1 and v_2 are any two code words of length **n** of the block code then $v_1 \oplus v_2$ is also a code word length **n** of the block code".

Example 4.1: Linear Block code with **k= 3**, and **n =6**

Messages	*Code words*	*Weight (No. of 1's in the code word)*
m_1 000	v_1 0 0 0 0 0 0	0
m_2 001	v_2 0 0 1 1 1 0	3
m_3 010	v_3 0 1 0 1 0 1	3
m_4 100	v_4 1 0 0 0 1 1	3
m_5 011	v_5 0 1 1 0 1 1	4
m_6 101	v_6 1 0 1 1 0 1	4
m_7 110	v_7 1 1 0 1 1 0	4
m_8 111	v_8 1 1 1 0 0 0	3

Observe the linearity property: With v_3 = (**010 101**) and v_4 = (**100 011**), $v_3 \oplus v_4$ = (**110 110**) = v_7**.**

Remember that **n** represents the word length of the code words and **k** represents the number of information digits and hence the block code is represented as **(n, k)** block code.

Thus by definition of a linear block code it follows that if $\mathbf{g_1}, \mathbf{g_2} \ldots \mathbf{g_k}$ are the **k** linearly independent code words then every code vector, **v**, of our code is a combination of these code words, i.e.

$$\mathbf{v} = \mathbf{u_1\, g_1 \oplus u_2\, g_2 \oplus \ldots \oplus u_k\, g_k} \qquad \ldots\ldots\ldots\ldots\ldots\ldots \qquad (4.1)$$

Where $\mathbf{u_j} = \mathbf{0}$ or $\mathbf{1}, \forall\ \boldsymbol{1 \leq j \leq k}$

Eq (6.1) can be arranged in matrix form by nothing that each $\mathbf{g_j}$ is an n-tuple, i.e.

$$\mathbf{g_j = (g_{j1}, g_{j2}, \ldots . g_{jn})} \qquad \ldots\ldots\ldots\ldots\ldots\ldots \qquad (4.2)$$

Thus we have $\mathbf{v = u\, G}$ $\ldots\ldots\ldots\ldots\ldots\ldots$ (4.3)

Where: $\mathbf{u = (u_1, u_2 \ldots u_k)}$ $\ldots\ldots\ldots\ldots\ldots\ldots$ (4.4)
represents the data vector and

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{g_1} \\ \boldsymbol{g_2} \\ \boldsymbol{g_3} \end{bmatrix} \begin{bmatrix} \boldsymbol{g_{11}} & \boldsymbol{g_{12}} & \cdots \boldsymbol{g_{1n}} \\ \boldsymbol{g_{21}} & \boldsymbol{g_{22}} & \cdots \boldsymbol{g_{2n}} \\ \vdots & \vdots & \\ \boldsymbol{g_{k1}} & \boldsymbol{g_{k2}} & \cdots \boldsymbol{g_{kn}} \end{bmatrix} \qquad \ldots\ldots\ldots\ldots\ldots\ldots \qquad (4.5)$$

is called the "**generator matrix**".

Notice that any **k** linearly independent code words of an **(n, k)** linear code can be used to form a Generator matrix for the code. Thus it follows that an **(n, k)** linear code is completely specified by the **k**-rows of the generator matrix. Hence the encoder need only to store **k** rows of **G** and form linear combination of these rows based on the input message **u.**

Example 4.2: The (**6, 3**) linear code of Example 6.1 has the following generator matrix:

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{g_1} \\ \boldsymbol{g_2} \\ \boldsymbol{g_3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

If $\mathbf{u = m_5}$ (say) is the message to be coded, i.e. **u** = (**011**)

We have $\mathbf{v = u\, .G} = \mathbf{0.g_1} + \mathbf{1.g_2} + \mathbf{1.g_3}$

$$= (\mathbf{0,0,0,0,0,0}) + (\mathbf{0,1,0,1,0,1}) + (\mathbf{0,0,1,1,1,0}) = (\mathbf{0, 1, 1, 0, 1, 1})$$

Thus **v = (0 1 1 0 1 1)**

"**v** can be computed simply by adding those rows of **G** which correspond to the locations of **1**`s of **u**."

4.7.2 Systematic Block Codes (Group Property):

A desirable property of linear block codes is the "Systematic Structure". Here a code word is divided into two parts –Message part and the redundant part. If either the first **k** digits or the last **k** digits of the code word correspond to the message part then we say that the code is a "Systematic Block Code". We shall consider systematic codes as depicted in Fig.4.5.

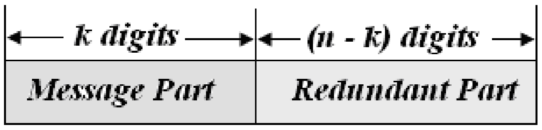


Fig 4.5 Systematic format of code word

In the format of Fig.4.5 notice that:

$$\mathbf{v_1 = u_1, v_2 = u_2, v_3 = u_3 \ldots v_k = u_k} \qquad \ldots\ldots\ldots\ldots\ldots \qquad (4.6\text{ a})$$

$$\left.\begin{aligned} v_{k+1} &= u_1\, p_{11} + u_2\, p_{21} + u_3\, p_{31} + \ldots + u_k\, p_{k1} \\ v_{k+2} &= u_1\, p_{12} + u_2\, p_{22} + u_3\, p_{32} + \ldots + u_k\, p_{k2} \\ &\quad\vdots \qquad\quad \vdots \\ v_n &= u_1 p_{1,n-k} + u_2 p_{2,n-k} + u_3\, p_{3,n-k} + \ldots\ldots + u_k\, p_{k,n-k} \end{aligned}\right\} \qquad \ldots\ldots\ldots\ldots\ldots\ldots \qquad (4.6\text{ b})$$

Or in matrix from we have

$$[v_1 \quad v_2 \quad \ldots \quad v_k \quad v_{k+1} \quad v_{k+2} \quad \ldots \quad v_n] =$$

$$[u_1 \quad u_2 \quad \ldots \quad u_k]\begin{bmatrix} 1 & 0 & 0 & \ldots & 0 & p_{11} & p_{12} & \ldots & p_{1,n-k} \\ 0 & 1 & 0 & \ldots & 0 & p_{21} & p_{22} & \ldots & p_{2,n-k} \\ \vdots & \vdots & \vdots & \vdots\vdots\vdots & \vdots & \vdots & \vdots & \vdots\vdots\vdots & \vdots \\ 0 & 0 & 0 & \ldots & 1 & p_{k1} & p_{k2} & \ldots & p_{k,n-k} \end{bmatrix} \ldots\ldots\ldots \qquad (4.7)$$

i.e., **v = u.G**

Where **G** = [**I**$_k$, **P**] ……………… (4.8)

$$\text{Where} \quad \mathbf{P} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1,n-k} \\ p_{21} & p_{22} & \cdots & p_{2,n-k} \\ \vdots & \vdots & & \vdots \\ p_{k,1} & p_{k,2} & \cdots & pk_{,n-k} \end{bmatrix} \qquad \ldots\ldots\ldots\ldots\ldots\ldots \qquad (4.9)$$

$\mathbf{I_k}$ is the $\mathbf{k \times k}$ identity matrix (unit matrix), **P** is the $\mathbf{k \times (n - k)}$ '**parity generator matrix**', in which $\mathbf{p_{i,\,j}}$ are either **0** or **1** and **G** is a $\mathbf{k \times n}$ matrix. The **(n – k)** equations given in Eq (4.6b) are referred to as **parity check equations**. Observe that the **G** matrix of Example 4.2 is in the systematic format.The **n**-vectors $\mathbf{a = (a_1, a_2 \ldots a_n)}$ and $\mathbf{b = (b_1, b_2 \ldots b_n)}$ are said to be orthogonal if their inner product defined by:

$$\mathbf{a.b = (a_1, a_2 \ldots a_n)\,(b_1, b_2 \ldots b_n)^T = 0.}$$

where, '**T**' represents transposition. Accordingly for any **k×n** matrix, **G**, with **k** linearly independent rows there exists a **(n-k) × n** matrix **H** with **(n-k)** linearly independent rows such that any vector in the row space of **G** is orthogonal to the rows of **H** and that any vector that is orthogonal to the rows of **H** is in the row space of **G**. Therefore, we can describe an **(n, k)** linear code generated by **G** alternatively as follows:

"An n – tuple, v is a code word generated by G, if and only if $\mathbf{v.H^T = O}$". ……… (4.9a)
(**O represents an all zero row vector**.)

This matrix **H** is called a "**parity check matrix**" of the code. Its dimension is $\mathbf{(n - k) \times n}$.

If the generator matrix has a systematic format, the parity check matrix takes the following form.

$$\mathbf{H = [P^T.I_{n-k}]} = \begin{bmatrix} p_{11} & p_{21} & \cdots & p_{k1} & 1 & 0 & 0 & \cdots & 0 \\ p_{12} & p_{22} & \cdots & p_{k2} & 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ p_{1,n-k} & p_{2,n-k} & \cdots & p_{k,n-k} & 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \quad \ldots\ldots\ldots \quad (4.10)$$

The **i**th row of **G** is:

$$\mathbf{g_i = (0\ 0 \ldots \underset{\substack{\uparrow \\ i^{th} \text{ element}}}{1} \ldots 0 \ldots 0 \quad p_{i,1} \quad p_{i,2} \ldots \underset{\substack{\uparrow \\ (k+j)^{th} \text{ element}}}{p_{i,j}} \ldots p_{i,\,n-k})}$$

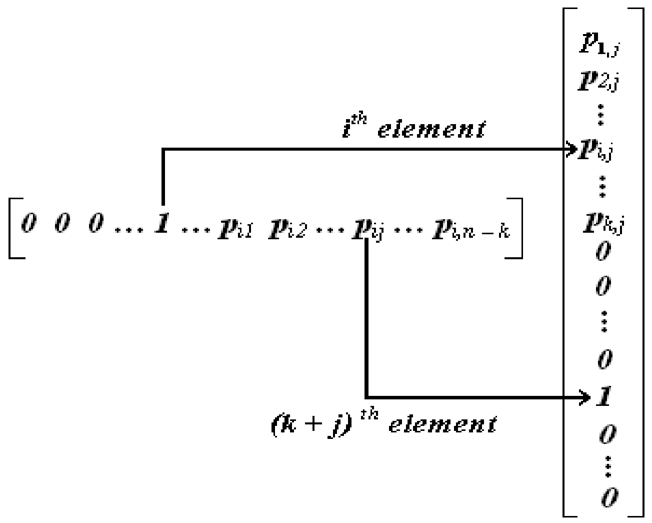
The **j**th row of **H** is:

$$\mathbf{h_j = (\ p_{1,j}\ p_{2,j} \ldots \overset{\substack{i^{th} \text{ element} \\ \downarrow}}{p_{i,j}} \ldots p_{k,\,j}\ 0\ 0 \ldots 0\ \overset{\substack{(k+j)^{th} \text{ element} \\ \downarrow}}{1}\ 0 \ldots 0)}$$

Accordingly the inner product of the above **n** – vectors is:

$$\mathbf{g_i \times h_j = (0\ 0 \ldots \underset{\uparrow}{1} \ldots 0 \ldots 0 \quad p_{i,1} \quad p_{i,2} \ldots \underset{\uparrow}{p_{i,j}} \ldots p_{i,\,n-k})\ (\ p_{1,j}\ p_{2,j} \ldots \underset{\uparrow}{p_{i,j}} \ldots p_{k,\,j}\ 0\ 0 \ldots 0\ \underset{\uparrow}{1}\ 0 \ldots 0)^T}$$

ith element **(k + j)** th element **i**th element **(k + j)** th element



= p_{ij} + p_{ij}. = 0 (as the p_{ij} are either **0** or **1** and in modulo – 2 arithmetic **X** + **X** = 0)

This implies simply that:

$$G.\,H^T = O_{k\times(n-k)} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (4.11)$$

Where $O_{k\times(n-k)}$ is an all zero matrix of dimension **k ×(n – k)**.

Further, since the **(n – k)** rows of the matrix **H** are linearly independent, the **H** matrix of Eq. (4.10) is a parity check matrix of the **(n, k)** linear systematic code generated by **G**. Notice that the parity check equations of Eq. (4.6b) can also be obtained from the parity check matrix using the fact

$$v.H^T = O.$$

Alternative Method of proving $v.H^T = O.$:

We have $v = u.G = u.[I_k : P] = [u_1, u_2 \ldots u_k, p_1, p_2 \ldots. P_{n-k}]$

Where $p_i = (u_1 p_{1,i} + u_2 p_{2,i} + u_3 p_{3,i} \ldots + u_k p_{k,i})$ are the parity bits found from Eq (4.6b).

Now $H^T = \begin{bmatrix} P \\ I_{n-k} \end{bmatrix}$

$$\therefore v.H^T = [u_1 p_{11} + u_2 p_{21} + \ldots. + \ldots. + u_k p_{k1} + p_1,\ u_1 p_{12} + u_2 p_{22} + \ldots.. + u_k p_{k2} + p_2,\ \ldots$$
$$u_1 p_{1,n-k} + u_2 p_{2,n-k} + \ldots. + u_k p_{k,n-k} + p_{n-k}]$$

$$= [p_1 + p_1, p_2 + p_2 \ldots p_{n-k} + p_{n-k}]$$

= [0, 0… 0]

Thus $\mathbf{v}.\ \mathbf{H}^T = \mathbf{O}$. This statement implies that an **n**- Tuple **v** is a code word generated by **G** if and only if

$$\mathbf{v}\ \mathbf{H}^T = \mathbf{O}$$

Since $\mathbf{v} = \mathbf{u}\ \mathbf{G}$, This means that: $\mathbf{u}\ \mathbf{G}\ \mathbf{H}^T = \mathbf{O}$

If this is to be true for any arbitrary message vector **v** then this implies: $\mathbf{G}\ \mathbf{H}^T = \mathbf{O}_{k\times(n-k)}$

Example 4.3:

Consider the generator matrix of Example 4.2, the corresponding parity check matrix is

$$\mathbf{H} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

4.7.3 Circuit implementation of Block codes:

The implementation of Block codes is very simple. We need only combinational logic circuits. Implementation of Eq (4.6) is shown in the encoding circuit of Fig.4.6. Notice that p_{ij} is either a '**0**' or a '**1**' and accordingly $\rightarrow p_{ij} \rightarrow$ indicates a connection if $p_{ij} = 1$ only (otherwise no connection). The encoding operation is very simple. The message $\mathbf{u} = (u_1, u_2 \ldots u_k)$ to be encoded is shifted into the message register and simultaneously into the channel via the commutator. As soon as the entire message has entered the message register, the parity check digits are formed using modulo -2 adders, which may be serialized using, another shift register – the parity register, and shifted into the channel. Notice that the complexity of the encoding circuit is directly proportional to the block length of the code. The encoding circuit for the (**6, 3**) block code of Example 2 is shown in Fig 4.7

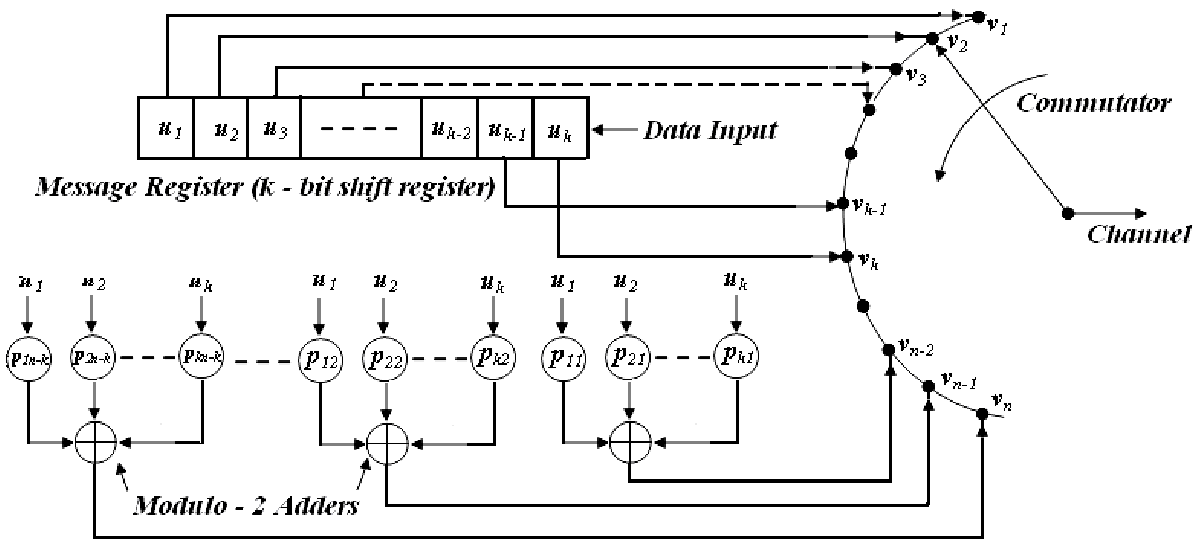


Fig 4.6 Encoding circuit for systematic block code

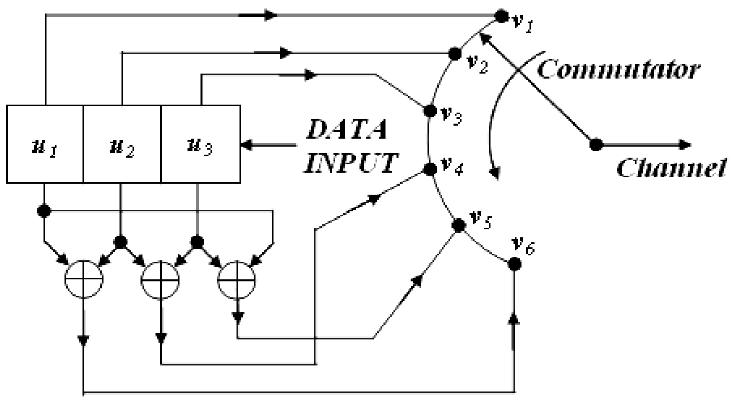


Fig 4.7 Encoder for the (6,3) block code of example 4.2

4.7.4 Syndrome and Error Detection:

Suppose $\mathbf{v} = (v_1, v_2 \ldots v_n)$ be a code word transmitted over a noisy channel and let:
$\mathbf{r} = (r_1, r_2 \ldots r_n)$ be the received vector. Clearly, **r** may be different from **v** owing to the channel noise. The vector sum

$$\mathbf{e} = \mathbf{r} - \mathbf{v} = (e_1, e_2 \ldots e_n) \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \qquad (4.12)$$

is an **n**-tuple, where $e_j = 1$ if $r_j \neq v_j$ and $e_j = 0$ if $r_j = v_j$. This **n** – tuple is called the "**error vector**" or "**error pattern**". The **1**'s in **e** are the transmission errors caused by the channel noise. Hence from Eq (4.12) it follows:

$$\mathbf{r} = \mathbf{v} \oplus \mathbf{e} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \qquad (4.12a)$$

Observer that the receiver noise does not know either **v** or **e**. Accordingly, on reception of **r** the decoder must first identify if there are any transmission errors and, then take action to locate these errors and correct them (**FEC** – Forward Error Correction) or make a request for re–transmission (**ARQ**). When **r** is received, the decoder computes the following **(n-k)** tuple:

$$\mathbf{s} = \mathbf{r}.\, H^T = (s_1, s_2 \ldots s_{n-k}) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (4.13)$$

It then follows from Eq (4.9a), that **s = 0** if and only if **r** is a code word and **s ≠ 0** iffy **r** is not a code word. This vector s is called "**The Syndrome**" (a term used in medical science referring to collection of all symptoms characterizing a disease). Thus if **s = 0**, the receiver accepts **r** as a valid code word. Notice that there are possibilities of errors undetected, which happens when **e** is identical to a nonzero code word. In this case **r** is the sum of two code words which according to our linearity property is again a code word. This type of error pattern is referred to an "**undetectable error pattern**". Since there are $\mathbf{2^k - 1}$ nonzero code words, it follows that there are $\mathbf{2^k - 1}$ error patterns as well. Hence when an undetectable error pattern occurs the decoder makes a "**decoding error**".
Eq. (4.13) can be expanded as below:

$$s = r.\, H^T = (s_1, s_2 \ldots s_{n-k}) = (r_1, r_2, \ldots, r_n) \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1,n-k} \\ p_{21} & p_{22} & \cdots & p_{2,n-k} \\ \vdots & & & \\ p_{k,1} & p_{k,2} & \cdots & p_{k,n-k} \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & & & \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

From which we have

$$\left.\begin{aligned} s_1 &= r_1 p_{11} + r_2 p_{21} + \ldots. + r_k p_{k1} + r_{k+1} \\ s_2 &= r_1 p_{12} + r_2 p_{22} + \ldots. + r_k p_{k2} + r_{k+2} \\ &\vdots \quad\quad \vdots \quad\quad \vdots \quad\quad \vdots \quad\quad \vdots \\ s_{n-k} &= r_1 p_{1,n-k} + r_2 p_{2,n-k} + \ldots. + r_k p_{k,n-k} + r_n \end{aligned}\right\} \quad \ldots\ldots\ldots\ldots \quad (4.14)$$

A careful examination of Eq. (4.14) reveals the following point. The syndrome is simply the vector sum of the received parity digits $(\mathbf{r_{k+1}, r_{k+2} \ldots r_n})$ and the parity check digits recomputed from the received information digits $(\mathbf{r_1, r_2 \ldots r_n})$. Thus, we can form the syndrome by a circuit exactly similar to that of Fig.6.6 and a general syndrome circuit is as shown in Fig. 4.8.

