Capítulo

1

Detecção de Erros

Errar é humano. Perdoar não é a política da empresa
(As 100 melhores leis de Murphy)

Autor: Constantino Seixas Filho
UFMG – Departamento de Engenharia Eletrônica

Detecção de Erros

Erros de transmissão de dados podem ter diversas causas:

Ruído	• Branco • Impulsivo
Distorções	• Atenuação em amplitude • Retardo de fase • Deslocamento de freqüência

Ruídos em geral ocorrem em rajadas (*bursts*):
Imagine uma rajada de 10 ms sobre uma comunicação de 9600 bps:
96 bits de dados serão atingidos.

A natureza de erros em rajada é muito importante para a detecção de erros.

Técnicas Primitivas de detecção:

Paridade simples ou paridade vertical ou TRC (*Transverse Redundancy Check*)

A cada caracter adicionamos um bit de paridade.

Paridade par	O número total de 1's na palavra considerando-se o bit de paridade é par.
Paridade ímpar	O número total de 1's na palavra considerando-se o bit de paridade é ímpar.

Seja o caracter:

01001100

Vamos calcular o bit de paridade ímpar:

0	1	0	0	1	1	0	0	0

Determine a expressão para cálculo do bit de paridade ímpar em uma palavra de 8 bits:

$P_i =$

Determine a expressão para cálculo do bit de paridade par em uma palavra de 8 bits:

$P_p =$

Vamos calcular a eficiência de utilização de bits para este código:

$$e = \frac{8}{8+1} = 88.8\%$$

Em geral este bit é calculado pelo hardware de transmissão de dados (USART) e é recebido, verificado e retirado pelo hardware de recepção.

Qual a capacidade de detecção de erros deste algoritmo ?

Apenas erros em um número ímpar de bits são detectados.

Exemplo1:

Caracter transmitido:

0	1	0	0	1	1	0	0	0

Caracter recebido:

0	1	1	0	1	1	0	0	0

A paridade calculada na recepção é 1 o que contraria o valor do último bit da palavra e o erro é detectado.

Exemplo2:

Caracter transmitido:

0	1	0	0	1	1	0	0	0

Caracter recebido:

0	1	1	0	1	0	0	0	0

Existem dois bits trocados. O valor do bit de paridade calculado na recepção é 0. Como o último bit da palavra que corresponde ao bit de paridade recebido também é 1, o erro não é detectado.

Paridade Horizontal ou LRC (*Longitudinal Redundancy Check*)

Considere o bloco de dados a serem transmitidos:

1	0	1	1	0	1	1	0	0	Caracter 1
1	1	0	1	0	1	1	1	1	Caracter 2
0	0	1	1	1	0	1	0	1	Caracter 3
1	1	1	1	0	0	0	1	0	Caracter 4
1	0	0	0	1	0	1	1	1	Caracter 5
1	1	0	1	1	1	1	0	0	Caracter de checagem

O último caracter representa a paridade dos caracteres anteriores calculada na vertical bit a bit.

Eficiência de utilização de bits para este código:

Supondo um bloco de 5 caracteres:

$$e = \frac{8*5}{9*6} = 74.1\%$$

A eficiência aumenta quando aumentamos o tamanho do bloco.

Dois erros em caracter são detectados.
Dois erros em bits de mesma ordem em dois caracteres não são detectados.

Outros códigos de detecção longitudinal de erros são normalmente implementados em automação. A maior parte não usa bits de paridade, mas uma palavra gerada pela soma de todos as demais palavras da mensagem. Esses códigos são conhecidos pelo nome genéricos de ***Checksum***.

Códigos Cíclicos de Detecção de Erros:

CRC – Cyclic Redundancy Code

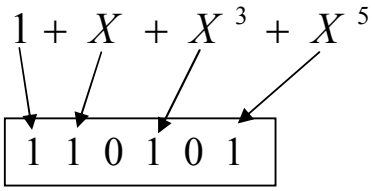
- São capazes de detectar uma grande faixa de erros de transmissão, isolados ou em rajadas.
- Possuem algoritmo de cálculo mais complexos.
- Podem ser calculados por hardware ou software.

Princípio:

1. Cada bit da mensagem m codificada em binário, é considerado como um coeficiente de um polinômio M(X) base 2.
2. A mensagem é deslocada para a esquerda de r posições, onde r é o número de bits do CRC (ordem do polinômio verificador = número de bits da representação do polinômio verificador – 1).
3. A mensagem deslocada é dividida por um polinômio característico G(X).
4. O resto da divisão é somado à mensagem deslocada para formar a mensagem composta T(X).
5. T(X) é transmitida.
6. O receptor divide T(X) por G(X).
7. Se o resultado for 0, existe grande probabilidade da mensagem estar correta, caso contrário, existe um erro.

Exemplo1:

Seja a mensagem: 110101
O polinômio correspondente é:



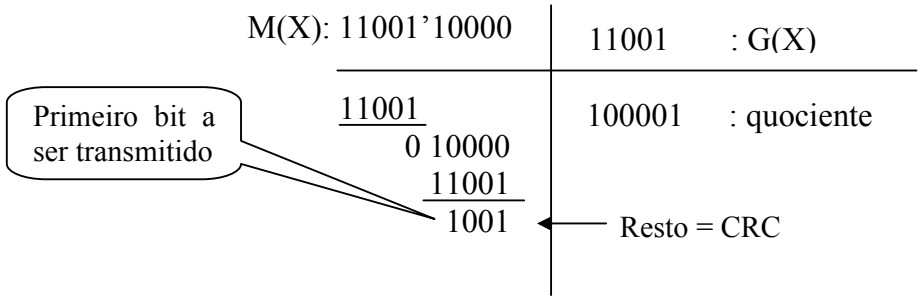
A palavra foi invertida julgando que a mensagem seria transmitida do LSb para o MSb (LSb primeiro).

Exemplo2:

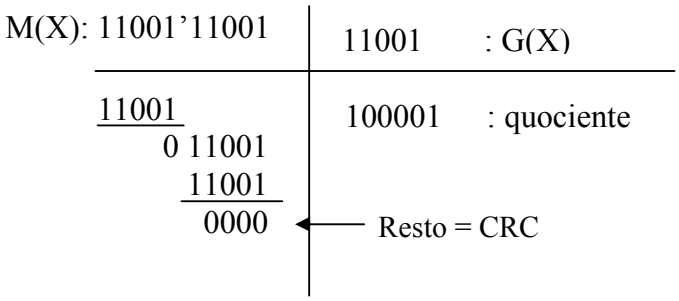
No próximo exemplo vamos considerar a transmissão no sentido inverso: MSb primeiro.

Mensagem: M(X) = 110011 ($X^5 + X^4 + X + 1$)
Polinômio: G(X) = 11001 ($X^4 + X^3 + 1$)

Cálculo do CRC:



T(X) = 1100111001
Observe que toda a aritmética empregada é base 2.
Vamos aplicar o algoritmo de recepção:



Como o resto foi 0, nenhum erro foi detectado.

Análise Matemática:

Seja M(X) a mensagem a ser transmitida e seja G(X) o polinômio verificador.

Representação →	Binária	Polomial
Mensagem original	m	$M(X)$
Mensagem deslocada:	m 0 0 .... 0 ← r →	$X^r M(X)$

Formação da mensagem:

$$X^r M(X) = Q(X)G(X) + R(X)$$
$$X^r M(X) - R(X) = Q(X)G(X) = T(X)$$
$$T(X) = X^r M(X) + R(X)$$

Observe que em módulo 2 as operações + e – se eqüivalem

T(x) é equivalente à nossa mensagem composta.

Transmissão:

$$T(X) \xrightarrow{\text{erros}} T(X) + E(X)$$

Na recepção:

$$\frac{T(X)+E(X)}{G(X)} = \frac{T(X)}{G(X)} + \frac{E(X)}{G(X)}$$

- Resto <> 0: Erro detectado
- Resto = 0:
  - Não houve Erro, ou
  - G(x) é fator de E(x):
  - Erro não detectado

Resto = 0

Polinômios verificadores:

Os polinômios são projetados para detectar erros que possuem certas características.

A referência [Peterson 61] apresenta todos os teoremas, demostrando as propriedades destes polinômios.

Erros simples

Teorema 1:

Um polinômio G(X) com mais de um termo é capaz de detectar qualquer erro simples.
G(X) = X + 1 // G(X) com dois termos
$E = 2^i$ ←-→ $E(X) = X^i$
i é a ordem do bit contada à partir da direita. i=0 para o LSb.

Demonstração:
Para que ocorra detecção de erros simples, é necessário que G(X) não divida X^i.
(X+1) não divide X^i, assim como nenhum polinômio de grau maior de 1.

Número ímpar de erros

Teorema 2:

Todo polinômio divisível por X + 1 tem um número par de termos.
A conseqüência é que X+1 detecta não só qualquer erro simples como também qualquer número ímpar de erros.

