TCOM 370 NOTES 99-9

CYCLIC CODES, AND THE CRC (CYCLIC REDUNDANCY CHECK) CODE

1. CYCLIC CODES

Cyclic codes are a *special type of linear block code* that are popular because they are very effective for error detection and correction and their coders and decoders are easy to implement in hardware.

Definition:
A cyclic code is a linear (N,k) block code with the property that ***every cyclic shift of a codeword results in another codeword***.

Cyclic shifts of any finite word $[b_{N-1}, b_{N-2}, ..., b_0]$ of binary digits are generated by writing out the sequence and shifting the bits (left or right) by the desired number, in such a way that any bits which exit the word at one end re-enter the word at the other end. In other words, we shift bits with carry-around of bits that fall of one end to the other end.

For example, the generator matrix $G=\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$ for a (3,2) linear block code produces the codewords [000], [110], [101], and [011] (corresponding to data words [00], [10], [01], and [11], respectively). Pick any codeword, say [110]. Its cyclic shifts [011] and [101] are both valid codewords.

2. CYCLIC REDUNDANCY CHECK CODES

A very popular **error detecting** code implemented in many data transmission schemes is the **cyclic redundancy check** (CRC) code. Remember that fundamentally this is a type of linear block code.

Polynomial Representation of Binary Words

To understand how a CRC coder works, let us first define the notion of the **polynomial** associated with a binary sequence. The polynomial is in terms of some dummy variable X, the powers of which are combined with binary (0 and 1) coefficients.

- For a bit sequence $[b_{k-1}, b_{k-2}, ..., b_1, b_0]$ the associated polynomial is

  $b_{k-1}X^{k-1}+ b_{k-2}X^{k-2} + ...+ b_1X + b_0$

  Note that this a polynomial of **order k-1** for a bit sequence (word) of **length k bits.**

For example, for the data (message) bit sequence [1010100101] of k=10 bits, the polynomial representation is

$$M(X)=X^9 + X^7 + X^5 + X^2 + 1.$$

Suppose k message or data bits are encoded into N code bits by **appending** to the message bits a sequence of **n=N-k bits** $[r_{n-1}, r_{n-2}, ..., r_1, r_0]$. Let R(X) be the polynomial representing these appended bits. Then the codeword of length N=k+n corresponding to the message M(X) is

$$[b_{k-1}, b_{k-2}, ..., b_1, b_0, r_{n-1}, r_{n-2}, ..., r_1, r_0]$$

for which the corresponding polynomial is clearly

$$T(X) = X^n M(X) + R(X).$$

This follows because the original message bits now occupy more significant bit positions in the codeword; each message bit is moved left by n bits to make room for the n appended bits.

For example, to the above 10-bit message sequence if we append the 3-bit sequence 111, the resulting 13-bit code sequence has the polynomial representation

$$X^3 (X^9 + X^7 + X^5 + X^2 + 1) + X^2 + X + 1= X^{12} + X^{10} + X^8 + X^5 + X^3 + X^2 + X + 1.$$

ENCODING AS COMPUTATION OF REMAINDER IN POLYNOMIAL DIVISION

To define a code we have to state how to obtain the polynomial R(X) (that is, the appended bits) corresponding to any set of message bits or M(X). The appended bits are called the **frame check sequence** for the data frame of k bits.

- A **CRC code** with **n appended bits or Frame Check Sequence (FCS) bits** is defined in terms of a special **generator polynomial G(X) of degree n** with non-zero highest and lowest-order coefficients.

(For example, for n=12, G(X) could be $X^{12} + X^{11} + X^3 + X^2 + X + 1$).

- The generator polynomial plays the role of the generator *matrix* for linear block codes.

- To obtain the FCS bits, or equivalently R(X), we **divide X^nM(X) by G(X) (modulo-2 division) and obtain the remainder, which is R(X).**

Note that in modulo-2 division of polynomials with binary coefficients,

(i) we need to **subtract coefficients of like powers in doing the long division** (as in any long division);

(ii) **subtracting binary coefficients modulo-2** is the same as **adding binary coefficients modulo-2.**

In modulo-2 arithmetic we have

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

Example: Message 11100110 (8 bits)

$M(X) = X^7 + X^6 + X^5 + X^2 + X$

The FCS is to be of length N-k=n=4, we are given $G(X)=X^4+ X^3 + 1$

Let's do the division of $X^nM(X)$ by $G(X)$:

$$\frac{X^{11} + X^{10} + X^9 + X^6 + X^5}{X^4 + X^3 + 1} = X^7 + \frac{X^9 + X^7 + X^6 + X^5}{X^4 + X^3 + 1}$$

$$= X^7 + X^5 + \frac{X^8 + X^7 + X^6}{X^4 + X^3 + 1}$$

$$= X^7 + X^5 + X^4 + \frac{X^6 + X^4}{X^4 + X^3 + 1}$$

$$= X^7 + X^5 + X^4 + X^2 + \frac{X^5 + X^4 + X^2}{X^4 + X^3 + 1}$$

$$= X^7 + X^5 + X^4 + X^2 + X + \frac{X^2 + X}{X^4 + X^3 + 1}$$

The remainder $R(X) = X^2 + X$, therefore the FCS is (since we know n=4) [0110]

Note that the remainder is always a polynomial of maximum order n-1 (3 in this example).

