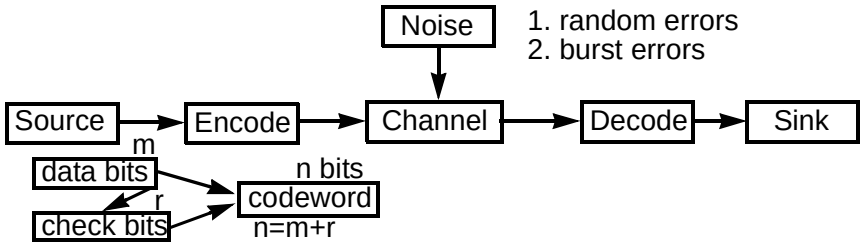
Error Control Techniques

Model of a conventional signaling system



Two categories of error control techniques

1. ARQ (automatic-repeat-request)

buffering, error-detection-codes, acknowledgment channel, retransmission.

2. FEC (forward error control)

error-correction codes (put enough redundancy information for correction).

FEC is inferior to ARQ except when

1. an acknowledgment channel is not available or expensive, or, even dangerous! (e.g. deep space comm. such as Voyage II)

2. the small fraction of correctable error patterns has almost all the probability weight.

Arithmetic Checksum

Error detection at the higher layer is usually done by ordinary arithmetic operations. This is simpler in software but somewhat less effective than a CRC.

Standard technique is to view packet as sequence of k numbers of n bits each, say $x_1, x_2,..., x_k$.

Checksum is then the n bit number $x_1+x_2+...+x_k$ using ordinary arithmetic with no carry.

Alternatively, checksum might be 2n bits; first n bits is (sum) $x_1+x_2+...+x_k$ and second n bits is (sum of sum) $x_1+2x_2+3x_3+...+kx_k$.

Example: In TCP, n=16, checksum is 16 bits and one's complement of the sum. In ISBN, the data are radix 10 digits, checksum is radix 11 digit (with 10 represented as X) and is (sum of sum of all digits)/11.

Weighted code used in ISBN number for Error Detection

Our textbook has a ISBN number 0-13-162959-X.

To check that this number is a proper ISBN number we proceed as follows:

Number	SUM	SUM of SUM
0	0	0
1	1	1
3	4	5
1	5	10
6	11	21
2	13	34
9	22	56
5	27	83
9	36	119
10=X	46	165=11x15

165 mod 11 =0

↓

a correct ISBN number

Coding Theory

"Coding and Information Theory", by Richard Hamming, Prentice-Hall.

A code consists of the rule/algorithm for computing check bits from data bits and for generating codewords from data bits and check bits.

The coding algorithm defines the legal or illegal codewords.

For fixed length codes,

For x, y $\in$ the set of codewords, *Hamming distance*, Hd(x,y) is the no. of 1's in d, and d=x$\oplus$y.

The *Hamming distance of a code*, C, is Hd(C)=min{z | z = Hd(x,y) where x, y are code-words of C, and x$\neq$y}.

To detect d (single) errors, we need a code C with Hd(C)=d+1.

To correct d errors, we need a code C with Hd(C)=2d+1.

Exercise: Prove the Hd(odd parity code) = 2.

For single error correcting code, where m(r) is the no. of data(check) bits,

How many check bits is required? Prove that r must satisfy $(m+r+1) \leq 2^r$.

Each of 2^m msgs has n illegal codes at Hamming distance 1 from it.

$\Rightarrow$ Each of 2^m msgs requires n+1 bit patterns from 2^n bit patterns.

$\Rightarrow (n+1)2^m \leq 2^n \Rightarrow (m+r+1)2^m \leq 2^{m+r} \Rightarrow (m+r+1) \leq 2^r$.

For m=8, $8+r+1 \leq 2^r$, $\Rightarrow$ r=4.

