1.CRC校验算法

```
const unsigned int crc_table[256] = {
    0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50a5, 0x60c6, 0x70e7,
    0x8108, 0x9129, 0xa14a, 0xb16b, 0xc18c, 0xd1ad, 0xe1ce, 0xf1ef,
    0x1231, 0x0210, 0x3273, 0x2252, 0x52b5, 0x4294, 0x72f7, 0x62d6,
    0x9339, 0x8318, 0xb37b, 0xa35a, 0xd3bd, 0xc39c, 0xf3ff, 0xe3de,
    0x2462, 0x3443, 0x0420, 0x1401, 0x64e6, 0x74c7, 0x44a4, 0x5485,
    0xa56a, 0xb54b, 0x8528, 0x9509, 0xe5ee, 0xf5cf, 0xc5ac, 0xd58d,
    0x3653, 0x2672, 0x1611, 0x0630, 0x76d7, 0x66f6, 0x5695, 0x46b4,
    0xb75b, 0xa77a, 0x9719, 0x8738, 0xf7df, 0xe7fe, 0xd79d, 0xc7bc,
    0x48c4, 0x58e5, 0x6886, 0x78a7, 0x0840, 0x1861, 0x2802, 0x3823,
    0xc9cc, 0xd9ed, 0xe98e, 0xf9af, 0x8948, 0x9969, 0xa90a, 0xb92b,
    0x5af5, 0x4ad4, 0x7ab7, 0x6a96, 0x1a71, 0x0a50, 0x3a33, 0x2a12,
    0xdbfd, 0xcbdc, 0xfbbf, 0xeb9e, 0x9b79, 0x8b58, 0xbb3b, 0xab1a,
    0x6ca6, 0x7c87, 0x4ce4, 0x5cc5, 0x2c22, 0x3c03, 0x0c60, 0x1c41,
    0xedae, 0xfd8f, 0xcdec, 0xddcd, 0xad2a, 0xbd0b, 0x8d68, 0x9d49,
    0x7e97, 0x6eb6, 0x5ed5, 0x4ef4, 0x3e13, 0x2e32, 0x1e51, 0x0e70,
    0xff9f, 0xefbe, 0xdfdd, 0xcffc, 0xbf1b, 0xaf3a, 0x9f59, 0x8f78,
    0x9188, 0x81a9, 0xb1ca, 0xa1eb, 0xd10c, 0xc12d, 0xf14e, 0xe16f,
    0x1080, 0x00a1, 0x30c2, 0x20e3, 0x5004, 0x4025, 0x7046, 0x6067,
    0x83b9, 0x9398, 0xa3fb, 0xb3da, 0xc33d, 0xd31c, 0xe37f, 0xf35e,
    0x02b1, 0x1290, 0x22f3, 0x32d2, 0x4235, 0x5214, 0x6277, 0x7256,
    0xb5ea, 0xa5cb, 0x95a8, 0x8589, 0xf56e, 0xe54f, 0xd52c, 0xc50d,
    0x34e2, 0x24c3, 0x14a0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
    0xa7db, 0xb7fa, 0x8799, 0x97b8, 0xe75f, 0xf77e, 0xc71d, 0xd73c,
    0x26d3, 0x36f2, 0x0691, 0x16b0, 0x6657, 0x7676, 0x4615, 0x5634,
    0xd94c, 0xc96d, 0xf90e, 0xe92f, 0x99c8, 0x89e9, 0xb98a, 0xa9ab,
    0x5844, 0x4865, 0x7806, 0x6827, 0x18c0, 0x08e1, 0x3882, 0x28a3,
    0xcb7d, 0xdb5c, 0xeb3f, 0xfb1e, 0x8bf9, 0x9bd8, 0xabbb, 0xbb9a,
    0x4a75, 0x5a54, 0x6a37, 0x7a16, 0x0af1, 0x1ad0, 0x2ab3, 0x3a92,
    0xfd2e, 0xed0f, 0xdd6c, 0xcd4d, 0xbdaa, 0xad8b, 0x9de8, 0x8dc9,
    0x7c26, 0x6c07, 0x5c64, 0x4c45, 0x3ca2, 0x2c83, 0x1ce0, 0x0cc1,
    0xef1f, 0xff3e, 0xcf5d, 0xdf7c, 0xaf9b, 0xbfba, 0x8fd9, 0x9ff8,
    0x6e17, 0x7e36, 0x4e55, 0x5e74, 0x2e93, 0x3eb2, 0x0ed1, 0x1ef0};

// 获取CRC校验码
ushort QtXmodem::get_CRC16(QByteArray data)
{
```

```
    ushort crc = 0x0000;
    for (int i = 0; i < data.length(); i++)
    {
        char item = data[i];
        int crctbl_idx = ((crc >> 8) ^ item) & 0xff;
        crc = ((crc << 8) ^ crc_table[crctbl_idx]) & 0xffff;
    }
    return crc & 0xffff;
}
```

