Data Coding and Error Checking Techniques

Bruce Peterson
Accolade Engineering Solutions
15520 Rockfield Blvd., Suite H
Irvine, CA 92618
949-597-8378
www.accoladeeng.com

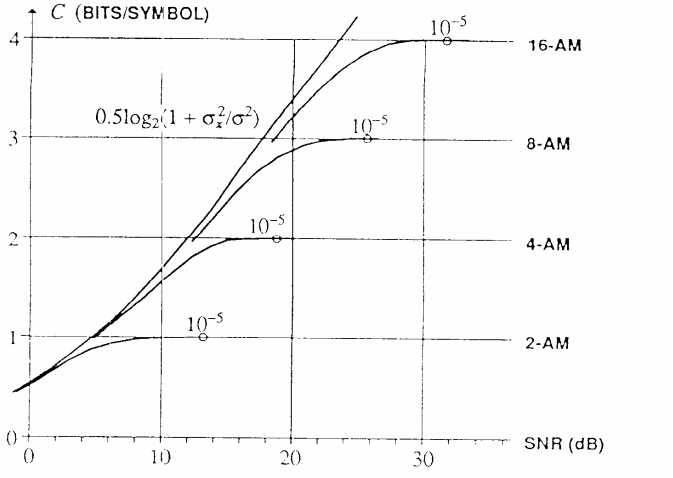
1.0 Introduction

The purpose of coding data is to efficiently transport it through a particular medium. The medium may be circuit board traces, ribbon cable, twisted pair copper, copper coax, fiber optic or air. Each type of medium suffers from a group of impairments. These impairments may include signal reflection, attenuation distortion, harmonic distortion, phase distortion, intermodulation distortion, dropout, echo, crosstalk, delay distortion manifesting itself as inter symbol interference (ISI), impulse noise, Gaussian noise and frequency shift, All these impairments in the medium affect the ability to transport data. In some cases, these factors can cause an excessive number of bit errors. For short transmission line lengths and low signaling rates, the simple linear lines codes may be employed. These codes may be unipolar or bipolar and may or may not have clocking information contained within the code. When the channel is bandwidth limited, more efficient codes are available. Such codes may utilize multi-level symbols and alter the message data to allow the receiver to synchronize to it. The most sophisticated codes use block coding or convolutional coding to improve the performance of transmission. For a bandwidth limited channel, the maximum upper limit for reliable information transfer is given by the Hartley-Shannon Law. This law equates the channel bandwidth and the signal to noise ratio to the maximum channel capacity and indicates the maximum number of symbols that can be transferred per second. This equation is given below:

$$C = B * \log_2(1 + SNR) \text{ symbols/second}$$

where: C is the channel capacity
B is the channel bandwidth
SNR is the signal to noise ratio

This equation implies that we can trade channel bandwidth for signal to noise ratio. When the data is coded, the system can tolerate a lower signal to noise ratio for the same bit rate. This difference is called coding gain and is expressed in dB. An example of coding gain is shown using the figure below.



For the 4-AM constellation, the dot in the curve shows that with the bit error rate of 10^{-5} an uncoded system can reliably transfer 2 bits per symbol with a signal to noise ratio of 19dB. If coding is used with the 8-AM constellation, we can transmit at essentially the same data rate with signal to noise ratio of about 13dB. Using coding and the 8-AM constellation, the coding gain is 19dB - 13dB = 6dB. Since the data rate is not increased over the 4-AM case, we can make use of all 6dB of the coding gain. When the symbol rate is increased, the additional noise will lower the signal to noise ratio causing some of the coding gain to be lost. In most cases, the coding gain exceeds the loses due to the additional noise. The overall gain (coding gain - noise gain) will allow higher bit rates than non-coded systems. The table below shows the coding gains and the noise gains for three of the coding techniques discussed in this paper.

Type of Code	Coding Gain	Noise Gain
Single-Parity Check	3 dB	1.75dB
Hamming	4.7 dB	2.36 dB
1/2 Convolutional	7 dB	3 dB

2.0 Linear Line Codes

Linear line codes are those in which the transmitted data depend linearly on the information bits.

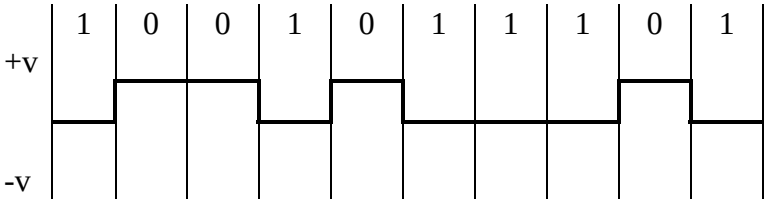
2.1 Binary antipodal codes

The binary antipodal codes require the minimum bandwidth but lack clocking information for receiver synchronization. These codes also have a DC content to their spectrum. The RZ code uses twice the bandwidth of the NRZ or NRZI codes. These codes are used only for the simplest communication systems where the transmitter and receiver are relatively close, DC coupled and low speed.

2.1.1 NRZ-L

rules:

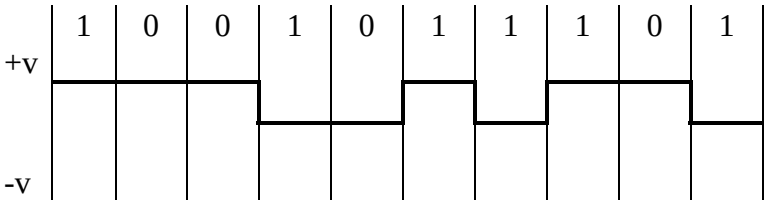
1. 0 = "+"
2. 1 = "0"



2.1.2 NRZI

rules:

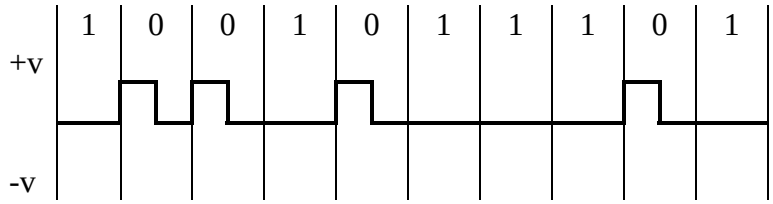
1. 0 = no transition at beginning of interval
2. 1 = transition at beginning of interval



2.1.3 Unipolar RZ

rules:

1. 0 = transition from “0” to “+” at start of interval
   transition from “+” to “0” at middle of interval
2. 1 = “0”



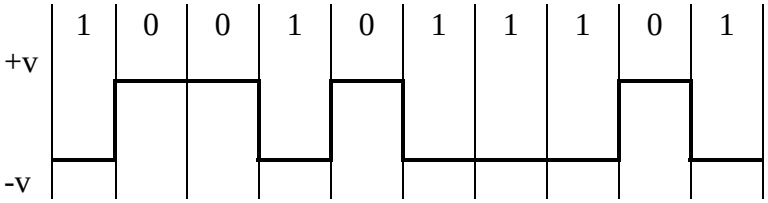
2.2 Binary Bipolar Codes

Like the binary antipodal codes, the polar NRZ and polar RZ require the minimum bandwidth but lack clocking information for receiver synchronization. If the number of ones and zeros are equal they have no DC content to their spectrum. These codes by themselves do not guarantee an equal number of ones and zeros. In these cases, special encoding techniques, such as scrambling, are employed to provide nearly an equal number of high and low pulses. The polar NRZ is the type of coding technique used by the serial port of a personal computer. The Biphase or Manchester pulses have the property of eliminating the DC component but require twice the bandwidth. The Biphase and Manchester also have the property that they contain clocking information which may be extracted by the receiver. Manchester coding is the coding method used for 802.3 (Ethernet).

