RS 纠错编码原理

—及其实现方法

陈文礼

January 08 于郑州

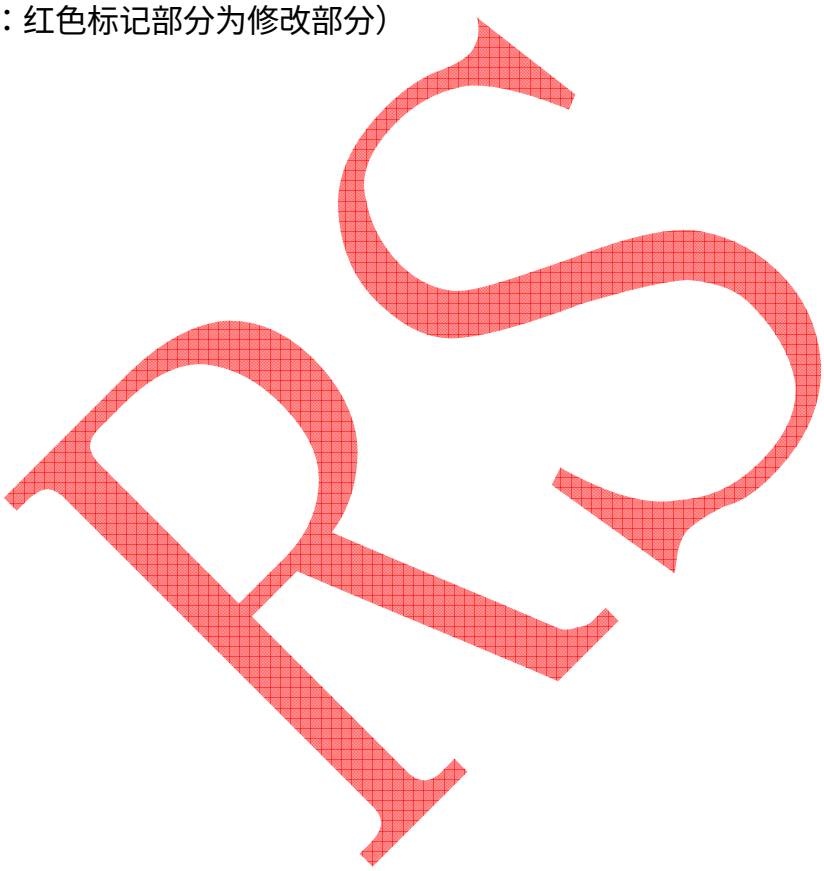
If you have any suggestion or criticism . please email to ciciendi@163.com

QQ：83902112

修订说明

自从发表《RS 纠错编码原理及其实现方法》以来，收到很多初识 RS 编码的读者的来信，大家纷纷表示这篇文章对初学者很有帮助，但同时也指出了很多不足。比如第一版的例子中都是按照码长 $n = 2^m - 1$，但在实际应用中并不总是这种情况，还有就是 MATLAB 程序，由于作者在工程中是在 DSP 上用 C 实现的，所以文章中的 MATLAB 程序只是用来说明问题，并没有经过调试。做事应该有始有终，这次修改，附有详细的经过调试的 MATLAB 程序。并尽量做到程序具有通用性。

（注：红色标记部分为修改部分）



陈文礼

2008-11　于郑州

前言

随着越来越多的系统采用数字技术来实现，纠错编码技术也得到了越来越广泛的应用。RS 码既可以纠正随机错误，又可以纠正突发错误，具有很强的纠错能力，在通信系统中应用广泛。近些年来，随着软件无线电技术的发展，RS 编码、译码一般都在通用的硬件平台上实现。通常采用基于 FPGA 的 VHDL 编码硬件实现，或者在 DSP、单片机上用 C 和汇编编程软件实现。

RS 纠错编码涉及的领域很广，特别是设计到很多数学知识。这对那些对数学不太感冒的工程技术人员来书是个不小的挑战。尽管讲 RS 编码的书籍很多，但是那些书都是采用循序渐进，逐步引入的方式，从汉明码到循环码，从循环码到 BCH 码，BCH 码再引入 RS 码。对于工程技术人员他们需要的是简明扼要的讲解，和详细的实现方法。

本人写这篇文章的宗旨就是尽量最简单的语言，最简短的篇幅，来讲 RS 纠错编码原理，把重点来放在实现方法上。

为了便于读者仿真，本文采样 MATLAB 程序实现，程序尽量符合硬件 C 语言写法，读者经过简单修改即可应用到工程中去。

本文读者对象

本文是为那些初识 RS 编码的学生、工程技术人员而写，并不适合做理论研究，如果你是纠错编码方面的学者、专家，那么本文并不适合你。

由于作者水平有限，错误在所难免，恳请读者批评指正。

陈文礼

2008-01　于郑州

一、必备的一些代数知识

1、在纠错编码代数中，把以二进制数字表示的一个数据系列看成一个多项式。例如二进制数字序列 10101111，可以表示成：

$$M(x) = a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x^1 + a_0x^0$$

$$= x^7 + x^5 + x^3 + x^2 + x^1 + 1$$

式中的 x^i 表示代码的位置，或某个二进制数位的位置， x^i 前面的系数 a_i 表示码的值。若 a_i 是一位二进制代码，则取值是 0 或 1。$M(x)$ 称为信息代码多项式。

多项式次数：

称系数不为 0 的 x 的最高次数为多项式 $f(x)$ 的次数，记为 $\partial \circ f(x)$。

2、域

域在 RS 编码理论中起着至关重要的作用。

简单点说域 $GF(2^m)$ 有 2^m （设 $2^m=q$ ）个符号 $\left[0,\alpha^0,\alpha^1\cdots\alpha^{q-2}\right]$

且具有以下性质：

域中的每个元素都可以用 $\alpha^0,\alpha^1,a^2\cdots\alpha^{m-1}$ 的和来表示。 $\alpha^{q-1}=1$

α 为本原多项式 $p(x)$ 的根。

运算规则有：

在纠错编码运算过程中，加、减、乘和除的运算是在伽罗华域中进行。现以 GF(2^4)域中运算为例：

加法例：$\alpha+\alpha^{10}=0010+0111=0101=\alpha^8$ （模 2 加法相当于 0010 与 0111 异或）

减法运算与加法相同

乘法例：$\alpha^8\bullet\alpha^{10}=\alpha^{(8+10)\bmod 15}=\alpha^3$

除法例：$\alpha^8/\alpha^{10}=\alpha^{-2}=\alpha^{-2+15}=\alpha^{13}$

不理解没关系，下面的例子也许对你有帮助。

例： m=4， $p(x)=x^4+x+1$ 求 $GF(2^m)$ 的所有元素

： 因为 α 为 $p(x)$ 的根 得到 $\alpha^4+\alpha+1=0$ 或 $\alpha^4=\alpha+1$ （根据运算规则）

由此可以得到域的所有元素

元素		二进制对应码	十进制对应值
0	0	0000	0
α^0	1	0001	1
α^1	α^1	0010	2
α^2	α^2	0100	4
α^3	α^3	1000	8
α^4	$\alpha+1$	0011	3
α^5	$\alpha(\alpha+1)=\alpha^2+\alpha \pmod{p(\alpha)}$	0110	6
α^6	$\alpha(\alpha^2+\alpha)=\alpha^3+\alpha^2 \pmod{p(\alpha)}$	1100	12
α^7	$\alpha(\alpha^3+\alpha^2)=\alpha^3+\alpha+1 \pmod{p(\alpha)}$	1011	11
α^8	$\alpha(\alpha^3+\alpha+1)=\alpha^2+1 \pmod{p(\alpha)}$	0101	5
α^9	$\alpha(\alpha^2+1)=\alpha^3+\alpha \pmod{p(\alpha)}$	1010	10
α^{10}	$\alpha(\alpha^3+\alpha)=\alpha^2+\alpha+1 \pmod{p(\alpha)}$	0111	7
α^{11}	$\alpha(\alpha^2+\alpha+1)=\alpha^3+\alpha^2+\alpha \pmod{p(\alpha)}$	1110	14
α^{12}	$\alpha(\alpha^3+\alpha^2+\alpha)=\alpha^3+\alpha^2+\alpha+1 \pmod{p(\alpha)}$	1111	15
α^{13}	$\alpha(\alpha^3+\alpha^2+\alpha+1)=\alpha^3+\alpha^2+1 \pmod{p(\alpha)}$	1101	13
α^{14}	$\alpha(\alpha^3+\alpha^2+1)=\alpha^3+1 \pmod{p(\alpha)}$	1001	9
α^{15}	$\alpha(\alpha^3+1)=1 \pmod{p(\alpha)}$	0001	1