2.控制字符

```
char EOT = 0x04;
char ACK = 0x06;
char NAK = 0x15;
char CAN = 0x18;
char CRC = 'C';
uchar SOH = 0x01;
uchar STX = 0x02;
```

3.Xmodem协议详解

xmodem传输只有文件内数据的传输，没有起始帧结束帧这些，且只能进行单个文件的传输。xmodem的传输过程中，除了控制字符之外，因为帧长度不一致，格式一共会出现两种：

（1）数据长度为128时，n < 128，格式如下：

SOH	freamNum ①	~freamNum ②	data[1] ~ data[n] ③	pad[128-n] ~ pad[128] ④	CRCH ⑤	CRCL ⑥
01H	01H	feH	data[1] ~ data[n]	pad[128-n] ~ pad[128]	CRCH	CRCL

（2）数据长度为1024时，n < 1024，格式如下：

STX	freamNum ①	~freamNum ②	data[1] ~ data[n] ③	pad[128-n] ~ pad[128] ④	CRCH ⑤	CRCL ⑥
02H	01H	feH	data[1] ~ data[n]	pad[128-n] ~ pad[128]	CRCH	CRCL

①字段为帧序号，第一个数据帧的序列号为01h。如果单个文件传输帧个数超出ffh大小时，重新从00h开始。

②字段为帧序号的反码，即由255减去帧序号得到
③字段为数据段，从文件中读取到的数据
④字段为补充段，即数据段不足1024字节或128字节时，补充0x1A到对应长度
⑤字段为CRC校验码（ushort，16位的数）的高8位
⑥字段为CRC校验码（ushort，16位的数）的低8位

> 注意：原理上帧格式由剩余数据长度来决定，如以1024传输时，如果剩余传输长度不足1024时，应发送用1ah补全的STX包。但是根据在CRT中测试，不足1024时则是拆成128字节的SOH包传输的。

3.1.Xmodem会话传输过程

发送端 —>	<— 接收端
	'C'
数据帧1	
	ACK
数据帧2	
	ACK
数据帧3	
	ACK
… …	… …
数据帧n	
	ACK
EOT	
	NAK
EOT	
	ACK

3.2.XModem数据组装

```
1 void QtXmodem::send_data()
```

```
{
    //处理帧序号和帧序号反码
    if(freamNum == 0xff){
        freamNum = 0x00;
        freamNum_C = 0xff;
    }
    else{
        freamNum += 1;
        freamNum_C -= 1;
    }
    m_data.clear();

    //读取文件内容128字节，作为数据段
    QByteArray file_str = m_file_fd->read(128);

    //如果读取到的文件内容为空，那么这个时候数据已经传输完成，需要发送EOT，结束Xmodem传输
    if(file_str.isEmpty()){
        m_data.append(EOT);
        emit sign_send_data(m_data);

        m_file_fd->close();
        freamNum = 0x00;
        freamNum_C = 0xff;
        m_data.clear();
        emit sign_finish_Xmodem();
        if(t1->isActive())
            t1->stop();
        qDebug()<<"xModem烧录完成";
    }
    else{
        //如果数据段长度小于128字节，那么就需要补充0x1a至128字节
        while(file_str.length() < 128){
            file_str.append(0x1a);
        }

        //获取CRC校验码
        ushort CRC16 = get_CRC16(file_str);

        //组装数据
```

```
        m_data.append(SOH);
        m_data.append(freamNum);
        m_data.append(freamNum_C);
        m_data.append(file_str);
        m_data.append(CRC16 >> 8);
        m_data.append(CRC16 & 0xff);

        //发送数据
        emit sign_send_data(m_data);
    }
}
```

3.3.Xmodem与串口数据交互

```
void QtXmodem::slot_recv_data(QByteArray data)
{
    //当接受到的窗口数据位CRC或者ACK时，发送数据
    //这里start_flag是一个Xmodem传输是否开始的标记，当检测到CRC时，那么会开始传输数据，并将start_flag=true
    if(data[0] == CRC && start_flag == false){
        send_data();
        start_flag = true;
    }
    else if(data[0] == ACK){
        t1->stop();
        send_data();
    }
}
```