Also note that $X^nM(X)$ is the polynomial corresponding to the message bit sequence to which a number n of 0's is appended (in this example, 111001100000). The polynomial division can also be thought of as a **division of this binary sequence (message bits + n appended 0's) by the sequence corresponding to the generator polynomial,** here 11001.

3. DECODING AS POLYNOMIAL DIVISION

Upon reception of the transmitted codeword **without error**, if the polynomial for the received codeword, $T(X)=X^n M(X) + R(X)$, is divided by $G(X)$, we get **remainder** $R(X) + R(X)$ = **the all-zero** row (note that all additions are modulo-2), because $\frac{X^n M(X)}{G(X)}$ has remainder $R(X)$ (codeword construction definition) and $\frac{R(X)}{G(X)}$ has remainder $R(X)$ since $R(X)$ is a lower order polynomial than $G(X)$.

If the remainder is not zero when the received codeword polynomial is divided by the generator polynomial $G(X)$, we have an indication that an **error** has occurred in transmission and the received word is not a valid codeword.

If an error occurs, let the **error pattern** (a length-N sequence with 1's in the positions where bit reversals or errors have occurred) have polynomial $E(X)$. Then the received polynomial is $T(X)+E(X)$. The remainder we get when dividing this by $G(X)$ is simply the remainder obtained in dividing $E(X)$ by $G(X)$. It can be shown (by considering when $\frac{E(X)}{G(X)}$ will have a remainder) that with an n-bit FCS, **we can design generator polynomials such that it is possible to detect**

(a) **all single errors;**

$G(X)$ contains at least two terms, X^n and 1. For single error, $E(X)$ is of the form X^i. Thus $G(X)$ can not divide $E(X)$ without remainder.

(b) **all double errors**

For two errors we have $E(X)=X^i+X^j$ for $N-1 \geq i>j \geq 0$. Thus $E(X)=X^j (X^{i-j}+1)$. So $G(X)$ must not divide X^p+1, for any p up to N-1; good low-degree $G(X)$ with the property that $G(X)$ not divide X^p+1 for up to very large p are readily available.

(c) **any odd number of errors** [as long as $G(X)$ has a factor $(1+X)$, as is the case for popular CRC polynomials used in practice]

For example, $E(X)$ could be X^5+X^2+1. It turns out that it is not possible to divide into this type of polynomial, with an odd number of terms, with any $G(X)$ that has $(X+1)$ as a factor! Suppose you could find $(X+1)P(X)$ that divides into such an

E(X). This would mean E(X)=(X+1)P(X)Q(X), where Q(X) is the result of the division (without remainder). Now put X=1 in this result; the left hand side has exactly an odd number of 1's, so the mod-2 sum is 1. On the right hand side, (X+1) will always give a 0 → contradiction.

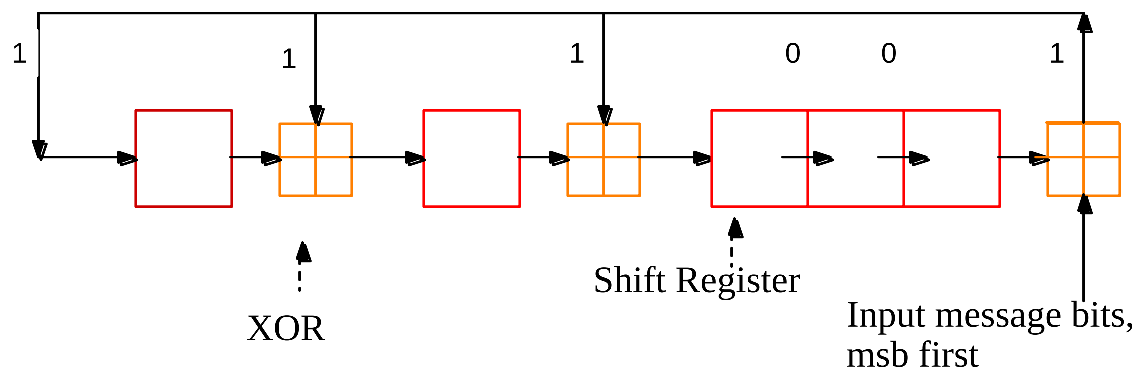
(d) **any error burst of length ≤ n,** and most larger error bursts.
For n bits in FCS, G(X) is a polynomial of degree n. For error burst of duration p≤n, E(X) is of the form $X^m + ... + X^{m-p+1} = X^{m-p+1}(X^{p-1}+...+1)$ But G(X), of degree n, cannot divide this without a remainder.