Hamming's Single Error Correcting Code (SECC)

- It can correct single bit error. Invented by Richard Hamming.
- Codewords are encoded with check bits interleaved with data bits in the following order with check bits C_0 C_1 C_2, C_3, ..., C_n in position 1, 2, 8,..., 2^n

  3 5 6 7 9 1011 Data bit Positions

  $C_oC_1D_1C_2D_2D_3D_4C_3D_5D_6D_7D_8D_9D_{10}D_{11}D_{12}D_{13}D_{14}D_{15}$

To generate the codeword,

- First line up the data bits on postions 3,5,6,7,9,...
- For each bit with value 1, use its bit positions in the codeword to generate binary parity bit pattern contributing to the check bits.
  e.g. if D1 is 1, then since its position in the codeword is 3, it contributes binary parity pattern of 0011. if D is 1, since its codeword position is 7, it contributes binary parity pattern of 0111.
  - The binary bit pattern corresponds to check bit $C_n...C_2C_1C_0$. Therefore 0011 pattern implies the contribution of 1 to the parity bit computation of check bits C_0 and C_1.
  - Reverse the binary bit pattern and align them beneath check bits, since in the codeword, the parity bits are lined up with $C_0C_1C_2C_3$... order.

- For each check bit, counting the parity bit contributing to it by all data bits and generate the check bit using the even parity bit.
  In the following example, C_0 receives 1 parity bit contributed by D_1 and another by D_4. With even number of parity bit contribution, we generate bit 0 for C_0. For C_2, only D_4 contribute one parity bit, therefore, we generate bit 1 for C_2 to make the total number of bit 1 even.
- After check bits are generated, the codeword is sent to the receiver.

Decoding SECC

- After receiving the codeword, the receiver will set aside those check bits and recompute the check bits only based on the data bits.
- If a check bit computed by the receiver is different from the received check bit, then mark the check position with X.
- For each error check bit position, add the value $2^{(\text{check bit position})}$ to a variable error_location. For example, in the following example, we have C_0, C_1, and C_2 error. We add error_location=$2^0+2^1+2^2=1+2+4=7$. This indicates that bit position 7 in the codeword is wrong. Let us correct it by complementing its value.

Hamming's Single Error Correcting Code

				3		5	6	7		9	10	11		Data bit Positions														
	C_0	C_1	D_1	C_2	D_2	D_3	D_4	C_3	D_5	D_6	D_7	D_8	D_9	D_{10}	D_{11}	D_{12}	D_{13}	D_{14}	D_{15}									
H			**1**		0	0	**1**		0	0	0									Message				C_3	C_2	C_1	C_0
	1	1																		D_1 is 1 in position 3⇒contribute				0	0	1	1
	1	1			1															D_4 is 1in position 7⇒contribute				0	1	1	1
	0	0	1	1	0	0	1	0	0	0	0									Encode the check bits using even parity							
							X													bit 7 Error							
	0	0	1	1	0	0	0	0	0	0	0									Receive							
	1	1																		only D_1 is 1 in position 3⇒contribute				0	0	1	1
	1	1		0				0												Regenerate Check bits							
	X	X		X																Errors in the check bits							
	1+	2+		4=7			X													position of error is bit 7.							
	0	0	1	1	0	0	1	0	0	0	0									Corrected Message							
a			1		1	0	0			0	0	1								ASCII code							
	1	1																		D_1 is 1 in position 3⇒contribute				0	0	1	1
	1			1																D_2 is 1 in position 5⇒contribute				0	1	0	1
	1	1							1											D_7 is 1 in position 11⇒contribute				1	0	1	1
	1	0	1	1	1	0	0	1	0	0	1									Encoded a							

Exercise: Illustrate how the receiver corrects bit 6 error in a Hamming code of 'U'?

Exercise on Hamming Code

Illustrate how the receiver correct bit 6 error in the Hamming code of 'U'=1010101.