Demonstração: (por absurdo)
E(X) tem um número ímpar de termos.
Vamos supor E(X) seja divisível por (X+1):
E(X) = (X+1) Q(X)

Para X = 1 => E(1) = (1+1) Q(1) = 0 . Q(X).
E(1) = 0
Mas E(X) =1 para X = 1 porque E(X) tem um número ímpar de termos.
Logo, chegamos a um absurdo.

Erro de 2 bits

$E = 2^i + 2^j$ $(i > j \text{ e } i - j = k)$
$E(X) = X^j (X^{i-j} + 1)$
G(X) não deve dividir $X^k + 1$

Erros em rajada (burst)

Definição: Rajada de tamanho k: qualquer padrão de erro no qual o número de símbolos entre o primeiro e o último erro, incluindo estes erros é k.

$$E(X) = X^j + \ldots + X^i \qquad j > i$$

Comprimento da rajada = k = j – i + 1

$$E(X) = X^3 + X^6 + X^7$$
$$= 0\,0\,0\,1\,0\,0\,1\,1\,0\,0\,0\,0\,0\,0\,0$$

O comprimento da rajada acima é k = 5.

$$E(X) = X^i (X^{j-i} + \ldots + 1)$$

$$E(X) = X^i E_1(X)$$

G(X) não pode ser um divisor de E_1 (X).

Alternativas	Capacidade de detecção
k <= r	G(X) não pode ser um divisor de E_1 (X) e portanto o polinômio é capaz de detectar qualquer rajada de comprimento inferior ou igual à sua ordem.
k = r +1	j – i + 1 = r + 1 ou j - i = r. Existem r-1 bits no meio da rajada que podem assumir o valor 0 ou 1. Pode-se demonstrar que a probabilidade de que o *pattern* de E_1 (X) coincida com G(X) é: $P = \frac{1}{2^{r-1}}$
k > r + 1	$P = \frac{1}{2^r}$

Polinômios mais utilizados:

CRC-16 $X^{16} + X^{15} + X^{2} + 1$	Usado em sistemas síncronos que utilizam caracteres de 8 bits. **Detecta erros:** Todos simples Todos duplos Todos com número ímpar de bits Todas rajadas de comprimento <= 16 99.997% das rajadas de comprimento 17 99.998% das rajadas >= 18 bits
CRC-CCITT $X^{16} + X^{12} + X^{5} + 1$	Sistema mais usado na Europa. Detecta rajadas de comprimento até 16 e mais de 99% das rajadas de comprimento maior que 16.
CRC-12 $X^{12} + X^{11} + X^{3} + X^{2} + X + 1$	Usado em sistemas síncronos utilizando caracteres de 6 bits. Detecta rajadas de comprimento até 12.

Cálculo do CRC:

O método de divisão polinomial que serviu de referência a este estudo não é usado na prática por ser muito trabalhoso.

Seja a mensagem: M(X) = 000000000000001
Considerando que vamos enviar o LSb primeiro, a mensagem fica:
$M_{LSb}(X)$ = 100000000000000
Polinômio: G(X) = CRC16 = 11000000000000101

Cálculo do CRC através de divisão polinomial:

Primeiro bit a ser enviado

1000 0000 0000 0000 0000 0000 0000 0000		1 1000 0000 0000 0101 : G(X)
1100 0000 0000 0010 1	1	1111111111111101 : quociente
100 0000 0000 0010 10	2	
110 0000 0000 0001 01	3	
10 0000 0000 0011 110	4	
11 0000 0000 0000 101	5	
1 0000 0000 0011 0110	6	
1 1000 0000 0000 0101	7	
1000 0000 0011 0011 0	8	
1100 0000 0000 0010 1	9	
100 0000 0011 0001 10	10	
110 0000 0000 0001 01	11	
10 0000 0011 0000 110	12	
11 0000 0000 0000 101	13	
1 0000 0011 0000 0110	14	
1 1000 0000 0000 0101	15	
1000 0011 0000 0011 0	16	
1100 0000 0000 0010 1	17	
100 0011 0000 0001 10	18	
110 0000 0000 0001 01	19	
10 0011 0000 0000 110	20	
11 0000 0000 0000 101	21	
1 0011 0000 0000 0110	22	
1 1000 0000 0000 0101	23	
1011 0000 0000 0011 0	24	
1100 0000 0000 0010 1	25	
111 0000 0000 0001 10	26	
110 0000 0000 0001 01	27	
1 0000 0000 0000 1100	28	
1 1000 0000 0000 0101	29	
1000 0000 0000 1001	30	← Resto = CRC

Primeiro bit a ser enviado

Cálculo do CRC através de hardware

Pode-se projetar um circuito formado por um registrador de deslocamento *(shift register*) de r bits, sendo r o número de bits do CRC, realimentado por portas XOR. Este tipo de circuito é denominado máquina seqüencial linear. Uma cobertura completa da teoria envolvendo este tipo de circuito pode ser encontrado em [Kohavi 78].

A teoria dos circuitos seqüenciais lineares são utilizados para projetar circuitos capazes de realizar a multiplicação e divisão polinomial em diversas bases numéricas.

Cada estágio de um registrador de deslocamento representa um atraso no sinal de entrada.

Seja o circuito que sintetiza a função: z(t) = x(t) + x(t-1) + x(t-3)

Usando o operador de atraso D (*Delay*) podemos escrever:

$z = x + Dx + D^3 x$ ou

$\frac{z}{x} = D^3 + D + 1$

O circuito que sintetiza esta função é denominado de registrador de deslocamento *feedforward*:

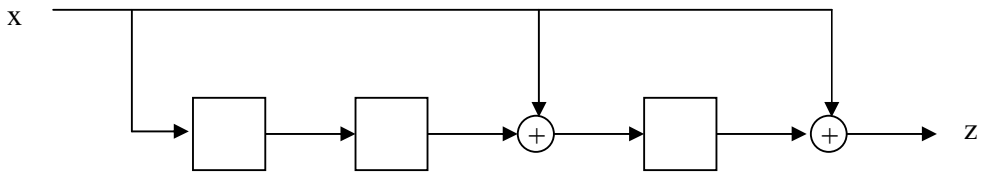


Figura 1: Realização da função $z = x + Dx + D^3 x$ **.**

Este circuito também realiza a multiplicação polinomial base 2.

A máquina que realiza a divisão polinomial (função inversa) é dada por:

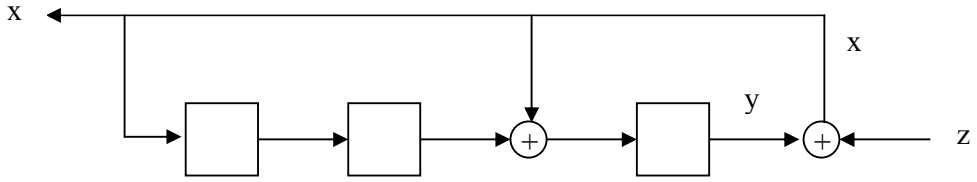


Figura 2: Máquina inversa da figura 1.

A função realizada é:

$$y = D(x + D^2 x) \text{ ou } \quad y = Dx + D^3 x$$

$$x = y + z$$

$$z = x - y = x + y = D^3 x + Dx + x$$

$$\frac{z}{x} = D^3 + D + 1 \text{ ou}$$

$$\frac{x}{z} = \frac{1}{D^3 + D + 1}$$

Observe que neste circuito z é a entrada e x a saída.

O circuito utilizado na prática traduz o algoritmo de divisão polinomial e é dado por:

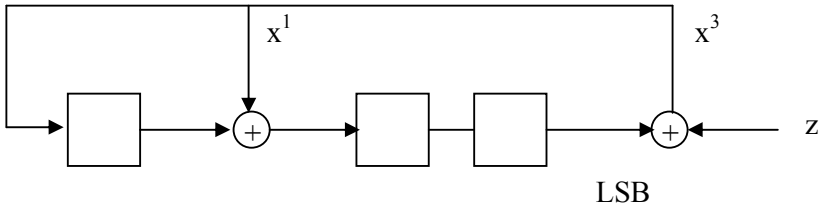


Figura 3: Circuito utilizado na prática

Para obter este circuito:
$P(X) = X^3 + X + 1$
Representação binária: P = 1 0 1 1
Inverte-se a seqüência: Q = 1 1 0 1
Elimina-se o bit menos significativo: Qr = 1 1 0
Cada 1 marca o início de um registrador de deslocamento (*shift register)*.
O valor a ser usado em nossos futuros algoritmos será justamente Qr = 1 1 0, que marca as posições das portas ou-exclusivo no registrador de deslocamento de ordem 3 (3 posições). A este valor chamaremos de **Operando**.

Este circuito realiza a divisão polinomial base 2 e é o circuito utilizado para o cálculo do CRC.