Example 4.4:

We shall compute the syndrome for the (**6, 3**) systematic code of Example 4.2. We have

$$\mathbf{s} = (s_1, s_2, s_3) = (r_1, r_2, r_3, r_4, r_5, r_6) \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

or $\mathbf{s_1 = r_2 + r_3 + r_4}$

$\mathbf{s_2 = r_1 + r_3 + r_5}$

$\mathbf{s_3 = r_1 + r_2 + r_6}$

The syndrome circuit for this code is given in Fig.4.9.

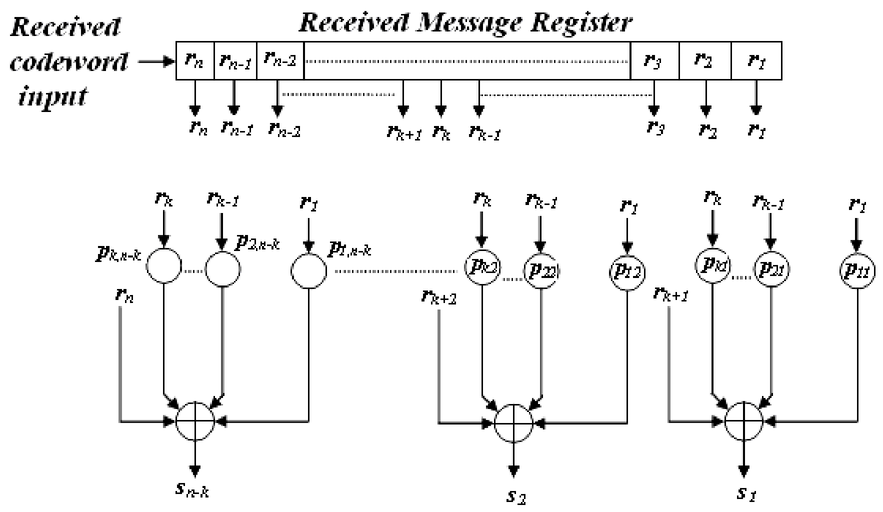

Received codeword input
Received Message Register
r_n r_{n-1} r_{n-2} r_3 r_2 r_1
r_{k+1} r_k r_{k-1}
$p_{k,n-k}$ $p_{2,n-k}$ $p_{1,n-k}$
p_{k2} p_{22} p_{12} p_{k1} p_{21} p_{11}
r_{k+2}
s_{n-k} s_2 s_1

Fig 4.8 Syndrome circuit for the (n,k) Linear systematic block code

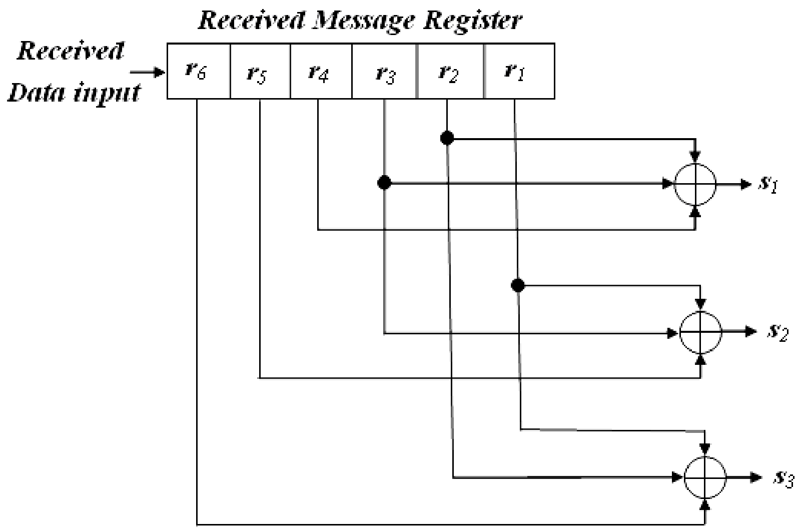

Received Message Register
Received Data input
r_6 r_5 r_4 r_3 r_2 r_1
s_1
s_2
s_3

Fig 4.8 Syndrome circuit for the (6,3) systematic block code

In view of Eq. (4.12a), and Eq. (4.9a) we have

$$\mathbf{s = r.H^T = (v \oplus e)\, H^T}$$

$$\mathbf{= v\,.H^T \oplus e.H^T}$$

or $\mathbf{s = e.H^T}$ …………… (4.15)

as $\mathbf{v.H^T = O}$. Eq. (4.15) indicates that the syndrome depends only on the error pattern and not on the transmitted code word **v**. For a linear systematic code, then, we have the following relationship between the syndrome digits and the error digits.

$$\left.\begin{array}{l} s_1 = e_1 p_{11} + e_2 p_{21} + \ldots + e_k p_{k,1} + e_{k+1} \\ s_2 = e_1 p_{12} + e_2 p_{22} + \ldots + e_k p_{k,2} + e_{k+2} \\ \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ s_{n-k} = e_1 p_{1,n-k} + e_2 p_{2,n-k} + \ldots + e_k p_{k,n-k} + e_n \end{array}\right\} \quad \ldots\ldots\ldots\ldots \quad (4.16)$$

Thus, the syndrome digits are linear combinations of error digits. Therefore they must provide us information about the error digits and help us in error correction.

Notice that Eq. (4.16) represents **(n-k)** linear equations for **n** error digits – an under-determined set of equations. Accordingly it is not possible to have a unique solution for the set. As the rank of the **H** matrix is **k**, it follows that there are $\mathbf{2^k}$ non-trivial solutions. In other words there exist $\mathbf{2^k}$ error patterns that result in the same syndrome. Therefore to determine the true error pattern is not any easy task!

Example 4.5:

For the (**6, 3**) code considered in Example 4.2, the error patterns satisfy the following equations:

$$s_1 = e_2 + e_3 + e_4, \quad s_2 = e_1 + e_3 + e_5, \quad s_3 = e_1 + e_2 + e_6$$

Suppose, the transmitted and received code words are **v = (0 1 0 1 0 1), r = (0 1 1 1 0 1)**

Then $\mathbf{s = r.H^T = (1, 1, 0)}$

Then it follows that:

$$\begin{array}{l} e_2 + e_3 + e_4 = 1 \\ e_1 + e_3 + e_5 = 1 \\ e_1 + e_2 + e_6 = 0 \end{array}$$

There are $\mathbf{2^3 = 8}$ error patterns that satisfy the above equations. They are:

{<u>0 0 1 0 0 0</u>, 1 0 0 0 0, 0 0 0 1 1 0, 0 1 0 0 1 1, 1 0 0 1 0 1, 0 1 1 1 0 1, 1 0 1 0 1 1, 1 1 1 1 1 0}

To minimize the decoding error, the "**Most probable error pattern**" that satisfies Eq (4.16) is chosen as the true error vector. For a **BSC**, the most probable error pattern is the one that has the smallest number of nonzero digits. For the Example 4.5, notice that the error vector **(0 0 1 0 0 0)** has the smallest number of nonzero components and hence can be regarded as the most probable error vector. Then using Eq. (4.12) we have

$$\hat{v} = \mathbf{r} \oplus \mathbf{e}$$

$$= (0\ 1\ 1\ 1\ 0\ 1) + (0\ 0\ 1\ 0\ 0\ 0) = (0\ 1\ 0\ 1\ 0\ 1)$$

Notice now that $\hat{v}$ indeed is the actual transmitted code word.

4.8 Minimum Distance Considerations:

The concept of distance between code words and single error correcting codes was first developed by R .W. Hamming. Let the n-tuples,

$$\alpha = (\alpha_1, \alpha_2 \ldots \alpha_n),\ \beta = (\beta_1, \beta_2 \ldots \beta_n)$$

be two code words. The "**Hamming distance**" **d (α,β)** between such pair of code vectors is defined as the number of positions in which they differ. Alternatively, using Modulo-2 arithmetic, we have

$$d(\alpha,\beta) \triangleq \sum_{j=1}^{n} (\alpha_j \oplus \beta_j) \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \qquad (4.17)$$

(Notice that **Σ** represents the usual decimal summation and ⊕ is the modulo-2 sum, the EX-OR function).

The "**Hamming Weight**" **ω(α)** of a code vector **α** is defined as the number of nonzero elements in the code vector. Equivalently, the Hamming weight of a code vector is the distance between the code vector and the '**all zero code vector**'.

Example 4.6: Let **α = (0 1 1 1 0 1), β = (1 0 1 0 1 1)**

Notice that the two vectors differ in **4** positions and hence **d (α,β) = 4**. Using Eq (4.17) we find

$$d(\alpha,\beta) = (0 \oplus 1) + (1 \oplus 0) + (1 \oplus 1) + (1 \oplus 0) + (0 \oplus 1) + (1 \oplus 1)$$

$$= 1 + 1 + 0 + 1 + 1 + 0$$

$$= 4$$ ….. (Here + is the algebraic plus not modulo – 2 sum)

Further, ω(α) = 4 and **ω(β) = 4.**

The "**Minimum distance**" of a linear block code is defined as the smallest Hamming distance between any pair of code words in the code or the minimum distance is the same as the smallest Hamming weight of the difference between any pair of code words. Since in linear block codes, the sum or difference of two code vectors is also a code vector, it follows then that **"the minimum distance of a linear block code is the smallest Hamming weight of the nonzero code vectors in the code".**

The Hamming distance is a metric function that satisfies the triangle inequality. Let α,β and δ be three code vectors of a linear block code. Then

$$\mathbf{d}(\alpha,\beta) + \mathbf{d}(\beta,\delta) \geq \mathbf{d}(\alpha,\delta) \quad \ldots\ldots\ldots\ldots\ldots \quad (4.18)$$

From the discussions made above, we may write

$$\mathbf{d}(\alpha,\beta) = \omega(\alpha \oplus \beta) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (4.19)$$

Example 4.7: For the vectors α and β of Example 4.6, we have:

$$\alpha \oplus \beta = (0\oplus1), (1\oplus0), (1\oplus1)\ (1\oplus0), (0\oplus1)\ (1\oplus1) = (11\ 0\ 1\ 1\ 0)$$

$$\therefore\ \omega(\alpha \oplus\beta) = 4 = \mathbf{d}(\alpha,\beta)$$

If $\delta = (1\ 0\ 1\ 01\ 0)$, we have $\mathbf{d}(\alpha,\beta) = 4$; $\mathbf{d}(\beta,\delta) = 1$; $\mathbf{d}(\alpha,\delta) = 5$

Notice that the above three distances satisfy the triangle inequality:

$$\mathbf{d}(\alpha,\beta) + \mathbf{d}(\beta,\delta) = 5 = \mathbf{d}(\alpha,\delta)$$

$$\mathbf{d}(\beta,\delta) + \mathbf{d}(\alpha,\delta) = 6 > \mathbf{d}(\alpha,\beta)$$

$$\mathbf{d}(\alpha,\delta) + \mathbf{d}(\alpha,\beta) = 9 > \mathbf{d}(\beta,\delta)$$

Similarly, the minimum distance of a linear block code, '**C**' may be mathematically represented as below:

$$\mathbf{d}_{\min} = \mathrm{Min}\{\mathbf{d}(\alpha,\beta): \alpha,\beta \in C,\ \alpha \neq \beta\} \quad \ldots\ldots\ldots\ldots \quad (4.20)$$

$$= \mathrm{Min}\{\omega(\alpha \oplus\beta): \alpha,\beta \in C,\ \alpha \neq \beta\}$$

$$= \mathrm{Min}\{\omega(v),\ v \in C,\ v \neq 0\} \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (4.21)$$

That is $d_{min} \trianglelefteq \omega_{min}$. The parameter ω_{min} is called the "**minimum weight**" of the linear code **C**.The minimum distance of a code, $\mathbf{d}_{\mathbf{min}}$, is related to the parity check matrix, **H,** of the code in a fundamental way. Suppose **v** is a code word. Then from Eq. (4.9a) we have:

$$\mathbf{0 = v.H^T}$$

$$\mathbf{= v_1h_1 \oplus v_2h_2 \oplus \ldots. \oplus v_nh_n}$$

Here $\mathbf{h_1, h_2 \ldots h_n}$ represent the columns of the **H** matrix. Let $\mathbf{v_{j1}, v_{j2} \ldots v_{jl}}$ be the '**l**' nonzero components of **v** i.e. $\mathbf{v_{j1} = v_{j2} = \ldots. v_{jl}} = 1$. Then it follows that:

$$\mathbf{h_{j1} \oplus h_{j2} \oplus \ldots \oplus h_{jl} = O^T} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots. \qquad (4.22)$$

That is "**if v is a code vector of Hamming weight 'l', then there exist 'l' columns of H such that the vector sum of these columns is equal to the zero vector**". Suppose we form a binary **n**-tuple of weight '**l**', viz. $\mathbf{x = (x_1, x_2 \ldots x_n)}$ whose nonzero components are $\mathbf{x_{j1}, x_{j2} \ldots x_{jl}}$. Consider the product:

$$\mathbf{x.H^T = x_1h_1 \oplus x_2h_2 \oplus\ldots. \oplus x_nh_n = x_{j1}h_{j1} \oplus x_{j2}h_{j2} \oplus \ldots. \oplus x_{jl}h_{jl} = h_{j1} \oplus h_{j2} \oplus \ldots \oplus h_{jl}}$$

If Eq. (4.22) holds, it follows $\mathbf{x.H^T = O}$ and hence **x** is a code vector. Therefore, we conclude that **"if there are 'l' columns of H matrix whose vector sum is the zero vector then there exists a code vector of Hamming weight 'l' ".**
From the above discussions, it follows that:

i) If no **(d-1)** or fewer columns of **H** add to $\mathbf{O^T}$, the all zero column vector, the code has a minimum weight of at least'**d**'.

ii) The minimum weight (or the minimum distance) of a linear block code **C**, is the smallest number of columns of **H** that sum to the all zero column vector.

For the H matrix of Example 6.3, i.e. $\mathbf{H} = \begin{bmatrix} 0\,1\,1\,1\,0\,0 \\ 1\,0\,1\,0\,1\,0 \\ 1\,1\,0\,0\,0\,1 \end{bmatrix}$, notice that all columns of **H** are non zero and distinct. Hence no two or fewer columns sum to zero vector. Hence the minimum weight of the code is at least 3.Further notice that the **1st, 2nd** and **3rd** columns sum to $\mathbf{O^T}$. Thus the minimum weight of the code is **3**. We see that the minimum weight of the code is indeed **3** from the table of Example 4.1.

4.8.1 Error Detecting and Error Correcting Capabilities:

The minimum distance, $\mathbf{d_{min}}$, of a linear block code is an important parameter of the code. To be more specific, it is the one that determines the error correcting capability of the code. To understand this we shall consider a simple example. Suppose we consider **3**-bit code words plotted at the vertices of the cube as shown in Fig.4.10.

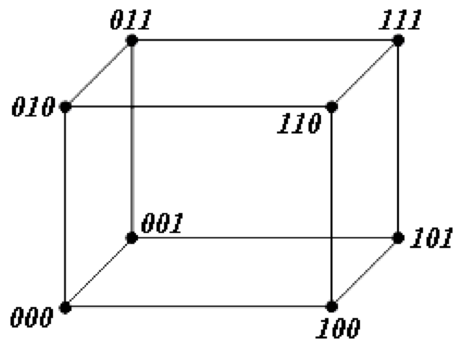


Fig 4.10 The distance concept

Clearly, if the code words used are **{000, 101, 110, 011}**, the Hamming distance between the words is **2**. Notice that any error in the received words locates them on the vertices of the cube which are not code words and may be recognized as single errors. The code word pairs with Hamming distance **= 3** are: **(000, 111), (100, 011), (101, 010)** and **(001, 110)**. If a code word **(000)** is received as **(100, 010, 001)**, observe that these are nearer to **(000)** than to **(111)**. Hence the decision is made that the transmitted word is **(000)**.

Suppose an **(n, k)** linear block code is required to detect and correct all error patterns (over a **BSC**), whose Hamming weight, $\boldsymbol{\omega \leq t}$. That is, if we transmit a code vector $\boldsymbol{\alpha}$ and the received vector is $\boldsymbol{\beta = \alpha \oplus e}$, we want the decoder out put to be $\hat{\alpha} = \boldsymbol{\alpha}$ subject to the condition $\boldsymbol{\omega(e) \leq t}$.

Further, assume that $\mathbf{2^k}$ code vectors are transmitted with equal probability. The best decision for the decoder then is to pick the code vector nearest to the received vector $\boldsymbol{\beta}$ for which the Hamming distance is the smallest. i.e., $\mathbf{d}\,(\boldsymbol{\alpha,\beta})$ is minimum. With such a strategy the decoder will be able to detect and correct all error patterns of Hamming weight $\boldsymbol{\omega(e) \leq t}$ provided that the minimum distance of the code is such that:

$$\mathbf{d_{min} \geq (2t + 1)} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \qquad (4.23)$$

$\mathbf{d_{min}}$ is either odd or even. Let '**t**' be a positive integer such that

$$\mathbf{2t + 1 \leq d_{min} \leq 2t + 2} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \qquad (4.24)$$

Suppose $\boldsymbol{\delta}$ be any other code word of the code. Then, the Hamming distances among $\boldsymbol{\alpha,\beta}$ and $\boldsymbol{\delta}$ satisfy the triangular inequality:

$$\mathbf{d(\alpha,\beta) + d(\beta,\delta) \geq d(\alpha,\delta)} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \qquad (4.25)$$

Suppose an error pattern of '**t′** ' errors occurs during transmission of **α**. Then the received vector **β** differs from **α** in '**t′** ' places and hence **d(α,β) = t′**. Since **α** and **δ** are code vectors, it follows from Eq. (6.24).

$$\mathbf{d(\alpha,\delta) \geq d_{min} \geq 2t + 1} \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (4.26)$$

Combining Eq. (4.25) and (4.26) and with the fact that **d(α,β) = t′**, it follows that:

$$\mathbf{d\,(\delta,\ \beta\,) \geq 2t + 1 - t'} \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (4.27)$$

Hence if **t′≤ t**, then: **d (δ,β) > t** ……………… (4.28)

Eq 4.28 says that if an error pattern of '**t**' or fewer errors occurs, the received vector **β** is closer (in Hamming distance) to the transmitted code vector **α** than to any other code vector **δ** of the code. For a BSC, this means **P (β|α)** > **P (β|δ)** for **α ≠ δ**. Thus based on the maximum likelihood decoding scheme, **β** is decoded as **α** , which indeed is the actual transmitted code word and this results in the correct decoding and thus the errors are corrected.

