Códigos de verificación por redundancia cíclica

Electrónica Digital

Marzo de 2020

1. Introducción

Este apunte pretende facilitar la comprensión de los códigos de verificación por redundancia cíclica (CRC, del inglés *cyclic redundancy check*). Aquí trataremos de cubrir la mayoría de los temas necesarios para que puedan ser capaces de generar y verificar mensajes con CRC.

CRC es un código de detección de errores que se usa comúnmente en redes digitales y dispositivos de almacenamiento para detectar cambios accidentales en datos sin procesar. A cada bloque de datos que ingresa a estos sistemas se le agrega un valor de verificación (el *checksum*) calculado por CRC. De esta manera, el receptor del mensaje puede detectar los errores existentes en los datos transmitidos. CRC se basa en la división, el contenido del mensaje es interpretado como una cadena de bits (dividendo) que es dividido por otro número binario fijo (divisor). El resto de esta división es el valor del *checksum*. Sin embargo, en realidad la teoría detrás de CRC es un poco más complicada. Los números binarios (dividendo y divisor) no son números enteros representados en binario, sino que son tratados como polinomios binarios donde los bits son los coeficientes. De esta forma, el valor de *checksum* es en realidad el resto de una división polinomial entre el contenido del mensaje y un *polinomio generador*. En la recuperación, el cálculo se repite y, en el caso de que los valores de verificación no coincidan, se pueden tomar medidas correctivas contra la corrupción de datos.

2. Aritmética modular

En matemática, se llama *aritmética modular* a un sistema aritmético utilizado con números enteros, en el cual los números vuelven a comenzar una vez que alcanzan un valor dado, denominado *módulo*.

Un uso familiar de la aritmética modular se da en el reloj de 12 horas, en el que el día se divide en dos períodos de 12 horas. Si la hora actual es 7:00, 8 horas más tarde será 3:00. La adición habitual sugeriría que la hora posterior debería ser $7 + 8 = 15$, pero esta no es la respuesta debido a que la hora del reloj "vuelve a comenzar" cada 12 horas. Debido a que el número de la hora comienza de nuevo después de llegar a 12, este es un módulo aritmético 12.

El resultado de la operación *módulo* (no confundir con el módulo de un vector a pesar que en castellano se llamen igual) se define como el resto de la división de un número por otro. Generalmente se la expresa con el símbolo % o el operador `mod`. Al realizar una división tenemos la siguiente ecuación:

$$s = n \times d + r$$

Donde s es el dividendo, d es el divisor, n es el cociente y r es el resto. De esta ecuación, tenemos dos operaciones:

$$s \div d = n$$

$$s \,\%d = r$$

Note que r **siempre** tiene que ser menor que d. A modo de ejemplo y continuando con el reloj de 12 horas: $15 = 1 \times 12 + 3$, por lo que $15 \div 12 = 1$ y $15 \,\%12 = 3$.

2.1. Aritmética módulo-2

La aritmética módulo-2 es un sistema aritmético donde cada resultado se toma módulo-2. Por ejemplo:

$$(10 + 5)\,_{\%2} = 1$$
$$(2 \times 8)\,_{\%2} = 0$$

Resumiendo, el resultado de la operación es 1 si el operando es impar y 0 si el operando es par. Dado que en nuestra codificación de datos estamos tratando cadenas de bits, **todos** nuestros operandos serán un 1 o un 0. Mientras que nuestros operandos sean 1 o 0 y nuestros resultados sean módulo-2, **todos** los números que escribiremos serán 1 o 0.

Una aplicación muy común de la aritmética módulo-2 es en circuitos digitales, donde las operaciones lógicas se realizan en módulo-2. Este será el caso a aplicar con los códigos CRC de mensajes binarios, como veremos a continuación.

2.1.1. Operaciones módulo-2 sobre bits

Nuestro sistema aritmético módulo-2 necesita que definamos nuevos operadores a fin de poder trabajar con bits. Notaremos que estos operadores son muy similares a los operadores utilizados en la lógica booleana, por lo que los discutiremos a continuación.

Suma: Para sumar dos números, se toma el módulo-2 del resultado. La tabla de verdad para la operación suma es:

+	0	1
0	0	1
1	1	0

Recordando la operaciones de la lógica booleana, al observar esta tabla podemos ver que la suma es idéntica a la operación `xor`. Por este motivo, utilizaremos los términos "suma" y "xor" de forma indistinta.

Multiplicación: Para multiplicar dos números en módulo-2 no se necesita definir un nuevo operador. La tabla de verdad para la operación multiplicación es:

×	0	1
0	0	0
1	0	1

Donde se puede ver que exactamente igual a la tabla de verdad de la operación booleana `and`. Por este motivo, utilizaremos los términos "multiplicación" y "and" de forma indistinta.

Dado que la resta y la división son raramente utilizadas en codificación de datos, no las discutiremos.