2.2.1 Polar NRZ

rules:

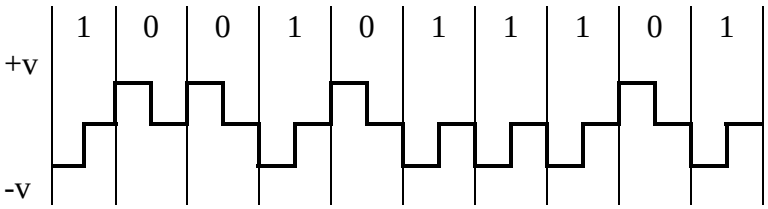
1. 0 = "+"
2. 1 = "-"



2.2.2 Polar RZ

rules:

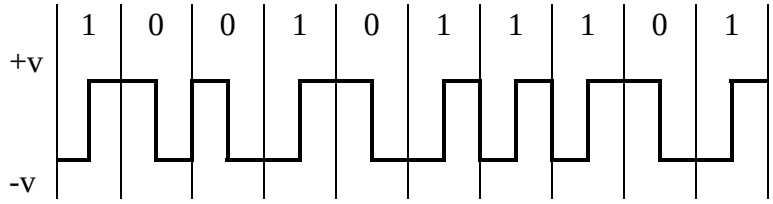
1. 0 = transition from "0" to "+" at start of interval
   transition from "+" to "0" at middle of interval
2. 1 = transition from "0" to "-"at start of interval
   transition from "-" to "0" at middle of interval



2.2.3 Manchester

rules:

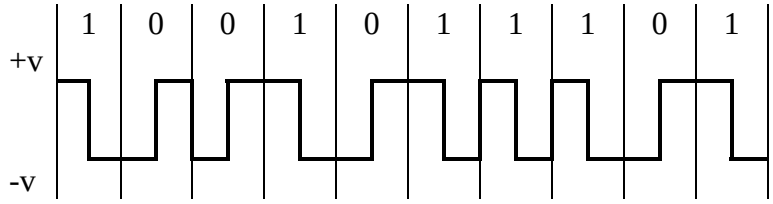
1. 0 = transition from “+” to “-” in middle of interval
2. 1 = transition from “-” to “+” in middle of interval



2.2.4 Biphase-L or Manchester II

rules:

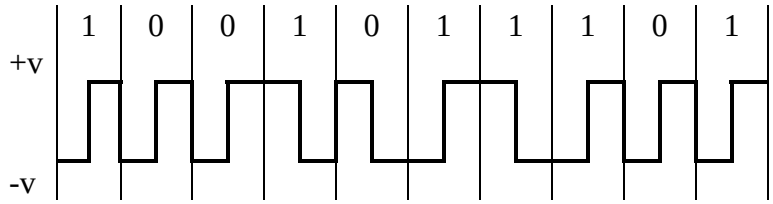
1. 0 = transition from “-” to “+” in middle of interval
2. 1 = transition from “+” to “-” in middle of interval



2.2.5 Differential Manchester

rules:

1. always transition in middle of interval
2. 0 = transition at beginning of interval
3. 1 = no transition at beginning of interval



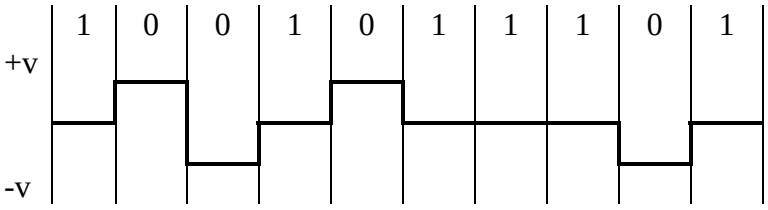
2.3 Twinned binary codes

To eliminate the DC component from the spectrum and to keep from increasing the symbol rate, the twinned binary codes can be used. A twinned binary code increases the number of available symbols by increasing the number of signal levels. The pseudoternary, Bipolar-AMI and the twinned binary codes are examples of codes which utilize three signal levels. The main disadvantage of Twinned binary codes is a slight loss in noise immunity. The most widely used of the multilevel binary codes is the Bipolar-AMI code. The Bipolar-AMI code was first used in the PCM based T1-carrier system by Bell Labs in 1962. The Bipolar-AMI suffers in that the receiver loses clocking information when a continuous stream of zeros is transmitted. The next section will describe a variation of the Bipolar-AMI code which solves this problem.

2.3.1 Pseudoternary

rules:

1. 0 = alternating "+" and "-"
2. 1 = 0

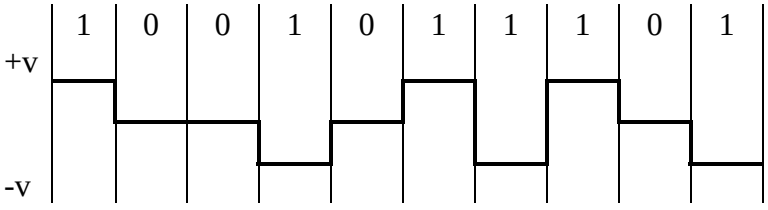


2.3.2 Bipolar-AMI

rules:

1. 0 = 0
2. 1 = alternating "+" and "-"

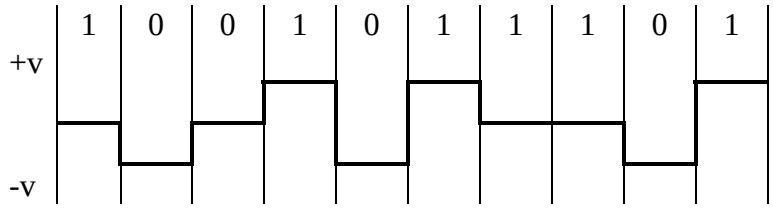
When a "1" is detected which does not have opposite polarity of the last detected "1", a bipolar violation occurs. The Bipolar-AMI, therefore, can be used to detect some errors.



2.3.3 Twinned binary code

rules:

1. 0 ⇨ 1 transmits “+”
2. 1 ⇨ 0 transmits “-”
3. 0 ⇨ 0 transmits “0”
4. 1 ⇨ 1 transmits “0”



3.0 Block Line Codes

Unlike the linear line codes which operate on a stream of information bits, the block line codes operate on a block of information bits. Block line codes are generally used to code multiple information bits (k) into fewer symbols (n). The possibilities for k and n must satisfy the expression: $2^k \leq L^n$, where L is the number of signal levels available. When the equality is not met, the redundancy can be used for error tolerance, timing information or to minimize the running digital sum (RDS) and its associated baseline wander. The running digital sum is a function which sums the digital levels which are transmitted. Each "1" that is transmitted increases the running digital sum by 1 while each "0" that is transmitted, decreases the running digital sum by 1. The other use of block codes is to solve the receiver synchronization problem when the line has no transitions for an excessive period of time due to the transmission of too many zero bits (or one bits). The B8ZS and HDB3 codes are examples codes which prevent the transmission of too many zeros. Block codes do have the disadvantage of requiring framing of the incoming block. Framing will increase the information overhead slightly and will increase the complexity of the receiver and transmitter, but are usually justified by the performance gains.

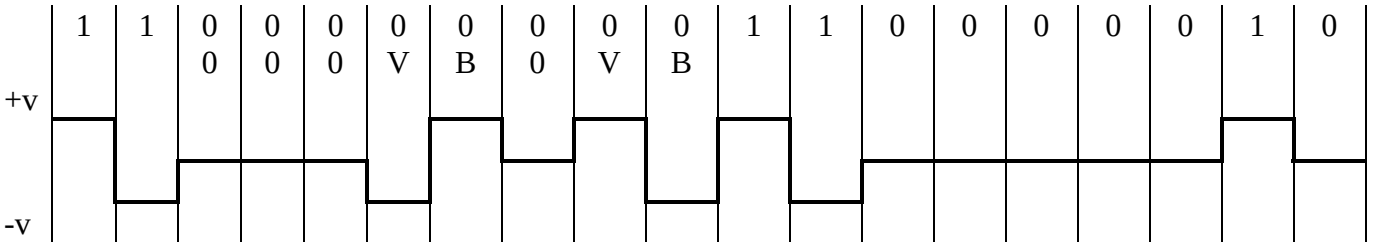
3.1 Binary Block Codes

3.1.1 B8ZS

rules:

1. same as bipolar AMI except that any string of eight zeros is replaced by a string with two bipolar code violations

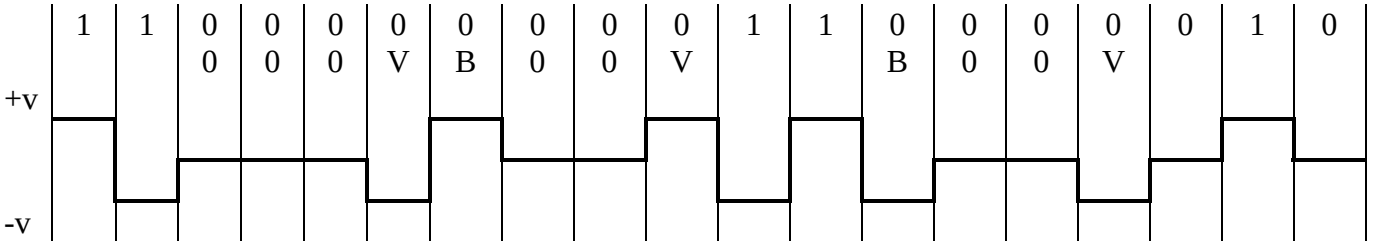
Each "V" below is a bipolar violation. The "B" following each "V" is to cancel the DC component of the bipolar violation



3.1.2 HDB3

rules:

1. same as bipolar AMI except that any string of four zeros is replaced by a string with one code violation



3.2 Multilevel Block Codes

When the medium is bandlimited and higher signaling rates are needed, multi-level block codes are a common solution. With multi-level block codes, the symbol rate is less than the bit rate. Due to the magnitude of the noise in the medium and the signal losses over the length of the medium, the number of discrete signals levels is limited. The greater the number of signal levels, the greater the chance that one symbol could be mistaken for another. For this reason, codes beyond four signal levels are not common except in the most complex transmission systems. In this section, the characteristics of three level codes will be explored.