由此可以看出本原多项式是求解域的全部元素的关键。读者也许会有这样的疑问我们如何得到 $p(x)$ 呢？　本原多项式 $p(x)$ 的特性是 $\frac{x^{2^m-1}+1}{p(x)}$ 得到的余式等于 0。

由于作者也是工程技术人员，具体怎么得到 $p(x)$，也没有深究过。

作者在设计 RS 编码时候都是根据 MATLAB 指令 rsgenpoly 来得到 $p(x)$。

其格式为 rsgenpoly(n,k)

参数 n 为码长一般 $n=2^m-1$，k 为信息码元个数。

例如 m=4，码长 n＝15，信息码元长度为 9

GF(2^4)的本原多项式可以根据指令

>>rsgenpoly(15,9)

得到：

ans = GF(2^4) array. Primitive polynomial = D^4+D+1 (19 decimal)

有读者来信问：我要做一个（158，128，31）的 RS 编码，在 MATLAB 中输入命令 rsgenpoly(158,128)，结果 MATLAB 报错：

Error using ==> rsgenpoly

N must equal 2^m-1 for some integer m between 3 and 16.

这里做一下解释，我们做 RS 编码时首先要根据码长选取 m，选择原则 是 $n<2^m$，

若码长为 158，那么我们可以选择 $m=8$，rsgenpoly 命令的第一个参数必须为 2^m-1，第二个参数可以随便选择只要小于 2^m-1 就形了。

在此给出 $m\in(2,16)$ 的所有本原多项式。

```
(m = 2)
P[m+1] = { 1, 1, 1 };

(m = 3)
/* 1 + x + x^3 */
P[m+1] = { 1, 1, 0, 1 };

(m = 4)
/* 1 + x + x^4 */
P[m+1] = { 1, 1, 0, 0, 1 };

(m = 5)
/* 1 + x^2 + x^5 */
P[m+1] = { 1, 0, 1, 0, 0, 1 };
```

```
(m = 6)
/* 1 + x + x^6 */
P[m+1] = { 1, 1, 0, 0, 0, 0, 1 };

(m = 7)
/* 1 + x^3 + x^7 */
P[m+1] = { 1, 0, 0, 1, 0, 0, 0, 1 };

(m = 8)
/* 1+x^2+x^3+x^4+x^8 */
P[m+1] = { 1, 0, 1, 1, 1, 0, 0, 0, 1 };

(m = 9)
/* 1+x^4+x^9 */
P[m+1] = { 1, 0, 0, 0, 1, 0, 0, 0, 0, 1 };

(m = 10)
/* 1+x^3+x^10 */
P[m+1] = { 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1 };

(m = 11)
/* 1+x^2+x^11 */
P[m+1] = { 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1 };

(m = 12)
/* 1+x+x^4+x^6+x^12 */
P[m+1] = { 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1 };

(m = 13)
/* 1+x+x^3+x^4+x^13 */
P[m+1] = { 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1 };

(m = 14)
/* 1+x+x^6+x^10+x^14 */
P[m+1] = { 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1 };

(m = 15)
/* 1+x+x^15 */
P[m+1] = { 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 };

(m = 16)
/* 1+x+x^3+x^12+x^16 */
P[m+1] = { 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1 };
```

二、线性分组码的一些基本概念

1、线性分组码一般用(n,k)或(n,k,d)表示 n为码长，k为信息码元的数目，$n-k$为监督码元的数目。d表示码元距离。

定义：两个码组上对应位置上数字不同的个数称为码组的距离。

发送的码字 $c=(c_1,c_2,c_3,...c_{n)}$

接收的矢量$r=(r_1,r_2,r_3,...r_{n)}$

信道错误图样：$e=c+r$

例如$c=$（1，1，0，0，0） $r=$（1，0，0，0，1）
$e=$（1+1，1+0，0+0，0+0，0+1）=（0，1，0，0，1）
从而可以看出从左端起第 2 位和第 5 位是错误的。

2、校验矩阵概念

码长为 n,信息数为 k，监督数为 r。

这样的一组码形式为：$c=m_1,m_2,...m_k,p_1,p_2,...p_r$

m_i表示第i个信息码，p_j表示第j个校验码

各个校验码可从下列线性方程组求得。

$$\begin{aligned}
&h_{11}m_1+h_{12}m_2+...+h_{1k}m_k+1p_1+0p_2+...+0p_r=0\\
&h_{21}m_1+h_{22}m_2+...+h_{2k}m_k+0p_1+1p_2+...+0p_r=0\\
&...\\
&h_{r1}m_1+h_{r2}m_2+...+h_{rk}m_k+0p_1+0p_2+...+1p_r=0
\end{aligned}\qquad (1\text{-}1)$$

式中h_{ij}是常数

校验方程组可写成校验矩阵

$$H=\begin{bmatrix}
h_{11} & h_{12} & \dots & h_{1k} & 1 & 0 & 0 & \dots & 0\\
h_{21} & h_{22} & \dots & h_{2k} & 0 & 1 & 0 & \dots & 0\\
..... & & \dots & & & & & & \\
h_{r1} & h_{r2} & \dots & h_{rk} & 0 & 0 & 0 & \dots & 1
\end{bmatrix}$$

该矩阵具有r行和n列
故式(1-1)可以写成

$Hc^T=0$或$cH^T=0$

H 矩阵称为[n,k,r]码的校验矩阵。

发送矢量为 C 接收矢量为 r 若 $rH^T \neq 0$

则说明接收到的码有错误。

设错误图样为 e

则可写成以下关系式 $r = c + e$

为了纠错必须知道那些位上存在错误。这可由校正子（又称伴随式） s 来确定

$$s = rH^T = cH^T + eH^T = eH^T$$

译码器的主要任务就是如何从 s 中得到最像 e 的错误图样 $\hat{e}$

从而译出 $\hat{c} = r - \hat{e}$

设第 i 个是错误的

因此 e =(0 0 … 0 1 0 … 0)

第 i 个有错误

$$s = rH^T = (0\ 0\ \ldots\ 0\ \ 1\ \ 0\ \ldots\ 0)\begin{bmatrix} h_{11} & h_{21} & \ldots & h_{r1} \\ h_{12} & h_{22} & \ldots & h_{r2} \\ \vdots & \vdots & \ldots & \vdots \\ h_{1k} & h_{2k} & \ldots & h_{rk} \\ 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ldots & \vdots \\ 0 & 0 & \ldots & 1 \end{bmatrix} = (h_{1i} \quad h_{2i} \quad \ldots \quad h_{ri})$$

计算出的矢量示出 i 是出错误的位置。

3、生成矩阵概念

生成矩阵 G ， 它是一个 k 行， n 列的矩阵

若已知信息组 m ，通过生存矩阵可求得相应的码字。

$c = m \times G$ （ m 是 k 个信息元组成的信息组）

这个应该比较容易理解，在此就不做过多解释。

三、RS 码的一些重要性质

1、RS 码生成多项式：

码长 $n = 2^m - 1$， 监督元数目 $r = n - k = 2t$ ，能纠正 t 个错误。

定义：在(n,k,d)的 RS 码中，存在唯一的$n-k$次多项式$g(x)$，使得每一个码多项式$c(x)$都是$g(x)$的倍式。$g(x)$称为[n,k,d]RS 码的生成多项式。

一般情况下 $g(x)=(x-\alpha)(x-\alpha^2)\cdots(x-\alpha^{2t})$

2、定理：

在$GF(2^m)$中，每个非 0 元素$(1,\alpha,\alpha^2\cdots\alpha^{2^m-2})$均满足$x^{2^m-1}=1$，反之$x^{2^m-1}-1=0$的根必在$GF(2^m)$中。

