# 用 UART 做文件传输（采用 Zmodem 协议）

***V1.0 - Aug 4, 2005***

***中文版***

19, Innovation First Road • Science Park • Hsin-Chu • Taiwan 300 • R.O.C.

Tel: 886-3-578-6005 Fax: 886-3-578-4418 E-mail: mcu@sunplus.com.cn

http://www.sunplusmcu.com http://mcu.sunplus.com

**版权声明**

# 目录

# 修订记录

| 版本 | 日期 | 编写及修订者 | 编写及修订说明 |
|---|---|---|---|
| 1.0 | 2005/08/04 | 马超 | 初版 |

# 1 ZMODEM 协议

## 1.1 协议的历史

在 70 年代后期，调制解调器刚刚开始打入新兴的台式机市场。由于缺乏为文件传输建立的标准，Ward Christensen 开发了后来成为 XMODEM 的文件传输协议。对 XMODEM 的增强很快就到来了，导致了整个协议族的产生，包括 YMODEM 和 XMODEM－1K。

XMODEM 并不是一个精心设计的协议，它来源于广泛的研究和试验。它更接近于一个“周末工程”，即个人的使用软件。对于这个最初的目的，随着时间的流逝，它还是支持的。XMODEM 确实很有能力，但他有几个严重的局限：

- 协议控制字符没有包装到数据包中，是的协议易于受到噪声和单字符出错的影响。
- XMODEM 要求完全清除的 8 位频道。频道需要通过所有控制字符，包括 XON、XOFF、CR 等等。许多老的分时系统不能满足这种要求。他们的设备驱动程序引起某些协议丢失。
- BL：XMODEM 家族所使用的短包长度，不是 128 就是 1024 字节，因为在发送下一个数据包之前每个数据包必须被认可，这就导致频道的低效使用。

在 1986 年，包交换网络提供商 Telenet 委托 Chuck Forsberg of Omen Technology 开发新的文件传输程序，它可以在网络上高效的使用。结果就产生了 ZMODEM。它由一个公共域程序和一个新协议组成。名称 ZMODEM 可能隐含着它是按 XMODEM 和 YMODEM 排列下来的后代，但并不真是这样的。ZMODEM 完全是新的协议，与这些早期系统中任何一个的共同点都非常少。

## 1.2 ZMODEM 概述

ZMODEM 不同于 XMODEM 之处是发送器和接收器之间传递的所有信息都包含在数据包中，在 ZMODEM 中称为帧。即使是简单的协议协议信息（比如确认信号）也在数据包中，这提供给 ZMODEM 很好的保护，以避免偶然的协议信息。

ZMODEM 帧有两个组件。每个帧以报头开始，它标识帧的类型，并带有至多 4 个字节的信息。这 4 个字节称为 ZF0 到 ZF3(按位置分别是 3 到 0)。数据子包的流是原始数据块，可以有选择地跟随一个报头。

每个数据子包可以包含至多 1024 个字节的数据，其后跟随一个 CRC 值用于校验。对可连接到一个报头之后的数据子包的个数没有限制，这就是说，一个文件的所有数据可在一个帧内传送。