4.Ymodem协议详解

YModem的传输过程中，除了控制字符之外，因为帧长度不一致，格式一共会出现两种：

（1）数据长度为128时，n < 128，格式如下：

SOH	freamNum ①	~freamNum ②	data[1] ~ data[n] ③	pad[128-n] ~ pad[128] ④	CRCH ⑤	CRCL ⑥
01H	01H	feH	data[1] ~ data[n]	pad[128-n] ~ pad[128]	CRCH	CRCL

（2）数据长度为1024时，n < 1024，格式如下：

SOH	freamNum ①	~freamNum ②	data[1] ~ data[n] ③	pad[128-n] ~ pad[128] ④	CRCH ⑤	CRCL ⑥
01H	01H	feH	data[1] ~ data[n]	pad[128-n] ~ pad[128]	CRCH	CRCL

①字段为帧序号，第一个数据帧的序列号为00h。如果单个文件传输帧个数超出ffh大小时，重新从00h开始。

②字段为帧序号的反码，即由255减去帧序号得到

③字段为数据段，从文件中读取到的数据

④字段为补充段，即数据段不足1024字节或128字节时，补充0x1A到对应长度

⑤字段为CRC校验码（ushort，16位的数）的高8位

⑥字段为CRC校验码（ushort，16位的数）的低8位

> 注意：原理上帧格式由剩余数据长度来决定，如以1024传输时，如果剩余传输长度不足1024时，应发送用1ah补全的STX包。但是根据在CRT中测试，不足1024时则是拆成128字节的SOH包传输的。

4.1.起始帧

在CRT中使用Ymodem传输，起始帧大小根据端口配置的传输大小决定，如果使用1k传输则起始帧长度为1024 + 5（帧头帧尾），如果使用128传输则起始帧长度为128 + 5（帧头帧尾）。

帧头中包含多个字段，这样设计目的是允许添加额外的包头字段，而不产生与旧字段的兼容性问题，但是这样会导致不同串口软件的兼容性出现问题，本文中以CRT为准，超级终端、Xshell等其他串口软件好像只有文件名和文件大小两个字段（未求证，评论区可留言）。

可选项前面用20h与前一字段隔开。

文件名	文件大小	文件最后修改时间	文件模式	序列号	自定义
31 31 31 2E 74 78 74 00	31 30 32 34	20 31 34 32 33 35 32 33 34 30 37 30	20 30	暂无	暂无

1. 第一块是文件名16进制数转为ascii码即为文件名字， 31 31 31 2E 74 78 74转换后得111.txt。
2. 第二块是文件大小16进制数转为ascii码即为文件大小(字节），31 30 32 34转换后得1024 bytes。
3. 第三块是文件修改日期（可选项），使用这种标准格式是为了消除由于在不同时区之间传输而产生的歧义。16进制数转为ascii码后，再将此acsii码当作8进制转换成10进制便得到文件的修改时间（自 1970 年 1 月 1 日（00:00:00 GMT）以来的秒数），之后便可以转换为各个地区时间。31 34 32 33 35 32 33 34 30 37 30 转换后得 14 235 234 070(8进制)，再转换后得 1,651,849,272秒（10进制）最后可转换后得 2022-05-06 23:01:12（北京时间）。如果日期为0，那么表示修改日期未知，应该保留为文件收到的日期。
4. 第四块是文件模式（从这个开始往下都不太清楚，也没有用到），如果文件模式被发送，一个空格将文件模式和修改日期分开。文件模式以八进制字符串的形式存储。除非文件源自Unix系统，否则文件模式设置为0。接收批

处理(1)检查文件模式中是否有表示Unix类型的0x8000位常规文件。具有0x8000位集的文件被假定是从另一个Unix(或类似的)系统使用相同的文件约定。这些文件在任何方式不会被翻译。

5. 第五块是序列号，如果发送了序列号，则序列号与文件模式之间用一个空格隔开。传输程序的序列号被存储为一个八进制字符串。没有序列号的程序应该省略该字段，或者将其设置为0。接收方对该字段的使用是可选的。