3.2.1 kBnT codes

Unlike the binary block codes which transmit only on bit per symbol, the kBnT codes are capable of transmitting an average of 1.58 bits per symbol. For the kBnT codes, k is the number of information bits in the block and n is the number of ternary symbols in the code. The notation defines the number of k bits that are encoded into the n available symbols. For three level signals, the values for k and n must satisfy the expression $2^k \leq 3^n$. The table below lists several possibilities for k and n and their efficiencies relative to the rate of 1.58 bits per symbol. Although it would be tempting to implement the 3B2T or the 6B4T codes due to their efficiency, they lack the redundancy that may be required to control the power spectrum or control the density of ones. The 4B3T code is a good compromise between efficiency and redundancy and will be described in more detail.

Efficiencies for k and n values when L (signal levels) = 3

k	n	code	efficiency
1	1	1B1T	63%
3	2	3B2T	95%
4	3	4B3T	84%
6	4	6B4T	95%
7	5	7B5T	89%

When implementing a kBnT code, the user must choose between coding efficiency and redundancy. Efficiency allows more bits to be transmitted for a given bandwidth. Redundancy allows for greater coding gain. The 4B3T is most often used since it is the best compromise between efficiency and redundancy.

3.2.2 Bi-mode 4B3T Coding

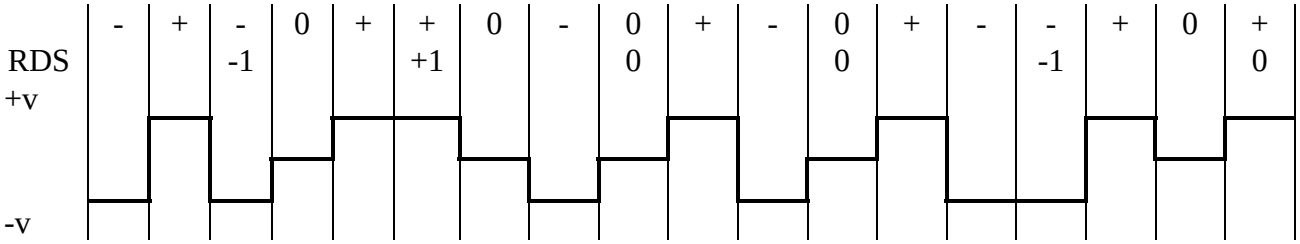
To eliminate a DC component of a bit stream, many methods can be employed. Some of these methods may include scrambling, bit stuffing or bi-mode coding. The scrambling techniques suffer since they must implement multiple memory stages since each output is determined by present and past inputs. Bit stuffing suffers because it increases the number of bits transmitted. Bi-mode coding does not require as much memory as scrambling and does not increase the number of bits transmitted. The three signal levels for the 4B3T will be labeled +, 0 and -. The redundancy in this code will be used to bound the running digital sum between +/- 3.

rules:

1. select symbols from column A if the RDS is between -3 and -1
2. select symbols from column B if the RDS is between 0 and 3

block value	symbol A	symbol B	RDS
0000	+ 0 -	+ 0 -	0
0001	- + 0	- + 0	0
0010	0 - +	0 - +	0
0011	+ - 0	+ - 0	0
0100	+ + 0	- - 0	+/-2
0101	0 + +	0 - -	+/-2
0110	+ 0 +	- 0 -	+/-2
0111	+ + +	- - -	+/-3
1000	+ + -	- - +	+/-1
1001	- + +	+ - -	+/-1
1010	+ - +	- + -	+/-1
1011	+ 0 0	- 0 0	+/-1
1100	0 + 0	0 - 0	+/-1
1101	0 0 +	0 0 -	+/-1
1110	0 + -	0 + -	0
1111	- 0 +	- 0 +	0

For a bit stream of 1010 0101 1100 0011 1001 0110 with an initial RDS of zero, the coded symbols will be the following:



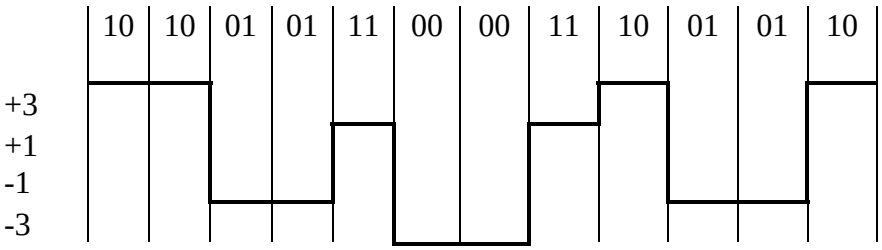
3.2.3 2B1Q (Two binary, one quaternary) Coding

Like the kBnT coding, 2B1Q transmits more than one bit per symbol. Each symbol in 2B1Q contains 2 bits. 2B1Q effectively halves the required bandwidth for a channel. The quaternary symbols are encoded as shown below:

First Bit (sign)	Second Bit (sign)	Quat
1	0	+3 (+2.5V)
1	1	+1 (+5/6V)
0	1	-1 (-5/6V)
0	0	-3 (-2.5V)

The two most common implementations of 2B1Q is the basic rate ISDN-U interface and the HDSL interface. The basic rate ISDN-U interface transmits 160Kbits/second. Most of the energy of the ISDN-U signal is under 40KHz. The HDSL (high bit rate digital subscriber line) transmits 784Kbits/second over twisted pair copper wire. 768Kbits of the transmitted signal is the payload. The remaining 8Kbits are used for overhead and signaling. Most of the energy of the HDSL signal is under 196KHz.

For a scrambled bit stream of 1010 0101 1100 0011 1001 0110, the coded symbols will be the following:



4.0 Error Checking Codes

4.1 Single-Parity Check Code

The parity check coder uses the least number of "check" bits, but is only able to detect bits errors which involve an odd number of bits. Any block with an even number of bits in error will not be detected by this method. The input to the Single-Parity Check coder is the m message bits and the output of the Single-Parity Check coder is an m+p codeword. The extra bit is appended to the original data by a device which implements the generator matrix. The codeword is transmitted to the receiver where it is checked with another device which implements a parity check matrix. The output of the parity check device is a single data bit. This output is called the Single-Parity Check syndrome. A bit value of zero indicates that no errors were detected and a bit value of one indicates that an error has been detected. All the matrix multiplication is performed modulo-two.

The generator matrix for the Single-Parity Check Code
for a 4 bit message block is:

$$G = \begin{vmatrix} 1\,0\,0\,0\,1 \\ 0\,1\,0\,0\,1 \\ 0\,0\,1\,0\,1 \\ 0\,0\,0\,1\,1 \end{vmatrix}$$

The parity-check matrix for the Single-Parity Check Code
for a 4 bit message block is:

$$H = \begin{vmatrix} 1\,1\,1\,1\,1 \end{vmatrix}$$

Example for odd value message block:

let the message block, b = 1 0 0 0

b * G = c

where:

G is the Single-Parity Check generator matrix
c is the resulting Single-Parity Check codeword

The format of the codeword is:

$$\left| \; m_1 \;\; m_2 \;\; m_3 \;\; m_4 \;\; p \; \right|$$

where:

m_1, m_2, m_3, m_4 are the message bits

p is the parity check bit

$$\left| \; 1\,0\,0\,0 \; \right| * \left| \begin{array}{c} 1\,0\,0\,0\,1 \\ 0\,1\,0\,0\,1 \\ 0\,0\,1\,0\,1 \\ 0\,0\,0\,1\,1 \end{array} \right| = \left| \; 1\,0\,0\,0\,1 \; \right|$$

To check the Single-Parity Check codeword, we multiply the codeword by the transpose of the parity check matrix to get the syndrome bit S. A syndrome bit value of zero indicates that no parity errors were detected. We must remember, however, that the data may still be in error but undetectable using this method (an even number of bit errors are not detectable with parity checking). A syndrome bit value of one indicates that a parity error has been detected.

$$c * H^T = S$$

where:

c is the Single-Parity Check codeword

H^T is the transpose of the parity check matrix

S is the syndrome bit

$$\left| \; 1\,0\,0\,0\,1 \; \right| * \left| \begin{array}{c} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{array} \right| = 0$$

The resulting zero indicates that no errors were detected. The parity check bit will be removed from the codeword to get the original message block. Now the same message block will be used but the codeword will have bit two inverted.

bit in error

$$\left| \; 1\,1\,0\,0\,1 \; \right| * \left| \begin{array}{c} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{array} \right| = 1$$

The resulting value of one for the syndrome indicates that an error has been detected.