E x e m p l o :

Mensagem = 10100001 (LSB primeiro)
Polinômio = 11001

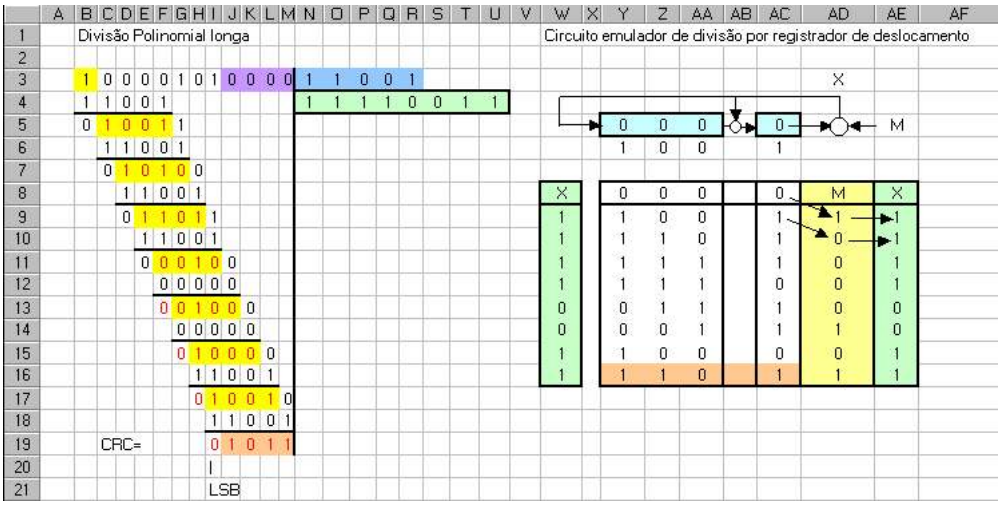


Figura 4 – Comparação do cálculo por divisão longa e por circuito emulador

CRC = 1101

Em seguida vamos apresentar os circuitos de cálculo de CRC para os principais polinômios utilizados.

CRC_12

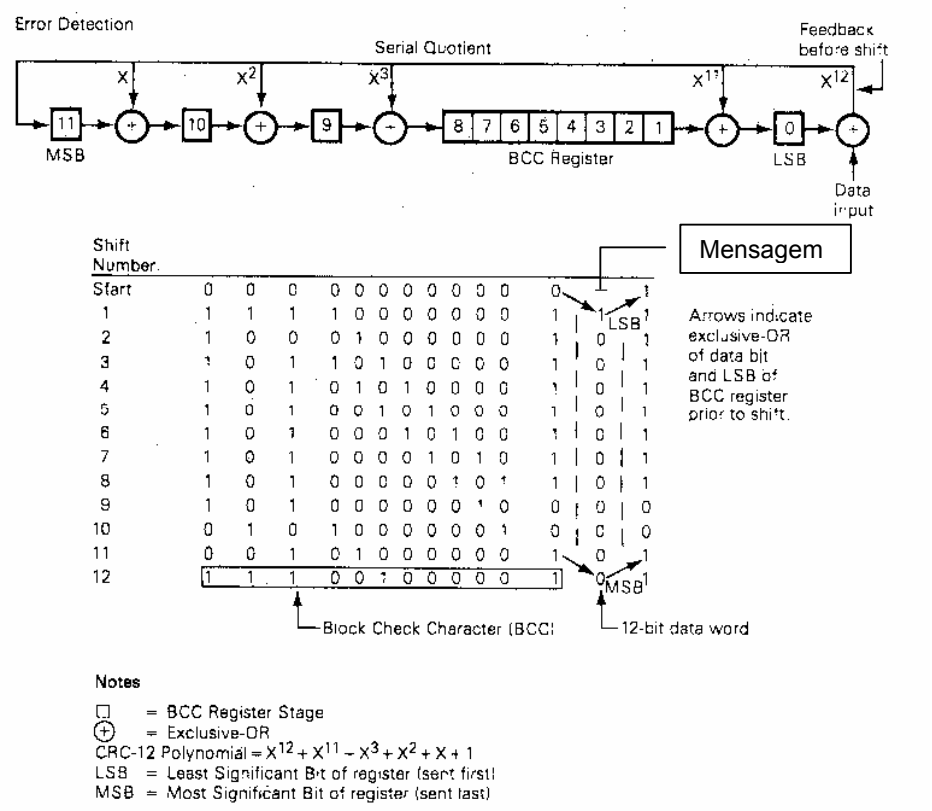


Figura 5: Cálculo de CRC usando CRC_12 – Seqüência de transmissão [McNamara 88]

O registrador é inicialmente zerado.
O string de dados é combinado bit a bit com o conteúdo do registrador de deslocamento. A cada bit as operações de xor são realizadas e o conteúdo do registrador é deslocado de uma posição para a direita. Quando todos os bits da mensagem tiverem sido processados, o conteúdo do registrador é anexado ao final da mensagem (LSB primeiro). A operação XOR deve ser realizada antes do deslocamento.

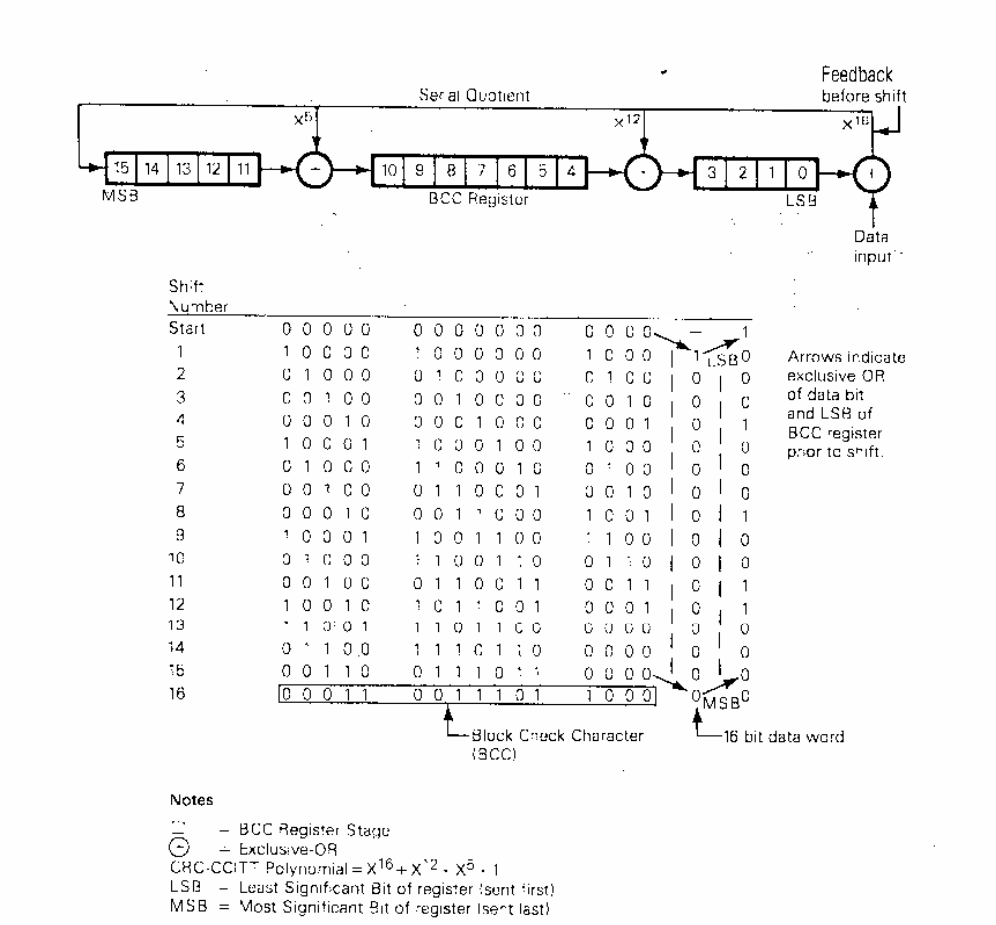


Figura 6: Cálculo de CRC usando CRC_CCITT – Seqüência de transmissão [McNamara 88]