所以

$x^n-1=(x-a^0)(x-a^1)(x-a^2)\cdots(x-a^{n-1})$

3、RS 码的校验多项式

由于生成多项式$g(x)$是x^n-1的因式

$x^n-1=g(x)h(x)$

$g(x)$为$n-k$次多项式，则$h(x)$为k次多项式，

$x^n-1=g(x)h(x)=(g_{n-k}x^{n-k}+\cdots+g_1x+g_0)(h_kx^k+\cdots+h_1x+h_0)$

由右式可以看出$x^{n-1},x^{n-2},\cdots x$的系数均等于 0

即

$$g_0h_0=-1$$
$$g_0h_1+g_1h_0=0$$
$$\vdots$$
$$g_0h_i+g_1h_{i-1}+\cdots+g_{n-k}h_{i-(n-k)}=0$$
$$\vdots$$
$$g_0h_{n-1}+g_1h_{n-2}+\cdots+g_{n-k}h_{k-1}=0$$
$$g_{n-k}h_k=1$$

式中$g_0h_i+g_1h_{i-1}+\cdots+g_{n-k}h_{i-(n-k)}$（表示$x^i$的系数）

上式可简写为

$$g_0 h_i + g_1 h_{i-1} + \cdots + g_{n-k} h_{i-(n-k)} = 0 \quad i = 1, 2, \cdots, n-1$$

$$g_0 h_0 + g_{n-k} h_k = 0$$

我们称

$h(x) = x^n - 1 / g(x)$ 为码的校验多项式。

4、RS 码的生成矩阵

$G = [I_k \; p]$

左边是 $k \times k$ 阶单位方阵。这相当于码字多项式的第 $n-1$ 次至 $n-k$ 次的系数是信息位。而其余的位校验位。

根据前面的定义 $c(x)$ 是 $g(x)$ 的倍式

$c(x) = m(x)x^{n-k} + r(x) = 0 (\mathrm{mod}\, g(x))$ （$m(x)x^{n-k}$ 表示在信息组后面插 $n-k$ 个监督码元）

$\Rightarrow r(x) = m(x)x^{n-k} (\mathrm{mod}\, g(x))$

由 $G = [I_k \; p]$ 可知，生成矩阵里的信息组（又叫基底矢量）分别为

(100…0), (010…0),…(000…1)

所以很容易得到：

$$G = [I_k \; p] = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & x^{n-1} (\mathrm{mod}\, g(x)) \\ 0 & 1 & \cdots & 0 & 0 & x^{n-2} (\mathrm{mod}\, g(x)) \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 1 & x^{n-k} (\mathrm{mod}\, g(x)) \end{bmatrix}$$

例：

求（7，3，5）RS 码生成矩阵，G

根据 n=7,k=3,t=2 我们选取域 $GF(2^3)$ $\alpha \in GF(2^3)$

本原多项式 $p(x) = x^3 + x + 1$

α 为 $p(x)$ 的根 $p(\alpha) = \alpha^3 + \alpha + 1 = 0$ 或 $\alpha^3 = \alpha + 1$

我们可以推算出$GF(2^3)$域的全部元素

元素		二进制对应码
0	0	000
α^0	1	001
α^1	α^1	010
α^2	α^2	100
α^3	$\alpha+1$	011
α^4	$a(\alpha+1)=\alpha^2+\alpha$	110
α^5	$a(\alpha^2+\alpha)=\alpha^2+\alpha+1\ (\mathrm{mod}\ p(\alpha))$	111
α^6	$a(\alpha^2+\alpha+1)=\alpha^2+1\ (\mathrm{mod}\ p(\alpha))$	101
α^7	$a(\alpha^2+1)=1\ (\mathrm{mod}\ p(\alpha))$	001

D=5 其生成多项式为

$$g(x)=(x-\alpha)(x-\alpha^2)(x-\alpha^3)(x-\alpha^4)=x^4+\alpha^3x^3+x^2+\alpha x+\alpha^3$$

$$x^{n-1}(\mathrm{mod}\ g(x))\equiv\alpha^4x^3+x^2+\alpha^4x+\alpha^5(\mathrm{mod}\ g(x))$$

$$x^{n-2}(\mathrm{mod}\ g(x))\equiv\alpha^2x^3+x^2+\alpha^6x+\alpha^6(\mathrm{mod}\ g(x))$$

$$x^{n-3}(\mathrm{mod}\ g(x))\equiv\alpha^3x^3+x^2+\alpha x+\alpha^3(\mathrm{mod}\ g(x))$$

由此可知其生成矩阵为

$$G=\begin{bmatrix}1&0&0&\alpha^4&1&\alpha^4&\alpha^5\\0&1&0&\alpha^2&1&\alpha^6&\alpha^6\\0&0&1&\alpha^3&1&\alpha&\alpha^3\end{bmatrix}$$

5、RS 码的校验矩阵

由于系统码时生成多项式的倍式

$$C(x)=q(x)g(x)$$

$$g(x)=(x-\alpha)(x-\alpha^2)\cdots(x-\alpha^{2t})$$

所以$c(x)$必以$\alpha,\alpha^2\cdots\alpha^{2t}$为根。

即若码字

$$C(x)=c_{n-1}x^{n-1}+c_{n-2}x^{n-2}+\cdots+c_1x+c_0$$

则

$$C(\alpha^i)=c_{n-1}(\alpha^i)^{n-1}+c_{n-2}(\alpha^i)^{n-2}+\cdots+c_1(\alpha^i)+c_0 \qquad \alpha^i\in\alpha,\alpha^2\cdots\alpha^{2t}$$

由此可得出 RS 码的校验矩阵

$$H=\begin{bmatrix} \alpha^{n-1} & \alpha^{n-2} & \cdots & \alpha & 1 \\ (\alpha^2)^{n-1} & (\alpha^2)^{n-2} & \cdots & \alpha^2 & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ (\alpha^{2t})^{n-1} & (\alpha^{2t})^{n-2} & \cdots & \alpha^{2t} & 1 \end{bmatrix}$$

四、一些基本运算的软件实现

由于编码译码的运算都是在域中进行的，那么我们首先要计算出域中的元素对应的二进制或十进制表示。

MATLAB 程序如下：

```
%generate_gf,生成域中的所有元素。
 alpha_to=zeros(1,2^m);
  mask = 1 ;
  alpha_to(m+1) = 0 ;
 for i=1:m
   alpha_to(i) = mask ;
     if (Pp(i)~=0)
         alpha_to(m+1)=bitxor(alpha_to(m+1),mask);
     end;
      mask =mask*2;
  end;
mask=alpha_to(m);
for i=m+2 : n
          if (alpha_to(i-1) >= mask)
            alpha_to(i)        =bitxor(        alpha_to(m+1)        ,
bitxor(alpha_to(i-1),mask)*2 );
          else
            alpha_to(i) = alpha_to(i-1)*2 ;
          end;
end;
alpha_to(2^m) = 0;       %把元素 0 放在最后一位。
```

例如可以根据计算出$GF(2^5)$的全部元素

```
alpha_to =
1     2     4     8    16     5    10    20    13    26    17     7
14    28    29    31    27    19     3     6    12    24    21    15
30    25    23    11    22     9    18     0
```

在前面的例子中

$$\alpha+\alpha^{10}=0010+0111=0101=\alpha^{8} \quad \alpha\in GF(2^4)$$

$$\alpha^{8}\bullet\alpha^{10}=\alpha^{(8+10)\bmod 15}=\alpha^{3}$$

这样的计算看似简单，但是在实际的计算中α^{i}都是用对应的二进制表示，例如在本例中α^{8}用 0101 表示，我们并不能直观看出 0101 对应的α的指数。为了便于运算，我们计算出一个检索数组:index_of,如 index_of（α^{m}）=m ，

index_of（α^{n}）=n 。

$\alpha^{m}\bullet\alpha^{n}$=alpha_to(index_of($\alpha^{m}$)+index_of（$\alpha^{n}$）)

