Algoritmo de Sarwate

Gonzalo Nahuel Vaca

7 de agosto de 2021

1. Introducción

El algoritmo de Sarwate permite realizar el cálculo de redundancia cíclica (CRC) operando byte a byte. Esto es una mejora frente al método tradicional de realizar una división contra un polinomio que obliga a analizar un bit por ciclo de máquina.

Para este trabajo se propone realizar un CRC de 8 bits.

1.1. Plataforma de desarrollo

Se utilizará la placa *NUCLEO-F429ZI* y el RTOS *Mbed*.

2. Máquina de estados finitos

Se plantea una solución como una máquina de estados finitos de tipo *Mealy*. En el cuadro 1 se puede observar la representación de las transiciones entre los 256 estados.

$S_{actual} \backslash S_{siguiente}$	S_1	S_2	$\cdots$	S_{256}
S_1	$I_{1,1}/O_1$	$I_{1,2}/O_2$	$\cdots$	$I_{1,256}/O_{256}$
S_2	$I_{2,1}/O_1$	$I_{2,2}/O_2$	$\cdots$	$I_{2,256}/O_{256}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
S_{256}	$I_{256,1}/O_1$	$I_{256,2}/O_2$	$\cdots$	$I_{256,256}/O_{256}$

Cuadro 1: transición de estados

Los estados se expresan como S_n, las entradas como $I_{i,j}$ y las salidas O_n. El estado inicial es S_1 con una salida O_1 que en todo caso será igual a cero.

3. Funcionamiento

El algoritmo funciona de la siguiente manera:

1. Se crea una tabla con los 256 resultados posibles de la división polinómica.
2. Se comienza cargando como salida el primer elemento de la tabla.
3. Se realiza una operación *XOR* con la salida actual y el primer byte de la secuencia.
4. El resultado de la operación el es índice (estado) de la tabla que tiene el valor de la nueva salida.
5. Se opera hasta finalizar los datos a verificar.
6. La última salida obtenida es el CRC del paquete.
7. El CRC calculado debe coincidir con el CRC transmitido.

4. Codificación

4.1. Obtención de tabla

Se decidió de manera arbitraria usar el siguiente polinomio:

$$0x^7 + 0x^6 + 0x^5 + x^4 + x^3 + x^2 + 0x^1 + 0x$$

Simplificado:

$$x^4 + x^3 + x^2$$

Si solo tomamos los coeficientes, obtenemos el número hexadecimal $1C$.

Como se mencionó en la sección 1, la tabla guarda los resultados de la división del polinomio con los 256 valores posibles de un byte. Se muestra el siguiente código para lograrlo:

```
POLYNOMIAL = 0x1C

def buildTable(poly=POLYNOMIAL) -> list:
    length = 256
    polynomial = poly
    mask = 0xFF
    table = [0] * length
    for i in range(length):
        crc = i
        for _ in range(8):
            if(crc & 0x80):
            crc = (crc << 1) ^ polynomial
            else:
            crc = crc << 1
        table[i] = (crc & mask)
    return table
```

Se arroja la siguiente tabla de valores:

```
TABLE = [
    0, 28, 56, 36, 112, 108, 72, 84, 224, 252, 216, 196, 144,
       140, 168, 180, 220,
    192, 228, 248, 172, 176, 148, 136, 60, 32, 4, 24, 76, 80,
       116, 104, 164,
    184, 156, 128, 212, 200, 236, 240, 68, 88, 124, 96, 52, 40,
       12, 16, 120,
    100, 64, 92, 8, 20, 48, 44, 152, 132, 160, 188, 232, 244,
       208, 204, 84,
    72, 108, 112, 36, 56, 28, 0, 180, 168, 140, 144, 196, 216,
       252, 224, 136,
    148, 176, 172, 248, 228, 192, 220, 104, 116, 80, 76, 24, 4,
       32, 60, 240, 236,
    200, 212, 128, 156, 184, 164, 16, 12, 40, 52, 96, 124, 88,
       68, 44, 48, 20, 8,
    92, 64, 100, 120, 204, 208, 244, 232, 188, 160, 132, 152,
       168, 180, 144, 140,
    216, 196, 224, 252, 72, 84, 112, 108, 56, 36, 0, 28, 116,
       104, 76, 80, 4, 24,
    60, 32, 148, 136, 172, 176, 228, 248, 220, 192, 12, 16, 52,
       40, 124, 96, 68, 88,
    236, 240, 212, 200, 156, 128, 164, 184, 208, 204, 232, 244,
       160, 188, 152, 132,
```

```
13      48, 44, 8, 20, 64, 92, 120, 100, 252, 224, 196, 216, 140,
            144, 180, 168, 28, 0,
14      36, 56, 108, 112, 84, 72, 32, 60, 24, 4, 80, 76, 104, 116,
            192, 220, 248, 228,
15      176, 172, 136, 148, 88, 68, 96, 124, 40, 52, 16, 12, 184,
            164, 128, 156, 200, 212,
16      240, 236, 132, 152, 188, 160, 244, 232, 204, 208, 100, 120,
            92, 64, 20, 8, 44, 48
17  ]
```

4.2. Obtención del CRC

Para obtener el CRC se ejecuta la máquina de estados finitos explicada en la sección 2. Se propone el siguiente código:

```
1  def buildCRC8(data, table=TABLE) -> int:
2      crc8: int = table[0]
3      indx: int = 0
4      for byte in data:
5          indx = crc8 ^ byte
6          crc8 = table[indx]
7      return crc8
```

En el código, el estado S_n está representada por la variable *indx*, la salida por *crc8* y la entrada por *byte*.

Finalmente se puede verificar el CRC invocando a la máquina de estados en el *payload* y comparando el CRC transmitido. Como se muestra en el siguiente código:

```
1  def computeCRC8(frame, table=TABLE) -> bool:
2      if len(frame) < 2:
3          return False
4      data: list = frame[0:-1]
5      crc8: str = frame[-1]
6      comp: str = buildCRC8(data, table)
7      return crc8 == comp
```

4.3. Codificación para microcontroladores

Creación de la tabla:

```
1  #define TABLE_LENGTH 256
2  #define OCTET_MASK 0X000000FF
```

```
3
4  unsigned char CRC8TABLE[TABLE_LENGTH];
5
6  void GenerateTable(void)
7  {
8      unsigned char crcValue;
9      unsigned char polynomial = 0x1C;
10     unsigned int i, j;
11
12     for(i=0; i < TABLE_LENGTH; i++)
13     {
14         crcValue = i;
15         for(j=0; j < 8; j++)
16         {
17             if(crcValue & 0x80) {
18                 crcValue = (crcValue << 1);
19                 crcValue = crcValue ^ polynomial;
20             }
21             else {
22                 crcValue = crcValue << 1;
23             }
24         }
25         CRC8TABLE[i] = crcValue & OCTET_MASK;
26     }
27 }
```

Cálculo del CRC:

```
1  unsigned char buildCRC8(unsigned int* bufferPtr, unsigned int
       bufferLenght)
2  {
3      unsigned int i;
4      unsigned char crcValue = 0x00;
5      unsigned int tableIndex;
6
7      for(i=0; i < bufferLenght; i++)
8      {
9          tableIndex = crcValue^(bufferPtr[i] & 0xFF);
10         crcValue = CRC8TABLE[tableIndex];
11     }
12     return crcValue;
13 }
```