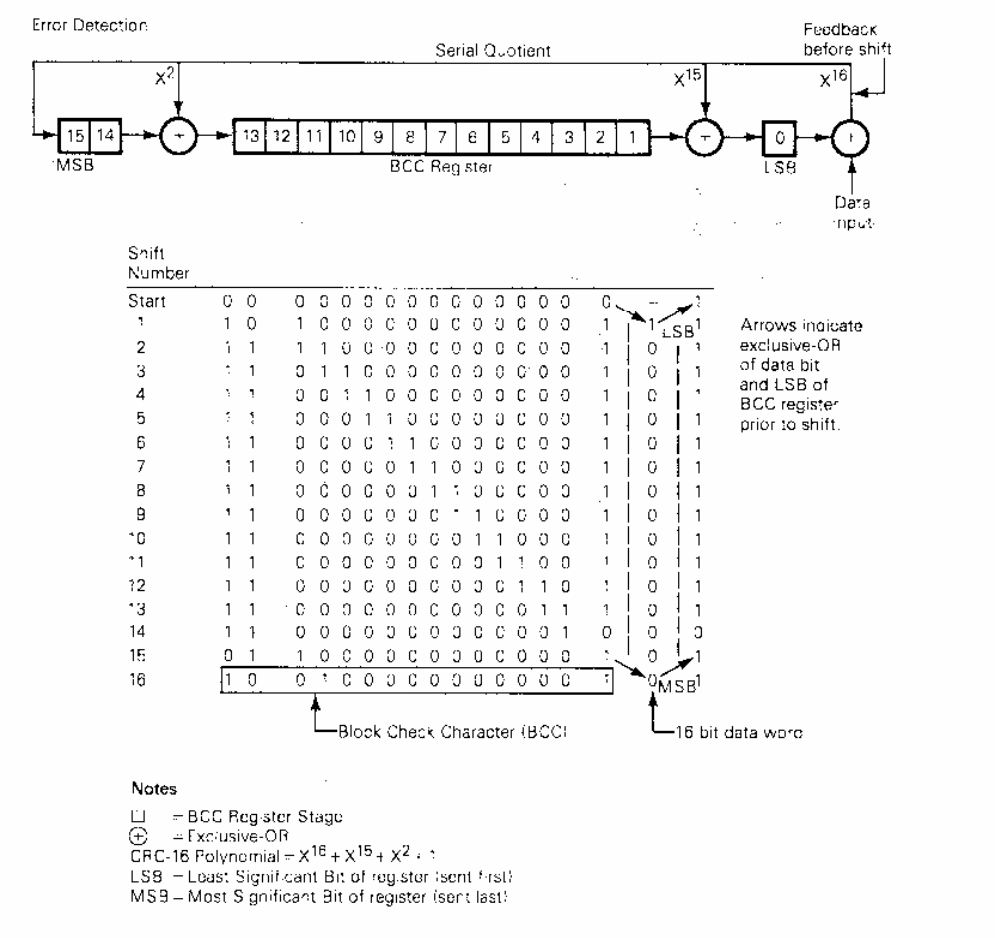


Figura 7: Cálculo de CRC usando CRC_16 – Seqüência de transmissão [McNamara 88]

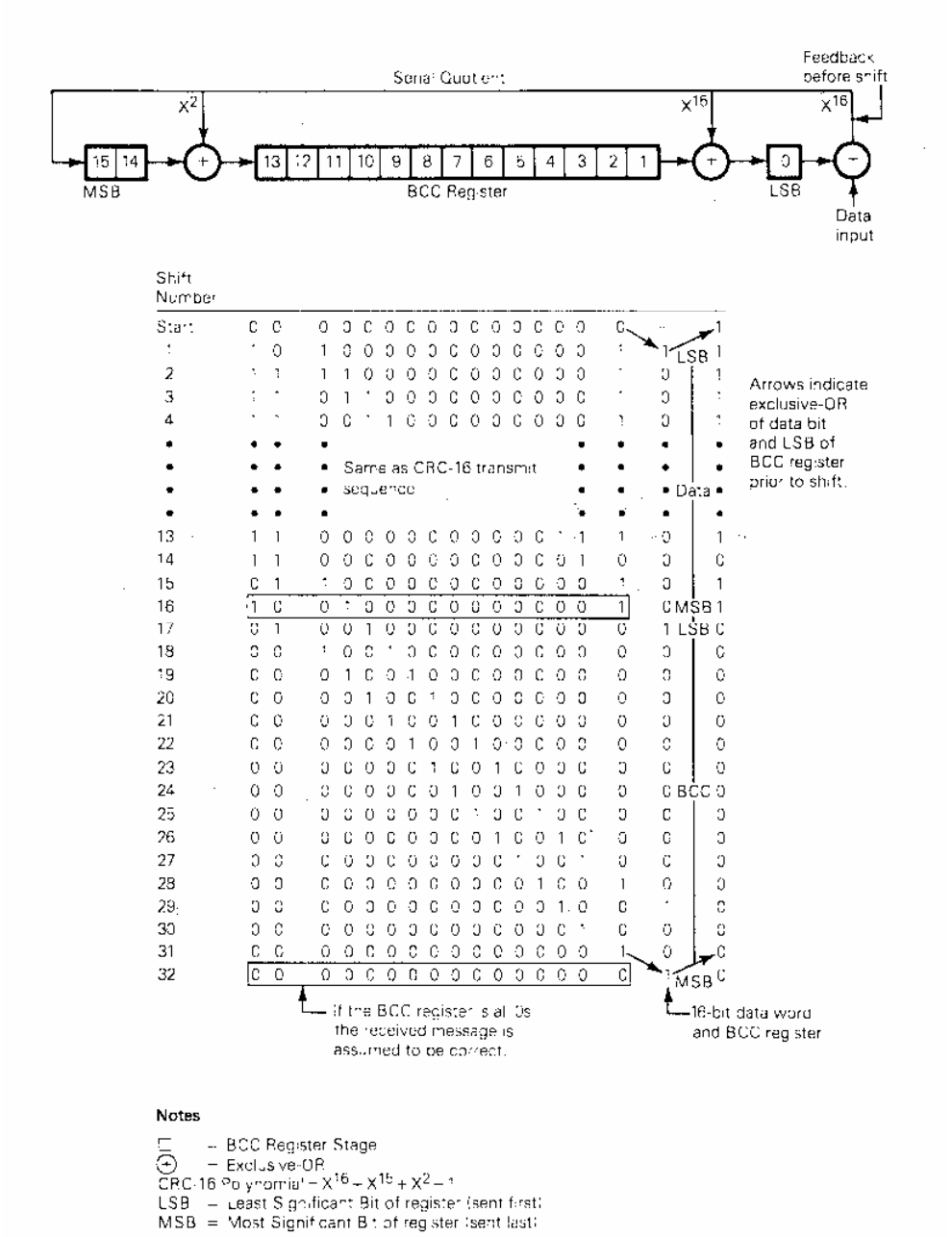


Figura 8: Cálculo do CRC na recepção

Cálculo do CRC bitwise

Os algoritmos de cálculo do CRC por software bit a bit são denominados algoritmos *bitwise*. Estes algoritmos em geral simulam a ação da implementação por hardware.

```
//      Cálculo do CRC bitwise
//      Autor: Constantino Seixas Filho
//      Data: 7/01/2001
//

#include <stdio.h>
#include <string.h>

unsigned CalcCRC(char *,int, unsigned );
unsigned CalcCRC2(char *,int, unsigned );

#define CRC_CCITT   0x8408
#define CRC_16      0xA001

char Mensagem[]= "Primeiro teste de CRC";
char Tabela[]      = {0x01, 0x00};  // Exemplo da figura 13-8  McNamara

void main()
{
        unsigned Result;

        Result = CalcCRC(Mensagem, strlen(Mensagem), CRC_CCITT);
        printf("CRC_CCITT calculado = %04x\n", Result);
        Result = CalcCRC(Tabela, 2, CRC_16);
        printf("CRC_16 calculado = %04x\n", Result);
        Result = CalcCRC2(Mensagem, strlen(Mensagem), CRC_CCITT);
        printf("CRC_CCITT calculado = %04x\n", Result);
        Result = CalcCRC2(Tabela, 2, CRC_16);
        printf("CRC_16 calculado = %04x\n", Result);
} // main


unsigned CalcCRC(char *pch, int nBytes, unsigned Operando)
{
        unsigned CRC;
        int bit0;

        CRC = 0;                                // Inicializa shift register para zero
        for (int cByte = nBytes; cByte >0; --cByte) {
                CRC ^= (*pch++ & 0x00FF); // Assegura que trabalhará com byte
                for (int cBit=8; cBit >0; --cBit) {
                        bit0 = 1 & CRC;
                        CRC >>= 1;
                        if (bit0 == 1) CRC ^= Operando;
                }
        }
        return (CRC);
}       // CalcCRC
```

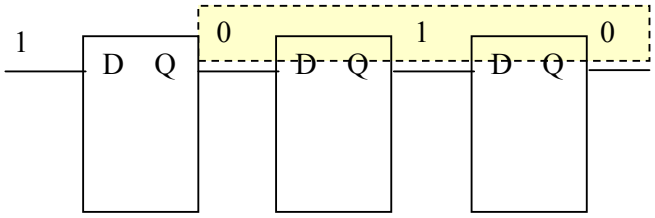
```
unsigned CalcCRC2(char *pch, int nBytes, unsigned Operando) {
unsigned CRC;
unsigned Dado;
int Bit;

        CRC = 0;    // Inicializa shift register para zero
        for (int cByte = nBytes; cByte >0; --cByte) {
                Dado = *pch++ & 0x00FF;    // Assegura que trabalhará com byte
                for (int cBit=8; cBit >0; --cBit) {
                        Bit = (Dado & 1) ^ (CRC & 1);
                        CRC >>= 1; Dado >>= 1;
                        if (Bit == 1) CRC ^= Operando;
                }
        }
        return (CRC);
}       // CalcCRC2
```

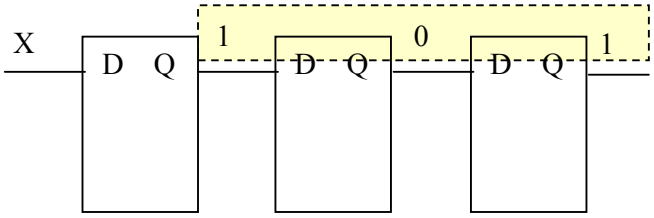
O algoritmo 2 sintetiza exatamente o algoritmo fornecido por McNamara, emulando o circuito com os registradores de deslocamento.