这样以来运算就变得简单多了。

index_of 检索数组可以由下面程序得出。

```
index_of=zeros(1,2^m);
for i=1:2^m-1
    index_of(alpha_to(i))=i-1;
end
index_of
```

例 m=5

```
index_of =
0     1    18     2     5    19    11     3    29     6    27    20
8    12    23     4    10    30    17     7    22    28    26    21
25     9    16    13    14    24    15     0
```

加法运算的实现：

```
function y=rs_add(a,b)

a1=de2bi(a,m);     %参数 m 为GF(2^m)域本原多项式最高次数，下同。

b1=de2bi(b,m);

y1=bitxor(a1,b1);

y=bi2de(y1);
```

乘法运算就可以通过下面的程序很简单的实现。

```
function y=rs_mul(a,b)
% for 'a' or 'b' is 0, output 'y' is 0
if a*b==0
    y=0;
else
     a1=index_of(a);
     b1=index_of(b);
     c=mod((a1+b1),(2^m-1));%参数(2^m-1)，不能用码长 n 代替，因为实际应
                            %用中 n 并不一定等于(2^m-1)
     y=alpha_to(c+1);
end
```

倒数运算的实现：

```
function y=rs_rev(a)
% check the table, represent a as a power of primitive root
% reduce the power from 31, get result of the reciprocal of a
% check the table and turn the result to a decimal number
a1=index_of(a);
y1=mod((2^m-1)-a1 , 2^m-1)+1;
y=alpha_to(y1);
```

把 x 代入多项式 f(x)即计算 f(x)的值（x 为一常数）。
程序中 T 为 GF 中元素对应的二进制表示值。

```
function y=rs_poly(f,x)
xx= index_of (x)-1;
L=length(f)-1;       %多项式的次数
y1=f(1);            %常数项
for i=1:L
    y1=rs_add(y1,rs_mul(f(i+1), alpha_to (mod(i*xx,n)+1)));  %累加
end
y=y1;
```

多项式的乘法运算：

```
function  p=rs_polymul(a,b)
%本函数实现多项式的相乘
   h=length(a)+length(b)-1;
   for i=1:h
       p(i)=0;
       for j=1:length(a)
           for l=1:length(b)
               if(j+l==i+1)
                  p(i)=rs_add(p(i),rs_mul(a(j),b(l)));
               end;
            end;
        end;
    end;
```

以下示例程序中的函数 rs_add、rs_mul、rs_rev 、rs_poly、rs_polymul 都表示上述程序，不再重复给出。

五、RS 编码

RS 编码软件实现，

其编码电路基本上可分为两类，k 级和 $n-k$ 级编码器

1、 $n-k$ 级编码器

其原理比较容易理解

由于系统码时生成多项式的倍式

$$C(x)=q(x)g(x)$$

$$c=(r_1,r_2,...r_{n-k},m_1,m_2,...m_k)$$

$$C(x)=q(x)g(x)=m(x)x^{n-k}+r(x)$$

信息码乘以 x^{n-k} 然后再除 $g(x)$，就得到了余式 $r(x)$，也就得到了校验位。

这里难点就是如何实现多项式乘法，除法运算。

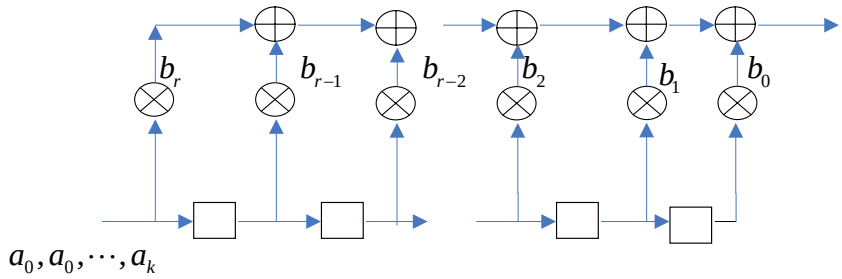
多项式乘法

设两多项式

$$A(x)=a_kx^k+a_{k-1}x^{k-1}+\cdots+a_1x+a_0$$

$$B(x)=b_rx^r+b_{r-1}x^{r-1}+\cdots+b_1x+b_0$$

相乘过程可用下图表示



图（1）

实现步骤如下：

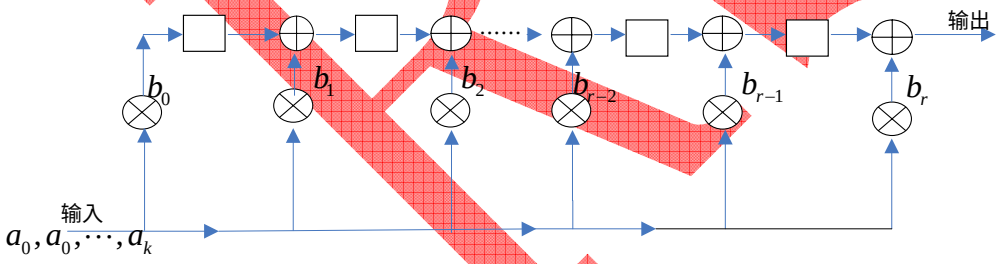
（1）r 个寄存器全部清 0。

（2）$A(x)$ 最高次系数 a_k 首先送入时，乘法器输出乘积的最高次项 x^{k+r} 的系数 $a_k b_r$，同时 a_k 存入寄存器的第一级。

（3）$A(x)$ 的第二个系数 a_{k-1} 送入时，a_k 由第一级进入第二级寄存器，同时与 b_{r-1} 相乘，$a_{k-1}b_r + a_k b_{r-1}$ 就得到了乘积的 x^{k+r-1} 的系数。

（4）这样重复进行，直至 $k+r+1$ 次移位后，乘法器输出乘积的常数项 $a_0 b_0$。

乘法过程还有另外一种表示方法。



图（2）

它的工作过程和图（1）类似。

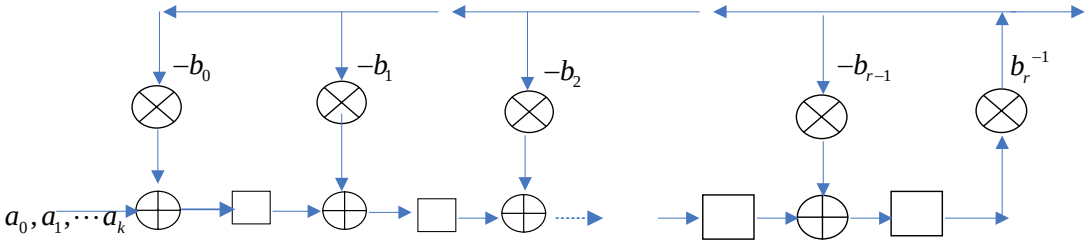
多项式除法：

设

$$A(x) = a_k x^k + a_{k-1} x^{k-1} + \cdots + a_1 x + a_0$$

$$B(x) = b_r x^r + b_{r-1} x^{r-1} + \cdots + b_1 x + b_0$$

流程图：



图（3）

（1） 开始运算时 r 个寄存器全部清 0，第一次移位时， $A(x)$ 最高次系数 a_k 首先进入最左一级寄存器， r 次移位后，寄存器 1 至寄存器 2t 的数值分别为

$a_{k-r+1}, a_{k-r+2}, \cdots a_{k-1}, a_k$

（2） 第 $r+1$ 次移位时，除法器输出 $a_k b_r^{-1}$，这就是商的第一项 x^{k-r} 的系数， $a_k b_r^{-1}$ 同时反馈到后面的各级寄存器中，即得到第一次除运算的余式。

