# 循环冗余校验的基本原理

CRC码的基本原理为多项式除法。

设信息多项式为C(x)，生成多项式为G(x)，满足：

，

CRC编码器需要产生校验位，使得：



能够被G(x)除尽。

上述CRC称为K阶CRC。在LTE协议中为8阶CRC。

产生CRC的方法举例如下：

设信息位为1010，生成多项式为1011，我们将1011后面补3个0，即执行下述除法：



产生余数011即为最后的结果。将余数粘贴于信息位后，即1010011，能够被1011除尽。

相关算法也能采用硬件逻辑实现，以LTE 为例，可以画为：



图 1 CRC的硬件实现

上述实现用C语言描述为：

|  |
| --- |
| void CrcEncode1( int \*parityOut, int \*dataIn, int crcSize, int poly, int InLen)  {  int i;  int crcState=0;  int feedBack;  // data input  for(i=0;i<InLen;i++)  {  // check feedback  feedBack = (crcState&0x1)\*poly;  // data input  crcState = crcState>>1;  crcState = crcState + (dataIn[i]<<(crcSize-1));  // xor state with feedback, note the 'crcSize' bit will always be 0  crcState = crcState^feedBack;  }  // zero input  for(i=0;i<crcSize;i++)  {  // check feedback  feedBack = (crcState&0x1)\*poly;  // data input with zero  crcState = crcState>>1;  // xor state with feedback, note the 'crcSize' bit will always be 0  crcState = crcState^feedBack;  }  // output crcstate  for(i=0;i<crcSize;i++)  {  parityOut[i] = (crcState)&0x1;  crcState = crcState>>1;  }  } |

# 循环冗余校验的另外一种写法

我们可以看到前述标准的CRC除法电路中，前crcSize个Cycle都是无用的。实际上，我们可以直接利用第一个值开始计算CRC。将图 1进行改进，得到下面一个逻辑：



图 2 CRC的硬件实现2

这套逻辑的基本原理还是除法，但思路改变了一下。这里设信息位为1010，生成多项式为1101来尝试解释一下改进的思路：



图 3 CRC硬件实现的改进思路

这套逻辑减少了计算CRC的Cycle个数，在输出最后数据时不再需要进行补零操作。

|  |
| --- |
| void CrcEncode2(int \*parityBits, int \*dataIn, int crcSize, unsigned int poly, int inLen)  {  int i;  unsigned int crcState=0; // shift regesiter  int shiftIn;    // encode the input bits  for (i=0; i<inLen; i++)  {  // cyclic encode  shiftIn = dataIn[i]^(crcState&0x1);  crcState = crcState>>1;  crcState = crcState^(shiftIn\*poly);  }    // output the crcState  for (i=0; i<crcSize; i++)  {  parityBits[i] = (crcState)&0x1;  crcState = crcState>>1;  }  } |

# CRC的硬件串行快速算法

CRC的串行快速算法从图 2开始推导。可以看出，CRC的状态和输入对于1个Cycle的变化满足以下关系式：



那么图 2可以改进为：



图 4 CRC硬件快速算法

这套硬件在计算CRC时，一次可以处理2个Bit，从而速度提升了一倍。相关验算代码如下，可以证明上述改动的正确性。

|  |
| --- |
| void CrcEncodeVerify(int \*parityBits, int \*dataIn, int crcSize, unsigned int poly, int inLen)  {  int i,k;  unsigned int crcState[8] = {0}; // shift regesiter  unsigned int crcStateTmp[8];  unsigned int crcIn0, crcIn1;    // encode the input bits  for (i=0; i<inLen/2; i++)  {  crcIn0 = dataIn[2\*i]; crcIn1 = dataIn[2\*i+1];  crcStateTmp[0] = crcIn0^crcIn1^crcState[7]^crcState[6];  crcStateTmp[1] = crcIn1^crcState[6];  crcStateTmp[2] = crcIn0^crcState[7]^crcState[0];  crcStateTmp[3] = crcIn0^crcState[7]^crcIn1^crcState[6]^crcState[1];  crcStateTmp[4] = crcIn1^crcState[6]^crcState[2];  crcStateTmp[5] = crcIn0^crcState[7]^crcState[3];  crcStateTmp[6] = crcState[4];  crcStateTmp[7] = crcIn0^crcState[7]^crcIn1^crcState[6]^crcState[5];  for(k=0;k<8;k++)  {  crcState[k] = crcStateTmp[k];  }  }    // output the crcState  for (i=0; i<crcSize; i++)  {  parityBits[i] = crcState[7-i];  }  } |

上述算法对于不同的生成多项式，都需要单独推导对应的快速运算逻辑。对于不能被2整除的数据长度，可以在其头部填零，不会改变CRC的最终输出结果。

# CRC的软件查表算法

CRC软件的快速算法主要为查表法。查表法的基本原理为：预先计算出N个Bit对生成多项式的余数，本次输入的N个Bit可以立即通过查表法得到余数，余数和下次输入的N个Bit异或后，组成新的N个输入Bit；如此反复。