Example for even value message word:

let the message block, b = 1 0 1 0

$$\begin{vmatrix} 1\ 0\ 1\ 0 \end{vmatrix} * \begin{vmatrix} 1\ 0\ 0\ 0\ 1 \\ 0\ 1\ 0\ 0\ 1 \\ 0\ 0\ 1\ 0\ 1 \\ 0\ 0\ 0\ 1\ 1 \end{vmatrix} = \begin{vmatrix} 1\ 0\ 1\ 0\ 0 \end{vmatrix}$$

Generating the syndrome bit as before:

$$\begin{vmatrix} 1\ 0\ 1\ 0\ 0 \end{vmatrix} * \begin{vmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{vmatrix} = 0$$

The resulting zero indicates that no errors were detected. The parity check bit will be removed from the codeword to get the original message block. Now the same message block will be used but the codeword will have bit three inverted.

bit in error

$$\begin{vmatrix} 1\ 0\ 0\ 0\ 0 \end{vmatrix} * \begin{vmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{vmatrix} = 1$$

The resulting value of one for the syndrome indicates that an error has been detected.

4.2 Checksum Codes

One popular method for error detection is the use of checksums. Checksums have the advantage of operating on a long sequence of data. Typically the data is 8 bit character data (or ASCII data) in a file or character frame. Each character byte is summed into a checksum variable. As the data is summed into the checksum variable, the carry's are ignored. The checksum variable is usually appended to the end of the data field and is called the checksum field. The checksum field can be one or more bytes. The ability to detect errors in a multibyte sequence is affected by the number of bytes in the sequence and the size of the checksum field. Four different types of checksums will be described: the single precision checksum, the double precision checksum, the Honeywell checksum and the residue checksum.

4.2.1 Single-Precision Checksum

For a single-precision checksum, each byte of the data is summed into a single byte. The example below shows how a single precision checksum is calculated. The limitation of the single precision checksum is its ability to detect stuck at one (SA1) failures in the most significant bit of the input data. Since the SA1 in the MSB of the checksum will be discarded often in the carry operation, our ability to detect SA1 is poor.

Transmitted Data:

Byte 1	0	0	1	0	1	1	0	0	2CH
Byte 2	0	1	0	1	0	0	1	1	53H
Byte 3	1	0	0	1	0	1	1	1	97H
Byte 4	1	1	0	1	0	1	0	0	D4H

Carry Ignored ⇩

Computed Checksum

1	1	1	0	1	0	1	0	EAh

Transmitted Block:

⇦ Data ⇨				Checksum
2C	53	97	D4	EA

4.2.2 Double-Precision Checksum

For a double-precision checksum, each location of the data of n-bits is summed into a location of 2n-bits. For the single precision example, a double precision checksum would be 16 bits (2 bytes). The example below shows how a double precision checksum is calculated for 8 bit data. If the data was 16 bit data, the double precision checksum would be 32 bits (4 bytes). The problem with the detection of SA1 faults can be detected using the double precision checksum. Carry's from the low order byte of the checksum is summed into the high order byte of the checksum. Carry's from the high order byte of the checksum are ignored.

Transmitted Data:

Byte 1	5A
Byte 2	EF
Byte 3	24
Byte 4	C5

⇩ ⇩

Computed Checksum

02	32

Transmitted Block:

⇦ Data ⇨				⇦ Checksum ⇨	
5A	EF	24	C5	02	32

4.2.3 Honeywell Checksum

The Honeywell Checksum is an alternative to the double precision checksum. The data is still summed into a location which is 2n bits in length where n is the size of the data word. Unlike the double precision checksum, the Honeywell checksum interleaves the data bytes into double length words before the addition to the checksum location. Carry's from the high order byte of the checksum are ignored. The Honeywell checksum can find SA1 and SA0 errors which occur in the same bit positions of all the data words. The example below shows how the Honeywell checksum is calculated for 8 bit data.

Transmitted Data:

Byte 1	C3
Byte 2	FE
Byte 3	DB
Byte 4	B4

⇩ ⇩

Interleaved Data

FE	C3
B4	DB

⇩ ⇩

Computed Checksum

B3	9E

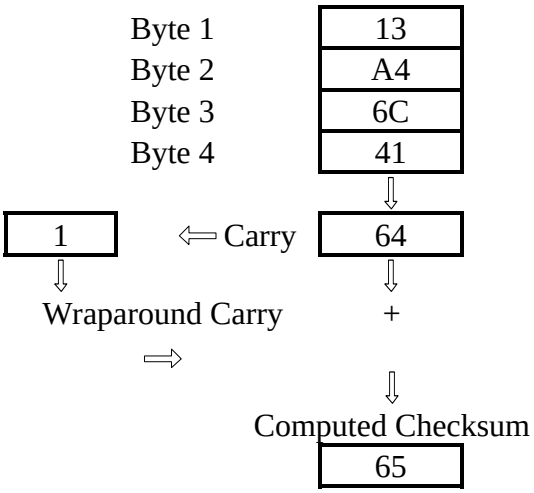
Transmitted Block:

⇦ Data ⇨				⇦ Checksum ⇨	
C3	FE	DB	B4	B3	9E

4.2.4 Residue Checksum

The residue checksum is a variation of the single precision checksum which solves the problem of the undetectable SA1 errors in the MSB of the data word. The residue checksum takes the carry out from the MSB of the checksum and adds it to the LSB of the checksum. The example below shows how a residue checksum is calculated for 8 bit data.

Transmitted Data:



Transmitted Block:

⇦ Data ⇨				Checksum
13	A4	6C	41	65

4.3 CRC Codes

A better method to detect errors in a large block of data involves the use of CRC codes. The hardware required to implement a CRC code is slightly more complicated than other error detection codes but it has an error detection effectiveness greater than 99.9%. The CRC is calculated by dividing the message block by a generator polynomial. The quotient is discarded and the remainder is appended to the end of the message block for transmission. CRC hardware is implemented using multi-stage shift registers so the length of the message block does not affect the operation of the CRC generation hardware. At the receiving end, the transmitted block with the CRC is divided by the same generator polynomial. Once again the quotient is discarded and the remainder is checked to determine if any errors occurred. A remainder of zero indicates that no errors were detected. Since the CRC is only an error detecting code, the position of an error in the received message can not be determined. CRC codes would be used in communication protocols that use automatic repeat request (ARQ). The four CRC codes described in this paper include the CRC-12, the CRC-16, the CRC-CCITT and the CRC-32.

The operation of all the CRC codes is identical. To explain the CRC generation process, a short simple message block and a small CRC generator polynomial will be used. The CRC performs its arithmetic on binary data which has been represented as a polynomial. Each bit position of the message block is represented as a coefficient of a polynomial. The general form of the polynomial is:

$$M(x) = b_nX^n + b_{n-1}X^{n-1} + b_{n-2}X^{n-2} + \ldots + b_2X^2 + b_1X + b_01$$

where:

b_n = the value of the message block at bit position n (0 or 1)

The degree of the polynomial will be one less than the total number of bits in the message block.

The message block "101001101" has 9 bits (k = 9). The polynomial representation for this message block will have a degree of 8. With the LSB to the right, the polynomial representation for the message block is:

$$M(x) = 1\bullet X^8 + 0\bullet X^7 + 1\bullet X^6 + 0\bullet X^5 + 0\bullet X^4 + 1\bullet X^3 + 1\bullet X^2 + 0\bullet X + 1\bullet 1$$

After simplification, the polynomial for the message block is:

$$M(x) = X^8 + X^6 + X^3 + X^2 + 1$$

For the generator polynomial we will use:

$G(x) = X^5 + X^2 + X + 1$

$= 1\ 0\ 0\ 1\ 1\ 1$

Since the degree of the generator polynomial is 5, the number of bits in the generated CRC will be 5. The total number of bits transmitted, therefore, will be 14 (n = 14).