On the contrary, the code is not capable of correcting error patterns of weight **l>t**. To show this we proceed as below:

Suppose **d (α,δ) = $\mathbf{d_{min}}$**, and let **$\mathbf{e_1}$** and **$\mathbf{e_2}$** be two error patterns such that:

i) $\mathbf{e_1 \oplus e_2 = \alpha \oplus \delta}$

ii) **$\mathbf{e_1}$** and **$\mathbf{e_2}$** do not have nonzero components in common places. Clearly,

$$\mathbf{\omega(e_1) + \omega(e_2) = \omega(\alpha \oplus \delta) = d(\alpha\,,\delta) = d_{min}} \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (4.29)$$

Suppose, **α** is the transmitted code vector and is corrupted by the error pattern **$\mathbf{e_1}$**. Then the received vector is:

$$\mathbf{\beta = \alpha \oplus e_1} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (4.30)$$

and $$\mathbf{d\,(\alpha,\beta) = \omega(\alpha \oplus \beta\,) = \omega(e_1)} \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (4.31)$$

$$\mathbf{d\,(\delta,\beta) = \omega(\delta \oplus \beta)}$$

$$\mathbf{= \omega(\delta \oplus \alpha \oplus e_1) = \omega(e_2)} \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (4.32)$$

If the error pattern $\mathbf{e_1}$ contains more than '**t**' errors, i.e. $\omega(\mathbf{e_1}) > \mathbf{t}$, and since $\mathbf{2t + 1 \leq d_{min} \leq 2t + 2}$, it follows

$$\omega(\mathbf{e_2}) \leq \mathbf{t - 1} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \qquad (4.33)$$

$$\therefore \mathbf{d}(\alpha,\beta) \geq \mathbf{d}(\delta,\beta) \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots. \qquad (4.34)$$

This inequality says that there exists an error pattern of **l > t** errors which results in a received vector closer to an incorrect code vector i.e. based on the maximum likelihood decoding scheme decoding error will be committed.

To make the point clear, we shall give yet another illustration. The code vectors and the received vectors may be represented as points in an **n**- dimensional space. Suppose we construct two spheres, each of equal radii,'**t**' around the points that represent the code vectors α and δ. Further let these two spheres be mutually exclusive or disjoint as shown in Fig.4.11 (a).

For this condition to be satisfied, we then require $\mathbf{d}(\alpha,\delta) \geq \mathbf{2t + 1}$.In such a case if $\mathbf{d}(\alpha,\beta) \leq \mathbf{t}$, it is clear that the decoder will pick α as the transmitted vector.

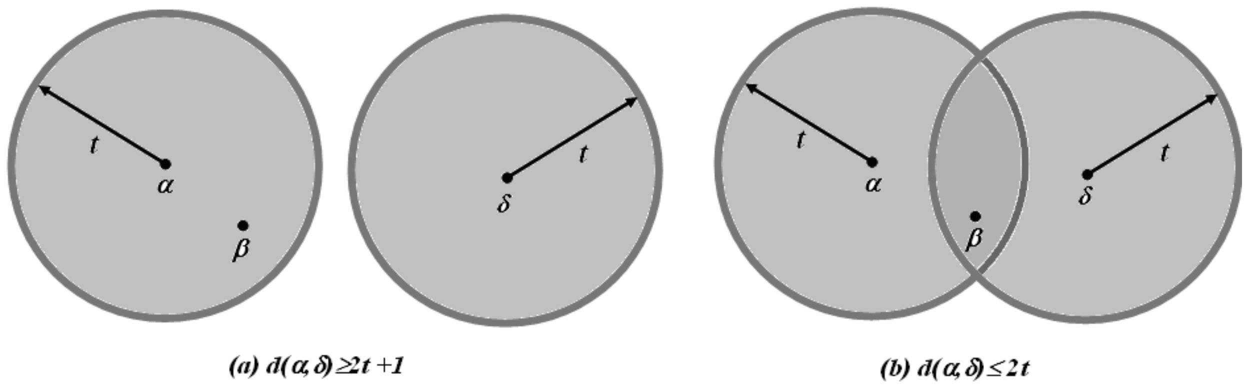


Fig. 4.11(a)

On the other hand, if $\mathbf{d}(\alpha,\delta) \leq \mathbf{2t}$, the two spheres around α and δ intersect and if 'β' is located as in Fig. 4.11(b), and α is the transmitted code vector it follows that even if $\mathbf{d}(\alpha,\beta) \leq \mathbf{t}$, yet β is as close to δ as it is toα**.** The decoder can now pick δ as the transmitted vector which is wrong. Thus it is imminent that "an **(n**, **k)** linear block code has the power to correct all error patterns of weight'**t**' or less if and only if $\mathbf{d}(\alpha,\delta) \geq \mathbf{2t + 1}$ for all α andδ". However, since the smallest distance between any pair of code words is the minimum distance of the code, $\mathbf{d_{min}}$, 'guarantees' correcting all the error patterns of

$$\mathbf{t} \leq \left\{\frac{1}{2}(d_{min} - 1)\right\} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots. \qquad (4.35)$$

where $\left\{\frac{1}{2}(d_{min}-1)\right\}$ denotes the largest integer no greater than the number $\left\{\frac{1}{2}(d_{min}-1)\right\}$. The parameter'**t**' = $\left\{\frac{1}{2}(d_{min}-1)\right\}$ is called the "**random-error-correcting capability**" of the code and the code is referred to as a "**t-error correcting code**". The (**6, 3**) code of Example 4.1 has a minimum distance of **3** and from Eq. (6.35) it follows **t = 1**, which means it is a '**Single Error Correcting'** (**SEC**) code. It is capable of correcting any error pattern of single errors over a block of six digits.

For an **(n, k)** linear code, observe that, there are 2^{n-k} syndromes including the all zero syndrome. Each syndrome corresponds to a specific error pattern. If '**j**' is the number of error locations in the **n**-dimensional error pattern **e**, we find in general, there are $\binom{n}{j} = nC_j$ multiple error patterns. It then follows that the total number of all possible error patterns = $\sum_{j=0}^{t}\binom{n}{j}$, where'**t**' is the maximum number of error locations in **e**. Thus we arrive at an important conclusion. "**If an (n, k) linear block code is to be capable of correcting up to't' errors, the total number of syndromes shall not be less than the total number of all possible error patterns**", i.e.

$$2^{n-k} \geq \sum_{j=0}^{t}\binom{n}{j} \qquad \text{..........................} \qquad (4.36)$$

Eq (6.36) is usually referred to as the "**Hamming bound**". A binary code for which the Hamming Bound turns out to be equality is called a "**Perfect code**".

.9 Standard Array and Syndrome Decoding:

The decoding strategy we are going to discuss is based on an important property of the syndrome.

Suppose $\mathbf{v_j}$, **j = 1, 2… 2^k,** be the 2^k distinct code vectors of an **(n, k)** linear block code. Correspondingly let, for any error pattern **e**, the 2^k distinct error vectors, $\mathbf{e_j}$, be defined by

$$\mathbf{e_j = e \oplus v_j}, \ \mathbf{j = 1, 2\ldots 2^k} \qquad \text{..........................} \qquad (4.37)$$

The set of vectors $\{\mathbf{e_j}, \mathbf{j = 1, 2 \ldots 2^k}\}$ so defined is called the "**co- set**" of the code. That is, a '**co-set**' contains exactly 2^k elements that differ at most by a code vector. It then fallows that there are 2^{n-k} **co- sets** for an **(n, k)** linear block code. Post multiplying Eq (4.37) by $\mathbf{H^T}$, we find

$$\mathbf{e_j H^T = eH^T \oplus v_j H^T}$$

$$\mathbf{= e H^T} \qquad \text{..} \qquad (4.38)$$

Notice that the RHS of Eq (4.38) is independent of the index **j**, as for any code word the term $\mathbf{v_j H^T = 0}$. From Eq (4.38) it is clear that "**all error patterns that differ at most by a code word have the same syndrome**". That is, each co-set is characterized by a unique syndrome.

Since the received vector **r** may be any of the $\mathbf{2^n}$ **n**-tuples, no matter what the transmitted code word was, observe that we can use Eq (4.38) to partition the received code words into $\mathbf{2^k}$ disjoint sets and try to identify the received vector. This will be done by preparing what is called the "**standard array**". The steps involved are as below:

Step1: Place the $\mathbf{2^k}$ code vectors of the code in a row, with the all zero vector $\mathbf{v_1} = \mathbf{(0, 0, 0\ldots\ 0)} = \mathbf{O}$ as the first (left most) element.

Step 2: From among the remaining $\mathbf{(2^n - 2^k) - n}$ – tuples, $\mathbf{e_2}$ is chosen and placed below the all-zero vector, $\mathbf{v_1}$. The second row can now be formed by placing $\mathbf{(e_2 \oplus v_j)}$, $\mathbf{j = 2, 3\ldots\ 2^k}$ under $\mathbf{v_j}$

Step 3: Now take an un-used **n**-tuple $\mathbf{e_3}$ and complete the **3rd** row as in **step 2**.

Step 4: continue the process until all the n-tuples are used.

The resultant array is shown in Fig. 4.12.

$v_1 = O$	v_2	v_3	……	v_{2^k}
e_2	$v_2 \oplus e_2$	$v_3 \oplus e_2$	……	$v_{2^k} \oplus e_2$
e_3	$v_2 \oplus e_3$	$v_3 \oplus e_3$	…….	$v_{2^k} \oplus e_3$
⋮	⋮	⋮	⋮	⋮
$e_{2^{n-k}}$	$v_2 \oplus e_{2^{n-k}}$	$v_3 \oplus e_{2^{n-k}}$	…….	$v_{2^k} \oplus e_{2^{n-k}}$

Fig 4.12: Standard Array for an (n,k) linear block code

Since all the code vectors, $\mathbf{v_j}$, are all distinct, the vectors in any row of the array are also distinct. For, if two **n**-tuples in the **l**-th row are identical, say $\mathbf{e_l \oplus v_j = e_l \oplus v_m}$, $\mathbf{j \neq m}$; we should have $\mathbf{v_j = v_m}$ which is impossible. Thus it follows that **"no two n-tuples in the same row of a slandered array are identical".**

Next, let us consider that an **n**-tuple appears in both **l**-th row and the **m**-th row. Then for some $\mathbf{j_1}$ and $\mathbf{j_2}$ this implies $\mathbf{e_l \oplus v_{j1} = e_m \oplus v_{j2}}$, which then implies $\mathbf{e_l = e_m \oplus (v_{j2} \oplus v_{j1})}$; (remember that $\mathbf{X \oplus X = 0}$ in modulo-**2** arithmetic) or $\mathbf{e_l = e_m \oplus v_{j3}}$ for some $\mathbf{j_3}$. Since by property of linear block codes $\mathbf{v_{j3}}$ is also a code word, this implies, by the construction rules given, that $\mathbf{e_l}$ must appear in the **m**-th row, which is a contradiction of our steps, as the first element of the **m**-th row is $\mathbf{e_m}$ and is an unused vector in the previous rows. This clearly demonstrates another important property of the array: **"Every n-tuple appearance in one and only one row".**

From the above discussions it is clear that there are $\mathbf{2^{n-k}}$ disjoint rows or co-sets in the standard array and each row or co-set consists of $\mathbf{2^k}$ distinct entries. The first n-tuple of each co-set, (i.e., the entry in the first column) is called the "**Co-set leader**". Notice that any element of the co-set can be used as a co-set leader and this does not change the element of the co-set - it results simply in a permutation.

Suppose $\mathbf{D_j^T}$ is the $\mathbf{j}^{th}$ column of the standard array. Then it follows

$$\mathbf{D_j} = \{\mathbf{v_j}, \mathbf{e_2} \oplus \mathbf{v_j}, \mathbf{e_3} \oplus \mathbf{v_j} \ldots \mathbf{e_{2^{n-k}}} \oplus \mathbf{v_j}\} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots.. \qquad (4.39)$$

where $\mathbf{v_j}$ is a code vector and $\mathbf{e_2}, \mathbf{e_3}, \ldots \mathbf{e_{2^{n-k}}}$ are the co-set leaders.

The $\mathbf{2^k}$ disjoints columns $\mathbf{D_1^T}, \mathbf{D_2^T} \ldots$ $D_{2^k}^{\ T}$ can now be used for decoding of the code. If $\mathbf{v_j}$ is the transmitted code word over a noisy channel, it follows from Eq (5.39) that the received vector r is in $\mathbf{D_j^T}$ if the error pattern caused by the channel is a co-set leader. If this is the case **r** will be decoded correctly as $\mathbf{v_j}$. If not an erroneous decoding will result for, any error pattern $\hat{e}$ which is not a co-set leader must be in some co-set and under some nonzero code vector is, say, in the **i**-th co-set and under $\mathbf{v} \neq \mathbf{0}$. Then it follows

$\hat{e} = \mathbf{e_i} \oplus \mathbf{v_l}$, and the received vector is $\mathbf{r} = \mathbf{v_j} \oplus \hat{e} = \mathbf{v_j} \oplus (\mathbf{e_i} \oplus \mathbf{v_l}) = \mathbf{e_i} \oplus \mathbf{v_m}$

Thus the received vector is in $\mathbf{D_m^T}$ and it will be decoded as $\mathbf{v_m}$ and a decoding error has been committed. Hence it is explicitly clear that **"Correct decoding is possible if and only if the error pattern caused by the channel is a co-set leader"**. Accordingly, the $\mathbf{2^{n-k}}$ co-set leaders (including the all zero vector) are called the "**Correctable error patterns**", and it follows "**Every (n, k) linear block code is capable of correcting $\mathbf{2^{n-k}}$ error patterns**".

So, from the above discussion, it follows that in order to minimize the probability of a decoding error, "**The most likely to occur**" error patterns should be chosen as co-set leaders. For a **BSC** an error pattern of smallest weight is more probable than that of a larger weight. Accordingly, when forming a standard array, error patterns of smallest weight should be chosen as co-set leaders. Then the decoding based on the standard array would be the '**minimum distance decoding**' (the maximum likelihood decoding). This can be demonstrated as below.

Suppose a received vector **r** is found in the $\mathbf{j^{th}}$ column and $\mathbf{l^{th}}$ row of the array. Then **r** will be decoded as $\mathbf{v_j}$. We have

$$\mathbf{d(r, v_j) = \omega(r \oplus v_j) = \omega(e_l \oplus v_j \oplus v_j) = \omega(e_l)}$$

where we have assumed $\mathbf{v_j}$ indeed is the transmitted code word. Let $\mathbf{v_s}$ be any other code word, other than $\mathbf{v_j}$. Then

$$\mathbf{d(r, v_s) = \omega(r \oplus v_s) = \omega(e_l \oplus v_j \oplus v_s)\ \omega(e_l) = \omega(e_l \oplus v_i)}$$

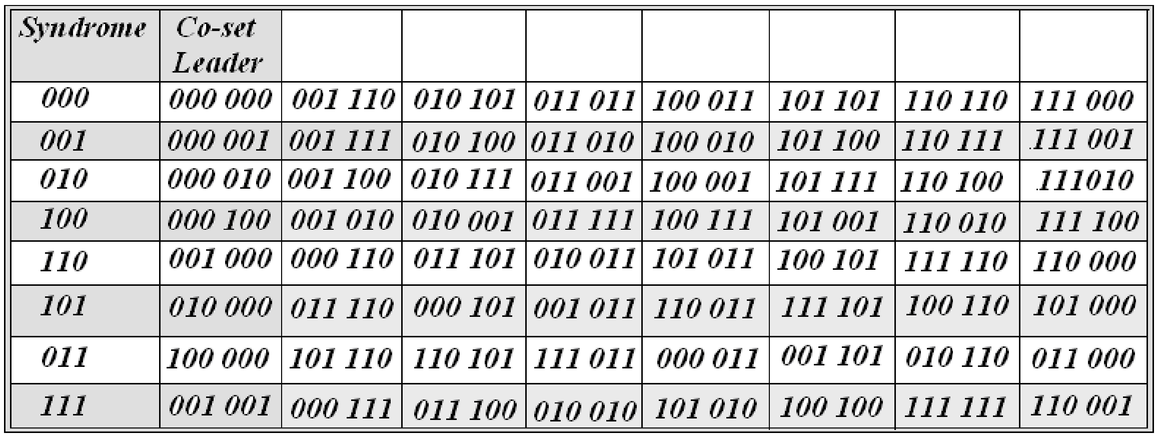
as $\mathbf{v_j}$ and $\mathbf{v_s}$ are code words, $\mathbf{v_i = v_j \oplus v_s}$ is also a code word of the code. Since $\mathbf{e_l}$ and $\mathbf{(e_l \oplus v_i)}$ are in the same co set and, that $\mathbf{e_l}$ has been chosen as the co-set leader and has the smallest weight it follows $\boldsymbol{\omega(e_l) \le \omega(e_l \oplus v_i)}$ and hence $\mathbf{d(r, v_j) \le d(r, v_s)}$. Thus the received vector is decoded into a closet code vector. Hence, if each co-set leader is chosen to have minimum weight in its co-set, the standard array decoding results in the minimum distance decoding or maximum likely hood decoding.

Suppose **"$a_0, a_1, a_2 \ldots, a_n$"** denote the number of co-set leaders with weights **0, 1, 2… n.** This set of numbers is called the "**Weight distribution**" of the co-set leaders. Since a decoding error will occur if and only if the error pattern is not a co-set leader, the probability of a decoding error for a BSC with error probability (transition probability) **p** is given by

$$P(E) = 1 - \sum_{j=0}^{n} a_j p^j (1-p)^{n-j} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \qquad (4.40)$$

Example 4.8:

For the (6, 3) linear block code of Example 4.1 the standard array, along with the syndrome table, is as below:



Syndrome	*Co-set Leader*							
000	000 000	001 110	010 101	011 011	100 011	101 101	110 110	111 000
001	000 001	001 111	010 100	011 010	100 010	101 100	110 111	111 001
010	000 010	001 100	010 111	011 001	100 001	101 111	110 100	111010
100	000 100	001 010	010 001	011 111	100 111	101 001	110 010	111 100
110	001 000	000 110	011 101	010 011	101 011	100 101	111 110	110 000
101	010 000	011 110	000 101	001 011	110 011	111 101	100 110	101 000
011	100 000	101 110	110 101	111 011	000 011	001 101	010 110	011 000
111	001 001	000 111	011 100	010 010	101 010	100 100	111 111	110 001

The weight distribution of the co-set leaders in the array shown are $\mathbf{a_0 = 1, a_1 = 6, a_2 = 1, a_3 = a_4 = a_5 = a_6 = 0}$.From Eq (5.40) it then follows:

$$\mathbf{P(E) = 1 - [(1-p)^6 + 6p(1-p)^5 + p^2(1-p)^4]}$$

With $p = 10^{-2}$, we have $P(E) = 1.3643879 \times 10^{-3}$

A received vector **(010 001)** will be decoded as **(010101)** and a received vector **(100 110)** will be decoded as **(110 110).**

Notice that an **(n, k)** linear code is capable of detecting $\mathbf{(2^n - 2^k)}$ error patterns while it is capable of correcting only $\mathbf{2^{n-k}}$ error patterns. Further, as **n** becomes large $\mathbf{2^{n-k}/(2^n - 2^k)}$ becomes smaller and hence the probability of a decoding error will be much higher than the probability of an undetected error.

Let us turn our attention to Eq (5.35) and arrive at an interpretation. Let $\mathbf{x_1}$and $\mathbf{x_2}$ be two **n**-tuples of weights'**t**' or less. Then it follows

$$\omega (\mathbf{x_1} \oplus \mathbf{x_2}) \leq \omega(\mathbf{x_1}) + \omega(\mathbf{x_2}) \leq 2t \leq d_{m-n}$$

Suppose $\mathbf{x_1}$ and $\mathbf{x_2}$ are in the same co-set then it follows that $(\mathbf{x_1} \oplus \mathbf{x_2})$ must be a nonzero code vector of the code. This is impossible because the weight of $(\mathbf{x_1} \oplus \mathbf{x_2})$ is less than the minimum weight of the code. Therefore, **"No two n-tuples, whose weights are less than or equal to't', can be in the same co-set of the code and all such n-tuples can be used as co-set leaders".**

Further, if **v** is a minimum weight code vector, i.e. $\omega(\mathbf{v}) = d_{min}$ and if the **n**-tuples, $\mathbf{x_1}$ and $\mathbf{x_2}$ satisfy the following two conditions:

i) $\mathbf{x_1} \oplus \mathbf{x_2} = \mathbf{v}$

ii) $\mathbf{x_1}$ and $\mathbf{x_2}$ do not have nonzero components in common places

It follows from the definition, $\mathbf{x_1}$ and $\mathbf{x_2}$ must be in the same co-set and

$$\omega (\mathbf{x_1}) + \omega(\mathbf{x_2}) = \omega(\mathbf{v}) = d_{min}$$

Suppose we choose $\mathbf{x_2}$ such that $\omega(\mathbf{x_2}) = t + 1$. Since $2t+1 \leq d_{min} \leq 2t+2$, we have $\omega (\mathbf{x_1}) = t$ or $(t+1)$. If $\mathbf{x_1}$ is used as a co-set leader then $\mathbf{x_2}$ cannot be a co-set leader.