4.2.结束帧

结束帧本质上结构与起始帧一致，如果传输时检查文件名位置，数据为00h既文件名为空，则说明整个会话已经结束，接收到的是结束帧。

文件名	文件大小	文件最后修改时间	文件模式	序列号	自定义
00	30	20 30	20 30	暂无	暂无

4.3.会话传输过程

发送端 —>	<— 接收端
	'C'
起始帧	
	ACK
	'C'
数据帧1	
	ACK
数据帧2	
	ACK
… …	… …
数据帧n	
	ACK
EOT	
	NAK
EOT	
	ACK

	'C'
结束帧	
	ACK

4.4.Ymodem起始帧组装

```
void QtYmodem::send_first()
{
    freamNum = 0x00;
    freamNum_C = 0xff;

    m_data.clear();

    //文件名+文件大小+文件最后修改时间（8进制）+文件模式
    QFileInfo info(m_file);
    QDateTime file_time = info.lastModified();
    int secs = file_time.secsTo(QDateTime::currentDateTime());
    QByteArray file_str = m_file.split("/").last().toLocal8Bit() + QString("%1").arg(info.size()).toLocal8Bit();
    file_str.append(0x20);
    file_str = file_str + get_time_str8((secs)).toLocal8Bit() + QString(QChar(0x20)).toLocal8Bit() + QString(QChar(0x30)).toLocal8Bit();
    while(file_str.size() < 128){
        file_str = file_str + QString(QChar(0x00)).toLocal8Bit();
    }

    ushort CRC16 = get_CRC16(file_str);
    m_data.append(SOH);
    m_data.append(freamNum);
    m_data.append(freamNum_C);
    m_data.append(file_str);
    m_data.append(CRC16 >> 8);
    m_data.append(CRC16 & 0xff);
    emit sign_send_data(m_data);
    error_count = 0;
    t1->start(100);
}
```

4.5.Ymodem数据帧组装

```
void QtYmodem::send_data()
{
    if(freamNum == 0xff){
        freamNum = 0x00;
        freamNum_C = 0xff;
    }
    else{
        freamNum += 1;
        freamNum_C -= 1;
    }

    m_data.clear();
    QByteArray file_str = m_file_fd->read(128);

    read += file_str.length();
    emit sign_ymodem_progress(read,total);

    if(file_str.length() == 0){
        m_data.append(EOT);
        emit sign_send_data(m_data);
    }

    else{
        while (file_str.size() < 128) {
            file_str.append(0x1a);
        }
        ushort CRC16 = get_CRC16(file_str);
        m_data.append(SOH);
        m_data.append(freamNum);
        m_data.append(freamNum_C);
        m_data.append(file_str);
        m_data.append(CRC16 >> 8);
        m_data.append(CRC16 & 0xff);
        emit sign_send_data(m_data);
        error_count = 0;
```

```
        t1->start(100);
    }
}
```

4.6.Ymodem结束帧组装

```
void QtYmodem::send_last()
{
    if(freamNum == 0xff){
        freamNum = 0x00;
        freamNum_C = 0xff;
    }
    else{
        freamNum += 1;
        freamNum_C -= 1;
    }

    m_data.clear();

    QByteArray file_str = QString(QChar(0x00)).toLocal8Bit() + QString(QChar(0x30)).toLocal8Bit() + QString(QChar(0x20)).toLocal8Bit() + QString(QChar(0x30)).toLocal8Bit() + QString(QChar(0x20)).toLocal8Bit() + QString(QChar(0x30)).toLocal8Bit();
    while(file_str.size() < 128){
        file_str = file_str + QString(QChar(0x00)).toLocal8Bit();
    }

    ushort CRC16 = get_CRC16(file_str);
    m_data.append(SOH);
    m_data.append(freamNum);
    m_data.append(freamNum_C);
    m_data.append(file_str);
    m_data.append(CRC16 >> 8);
    m_data.append(CRC16 & 0xff);
    emit sign_send_data(m_data);
    error_count = 0;
    t1->start(100);
}
```

4.7.Ymodem与串口数据交互

```
void QtYmodem::slot_recv_data(QByteArray data)
{
    if(data[0] == CRC && start_flag == false){
        send_first();
        start_flag = true;
        timeout = QTime(0,0,0,0);
        t2->stop();
    }
    else if(data[0] == ACK && m_data[0] != EOT){
        t1->stop();
        send_data();
    }
    else if(data[0] == ACK && m_data[0] == EOT){
        send_last();
        qDebug()<<"send last";

        m_file_fd->close();
        freamNum = 0x00;
        freamNum_C = 0xff;
        m_data.clear();

        emit sign_finish_Ymodem();
        t1->stop();
        qDebug()<<"yModem烧录完成";
    }
}
```