The calculation of the CRC for transmission proceeds as follows:

1. Multiply the message polynomial by X^{n-k}. In this example the value will be $X^{14-9} = X^5$.

$$X^{n-k} \bullet [M(x)] = X^5 \bullet [X^8 + X^6 + X^3 + X^2 + 1]$$

$$= X^{13} + X^{11} + X^8 + X^7 + X^5$$

$$= 1\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0$$

2. Divide $X^{n-k} \bullet [M(x)]$ by the generator polynomial G(x) and discard the quotient. The remainder is the CRC or the Block Check Character (BCC) which will be called B(x).

```
                    1 0 1 1 1 1 1 1 0   <=  discard quotient
  1 0 0 1 1 1 | 1 0 1 0 0 1 1 0 1 0 0 0 0 0
                1 0 0 1 1 1
                ------------
                  1 1 1 0 1 0
                  1 0 0 1 1 1
                  -----------
                    1 1 1 0 1 1
                    1 0 0 1 1 1
                    -----------
                      1 1 1 0 0 0
                      1 0 0 1 1 1
                      -----------
                        1 1 1 1 1 0
                        1 0 0 1 1 1
                        -----------
                          1 1 0 0 1 0
                          1 0 0 1 1 1
                          -----------
                            1 0 1 0 1 0
                            1 0 0 1 1 1
                            -----------
                              1 1 0 1 0   <=  remainder (BBC), B(x)
```

3. Append the CRC, B(x), to the message block, M(x), to create the transmitted message block, T(x).

$$T(x) = X^{n-k} \bullet [M(x)] + B(x)$$

```
1 0 1 0 0 1 1 0 1 0 0 0 0 0   ⇐   X^(n-k)[M(x)]
+                 1 1 0 1 0   ⇐   B(x)
-----------------------------
1 0 1 0 0 1 1 0 1 1 1 0 1 0   ⇐   T(x)
```

Checking the transmitted block at the receiving end proceeds as follows:

1. Divide T(x) by the generator polynomial G(x) and discard the quotient. A remainder of zero indicates that the block was received without error.

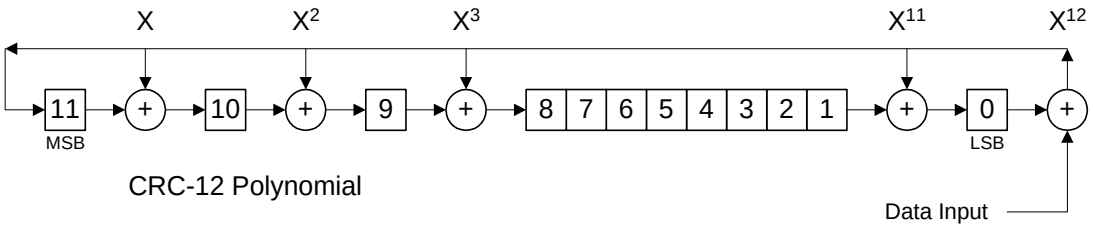
```
                  1 0 1 1 1 1 1 1 0   ⇐  discard quotient
1 0 0 1 1 1 | 1 0 1 0 0 1 1 0 1 1 1 0 1 0
              1 0 0 1 1 1
              -----------
                1 1 1 0 1 0
                1 0 0 1 1 1
                -----------
                  1 1 1 0 1 1
                  1 0 0 1 1 1
                  -----------
                    1 1 1 0 0 1
                    1 0 0 1 1 1
                    -----------
                      1 1 1 1 0 1
                      1 0 0 1 1 1
                      -----------
                        1 1 0 1 0 0
                        1 0 0 1 1 1
                        -----------
                          1 0 0 1 1 1
                          1 0 0 1 1 1
                          -----------
                                0     ⇐  remainder equals zero (no errors)
```

4.3.1 CRC-12

The CRC-12 polynomial is used for block containing characters that are six bits in length. The generator polynomial for the CRC-12 is:

$$G(x) = X^{12} + X^{11} + X^3 + X^2 + X + 1$$

The circuit which implements this polynomial is shown below. The block of data is shifted into the data input line. After all the data has been shifted in, the remainder will be stored in the registers 0 through 11.

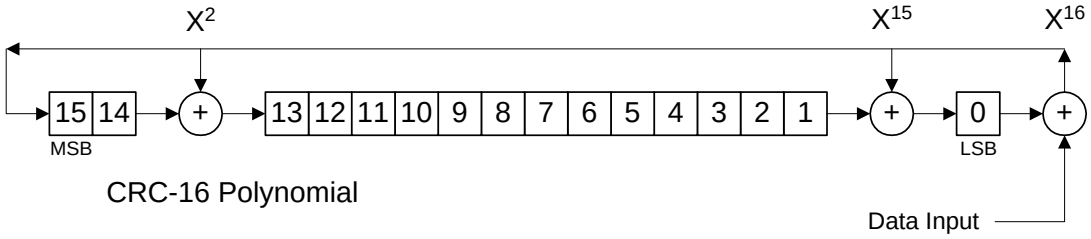


4.3.2 CRC-16

The CRC-16 polynomial is used for block containing characters that are eight bits in length. The generator polynomial for the CRC-16 is:

$$G(x) = X^{16} + X^{15} + X^2 + 1$$

The circuit which implements this polynomial is shown below. The block of data is shifted into the data input line. After all the data has been shifted in, the remainder will be stored in the registers 0 through 15.

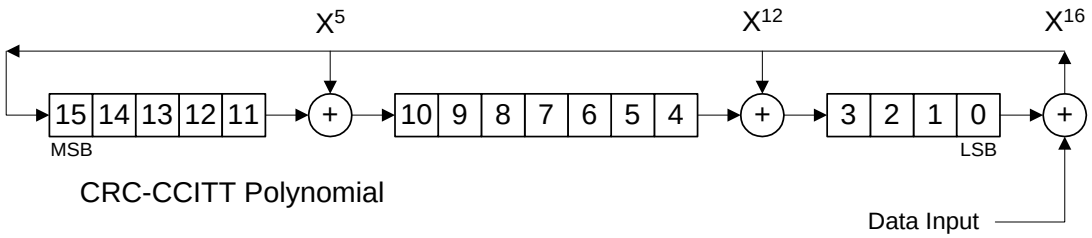


4.3.3 CRC-CCITT

The CRC-CCITT polynomial is used for block containing characters that are eight bits in length. The generator polynomial for the CRC-CCITT is:

$$G(x) = X^{16} + X^{12} + X^5 + 1$$

The circuit which implements this polynomial is shown below. The block of data is shifted into the data input line. After all the data has been shifted in, the remainder will be stored in the registers 0 through 15.

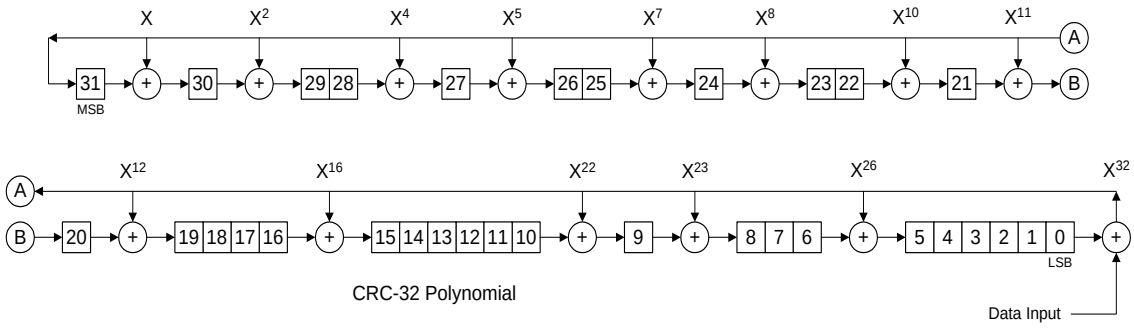


4.3.4 CRC-32

The CRC-32 polynomial is used for block containing characters that are sixteen bits in length. The generator polynomial for the CRC-32 is:

$$G(x) = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$

The circuit which implements this polynomial is shown below. The block of data is shifted into the data input line. After all the data has been shifted in, the remainder will be stored in the registers 0 through 31.



