

QONTINUUM

BOLETIN TECNICO DE PRODUCTO

Código: BTP011

Título: CRC: una ayuda para la detección de errores

Revisión:

Fecha: 30-8-1997

Indice:

<u>CAPITULO</u>	<u>PAG.</u>
1 PRESENTACION	3
2 GENERALIDADES	4
3 USO PRACTICO	7

Observaciones:

Como norma general de interpretación de este documento, toda palabra, acrónimo o frase realizada en **negrilla** que no esté subrayada tiene su explicación en el capítulo GLOSARIO DE TERMINOS de este documento y/o de otro cuando así se indique, mientras que las palabras, acrónimos o frases que se inicien o se escriban totalmente con mayúsculas o entre apóstrofes hacen referencia a cosas o conceptos que se presume que son del conocimiento de los lectores a los que se dirige este documento (tanto por ser de uso común como por estar explicadas en el propio documento), quedando los entrecorillados como indicación de sentido virtual o de sentido circunstancial.

QONTINUUM PLUS, s.l. se reserva el derecho de modificar todas o cualquiera de las especificaciones que se indican en este documento sin previo aviso.

Tanto el contenido íntegro de este documento como los productos reales existentes y/o resultantes a los que se alude constituyen una obra colectiva formada por las aportaciones de los técnicos asignados, directa o indirectamente, por QONTINUUM PLUS, s.l. a cada proyecto, siendo propiedad de QONTINUUM PLUS, s.l. los derechos de propiedad intelectual sobre los programas y los productos electrónicos realizados bajo la iniciativa y coordinación de ésta, de acuerdo con el artículo 8 de la Ley de Propiedad Intelectual.

R	FECHA	PAGINA/S	OBSERVACIONES
	30-8-1997	(total)	- 1ª edición

1 PRESENTACION

En general, las series de datos o paquetes pueden ser verificados de diversas maneras. Una de las más comunes es la de incluir un bit adicional en cada paquete que ayude a comprobar si ha ocurrido un error. Por ejemplo, para paquetes de ocho bits que contengan caracteres ASCII, se añade un noveno bit que se valorará a 0 ó 1 para que el total de unos del paquete sea siempre un número impar (es la misma técnica empleada para los paquetes de 4 bits que componen la pista 2 de las tarjetas de banda magnética). Esta simple inclusión de un bit permite comprobar los paquetes a su recepción simplemente contando los bits que son unos y comprobando que sean un número impar. Este sistema está limitado a la detección, en un paquete, del cambio de un número impar de bits, pero resulta ineficaz cuando el cambio afecta a un número par de bits dado que entonces el número de unos contados sigue siendo impar.

Verificaciones algo más robustas se consiguen utilizando métodos derivados del simple control de paridad por paquete, como son la 'verificación del dígito de control' (en acrónimo inglés CDV) o la 'comprobación de redundancia longitudinal' (en acrónimo inglés LRC), implementados ambos en las tarjetas bancarias de banda magnética. Aunque tales métodos se siguen usando masivamente no hay que olvidar que es muy fácil para un terminal bancario pedir al usuario que presente de nuevo su tarjeta cuando se ha detectado un error, pero que no lo es tanto para un sistema de telecomunicaciones donde el compromiso con la velocidad y el flujo global de información (redes locales, redes públicas de conmutación de paquetes, etc.) es muy grande.

Para paliar las limitaciones apuntadas se han estudiado y desarrollado diferentes sistemas, siendo uno de los que mejores resultados ofrece el llamado 'comprobación de redundancia cíclica', conocido generalmente por su acrónimo en inglés CRC.

Aunque quizá el más conocido de los CRC utilizados sea el homologado por el ITU-TSS (anteriormente conocido por CCITT), existen variantes que, manteniendo el mismo estándar de seguridad, facilitan su implementación directa a nivel de Hardware, lo cual hace que se utilicen de manera preferente.

Dado que el fundamento para calcular el CRC es el mismo para todas las variantes, este texto se basará en el CRC_ITU-TSS para introducir al lector en el cálculo de los CRC polinomiales.

2 GENERALIDADES

Para realizar el cálculo del CRC, los bits que forman la cadena a verificar se consideran formando un polinomio de grado n (siendo n el n° de bits - 1 de la trama) que se divide por otro polinomio. El resto de la división es el CRC calculado.

Sabiendo que un polinomio se representa de la forma

$$a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x^1 + a_0 x^0$$

y siendo a_i el coeficiente i -ésimo y x^i la incógnita de grado i , la obtención de un polinomio a partir de una cadena de bits consiste en multiplicar cada bit por la incógnita de grado n que le corresponda según su posición en la cadena de bits.

Ejemplo con 3 Bytes de una cadena:

10010100 10101011 00010001

Se tienen 24 bits ($3 \cdot 8$), de modo que se obtiene un polinomio de grado $n = 23$:

$$1 \cdot x^{23} + 0 \cdot x^{22} + 0 \cdot x^{21} + 1 \cdot x^{20} + 0 \cdot x^{19} + 1 \cdot x^{18} + \dots + 1 \cdot x^4 + 0 \cdot x^3 + 0 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0$$

simplificando, se obtiene el polinomio definitivo:

$$x^{23} + x^{20} + x^{18} + x^{15} + x^{13} + x^{11} + x^9 + x^8 + x^4 + 1$$

El polinomio divisor utilizado en el CRC_ITU-TSS es el $x^{16} + x^{12} + x^5 + x^0$, cuya cadena de bits es 10001000000100001.