The above discussions may be summarized by saying **"For an (n , k) linear block code with minimum distance d_{min}, all n-tuples of weight $t \leq \left[\frac{1}{2}(d_{min} - 1)\right]$ can be used as co-set leaders of a standard array. Further, if all n-tuples of weight ≤ t are used as co-set leaders, there is at least one n-tuple of weight (t + 1) that cannot be used as a co-set leader".**

These discussions once again re-confirm the fact that an **(n, k)** linear code is capable of correcting error patterns of $\left[\frac{1}{2}(d_{min} - 1)\right]$ or fewer errors but is incapable of correcting all the error patterns of weight **(t + 1).**

We have seen in Eq. (4.38) that each co-set is characterized by a unique syndrome or there is a one- one correspondence between a co- set leader (a correctable error pattern) and a syndrome. These relationships, then, can be used in preparing a decoding table that is made up of 2^{n-k} co-set leaders and their corresponding syndromes. This table is either stored or wired in the receiver. The following are the steps in decoding:

Step 1: Compute the syndrome $\mathbf{s} = \mathbf{r}.\mathbf{H}^T$

Step 2: Locate the co-set leader $\mathbf{e_j}$ whose syndrome is $\mathbf{s}$. Then $\mathbf{e_j}$ is assumed to be the error pattern caused by the channel.

Step 3: Decode the received vector $\mathbf{r}$ into the code vector $\mathbf{v = r \oplus e_j}$

This decoding scheme is called the "**Syndrome decoding**" or the "**Table look up decoding**". Observe that this decoding scheme is applicable to any linear (**n, k**) code, i.e., it need not necessarily be a systematic code. However, as **(n-k)** becomes large the implementation becomes difficult and impractical as either a large storage or a complicated logic circuitry will be required.

For implementation of the decoding scheme, one may regard the decoding table as the truth table of **n**-switching functions:

$$\mathbf{e_1 = f_1 (s_1, s_2 \ldots s_{n-k}); \; e_2 = f_2 (s_1, s_2 \ldots s_{n-k}); \; \ldots \; e_n = f_n (s_1, s_2 \ldots s_{n-k})}$$

where $\mathbf{s_1, s_2 \ldots s_{n-k}}$ are the syndrome digits and are regarded as the switching variables and $\mathbf{e_1, e_2 \ldots e_n}$ are the estimated error digits. The stages can be released by using suitable combinatorial logic circuits as indicated in Fig 4.13.

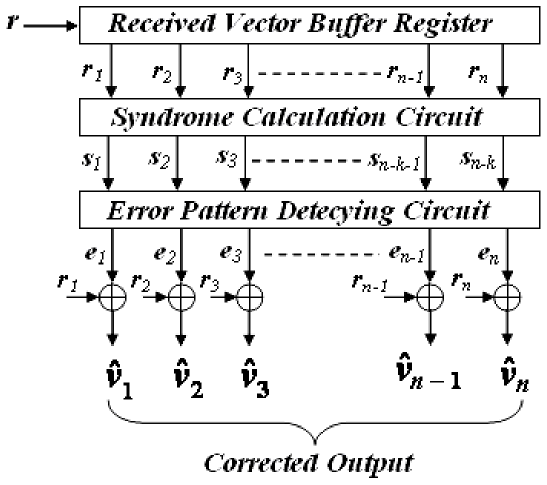


Fig. 4.13 General Decoding scheme for an (n,k) linear block code

Example 4.9:

From the standard array for the (**6, 3**) linear block code of Example 4.8, the following truth table can be constructed.

s_1	s_2	s_3	e_1	e_2	e_3	e_4	e_5	e_6
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1
0	1	0	0	0	0	0	1	0
1	0	0	0	0	0	1	0	0
1	1	0	0	0	1	0	0	0
1	0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0	0
1	1	1	0	0	1	0	0	1

Truth table for the Error patterns from the Standard Array of Example 6.8

The two shaded portions of the truth table are to be observed carefully. The top shaded one corresponds to the all-zero error pattern and the bottom one corresponds to a double error patter which cannot be corrected by this code. From the table we can now write expressions for the correctable single error patterns as below:

$$e_1 = \bar{s}_1 . s_2 s_3 \quad e_2 = s_1 \bar{s}_2 s_3 \quad e_3 = s_1 s_2 \bar{s}_3$$
$$e_4 = s_1 \bar{s}_2 \bar{s}_3 \quad e_5 = \bar{s}_1 s_2 \bar{s}_3 \quad e_6 = \bar{s}_1 \bar{s}_2 s_3$$

The implementation of the decoder is shown in Fig.4.14.

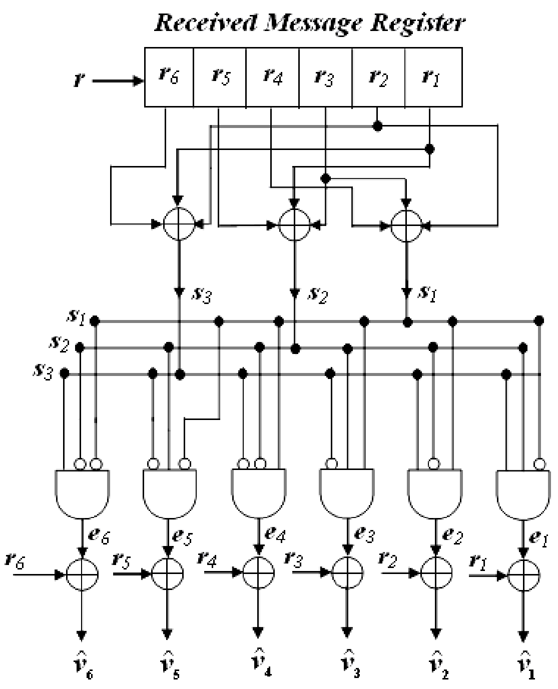


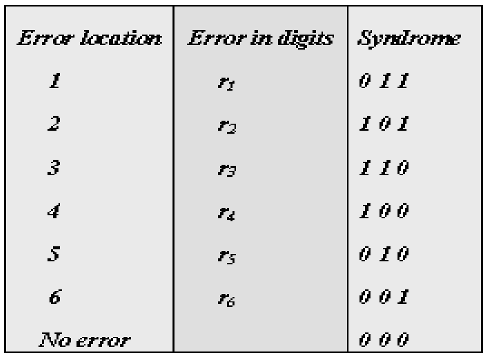
Fig 4.14: Decoding circuit for (6,3) code

Comments:

1) Notice that for all correctable single error patterns the syndrome will be identical to a column of the **H** matrix and indicates that the received vector is in error corresponding to that column position.

For Example, if the received vector is (**010001**), then the syndrome is (**100**). This is identical withthe**4th** column of the **H**- matrix and hence the **4th** – position of the received vector is in error. Hence the corrected vector is **010101**. Similarly, for a received vector (**100110**), the syndrome is **101** and this is identical with the second column of the **H**-matrix. Thus the second position of the received vector is in error and the corrected vector is (**110110**).

2) A table can be prepared relating the error locations and the syndrome. By suitable combinatorial circuits data recovery can be achieved. For the (**6, 3**) systematic linear code we have the following table for $\mathbf{r = (r_1\ r_2\ r_3\ r_4\ r_5\ r_6.)}$.



Error location	*Error in digits*	*Syndrome*
1	r_1	*0 1 1*
2	r_2	*1 0 1*
3	r_3	*1 1 0*
4	r_4	*1 0 0*
5	r_5	*0 1 0*
6	r_6	*0 0 1*
No error		*0 0 0*

Notice that for the systematic encoding considered by us $\mathbf{(r_1\ r_2\ r_3)}$ corresponds to the data digits and $\mathbf{(r_4\ r_5\ r_6)}$ are the parity digits.

Accordingly the correction for the data digits would be

$$\hat{v}_1 = r_1 + (s_2 . s_3),\ \hat{v}_2 = r_2 + (s_1 . s_3),\ \hat{v}_3 = r_3 + (s_1 . s_2)$$

Hence the circuit of Fig 6.14 can be modified to have data recovery by removing only the connections of the outputs $\hat{v}_4, \hat{v}_5$ ***and*** $\hat{v}_6$.

4.10 Hamming Codes:

Hamming code is the first class of linear block codes devised for error correction. The single error correcting (**SEC**) Hamming codes are characterized by the following parameters.

Code length: $\mathbf{n = (2^m - 1)}$

Number of Information symbols: $\mathbf{k = (2^m - m - 1)}$

Number of parity check symbols :$\mathbf{(n - k) = m}$

Error correcting capability: **t = 1, (d_{min}= 3)**

The parity check matrix **H** of this code consists of all the non-zero **m**-tuples as its columns. In systematic form, the columns of **H** are arranged as follows

H = [Q ⋮ I_m]

Where **I_m** is an identity (unit) matrix of order **m × m** and **Q** matrix consists of

(2^m-m-1) columns which are the **m**-tuples of weight **2** or more. As an illustration for **k=4** we have from **k = 2^m – m – 1.**

m=1 k=0, m=2 k=1, m=3 k=4

Thus we require **3** parity check symbols and the length of the code **2^3 – 1 = 7**. This results in the **(7, 4)** Hamming code.

The parity check matrix for the (**7, 4**) linear systematic Hamming code is then

$$H = \left[\begin{array}{cccc:ccc} 1&1&1&0&1&0&0 \\ 1&1&0&1&0&1&0 \\ 1&0&1&1&0&0&1 \end{array}\right]$$

The generator matrix of the code can be written in the form

$$G = \left[I_{2^m - m - 1} \,\vdots\, Q^T \right]$$

And for the **(7, 4)** systematic code it follows:

$$G = \left[\begin{array}{cccc:ccc} 1&0&0&0&1&1&1 \\ 0&1&0&0&1&1&0 \\ 0&0&1&0&1&0&1 \\ 0&0&0&1&0&1&1 \end{array}\right]$$

A non systematic Hamming code can be constructed by placing the parity check bits at **$2^{l,}$ l=0, 1, 2…**locations. It was the conventional method of construction in switching and computer applications (Refer, for example ‘Switching circuits and applications -Marcus).One simple procedure for construction of such code is as follows:

Step 1: Write the BCD of length **(n – k)** for decimals from **1 to n.**

Step 2: Arrange the sequences in the reverse order in a matrix form.

Step 3: Transpose of the matrix obtained in step 2 gives the parity check matrix **H** for the code.

The code words are in the form

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
$\mathbf{p_1}$	$\mathbf{p_2}$	$\mathbf{m_1}$	$\mathbf{p_3}$	$\mathbf{m_2}$	$\mathbf{m_3}$	$\mathbf{m_4}$	$\mathbf{p_4}$	$\mathbf{m_5}$	$\mathbf{m_6}$	$\mathbf{m_7}$	$\mathbf{m_8}$	$\mathbf{m_9}$	$\mathbf{m_{10}}$	$\mathbf{m_{11}}$	$\mathbf{p_5}$	$\mathbf{m_{12}}$

Where $\mathbf{p_1, p_2, p_3}$**…**are the parity digits and $\mathbf{m_1, m_2, m_3}$**…**are the message digits. For example, let us consider the non systematic **(7, 4)** Hamming code.

Step1:

Numerals	*BCD of length (n - k) = 3*
1	*001*
2	*010*
3	*011*
4	*100*
5	*101*
6	*110*
7	*111*

Step2:

$$H^T = \begin{bmatrix} 1\,0\,0 \\ 0\,1\,0 \\ 1\,1\,0 \\ 0\,0\,1 \\ 1\,0\,1 \\ 0\,1\,1 \\ 1\,1\,1 \end{bmatrix}$$

Step3:

$$H = \begin{bmatrix} 1\,0\,1\,0\,1\,0\,1 \\ 0\,1\,1\,0\,0\,1\,1 \\ 0\,0\,0\,1\,1\,1\,1 \end{bmatrix}$$

Notice that the parity check bits, from he above **H** matrix apply to positions.

$\mathbf{p_1}$ **= 1, 3, 5, 7, 9, 11, 13, 15…**

$\mathbf{p_2}$ **= 2, 3, 6, 7, 10, 11, 14, 15 …**

$\mathbf{p_3}$ **= 4, 5, 6, 7, 12, 13, 14, 15…**

$\mathbf{p_4}$ **= 8, 9, 10, 11, 12, 13, 14, 15** and so on

Accordingly, the check bits can be represented as linear combinations of the message bits. For the **(7, 4)** code under consideration we have

$\mathbf{p_1 = m_1 + m_2 + m_4}$

$\mathbf{p_2 = m_1 + m_3 + m_4}$

$\mathbf{p_3 = m_2 + m_3 + m_4}$

Accordingly, the generator matrix can written as

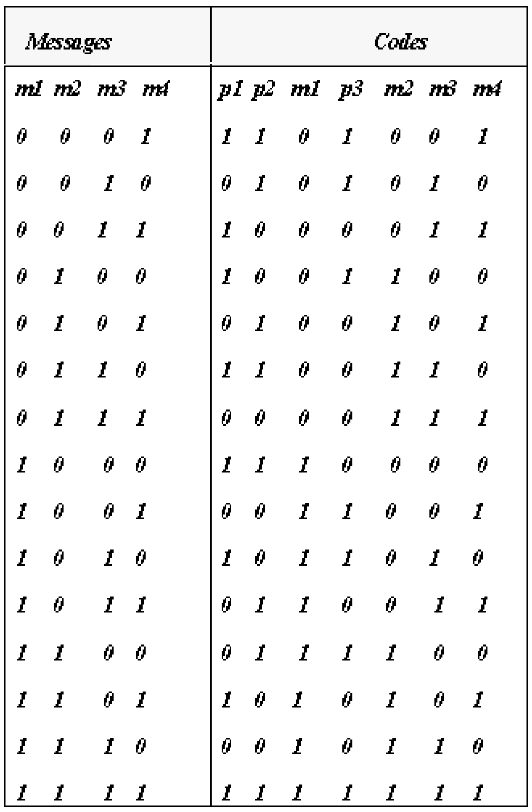
$$G = \begin{array}{c} \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \\ \begin{matrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{matrix} \end{array}$$

Notice that the message bits are located at the positions other than **2^l, l = 0, 1, 2, 3….** locations. i.e., they are located in the positions of **3, 5, 7, 9, 11, 13, 15, 17, 18…..** The **k**- columns of the identity matrix $\mathbf{I_k}$ are distributed successively to these locations. The **Q** sub-matrix in the **H** matrix can be identified to contain those columns which have weights more than one. The transpose of this matrix then gives the columns to be filled, in succession, in the **G**- matrix. For the Example of the **(7, 4)** linear code considered, the **Q-** sub-matrix is:

$$\mathbf{Q} = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \text{, and hence } \mathbf{Q^T} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

The first two columns of this matrix then are the first two columns of the **G**: matrix and the third column is the Forth column of the **G** matrix. Table below gives the codes generated by this method.

Table: Non systematic (7, 4) Hamming code.



Messages				*Codes*						
m1	*m2*	*m3*	*m4*	*p1*	*p2*	*m1*	*p3*	*m2*	*m3*	*m4*
0	0	0	1	1	1	0	1	0	0	1
0	0	1	0	0	1	0	1	0	1	0
0	0	1	1	1	0	0	0	0	1	1
0	1	0	0	1	0	0	1	1	0	0
0	1	0	1	0	1	0	0	1	0	1
0	1	1	0	1	1	0	0	1	1	0
0	1	1	1	0	0	0	0	1	1	1
1	0	0	0	1	1	1	0	0	0	0
1	0	0	1	0	0	1	1	0	0	1
1	0	1	0	1	0	1	1	0	1	0
1	0	1	1	0	1	1	0	0	1	1
1	1	0	0	0	1	1	1	1	0	0
1	1	0	1	1	0	1	0	1	0	1
1	1	1	0	0	0	1	0	1	1	0
1	1	1	1	1	1	1	1	1	1	1

Observe that the procedure outlined for the code construction starts from selecting the **H** matrix which is unique and hence the codes are also unique. We shall consider the correctable error patterns and the corresponding syndromes listed in the table below.

Table: Error patterns and syndromes for the (7, 4) linear non-systematic code

Error Pattern						**Syndrome**		
e_1	e_2	e_3	e_4	e_5	e_6	s_1	s_2	s_3
1	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
0	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
0	**0**	**1**	**0**	**0**	**0**	**1**	**1**	**0**
0	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**
0	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**1**
0	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**1**
0	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**

If the syndrome is read from right to left i.e. if the sequence is arranged as '**s_3 s_2 s_1'** it is interesting to observe that the decimal equivalent of this binary sequence corresponds to the error location. Thus if the code vector **1 0 1 1 0 1 0** is received as **1 0 1 0 0 1 0**, the corresponding syndrome is '**0 0 1**', which is exactly the same as the **4th** column of the **H**-matrix and also the sequence **100** corresponds to decimal **4.**

It can be verified that **(7, 4), (15, 11), (31, 26), (63, 57)** are all single error correcting Hamming codes and are regarded quite useful.

An important property of the Hamming codes is that they satisfy the condition of Eq. (4.36) with equality sign, assuming that **t=1**.This means that Hamming codes are "**single error correcting binary perfect codes**". This can also be verified from Eq. (4.35)

We may delete any '**l**'columns from the parity check matrix **H** of the Hamming code resulting in the reduction of the dimension of **H** matrix to $\mathbf{m \times (2^m - l - 1)}$.Using this new matrix as the parity check matrix we obtain a "**shortened**" Hamming code with the following parameters.

Code length:	$\mathbf{n = 2^m - l - 1}$
Number of Information symbols:	$\mathbf{k = 2^m - m - l - 1}$
Number of parity check symbols:	$\mathbf{n - k = m}$
Minimum distance:	$\mathbf{d_{min} \geq 3}$

Notice that if the deletion of the columns of the H matrix is proper, we may obtain a Hamming code with $\mathbf{d_{min} = 4}$.For example if we delete from the sub-matrix **Q** all the columns of even weight, we obtain an $\mathbf{m \times 2^{m-1}}$ matrix

$$\overline{H} = [\overline{Q} : I_m]$$

Where $\overline{Q}$ contains $\mathbf{(2^{m-1} - m)}$ columns of odd weight. Clearly no three columns add to zero as all columns have odd weight .However, for a column in $\overline{Q}$, there exist three columns in $\mathbf{I_m}$ such that four columns add to zero .Thus the shortened Hamming codes with $\overline{H}$ as the parity check matrix has minimum distance exactly **4**. The distance – **4** shortened Hamming codes can be used for correcting all single error patterns while simultaneously detecting all double error patterns. Notice that when single errors occur the syndromes contain odd number of one's and for double errors it contains even number of ones. Accordingly the decoding can be accomplished in the following manner.

(1) If **s = 0**, no error occurred.

(2) If **s** contains odd number of ones, single error has occurred .The single error pattern pertaining to this syndrome is added to the received code vector for error correction.

(3) If **s** contains even number of one's an uncorrectable error pattern has been detected.

Alternatively the **SEC** Hamming codes may be made to detect double errors by adding an extra parity check in its **(n+1)** $^{\mathbf{Th}}$ position. Thus (**8, 4**), (**6, 11**) etc. codes have $\mathbf{d_{min}} = \mathbf{4}$ and correct single errors with detection of double errors.