Observe que aplicar um clock no registrador de deslocamento eqüivale a realizar um shift para a direita do valor que representa o conteúdo do registrador e em seguida realizar um XOR do bit mais significativo do registro com o bit que alimenta a cadeia (XOR do dado com LSb do registrador):

Situação antes do pulso de clock:



Situação após o pulso de clock:



Simulação através de registrador:

0	1	0	Situação antes do clock
0	0	1	Registrador após deslocamento
1	0	1	Registrador após XOR com 100

Alimentar 1 em um registrador de deslocamento = shift right + XOR 1000...

O primeiro algoritmo é mais eficiente pois combina o byte de dados com o CRC uma única vez e depois toma a decisão de combinar o operando com o CRC apenas em função do conteúdo do CRC.

Cálculo do CRC bytewise

Um algoritmo mais eficiente foi publicado pela primeira vez na referência [Perez 83] e passou a ser adotado em todas as implementações práticas por oferecer um algoritmo muito mais eficiente (cerca de 6 vezes mais rápido, segundo minhas observações).

Vamos observar passo a passo o cálculo do CRC 16 e o conteúdo do *shif register* após cada operação:

Convenção:
Conteúdo inicial do registrador de deslocamento: $C_0..C_{15}$
Mensagem de entrada: $M_0..M_7$

a) Posição inicial:

				*		*											*
SH	**IN**	**R_{15}**	**R_{14}**	**R_{13}**	**R_{12}**	**R_{11}**	**R_{10}**	**R_9**	**R_8**	**R_7**	**R_6**	**R_5**	**R_4**	**R_3**	**R_2**	**R_1**	**R_0**
0		C_{15}	C_{14}	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0

b) Posição após primeiro passo:

SH	**IN**	**R_{15}**	**R_{14}**	**R_{13}**	**R_{12}**	**R_{11}**	**R_{10}**	**R_9**	**R_8**	**R_7**	**R_6**	**R_5**	**R_4**	**R_3**	**R_2**	**R_1**	**R_0**
1	M_0	$C_0 \oplus M_0$	C_{15}	$C_{14} \oplus C_0 \oplus M_0$	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8	C_7	C_6	C_5	C_4	C_3	C_2	$C_1 \oplus C_0 \oplus M_0$

c) Posição após dois passos:

SH	**IN**	**R_{15}**	**R_{14}**	**R_{13}**	**R_{12}**	**R_{11}**	**R_{10}**	**R_9**	**R_8**	**R_7**	**R_6**	**R_5**	**R_4**	**R_3**	**R_2**	**R_1**	**R_0**
2	M_1	$C_1 \oplus C_0 \oplus M_0 \oplus M_1$	$C_0 \oplus M_0$	$C_{15} \oplus C_1 \oplus C_0 \oplus M_0 \oplus M_1$	$C_{14} \oplus C_0 \oplus M_0$	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8	C_7	C_6	C_5	C_4	C_3	$C_2 \oplus C_1 \oplus C_0 \oplus M_0 \oplus M_1$

c) Posição após oito passos (omitindo o símbolo ⊕):

SH	IN	R_{15}	R_{14}	R_{13}	R_{12}	R_{11}	R_{10}	R_9	R_8	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0
8	M_7	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0	C_{15}	C_{14}	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
		C_6	C_5	C_4	C_3	C_2	C_1	C_0	M_0	C_1	C_0						C_7
		C_5	C_4	C_3	C_2	C_1	C_0	M_0	C_2	C_0	M_0						C_6
		C_4	C_3	C_2	C_1	C_0	M_0	M_1	C_1	M_0							C_5
		C_3	C_2	C_1	C_0	M_0	M_1	C_3	C_0	M_1							C_4
		C_2	C_1	C_0	M_0	M_1	M_2	C_2	M_0								C_3
		C_1	C_0	M_0	M_1	M_3	C_4	C_1	M_1								C_2
		C_0	M_0	M_1	M_3	C_5	C_3	C_0	M_2								C_1
		M_0	M_1	M_2	M_4	C_4	C_2	M_0									C_0
		M_1	M_2	M_3	C_6	C_3	C_1	M_1									M_0
		M_2	M_3	M_4	C_5	C_2	C_0	M_2									M_1
		M_3	M_4	M_5	C_4	C_1	M_0	M_3									M_2
		M_4	M_5	C_7	C_3	C_0	M_1										M_3
		M_5	M_6	C_6	C_2	M_0	M_2										M_4
		M_6		C_5	C_1	M_1	M_3										M_5
		M_7		C_4	C_0	M_2	M_4										M_6
				C_3	M_0	M_3											M_7
				C_2	M_1	M_4											
				C_1	M_2	M_5											
				C_0	M_3												
				M_0	M_4												
				M_1	M_5												
				M_2	M_6												
				M_3	C_1												
				M_4													
				M_5													
				M_6													
				M_7													

Realizando as simplificações:

a) Xi = Ci ⊕Mi
b) A ⊕ B = B ⊕ A (comutatividade)
c) A ⊕ B ⊕ C = A ⊕ C ⊕ B (associatividade)
d) A ⊕ A = 0 (involução)
e) A ⊕ 0 = A (elemento neutro)

Obtemos:

SH	IN	R_{15}	R_{14}	R_{13}	R_{12}	R_{11}	R_{10}	R_9	R_8	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0
8	M_7	X_7	X_6	X_7	X_6	X_5	X_4	X_3	X_2	C_{15}	C_{14}	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
		X_6	X_5	X_6	X_5	X_4	X_3	X_2	X_1	X_1	X_0						X_7
		X_5	X_4							X_0							X_6
		X_4	X_3														X_5
		X_3	X_2														X_4
		X_2	X_1														X_3
		X_1	X_0														X_2
		X_0															X_1
																	X_0

- Observe que os 8 bits menos significativos são função de $C_8..C_{15}$ e de $X_0..X_7$.
- Os 8 bits mais significativos são função de $X_0..X_7$

SH	IN	R_{15}	R_{14}	R_{13}	R_{12}	R_{11}	R_{10}	R_9	R_8	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0
8	M_7	0	0	0	0	0	0	0	0	C_{15}	C_{14}	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
		X_7	X_6	X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0						X_7
		X_6	X_5	X_6	X_5	X_4	X_3	X_2	X_1	X_0							X_6
		X_5	X_4														X_5
		X_4	X_3														X_4
		X_3	X_2														X_3
		X_2	X_1														X_2
		X_1	X_0														X_1
		X_0															X_0

Algoritmo:
Para todos os bytes da mensagem faça:

1. Calcule Xi = Low(CRC ⊕ Mensagem)
2. Deslocar o CRC oito bits para a direita.
3. Calcular o valor combinado da função dos Xis abaixo da linha horizontal
4. Realizar o ou exclusivo do CRC com o valor calculado

Observe que uma vez escolhido X (existem 256 possibilidades), o valor calculado no passo 3 fica determinado. Logo podemos pré calcular estes valores e guarda-los em uma *look up table*.

X	VALOR	X	VALOR	X	VALOR	X	VALOR	X	VALOR	X	VALOR	X	VALOR	X	VALOR
0	0000	32	D801	64	F001	96	2800	128	AO01	160	7800	192	5000	224	8801
1	COC1	33	18C0	65	30C0	97	E8C1	129	60CO	161	B8C1	193	90C1	225	48C0
2	C181	34	1980	66	3180	98	E981	130	6180	162	B981	194	9181	226	4980
3	0140	35	D941	67	F141	99	2940	131	A141	163	7940	195	5140	227	8941
4	C301	36	1B00	68	3300	100	EB01	132	6300	164	BB01	196	9301	228	4B00
5	03CO	37	DBC1	69	F3C1	101	2BCO	133	A3C1	165	7BC0	197	53CO	229	8BC1
6	0280	38	DA81	70	F281	102	2A80	134	A281	166	7A80	198	5280	230	8A81
7	C241	39	1A40	71	3240	103	EA41	135	6240	167	BA41	199	9241	231	4A40
8	C601	40	1E00	72	3600	104	EE01	136	6600	168	BE01	200	9601	232	4E00
9	06CO	41	DEC1	73	F6C1	105	2ECO	137	A6C1	169	7EC0	201	56CO	233	8EC1
10	0780	42	DF81	74	F781	106	2F80	138	A781	170	7F80	202	5780	234	8F81
11	C741	43	1F40	75	3740	107	EF41	139	6740	171	BF41	203	9741	235	4F40
12	0500	44	DD01	76	F501	108	2D00	140	A501	172	7DOO	204	5500	236	8D01
13	C5C1	45	1DC0	77	35CO	109	EDC1	141	65CO	173	BDC1	205	95C1	237	4DC0
14	C481	46	1C80	78	3480	110	EC81	142	6480	174	BC81	206	9481	238	4C80
15	044D	47	DC41	79	F441	111	2C40	143	A441	175	7C40	207	5440	239	8C41
16	CC01	48	1400	80	3C00	112	E401	144	6C00	176	B401	208	9C01	240	4400
17	0CCO	49	D4C1	81	FCC1	113	24CO	145	ACC1	177	74C0	209	5CC0	241	84C1
18	0D80	50	D581	82	FD81	114	2580	146	AD81	178	7580	210	5D80	242	8581
19	CD41	51	1540	83	3D40	115	E541	147	6D40	179	B541	211	9D41	243	4540
20	0F00	52	D701	84	FF01	116	2700	148	AF01	180	7700	212	5F00	244	8701
21	CFC1	53	17CO	85	3FCO	117	E7C1	149	6FCO	181	B7C1	213	9FC1	245	47C0
22	CE81	54	1680	86	3E80	118	E681	150	6E80	182	B681	214	9E81	246	4680
23	0E40	55	D641	87	FE41	119	2640	151	AE41	183	7640	215	5E40	247	8641
24	0A00	55	D201	88	FA01	120	2200	152	AA01	184	7200	216	5A00	248	8201
25	CAC1	57	12CO	89	3ACO	121	E2C1	153	6ACO	185	B2C1	217	9AC1	249	42C0
26	CB81	58	1380	90	3B80	122	E381	154	6B80	186	B381	218	9B81	250	4380
27	0B40	59	D341	91	FB41	123	2340	155	AB41	187	7340	219	5B40	251	8341
28	C901	60	1100	92	3900	124	E101	156	6900	188	B101	220	9901	252	4100
29	09CO	61	D1C1	93	F9C1	125	21CO	157	A9C1	189	71CO	221	59CO	253	81C1
30	0880	62	D081	94	F881	126	2080	158	A881	190	7080	222	5880	254	8081
31	C841	63	1040	95	3840	127	E041	159	6840	191	B041	223	9841	255	4040