## 1.3 ZMODEM 帧类型

ZMODEM 支持多种帧类型。要实现简单的 ZMODEM 并不需要所有这些帧类型；有些用于更复杂的出错恢复和传输机制。这里提供一个完整的列表：

- ZRQINIT＝0：当 ZMODEM 发送器启动时，它发送这种帧。它是请求接收器发送它的 ZRINIT 帧，该帧将开始文件传输。ZRQINIT 帧头可用于在接收程序中触发一个自动下载。
  ZRQINIT 帧不发送任何数据子包。如果发送器试图发送一个命令给接收器，头字节 ZF0 包含常量 ZCOMMAND（这个特性应用例不支持）；否则它包含一个 0。
- ZRINIT＝0: 接收器发送这种帧以指明它准备好从发送器处接收文件。可以自发地发送它，或者应答 ZRQINIT 帧。这个帧具有 4 个字节地能力信息包装在报头中。ZF0 和 ZF1 的下列位可以进行设置和清除，这要取决于接收器的能力。
  - CANFDX＝1: 接收器具有确实的全双工操作，意味着它可以同时发送和接收数据。
  - CANOVIO＝2:接收器在写磁盘时可以接收数据，要充分利用 ZMODEM 的流特性，就需要这种能力。
  - CANBRK＝4: 接收器可以发送一个中断信号。
  - CANCRY＝8: 接收器可以解码 RLE 帧。应用例中不支持这种能力（Omem Technology 可能把这种能力看做对 ZMODEM 的专有扩展）。
  - CANLZW＝16: 接收器可以解压缩 UNIX 压缩式的数据。这种能力在应用例中不支持。
  - CANFC32＝64: 接收器可以接受 CRC32，应用例不支持 CRC32。
- ESCCTL=64: 接收器需要看到所有控制字符转换码，而不是仅 XON/XOFF 等少数几个。
  ZF2 和 ZF3 包含接收器输入缓冲的尺寸。如果这个值是非零的就意味着接收器不能在完全的流模式中工作。相反，当它向磁盘写数据时，它将停止接收。
- ZSINIT＝2： 在处理 ZRINIT 帧后，这个帧可以被发送器有选择地发送给接收器。它提供给接收器一些信息而不管接收器的能力。两个位装入 ZF0。
- TESCCTL＝64: 此位用于指明发送器期望所有控制字符都进行转换。
- TESC8＝128: 此位用于指明发送器期望转换 8 位。ZMODEM 没有完全实现这种能力，但包含在未来改进的规范中。
  仅有一个数据子包跟随在报头后面。这个子包包含不超过 32 个字符的字符串，以空字符终止。这个字符串用于在出错时唤醒发送器。
- ZACK＝3: 这种帧类型用于确认 ZSINT 和 ZCHALLENGE 帧，以及其后跟随 ZCRCQ 或 ZCRCW 终止符的数据子包。如果应答是给 ZCHALLENGE 帧的，4 个报头标志字节由 ZCHALLENGE 报头所发送的 4 个字节副本来填充。
- ZFILE＝4: 这种帧类型用于初始化实际的文件传输。它包含一个报头，后面跟随单个数据子包，子包包含文件信息。4 个报头字节中填入了与要传输文件有关的各种标志。ZF0 包含文件转换选项；ZF1 包含可选的管理选项；ZF2 具有传输选项；ZF3 包含扩充选项。
  这个帧考虑的选项最多，而且很少使用。对于我们的目的，在 ZFILE 帧中传送的重要项是文件名称、长度及可选的日期。在此详细说明选项的其余部分是为了完整性。

- ZSKIP＝5: 这是一个简单的帧，当接收器选择不接收发送器的 ZFILE 帧制定的文件时，就发送这个帧。它在 4 个报头字节中没有存储数据，且不发送任何数据子包。
- ZNAK＝6: 用于指明最后的报头由于许多原因是非法的。
- ZABORT＝7: 接收器发送这个帧以指明会话将终止。通常发送这个帧响应用户发出的取消操作。
- ZFIN＝8: 当发送器再没有文件传送时发送这个帧。接收器在退出前以它自己的 ZFIN 帧回答。ZMODEM 规范的特别之处是，在接收由接收器发送的 ZFIN 之后，是发送器发两个字符（00）。
- ZRPOS＝9: 接收程序在任何时候都可以发送这个帧。4 个字节的偏移填充到报头信息的 4 字节中。偏移量是接收器从文件的某个位置开始发送数据的请求。若要进行恢复，或在传输期间在到来的数据流中检测到错误，可在文件传输开始时发送这个帧。
- ZDATA＝10: 这个帧的 4 个报头字节包含跟随数据文件偏移量。可跟随任意个数据子包。
- ZEOF＝11: 这指明所有文件数据都已经发送了。4 个报头字节包含 EOF 的偏移量。这个选项增加了协议的可靠性，以免目标文件上过早出现 EOF 的可能性。
- ZFERR＝12: 如果在访问文件时发生错误，发送器或接收器都可以发送这个帧。接收这个帧字节使会话终止。
- ZCRC＝13: 接收器发送这个帧，对用 ZFILE 帧指明的文件请求 32 位 CRC。发送器发送这个帧，使 CRC32 包装到 4 个报头字节中。
- ZCHALLENGE＝14:发送器使用这个帧测试质询接收器。把随即数包装到报头的 4 字节数据中。然后接收器负责回送相同的 4 个数字到另一个 ZCHALLENGE 帧。这个帧类型用于帮助防止未经接收器统一就把文件恶意下载到系统中。
- ZCOMPL＝15:当 ZCOMMAND 请求完成时接收器发送这个帧。4 个报头字节包含命令返回的状态码。
- ZCAN＝16: 至少 5 个 CAN，请求临时的接收传送。
- ZFREECNT＝17: 发送器可以向接收器请求默认卷上可用的空间的计数。接收器用 ZACK 帧响应，它以字节为单位，并使空闲空间数量包装到 4 个报头直接中。
- ZCOMMAND＝18: 这个帧为发送器提供把命令发送到接收系统的能力。如果 ZF0 设置为 0，接收器执行命令并在完成时返回 ZCOMPL 帧。如果 ZF0 设置为 ZCACK1，接收器立即返回 ZCOMPL 包。