（3） 以此类推，经 k 次移位后，完成了整个除法运算过程，移位寄存器中的值就是余式 $r(x)$

多项式相乘相除

分别设

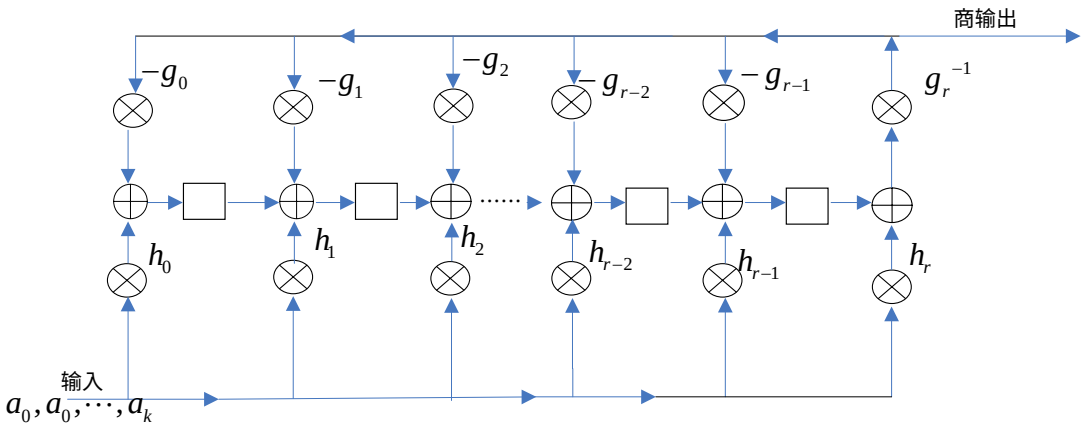
$$A(x) = a_k x^k + a_{k-1} x^{k-1} + \cdots + a_1 x + a_0$$

$$H(x) = h_r x^r + h_{r-1} x^{r-1} + \cdots + h_1 x + h_0$$

$$G(x) = g_r x^r + g_{r-1} x^{r-1} + \cdots + g_1 x + g_0$$

计算 $A(x) \bullet H(x) / G(x)$

该电路图是图（2）图（3）两种电路的结合。



图（4）

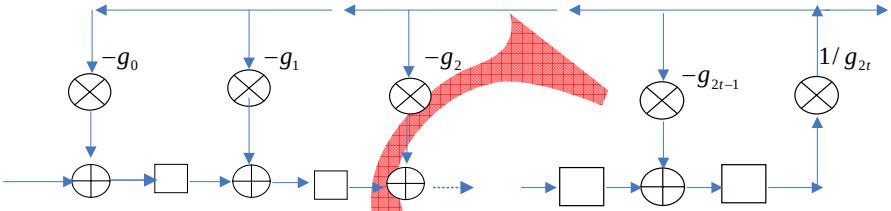
如果 $H(x),G(x)$ 次数不等，只需按最高次数设计即可。

RS 编码电路

$$g(x)=g_{2t}x^{2t}+g_{2t-1}x^{2t-1}+\cdots+g_1x+g_0 \qquad (g_{2t}=1)$$

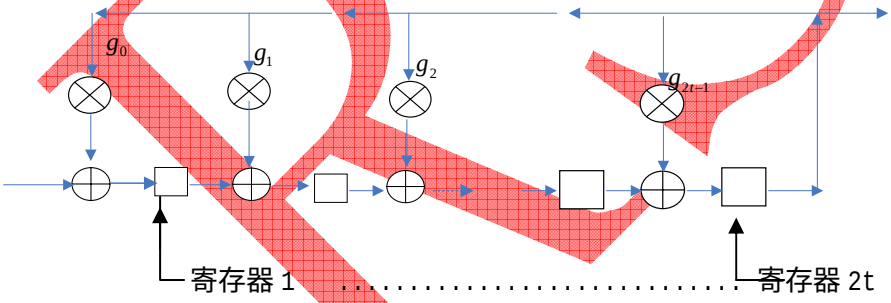
$$m(x)x^{n-k}=m_kx^{n-1}+m_{k-1}x^{n-2}+\cdots+m_1x^{n-k}+0x^{n-k-1}+\cdots+0x+0$$

那么 $m(x)x^{n-k}/g(x)$ 的除法电路根据图（3）可用下图表示



输入为 $m(x)x^{n-k}$

因为 $g_{2t}=1$，加法减法运算规则相同， 所以又可以表示为



上述方法相当于事先做好 $m(x)x^{n-k}$，再做除法，这样需要移位 n 次，

而图（4）的电路我们只需移位 k 次。

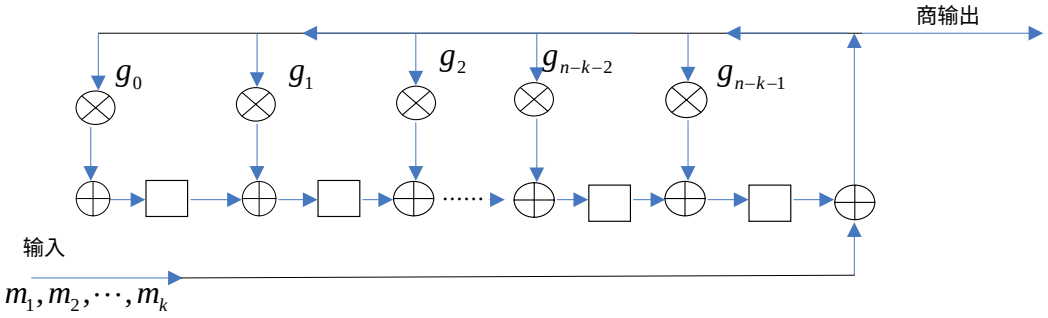
根据

$$g(x)=g_{2t}x^{2t}+g_{2t-1}x^{2t-1}+\cdots+g_1x+g_0=g_{n-k}x^{n-k}+g_{n-k-1}x^{n-k-1}+\cdots+g_1x+g_0$$

$g_{n-k}=1$ 输入

$$x^{n-k}=x^{n-k}+0x^{n-k-1}+\cdots+0x+0$$

根据图（4），$m(x)x^{n-k}/g(x)$ 的电路可表示为



MATLAB 程序实现

```
rr=zeros(1, n-k);
  for i=k:-1:1
    feedback = rs_add(data(i),rr(nn-kk)) ;
     if (feedback ~= 0)
          for j=n-k-1:-1:1
            if (g(j) ~= 0)
              rr(j+1) =rs_add( bb(j),rs_mul((g(j),feedback) ;
            else
              rr(j+1) = rr(j) ;
            end;
         rr(1) = rs_mul (gg(1),feedback) ;
        end;
  else
    for j=n-k-1:-1:1
        rr(j+1) = rr(j) ;
    end;
    rr(1) = 0 ;
end;
end;
```

下面的程序是另一种比较简洁的程序

```
rr=zeros(1,n-k);
for i=1:k
    feedback =rs_add(rr(n-k),data(k-i+1));
    rr(n-k)=rs_add(rr(n-k-1),rs_mul(feedback,g(n-k)));
    rr(n-k-1)=rs_add(rr(n-k-2),rs_mul(feedback,g(n-k-1)));
    rr(n-k-2)=rs_add(rr(n-k-3),rs_mul(feedback,g(n-k-2)));
    ..........................................
    rr(3)=rs_add(rr(2),rs_mul(feedback,g(3)));
    rr(2)=rs_add(rr(1),rs_mul(feedback,g(2)));
    rr(1)=rs_mul(feedback,g(1));
end
```

2、k 级编码器

(n,k) RS 码也可根据校验多项式 $h(x)=h_k x^k+h_{k-1}x^{k-1}+\cdots+h_1 x+h_0$ 构造，

设系统码 $C(x)=c_{n-1}x^{n-1}+c_{n-2}x^{n-2}+\cdots+c_1x+c_0$

$C(x)=q(x)g(x)$

$h(x)C(x)=q(x)g(x)h(x)=q(x)(x^n-1)=q(x)x^n-q(x)$

由于 $C(x)$ 的次数 $\le n-1$，$g(x)$ 的次数 $=n-k$，$q(x)$ 的次数 $\le k-1$

多项式 $q(x)x^n$ 最低次数 $\ge n$，

所以 $h(x)C(x)$ 乘积中，$x^{n-1},x^{n-2},\cdots x^k$ 的系数应为 0

x^{n-1} 的系数为 $c_{n-1}h_0+c_{n-2}h_1+\cdots+c_{n-1-k}h_k$

x^{n-2} 的系数为 $c_{n-2}h_0+c_{n-3}h_1+\cdots+c_{n-2-k}h_k$

不难得出

$$\sum_{j=0}^{k}c_{n-i-j}h_j=0 \qquad i=1,2,\cdots,n-k$$

由于

根据 $h(x)=(x^n-1)/g(x)$ 可以看出 $h(x)$ 的最高次项系数为 1

所以上式又可写为

$$c_{n-k-i}=\sum_{j=0}^{k-1}c_{n-i-j}h_j$$

MATLAB 程序实现

```
  t_x(n-k+1:n)=m_x;    %m_x 为信息组,n 为码长，k 为信息码长
for i=1:n-k
       for j=1:k
          data= t_x (n-i-j+2);
           if data~=0
            h=h_x(j);
           y=bitxor(y,mod(index_of (data)+ index_of (h),n));
           end;
       end;
       t_x(n-k-i+1)=alpha_to(y+1);
 end;
```