RECOMMENDATION QUESTIONS

1. Consider a (**7, 4**) linear code whose generator matrix is

$$G = \begin{bmatrix} 1\,0\,0\,0 & 1\,0\,1 \\ 0\,1\,0\,0 & 1\,1\,1 \\ 0\,0\,1\,0 & 1\,1\,0 \\ 0\,0\,0\,1 & 0\,1\,1 \end{bmatrix}$$

a) Find all code vectors of this code b) Find the parity check matrix for this code
c) Find the minimum weight of this code

2. For linear (**n, k**) block code, **C**, prove that $\mathbf{CH^T = O}$, where **H** is the parity check matrix.

3. The parity check bits of a (**8,4**) block code are generated by

$$\mathbf{c_5 = d_1 + d_2 + d_4}$$
$$\mathbf{c_6 = d_1 + d_2 + d_3}$$
$$\mathbf{c_7 = d_1 + d_3 + d_4}$$
$$\mathbf{c_8 = d_2 + d_3 + d_4}$$

Where $\mathbf{d_1}$, $\mathbf{d_2}$, $\mathbf{d_3}$, and $\mathbf{d_4}$ are the message bits

a) Find the generator matrix and parity check matrix for this code.
b) Find the minimum weight of this code
c) Find the error detecting and error correcting capabilities of this code
d) Show through an example that this code can detect **3** errors/code word.

4. Construct an encoder for the code given in problem 3.

5. Construct a syndrome circuit for the code given in problem 3.

6. Let **H** be the parity check matrix of an **(n, k)** linear code **C** that has both odd and even weight code vectors. Construct new linear codes $\mathbf{C_1}$ and $\mathbf{C_2}$ with the following parity check matrices respectively.

$$\textit{i)}\ H_1 = \left[\begin{array}{c|c} 0 & \\ 0 & \\ 0 & H \\ \vdots & \\ 0 & \\ \hline 1 & 1\ 1 \ldots\ldots\ 1 \end{array}\right] \qquad \textit{ii)}\ H_2 = \left[\begin{array}{c|c} & 0 \\ & 0 \\ H & 0 \\ & \vdots \\ & 0 \\ \hline 1\ 1\ 1 \ldots\ldots & 1 \end{array}\right]$$

(Note that the last row of $\mathbf{H_1}$ $\mathbf{(H_2)}$ consists of all **1**'s)

a) Show that $\mathbf{C_1}$ and $\mathbf{C_2}$, called extensions of **C,** are (**n**+**1**, **k**) linear block codes.

b) Show that every code vector of $\mathbf{C_1}$ $\mathbf{(C_2)}$ has an even weight.
c) Show that the minimum distance of $\mathbf{C_1}$ $\mathbf{(C_2)}$ is (**d**+**1**),), where **d** is the minimum distance of C.
d) Show that $\mathbf{C_1}$ can be obtained from **C** by adding an extra parity check digit denoted by $\mathbf{v_o}$ to the left of each code vector **v** as follows (1) If **v** has odd weight then $\mathbf{v_o}$ = **1** and (2) If **v** has even weight $\mathbf{v_o}$ = **0**. The parity check digit $\mathbf{v_o}$ is called an "overall parity check digit".

7. Let **C** be a linear code with both even and odd-weight code words. Show that the number of even weight code vectors is equal to the number of odd weight code vectors.

8. Since the (**8, 4**) linear code of problem 3 has a minimum distance **4**, it is capable of correcting all the single error patterns and simultaneously detecting any combination of double errors. Construct a decoder for this code. The decoder must be capable of correcting any single error and detecting any double error.

9. The (**8, 4**) linear code of problem 3 is capable of correcting **16** error patterns (The co-set leaders of a standard array). Suppose that this code is used for a **BSC**. Device a decoder for this code based on the table-look up decoding scheme. The decoder is designed to correct the **16** most probable error patterns.

10. Verify whether the dual code of the (**8, 4**) linear code of problem 3 is identical to the code itself. Is the code self dual?

11. Form a parity check matrix for a (**15, 11**) systematic Hamming code. Device a decoder for this code.

12. Let $\mathbf{C_1}$ be an $\mathbf{(n_1, k)}$ linear systematic code with $\mathbf{d_{min} = d_1}$ and generator matrix $\mathbf{G_1 = [P_1, I_k]}$. Let $\mathbf{C_2}$ be an $\mathbf{(n_2, k)}$ linear systematic code with $\mathbf{d_{min} = d_2}$ and Generator matrix $\mathbf{G_2 = [P_2, I_k]}$. Consider an $\mathbf{(n_1 + n_2, k)}$ linear code with the following parity check matrix.

$$H = \left[\begin{array}{c:c} & P_1^T \\ I_{n_1+n_2-k} & I_k \\ & P_2^T \end{array}\right]$$

Show that this code has minimum distance at least $\mathbf{(d_1 + d_2)}$.

13. The "design distance" of an **(n, k)** linear block code is defined as being equal to **(n - k +1)**. Show that the minimum distance of the code can never exceed its design distance.

14. "**Repetition codes**" represent the simplest type of linear block codes. The generator matrix of a (**5, 1**) repetition code is given as G = [**1 1 1 1**| **1**]
a) Write its parity check matrix.
b) Evaluate the syndrome for:
i) All five possible single error patterns. ii) All **10** possible double error patterns.

15. Show that the decoder for a Hamming code fails if there are two or more than two transmission errors in the received sequence.

OUTCOMES

- Detection of the errors.
- Correction of the errors using different techniques.

RESOURCES

- https://en.wikipedia.org/wiki/**Block_code**
- www.inference.phy.cam.ac.uk/mackay/itprnn/1997/l1/node7.html
- web.ntpu.edu.tw/~yshan/intro_lin_**code**.pdf
- users.ece.cmu.edu/~koopman/des_s99/**coding**/
- elearning.vtu.ac.in/P4/EC63/S11.pdf

CHAPTER 2 - BINARY CYCLIC CODES

STRUCTURE

- Generator Polynomial for Cyclic Codes
- Multiplication Circuits
- Dividing Circuits
- Systematic Cyclic Codes
- Generator Matrix for Cyclic Codes
- Syndrome Calculation - Error Detection and Error Correction

OBJECTIVE

- Discuss about cyclic codes and also study about implementation methods using feedback shift registers.
- Study about syndrome calculator for cyclic codes.

INTRODUCTION

"**Binary cyclic codes"** form a sub class of linear block codes. Majority of important linear block codes that are known to-date are either cyclic codes or closely related to cyclic codes. Cyclic codes are attractive for two reasons: First, encoding and syndrome calculations can be easily implemented using simple shift registers with feed back connections. Second, they posses well defined mathematical structure that permits the design of higher-order error correcting codes.

A binary code is said to be "**cyclic**" if it satisfies:

1. Linearity property – sum of two code words is also a code word.
2. Cyclic property – Any lateral shift of a code word is also a code word.

The second property can be easily understood from Fig, 4.1. Instead of writing the code as a row vector, we have represented it along a circle. The direction of traverse may be either clockwise or counter clockwise (right shift or left shift).

For example, if we move in a counter clockwise direction then starting at '**A**' the code word is **110001100** while if we start at **B** it would be **011001100**. Clearly, the two code words are related in that one is obtained from the other by a cyclic shift.

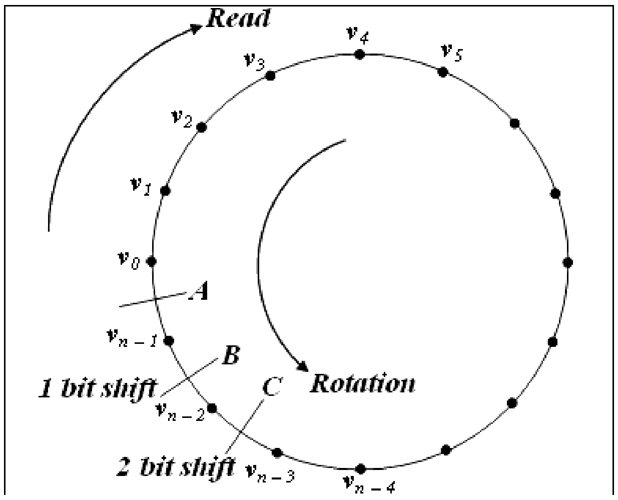


Fig 4.1: Illustrating the cyclic property

If the **n** - tuple, read from '**A**' in the **CW** direction in Fig 4.1,

$$\mathbf{v = (v_o, v_1, v_2, v_3, v_{n-2}, v_{n-1})} \qquad \ldots\ldots\ldots\ldots\ldots\ldots \qquad (4.1)$$

is a code vector, then the code vector, read from **B**, in the **CW** direction, obtained by a one bit cyclic right shift:

$$\mathbf{v^{(1)} = (v_{n-1}\ , v_o, v_1, v_2, \ldots\ v_{n-3}, v_{n-2},)} \qquad \ldots\ldots\ldots\ldots\ldots\ldots \qquad (4.2)$$

is also a code vector. In this way, the **n** - tuples obtained by successive cyclic right shifts:

$$\mathbf{v^{(2)} = (v_{n-2}, v_{n-1}, v_n, v_0, v_1 \ldots v_{n-3})} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \qquad (4.3a)$$

$$\mathbf{v^{(3)} = (v_{n-3}, v_{n-2}, v_{n-1}, v_n, \ldots v_o, v_1, v_{n-4})} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \qquad (4.3b)$$

$$\vdots$$

$$\mathbf{v^{(i)} = (v_{n-i}, v_{n-i+1}, \ldots v_{n-1}, v_o, v_1, \ldots . v_{n-i-1})} \qquad \ldots\ldots\ldots\ldots\ldots \qquad (4.3c)$$

are all code vectors. This property of cyclic codes enables us to treat the elements of each code vector as the co-efficients of a polynomial of degree **(n-1).**

This is the property that is extremely useful in the analysis and implementation of these codes. Thus we write the "code polynomial' **V(X)** for the code in Eq (6.1) as a vector polynomial as:

$$\mathbf{V(X) = v_o + v_1 X + v_2 X^2 + v_3 X^3 + \ldots + v_{i-1} X^{i-1} + \ldots + v_{n-3} X^{n-3} + v_{n-2} X^{n-2} + v_{n-1} X^{n-1}} \qquad \ldots.. \quad (4.4)$$

Notice that the co-efficients of the polynomial are either '**0**' or '**1**' (binary codes), i.e. they belong to **GF (2)** as discussed in sec 5.7.1.

. Each power of **X** in **V(X)** represents a one bit cyclic shift in time.

. Therefore multiplication of **V(X)** by **X** maybe viewed as a cyclic shift or rotation to the right subject to the condition $\mathbf{X^n = 1}$. This condition (i) restores **XV(X)** to the degree **(n-1)** (ii) Implies that right most bit is fed-back at the left.

. This special form of multiplication is called "**Multiplication modulo** "$\mathbf{X^n + 1}$"

. Thus for a single shift, we have

$$\mathbf{XV(X) = v_o X + v_1 X^2 + v_2 X^3 + \ldots\ldots + v_{n-2} X^{n-l} + v_{n-l} X^n}$$

$\mathbf{(+ v_{n-1} + v_{n-1})}$ … (Manipulate **A + A =0** Binary Arithmetic)

$$\mathbf{= v_{n-1} + v_0 X + v_1 X^2 + + v_{n-2} X^{n-1} + v_{n-1}(X^n + 1)}$$

$\mathbf{=V^{(1)}(X)}$ = Remainder obtained by dividing **XV(X)** by $\mathbf{X^n + 1}$
(Remember: **X** mod **Y** means remainder obtained after dividing **X** by **Y**)

Thus it turns out that

$$\mathbf{V^{(1)}(X) = v_{n-1} + v_o X + v_1 X^2 + \ldots + v_{n-2} X^{n-1}} \qquad \ldots\ldots\ldots\ldots\ldots\ldots \qquad (4.5)$$

I is the code polynomial for $\mathbf{v^{(1)}}$. We can continue in this way to arrive at a general format:

$$X^{i}\, V(X) = \underset{\substack{\uparrow \\ \textbf{Remainder}}}{V^{(i)}(X)} + \underset{\substack{\uparrow \\ \textbf{Quotient}}}{q(X)}\,(X^{n} + 1) \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \qquad (4.6)$$

Where

$$V^{(i)}(X) = v_{n-i} + v_{n-i+1}X + v_{n-i+2}X^{2} + \ldots v_{n-1}X^{i} + \ldots v_{0}X^{i} + v_{1}X^{i+1} + \ldots v_{n-i-2}X^{n-2} + v_{n-i-1}X^{n-} \quad \ldots\ldots \quad (4.7)$$

4.1 GENERATOR POLYNOMIAL FOR CYCLIC CODES:

An **(n, k)** cyclic code is specified by the complete set of code polynomials of degree ≤ **(n-1)** and contains a polynomial **g(X)**, of degree **(n-k)** as a factor, called the "**generator polynomial**" of the code. This polynomial is equivalent to the generator matrix **G**, of block codes. Further, it is the only polynomial of minimum degree and is unique. Thus we have an important theorem

Theorem 4.1 "If **g(X)** is a polynomial of degree **(n-k)** and is a factor of **$(X^{n} + 1)$** then **g(X)** generates an **(n, k)** cyclic code in which the code polynomial **V(X)** for a data vector **$u = (u_0, u_1 \ldots\ u_{k-1})$** is generated by

$$V(X) = U(X) \times g(X) \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \qquad (4.8)$$

Where $$U(X) = u_0 + u_1 X + u_2 X^{2} + \ldots + u_{k-1} X^{k-I} \qquad \ldots\ldots\ldots\ldots\ldots\ldots.. \qquad (4.9)$$

is the data polynomial of degree **(k-1)**.

The theorem can be justified by Contradiction: - If there is another polynomial of same degree, then add the two polynomials to get a polynomial of degree < **(n, k)** (use linearity property and binary arithmetic). Not possible because minimum degree is (**n-k**). Hence **g(X)** is unique

Clearly, there are **2^{k}** code polynomials corresponding to **2^{k}** data vectors. The code vectors corresponding to these code polynomials form a linear (**n, k**) code. We have then, from the theorem

$$g(X) = 1 + \sum_{i=1}^{n-k-1} g_i X^{i} + X^{n-k} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \qquad (4.10)$$

As $$g(X) = g_o + g_1 X + g_2 X^{2} + \ldots\ldots + g_{n-k-1} X^{n-k-1} + g_{n-k} X^{n-k} \qquad \ldots\ldots\ldots\ldots \qquad (4.11)$$

is a polynomial of minimum degree, it follows that **$g_0 = g_{n-k} = 1$** always and the remaining co-efficients may be either' **0**' of '**1**'. Performing the multiplication said in Eq (4.8) we have:

$$U(X)\, g(X) = u_o\, g(X) + u_1 X\, g(X) + \ldots + u_{k-1}X^{k-1} g(X) \qquad \ldots\ldots\ldots\ldots. \qquad (4.12)$$

Suppose $\mathbf{u_0=1}$ and $\mathbf{u_1=u_2= \ldots =u_{k-1}=0}$. Then from Eq (4.8) it follows **g(X)** is a code word polynomial of degree **(n-k)**. This is treated as a '**basis code polynomial**' (All rows of the **G** matrix of a block code, being linearly independent, are also valid code vectors and form '**Basis vectors**' of the code). Therefore from cyclic property $\mathbf{X^i\ g(X)}$ is also a code polynomial. Moreover, from the linearity property - a linear combination of code polynomials is also a code polynomial. It follows therefore that any multiple of **g(X)** as shown in Eq (4.12) is a code polynomial. Conversely, any binary polynomial of degree ≤ **(n-1)** is a code polynomial if and only if it is a multiple of **g(X).** The code words generated using Eq (4.8) are in non-systematic form. Non systematic cyclic codes can be generated by simple binary multiplication circuits using shift registers. .

In this book we have described cyclic codes with right shift operation. Left shift version can be obtained by simply re-writing the polynomials. Thus, for left shift operations, the various polynomials take the following form

$$\mathbf{U(X) = u_o X^{k-1} + u_1 X^{k-2} + \ldots\ldots + u_{k-2}X + u_{k-1}} \qquad \ldots\ldots\ldots\ldots\ldots\ldots.. \qquad (4.13a)$$

$$\mathbf{V(X) = v_0\ X^{n-1} + v_1 X^{n-2} + \ldots. + v_{n-2}X + v_{n-1}} \qquad \ldots\ldots\ldots\ldots\ldots\ldots \qquad (4.13b)$$

$$\mathbf{g(X) = g_0 X^{n-k} + g_1 X^{n-k-1} + \ldots..+g_{n-k-1}\ X + g_{n-k}} \qquad \ldots\ldots\ldots\ldots\ldots \qquad (4.13c)$$

$$= \boldsymbol{X_{n-k} + \sum_{i=1}^{n-k} g_i X^{n-k-i} + g_{n-k}} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \qquad (4.13d)$$

Other manipulation and implementation procedures remain unaltered.

4.2 MULTIPLICATION CIRCUITS

Construction of encoders and decoders for linear block codes are usually constructed with combinational logic circuits with mod-2 adders. Multiplication of two polynomials **A(X)** and **B(X)** and the division of one by the other are realized by using sequential logic circuits, mod-2 adders and shift registers. In this section we shall consider multiplication circuits.

As a convention, the higher-order co-efficients of a polynomial are transmitted first. This is the reason for the format of polynomials used in this book.

For the polynomial: $\mathbf{A(X) = a_0 + a_1\ X + a_2\ X^2 + \ldots + a_{n-1}X^{n-1}}$ .... (4.14)

where $\mathbf{a_i}$**'s** are either a ' **0**' or a '**1**', the right most bit in the sequence $\mathbf{(a_0, a_1, a_2 \ldots a_{n-1})}$ is transmitted first in any operation. The product of the two polynomials **A(X)** and **B(X)** yield:

$\mathbf{C(X) = A(X) \times B(X)}$

$$= (a_0 + a_1 X + a_2 X^2 + \ldots \ldots\ldots + a_{n-1}X^{n-1})\,(b_0 + b_1 X + b_2X^2 + \ldots + b_{m-1} X^{m-1})$$
$$= a_0b_0 + (a_1b_0+a_0b_1) X + (a_0b_2 + b_0a_2+a_1b_1) X^2 + \ldots + (a_{n-2}b_{m-1}+ a_{n-1}b_{m-2}) X^{n+m-3} + a_{n-1}b_{m-1}X^{n+m-2}$$

This product may be realized with the circuits of Fig 4.2 (a) or (b), where **A(X)** is the input and the co-efficient of **B(X)** are given as weighting factor connections to the mod - 2 .adders. A '**0**' indicates no connection while a '**1**' indicates a connection. Since higher order co-efficients are first sent, the highest order co-efficient $a_{n-1}\times b_{m-1}$ of the product polynomial is obtained first at the output of Fig 6.2(a). Then the co-efficient of X^{n+m-3} is obtained as the sum of $\{a_{n-2}b_{m-1} + a_{n-1} b_{m-2}\}$, the first term directly and the second term through the shift register **SR1**. Lower order co-efficients are then generated through the successive **SR**'s and mod-2 adders. After (**n + m - 2**) shifts, the **SR**'s contain $\{0, 0\ldots 0, a_0, a_1\}$ and the output is $(a_0 b_1 + a_1 b_0)$ which is the co-efficient of **X.** After **(n + m-1)** shifts, the SR's contain $(0, 0, 0,0, a_0)$ and the out put is $a_0\times b_0$. The product is now complete and the contents of the **SR**'s become (**0, 0, 0 …0, 0**). Fig 4.2(b) performs the multiplication in a similar way but the arrangement of the **SR**'s and ordering of the co-efficients are different (reverse order!). This modification helps to combine two multiplication operations into one as shown in Fig 4.2(c).

From the above description, it is clear that a non-systematic cyclic code may be generated using (**n-k**) shift registers. Following examples illustrate the concepts described so far.