### 1.3.1 ZF0 转换选项

ZF0 可以设置为下列值之一，它们指明存储文件数据时使用的转换方法。

- ZCBIN＝1: 二进制传输，数据不需要转换。
- ZCNL＝2: 使用本地约定转换接收行的结尾。这个选项在 UNIX 和 MS－DOS 之间发送 ASCII 文件是有用的。
- ZCRESUM＝3: 从中断的文件传输中恢复。接收器检查目标文件是否比发送的文件短。若是，文件传输可在异常终止的地方重新开始。

### 1.3.2 ZF1 管理选项

ZF1 可以设置为以下的数值。

- ZMNEW＝1: 只有在源文件更新或更长时覆盖目标文件。
- ZMCRC＝2: 对比源文件和目标文件的 CRC。如果数值相同，传输文件，否则跳过传输。
- ZMAPND＝3: 把源文件追加到目标文件。
- ZMCLOB＝4: 如果目标文件存在，则无条件的覆盖目标文件。
- ZMSPARS＝5:如果目标文件存在，只有文件比较新才覆盖目标文件。
- ZMDIFF＝6: 如果目标文件存在，只有文件的长度或日期不同才覆盖目标文件。
- ZMPROT＝7: 这个选项与 ZMCLOB 相对。它告诉接收器只有在目标文件不存在时才传输文件。

### 1.3.3 ZF2 传输选项

ZF2 可以设置为以下的数值。

- ZTLZW＝1: 要传输的数据经过 UNIX 压缩程序压缩的。在应用例中不支持这个选项。
- ZTRLE＝3: 数据是使用 Run Length Encoding 压缩的。通过创建专用的 RLE 帧，在 ZMODEM 后来的版本中，这部分规范被取代。

丢失的选项 2 属于 ZTCRYPT，是在最初的 ZMODEM 规范中定义，但从未实现的加密选项。

### 1.3.4 ZF3 扩展选项

ZF3 是位图，其中由条件的设置了扩充选项。在 ZMODEM 规范中定义的唯一选项是 ZTSPARS 选项，它考虑到对“稀疏文件”的特殊处理。这个选项在 ZMODEM 规范中定义，但它很少得到过支持。

### 1.3.5 数据子包

跟随 ZFILE 报头的数据子包包含将要传输的文件相关信息。下列字段在数据帧中是连续的：

Filename： 文件名，是以空值终止的字符串。

Length： 文件长度，以十六进制给出的 ASCII 表示。

Data： UNIX 格式的日期和时间。这表示自 1970 年 1 月 1 日开始、以秒为单位的当前的计数值。数值 0 用于指明日期是未知的。值是以十六进制给出的 ASCII 表示。

Mode： 以 UNIX 格式表示的文件模式位，同样是以 ASCII 表示的实力欧进制值。对于从 MS－DOS 系统发送的文件，其模式设置为 0.