These properties make CRC codes very useful for error detection in transmitting long frames of binary digits. The best known CRC codes for communications applications use the CRC-16, CRC-32, and CRC-CCITT generator polynomials. Useful shift-register implementations of division of polynomials can be implemented easily.

4. CRC LONG DIVISION IMPLEMENTATION

The CRC code is generated through a process of long division. This can be implemented using a digital logic circuit consisting of exclusive-or (XOR) gates and a shift register. An XOR gate is a mod-2 adder with two one-bit inputs. It implements 0+0=0, 1+0=1, 0+1=1, and 1+1=0. (In practice each input and output takes on one of two possible voltage levels in an electrical circuit). The shift register is a string of 1-bit storage devices, each with an input line and an output line. At specific clock times, the stored bit in each register is shifted out and is replaced by a new bit from the input line.

Consider dividing a binary message $[b_{k-1}, b_{k-2}, ... , b_0]$ of length k with n=5 appended zeros [00000], by the divisor [100111] to find the remainder (the FCS). Here the left-most bit position is the msb. The following digital circuit accomplishes this division, and functions as the **FCS generator.**



Middle 4 bits of divisor 1-0011-1 implemented in the locations of the XOR's between the n=5 shift registers. (First and last bits of the divisor are always "1"). Each shift register location (n=5 total) with an XOR before it implements a "1" of the divisor. Note order for these locations is from lsb to msb

The 1's and 0's shown in the figure simply indicate that the appropriate divisor has been implemented in the specific hardware arrangement of the XOR's above.

Initially the shift register contents are set to 0. After all k message bits have been clocked into the circuit through the right-most XOR, the contents of the shift registers are the desired n=5 bits of remainder. These bits can be clocked out once they are available in the shift registers, by putting 0's on the feedback line after isolating it from the output of the right-most shift register.

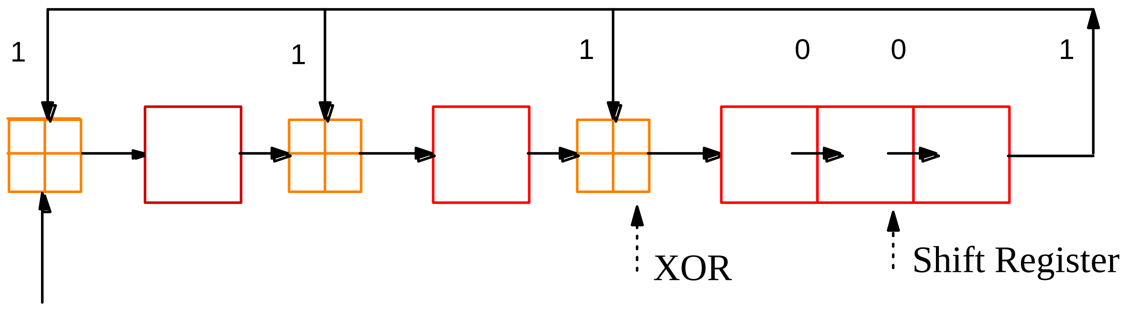
Check: Assume message sequence of "1" and seven "0's", input these into the system above, and check if you see the remainder sequence 11010 in the shift registers at the end.

Input	feedback	Shift Register Contents (Left to Right)				
		0	0	0	0	0
1	1	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	0	1	1	1
0	1	1	1	1	1	1
0	1	1	0	0	1	1
0	1	1	0	1	0	1
0	1	1	0	1	1	0
0	0	**0**	**1**	**0**	**1**	**1**

Remainder LSB to MSB

(Verify that this is the correct remainder by doing the long division of 10000000 00000 by 100111)

The **receiver (decoder)** is very similar. The only difference is that the rightmost XOR is now moved over to just before the leftmost shift register.



Received message bits, msb first, including FCS

Now the received (input) bits are clocked in from the left, again with msb first, and with the registers initially containing 0's. After the message and the FCS bits have been all clocked in, the shift registers contain the remainder which must be all 0 if there are no errors. The remainder can also be clocked out by continuing to feed extra 0’s at the left and cutting the feedback path from the output of the right-most shift register.

Note that since this receiver implements division, it could have been used as the transmit encoder also. In fact, this (second) scheme is the standard scheme for dividing polynomials and producing the results. However, the first scheme above is more efficient for the transmitter because the trailing 0’s are not explicitly clocked in for the division producing the FCS; the fact that the message bits are clocked in at the right side of the shift registers takes care of this. So the first scheme is a more efficient encoding scheme for producing a remainder, it implements division with implicit trailing 0’s.

Note finally that these simple digital structures are practical implementations for producing the FCS and for checking a received message, and are possible because of the **cyclic** structure of the CRC.