Hamming's Single Error Correcting Code

3 5 6 7 8 9 10 11 Positions

$C_0C_1D_1C_2D_2D_3D_4C_3D_5D_6D_7D_8D_9D_{10}D_{11}D_{12}D_{13}D_{14}D_{15}$

C0 checks positions 1, 3, 5, 7, 9, 11, 13, 15,...(make it even parity.)
C1 checks positions 2, 3, 6, 7, 10, 11, 14, 15,...
C2 checks positions 4, 5, 6, 7, 12, 13, 14, 15,...
C3 checks positions 8, 9, 10, 11, 12, 13, 14, 15, 24, 25,...

U 1010101 Message

		1		0	1	0		1	0	1	Data bits
1	1	1	1	0	1	0	0	1	0	1	Encode
					X						bit 6 Error
1	1	1	1	0	0	0	0	1	0	1	Code Received
		1		0	0	0		1	0	1	Data bits Received
1	0		0				0				Regenerate Check bits Using Received Data
	X		X								Errors in the check bits,
	2+		4=6	X							Add the weight of error checkit position=6.
		1		0	1	0		1	0	1	Reverse bit 6⇒Corrected Message

Burst Error Correction

Arrange k Hamming codewords in a matrix

Use k*r check bits to correct a single burst error of length k.
Trade-off is the delay increases from n/C to k*n/C where C is the link capacity.
ECC vs. EDC

For error rate=10^{-6}, 1000-bit data per block, and a msg=1000 blocks is to be sent, use odd parity code (EDC),

Msg=10^6 bits $\Rightarrow$ only one bit error $\Rightarrow$ only one error block needs retransmit

$\Rightarrow$ the overhead is (1001+1000x1)/1001x1001 $\approx$ 0.002;

use ECC,

1000+r+1 $\leq 2^r \Rightarrow$ r=10 $\Rightarrow$ each codeword has 1010 bits,

the overhead is 10/1010 $\approx$ 0.0099.

Polynomial Code, Cyclic Redundancy (CRC) Code

Polynomial Code, also called CRC code, is a class of Error Detecting Codes, it uses polynomial arithmetic to calculate the check bits.

A bit string can be represented as a polynomial with coefficient 0 or 1.

110001$\Rightarrow$ M(x) = x^5+x^4+1, degree of M(x) = 5.

x^4M(x)=$x^9+x^8+x^4\Rightarrow$1100010000 multiply x^4 is equivalent to "logical shift" M(x) to left by 4 bit. **Polynomial arithmetic:**

```
  10011011        10011011
+ 11001010      - 11001010
----------      ----------
  01010001        01010001
```

+ and - are same xor operation; no carry-over

```
        110
   ------
11 | 1011
     11
     --
      11
      11
      --
       1   <- remainder
```

It sends checksumed code (frame) T(x)=x^rM(x)–(x^rM(x)%G(x)) where M(x) is the msg and G(x) is the generator polynomial. T(x) is divisible by G(x). G(x) has degree of r.

Assume that T(x)+E(x) is received, here E(x) represents the error bits,

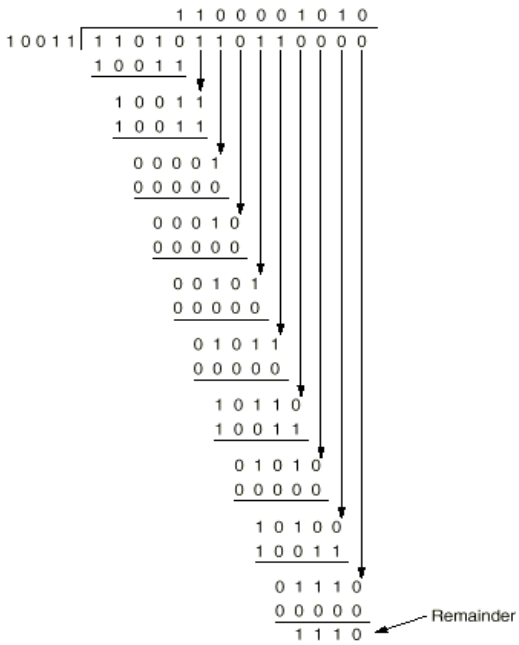
If T(x)+E(x) is not divisible by G(x), then error is detected.