S/N： 传输程序的序列号。

File left: 要发送的剩余文件数。

Byte left： 要发送的剩余字节总数。

文件名后面的各个字段是可选的。多数实现支持文件长度和日期，但一般不会之处更多的。这个字段使用分隔符好是特殊的位。文件名是以'\0'字符结束得，其余的字段是由空格分隔的。

## 1.4 头格式

ZMODEM 帧是由报头以及可能跟随在数据子包流组成的。报头本身相对简单，它由 4 个基本成分：

(1) 报头类型字节，指明要使用 3 种可能报头类型中的那种类型。
(2) 帧类型字节，指明要编码 18 种可能帧类型中的那种类型。
(3) 4 个数据字节。
(4) 报头的 CRC

对于 ZMODEM 实现，报头由三种不同的变化。ZMODEM 报头可以编码为：

(1) 十六进制报头，仅使用可打印的字符对它编码。如果数据子包跟随这个报头，它们以二进制而不是十六进制。
(2) CRC16 二进制报头。为了更高效地使用带宽，跟随这个报头地数据子包也直接以二进制发送。
(3) CRC32 二进制报头。数据子包以二进制发送，使用 32 位 CRC 是为了可靠性。

### 1.4.1 十六进制报头

十六进制报头使用可以打印地字符传送，唯一的例外是 ZDLE 字符（^X 或 ASCII 的 24）。格式是：

ZPAD ZPAD ZDLE B FrameType ZF3 ZF2 ZF1 ZF0 CRC1 CRC2 CR LF XON

前面的 ZPAD 字符定义为'*'。为了检测帧的开始，接收器使用 ZPAD ZPAD ZDLE 序列。字符 B 用于表示十六进制报头。在 B 字符之后，在结尾的 CR/LF/XON 序列之前，所有字符都是十六进制编码的，这意味着每一个字节使用两个字符的可打印值，而不是纯二进制。

在接收器响应发送器时，它完全使用十六进制报头。对于没有任何数据子包跟随着报头的任何帧，发送器也使用十六进制报头。当数据子包包含在帧中时，一般使用这两种形式的二进制报头。然而，即使在这种情形，若要求的话，发送器可以使用十六进制报头，这是以某种效率稍微降低作为代价的。

当远程发送器开始 ZMODEM 传输时，它首先给接收器一个 ZRINIT 帧，要求一个 ZTINIT 帧。

### 1.4.2 CRC16 和 CRC32 二进制报头

十六进制报头不是发送数据的最高效的方法。为了更高效的包装信息，ZMODEM 发送器使

用二进制报头。在二进制报头中，信息被包装为单个二进制字符而不是十六进制字符。

两个报头的格式显示如下：

16 位： ZPAD ZDLE A FrameType ZF3 ZF2 ZF1 ZF0 CRC1 CRC2

32 位： ZPAD ZDLE C FrameType ZF3 ZF2 ZF1 ZF0 CRC1 CRC2 CRC3 CRC4

32 位 CRC 以某种效率的代价提供稍微高一些的可靠性。当数据子包跟随着报头时，子包的格式由报头的格式决定。跟随 16 位二进制报头（或者，还可能是十六进制报头）的数据子包将使用 CRC16 检查。跟随 32 位二进制的数据子包将使用 CRC32 检查。只有接收器在 ZRINIT 中发送 CANFC32 位，才可以使用 CRC32 报头。CRC16 或 CRC32 的值应用到帧类型和位置标志中。

## 1.5 数据子包的格式

ZMODEM 取得它的最高效率在某些方面是依靠它发送数据子包的方法。紧跟着报头，ZMODEM 发送器想要发送多少数据子包可以发送多少个。数据子包由至多 1024 个二进制字符组成，而某些字符经过编码，后面跟随一个 ZDLE 和一个子包接收字符。

有 4 种不同的子包结束字符，每个都有略微不同的目的。它们都指明 CRC16 或 CRC32 值将紧随其后。4 种不同字符还有其它意义：

- ZCRCE（'h'）：这个字符标志数据子包的结尾帧的结尾。当接收器在子包的结尾看到这个字符时，它就知道它需要开始查找下一个报头了。这个子包一般用在文件的结尾。
- ZCRCG（' i'）：这个字符终止当前子包，但指明至少还有一个子包跟随其后。接收器需要开始检查其后的 CRC 值，然后开始接收一个新的子包。
- ZCRCQ（'j'）：这个字符终止当前子包，但不终止帧。与 ZCRCE 不同，接收器必须多做一些事情，而不仅是校验子包校验值。此外，接收器发送一个 ZACK 帧指明子包接收成功。发送器周期性的发送这个字符以迫使接收器响应。这有助于检测主动连接。
- ZCRCW（'k'）：这个字符以终止当前子包和帧。在检验 CRC 后，接收器可预期一个新的报头开始下一个帧。这个终止字符还指明接收器需要发送一个 ZACK 以校验子包的接收。

