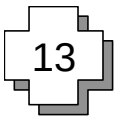

Error Coding (Detection)

13.1 Introduction

The most important measure of error detection is the Hamming distance. This defines the number of changes in the transmitted bits that are required in order for a code word to be received as another code word. The more bits that are added, the greater the Hamming distance can be, and the objective of a good error detecting code is to be able to maximize the minimum Hamming distance between codes. For example, a code which has a minimum Hamming distance of 1 cannot be used to detect errors. This is because a single error in a specific bit in one or more code words causes the received code word to be received as a valid code word. A minimum Hamming distance of 2 will allow one error to be detected. In general, a code C can detect up to N errors for any code word if $d(C)$ is greater than or equal to $N+1$ (i.e. $d(C) \geq N+1$). For this it can be shown that:

$$\text{The number of errors detected} = d - 1$$

where d is the minimum Hamming distance.

Error detection allows the receiver to determine if there has been a transmission error. It cannot rebuild the correct data and must either request a retransmission or discard the data.

13.2 Parity

Simple parity adds a single parity bit to each block of transmitted symbols. This parity bit either makes them have an even number of 1's (even parity) or an odd number of 1's (odd parity). It is a simple method of error detection and requires only exclusive-OR (XOR) gates to generate the parity bit. This output can be easily added to the data using a shift register.

Parity bits can only detect an odd number of errors, i.e. 1, 3, 5, and so on. If an even number of bits is in error then the parity bit will be correct and no error will be detected. This type of coding is normally not used on its own or where there is the possibility of several bits being in error.

13.3 Block parity

Block parity is a block code which adds a parity symbol to the end of a block of code. For example a typical method is to transmit the one's complement (or sometimes the two's complement) of the modulo-2 sum of the transmitted values. Using this coding, and a transmitted

block code after every 8 characters, the data:

1, 4, 12, –1, –6, 17, 0, –10

would be arranged as:

1	0000 0001
4	0000 0100
12	0000 1100
–1	1111 1111
–6	1111 1010
17	0001 0001
0	0000 0000
–10	1111 0110
	1110 1011

It can be seen that modulo-2 addition is 1110 1011 (which is –21 in decimal). Thus the transmitted data would be:

0000 0001 0000 0100 0000 1100 1111 1111 1111
1010 0001 0001 0000 0000 1111 0110 1110 1011 ...

In this case, a single error will cause the checksum to be wrong. Unfortunately, as with simple parity, even errors in the same column will not show-up an error, but single errors in different columns will show up as an error. Normally when errors occur they are either single-bit errors or large bursts of errors. With a single-bit error the scheme will detect an error and it is also likely to detect a burst of errors, as the burst is likely to affect several columns and also several rows.

This error scheme is used in many systems as it is simple and can be implemented easily in hardware with XOR gates or simply calculated with appropriate software.

The more symbols are used in the block, the more efficient the code will be. Unfortunately, when an error occurs the complete block must be retransmitted.

13.4 Checksum

The checksum block code is similar to the block parity method but the actual total of the values is sent. Thus it is it very unlikely that an error will go undiscovered. It is typically used when ASCII characters are sent to represent numerical values. For example, the previous data was:

1, 4, 12, –1, –6, 17, 0, –10

which gives a total of 17. This could be sent in ASCII characters as:

‘1’ SPACE ‘4’ SPACE ‘1’ ‘2’ SPACE ‘–’ ‘1’ SPACE ‘–’ ‘6’ SPACE ‘1’ ‘7’ SPACE ‘0’ SPACE ‘–’ ‘1’ ‘0’ SPACE ‘1’ ‘7’

where the SPACE character is the delimiting character between each of the transmitted values. Typically, the transmitter and receiver will agree the amount of numbers that will be transmitted before the checksum is transmitted.

13.5 Cyclic redundancy checking (CRC)

CRC is one of the most reliable error detection schemes and can detect up to 95.5% of all errors. The most commonly used code is the CRC-16 standard code which is defined by the CCITT.