特殊说明：

对于码长 $n\neq 2^m-1$ 的情况，编程相当于在信息码后补 $(2^m-1)-n$ 个 0 做编码，例如对序

列：

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,0,0,0,0,0

做（31，25，7）编码，生成：

26 30 30 23 13 2 1 2 3 4 5 6 7

8 9 10 11 12 13 14 15 16 17 18 19 20

0 0 0 0 0

和对序列

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20

做（26，20，7）编码 ，生成：

26 30 30 23 13 2 1 2 3 4 5 6 7

8 9 10 11 12 13 14 15 16 17 18 19 20

我们可以看出他们的校验码是一样的。

六、RS 译码

RS 码的译码过程可分为以下几步

1） 由接收到的$r(x)$求得s_j

2） 由s_j求得错误位置多项式$\Lambda(x)$

3） 用钱搜索解出$\Lambda(x)$的根，得到错误位置数。确定错误位置

4） 由错误位置数求得错误值，从而得到错误图样

5） $r-\hat{e}$ 完成纠错

1、 由接收到的$r(x)$求伴随多项式S_j

这一步比较简单

由前面可知

$$S^T = HR^T = \begin{bmatrix} \alpha^{n-1} & \alpha^{n-2} & \cdots & \alpha & 1 \\ (\alpha^2)^{n-1} & (\alpha^2)^{n-2} & \cdots & \alpha^2 & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ (\alpha^{2t})^{n-1} & (\alpha^{2t})^{n-2} & \cdots & \alpha^{2t} & 1 \end{bmatrix} \begin{bmatrix} c_{n-1} \\ \vdots \\ c_1 \\ c_0 \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_{2t} \end{bmatrix}$$

$$H=\begin{bmatrix} \alpha^{n-1} & \alpha^{n-2} & \cdots & \alpha & 1 \\ (\alpha^2)^{n-1} & (\alpha^2)^{n-2} & \cdots & \alpha^2 & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ \left(\alpha^{2t}\right)^{n-1} & \left(\alpha^{2t}\right)^{n-2} & \cdots & \alpha^{2t} & 1 \end{bmatrix}$$

$S=RH^T$

设接收矢量 $R(x)=c_{n-1}x^{n-1}+c_{n-2}x^{n-2}+\cdots+c_1x+c_0$

把 $x=\alpha$ 代入 $R(x)$ 得到 S_1

把 $x=\alpha^2$ 代入 $R(x)$ 得到 s_2 ……..

……..

把 $x=\alpha^{2t}$ 代入 $R(x)$ 得到 s_{2t}

$S=RH^T=\left[R(\alpha),R(\alpha^2),\cdots R(\alpha^{2t})\right]$

或表示为 $S_j=R(\alpha^j)\quad j=1,2,\cdots,2t$

计算伴随多项式 S

```
% a function to calculate the syndrome polynomial accoding to the received sequence
'r_x'
s=zeros(1,nn-kk);
for j=1: nn-kk
   s(j)=rs_poly(r_x, alpha_to (j+1));  % use the function rs_poly(t,x) to calculate
s(j)=r(a^j)
end
```

2、 Berlekamp-Massey Algorithm 求错误位置多项式

译码无非式通过 2t 个方程，求出 2t 未知数， x_i y_i $i=1,2,\cdots t$

x_i 表示错误位置 y_i 表示错误值

所以要先求出错误位置数 x_i ，再求错误值 y_i

为此引入错误位置多项式。

$$\Lambda(x)=(1-X_1x)(1-X_2x)\cdots(1-X_\nu x)$$

$$=\prod_{l=1}^{\nu}(1-X_l x)=\Lambda_\nu x^\nu+\Lambda_{\nu-1}x^{\nu-1}+\cdots+\Lambda_1 x+\Lambda_0$$

根为$1/X_1$，$1/X_2$，$\cdots 1/X_\nu$　错误位置为 X_1，$X_2 \cdots X_\nu$

例如在（15，9）RS 码中，错误位置多项式根为$\alpha^3,\alpha^9,\alpha^{12}$，他们的倒数即为错误位置$\alpha^{12}$，$\alpha^6$，$\alpha^3$，即错误发生在$x^{12}$，$x^6$，$x^3$上。

Berlekamp-Massey Algorithm(BM 迭代算法)能够快速根据伴随多项式 S 求$\Lambda(x)$，且容易实现。

迭代步骤如下：

1、 由初始值

$\Lambda^{(-1)}(x)=1$，$D(-1)=0$，$d_{-1}=1$

$\Lambda^{(0)}(x)=1$，$D(0)=0$，$d_0=s_1$

开始迭代

2、 按$d_j=s_{j+1}+\sum_{i=1}^{\partial\cdot\Lambda^{(j)}(x)} s_{j+1-i}\Lambda_i^{(j)}$ 计算d_j，若$d_j=0$，则有

$\Lambda^{(j+1)}(x)=\Lambda^{(j)}(x)$　，　$D^*(j+1)=D^*(j)$

并计算d_{j+1}，再进行下一次迭代

如果$d_j\neq 0$，则找出 j 之前的某一行 i，它在所有 j 行之前各行中的$i-D(i)$最大，且$d_i\neq 0$，于是按

$\Lambda^{(j+1)}(x)=\Lambda^{(j)}(x)-d_j d_i^{-1}x^{j-i}\Lambda^{(i)}(x)$ 计算 $\Lambda^{(j+1)}(x)$