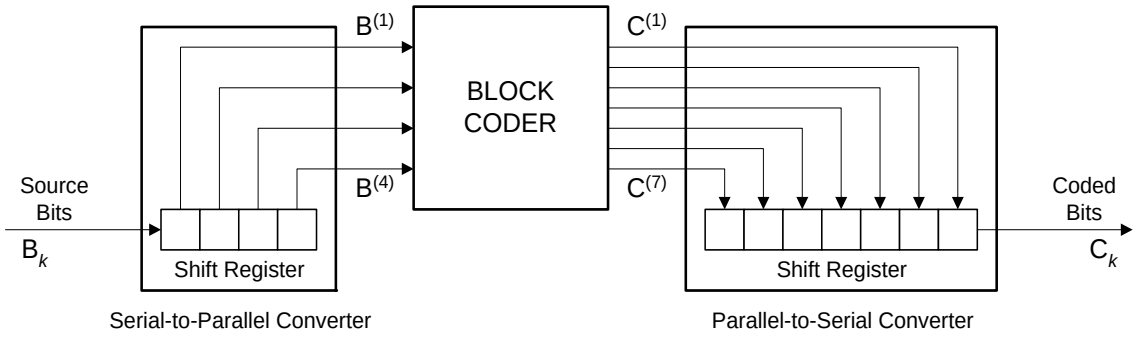
5.0 Memoryless Error Checking and Correcting Block Codes

One of the best uses for the redundancy in a multi-level code is to implement error checking and correcting. Like other error checking methods, the check bits are transmitted along with the message bits. The ability to perform error correction at the receiving end without retransmission is called forward error correction (FEC). FEC is one of the best methods when the communication is simplex or broadcasted to many users. Two popular methods to check and correct errors using block codes are Parity Check coding and Hamming coding. These techniques are called "memoryless" since the output of the block coder is only dependent on the present state of its inputs. With each error checking and correcting code we have a specification called the Hamming distance. The Hamming distance determines the maximum number of bits in error that can be detected in a block and the maximum number of bits that can be corrected in a block. The formulas below show the relationship between the Hamming distance and the number of errored bits that can be detected and corrected.

maximum number of detectable bits in error is $\leq H - 1$

maximum number of correctable bits in error is $\leq (H - 1)/2$

The block coder block diagram for (7,4) coding is shown below:



5.1 Parity Check Code

The single parity check coding method can be extended in a manner which allows for errors to be detected and corrected. The number of parity (check) bits increases but the ability to detected up to two bits in error and correct one bit is achieved. The figure below illustrates the method employed to implement multiple parity check bits.

m_1	m_2	m_3	c_1
m_4	m_5	m_6	c_2
m_7	m_8	m_9	c_3
c_4	c_5	c_6	

In this example, the bits m_1 - m_9 are the 9 original message bits and bits c_1 - c_6 are the 6 check bits. The even parity, the check bits c_1 - c_6 are encoded as follows:

$c_1 = m_1 \oplus m_2 \oplus m_3$
$c_2 = m_4 \oplus m_5 \oplus m_6$
$c_3 = m_7 \oplus m_8 \oplus m_9$
$c_4 = m_1 \oplus m_4 \oplus m_7$
$c_5 = m_2 \oplus m_5 \oplus m_8$
$c_6 = m_3 \oplus m_6 \oplus m_9$

where $\oplus$ is the modulo 2 addition operator (exclusive OR)

The codeword transmitted could have the form:

$m_1\ m_2\ m_3\ m_4\ m_5\ m_6\ m_7\ m_8\ m_9\ c_1\ c_2\ c_3\ c_4\ c_5\ c_6$

The position of the check bits can be placed anywhere in the codeword. For simplicity, they are shown here clustered together at the end of the codeword. In practice, the check bits would be separated in the codeword to ensure that a noise burst would not destroy more than one of the check bits.

5.2 Hamming Codes

Hamming coders are preferred over the Single-Parity Check coders when detection as well as correction is desired. Hamming codes are capable of detecting up to two bits in error and have the ability to correct single bit errors. Hamming codes come in several varieties based on the number of message bits (m) and check bits (c) in its output. All Hamming codes are classified as linear block codes. The input to the Hamming coder is the m message bits and the output of the Hamming coder is an m+c codeword. The extra bits are appended to the original data by a device which implements the generator matrix. The codeword is transmitted to the receiver where it is checked with another device which implements a parity check matrix. The output of the parity check device is a string of data bits with the same number of bits as the check bits. This output is called the Hamming syndrome and, through a look-up table, a single bit fault (if it exists) can be identified and corrected. All the matrix multiplication is performed modulo-two. Hamming codes are identified by a set of ordered pairs which indicate their number of message bits and the number of bits in the codeword. For an (n,k) Hamming coder, k is the number of input bits and n is the number of output bits. The number of Hamming bits is determined by the number of data bits. The equation which determines the number of Hamming bits is shown below.

$$2^{n-k} \geq n + 1$$

where:

k = number of data bits
n = total number of bits transmitted in a message stream

All Hamming codes have a minimum Hamming distance of 3 regardless of the number of check bits. A Hamming code, therefore, can be used for single error correction or double error detection. The table below shows the relative efficiency of three sizes of Hamming codes along with the polynomial coefficients. The greater the R_c value (the code rate), the greater the efficiency.

n (codeword size)	k (block size)	R_c (code rate)	G(p) (polynomial)
7	4	0.57	1 011
15	11	0.73	10 011
31	26	0.84	100 101

The generator matrix for encoding the (7,4) Hamming codeword is:

$G = [\, I \mid P \,]$

where:

I = Identity Matrix
P = Submatrix

$$G = \left|\begin{array}{cc} 1\,0\,0\,0 & 1\,0\,1 \\ 0\,1\,0\,0 & 1\,1\,1 \\ 0\,0\,1\,0 & 1\,1\,0 \\ 0\,0\,0\,1 & 0\,1\,1 \end{array}\right|$$

The parity-check matrix for determining the syndrome value for the (7,4) Hamming codeword is:

$$H = \left|\, P^T \,\middle|\, I \,\right|$$

where:

P^T = The transpose of the Submatrix of the generator matrix
I = Identity Matrix

$$H = \left|\begin{array}{cc} 1\,1\,1\,0 & 1\,0\,0 \\ 0\,1\,1\,1 & 0\,1\,0 \\ 1\,1\,0\,1 & 0\,0\,1 \end{array}\right|$$

The syndrome translation table for the (7,4) Hamming code is:

Syndrome Bits (S)			Bit in error (E)						
0	1	2	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0
1	1	1	0	1	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0
0	1	1	0	0	0	1	0	0	0
1	0	0	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	1

Example of Hamming Code:

let the message block, m = 1 0 1 0

$m * G = c$

where:
G is the Hamming generator matrix
c is the resulting Hamming codeword

The format of the codeword is:

$$\left| m_1 \;\; m_2 \;\; m_3 \;\; m_4 \;\; c_1 \;\; c_2 \;\; c_3 \right|$$

where:
m_1, m_2, m_3, m_4 are the message bits
c_1, c_2, c_3 are the check bits

$$\left| 1\ 0\ 1\ 0 \right| * \left| \begin{array}{c} 1\ 0\ 0\ 0\ 1\ 0\ 1 \\ 0\ 1\ 0\ 0\ 1\ 1\ 1 \\ 0\ 0\ 1\ 0\ 1\ 1\ 0 \\ 0\ 0\ 0\ 1\ 0\ 1\ 1 \end{array} \right| = \left| 1\ 0\ 1\ 0\ 0\ 1\ 1 \right|$$

To check Hamming codeword we multiply the codeword by the transpose of the Hamming parity check matrix to get the syndrome matrix S.

$c * H^T = S$

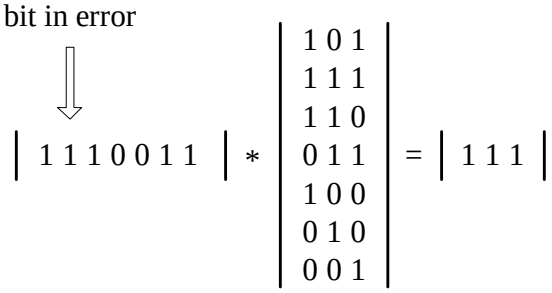
where:
c is the Hamming codeword
H^T is the transpose of the Hamming parity check matrix
S is the syndrome matrix

$$\left| 1\ 0\ 1\ 0\ 0\ 1\ 1 \right| * \left| \begin{array}{c} 1\ 0\ 1 \\ 1\ 1\ 1 \\ 1\ 1\ 0 \\ 0\ 1\ 1 \\ 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{array} \right| = \left| 0\ 0\ 0 \right|$$

Since the syndrome bits = 0, no errors have been detected. The check bits will be removed from the codeword to get the original message block. Now the same message

(1010) will be used but the Hamming codeword will have the second bit from the left inverted.

bit in error



$$\begin{vmatrix} 1\,1\,1\,0\,0\,1\,1 \end{vmatrix} * \begin{vmatrix} 1\,0\,1 \\ 1\,1\,1 \\ 1\,1\,0 \\ 0\,1\,1 \\ 1\,0\,0 \\ 0\,1\,0 \\ 0\,0\,1 \end{vmatrix} = \begin{vmatrix} 1\,1\,1 \end{vmatrix}$$

Checking the syndrome bits against the syndrome translation table confirms the reversal of the second bit.