3. División de polinomios

Encontremos el cociente y el resto de la división de $x^5+x^4+x^2+1$, el dividendo y x^2+1, el divisor. El primer paso es reescribir el dividendo como un polinomio completo: $x^5 + x^4 + 0x^3 + x^2 + 0x + 1$.

El cociente y el resto se calculan de la siguiente manera:

1. Dividimos el primer término del dividendo por el término termino de mayor exponente de x. Poner el resultado sobre la barra ($x^5 \div x^2 = x^3$).

$$\begin{array}{r} x^3 \\ x^2+1\overline{)\,x^5+x^4+0x^3+x^2+0x+1} \end{array}$$

2. Multiplicamos el divisor por el resultado obtenido (el primer término del eventual cociente) y escribimos el resultado debajo de los términos del dividendo ($x^3(x^2+1) = x^5 + x^3$)

$$\begin{array}{r} x^3 \\ x^2+1 \overline{)x^5+x^4+0x^3+x^2+0x+1} \\ x^5+0x^4+x^3 \end{array}$$

3. Restamos el producto obtenido de los terminos correspondientes y escribimos el resultado debajo ($(x^5+x^4+0x^3)-(x^5+0x^4+x^3) = x^4-x^3$). Luego "bajamos" el siguiente término del dividendo.

$$\begin{array}{r} x^3 \\ x^2+1 \overline{)x^5+x^4+0x^3+x^2+0x+1} \\ \underline{x^5+0x^4+x^3} \\ +x^4-x^3+x^2 \end{array}$$

4. Repetimos los tres pasos previos, excepto que en esta ocasión utilizamos los términos que han sido escritos como el dividendo.

$$\begin{array}{r} x^3+x^2-x \\ x^2+1 \overline{)x^5+x^4+0x^3+x^2+0x+1} \\ \underline{x^5+0x^4+x^3} \\ +x^4-x^3+x^2 \\ \underline{+x^4+0x^3+x^2} \\ -x^3+0x^2+0x \\ \underline{-x^3+0x^2-x} \\ +x+1 \end{array}$$

El polinomio sobre la barra es el cociente $q(x)$ y lo que nos quedó en el item (4) es el resto $r(x)$.

$$x^5+x^4+x^2+1 = (x^2+1)\underbrace{(x^3+x^2-x)}_{q(x)} + \underbrace{x+1}_{r(x)}$$

3.1. División de polinomios módulo 2

Encontremos el cociente y el resto de la división módulo-2 de $x^5+x^4+x^2+1$, el dividendo y x^2+1, el divisor. De la misma forma que en el ejemplo anterior, el primer paso es reescribir el dividendo como un polinomio completo: $x^5+x^4+0x^3+x^2+0x+1$. Ahora, debemos verificar que todos los coeficientes sean 0 o 1. Luego de realizar cada resta (Item 3 de la Sección 3), se debe aplicar la operación módulo-2 a cada resultado parcial antes de a volver a dividir.

$$\begin{array}{rr} & x^3+x^2+x \\ x^2+1\ ) & \overline{x^5+x^4+0x^3+x^2+0x+1} \\ & \underline{x^5+0x^4+x^3} \\ & +x^4-x^3+x^2 \\ \texttt{mod-2} & +x^4+x^3+x^2 \\ & \underline{+x^4+0x^3+x^2} \\ & +x^3+0x^2+0x \\ & \underline{+x^3+0x^2+x} \\ & -x+1 \\ \texttt{mod-2} & +x+1 \end{array}$$

De esta forma, operando en módulo-2 tenemos que:

$$x^5+x^4+x^2+1 = (x^2+1)\underbrace{(x^3+x^2+x)}_{q(x)} + \underbrace{x+1}_{r(x)}$$

4. Operatoria para calcular un mensaje con CRC

Primero definamos los elementos que utilizaremos

M : Mensaje de m bits.

C : divisor de c bits

M puede variar, pero se debe cumplir que $m > c - 1$. Nuestro campo CRC consiste de una cadena R de $c - 1$ bits.

4.1. ¿Cómo genero el mensaje?

1. Agrego $c-1$ "0" al final de M. Estos determinan el lugar en el que se guardará el CRC. Llamamos M' a este mensaje extendido: M' = M + R.
2. Dividir M' por C utilizando **aritmética de módulo-2**. Prestar atención que módulo-2 no equivale a base-2. Recordemos que estas cantidades no son números en el sentido usual (cada vez que se divide C (de c bits) por otra cantidad Z de c-bits, decimos que C "entra" en Z una vez si el primer bit es 1). Llamaremos R al resto de C-1 bits.
3. Reemplazar los C-1 '0' que agregamos por el resto. Este es nuestro mensaje con CRC.
4. Enviar nuestro mensaje con CRC al receptor.

La Figura 1 nos muestra el diagrama de flujo correspondiente al algoritmo que utilizaremos para generar y verificar mensajes con CRC.

Para comprobar, se debe dividir el mensaje con CRC por el polinomio divisor y su resto debe ser 0.