这就是第 $j+1$ 步的解，

3、 计算 d_{j+1}，重复 2 步进行下一次迭代，这样迭代 2t 次后得到的$\Lambda^{(2t)}(x)$即为所求的$\Lambda(x)$。

求$\Lambda(x)$的迭代过程表

j	$\Lambda^{(j)}(x)$	$D(j)$	$j-D(j)$	d_j
−1 0	1 1	0 0	-1 0	1 s_1
1 2 ⋮ 2t				

d_j 为第 $j+1$ 步与第 j 步的差值。 $D(j)$ 为多项式 $\Lambda^{(j)}(x)$ 的次数。

也许你看了上述步骤还不是太明白，那么我们来看下它的 MATLAB 程序吧。

MATLAB 程序实现

```
lambda=zeros(nn-kk+2, nn-kk+1);    % lambd 矩阵用于记录 Λ(x)，最后一行为最终结果。
lambda (1,1)=1;            %错误位置多项式初始值
lambda (2,1)=1;
D=zeros(1, nn-kk+2);         %D 矩阵记录 D(j)
D(1)=0;
D(2)=0;
d=zeros(1, nn-kk+2);          %d 记录 d_j
d(1)=1;
d(2)=synd_x(2);            % 数组 synd_x 代表伴随多项式 s0=1, s1,s2…s2t-1
%为保持统一，我们规定多项式都是从 0 次项开始。所以伴随多项式 synd_x(0)＝1（当然也
%可以为其他值，因为它并不参与运算）；
flag=-1;                  %flag 记录 j 行之前的 i-D(i) 最大，且 d_i≠0，的位置
j_D=-1;
for j=0: nn-kk-1
  if d(j+2)==0           %迭代步骤第二步，若 d_j＝0
       lambda(j+2+1,:)=lambda (j+2,:);
       D(j+2+1)=D(j+2);
  else                  %  d_j≠0
     temp=circshift(lambda(flag+2,:),[0 j-flag]);      %计算公式中的 x^(j-i)Λ^(i)(x)
                                                       %Λ^(i)(x) 左移 j-i 位
     for i=1:nn-kk+1                                   %Λ^(j+1)(x)=Λ^(j)(x)-d_j d_i^-1 x^(j-i) Λ^(i)(x)
       rev_flag= rs_rev(d(flag+2));
       mul_temp= rs_mul(d(j+2), rev_flag);
       lambda(j+2+1,i)=rs_add(lambda(j+2,i),rs_mul(mul_temp,temp(i)));
     end

D(j+2+1)=max(D(+2),j-flag+D(flag+2));

     if j-D(j+2)>=j_D
       j_D=j-D(j+2);
```

```
        flag=j;
       end;
    end

if j~=nn-kk-1
    r=j+1;

    d(j+2+1)=synd_x(j+1+1+1);              %计算 d_j = s_{j+1} + sum_{i=1}^{∂·Λ^(j)(x)} s_{j+1-i} Λ_i^(j)

    for z=1:D(j+2+1)
    d(j+2+1)=rs_add(d(j+2+1),rs_mul(lambda(j+2+1,z+1),synd_x(j+1+1-z+1)));
 end
end
end
lambda_x= lambda(nn-kk+2,1:(D(nn-kk+2)+1))
 % lambda 矩阵最后一行即为所求错误位置多项式
```

3、求错误多项式的根

求得 $\Lambda(x)$　下一步的问题就是从工程的观点看，如何简单地求出它的根即错误位置。

下面介绍一个实用的方法：
钱搜索解错误多项式的根
这部分比较容易理解，在此做简要说明

$$\Lambda(x)=(1-X_1x)(1-X_2x)\cdots(1-X_\nu x)$$

$$=\prod_{l=1}^{\nu}(1-X_l x)=\Lambda_\nu x^\nu+\Lambda_{\nu-1}x^{\nu-1}+\cdots+\Lambda_1 x+\Lambda_0$$

首先验证位置 x^0 是否有错误，即把 $x=\frac{1}{\alpha^0}=\alpha^n$ （$\alpha^n=1$）代入 $\Lambda(x)$ 若结果等于 0，说明第一个码元有错误，否则第一个码元是正确的。

把 $x=\frac{1}{\alpha^1}=\alpha^{n-1}$ 代入 $\Lambda(x)$，验证位置 x^1 是否有错误。

同理把 $\alpha^{n-2},\alpha^{n-3}\ldots\alpha$ 代入 $\Lambda(x)$ 分别验证位置 $x^2\cdots x^{n-1}$ 是否错误。

```
% a function to calculate the roots of locator polynomial
j=1;
for i=1:n
```

```
    result=rs_poly(lambda_x, alpha_to (i));
        if result==0
            root(j)= alpha_to (i);
            j=j+1;
        end
end
```

根的倒数即为错误位置

4 、根据 Forney 算法计算错误图样：

根据前面的计算,我们定义伴随多项式：

$$S(x) = s_1 + s_2 x + s_3 x^2 + \cdots + s_{2t} x^{2t-1}$$

定义错误位置多项式：

$$\Lambda(x) = \prod_{l=1}^{\nu} (1 - X_l x) = \Lambda_\nu x^\nu + \Lambda_{\nu-1} x^{\nu-1} + \cdots + \Lambda_1 x + \Lambda_0$$

式中 $\Lambda_0 = 1$， if $x = X_l^{-1}$ then $\Lambda(x) = 0$ 。

定义求值多项式：

$$\Omega(x) = S(x)\Lambda(x) \pmod{x^{2t}}$$

此等式被称作求错误值的关键等式。

Note ： the effect of computing modulo x^{2t} is to discard all terms of degree ***2t*** or higher.

(Forney's algorithm)：RS码错误值计算公式：

$$e_{ik} = -\frac{\Omega(X_k^{-1})}{\Lambda'(X_k^{-1})}$$

$\Lambda'(x)$ 为 $\Lambda(x)$ 的形式导数

——

注：有些作者定义 $S(x) = s_1 x + s_2 x^2 + s_3 x^3 + \cdots + s_{2t} x^{2t}$ 在这种情况下

$$\Omega(x) = (1 + S(x))\Lambda(x) \pmod{x^{2t+1}} \quad ， \quad e_{ik} = -\frac{X_k \Omega(X_k^{-1})}{\Lambda'(X_k^{-1})}$$

——

例：

在 GF(8)中 RS 码 设 t=2 ， $S(x) = \alpha^6 + \alpha^3 x + \alpha^4 x^2 + \alpha^3 x^3$ 我们可以计算出（using the B-M algorithm and the Chien search）错误位置多项式：

$$\Lambda(x) = 1 + \alpha^2 x + \alpha x^2 = (1 + \alpha^3 x)(1 + \alpha^5 x)$$

由此我们可以看出错误位置为 $X_1=\alpha^3$ ， $X_2=\alpha^5$

根据 Forney 算法公式

$$\Omega(x)=(\alpha^6+\alpha^3x+\alpha^4x^2+\alpha^3x^3)(1+\alpha^2x+\alpha x^2) \pmod{x^4}$$

$$=(\alpha^6+x+\alpha^4x^2) \pmod{x^4} \quad =\alpha^6+x$$

错误多项式的导式：

$\Lambda'(x)=\alpha^2+2\alpha x=0$ 根据加法规则 $2\alpha x=\alpha x+\alpha x=0$

所以错误多项式的导数可以从错误多项式的奇数项中得到。

所以 $e_{ik}=-\left.\frac{\alpha^6+x}{\alpha^2}\right|_{x=X_k^{-1}}=\alpha^4+\alpha^5X_k^{-1}$

把 $X_1=\alpha^3$ 代入 得到 $e_3=\alpha$

把 $X_2=\alpha^5$ 代入得到 $e_5=\alpha^5$

错误图样多项式：$e(x)=\alpha x^3+\alpha^5x^5$

MATLAB 程序：

```
% according to syndrome polynomial & locator polynomial & roots of
% 程序中 synd_x 为伴随多项式，lambda_x 为错误位置多项式，root 为错误位置多项式的根
value=zeros(1,tt);
w=zeros(1,length(synd_x)+length(lambda_x)-1);
% 'tt' is the max number of errors
% calculate the value of w(j)
for i=1:length(synd_x)-1
    for j=1:length(lambda_x)
        w(i+j-1)=rs_add(w(i+j-1),rs_mul(synd_x(i+1), lambda_x(j)));
    end
end
% evaluator polynomial
omega=w(1:2*tt);
s=zeros(1,length(lambda_x));
% differential coefficient of locator polynomial
for h=1:length(lambda_x)
    if mod(h,2)==0
```

```
        lambda_dx(h-1)= lambda_x(h);
    end
end
lambda_dx(length(lambda_x))=0;
for k=1:length(root)
    omega_final=rs_poly(omega,root(k));
    lambda_final=rs_poly(lambda_dx,root(k));
% error value
    value(k)=rs_mul(omega_final,rs_rev(lambda_final));
end
```

5、$r-\hat{e}$ 完成纠错

```
t_x=r_x;
for i=1:length(site)
        t_x(site(i)+1)=rs_add(r_x(site(i)+1),value(i));
end
```