If T(x)+E(x) is divisible by G(x), then

case 1. E(x)=0, no error.

case 2. E(x)!=0, and E(x) is divisible by G(x). What are these E(x)s?

Frame : 1 1 0 1 0 1 1 0 1 1
Generator: 1 0 0 1 1
Message after appending 4 zero bits: 1 1 0 1 0 1 1 0 0 0 0



Transmitted frame: 1 1 0 1 0 1 1 0 1 1 1 1 1 0

Exercise on CRC code

Assume we use generator polynomial $G(x)=x^4+x+1$ for computing the checksum of a frame.

a) Given data=11111111, what is the check sum?

```
          11101100                 11101100                 11100100
10011|111111110000      10011|111111110100 10011|111101110100
     10011                    10011                    10011
      11001                    11001                    11011
      10011                    10011                    10011
       10101                    10101                    10001
       10011                    10011                    10011
        11010                    11010                       10001
        10011                    10011                       10011
         10010                    10011                         1000 <- remainder
         10011                    10011
           0100 <- checksum         0000 <- remainder
```

b) Transmission frame T(x)= 111111110100

c) If not bit was corrupted, the Receiver receives T(x)

d) Receiver performs T(X)/G(x) and remainder =0 (Shown in the middle). That indicates no error.

c2) Assume bit 5 errors, receiving frame R(x)=111101110100

d2) The Receiver performs R(x)/G(x) and get 1000 as remainder (Shown in the right) That indicates the frame got garbled.

CRC Code

It can be shown [PETE61] that all the following are not divisible by G(x).

1. All single-bit errors $E(x)=x^i$. If G(x) contains two terms. E(x)!= G(x)*H(x)

2. All double-bit errors $E(x)=x^i+x^j=x^i(1+x^{j-i})$, if G(x) has a factor with at least three terms. i th and jth bits have errors.

2a. All double-bit errors $E(x)=x^i+x^j=x^i(1+x^{j-i})$, if G(x) is a primitive with degree of c and the length of codeword is less than 2^c-1. It implies that j-i <=2^c-1.

3. Any odd number errors, i.e.,E(x=1)=1, as long as G(x) contains a factor (x+1).
If G(x)=(x+1)*H(x), then G(x=1)=0*H(x)=0 ⇒E(x=1)=1 != G(x=1)

4. Any burst error for which the length of the burst is less than the length of checksum.

5. Most larger burst errors.

Four version of G(x) are widely used:

CRC-12= $x^{12}+x^{11}+x^3+x^2+x^1+1=(x+1)(x^{11}+x^2+1)$

CRC-16= $x^{16}+x^{15}+x^2+1=(x+1)(x^{15}+x+1)$

CRC-CCITT= $x^{16}+x^{12}+x^5+1$

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

They all contain x+1 as prime factor. CRC-12 is used for transmission of streams of 6-

bit characters with 12-bit checksum. CRC-16 and CRC-CCITT are popular for 8-bit characters with 16-bit checksum. CRC-32 are used in IEEE802 standards with 32bit checksum.

The following definitions are from "Coding Theory: The Essentials" by Hoffman et al.

What is a primitive polynomial? (It was referenced in properties 2a above).

- Let K={0,1}.
- Let K[x] be the set of polynomials whose coefficients are in K.
- Let f(x), g(x), d(x)be a polynomial whose coefficients are in K. They are called polynomials over K.
- Let K[x] be the set of polynomials whose coefficients are in {0, 1}.
- if f(x)=g(x)d(x), then d(x) is a divisor of f(x).
- A **proper divisor** of f(x), say p(x), is a polynomial over K, if p(x)!=1, p(x)!=f(x).
- f(x) is said to be **irreducible** if it has no proper divisors in K[x].
- An irreducible polynomial over K of degree n, n>1, is said to be **primitive** if it is not a divisor of $1+x^m$ for any m<2^n-1.
- Examples of primitive polynomial:
  $1+x+x^2$ is not a factor of $1+x^m$ for any m<3=2^2-1(1+x; $1+x^2$)⇒ it is primitive.
  $1+x+x^3$ is not a factor of $1+x^m$ for any m<7=2^3-1--> it is primitive.
  $1+x+x^2+x^3+x^4$ is irreducible but there is a m=5<15=2^4-1 where $1+x^5=(1+x)(1+x+x^2+x^3+x^4)$. $1+x+x^2+x^3+x^4$ is a factor of $1+x^5$.