Tabela 1: Tabela de operandos para cálculo de CRC16 (valores calculados abaixo da linha horizontal)

Propriedades:

Da observação de como Tab[X] é calculado acima, podemos tirar algumas conclusões. Estamos supondo que o valor inicial do CRC é 0.

a) Observe que Tab[X] = CRC (X) onde X é um valor correspondendo a um byte: X_7 .. X_0. X varia de 0 a 255.

b) CRC(0) = Tab[0] = 0, independente do polinômio, pois o resto da divisão de 0 por qualquer polinômio é 0.

c) CRC(0xFF) = Tab[0xFF] = T_{15} .. T_0, onde T_i = XOR $_{\text{Número\_Par\_de\_Termos}}$ (1) = 0, ou T_i = XOR $_{\text{Número\_Impar\_de\_Termos}}$ (1) = 1. Observe que para o CRC16, CRC(0xFF) terá valor 1 apenas nos bits nas posições 6 e 14 onde o número de termos X_j combinantes é ímpar. Portanto CRC16(0xFF) = 0x4040.

d) CRC(not M) = CRC(M ρ 0xFF) = CRC(M) ρ CRC(0xFF), onde M é uma mensagem de um byte.
Imagine que conhecemos o CRC(M) = T_{15} ..T_0. O CRC(not M) terá o mesmo valor do CRC de M para os bits em que o número de termos de X_i = M_i for par e terá o valor complementar ao de M onde o número de termos de X_i = M_i for ímpar. A máscara que determina onde o número de bits combinantes de X_i é para ou ímpar é exatamente o CRC(0xFF). Logo devemos trocar os bits do CRC(M) nestas posições onde CRC(0xFF) tem um bit igual a 1, ou seja basta realizar o ou exclusivo de CRC(M) com o CRC(0xFF).
Exemplo:
Seja calcular o CRC(254).
CRC(254) = CRC(1) ρ CRC(0xFF) = 0xC0C1 ρ 0x4040 = 0x8081.
Esta propriedade implica que precisamos calcular apenas metade das posições da tabela, pois a outra metade é determinada diretamente pela equação acima.

X	VALOR	X	VALOR	X	VALOR	X	VALOR	X	VALOR	X	VALOR	X	VALOR	X	VALOR
0	0000	32	2102	64	4204	96	6306	128	8408	160	A50A	192	C60C	224	E70E
1	1189	33	308B	65	538D	97	728F	129	9581	161	B483	193	D785	225	F687
2	2312	34	0210	66	6116	98	4014	130	A71A	162	8618	194	E51E	226	C41C
3	329B	35	1399	67	70F9	99	519D	131	B693	163	9791	195	F497	227	D595
4	4624	36	6726	68	0420	100	2522	132	C22C	164	E32E	196	8028	228	A12A
5	57AD	37	76AF	69	15A9	101	34AB	133	D3A5	165	F2A7	197	91A1	229	B0A3
6	6536	38	4434	70	2732	102	0630	134	E13E	166	C03C	198	A33A	230	8238
7	74BF	39	55BD	71	36BB	103	17B9	135	F0B7	167	D1B5	199	B2B3	231	93B1
8	8C49	40	AD4A	72	CE4C	104	EF4E	136	0840	168	2942	200	4A44	232	6B46
9	9DC1	41	BCC3	73	DFC5	105	FEC7	137	19C9	169	38CB	201	5BCD	233	7ACF
10	AF5A	42	8E58	74	ED5E	106	CC5C	138	2B52	170	0A50	202	6956	234	4854
11	BED3	43	9FD1	75	FCD7	107	DDD5	139	3ADB	171	1BD9	203	78DF	235	59DD
12	CA6C	44	EB6E	76	8868	108	A96A	140	4E64	172	6F66	204	0C60	236	2D62
13	DBE5	45	FAE7	77	99E1	109	B8E3	141	5FED	173	7EEF	205	1DE9	237	3CEB
14	E97E	46	C87C	78	AB7A	110	8A78	142	6D76	174	4C74	206	2F72	238	0E70
15	F8F7	47	D9F5	79	BAF3	111	9BF1	143	7CFF	175	5DFD	207	3EFD	239	1FF9
16	1081	48	3183	80	5285	112	7387	144	9489	176	B58B	208	D68D	240	F78F
17	0108	49	200A	81	430C	113	620E	145	8500	177	A402	209	C704	241	E606
18	3393	50	1291	82	7197	114	5095	146	B79B	178	9699	210	F59F	242	D49D
19	221A	51	0318	83	601E	115	411C	147	A612	179	8710	211	E416	243	C514
20	56A5	52	77A7	84	14A1	116	35A3	148	D2AD	180	F3AF	212	90A9	244	B1AB
21	472C	53	662E	85	0528	117	242A	149	C324	181	E226	213	8120	245	A022
22	75B7	54	54B5	86	37B3	118	16B1	150	F1BF	182	D0DB	214	B3BB	246	92B9
23	643E	55	453C	87	263A	119	0738	151	E036	183	C134	215	A232	247	8330
24	9CC9	56	BDCD	88	DECD	120	FFCF	152	18C1	184	39C3	216	5AC5	248	7BC7
25	8D40	57	AC42	89	CF44	121	EE46	153	0948	185	284A	217	4B4C	249	6A4E
26	BFDB	58	9ED9	90	FDDF	122	DCDD	154	3BD3	186	1AD1	218	79D7	250	58D5
27	AE52	59	8F50	91	EC56	123	CD54	155	2A5A	187	0B58	219	685E	251	495C
28	DAED	60	FEEB	92	98E9	124	B8EB	156	5EE5	188	7FE7	220	1CE1	252	3DE3
29	CB64	61	EA66	93	8960	125	A862	157	4F6C	189	6E6E	221	0D68	253	2C6A
30	F9FF	62	D8FD	94	BBFB	126	9AF9	158	7DF7	190	5CF5	222	3FF3	254	1EF1
31	E876	63	C974	95	AA72	127	8B70	159	6C7E	191	4D7C	223	2E7A	255	0F78

Tabela 2: Tabela de operandos para cálculo de CRC_CCITT

O algoritmo final fica:

Algoritmo final:

Para todos os bytes da mensagem faça:

1. Calcule Xi = Low(CRC ⊕ Mensagem).
2. Deslocar o CRC oito bits para a direita.
3. Realizar o ou exclusivo do CRC com o valor da tabela indexado por X.