程序中 site 表示错误位置，value 表示错误图样，r_x 表示接收序列，t_x 表示解码输出序列。

附录

下面给出作者经过调试的 n=26,k=20,t=3,m=5、RS 编译码 MATLAB 程序，为了保证通用性，作者特意选择 $n \neq 2^m - 1$。当然你可以随意更改码长，经作者测试，他们运算都是正确的。 本程序共有 8 个文件，下面一一列出，读者只要照着写一遍，那么你的 RS 编译码程序就诞生了。如果你想更省事点，那就直接来信索取吧。

```
%rs_main.m
clear all;

nn=26;      %码长
mm=5;
kk=20;      %信息码长
tt=3  ;      %能纠正的错误个数

alpha_to =[1 ,2 , 4,  8, 16, 5, 10, 20, 13 , 26 ,17, 7,14,28,29, 31,27 ,19 ,3 , 6 ,
12, 24,21 ,15,30 ,25,23 ,11, 22 , 9 ,18, 0];


index_of =[ 0 ,1,18,  2 ,  5 ,19 ,  11,3 ,29,6,27,20,  8,12,  23,4,10,  30,17,7,
22,28,26,21,25,9,16,13, 14,24,15, 0];


gx =[24,30,27,30,26 ,17,1];

m_xi=zeros(1,kk);

for i=1:kk
    m_xi(i)=i;
end;


     t_x=encode(m_xi,nn,kk,mm,alpha_to,index_of,gx);


 t_x
disp('信道产生的错误位置：')
site=[0 5 12]
disp('信道产生的错误数值：')
err_value=[1 2 3]
disp('接收序列')
r_x=t_x;
```

```
   for i=1:3
        r_x(site(i)+1)=rs_add(t_x(site(i)+1),err_value(i),mm);
   end

r_x

m_xo=decoder(r_x,nn,kk,mm,tt,m_xi,alpha_to,index_of,gx)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%




% encode.m
%利用校验多项式 h(x)编码  即 k 级 编码

 function t_x=encode(m_xi,n,k,m,alpha_to,index_of,gx)

 %K 级编码
 r=zeros(1,n-k);
 for i=k:-1:1
 feedback=rs_add(m_xi(i),r(n-k),m) ;
   for j=n-k:-1:2
       if (gx(j) ~= 0)
        r(j) =rs_add(r(j-1),rs_mul(gx(j),feedback,m,alpha_to,index_of),m) ;
       else
        r(j) = r(j-1) ;
       end;
   end;
     r(1) = rs_mul (gx(1),feedback,m,alpha_to,index_of) ;


 end;
t_x=[r(1:n-k),m_xi];                              %低地址对应多项式表示的低次项




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% decoder.m
%%%%%%%%%%RS 译码

function m_xo=decoder(r_x,n,k,m,t,m_xi,alpha_to,index_of,gx)

% disp('伴随多项式：')
%%%%%%%%%%%%%%%计算伴随多项式 S=H*R           calculate the syndrome polynomial
accoding to the received sequence 'r_x'
% all calculations are in the GF(2^m)
 s=zeros(1,n-k);    %n-k 个校正子（伴随式）
for j=1:n-k
   s(j)=rs_poly(r_x,alpha_to(j+1),m,alpha_to,index_of);    % use the function
rs_poly(t,x) to calculate s(j)=r(a^j)
end
synd_x=[1,s];

synd_x

disp('错误位置多项式：')
%用迭代算法 计算错误位置多项式这一步也是最复杂的一步也是决定译码速度的主要因素。
sigma=zeros(n-k+2,n-k+1); %sigma 矩阵用于记录错误多项式 ，最后一行为最终结果。
step=zeros(1,n-k+2); %step （i）表示第 i 次叠带产生的错误位置多项式的阶数
high=zeros(1,n-k+2); %,high 是第 i+1 次叠带的最高项系数
sigma(1,1)=1;
step(1)=0;
high(1)=1;
sigma(2,1)=1;
step(2)=0;
high(2)=synd_x(2);

for j=2:n-k+1

    if (high(j)==0)
        sigma(j+1,:)=sigma(j,:);
        step(j+1)=step(j);
    else
        for h=1:j-1
            if(high(h)~=0)
                i=h;
            end;
        end;
        temp=zeros(1,j-i+1);
        temp(j-i+1)=1; %a^(j-i)的多项式表示
        temp1=rs_polymul(temp,sigma(i,:),m,alpha_to,index_of); %多项式相乘。
```

```
        len=length(temp1);
        temp2=zeros(1,n-k+1);
        temp2=temp1(1:n-k+1);

sigma(j+1,:)=sigma(j,:)+high(j)*rs_rev(high(i),m,alpha_to,index_of)*temp2;

temp4=rs_mul(high(j),rs_rev(high(i),m,alpha_to,index_of),m,alpha_to,index_of);
        for l=1:n-k+1

sigma(j+1,l)=rs_add(sigma(j,l),rs_mul(temp4,temp2(l),m,alpha_to,index_of),m);
        end;

    end;
        for h=1:n-k+1
            if (sigma(j+1,h)~=0)
                step(j+1)=h-1;
            end;
        end;
        for h=1:step(j+1)
            temp3=rs_mul(sigma(j+1,h+1),synd_x(j+1-h),m,alpha_to,index_of);
            high(j+1)=rs_add(high(j+1),temp3,m);
        end;
        if(j+1<n-k+2)
        high(j+1)=rs_add(high(j+1),synd_x(j+1),m);
        end;
end;
sigma_x=sigma(n-k+2,:);

sigma_x

j=1;
root=[];
for i=1:(2^m-1)
    result=rs_poly(sigma_x,alpha_to(i),m,alpha_to,index_of);
        if result==0
            root(j)=alpha_to(i);
            j=j+1;
        end
end
 root

site=[];
gf_site=[];
for i=1:length(root)
```

```
   temp=rs_rev(root(i),m,alpha_to,index_of);        %根的倒数即为错误位置
   site(i)=index_of(temp);
end
%site

% for i=1:length(root)
%    gf_site(i)=alpha_to(site(i)+1);
% end
% gf_site

%%%%%%%%%%第四步根据伴随式，错误多项式及错误位置计算错误值，从而得到错误图样 E
value=zeros(1,t);
w1=zeros(1,length(synd_x)+length(sigma_x)-1);
% 't' is the max number of errors

% calculate the value of w(j)
for i=1:length(synd_x)-1
    for j=1:length(sigma_x)

w1(i+j-1)=rs_add(w1(i+j-1),rs_mul(synd_x(i+1),sigma_x(j),m,alpha_to,index_of),m
);
    end
end

% evaluator polynomial
%w1
w=w1(1:2*t);

s=zeros(1,length(sigma_x));
% differential coefficient of locator polynomial
for h=1:length(sigma_x)
    if mod(h,2)==0
        s(h-1)=sigma_x(h);
    end
end
s(length(sigma_x))=0;
for k=1:length(root)
    w_final=rs_poly(w,root(k),m,alpha_to,index_of);
    s_final=rs_poly(s,root(k),m,alpha_to,index_of);
    % error value

value(k)=rs_mul(w_final,rs_rev(s_final,m,alpha_to,index_of),m,alpha_to,index_of
);
```

```
end

temp=[];
m_xo=[];
   temp=r_x;
%%%%%%%%%%第五步 R-E=C 完成纠错过程
 for i=1:length(site)
        temp(site(i)+1)=rs_add(r_x(site(i)+1),value(i),m);
 end

%        disp('解码输出：')
       m_xo=temp;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%




% rs_add.m
function y=rs_add(a,b,m)
a1=de2bi(a,m);
b1=de2bi(b,m);
y1=bitxor(a1,b1);
y=bi2de(y1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%




% rs_mul.m
function y=rs_mul(a,b,m,alpha_to,index_of)
if a*b==0
    y=0;
else
     a1=index_of(a);
     b1=index_of(b);
     c=mod((a1+b1),(2^m-1));
     y=alpha_to(c+1);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%




% rs_rev.m
```

```
function y=rs_rev(a,m,alpha_to,index_of)
a1=index_of(a);
y1=mod((2^m-1)-a1 , 2^m-1)+1;
y=alpha_to(y1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% rs_poly.m
function y=rs_poly(f,x,m,alpha_to,index_of)
xx=index_of(x);
L=length(f)-1;
y1=f(1);
for i=1:L

y1=rs_add(y1,rs_mul(f(i+1),alpha_to(mod(i*xx,(2^m-1))+1),m,alpha_to,index_of),m
);
end
y=y1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% rs_polymul.m
function  p=rs_polymul(a,b,m,alpha_to,index_of)
%本函数实现多项式的相乘
   h=length(a)+length(b)-1;
   for i=1:h
       p(i)=0;
       for j=1:length(a)
           for l=1:length(b)
               if(j+l==i+1)
                  p(i)=rs_add(p(i),rs_mul(a(j),b(l),m,alpha_to,index_of),m);
               end;
           end;
        end;
    end;
```