6.0 Error Checking and Correcting Block Codes with Memory

One of the main disadvantages of Hamming codes is the fixed Hamming distance and the difficulty of implementing coders for large blocks. The fixed Hamming distance of Hamming codes allows for detection of two errored bits and the ability to correct only a single errored bit. A code which allows the implementor to choose the desired Hamming distance would be beneficial. The largest practical Hamming coder allows for codewords of only 31 bits. Because of these limitations, cyclic block coders are preferred when selected Hamming distances (redundancy) are or long block lengths are needed. The type of cyclic block coder that will be described are BCH coders. BCH codes were developed by R.C. Bose, D.K. Ray-Chaudhuri and A. Hocquenghem.

6.1 The general form for the polynomial for implementing the BCH codes is shown below.

$$G(x) = (x + D^{b+1}) (x + D^{b+2}) (x + D^{b+3})... (x + D^{b+2t}), P(x)$$

where:

t = number of errors to correct per block
b = 0 for narrow-sense BCH codes ≠ 0 for wide-sense BCH codes
$D^{(\ )}$ = roots of the polynomial
P(x) = additional polynomial that ensures that all coefficients in G(x) are 1 or 0

The methods to determine b, $D^{(\ )}$ and P(x) are beyond the scope of this paper. A simple procedure using a table driven approach can be used for a small codeword size. The procedure to generate the BCH polynomial G(x) is given below.

1. Select values for n (codeword length) and m where the following equality is satisfied:

$$n = 2^m - 1$$

2. Select the desired number of errored bits to correct in each block of n bits. The distance of the code is:

$$d = 2t + 1$$

where:

t = number of errored bits to correct
d = code distance

3. Select an integer j between 1 and $2^m - 2$ where j and $2^m - 1$ have no factors in common. A reasonable default value for j is 1.

4. For the values of t and j selected earlier, find the first t odd multiples of j. When one of the odd multiples exceeds $2^m - 2$, reduce the value modulo $2^m - 1$. The resulting generator polynomial will have a root D^i for each value of i = j, 3j, 5j...(2t-1)j.

5. Use the table to find the minimal polynomial for each root of D^y.

6. Multiply all the unique minimal polynomials. (a minimal polynomial may have more than one D^y root.

6.2 Example of a generator polynomial for a BCH code

Design Requirements:

codeword length = 15
number of errors to correct = 3

Solution:

1. Using the equation: $n = 2^m - 1$, set n = 15 and solve for m

$15 = 2^m - 1$

$m = 4$

2. The valid values for j that lie between 1 and $2^m - 2 = 14$ that are not factors of $2^m - 1$ are (1, 2, 4, 7, 8, 11, 13, 14) . The factors of $2^m - 1 = 15$ are 3 and 5 and are therefore excluded from the list of possible j values. For this example let j = 1.

3. Since t = 3, the first 3 odd multiples of j are 1*j = 1, 3*j = 3 and 5*j = 5.

4. Using Table 1, the minimal polynomials are:

$M^{(1)}(x) = x^4 + x + 1$

$M^{(3)}(x) = x^4 + x^3 + x^2 + x + 1$

$M^{(5)}(x) = x^2 + x + 1$

5. Now multiply $M^{(1)}(x)$, $M^{(2)}(x)$ and $M^{(3)}(x)$ together to get:

$G(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$

6. The total number of bits in the codeword is given by the following equation:

$$n = k + c$$

where:

n = number of bits in codeword
k = number of information bits
c = number of check bits

Since G(x) has a degree of 10, the number of information bits is $15 = k + 10 \Rightarrow k = 5$.

Table 1 - 4th order Minimal Polynomials

i	minimal polynomial $M^{(i)}(x)$
1	$x^4 + x + 1$
3	$x^4 + x^3 + x^2 + x + 1$
5	$x^2 + x + 1$
7	$x^4 + x^3 + 1$

The Generator Polynomial is used to generate the check bits in the same manner as was done for the CRC codes. On the receive side, the received codeword will have the form:

$$R(x) = C(x) + E(x)$$

where:

$R(x)$ = the received codeword
$C(x)$ = the transmitted codeword
$E(x)$ = the error polynomial

When $E(x) = 0$, no errors are present. When errors are present, the bits in error can be determined by solving a system of polynomials which correspond to the number of possible error syndromes.

7.0 Convolutional Coding

Convolutional coding, unlike the block coding, uses a finite memory system to generate the redundant codes. Convolutional coding is more popular than block coding because it is simple to implement and their performance matches or exceeds that of block codes. The input to the Convolutional coder is the m message bits and the output of the Convolutional coder is the n data bits. The convolutional coder is called an m/n coder. The new bits are created by a device which implements the generator matrix. The encoded data is transmitted to the receiver where it is checked with another device which implements a parity check matrix. The output of the parity check device is a single data bit. This output is called the syndrome. A bit value of zero indicates that no errors were detected and a bit value of one indicates that an error has been detected. All the matrix multiplication is performed modulo-two. Using more complicated checkers and decoders will allow the detection and correction of faulty data bits.

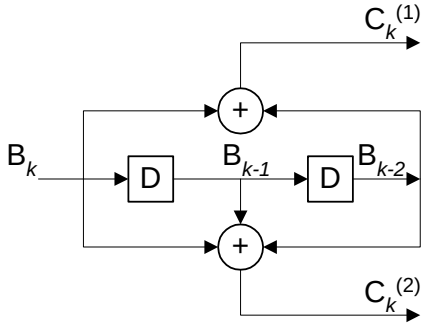
7.1 The 1/2 Convolutional Coder

The 1/2 convolutional coder is the simplest convolutional coder to implement. This coder inputs a single bit (B_k) and outputs two bits ($C_k^{(1)}$ and $C_k^{(2)}$).

7.1.1 Coder for the 1/2 convolutional code

Generator Matrix is $G(D) = [\ (1+D^2)\ \ (1+D+D^2)\ ]$

The circuit which implements this coder polynomial is shown below:



Example for 1/2 convolutional coder/decoder:

Bit sequence for B_k is 11011010

With initial conditions of $B_{k-1} = 0$, the state table which shows the movement of data through the coder is shown on the next page. The two bits ($C_k^{(1)}$ and $C_k^{(2)}$) are the outputs of the coder.

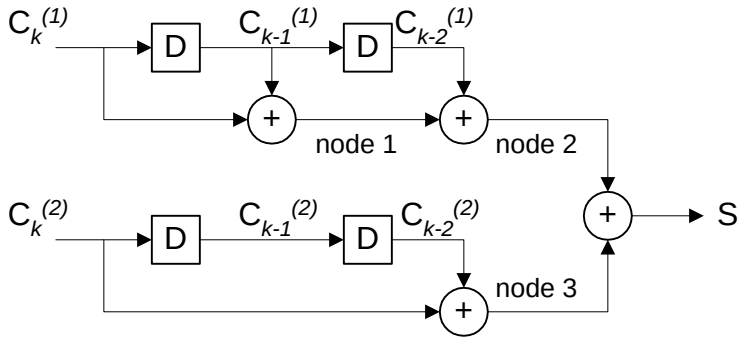
State Table for the 1/2 convolutional coder

B_k	B_{k-1}	B_{k-2}	$C_k^{(1)}$	$C_k^{(2)}$
0	0	0	0	0
1	0	0	1	1
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0
0	1	1	1	0
1	0	1	0	0
1	1	0	1	0

7.1.2 Checker for the 1/2 convolutional code

Parity Check Matrix is $H(D) = [\ (1+D+D^2)\ \ (1+D^2)\ ]$

The circuit which implements this parity checker polynomial is shown below:



For the generated $C_k^{(1)}$ and $C_k^{(2)}$ bits above and with initial conditions of $C_{k-2} = C_{k-1} = C_k = 0$, the state table which shows the movement of data through the decoder is shown below. The S (syndrome) bit is the output of the decoder.

Parity Checker State Table (no data errors)

$C_k^{(1)}$	$C_{k-1}^{(1)}$	$C_{k-2}^{(1)}$	$C_k^{(2)}$	$C_{k-1}^{(2)}$	$C_{k-2}^{(2)}$	S	node 1	node 2	node 3
0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	1	1	1
0	0	1	0	1	1	0	0	1	1
1	0	0	0	0	1	0	1	1	1
1	1	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	1	0	0
1	0	1	0	0	0	0	1	0	0

All zeros in the syndrome bit column of the state table indicates that no errors were detected.

Now the same data will be used but the second bit of $C_k^{(1)}$ will be inverted.