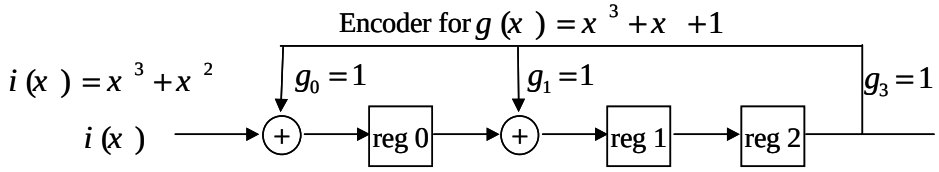
$1+x+x^2+x^3+x^4 \Rightarrow$ is not primitive. (It can not detect double errors separated by 5 bits in the code word, since $(1+x^5)\%1+x+x^2+x^3+x^4$ is zero.
CRC-16 polynomial $(1+x)(x^{15}+x+1)$. Here $x^{15}+x+1$ is a primitive polynomial since it is not factor of $1+x^m$ for any $m < 2^{15}-1=32,767$.
If the length of the codeword is less than 32,767, CRC-16 can detect all double errors.

- case 3. $E(X)=x^4+x^2+1 \Rightarrow$ 3bit errors $\Rightarrow$odd number of errors

$$
\begin{array}{r}
x^3+x^2 \\
x+1 \overline{)\, x^4+\quad x^2+\ 1} \\
\underline{x^4+x^3} \\
\quad x^3+x^2 \\
\quad \underline{x^3+x^2} \\
\qquad\qquad 1
\end{array}
$$

1 with remainder!=0$\Rightarrow$ not divisible.
(x+1) can not divide E(x) with odd number of terms.

CRC Generation Using Shift Registers



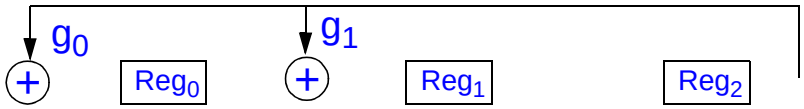
clock	input	reg 0	reg 1	reg 2
0	-	0	0	0
1	$1=i_3$	1	0	0
2	$1=i_2$	1	1	0
3	$0=i_1$	0	1	1
4	$0=i_0$	1	1	1
5	0	1	0	1
6	0	1	0	0
7	0	**0**	**1**	**0**
	check bits:	$r_0 = 0$	$r_1 = 1$	$r_2 = 0$

Implement CRC using Shift Registers

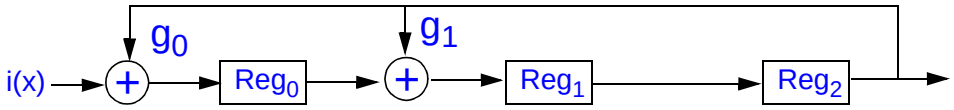
Given a Generator Polynomial $G(x)=g_nX^n+...+g_2X^2+g_1X^1+g_0X^0$,

- Create n registers, label them reg_0, to reg_{n-1} from left to right. e.g., $g(x)=x^3+x+1$; n=3; draw reg_0 to $reg_{3-1}=reg_2$;

- For each non-zero terms, g_i , 0<=i<n, (n not included) draw an exclusive-or operator on the left of reg_i and a line with arrow from the right of reg_{n-1} to the top of the exclusive-or operator. Write a label "gi" to the right of the arrow.