下面给出了以byte为单位进行查表的CRC计算代码。

|  |
| --- |
| void CrcEncode3(int \*parityBits, int \*dataIn, int crcSize, unsigned int poly, int inLen)  {  int i,k;  unsigned int crcState=0; // shift regesiter  int shiftIn;  static int crcTable[256];  static int polyStatic=0;  // need to regenerate crc table  if(polyStatic!=poly)  {  for(i=0;i<256;i++)  {  // reset crcState  crcState = 0;  for(k=0;k<8;k++)  {  // cyclic encode  shiftIn = ((i>>k)&0x1)^(crcState&0x1);  crcState = crcState>>1;  crcState = crcState^(shiftIn\*poly);  }  // store crcState to crcTable  crcTable[i] = crcState;  }  }    // encode the input bits  crcState = 0;  for (i=0; i<inLen/8; i++)  {  // combine data input to byte  shiftIn = 0;  for(k=0;k<8;k++)  shiftIn ^= (dataIn[8\*i+k]<<k);  shiftIn = shiftIn^(crcState&0xff);  crcState = crcTable[shiftIn]^(crcState>>8);  }    // output the crcState  for (i=0; i<crcSize; i++)  {  parityBits[i] = (crcState)&0x1;  crcState = crcState>>1;  }  } |

# CRC的分段算法

假设待译码数据是乱序分段进入CRC校验模块，此时我们可能使用到分段CRC算法。

设信息位为1010，生成多项式为1011，我们可以将信息位一位一位进行处理，比如第1位1，可以先变为1000000除以1011；第3位1，可以变为10000除以1011。如下式：

和

将余数求异或，可得011，为最后的结果。故，每一位输出都可以在补零后，单独除以生成多项式，然后再将得到的余数求异或，依然能够得到正确的结果。

另外，我们可以利用：



这样的公式，预先计算出补零部分的CRC结果，从而达到并行提速的效果。

下面给出了分段CRC的算法：

|  |
| --- |
| #define parallelNum 8  void CrcEncode4(int \*parityBits, int \*dataIn, int crcSize, unsigned int poly, int inLen)  {  int i,k;  unsigned int crcState=0; // shift regesiter  int shiftIn, feedBack;  int parallelTable[parallelNum-1];  int midCrc;  int crcStateRevHigh, crcStateRevLow, paraTableRev;  //prepare parallel table  //init crcState  crcState = 0;  i = 0;  for(k=0;k<inLen;k++)  {  if(k==0)  shiftIn = 1;  else  shiftIn = 0;  // check feedback  feedBack = (crcState&0x1)\*poly;  // data input  crcState = crcState>>1;  crcState = crcState + (shiftIn<<(crcSize-1));  // xor state with feedback, note the 'crcSize' bit will always be 0  crcState = crcState^feedBack;  if( (k%(inLen/parallelNum)==0) && (k!=0) )  {  parallelTable[i] = crcState;  i++;  }  }    midCrc = 0;  // encode the input bits  for(k=0; k<parallelNum; k++)  {  crcState = 0;  for(i=0; i<(inLen/parallelNum); i++)  {  // cyclic encode  shiftIn = dataIn[k\*(inLen/parallelNum)+i]^(crcState&0x1);  crcState = crcState>>1;  crcState = crcState^(shiftIn\*poly);  }  if(k!=parallelNum-1)  {  // rev  paraTableRev = 0;  for(i=0;i<crcSize;i++)  {  paraTableRev = paraTableRev<<1;  paraTableRev += (parallelTable[parallelNum-2-k]>>i)&0x1;  }  // mult  crcStateRevHigh = 0;  crcStateRevLow = 0;  for(i=0;i<crcSize;i++)  {  if( ((crcState>>(crcSize-1-i))&0x1) == 1 )  {  crcStateRevLow ^= (paraTableRev<<i)&((1<<crcSize)-1);  crcStateRevHigh ^= (paraTableRev>>(crcSize-i));  }  }  // mod  crcState = 0;  for(i=0; i<crcSize; i++)  {  // cyclic encode  shiftIn = ((crcStateRevHigh>>(crcSize-i-1))&0x1)^(crcState&0x1);  crcState = crcState>>1;  crcState = crcState^(shiftIn\*poly);  }  for(i=0; i<crcSize; i++)  {  crcState ^= ((crcStateRevLow>>(crcSize-i-1))&0x1)<<i;  }  }  midCrc ^= crcState;  }  crcState = midCrc;  // output the crcState  for (i=0; i<crcSize; i++)  {  parityBits[i] = (crcState)&0x1;  crcState = crcState>>1;  }  } |

# CRC的matlab调用

以LTE协议中为例，最高位D8是不需要输入到matlab中的，按照从高阶到低阶的方式建立crc生成多项式，本例为[1 0 0 1 1 0 1 1]或者0x9B作为生成多项式，即可产生编码后结果。

示例代码如下：

|  |
| --- |
| dataInLen = 8192;  dataIn = randint(1,dataInLen);  h = crc.generator('Polynomial', '0x9B');  encoded = generate(h, dataIn')'; |

其产生数据的效果和前述各种算法产生的效果相同。