Esta tabela pode ser calculada para qualquer polinômio automaticamente através de um programa, que é mostrado no exemplo completo que se segue:

```
//      Cálculo de CRC-CCITT byte-wise
//
//      Autor: Constantino Seixas Filho
//
//      Data: 19/01/92
//

#include <stdio.h>
#include <string.h>

// ---------------------- Protótipos de funções ----------------------------
unsigned MakeOper(int *);
```

```
unsigned CalcCRCBitwise(char *, int , unsigned );
unsigned CalcCRC(char *, int );
void GeraTabCrcCCITT(void);

// -------------------------- Definições de Operandos Típicos  ----------------------------
#define   CRC_CCITT    0x8408

// ------------------------- Variáveis Globais ------------------------------
char tabela[]= "ola como vai tudo bem ?";
char string[]= "Segundo teste de CRC";
unsigned operando = CRC_CCITT;          // p(x)= x16 + x12 + x5 + 1
unsigned tab[256];    // resultados parciais para calculo do crc byte-wise

// ---------------------------------------------------------------------------
int px[17]=
//  16  15  14  13  12  11  10  09  08  07  06  05  04  03  02  01 00
// x   x   x   x   x   x   x   x   x   x   x   x   x   x   x   x   x
 { 1,  0,  0,  0,  1,  0,  0,  0,  0,  0,  0,  1,  0,  0,  0,  0,  1 };

// ---------------------------------------------------------------------------
void main()
{
int result;

// teste da geração da tabela auxiliar
operando = MakeOper(px);
printf("\noperando = %04x", operando);
GeraTabCrcCCITT();

// teste do cálculo do CRC
result = CalcCRC(tabela, strlen(tabela));
printf("\nString 1: byte wise result = %04x", result);
result = CalcCRC(string, strlen(string));
printf("\nString 2: byte wise result = %04x\n", result);
result = CalcCRCBitwise(tabela, strlen(tabela), operando);
printf("\nString 1: bit wise result = %04x", result);
result = CalcCRCBitwise(string, strlen(string), operando);
printf("\nString 2: bit wise result = %04x\n", result);
} // main

// ---------------------------------------------------------------------------------------
// Gera Operando ser utilizado nos algoritmos a partir do Polinômio P(x)
// ---------------------------------------------------------------------------------------
unsigned MakeOper(int *px)
{
   unsigned operando;

   operando = 0;
   // inverte ordem dos bits na palavra e despreza x16
  for (int index=16; index > 0; --index)
        operando = (operando << 1) | (*(px+index));
   return(operando);
} // MakeOper

// ---------------------------------------------------------------------------
//  Gera tabela auxiliar para cálculo de CRC-CCITT
// ---------------------------------------------------------------------------
void GeraTabCrcCCITT(void)
//  gera tabela auxiliar para calculo deCRC CCITT
//  P(x) = x^16  + x^12  + x^5  + 1
```

```
//
//  CRC_CCITT = 0x8408
//
{
static unsigned v[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
unsigned x1, x2, x3, x4, x5, x6, x7, x8;
unsigned x;
int i, cont;

i = 0;
for (x8=0; x8<2; x8++)
        for (x7=0; x7<2; x7++)
                for (x6=0; x6<2; x6++)
                        for (x5=0; x5<2; x5++)
                                for (x4=0; x4<2; x4++)
                                        for (x3=0; x3<2; x3++)
                                                for (x2=0; x2<2; x2++)
                                                for (x1=0; x1<2; x1++) {
                                                    x = x8 ^ x7 ^ x6 ^ x5 ^ x4 ^ x3 ^ x2 ^ x1;
                                                    v[15]= x8 ^ x4;
                                                    v[14]= x7 ^ x3;
                                                    v[13]= x6 ^ x2;
                                                    v[12]= x5 ^ x1;
                                                    v[11]= x4;
                                                    v[10]= x8 ^ x4 ^ x3;
                                                    v[ 9]= x7 ^ x3 ^ x2;
                                                    v[ 8]= x6 ^ x2 ^ x1;
                                                    v[ 7]= x5 ^ x1;
                                                    v[ 6]= x4;
                                                    v[ 5]= x3;
                                                    v[ 4]= x2;
                                                    v[ 3]= x8 ^ x4 ^ x1;
                                                    v[ 2]= x7 ^ x3;
                                                    v[ 1]= x6 ^ x2;
                                                    v[ 0]= x5 ^ x1;
                                                    tab[i]=0;
                                                    for (cont=15; cont >= 0; --cont)
                                                      tab[i] = (tab[i] << 1) | v[cont];
                                                    ++i;
                                                  } // for
} // GeraTabCrcCCITT

// ------------------------------------------------------------------------------
//  Gera tabela auxiliar para calculo de CRC, dado o polinômio divisor
// ------------------------------------------------------------------------------
void GeraTabCRC(unsigned operando)
{
// deveria fazer um xor com o conteúdo inicial do CRC suposto igual a 0
  for (int index = 0; index < 256; ++index)
        tab[index] = CalcCRCBitwise((char *) &index, 1, operando);

} // GeraTabCRC

// ------------------------------------------------------------------------
// Calcula CRC bytewise
// ------------------------------------------------------------------------
unsigned CalcCRC(unsigned char *pch, int n_bytes)
{
  register unsigned crc;
  unsigned index;
```

```
  crc = 0;
  for (int cont=n_bytes; cont > 0; --cont)   {
    index = (crc ^ *pch++) & 0x00FF;
    crc = (crc >> 8) ^ tab[index];
  }
  return(crc);
}  // CalcCRC


// ---------------------------------------------------------------------
// Calcula CRC bitwise
// ---------------------------------------------------------------------
unsigned CalcCRCBitwise(unsigned char *pch, int nBytes, unsigned Operando)
{
        unsigned CRC;
        int bit0;

        CRC = 0;                                // Inicializa shift register para zero
        for (int cByte = nBytes; cByte >0; --cByte) {
        CRC ^=  *pch++;
                for (int cBit=8; cBit >0; --cBit) {
                        bit0 = 1 & CRC;
                        CRC >>= 1;
                        if (bit0 == 1) CRC ^= Operando;
                }
        }
        return (CRC);
}       // CalcCRCBitwise
```

CRC-32

Este polinômio possui maior capacidade de detecção de erros que os polinômios de 16 bits, sendo usado na rede Ethernet, WinZip e PKZIP, etc.
O polinômio utilizado é:

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

```
static unsigned long crc32_table[256];

void gen_table(void)
{
    unsigned long crc, poly;
    int i, j;

    poly = 0xEDB88320L;
    for (i = 0; i < 256; i++)
        {
        crc = i;
        for (j = 8; j > 0; j--)
            {
            if (crc & 1)
                crc = (crc >> 1) ^ poly;
            else
                crc >>= 1;
            }
        crc32_table[i] = crc;
    }
```

```
}


unsigned long calc_crc32(unsigned char *pch, int nBytes)
{
    register unsigned long crc;

    crc = 0xFFFFFFFF;
    for (int cByte = nBytes; cByte >0; --cByte) {
        crc = (crc>>8) ^ crc32_table[ (crc ^ *pch) & 0xFF];

    return( crc^0xFFFFFFFF);
}
```

Revertendo o CRC

Este é um assunto academicamente interessante, principalmente se você for um hacker :) . Vamos discutir como alterar um conjunto de bytes em um string de bytes de modo que o CRC não seja alterado.

Este problema pode ser formulado da seguinte maneira:
Considere que uma mensagem possui N bytes e que o seu CRC calculado utilizando o polinômio de 16 bits P_{16} é dado por:

$CRC_{16}(M_n) = K$

Mensagem Original

N		k+x	k+x-1	k+x-2	k+x-3			k	k-1		4	3	2	1
b_n											b_4	b_3	b_2	b_1
									CRC = K1					
			CRC = K2											
CRC = K														

Parte desta mensagem será substituída por x novos bytes a partir da posição k:

Mensagem Alterada

N		k+x	k+x-1	k+x-2	***	k+2	k+1	k	k-1		4	3	2	1
b_n			m_x	m_{x-1}	***	m_3	m_2	m_1			b_4	b_3	b_2	b_1
parte não modificada			y	x					parte não modificada					
			patch introduzido						CRC = K1					

Os últimos bytes da modificação, posições mx-1 e mx conterão dois bytes de ajuste que chamaremos de b e a respectivamente.

O problema consiste em calcular a e b de tal forma que o CRC final da mensagem seja K.

Considerações:

1. Evidentemente o CRC da posição 1 até a posição k- 1 é o mesmo para as duas mensagens. Vamos chamá-lo de K1.
2. A influência dos bytes da posição k+x até a posição N será a mesma nos dois casos.
3. Temos que fazer com que os CRCs ao chegar n aposição k+x+1 seja o mesmo nos dois strings. O valor inicial no registrador de CRC ao chegar na posição k será K1.

Ao calcular o CRC da mensagem modificada, ao chegar em k+x-3 teremos:
$CRC\ (b_1..b_{k-1}, m_1..m_{x-2}) = R = R_H \mid R_L$

Onde o símbolo | indica concatenação. R_H é o byte mais significativo no CRC e R_L o byte menos significativo.

Para conservar o valor do CRC devemos ter:

$R \circ x \circ y = K2$

O operador "$\circ$" indica uma combinação segundo o algoritmo do cálculo do CRC do valor do registrador quando acabamos de processar o patch com os bytes x e y em seqüência.

Como calcular x e y ?

Nós conhecemos R e K2.