The basic idea of a CRC can be illustrated using an example. Suppose the transmitter and receiver were both to agree that the numerical value sent by the transmitter would always be divisible by 9. Then should the receiver get a value which was not divisible by 9 would know it knows that there had been an error. For example, if a value of 32 were to be transmitted it could be changed to 320 so that the transmitter would be able to add to the least significant digit, making it divisible by 9. In this case the transmitter would add 4, making 324. If this transmitted value were to be corrupted in transmission then there would only be a 10% chance that an error would not be detected.

In CRC-CCITT, the error correction code is 16 bits long and is the remainder of the data message polynomial $G(x)$ divided by the generator polynomial $P(x)$ ($x^{16}+x^{12}+x^5+1$, i.e. 10001000000100001). The quotient is discarded and the remainder is truncated to 16 bits. This is then appended to the message as the coded word.

The division does not use standard arithmetic division. Instead of the subtraction operation an exclusive-OR operation is employed. This is a great advantage as the CRC only requires a shift register and a few XOR gates to perform the division.

The receiver and the transmitter both use the same generating function $P(x)$. If there are no transmission errors then the remainder will be zero.

The method used is as follows:

1. Let $P(x)$ be the generator polynomial and $M(x)$ the message polynomial.
2. Let `n` be the number of bits in $P(x)$.
3. Append n zero bits onto the right-hand side of the message so that it contains $m+n$ bits.
4. Using modulo-2 division, divide the modified bit pattern by $P(x)$. Modulo-2 arithmetic involves exclusive-OR operations, i.e. `0 - 1 = 1`, `1 - 1 = 0`, `1 - 0 = 1` and `0 - 0 = 0`.
5. The final remainder is added to the modified bit pattern.

Example: For a 7-bit data code 1001100 determine the encoded bit pattern using a CRC generating polynomial of $P(x) = x^3 + x^2 + x^0$. Show that the receiver will not detect an error if there are no bits in error.

Answer

$P(x) = x^3 + x^2 + x^0$ (1101)
$G(x) = x^6 + x^3 + x^2$ (1001100)

Multiply by the number of bits in the CRC polynomial.

$$x^3(x^6+x^3+x^2)$$
$$x^9+x^6+x^5 \qquad (1001100000)$$

Figure 13.1 shows the operations at the transmitter. The transmitted message is thus:

1001100001

and Figure 13.2 shows the operations at the receiver. It can be seen that the remainder is zero, so there have been no errors in the transmission.

```
        1111101
1101 | 1001100000
       1101
        1001
        1101
         1000
         1101
          1010
          1101
           1110
           1101
             1100
             1101
              001
```

Figure 13.1 CRC coding example.

```
        1111101
1101 | 1001100001
       1101
        1001
        1101
         1000
         1101
          1010
          1101
           1110
           1101
             1101
             1101
              000
```

Figure 13.2 CRC decoding example.

The CRC-CCITT is a standard polynomial for data communications systems and can detect:

- All single and double bit errors.
- All errors with an odd number of bits.

- All burst errors of length 16 or less.
- 99.997% of 17-bit error bursts.
- 99.998% of 18-bit and longer bursts.

Table 13.1 lists some typical CRC codes. CRC-32 is used in Ethernet, Token Ring and FDDI networks, whereas ATM uses CRC-8 and CRC-10.

Table 13.1 Typical schemes.

Type	*Polynomial*	*Polynomial binary equivalent*
CRC-8	$x^8+x^2+x^1+1$	100000111
CRC-10	$x^{10}+x^9+x^5+x^4+x^1+1$	11000110011
CRC-12	$x^{12}+x^{11}+x^3+x^2+1$	1100000001101
CRC-16	$x^{16}+x^{15}+x^2+1$	11000000000000101
CRC-CCITT	$x^{16}+x^{12}+x^5+1$	10001000000100001
CRC-32	$x^{32}+x^{26}+x^{23}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$	10000010010000001000111011011 0111

13.5.1 Mathematical representation of the CRC

The main steps to CRC implementation are:

1. Prescale the input polynomial of $M'(x)$ by the highest order of the generator polynomial $P(x)$. Thus:

$$M'(x) = x^n M(x)$$

2. Next divide $M'(x)$ by the generator polynomial to give:

$$\frac{M'(x)}{G(x)} = \frac{x^n M(x)}{G(x)} = Q(x) + \frac{R(x)}{G(x)}$$

which gives:

$$x^n M(x) = G(x)Q(x) + R(x)$$

and rearranging gives:

$$x^n M(x) + R(x) = G(x)Q(x)$$

This means that the transmitted message ($x^n M(x) + R(x)$) is now exactly divisible by $G(x)$.