在正常的文件传输期间，ZMODEM 发送器仅能以一系列子包文件的方式发送整个文件，除了最后一个子包外，使用 ZCRCG 终止每个子包。最后的子包可以由 ZCRCE 来终止，它使接收器确认整个帧的正确接收。在接收到 ZACK 后，发送器保证所有数据都正确的接收。

# 2 文件传输

所有这些帧均放到一起，以相对简单的方式完成文件传输。尽管 ZMODEM 在最低层是复杂的，但从高层的视点看，它具有吸引人的简单性。为了发送一个文件，只要求 9 个帧。每附加一个文件，仅对总帧数再增加 4 个帧。下表显示了进行可能最小的文件传输所必须的步骤。这个表仅仅考虑对于直接连接两个设备如何使 ZMODEM 运行，而且不会有高的出错率，也不会需要大缓冲区。

最简单的 ZMODEM 文件传输显示如下：

| 发送器 | 接收器 | 帧类型 |
|---|---|---|
| 发送器请求接收器发送它的初始化参数 | | ZRQINIT |
| | 接收器以提供它的大多数能力的帧进行响应 | ZRINIT |
| 发送器提供要传输文件的名称，以及关于长度和日期等可选数据 | | ZFILE |
| | 接收器请求文件数据再某个位置开始。对于新文件这一般是位置 0 | ZRPOS |
| 发送包含所有文件内容的单个数据帧。所有数据子包以 ZCRCG 结束，最后一个例外，它以 ZCRCE 结束 | | ZDATA |
| 发送所有数据，并指明了文件尾 | | ZEOF |
| | 接收器指明已经准备好开始接收新文件 | ZRINIT |
| 发送器指明它要终止会话。如果发送器有更多文件要发送，它将再发送一个 ZFILE 帧 | | ZFIN |
| | 接收器同意而连接中断 | ZFIN |

以 ZCRCQ 结束子包引起接收器的 ZACK 响应。发送器可以使用这个响应来调整它自己的速度，以保证它不会超出接收器太远，方法在它发送另一个 ZCRCQ 接收的子包之前等待一个 ZACK。

在读数据子包时若 CRC 发生错误，就发送附加帧。当发生错误时，接收器发送一个 ZRPOS 帧，带有一个偏移量，来说明它想要发送器重新开始的地方。在发送 ZRPOS 帧后，接收器必须等到缓冲区中积累的数据被刷新。最终，新的 ZDATA 帧开始，而它的位置应与最近的请求相匹配。

# 3 软件说明

## 3.1 软件说明

应用例程序部分主要实现 ZMODEM 的协议部分，完成文件的 Upload。底层协议的完成使用者可以不必了解，对数据存储，程序留出专门的函数，使用者仅仅需要编写使用者的存储数据的程序部分来存储 Upload 的数据。

## 3.2 档案构成

| 文件名称 | 功能 | 类型 |
|---|---|---|
| main.C | 初始化及文件传输测试 | C |
| Hardware.C | UART 及定时相关 MCU 的操作 | C |
| Zmodem.C | Zmodem 协议实现 | C |
| File.C | 为使用者预留出存储数据的接口 | C |
| AN_SPMC75_0115.H | API 引用和相关参数定义及结构数据定义 | H |

## 3.3 子程序说明

***Spmc75_Zmodem_Init()***

**原　　形**　void Spmc75_Zmodem_Init(void)
**描　　述**　对 SPMC75F2413A 的 UART 及 CMT0 初始化。
**输入参数**　无
**输出参数**　无
**头 文 件**　AN_SPMC75_0115.H
**库 文 件**　无
**注意事项**　默认使用 UART2 和 CMT0
**例　　子 1**　Spmc75_Zmodem_Init();　//初始化