- Draw horizontal lines with arrow that connect all the exclusive-or operators and registers along the way.



Exercise

Prob. 1. Chapter 3-42. ATM uses an eight-bit CRC on the information contained in the header. The header has six fields:

First 4 bits: GFC field

Next 8 bits: VPI field

Next 16 bits: VCI field

Next 3 bits: Type field

Next 1 bit: CLP field

Next 8 bits: CRC

a. The CRC is calculated using the following generator polynomial: $x^8 + x^2 + x + 1$. Find the CRC bits if the GFC VPI Type, and CLP fields are all zero and the VCI field is 00000000 00001111. Assume the GFC bits correspond to the highest-order bits in the polynomial.

Ans: Generator polynomial: $g(x)=x^8 + x^2 + x + 1$

Information: 0000 00000000 00000000 00001111 000 0

$i(x)=x^7 + x^6 + x^5 + x^4$

Encoding: degree of 8, x^8 $i(x)=x^{15} + x^{14} + x^{13} + x^{12}$

Perform polynomial division

$$x^8+x^2+x+1 \;\overline{)\; x^{15}+x^{14}+x^{13}+x^{12}}\;\; x^7+x^6+x^5+x^4+x^1$$

$$x^{15}+\qquad\qquad x^9+x^8+x^7$$
$$x^{14}+x^{13}+x^{12}+x^9+x^8+x^7$$
$$x^{14}+\qquad\qquad x^8+x^7+x^6$$
$$x^{13}+x^{12}+x^9+\qquad x^6$$
$$x^{13}+\qquad\qquad x^7+x^6+x^5$$
$$x^{12}+x^9+\quad x^7+\quad x^5$$
$$x^{12}+\qquad\quad x^6+x^5+x^4$$
$$x^9+\quad x^7+x^6+\quad x^4$$
$$x^9+\qquad\qquad\qquad x^3+x^2+x^1$$
$$x^7+x^6+\quad x^4+x^3+x^2+x^1$$

$$100000111 \;\overline{)\; 1111000000000000}\;\; 11110010$$

100000111
111001110
100000111
110010010
100000111
100101010
100000111
101101000
100000111
11011110

The CRC bit is 11011110

b. Can this code detect single errors? Explain why.

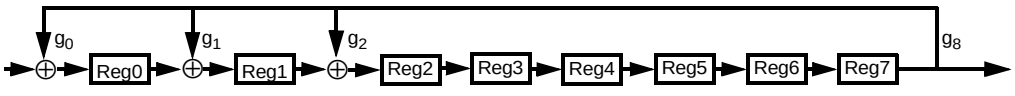
Ans: Yes. Condition 1 in Figure3.60 indicates to detect single errors, G(x) must have more than one term. Since g(x) has 4 terms, it will detect all single errors.

c. Draw the shift register division circuit for this generator polynomial.

Ans:

$G(x)=g_8x^8+g_2x^2+g_1x+g_0$ 1 Here degree of n=8.

- Draw 8 registers with label "Reg i" i=0, 7.
- For each non-zero term g_i, 0<=i<n (n not included), draw an exclusive or symbol (circle with +) to the left of reg_i and and a line with arrow from the right of reg_{n-1} to the top of the exclusive-or operator. Write a label “gi” to the right of the arrow.
  Here g_2, g_1, and g_0 are non-zero terms.
- Draw horizontal lines with arrow that connect all the exclusive-or operators and registers along the way.



Correction on CRC-16, page 164

- There is an error in CRC-16 page 164.
- CRC-16 = $(x+1)(x^{15}+x+1)=X^{16}+x^{15}+x^{2}+1$ (there is no x term. they cancel out)

$$
\begin{array}{rl}
 & x^{15}+\quad x+1 \\
x & \quad x+1 \\
\hline
 & x^{15}+\quad x+1 \\
x^{16}+ & x^{2}+\ x \\
\hline
x^{16}+x^{15}+x^{2}+ & \quad 1
\end{array}
$$

-