13.5.2 CRC example

Question A
A CRC system uses a message of $1+x^2+x^4+x^5$. Design a FSR cyclic encoder circuit with generator polynomial $G(x)=1+x^2+x^3$ and having appropriate gating circuitry to enable/disable the shift out of the CRC remainder.

Answer - Part A
The generator polynomial is $G(x)=1+x^2+x^3$, the circuit is given in Figure 13.3.

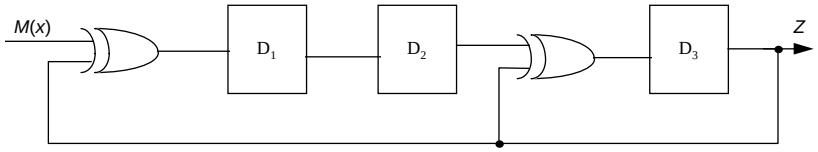

Figure 13.3 CRC coder.

Now to prove that this circuit does generate the polynomial. The output $Z(x)$ will be:

$$Z(x) = Z(x)x^{-1} + \left[M(x)x^{-2} + Z(x)x^{-2} \right]x^{-1}$$
$$= Z(x)\left(x^{-3} + x^{-1}\right) + M(x)x^{-3}$$

Thus:

$$M(x) = \frac{Z(x)\left[1 + x^{-1} + x^{-3}\right]}{x^{-3}}$$

giving:

$$P(x) = \frac{M(x)}{Z(x)} = x^3 + x^2 + 1$$

Question B
If the previous CRC system uses a message of $1+x^2+x^4+x^5$ then determine the sequence of events that occur and hence determine the encoded message as a polynomial $T(x)$. Synthesize the same code algebraically using modulo-2 division.

Answer B
First prescale the input polynomial of $M(x)$ by x^3, the highest power of $G(x)$, thus:

$$M'(x)=x^3.M(x)= x^3+x^5+x^7+x^8$$

The input is thus $x^3+x^5+x^7+x^8$ (000101011), and the generated states are:

Time	$M'(x)$	D_1	D_2	D_3	D_4	
1	000101011	0	0	0	0	
2	00010101	1	0	0	0	← MSD
3	0001010	1	1	0	0	
4	000101	0	1	1	1	
5	00010	0	0	0	0	
6	0001	0	0	0	0	
7	000	1	0	0	0	
8	00	0	1	0	0	
9	0	0	0	1	1	← LSD
10		**1**	**0**	**1**		

The remainder is thus 101, so $R(x)$ is x^2+1. The transmitted polynomial will be:

$$T(x) = x^3 M(x) + R(x) = x^8 + x^7 + x^5 + x^3 + x^2 + 1 \text{ (110101101)}$$

To check this, use either modulo-2 division to give:

$$\begin{array}{r|l} & x^5 \quad +1 \\ \hline x^3+x^2+1 & x^8+x^7+x^5+x^3 \\ & x^8+x^7+x^5 \\ \hline & x^3 \\ & x^3+x^2+1 \\ \hline \text{Remainder} \rightarrow & \boxed{x^2+1} \end{array}$$

This gives the same answer as the state table, i.e. x^2+1.

Question C
Prove that the transmitted message does not generate a remainder when divided by $P(x)$.

Answer C
The transmitted polynomial, $T(x)$, is $x^8+x^7+x^5+x^3+x^2+1$ (110101101) and the generator polynomial, $G(x)$, is $1+x^2+x^3$. Thus:

$$\begin{array}{r|l} & x^5 \quad +1 \\ \hline x^3+x^2+1 & x^8+x^7+x^5+x^3+x^2+1 \\ & x^8+x^7+x^5 \\ \hline & x^3+x^2+1 \\ & x^3+x^2+1 \\ \hline \text{Remainder} \rightarrow & \boxed{0} \end{array}$$

As there is a zero remainder, there is no error.

- Prof Bill Buchanan, 2013