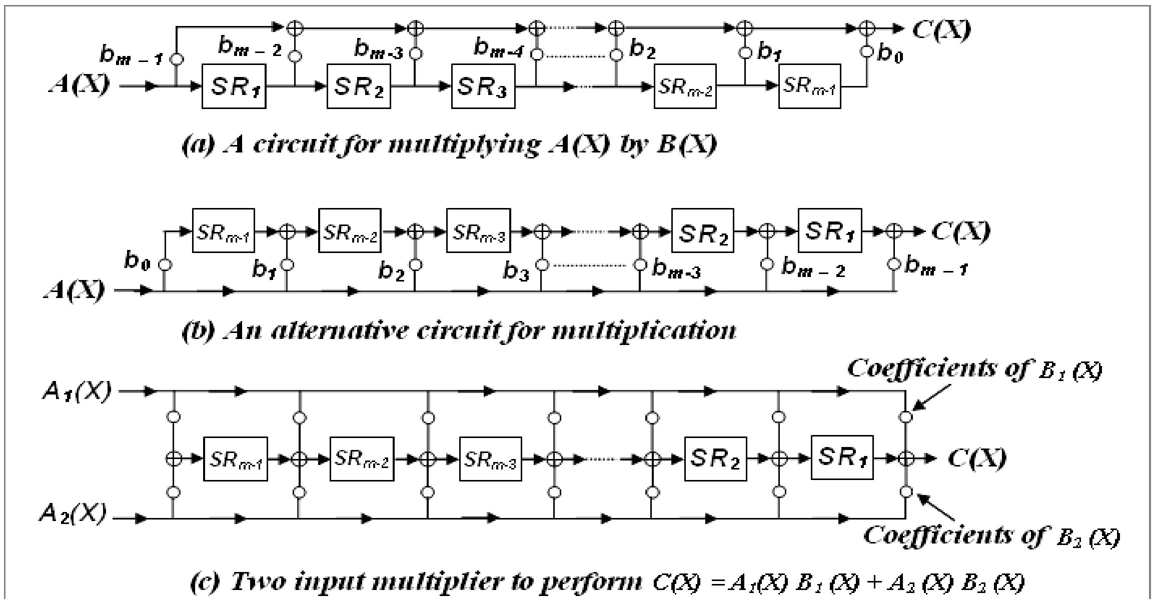


Fig 4.2: Multiplication circuits

Example 4.1: Consider that a polynomial **A(X)** is to be multiplied by

$$B(X) = 1 + X + X^3 + X^4 + X^6$$

The circuits of Fig 4.3 (a) and (b) give the product **C(X) = A(X). B(X)**

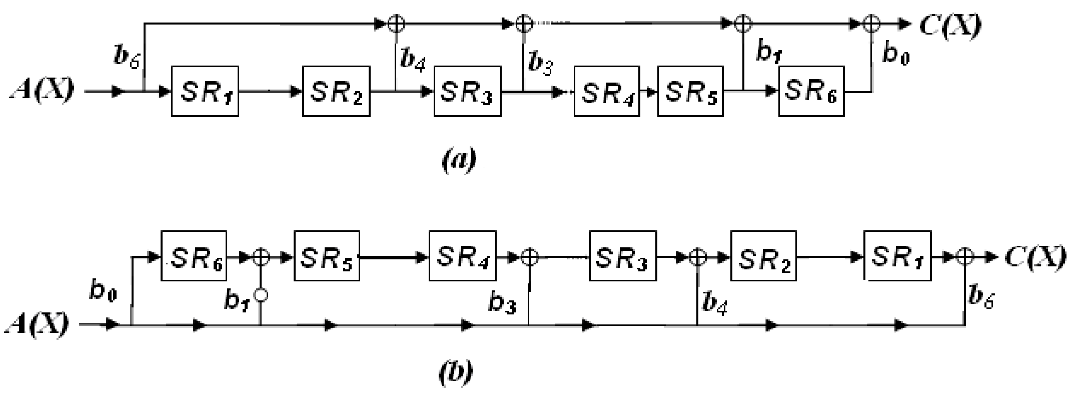


Fig 4.3: Circuit to perform C(X)*(1+X^2+X^3+X^4+X^6)

Example 4.2: Consider the generation of a (**7, 4**) cyclic code. Here **(n- k)** = (**7-4**) = **3** and we have to find a generator polynomial of degree **3** which is a factor of **$X^n + 1 = X^7 + 1$**.

To find the factors of' degree **3**, divide **X^7+1** by **X^3+aX^2+bX+1**, where '**a**' and '**b**' are binary numbers, to get the remainder as **abX^2+ (1 +a +b) X+ (a+b+ab+1).** Only condition for the remainder to be zero is **a +b=1** which means either **a = 1, b = 0** or **a = 0, b = 1**. Thus we have two possible polynomials of degree 3, namely

$$\mathbf{g_1 (X) = X^3+ X^2+ 1} \text{ and } \mathbf{g_2 (X) = X^3+X+1}$$

In fact, **X^7 + 1** can be factored as:

$$\mathbf{(X^7+1) = (X+1) (X^3+X^2+1) (X^3+X+1)}$$

Thus selection of a 'good' generator polynomial seems to be a major problem in the design of cyclic codes. No clear-cut procedures are available. Usually computer search procedures are followed.

Let us choose **g (x) = X^3+ X + 1** as the generator polynomial. The encoding circuits are shown in Fig 4.4(a) and (b).

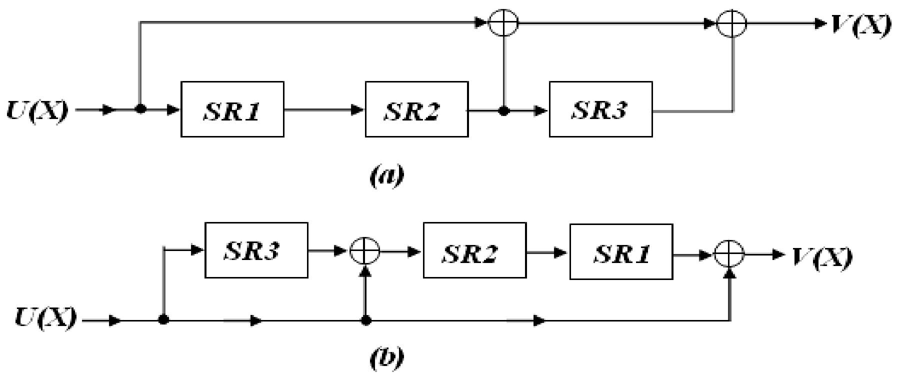


Fig 4.4 Generation of Non-systematic cyclic codes

To understand the operation, Let us consider **u = (10 1 1)** i.e.

$$U(X) = 1 + X^2 + X^3.$$

We have $V(X) = (1 + X^2 + X^3)(1 + X + X^3)$.

$$= 1 + X^2 + X^3 + X + X^3 + X^4 + X^3 + X^5 + X^6$$

$$= 1 + X + X^2 + X^3 + X^4 + X^5 + X^6 \quad \text{because } (X^3 + X^3 = 0)$$

=> **v = (1 1 1 1 1 1 1)**

The multiplication operation, performed by the circuit of Fig 6.4(a), is listed in the Table below step by step. In shift number 4, '**000**' is introduced to flush the registers. As seen from the tabulation the product polynomial is:

$$V(X) = 1 + X + X^2 + X^3 + X^4 + X^5 + X^6,$$

and hence out put code vector is **v = (1 1 1 1 1 1 1),** as obtained by direct multiplication. The reader can verify the operation of the circuit in Fig 4.4(b) in the same manner. Thus the multiplication circuits of Fig 6.4 can be used for generation of non-systematic cyclic codes.

Table showing sequence of computation

Shift Number	Input Queue	Bit shifted IN	Contents of shift registers.			Ou t put	Remarks
			SRI	SR2	SR3		
0	0001011	-	0	0	0	-	Circuit In reset mode
1	000101	1	1	0	0	1	Co-efficient of X^6
2	00010	1	1	1	0	1	Co-efficient of X^5
3	0001	0	0	1	1	1	X^4 co-efficient
*4	000	1	1	0	1	1	X^3 co-efficient
5	00	0	0	1	0	1	X^2 co-efficient
6	0	0	0	0	1	1	X^1 co-efficient
7	-	0	0	0	0	1	X^0co-efficient

4.3 DIVIDING CIRCUITS:

As in the case of multipliers, the division of **A (X)** by **B (X)** can be accomplished by using shift registers and Mod-2 adders, as shown in Fig 4.5. In a division circuit, the first co-efficient of the quotient is $(a_{n-1} \div (b_{m-1}) = q_1$, and $q_1.B(X)$ is subtracted from **A (X)**. This subtraction is carried out by the feed back connections shown. This process will continue for the second and subsequent terms.

However, remember that these coefficients are binary coefficients. After **(n-1)** shifts, the entire quotient will appear at the output and the remainder is stored in the shift registers.

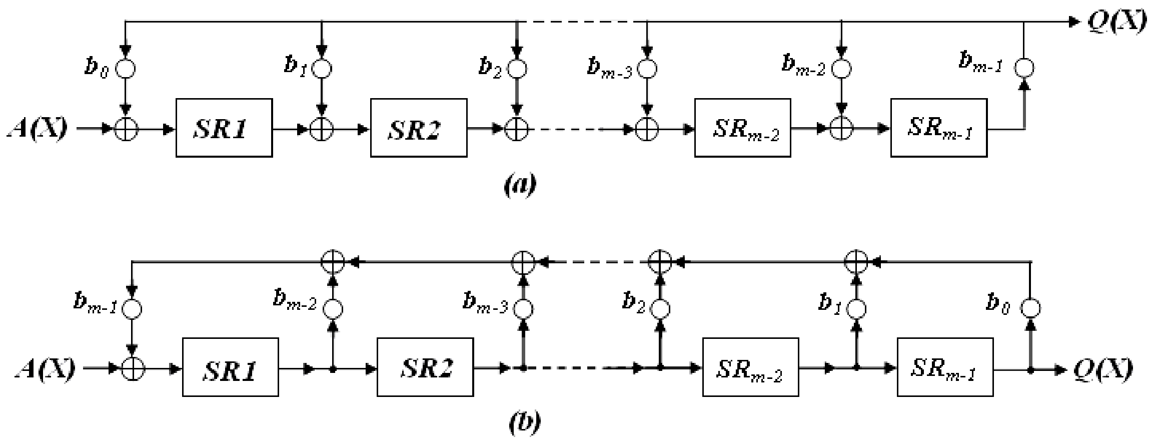


Fig 4.5: Dividing circuit

It is possible to combine a divider circuit with a multiplier circuit to build a "composite multiplier-divider circuit" which is useful in various encoding circuits. An arrangement to accomplish this is shown in Fig 4.6(a) and an illustration is shown in Fig 4.6(b).

We shall understand the operation of one divider circuit through an example. Operation of other circuits can be understood in a similar manner.

Example 4.3:

Let **A(X) = X^3+X^5+X^6**, **→A= (0001011)**, **B(X) = 1 +X+X^3**. We want to find the quotient and remainder after dividing **A(X)** by **B(X).** The circuit to perform this division is shown in Fig 4.7, drawn using the format of Fig 4.5(a). The operation of the divider circuit is listed in the table:

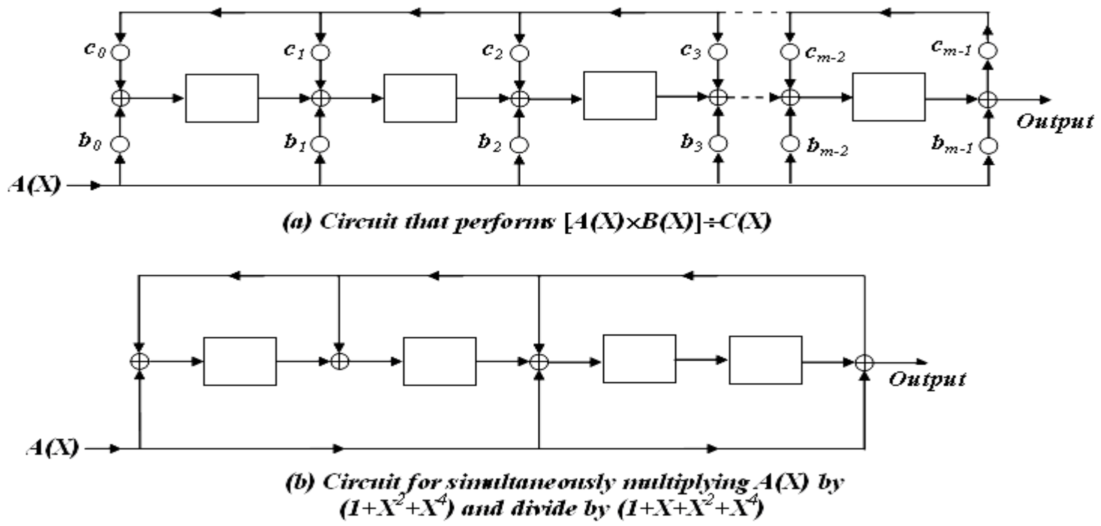

c_0
c_1
c_2
c_3
c_{m-2}
c_{m-1}
Output
b_0
b_1
b_2
b_3
b_{m-2}
b_{m-1}
A(X)
(a) Circuit that performs [A(X)×B(X)]÷C(X)
Output
A(X)
(b) Circuit for simultaneously multiplying A(X) by ($1+X^2+X^4$) and divide by ($1+X+X^2+X^4$)

Fig 4.6 Circuits for Simultaneous Multiplication and division

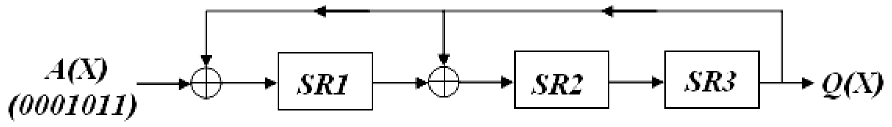


Fig 4.7 Circuits for dividing A(x) by $(1 + X + X^3)$

Table Showing the Sequence of Operations of the Dividing circuit

Shift Number	Input Queue	Bit shifted IN	Contents of shift Registers.			Out put	Remarks
			SRI	SR 2	SR 3		
0	**0001011**	**-**	**0**	**0**	**0**	**-**	**Circuit in reset mode**
1	**000101**	**1**	**1**	**0**	**0**	**0**	**Co-efficient of X^6**
2	**00010**	**1**	**1**	**1**	**0**	**0**	**Co-efficient of X^5**
3	**0001**	**0**	**0**	**1**	**1**	**0**	**X^4 co-efficient**
4	***000**	**1**	**0**	**1**	**1**	**1**	**X^3 co-efficient**
5	**00**	**0**	**1**	**1**	**1**	**1**	**X^2 co-efficient**
6	**0**	**0**	**1**	**0**	**1**	**1**	**X^1 co-efficient**
7	**-**	**0**	**1**	**0**	**0**	**1**	**X^o co-efficient**

The quotient co-efficients will be available only after the fourth shift as the first three shifts result in entering the first 3-bits to the shift registers and in each shift out put of the last register, **SR3**, is zero.

The quotient co-efficient serially presented at the out put are seen to be **(1111)** and hence the quotient polynomial is $\mathbf{Q(X) = 1 + X + X^2 + X^3}$. The remainder co-efficients are **(1 0 0)** and the remainder polynomial is **R(X) = 1**.

4.4 SYSTEMATIC CYCLIC CODES:

Let us assume a systematic format for the cyclic code as below:

$$\mathbf{v = (p_0, p_1, p_2 \ldots p_{n-k-1}, u_0, u_1, u_2 \ldots u_{k-1})} \quad \ldots\ldots\ldots\ldots\ldots \quad (4.15)$$

The code polynomial in the assumed systematic format becomes:

$$\mathbf{V(X) = p_0 + p_1X + p_2X^2 + \ldots + p_{n-k-1}X^{n-k-1} + u_0X^{n-k} + u_1X^{n-k+1} + \ldots + u_{k-1}X^{n-1}} \quad \ldots\ldots\ldots.. \quad (4.16)$$

$$\mathbf{= P(X) + X^{n-k}U(X)} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (4.17)$$

Since the code polynomial is a multiple of the generator polynomial we can write:

$$\mathbf{V(X) = P(X) + X^{n-k}U(X) = Q(X)\,g(X)} \qquad \text{.......................} \qquad (4.18)$$

$$\Rightarrow \frac{X^{n-k}U(X)}{g(X)} = Q(X) + \frac{P(X)}{g(X)} \qquad \ldots\ldots\ldots\ldots\ldots\ldots \qquad (4.19)$$

Thus division of **X^{n-k} U (X)** by **g (X)** gives us the quotient polynomial **Q (X)** and the remainder polynomial **P (X)**. Therefore to obtain the cyclic codes in the systematic form, we determine the remainder polynomial **P (X)** after dividing **X^{n-k} U (X)** by **g(X)**. This division process can be easily achieved by noting that "multiplication by **X^{n-k}** amounts to shifting the sequence by **(n-k)** bits". Specifically in the circuit of Fig 4.5(a), if the input **A(X)** is applied to the Mod-2 adder after the **(n-k)** th shift register the result is the division of **X^{n-k} A (X)** by **B (X).**

Accordingly, we have the following scheme to generate systematic cyclic codes. The generator polynomial is written as:

$$\mathbf{g(X) = 1 + g_1X + g_2X^2 + g_3X^3 + \ldots + g_{n-k-1}X^{n-k-1} + X^{n-k}} \qquad \ldots\ldots\ldots\ldots \qquad (4.20)$$

The circuit of Fig 4.8 does the job of dividing **X^{n-k}U (X)** by **g(X)**. The following steps describe the encoding operation.

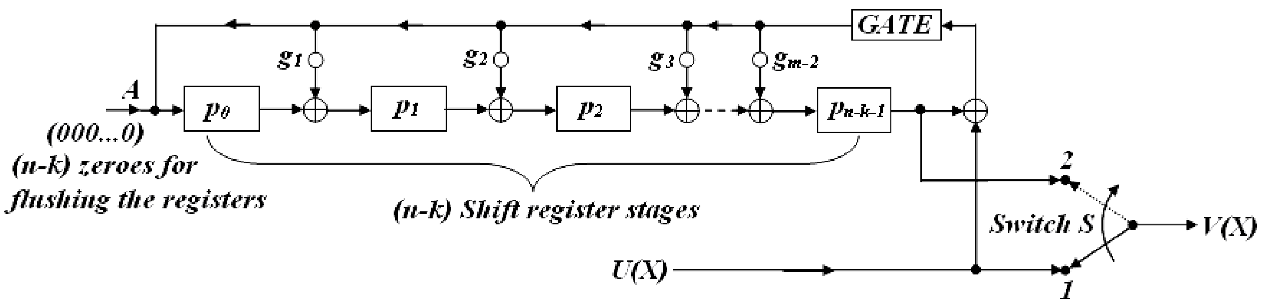


Fig 4.8 Syndrome encoding of cyclic codes using (n-k) shift register stages

1. The switch **S** is in position **1** to allow transmission of the message bits directly to an out put shift register during the first **k**-shifts.
2. At the same time the '**GATE**' is '**ON**' to allow transmission of the message bits into the **(n-k)** stage encoding shift register
3. After transmission of the **k**th message bit the **GATE** is turned **OFF** and the switch **S** is moved to position **2**.
4. **(n-k)** zeroes introduced at "**A**" after step **3**, clear the encoding register by moving the parity bits to the output register
5. The total number of shifts is equal to **n** and the contents of the output register is the code word polynomial **V (X) =P (X) + X^{n-k} U (X).**
6. After step-**4**, the encoder is ready to take up encoding of the next message input

Clearly, the encoder is very much simpler than the encoder of an **(n, k)** linear block code and the memory requirements are reduced. The following example illustrates the procedure.