***ZmodemReceiveFile()***

**原　　形**　int ZmodemReceiveFile(void)
**描　　述**　使用 ZMODEM 协议接收文件。
**输入参数**　无
**输出参数**　状态信息
　　0=OK ； －1=ERROR ； -4＝NOWAKE
**头 文 件**　AN_SPMC75_0115.H
**库 文 件**　无
**注意事项**　完成文件的接收
**例　　子**　err = ZmodemReceiveFile();　　//接收文件

***File_SaveToMemory()***

| | |
|---|---|
| **原　　形** | UInt16 File_SaveToMemory(char *ptrdat, UInt16 length)； |
| **描　　述** | 存储接收来的数据到存储器中。 |
| **输入参数** | Ptrdat：存储接收来的缓存数据块的首址<br>Length：数据快的长度 |
| **输出参数** | 存储数据是否成功<br>OK＝0; ERROR＝－1 |
| **头 文 件** | AN_SPMC75_0115.H |
| **库 文 件** | 无 |
| **注意事项** | 这个函数时使用者来决定数据储存的函数，对于不同的使用可能有不同的存储介质，所以需要使用着来填充着部分程序，完成数据的存储。<br>注意：存储没有问题则必须返回 OK＝0，否则返回 ERROR＝－1。<br>这个函数在协议实现中调用。 |
| **例　　子** | 无 |

***OpenInputFile()***

| | |
|---|---|
| **原　　形** | UInt16 OpenInputFile(char *ptrstr); |
| **描　　述** | 函数在协议实现的过程中接受到 ZFILE 帧后调用，这时可以得到发送器发送文件的文件名,长度,修改日期...。使用则可以自行编写程序了解信息，作出处理。 |
| **输入参数** | Ptrstr：ZFILE 数据帧的内容。 |
| **输出参数** | 无 |
| **头 文 件** | AN_SPMC75_0115.H |
| **库 文 件** | 无 |
| **注意事项** | 协议实现的过程中需要调用的一个内部函数，使用者可以修改来了解传输文件的一些重要信息。不做任何处理将忽略 ZFILE 数据帧所提供的信息。 |
| **例　　子** | 无 |

# 4 程序范例

## 4.1 范例程序

示范性程序很简单，对于协议的实现可以不必关心。

```
//=====================================================================
#include    "AN_SPMC75_0115.H"
//=====================================================================
main()
{
   int err=0;

   Spmc75_Zmodem_Init();               //初始化
   err = ZmodemReceiveFile();          //接收文件
   NOP();
   NOP();
   NOP();
   while(1);
}
```

## 4.2 数据存储

协议的底层可以不必关心，但是数据储存是使用者必须关心的。

```
//=============================================================
// ----Function: File_SaveToMemory()
// ------Syntax: UInt16 File_SaveToMemory(char *ptrdat, UInt16 length)
// -Description: 存储接收到的数据
// -------Notes: 使用者自行编辑,用于存储接收来的数据
// --parameters: ptrdat:数据块的首指针, length:数据块的大小
// -----returns: 存储数据 OK/ERROR,错误请返回 ERROR.
//=============================================================
UInt16 File_SaveToMemory(char *ptrdat, UInt16 length)
{
   NOP();
   NOP();
   NOP();
   return(OK);
}
```

对于发送器提供的文件信息的处理可以忽略，也可以用以决定存储是否够用，对升级日期的维护和文件名的相关管理等。

```
//=============================================================
// ----Function: OpenInputFile()
// ------Syntax: UInt16 OpenInputFile(char *ptrstr)
// -Description: 接收文件的 ZFILE 包的解释函数,自行处理
// -------Notes: 数据包含接收文件的文件名,长度,修改日期...
// --parameters: ZFILE 包数据块的指针
// -----returns: 能否接收文件,OK/ERROR,错误请返回 ERROR.
//=============================================================
UInt16 OpenInputFile(char *ptrstr)
{
/*-------------------------------------
/*Note.
/* 可以添加对 updown 文件的 ZFILE 报头的数据
/* 子包信息的解释程序.
/*-------------------------------------*/
   return(OK);
}
```