Segundo o algoritmo *bytewise* que deduzimos temos:

Após processar o byte x:
$Temp = (R >> 8) \oplus Tab[(R \oplus x) \& 0xFF]$
$Temp = R_H \oplus Tab[(R_L \oplus x)]$

Vamos supor que a posição apontada por $R_L \oplus b$ contenha o dado: $b_H \mid b_L$

$Temp = b_H \mid R_H \oplus b_L$

Após processar o byte y:
$CRC = (Temp >> 8) \oplus Tab[(Temp \oplus y) \& 0xFF] = K2$
$CRC = b_H \oplus Tab[R_H \oplus b_L \oplus y] = K2$

Vamos supor que a posição apontada por $R_L \oplus b_L \oplus y$ contenha o dado: $c_H \mid c_L$

$CRC = c_H \mid b_H \oplus c_L = K2 = K2_H \mid K2_L$

Logo pela equação acima nós deduzimos o valor de c_H :

$\boxed{c_H = K2_H}$

Sabendo este valor nós podemos procurar na tabela por uma entrada de índice I_c tal que o seu bytes mais significativo seja o valor desejado ($K2_H$). Assim determinamos c_L.

Como $b_H \oplus c_L = K2_L$ temos que $b_H \oplus c_L \oplus c_L = K2_L \oplus c_L$

Logo daí determinamos: $\boxed{b_H = K2_L \oplus c_L}$

Devemos novamente procurar na tabela por uma entrada de índice I_b cujo byte mais significativo coincida com b_H.
Desta forma b_L também fica determinado.

Já conhecemos b e c e também os índices destas posições: I_b e I_c.

$R_H \oplus b_L \oplus y = I_c$

Logo $\boxed{y = b_L \oplus R_H \oplus I_c}$

$R_L \oplus x = I_b$

Logo $\boxed{x = R_L \oplus I_b}$

Exercícios

1) Demonstre que a soma módulo 2 é equivalente à subtração módulo 2.

2) Demonstre que a operação ou-exclusivo é comutativa, associativa e tem elemento neutro.

3) Considere a seguinte mensagem:

   M = 1010010001 que será transmitida MSB primeiro.
   O polinômio gerador é : $P(X) = 1 + X^2 + X^4 + X^5$
   a) Calcule o polinômio G(X) correspondente à mensagem.
   b) Calcule o CRC.
   c) Calcule a mensagem final.
   d) Refaça os cálculos considerando a transmissão do LSB primeiro.

4) Calcule o circuito para gerar o CRC relativo ao polinômio: $P(X) = 1 + X^2 + X^4 + X^5$. Considere a transmissão do LSB primeiro.

   Calcule o CRC para a mensagem da questão 2 simulando passo a passo. Confira os resultados.

5) Explique a utilidade da função MakeOper mostrada no programa de cálculo de CRC.

6) Calcule a Tabela 2 para cálculo do CRC *bytewise* para o CRC_CCITT.

7) Uma mensagem m=101110101000 foi transmitida em um canal de comunicação. À esta mensagem foi anexado o CRC gerado pelo polinômio: $P(X) = X^3 + X^2 + 1$. Durante a transmissão ocorreu um erro que não foi detectado na recepção pelo algoritmo de CRC. Sugira um possível polinômio representando o padrão de bits do erro.

8) Questão do provão 2000:
   A camada de enlace de dados de uma estação de rede recebeu a seqüência de bits abaixo:

111001101110

   Considerando que a técnica de detecção de erros adotada é a CRC ("*Cyclic Redundancy Check*"), e que o polinômio gerador utilizado é:

   $$G(x) = x^4 + x^3 + 1,$$

   verifique se os dados serão aceitos pelo receptor como corretos. Justifique sua resposta **(valor: 10,0 pontos)**

9) Observe o circuito que se segue e responda:

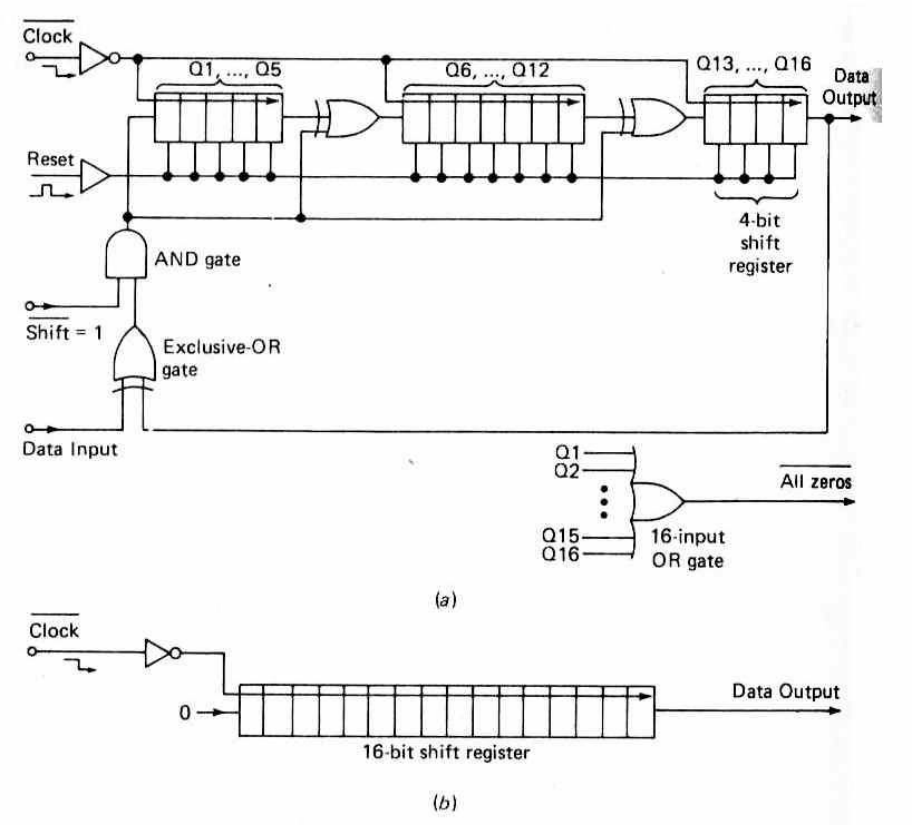


Figura 9 – Circuito de cálculo de CRC Motorola MC8503
(segundo *Peatman – Microcomputer based design*)

a) Como é a operação do circuito se Shift* = 1 ?

b) Como é a operação do circuito se Shift* = 0 ?

c) Resuma como você controlaria a operação deste circuito para calcular o CRC e logo após apendar o valor do CRC ao final da mensagem.

d) Na recepção antes dos 16 últimos clocks serem aplicados e supondo que a transmissão foi correta, o valor do registador coincidirá com o valor dos próximos 16 bits sendo recebidos. Explique porque a chegada dos últimos 16 bits causa o preenchimento do registrador com 16 zeros.

e) Qual o valor do operando para o nosso algoritmo de CRC que este circuito implementa ?

f) Qual o valor do polinômio divisor ?

10) Existe diferença entre calcular o CRC-16 com o registro de CRC inicializado para 0 ou para 0xFFFF ? Você é capaz de prever este resultado ? Tente com uma mensagem qualquer e comente o resultado.

11) A mensagem M foi enviada por enviada por um canal de comunicação:
M = The quick brown fox jumps over the lazy dog

a) Calcule O CRC-16 desta mensagem.

b) Você deve trocar a expressão *brown fox* por *mad cat*. Depois deve calcular dois bytes extras para serem apendados ao final da mensagem de tal forma que o CRC não se altere.

12) Desenvolva uma calculadora de CRC didática com as seguintes funcionalidades:

a) O polinômio pode ser escolhido pelo usuário. Vários polinômios clássicos dever estar pré cadastrados.

b) A calculadora calcula o CRC de um string, bytes avulsos introduzidos na janela ou de bytes em um arquivo dado.

c) Ela informa o tamanho da mensagem e o valor do CRC em hexadecimal.

d) O valor inicial do registro de CRC pode ser escolhido pelo usuário.

e) O simulador desenha o circuito emulador de divisão e mostra o seu conteúdo.

f) O usuário pode processar a mensagem de uma única vez ou byte a byte ou bit a bit. O sinal de realimentação do circuito ($X0 = C0 \oplus M0$) é exibido a cada passo.

Programe esta aplicação em Delphi.

13) A tabela 1 e 2 mostram os valores pré calculados do CRC de 0 a 255 que são usados no cálculo do CRC *bytewise*. O que é necessário modificar nas tabelas se estamos calculando o CRC modificado, isto é o CRC para um valor inicial do registrador de CRC de 0xFFFF ?

14) Calcule o CRC modificado de 0x75.

15) Calcule o CRC *bytewise* da mensagem: "EC&A". Não considere o caracter *null* ao final da mensagem. O primeiro caracter a ser processado é o 'E'.

Bibliografia

[Morse 86] Greg Morse. Calculating CRCs by bits and bytes. BYTE, September 1986, Pg 115..124.

[Mc Namara 88] John McNamara. Technical Aspects of Data Communication, 3rd edition, 1988, Digital Equipment Corporation.

[Perez 83] Perez, Wizmer & Becker. Byte-wise CRC Calculations, IEEE Micro, June 1983.

[Peterson 61] W.W.Peterson. Cyclic Codes for Error Detection, Proceedings of the IRE.January 1961 pp 228..235

[Kohavi 78] Zvi Kohavi, Switching and finite automata theory, 2nd edition, TATA Mc Graw Hill, 1978.

Sites a Visitar

CRC calculator www.efg2.com/Lab/Mathematics/CRC.htm