Example 4.4:

Let **u = (1 0 1 1)** and we want a **(7, 4)** cyclic code in the systematic form. The generator polynomial chosen is **g (X) = 1 + X + X3**

For the given message, **U (X) = 1 +** X^2+X^3

$$X^{n-k}\,U(X) = X^3U(X) = X^3 + X^5 + X^6$$

We perform direct division $X^{n-k}U(X)$ **by g (X**) as shown below. From direct division observe that $p_0=1,\ p_1=p_2=0$. Hence the code word in systematic format is:

v = $(p_0, p_1, p_2; u_0, u_1, u_2, u_3) = (1, 0, 0, 1, 0, 1, 1)$

$$\begin{array}{r|l}
 & X^3+X^2+X+1 \\ \hline
 & X^6+X^5+X^3 \\
X^3+X^2+X+1 & X^6+X^4+X^3 \\ \cline{2-2}
 & X^5+X^4 \\
 & X^5+X^3+X^2 \\
 & X^4+X^3+X^2 \\
 & X^4+X^2+X \\ \cline{2-2}
 & X^3+X \\
 & X^3+X+1 \\ \cline{2-2}
 & 1 \quad \text{(Remainder)}
\end{array}$$

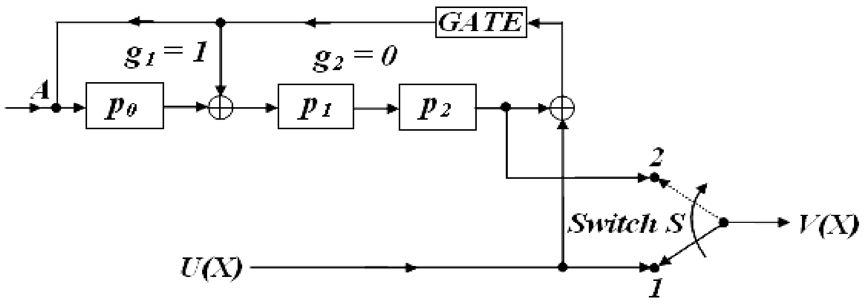


Fig 4.9 Encoder for the (7,4) cyclic code

The encoder circuit for the problem on hand is shown in Fig 4.9. The operational steps are as follows:

Shift Number	**Input Queue**	**Bit shifted IN**	**Register contents**	**Output**
0	**1011**	**-**	**000**	**-**
1	**101**	**1**	**110**	**1**
2	**10**	**1**	**101**	**1**
3	**1**	**0**	**100**	**0**
4	**-**	**1**	**100**	**1**

After the Fourth shift **GATE** Turned **OFF**, switch **S** moved to position **2,** and the parity bits contained in the register are shifted to the output. The out put code vector is **v = (100 1011)** which agrees with the direct hand calculation.

4.5 GENERATOR MATRIX FOR CYCLIC CODES:

The generator polynomial **g(X)** and the parity check polynomial **h(X)** uniquely specify the generator matrix **G** and the parity check matrix **H** respectively. We shall consider the construction of a generator matrix for a (**7, 4**) code generated by the polynomial $\mathbf{g(X) = 1 + X + X^3}$.

We start with the generator polynomial and its three cyclic shifted versions as below:

$$\begin{aligned} g(X) &= 1 + X + X^3 \\ X\,g(X) &= X + X^2 + X^4 \\ X^2 g(X) &= X^2 + X^3 + X^5 \\ X^3 g(X) &= X^3 + X^4 + X^6 \end{aligned}$$

The co-efficients of these polynomials are used as the elements of the rows of a **(4×7)** matrix to get the following generator matrix:

$$G = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Clearly, the generator matrix so constructed is not in a systematic format. We can transform this into a systematic format using **Row manipulations**. The manipulations are:

First row = First row; Second row = Second row; Third row = First row + Third row; and Fourth row = First row + second row + Fourth row.

These operations give the following result:

$$G = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} = [P \vdots I_4]$$

Using this generator matrix, which is in systematic form the code word for **u = (1 0 1 1)** is **v = (1 0 0 1 0 1 1)** (obtained as sum of 1st row + Third row + Fourth row of the **G**-matrix). The result agrees with direct hand calculation.

To construct **H**-matrix directly, we start with the reciprocal of the parity check polynomial defined by $\mathbf{X^k h(X^{-1})}$. Observe that the polynomial $\mathbf{X^k h(X^{-1})}$ is also a factor of the polynomial $\mathbf{X^n + 1}$. For the polynomial $\mathbf{(X^7+1)}$ we have three primitive factors namely, **(X + 1)**, $\mathbf{(X^3+X+1)}$ and $\mathbf{(X^3+X^2+1)}$. Since we have chosen $\mathbf{(X^3+X+1)}$ as the generator polynomial the other two factors should give us the parity check polynomial.

$$h(X) = (X + 1)(X^3+X^2+1) = X^4+X^2+X+1$$

There fore with $\mathbf{h(X) = 1 + X + X^2 + X^4}$, we have

$\mathbf{h(X^{-1}) = 1 + X^{-1} + X^{-2} + X^{-4}}$**,** and

$\mathbf{X^k\, h(X^{-1}) = X^4 h(X^{-1}) = X^4 + X^3 + X^2 + 1}$

The two cyclic shifted versions are:

$\mathbf{X^5 h(X^{-1}) = X^5 + X^4 + X^3 + X}$

$\mathbf{X^6 P(X^{-1}) = X^6 + X^5 + X^4 + X^2}$

Or $\mathbf{X^4 h(X^{-1}) = X^4 + X^3 + X^2 + 1}$

$\mathbf{X^5 h(X^{-1}) = X^5 + X^4 + X^3 + X}$

$\mathbf{X^6 h(X^{-1}) = X^6 + X^5 + X^4 + X^2}$

Using the co-efficients of these polynomials, we have:

$$H = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Clearly, this matrix is in non systematic form. It is interesting to check that for the non-systematic matrixes obtained $\mathbf{GH^T = O}$. We can obtain the **H** matrix in the systematic format H $= [I_3 \vdots P^T]$, by using **Row manipulations**. The manipulation in this case is simply. 'First row = First row + Third row'. The result is

$$H = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Observe the systematic format adopted: $G = [P \vdots I_k]$ *and* $H = [I_{n-k} \vdots P^T]$

4.6 SYNDROME CALCULATION - ERROR DETECTION AND ERROR CORRECTION :

Suppose the code vector $\mathbf{v = (v_0, v_1, v_2 \ldots v_{n-1})}$ is transmitted over a noisy channel. Hence the received vector may be a corrupted version of the transmitted code vector. Let the received code vector be $\mathbf{r = (r_0, r_1, r_2 \ldots r_{n-1})}$. The received vector may not be anyone of the $\mathbf{2^k}$ valid code vectors. The function of the decoder is to determine the transmitted code vector based on the received vector.

The decoder, as in the case of linear block codes, first computes the syndrome to check whether or not the received code vector is a valid code vector. In the case of cyclic codes, if the syndrome is zero, then the received code word polynomial must be divisible by the generator polynomial. If the syndrome is non-zero, the received word contains transmission errors and needs error correction. Let the received code vector be represented by the polynomial

$$\mathbf{R(X) = r_0+r_1X+r_2X^2+\ldots+r_{n-1}X^{n-l}}$$

Let **A(X)** be the quotient and **S(X)** be the remainder polynomials resulting from the division of **R(X)** by **g(X)** i.e.

$$\frac{R(X)}{g(X)} = A(X) + \frac{S(X)}{g(X)} \qquad \ldots\ldots\ldots\ldots\ldots\ldots.. \qquad (4.21)$$

The remainder **S(X)** is a polynomial of degree **(n-k-1)** or less. It is called the "Syndrome polynomial". If **E(X)** is the polynomial representing the error pattern caused by the channel, then we have:

$$\mathbf{R(X) = V(X) + E(X)} \qquad \ldots\ldots\ldots\ldots\ldots\ldots.. \qquad (4.22)$$

And it follows as **V(X) = U(X) g(X)**, that:

$$\mathbf{E(X) = [A(X) + U(X)]\, g(X) + S(X)} \qquad \ldots\ldots\ldots\ldots\ldots\ldots \qquad (4.23)$$

That is, the syndrome of R(X) is equal to the remainder resulting from dividing the error pattern by the generator polynomial; and the syndrome contains information about the error pattern, which can be used for error correction. Fig 4.5. A “**Syndrome calculator**” is shown in Fig 4.10.

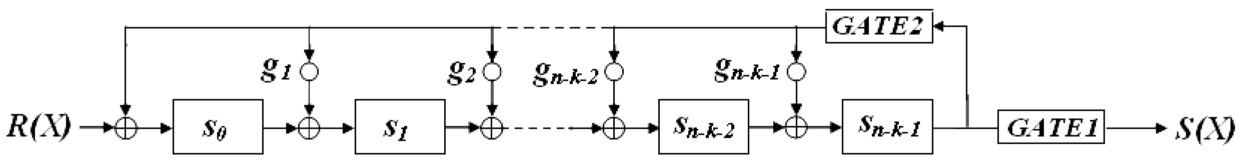


Fig 4.10 Syndrome calculator using (n-k) shift register

The syndrome calculations are carried out as below:

1 The register is first initialized. With **GATE 2 -ON** and **GATE1- OFF**, the received vector is entered into the register

2 After the entire received vector is shifted into the register, the contents of the register will be the syndrome, which can be shifted out of the register by turning **GATE-1 ON** and **GATE-2 OFF**. The circuit is ready for processing next received vector.

Cyclic codes are extremely well suited for **'error detection'** .They can be designed to detect many combinations of likely errors and implementation of error-detecting and error correcting circuits is practical and simple. Error detection can be achieved by employing (or adding) an additional R-S flip-flop to the syndrome calculator. If the syndrome is nonzero, the flip-flop sets and provides an indication of error. Because of the ease of implementation, virtually all error detecting codes are invariably 'cyclic codes'. If we are interested in error correction, then the decoder must be capable of determining the error pattern **E(X)** from the syndrome **S(X)** and add it to **R(X)** to determine the transmitted **V(X)**. The following scheme shown in Fig 6.11 may be employed for the purpose. The error correction procedure consists of the following steps:

Step1. Received data is shifted into the buffer register and syndrome registers with switches $\mathbf{S_{IN}}$ closed and $\mathbf{S_{OUT}}$ open and error correction is performed with $\mathbf{S_{IN}}$ open and $\mathbf{S_{OUT}}$ closed.

Step2. After the syndrome for the received code word is calculated and placed in the syndrome register, the contents are read into the error detector. The detector is a combinatorial circuit designed to output a '**1**' if and only if the syndrome corresponds to a correctable error pattern with an error at the highest order position $\mathbf{X^{n-l}}$. That is, if the detector output is a '**1**' then the received digit at the right most stage of the buffer register is assumed to be in error and will be corrected. If the detector output is '**0**' then the received digit at the right most stage of the buffer is assumed to be correct. Thus the detector output is the estimate error value for the digit coming out of the buffer register.

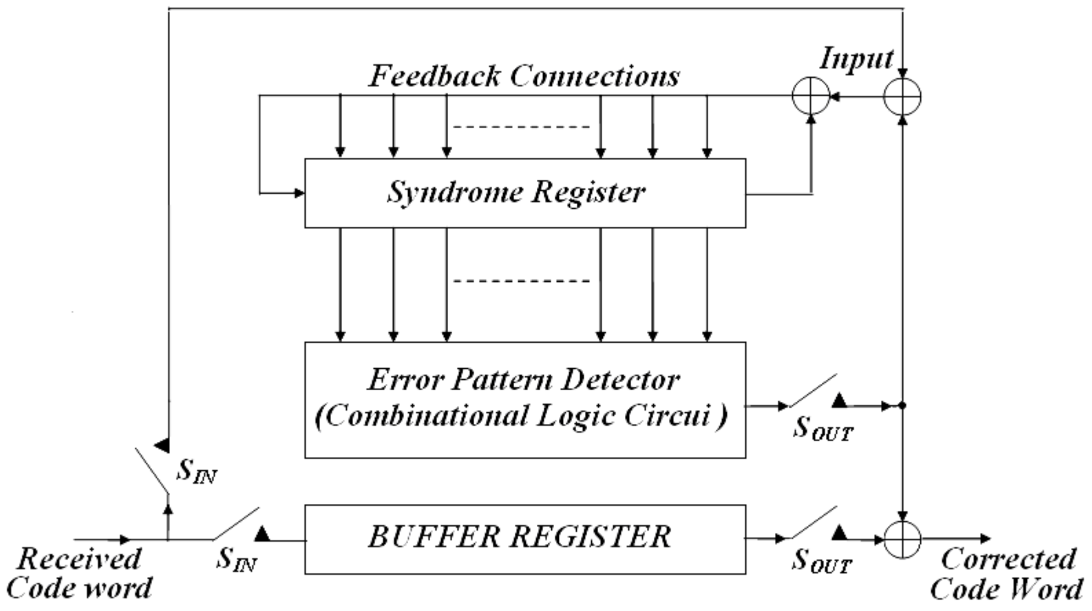


Fig 4.11 General decoder for cyclic code

Step3. In the third step, the first received digit in the syndrome register is shifted right once. If the first received digit is in error, the detector output will be '**1**' which is used for error correction. The output of the detector is also fed to the syndrome register to modify the syndrome. This results in a new syndrome corresponding to the '**altered** 'received code word shifted to the right by one place.

Step4. The new syndrome is now used to check and correct the second received digit, which is now at the right most position, is an erroneous digit. If so, it is corrected, a new syndrome is calculated as in step-3 and the procedure is repeated.

Step5. The decoder operates on the received data digit by digit until the entire received code word is shifted out of the buffer.

At the end of the decoding operation, that is, after the received code word is shifted out of the buffer, all those errors corresponding to correctable error patterns will have been corrected, and the syndrome register will contain all zeros. If the syndrome register does not contain all zeros, this means that an un-correctable error pattern has been detected. The decoding schemes described in Fig 6.10 and Fig6.11 can be used for any cyclic code. However, the practicality depends on the complexity of the combinational logic circuits of the error detector. In fact, there are special classes of cyclic codes for which the decoder can be realized by simpler circuits. However, the price paid for such simplicity is in the reduction of code efficiency for a given block size.

A decoder of the form described above operates on the received data bit by bit; and each bit is tested in turn for error and is corrected whenever an error is located. Such a decoder is called a"**Meggitt decoder**".

For illustration let us consider a decoder for a (**7, 4**) cyclic code generated by

$$\mathbf{g(X) = 1 + X + X^{3}}$$

The circuit implementation of the Meggitt decoder is shown on Fig 6.12. The entire received vector **R(X)** is entered in to the **SR**'s bit by bit and at the same time it is stored in the buffer memory. The division process will start after the third shift and after the seventh shift the syndrome will be stored in the **SR**'s. If **S(X) = (000)** then **E(X) = 0** and **R(X)** is read out of the buffer. Since **S(X)** can be found from **E(X)** with nonzero coefficients, suppose **E(X) = (000 0001)**. Then the **SR** contents are given as: **(001, 110, 011, 111, 101)** showing that **S(X) = (101)** after the seventh shift. At the eighth shift, the **SR** content is **(100)** and this may be used through a coincidence circuit to correct the error bit coming out of the buffer at the eighth shift. On the other hand if the error polynomial were **E(X) = (000 1000)** then the **SR** content will be **(100)** at he eleventh shift and the error will be corrected when the buffer delivers the error bit at the eleventh shift. The **SR** contents for different shifts, for two other error patterns are as shown in the table below:

SR contents for the error patterns (1001010) and (1001111)

Shift Number	Input	SR-content for (1001010)	Input	SR- content for (1001111)
1	0	000	1	100
2	1	100	1	110
3	0	010	1	111
4	1	101	1	001
5	0	100	0	110
6	0	010	0	011
7	1	101	1	011 *Indicates an error
8	0	100	0	111
9	-	-	0	101
10	-	-	0	100

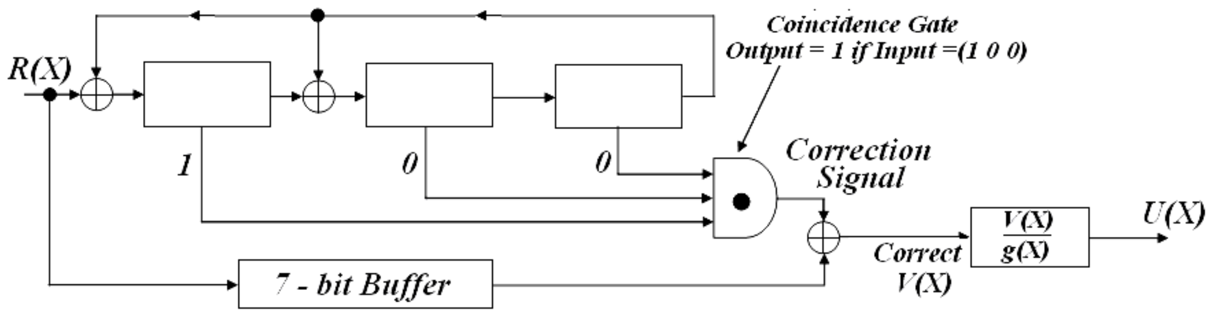


Fig 4.12 Meggitt decoder for (7,4) cyclic code

For **R(X) = (1001010)**, the **SR** content is **(100)** at the **8-th** shift and the bit in $\mathbf{X^6}$ position of **R(X)** is corrected giving correct **V(X) = (1001011).** On the other hand , if **R(X) = (1001111)**, then it is seen from the table that at the **10-th** shift the syndrome content will detect the error and correct the $\mathbf{X^4}$ bit of **R(X)** giving **V(X) = (1001011).**

The decoder for the (**15, 11**) cyclic code, using $\mathbf{g(X) = 1 + X + X^4}$, is shown in Fig 6.13. It is easy to check that the SR content at the **16-th** shift is **(1000)** for $\mathbf{E(X) = X^{14}}$. Hence a coincidence circuit gives the correction signal to the buffer out put as explained earlier.

Although the Meggitt decoders are intended for Single error correcting cyclic codes, they may be generalized for multiple error correcting codes as well, for example (**15, 7**) BCH code.
An **error trapping decoder** is a modification of a Meggitt decoder that is used for certain cyclic codes.

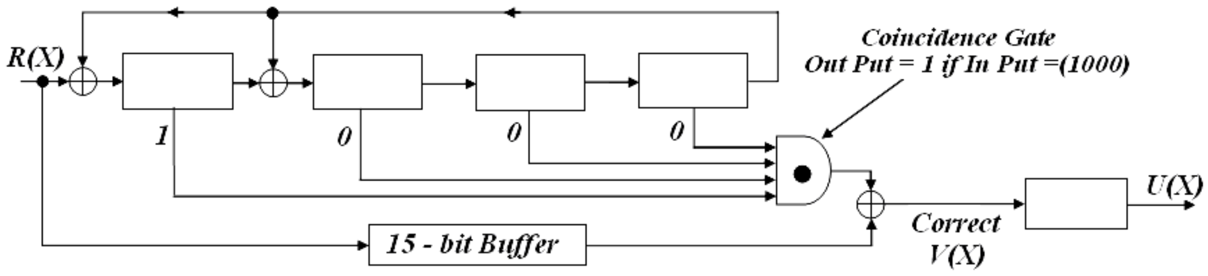


Fig 4.13 Meggitt decoder for (15,11) cyclic code

The syndrome polynomial is computed as: **S(X) = Remainder of [E(X) / g(X)].** If the error **E(X)** is confined to the **(n-k)** parity check positions $\mathbf{(1, X, X^2\ldots\ X^{n-k-1})}$ of **R(X),** then **E(X) = S(X)**, since the degree of **E(X)** is less than that of **g(X)**. Thus error correction can be carried out by simply adding **S(X)** to **R(X)**. Even if **E(X)** is not confined to the **(n-k)** parity check positions of **R(X)** but has nonzero values clustered together such that the length of the nonzero values is less than the syndrome length, then also the syndrome will exhibit an exact replica of the error pattern after some cyclic shifts of **E(X)**. For each error pattern, the syndrome content **S(X)** (after the required shifts) is subtracted from the appropriately shifted **R(X)**, and the corrected **V(X)** recovered.

"**If the syndrome of R(X) is taken to be the remainder after dividing $\mathbf{X^{n-k}}$ R(X) by g(X), and all errors lie in the highest-order (n-k) symbols of R(X), then the nonzero portion of the error pattern appears in the corresponding positions of the syndrome**". Fig 4.14 shows an error trapping decoder for a (**15, 7**) **BCH** code based on the principles described above. A total of **45** shifts are required to correct the double error, **15** shifts to generate the syndrome, **15** shifts to correct the first error and **15** shifts to correct the second error.