Para calcular el CRC de una secuencia de n Bytes se considera a la secuencia como una sola cadena de $n \cdot 8$ bits seguidos de 16 bits a cero, siendo el CRC el resultado de dividir dicha cadena de bits por el polinomio divisor en módulo 2.

La división en módulo 2 significa que allí donde en una división normal se sumaría con acarreo, en módulo 2 se ignora dicho acarreo. Al trabajar en binario esto significa que las sumas se simplifican a una XOR de bits, como se verá.

EJEMPLO

Calcular el CRC sobre un solo Byte:

ASCII	Decimal	Hexadecimal	Binario
A	65	41h	01000001

Añadir 16 0's: 010000010000000000000000

División larga en módulo 2:

	Polinomio Divisor
010000010000000000000000 / 10001000000100001	
000000000000000000000000.....	

.100000100000000000000000.....	
.10001000000100001.....	

..00010100001000010.....	
..000000000000000000000000.....	

...00101000010000100....	
...000000000000000000000000....	

....01010000100001000...	
....000000000000000000000000...	

.....10100001000010000..	
.....10001000000100001..	

.....01010010001100010.	
.....00000000000000000000.	

.....10100100011000100	
.....10001000000100001	

.....0101100011100101 ----> Resto (CRC)	
.....	
01000101 -----> Cociente	

A continuación se presentan dos funciones para calcular el CRC del ITU-TSS, una en lenguaje C y otra en ASSEMBLER del i8086.

```
#define CRC_ITU-TSS (unsigned)0x1021 /* Polinomio Divisor */

unsigned crc_ITU-TSS_C ( char byte, unsigned crc )
{
  static unsigned dato;
  static int c;

  for ( dato= (int)byte << 8, c= 8; c > 0; c--, dato <<= 1 )
    crc= ((dato^crc) & 0x8000)? ((crc << 1)^CRC_ITU-TSS) : (crc << 1);

  return ( crc );
}
```

```
-----

unsigned crc_ITU-TSS_ASM ( char dato, unsigned crc )
{
  asm  push ax
  asm  push cx
  asm  push dx
  asm  xor ax, ax
  asm  mov ah, dato
  asm  mov cx, crc
  asm  xor ah, ch
  asm  mov dh, ah
  asm  ror ah, 4
  asm  and ah, 0Fh
  asm  xor ah, dh
  asm  mov dl, ah
  asm  ror ah, 3
  asm  mov dh, ah
  asm  and ah, 1Fh
  asm  xor ah, cl
  asm  mov ch, ah
  asm  mov ah, dh
  asm  and ah, 0E0h
  asm  xor ah, dl
  asm  mov cl, ah
  asm  mov ah, dh
  asm  ror ah, 1
  asm  and ah, 0F0h
  asm  xor ah, ch
  asm  mov ch, ah
  asm  mov crc, cx
  asm  pop dx
  asm  pop cx
  asm  pop ax
  return crc;
}
```

3 USO PRACTICO

Los protocolos de comunicación Q-II y Q-III utilizan dos CRC para la detección de errores: uno de ocho bits llamado CRC_8 y otro de 16 bits llamado CRC_16 (este último no debe ser confundido con el CRC_ITU-TSS, el cual, aunque también es de 16 bits, utiliza un polinomio divisor distinto).

La diferencia básica entre ambos CRC, además de su distinta longitud, es que utilizan polinomios divisores también distintos:

CRC_8 Polinomio divisor: $x^8+x^5+x^4+1$.

CRC_16 Polinomio divisor: $x^{16}+x^{15}+x^2+1$.

A continuación se muestran las dos funciones utilizadas realmente para el cálculo del CRC_8 y del CRC_16.

```

//=====
// Funciones para calcular el CRC_8 y el CRC_16
//
// Ejemplo: calcular el CRC de la cadena { 0x9B, 0xF1, 0x5E }
// unsigned char crc8=0;
// unsigned int crc16=0;
//          crc8
//          -----
//          DEC  HEX
//          0    0
// crc8=CRC8(0x9B,0); // 211 D3
// crc8=CRC8(0xF1,crc8); // 228 E4
// crc8=CRC8(0x5E,crc8); // 100 64 ---> crc8
//
//          crc16
//          -----
//          DEC  HEX
//          0    0
// crc16=CRC16(0x9B,0); // 43841 AB41
// crc16=CRC16(0xF1,crc16); // 46250 B4AA
// crc16=CRC16(0x5E,crc16); // 34741 87B5 ---> crc16
//=====
// función para el cálculo del CRC_8
#define DefCRC8 (unsigned char)0x31
unsigned char CRC_8(char byte, unsigned char CRC)
{
    static int c;
    for(c=8;c>0;c--,byte<<=1)
    {
        CRC=((byte^CRC) & 0x80)? ((CRC <<1) ^ DefCRC8) : (CRC<<1);
    }
    return(CRC);
}
//=====
// función para el cálculo del CRC_16
unsigned int CRC_16(unsigned char byte, unsigned int CRC16)
{
    unsigned int data;
    int oddparity[16] = { 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0 };
    data=(int)byte;
    data = (data ^ (CRC16 & 0xff)) & 0xff;
    CRC16 >>= 8;
    if (oddparity[data & 0xf] ^ oddparity[data >> 4])
        CRC16 ^= 0xc001;
    data <<= 6;
    CRC16 ^= data;
    data <<= 1;
    CRC16 ^= data;
    return(CRC16);
}
//=====

```