# 5 建立连接

下面将介绍在使用过程中和相关 ZMODEM 发送终端的连接。

1、 使用 PC 的超级终端的 ZMODEM 协议 Upload 数据，SPMC75F2413A 来 Download 数据。

如图 5-1 所示，首先在 PC 找到超级终端【开始】->【程序】->【附件】->【通讯】->【超级终端】。

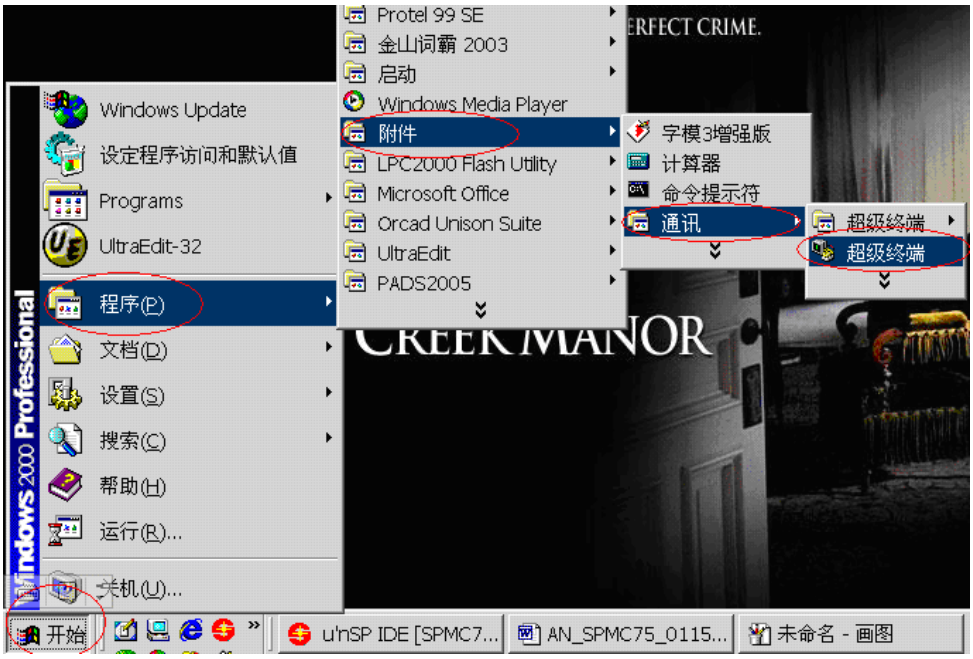


图 5-1

2、 在打开超级终端后的对话框的提示下建立连接，如果使用 COM1 来通讯，则接下来对 COM1 的属性进行设置。在 SPMC75F2413A 接收程序中默认 UART 通讯的波特率为 19200BPS，其它的设置如图 5-2 所示。