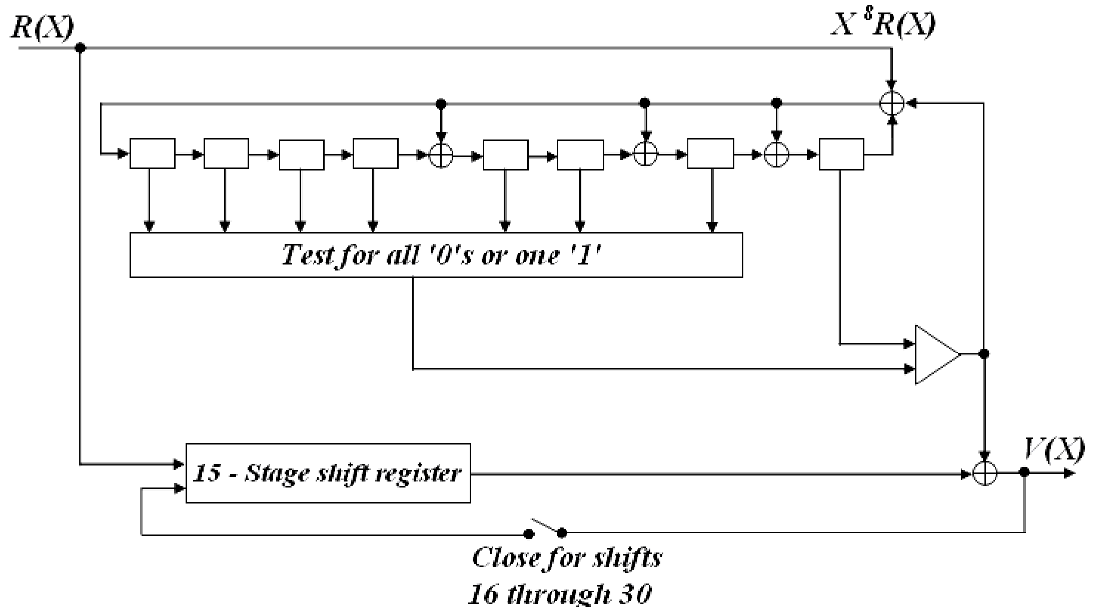


Fig 4.14 Error trapping decoder for (15,7) BCH code

Illustration:

$U(X) = X^6 + 1$; $g(X) = X^8 + X^7 + X^6 + X^4 + 1$

$V(X) = X^{14} + X^{13} + X^{12} + X^{10} + X^8 + X^7 + X^4 + 1$

$E(X) = X^{11} + X$

$R(X) = X^{14} + X^{13} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^4 + X + 1$

r = (110010011011111)

Shift Number	Syndrome Generator Register	Shift Number	Middle Register	Shift Number	Bottom Register
1	10001011	16	01100011	31	00000100
2	01000101	17	10111010	32	00000010
3	00100010	18	01011101	33	00000001
4	10011010	19	10100101	34	00000000
5	11000110	20	11011001	35	;
6	01100011	21	11100111	36	:
7	00110001	22	11111000	37	:
8	00011000	23	01111100	38	All zeros
9	00001100	24	00111110	39	:
10	00000110	25	00011111	40	:
11	10001000	26	10000100	41	:
12	01000100	27	01000010	42	:

13	00100010	28	00100001	43	:
14	10011010	29	00010000	44	:
15	11000110	30	00001000	45	:

Errors trapped at shift numbers 28 and 33.

Some times when error trapping cannot be used for a given code, the test patterns can be modified to include the few troublesome error patterns along with the general test. Such a modified error trapping decoder is possible for the (**23, 12**) Golay code in which the error pattern **E(X)** will be of length **23** and weight of **3** or less (**t** ≤ **3**). The length of the syndrome register is **11** and if **E(X)** has a length greater than 11the error pattern is not trapped by cyclically shifting **S(X)**. In this case, it is shown that one of the three error bits must have at least five zeros on one side of it and at least six zeros the other side. Hence all error patterns can be cyclically shifted into one of the following three configurations (numbering the bit positions, $\mathbf{e_0, e_1, e_2 \dots e_{22}}$):

(i) All errors (**t ≤ 3**) occur in the **11** high-order bits

(ii) One error occurs in position $\mathbf{e_5}$ and the other two errors occur in the **11** high-order bits.

(iii) One error occurs in position $\mathbf{e_6}$ and the remaining two errors occur in the **11** high-order bits.

In the decoder shown in Fig 4.15, the received code vector **R(X)** is fed at the rightmost stage of the syndrome generator (as was done in Fig 6.14), equivalent to multiplying **R(X)** by $\mathbf{X^{11}}$. Then the syndrome corresponding to $\mathbf{e_5}$ and $\mathbf{e_6}$ are obtained (using $\mathbf{g_1(X)}$ as the generator polynomial) as:

$$\mathbf{S(e_5) = \text{Remainder of } [X^{16}/g_1(X)] = X + X^2 + X^5 + X^6 + X^8 + X^9} \quad \text{and}$$
$$\mathbf{S(e_6) = \text{Remainder of } [X^{17}/g_1(X)] = X^2 + X^3 + X^6 + X^7 + X^9 + X^{10}}$$

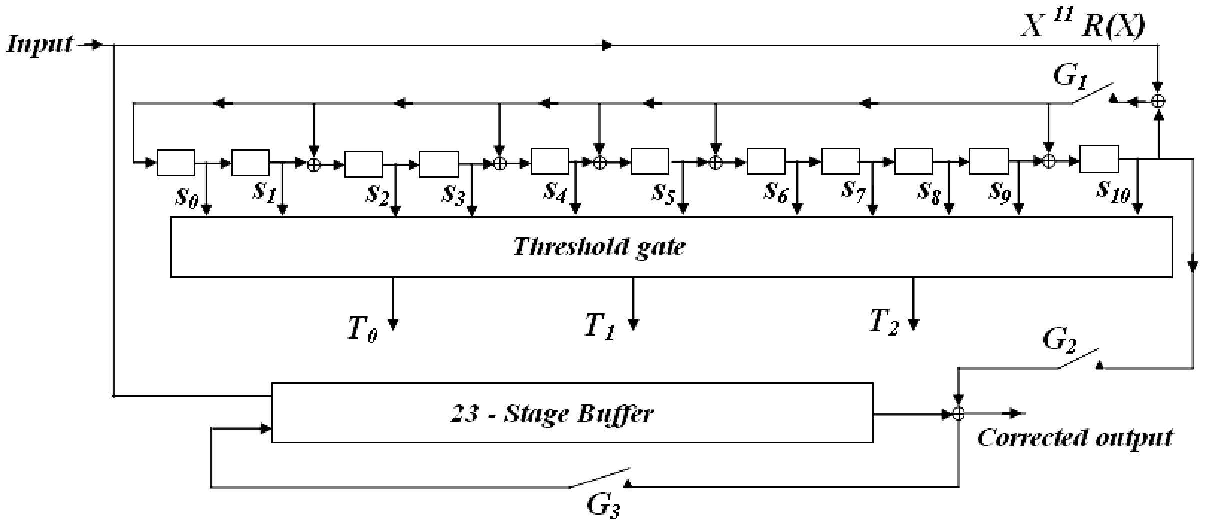


Fig 4.15 Error trapping decoder for (23,12) Golay code

The syndrome vectors for the errors $\mathbf{e_5}$ and $\mathbf{e_6}$ will be (**01100110110**) or (**00110011011**) respectively. Two more errors occurring in the **11** high-order bit positions will cause two **1**'s in the appropriate positions of the syndrome vectors, thereby complementing the vector for $\mathbf{e_5}$ or $\mathbf{e_6}$. Based on the above relations, the decoder operates as follows:

(i).The entire received vector is shifted into the syndrome generator (with switch $\mathbf{G_1}$ closed) and the syndrome **S(X)** corresponding to $\mathbf{X^{11} R(X)}$ is formed.

(ii).If all the three or less errors are confined to $\mathbf{X^{12}}$, $\mathbf{X^{13}}$ … $\mathbf{X^{22}}$ of **R(X)**, then the syndrome matches the errors in these positions. The weight of the syndrome is now 3 or less. This is checked by a threshold gate and the gate output $\mathbf{T_0}$ switches $\mathbf{G_2}$ ON and $\mathbf{G_1}$ OFF. R(X) is now received from the buffer and corrected by the syndrome bits (as they are clocked bit by bit) through the modulo-2 adder circuit.

(iii).If the test in (ii) fails then it is assumed that one error is either at $\mathbf{e_5}$ or at $\mathbf{e_6}$**,** and the other **two** errors are in the **11** high-order bits of **R(X)**. Then if the weight of **S(X)** is more than **3** (in test (ii)), then the weights of [**S(X) + S (**$\mathbf{e_5}$**)**] and [**S(X) + S (**$\mathbf{e_6}$**)**] are tested. The decisions are:

1. If weight of [**S(X) + S (**$\mathbf{e_5}$**)**] ≤ **2** then the decision ($\mathbf{T_1}$ = **1**) is that one error is at position $\mathbf{e_5}$ and two errors are at positions where [**S(X) + S (**$\mathbf{e_5}$**)**] are nonzero.

2. If weight of [**S(X) + S (**$\mathbf{e_6}$**)**] ≤ **2** then the decision ($\mathbf{T_2}$ = **1**) is that one error is at position $\mathbf{e_6}$ and two errors are at positions where [**S(X) + S (**$\mathbf{e_6}$**)**] are nonzero. The above tests are arranged through combinatorial switching circuits and the appropriate corrections in **R(X)** are made as **R(X)** is read from the buffer.

(iv). If the above tests fail then with $\mathbf{G_1}$ and $\mathbf{G_3}$ **ON** and $\mathbf{G_2}$ **OFF**, the syndrome and buffer contents are shifted by **one** bit. Tests (ii) and (iii) are now repeated. Bit by bit shifting of **S(X)** and **R(X)** is continued till the errors are located, and then corrected. A maximum of **23** shifts will be required to complete the process. After correction of **R(X)**, the corrected **V(X)** is further processed through a divider circuit to obtain the message **U(X) = V(X) / g(X)**.

Assuming that upon shifting the block of **23** bits with **t** ≤ **3** cyclically, 'at most one error will lie outside the **11** high-order bits of **R(X)**' at some shift, an alternative decoding procedure can be devised for a Golay coder – The systematic search decoder. Here the test (ii) is first carried out. If the test fails, then **first bit** of **R(X)** is inverted and a check is made to find if the weight of **S(X) ≤ 2**. If this test is successful, then the nonzero positions of **S(X)** give the **two** error locations (similar to test (iii) above) and the other error is at **first** position. If this test fails, then the syndrome content is cyclically shifted, each time testing for weight of **S(X) ≤ 3**; and if not, invert **2nd**, **3rd** ……and **12th** bit of **R(X)** successively and test for weight of **S(X) ≤ 2**. Since all errors are not in the parity check section, an error must be detected in one of the shifts. Once one error is located and corrected, the other two errors are easily located and corrected by test (ii). Some times the systematic search decoder is simpler in hardware than the error trapping decoder, but the latter is faster in operation. The systematic search decoder can be generalized for decoding other multiple-error-correcting cyclic codes. It is to be observed that the Golay (**23, 12**) code cannot be decoded by majority-logic decoders.

RECOMMENDATION QUESTIONS

1. Sketch the shift register circuit for multiplying **A(X)** by $\mathbf{B(X) = 1 + X^3 + X^4 + X^5 + X^6}$. Compute the circuit output for $\mathbf{A(X) = 1 + X + X^4}$ by examining the shift register contents at each shift.

2. Draw a shift register circuit for simultaneous multiplication by **B(X)** and division by **D(X)**, where $\mathbf{B(X) = 1 + X^2 + X^5 + X^6}$ and $\mathbf{D(X) = 1 + X^3 + X^4 + X^5 + X^8}$
   If the input to the circuit is $\mathbf{A(X) = 1 + X^7}$, calculate the quotient and the remainder.

3. Determine which, if any, of the following polynomials can generate a cyclic code with code word length $\mathbf{n \leq 7}$. Find the **(n, k)** values of any such codes that can be generated.

   (a) $\mathbf{1 + X^3 + X^4}$ **(d)** $\mathbf{1 + X + X^2 + X^4}$
   (b) $\mathbf{1 + X^2 + X^4}$ **(e)** $\mathbf{1 + X^3 + X^5}$
   (c) $\mathbf{1 + X + X^3 + X^4}$

4. A (**15, 5**) linear cyclic code has a generator polynomial: $\mathbf{g(X) = 1+X+X^2+X^4+X^5+X^8+X^{10}}$

   a) Draw block diagrams of an encoder and syndrome calculator for this code.
   b) Find the code polynomial for the message polynomial $\mathbf{U(X) = 1+X^2+X^4}$ in systematic form.
   c) Is $\mathbf{V(X) = 1+X^4+X^6+X^8+X^{14}}$ a code polynomial? If not, find the syndrome of **v** (X).
   d) Hard-decision detection gives the received code word as:

   $\mathbf{R(X) = 1 + X^4 + X^7 + X^8 + X^9 + X^{10} + X^{14}}$. Locate the errors.

5. Write the H Matrix for the (**15, 11**) cyclic code using $\mathbf{g(X) = 1 + X + X^2 + X^3 + X^4}$ determine the code polynomial for $\mathbf{U(X) = 1 + X^3 + X^7 + X^{10}}$**.** Construct the decoder for the code.

6. Write the generator polynomial for (**31, 26**) **SEC** cyclic code. Write the **G** and **H** matrices for the code. Find the code polynomial for the message:

   $\mathbf{U(X) = 1 + X + X^4 + X^{20}}$**.** Draw the Meggitt decoder circuit for the code.

7. Write the **G** matrix for the (**15, 11**) **SEC** code. Show that by successively removing some of the rows of **G**, one obtains (**14, 10**), (**13, 9**), (**12, 8**), (**11, 7**), (**10, 6**) etc. **SEC** codes. Write **H** matrix for the (**10, 6**) code. Construct the coder and decoder for the code.

8. Construct a Meggitt decoder for the (**15, 7**) **BCH** code. What are the error patterns which will form the test syndromes for the decoder? Is it possible to reduce the test syndromes to only 8 error patterns as below:

   0 0 0 0 0 0 0 1, 0 0 0 0 0 0 1 1
   0 0 0 0 0 1 0 1, 0 0 0 0 1 0 0 1
   0 0 0 1 0 0 0 1, 0 0 1 0 0 0 0 1
   0 1 0 0 0 0 0 1, 1 0 0 0 0 0 0 1

   What is the modification required in the new decoder? What is the total number of shifts required to complete the decoding of a block?

9. The generator polynomial for a cyclic code is **$g(X) = 1+X^4+X^6+X^7+X^8$**
   a) Show that its length is **15.**
   b) Find the generator matrix and parity check matrix in systematic form.
   c) Devise two shift register encoder circuits using **$k = 7$** stages and **$(n - k) = 8$** stages.
   d) Find the code vector (In systematic form) for the message polynomial
   $U(X) = 1+X^2+X^3+X^4$
   e) Assume that the first and last bits of the code vector **V(X)** for **U(X)** given in (a) Suffer transmission errors. Find the syndrome of **V(X)**.

10. The decoder for a class of single error correcting cyclic codes (Hamming codes) is shown in Fig **P7.1** Show by way of an example, that single error in a **(15, 11)** Hamming code generated by **$g(X) = 1+X+X^4$** can be decoded using the decoder shown in the figure.

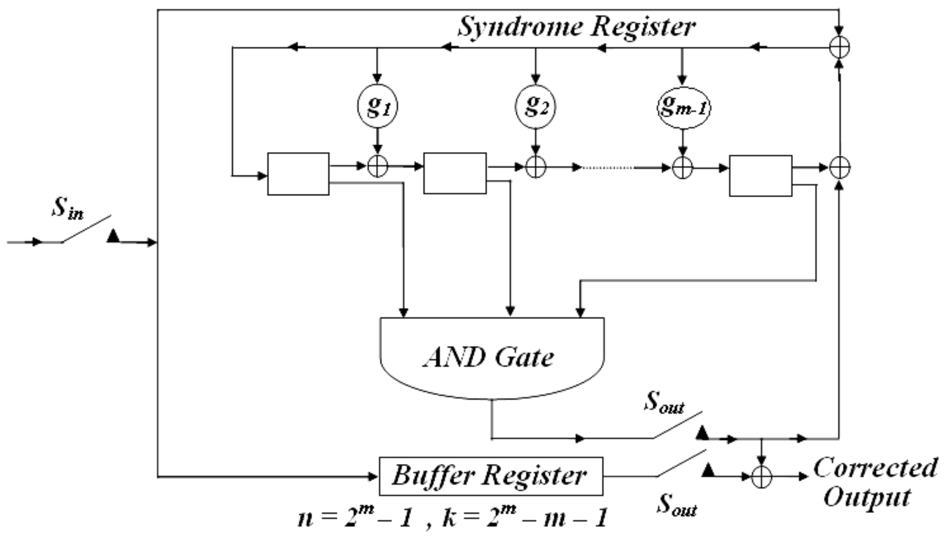


Fig P 7.1 Decoder for Hamming codes.

11. A **t**-error correcting code is said to be a "perfect code" if it is possible to form a standard array, with all patterns of'**t**' or fewer errors and no others as co-set leaders. Show that a (**7, 4**) linear block code generated by **$g(X) = 1+X+X^3$** is a perfect code.

12. The "**Expurgated (n, k-1) Hamming code**" is obtained from the original (**n, k**) Hamming code by discarding some of the code words. Let **g(X)** Denote the generator polynomial of the original code. The most common expurgated code is the one generated by **$g_1(X) = (1+X)\ g(X)$**, where **(1+X**) is a factor of **$(1+X^n)$.** Consider the **(7, 4)** Hamming code generated by **$g(X) = 1+X^2+X^3$.**

   a) Construct the eight code words in the expurgated **(7, 3)** Hamming code, assuming a systematic format. Hence show that the minimum distance of the code is 4.
   b) Determine the generator matrix **G** and the parity check matrix **H** of the expurgated Hamming code.
   c) Device the encoder and syndrome calculator for the expurgated Hamming code. Hence determine the syndrome for the received sequence **0111110**.

13. A systematic **(7, 4)** cyclic code is generated by **$g(X) = 1 + X^2 + X^3$**. The message is **$U(X) = 1+X+X^3$**, and after detection the effective error polynomial is **$E(X) = X^4$**. Find the first syndrome word generated by a Meggitt decoder for decoding the first received symbol.

14. For the **(7, 4)** cyclic code generated by **g(X)** of problem 9, write the generator matrix and the parity check matrices. Delete the last three columns in the **H** matrix in order to generate a **(4, 1)** shortened cyclic code. If the code word **v = (1101)** is received as **r = (1111)**, show that the decoder corrects the error on the fifth clock pulse.

15. Show that a binary cyclic code of length **n** generated by **g(X)** has minimum weight of at least **3** if **n** is the smallest integer for which **g(X)** divides $(X^n + 1)$.

16. Let **G = [I: P]** be the generator matrix of a cyclic code. If $h(X)=1+h_1X+h_2X^2+\ldots+h_{k-1}X^{k-1}+X^k$ is the parity check polynomial of the code, show that the last column of the matrix **P** is:

$$(h_{k-1}, h_{k-2} \ldots.. h_2, h_1, 1)$$

17. A (**31, 21**) binary double error correcting code has the generator polynomial:

$$g(X) = 1 + X^3 + X^5 + X^6 + X^8 + X^9 + X^{10}$$

   i. Show that an error-trapping decoder cannot decode this code to the designed distance.
   ii. Show a simple modification of the error-trapping decoder that will decode to the designed distance.

18. The triple-error-correcting Golay (**23, 12**) code may be constructed with the generator polynomial: $g(X) = 1 + X + X^5 + X^6 + X^7 + X^9 + X^{11}$. Find the parity check polynomial h(X). Construct a decoder for the code. What are the test syndromes to be checked in the decoder?

OUTCOMES

- How the cyclic codes can be implemented using feedback shift registers.
- How the errors can be detected and corrected.

REFERENCE

- www.mcs.csueastbay.edu/~malek/Class/Cyclic.pdf
- www.fi.muni.cz/usr/.../CHAPTER%2003%20-%20Cyclic%20codes.ppt
- elearning.vtu.ac.in/P4/EC63/S11.pdf
- nptel.ac.in/courses/IIT...Of.../Lecture40-41_ErrorControlCoding.pdf