5. Ejemplos

Como ejemplo generaremos un mensaje con CRC y luego veremos la forma de comprobarlo. Como se puede observar, el mismo algoritmo sirve tanto para generar como para comprobar un mensaje con CRC.

Nuestro mensaje será M = 110101, el polinomio generador será C = 101 (sus coeficientes equivalen a $x^2 + 1$), por lo tanto, si seguimos las instrucciones al inicio del capítulo tenemos que M'= 11010100.

5.1. Ejemplo 1: Generación del mensaje

```
1 1 0 1 0 1 0 0  Mensaje con 2 ceros agregados
1 0 1            Alineo el polinomio generador con el primer 1
---------
  1 1 1          XOR con los datos
  1 0 1          Desplazamiento a la derecha
  -----
    1 0 0        XOR con los datos
    1 0 1        Desplazamiento a la derecha
    -----
      1 1 0      XOR con los datos
      1 0 1      Alineo el polinomio generador con el primer 1
      -----
        1 1 0    XOR con los datos
        1 0 1    Desplazamiento a la derecha
        -----
          1 1    <- Resultado luego del último XOR
```

El mensaje con CRC se forma cambiando los ceros que agregamos por el resultado luego del último XOR. En consecuencia, el mensaje con CRC es 11010111.

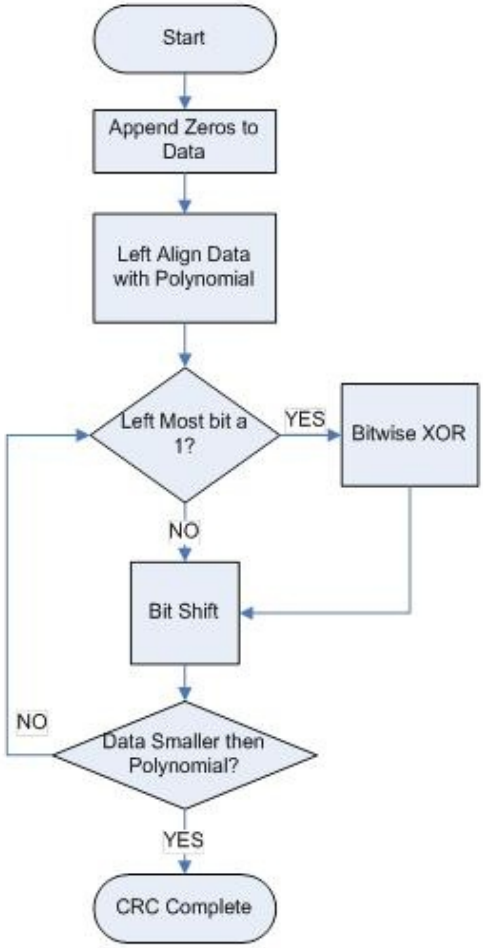

Figura 1: Algoritmo para realizar CRC. Fuente: `https://www.digikey.com/eewiki/display/microcontroller/CRC+Basics`.

5.2. Ejemplo 2: Comprobación del mensaje

En este ejemplo, recibimos el mensaje M=11010111. Tanto el emisor como el receptor saben que el polinomio generador es 101. Para comprobar un mensaje con CRC se ejecuta el mismo algoritmo. Si el resto de la operación equivale a 0, el mensaje no tiene errores.

```
1 1 0 1 0 1 1 1  Datos con CRC recibidos
1 0 1            Alineo el polinomio generador con el primer 1
---------
  1 1 1          XOR con los datos
  1 0 1          Desplazamiento a la derecha
  -----
    1 0 0        XOR con los datos
    1 0 1        Desplazamiento a la derecha
    -----
      1 1 1      XOR con los datos
      1 0 1      Alineo el polinomio generador con el primer 1
      -----
        1 0 1    XOR con los datos
        1 0 1    Desplazamiento a la derecha
        -----
          0 0    <- Resultado luego del último XOR
```

6. Referencias

1. Data Coding Theory/Modulo-2 Arithmetic: https://en.wikibooks.org/wiki/Data_Coding_Theory/Modulo-2_Arithmetic
2. Cyclic redundancy check: https://en.wikipedia.org/wiki/Cyclic_redundancy_check
3. Verificación de redundancia cíclica: https://es.wikipedia.org/wiki/Verificaci%C3%B3n_de_redundancia_c%C3%ADclica
4. Polynomial long division: https://en.wikipedia.org/wiki/Polynomial_long_division
5. CRC Generating and Checking http://ww1.microchip.com/downloads/en/appnotes/00730a.pdf
6. Understanding and implementing CRC (Cyclic Redundancy Check) calculation http://www.sunshine2k.de/articles/coding/crc/understanding_crc.html
7. CRC Basics https://www.digikey.com/eewiki/display/microcontroller/CRC+Basics
8. Tutorial: Cyclic Redundancy Check (CRC) computation https://netfuture.ch/tutorials/crc/