Parity Checker State Table (with data error)

$C_k^{(1)}$	$C_{k-1}^{(1)}$	$C_{k-2}^{(1)}$	$C_k^{(2)}$	$C_{k-1}^{(2)}$	$C_{k-2}^{(2)}$	S	node 1	node 2	node 3
0	0	0	0	0	0	0	0	0	0
0⇐	0	0	1	0	0	1	0	0	1
0	0⇐	0	1	1	0	1	0	0	1
0	0	0⇐	0	1	1	1	0	0	1
1	0	0	0	0	1	0	1	1	1
1	1	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	1	0	0
1	0	1	0	0	0	0	1	0	0

The one values for the syndrome bit indicate that an error was detected starting at the second set of bits in the bit stream. Since the output of the decoder would be a single bit, the second bit in the bit stream could be reversed to correct the error

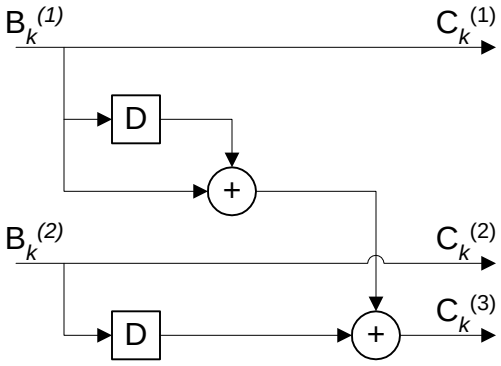
7.2 The 2/3 convolutional code

The 2/3 convolutional coder is also easy to implement and provides three output bits ($C_k^{(1)}$, $C_k^{(2)}$and $C_k^{(3)}$) for two input bits ($B_k^{(1)}$ and $B_k^{(2)}$).

7.2.1 Coder for the 2/3 convolutional code:

Generator Matrix is G(D) = $\begin{matrix} 1 & 0 & (1+D) \\ 0 & 1 & D \end{matrix}$

The circuit which implements this coder polynomial is shown below:



Example for the 2/3 convolutional coder/decoder:

Bit sequence for $B_k^{(1)}$ is 11011010
Bit sequence for $B_k^{(2)}$ is 10100101

With initial conditions of $B_{k-1}^{(1)} = B_{k-1}^{(2)} = 0$

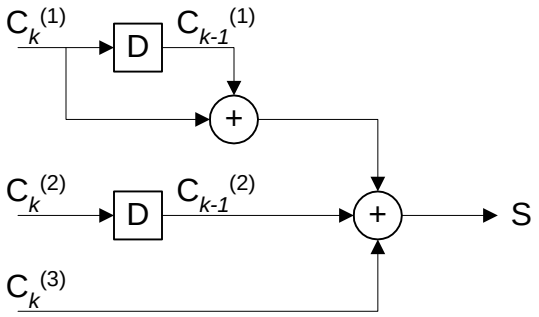
State Table for the 1/2 convolutional coder

$B_k^{(1)}$	$B_{k-1}^{(1)}$	$B_k^{(2)}$	$B_{k-1}^{(2)}$	$C_k^{(1)}$	$C_k^{(2)}$	$C_k^{(3)}$	node 1
0	0	1	0	0	1	0	0
1	0	0	1	1	0	0	1
0	1	1	0	0	1	1	1
1	0	0	1	1	0	0	1
1	1	0	0	1	0	0	0
0	1	1	0	0	1	1	1
1	0	0	1	1	0	0	1
1	1	1	0	1	1	0	0

7.2.2 Checker for the 2/3 convolutional code

Parity Check Matrix is H(D) = [(1+ D) D 1]

The circuit which implements this parity checker polynomial is shown below:



Parity Checker State Table (no data errors)

$C_k^{(1)}$	$C_{k-1}^{(1)}$	$C_k^{(2)}$	$C_{k-1}^{(2)}$	$C_k^{(3)}$	S	node 1
0	0	1	0	0	0	0
1	0	0	1	0	0	1
0	1	1	0	1	0	1
1	0	0	1	0	0	1
1	1	0	0	0	0	0
0	1	1	0	1	0	1
1	0	0	1	0	0	1
1	1	1	0	0	0	0

The zeros for all values of the syndrome bit S indicates that no errors were detected.

Now the same data will be used but the second bit of $C_k^{(1)}$ will be inverted.

Parity Checker State Table (with data error)

$C_k^{(1)}$	$C_{k-1}^{(1)}$	$C_k^{(2)}$	$C_{k-1}^{(2)}$	$C_k^{(3)}$	S	node 1
0	0	1	0	0	0	0
0⇦	0	0	1	0	1	0
0	0⇦	1	0	1	1	0
1	0	0	1	0	0	1
1	1	0	0	0	0	0
0	1	1	0	1	0	1
1	0	0	1	0	0	1
1	1	1	0	0	0	0

The one values for the syndrome bit indicate that an error was detected starting at the second set of bits in the bit stream.

8.0 Summary

This paper has demonstrated many data coding techniques. The coding technique chosen for a particular application is determined by the performance requirements of the system. Short links with high signal to noise ratio can use simple linear line codes such as RZ, NRZ, AMI or Manchester. When the link is longer and affected by noise, more advanced coding techniques are employed. Redundancy can be added to the line code to provide coding gain. The coding gain improves system performance. Line codes which add redundancy include multilevel line codes, bipolar violation insertion codes and codes which maintain a bounded RDS. For more advanced systems, error checking codes may be used when the system also uses one of the ARQ protocols. Error checking codes include parity checking, checksums and CRCs. When an ARQ protocol is not appropriate due to the nature of the system, FEC techniques are used. FEC techniques include Hamming coders, BCH coders and convolutional coders. In all these cases, the redundancy resulting from coding allows the system to tolerate a lower signal to noise ratio for a fixed data rate. The ability to tolerate a lower signal to noise ratio allows the system to maintain a fixed data rate with fewer errors or increase the data rate for a fixed number of errors.

Appendix A - Review of Matrix Arithmetic

A matrix is a rectangular array of numbers. The dimensions of a matrix are defined by the number of rows and columns. A matrix with 2 rows and 3 columns is called a 2x3 matrix. A square matrix has the same number of rows and columns.

Matrix Addition and Subtraction:

1. The dimensions of the two matrices must be the same

2. For addition (and subtraction), corresponding elements are added (or subtracted).

3. The dimensions of the resulting matrix are the same as the two operand matrices.

4. Example:

$$\begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{vmatrix} + \begin{vmatrix} 7 & 8 & 9 \\ 1 & 4 & 7 \end{vmatrix} = \begin{vmatrix} 8 & 10 & 12 \\ 5 & 9 & 13 \end{vmatrix}$$

Matrix Multiplication:

1. For matrix A and matrix B where the product is A * B, the dimensions of the two matrices do not need to be the same but must satisfy the requirement that the number of columns in matrix A must equal the number of rows in matrix B.

2. The product A * B = B * A is only possible for square matrices.

3. For a product of two matrices A * B, the dimensions of the product matrix is the number of rows from the A matrix and the number of columns from the B matrix. For matrix A, an l x m matrix, and B, an m x n matrix, the result will be an l x n matrix.

4. The method of multiplying two matrices is given below:

A * B = AB

$$\begin{vmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \end{vmatrix} * \begin{vmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \\ B_{31} & B_{32} \end{vmatrix} = \begin{vmatrix} (A_{11}*B_{11} + A_{12}*B_{21} + A_{13}*B_{31}) & (A_{11}*B_{12} + A_{12}*B_{22} + A_{13}*B_{32}) \\ (A_{21}*B_{11} + A_{22}*B_{21} + A_{23}*B_{31}) & (A_{21}*B_{12} + A_{22}*B_{22} + A_{23}*B_{32}) \end{vmatrix}$$

Transpose of a Matrix:

1. The transpose of an m x n matrix A (denoted by A^T) is defined to be the n x m matrix whose first column is the first row of A, whose second column is the second row of A, and so on.

2. Example of a transpose of an array:

$$A = \begin{vmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \end{vmatrix}$$

$$A^T = \begin{vmatrix} A_{11} & A_{21} \\ A_{12} & A_{22} \\ A_{13} & A_{23} \end{vmatrix}$$

References

1. Carlson, A. Bruce, Communication Systems, McGraw-Hill, Inc. 1986, Chapter 13

2. Freeman, Roger L., Telecommunication Transmission Handbook, John Wiley and Sons, Inc., New York, Chapter 12

3. Hioki, Warren, Telecommunications, Prentice Hall, Inc., New Jersey, 1995, Chapter 12

4. Lee, Edward A. and Messerschmitt, David G., Digital Communications, Kluwer Academic Publishers, Boston, 1994, Chapters 12-14

5. Reeve, Whitham D., Subscriber Loop Signaling and Transmission Handbook, The Institute of Electrical and Electronics Engineers, Inc., New York, 1995, Chapter 3 & 5

6. Rorabaugh, C. Britton, Error Coding Cookbook, McGraw-Hill, New York, 1996, Chapter 3 & 4

7. Stallings, William, Data and Computer Communications, Macmillan Publishing Company, 1991, Chapter 3