图 5-2

3、 在【传送】中打开【发送文件】，在对话框中设置传输协议为 Zmodem，在【浏览…】中找到需要传输的文件路径。如图 5-3 所示。

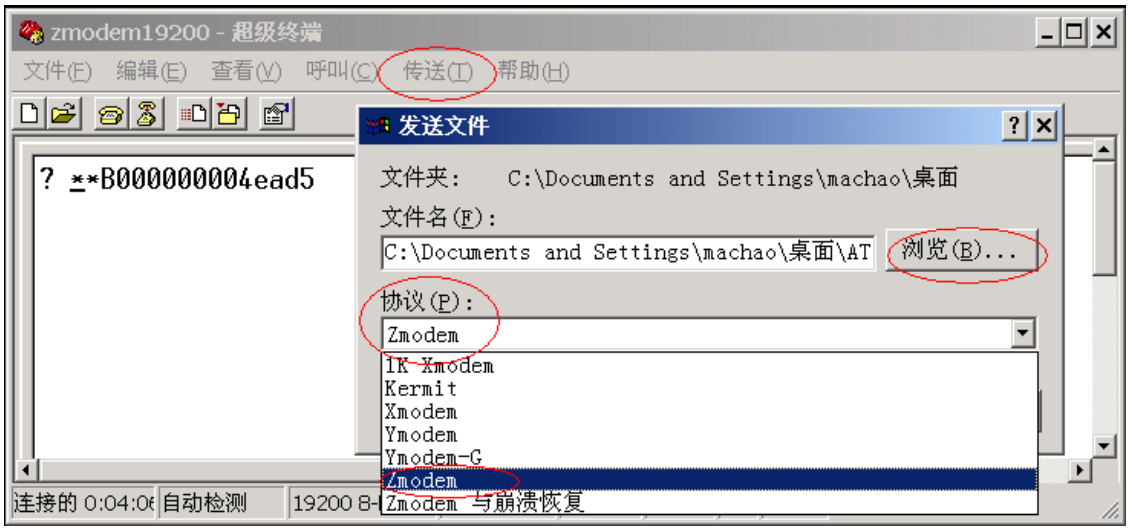


图 5-3

4、 确认发送，开始文件的传输，在传输的过程中显示传输的状态。如图 5-4 所示。

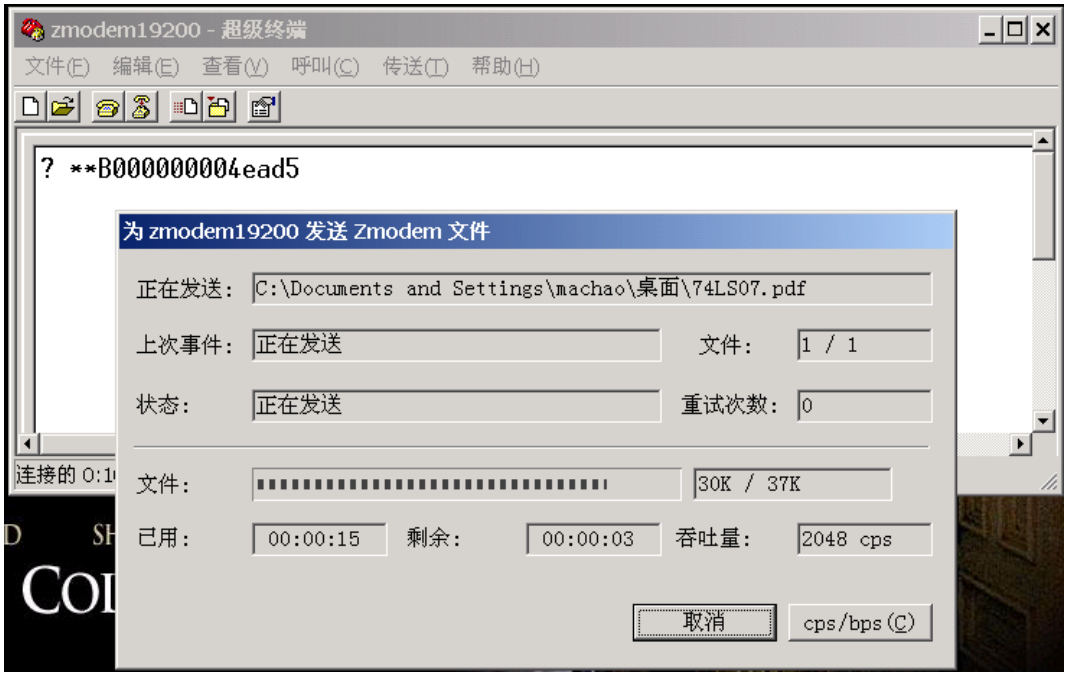


图 5-4

# 6 MCU 使用资源

## 6.1 MCU 硬件使用资源说明

| | | | |
|---|---|---|---|
| CPU 型号 | SPMC75F2413A | 封装 | QFP80-1.0 |
| 振荡器 | ☒ crystal | 频率 | 6MHz |
| | ☐外部 | 输入频率 | |
| WATCHDOG | ☒ 有 ☐ 无 | ☐ 启用 ☒ 未启用 | |
| IO 口使用情况 | IOC[0-1] | 串行通讯接口 | |
| | 剩余 IO 及处理方式 | 主控应用/GND | |
| Timer 使用情况 | CMT0 | 系统定时 | |
| 中断使用情况 | CMT0(IRQ7) | 系统服务 | |
| ROM 使用情况 | 4.45KWord | | |

# 7 参考文献

【1】SUNPLUS SPMC75F2413A 编程指南

【2】中国水利水电出版社 串口通信开发指南(第二版) [美]Mark Nelson 着

【3】Omen Technology Inc ZMODEM PROTOCOL REFERENCE File Transfer Protocol