PC 机 UART（NS8250）详解

异步串行通信原理

两台计算机/设备进行数据交换，即通信，必须像人们对话一样使用同一种语言。在计算机通信术语中，我们把计算机/设备与计算机/设备之间的“语言”称为通信协议。通信协议规定了传送一个有效数据长度单位的格式。通常我们使用术语“帧”来形容这种格式。为了能让通信双方确定收/发的顺序和进行一些错误检测操作，除了必要的数据以外，在传输的一帧信息中还包含用于同步和错误检测的其它信息，例如在开始传输数据信息之前先发送起始/同步或通信控制信息，并且在发送完需要的数据信息之后再传输一些校验信息等。

串行通信是指在线路上以比特流一次一个比特进行传输的通信方式。串行通信可分为异步和同步串行通信两种类型。它们之间的主要区别在于传输时同步的通信单位或帧的长度不同。异步串行通信以一个字符作为一帧进行传输，而同步串行通信则以多个字符或字节组成的序列作为一帧数据进行传输。若再以人们之间的对话作比喻，那么异步通信如同对话双方讲话速度很慢，说话时一个字一个字地 “蹦”出来，在说出每个字后可以停顿任意长时间。而同步通信则如同通信双方以连贯的一句话作为对话单位。可以看出，实际上如果我们把传输单位缩小到一个比特位时（对话时用一个个拼音字母），那么以一个字符进行传输的异步串行通信也可以看作是一种同步传输通信方式。因此异步和同步通信的区分主要是一种习惯或惯例上的划分。

异步串行传输格式

异步串行通信传输的帧格式如下图所示。传输一个字符由起始位、数据位、奇偶校验位和停止位构成。其中起始位起同步作用，值恒为0。数据位是传输的实际数据，即一个字符的代码。其长度可以是5到8个比特。奇偶校验位可有可无，由软件编程设定。停止位恒为1，可由软件设定为1、1.5 或 2 个比特位。在通信开始发送信息之前，双方必须设置成相同的格式。在异步通信规范中，把传送 1 称为传号（MARK），传送 0 称为空号（SPACE）。



当无数据传输时，发送方处于传号（MARK） 状态，持续发送 1。若需要发送数据，则发送方需要先发送一个持续时间为1比特位的空号，作为起始位。接收方收到空号后，就开始与发送方同步，然后接收随后的数据。若程序中设置了奇偶校验位，那么在数据传输完之后还需要接收奇偶校验位。最后是停止位。在一帧发送完后可以立刻发送下一帧，也可以先发送传号，过一段时间再发下一帧。

在接收一字符帧时，接收方可能会检测到三种错误之一：

奇偶校验错误。此时程序应该要求对方重新发送该字符；

过速错误。当程序取字符速度小于接收速度，就会发生这种错误。此时应该修改程序，加大取字符的频率；

帧格式错误。当接收到的格式信息不正确时会发生这种错误。例如在应该收到停止位时却收到了空号。通常造成这种错误的情况除了线路干扰以外，很可能是通信双方设置的帧格式不同。

通用异步接收/发送控制器（UART）

为实现串行通信，IBM PC/XT 机（下文简称 PC 机）上通常都带有 2 个符合 RS-232C 标准的串行接口，并使用通用异步接收/发送器控制芯片 UART(Universal Asynchronous Receiver-Transmitter) 组成的串行控制器来处理串行数据的收发工作。PC 机上的串行接口通常使用 25 芯的 DB-25 或 9 芯的 DB-9 连接器，主要用来连接 MODEM 设备，因此 RS-232C 标准规定了很多 MODEM 专用接口引线。

以前的 PC 机都使用国家半导体公司的 NS8250 或 NS16450 UART 芯片，现在的PC机则使用了 16650A 及其兼容芯片，但都与 NS8250/NS16450 芯片兼容。 NS8250/NS16450 与 16650A 芯片的主要区别在于16650A 芯片还支持 FIFO 传输方式。在这种方式下，UART 可以在接收或发送了最多16个字符后才引发一次中断，从而可以减轻系统和 CPU 的负担。

当 PC 机上电启动时，系统 RESET 信号通过 NS8250 的 MR 引脚使得 UART 内部寄存器和控制逻辑复位。此后若要使用 UART 就需要对其进行初始化编程操作，以设置 UART 的波特率、数据位数以及工作方式等。

UART 硬件逻辑

PC 机中使用 UART 的异步串行口硬件逻辑如下图所示。其中可分成 3 部分。



第一部分主要包括数据总线缓冲 D7~D0 、 内部寄存器选择引脚A0~A2 、 CPU 读/写数据选通引脚 DISTR 和 DOSTR、芯片复位引脚 MR、中断请求输出引脚 INTRPT 以及用户自定义的用于禁止/允许中断的引脚 OUT2。 当 OUT2 为 1 时可禁止 UART 发出中断请求信号。

第二部分主要包括 UART与 RS-232 接口的引脚部分。这些引脚主要用于接收/发送串行数据和产生或接收 MODEM 控制信号。串行输出数据引脚(SOUT)向线路上发送比特数据流；输入数据引脚(SIN)接收线路上传来的比特数据流；数据设备就绪引脚(DSR)用于通信设备(MODEM)通知 UART 已准备好接收数据；数据终端就绪引脚(DTR)则用于计算机通知 MODEM 已准备好接收数据…由于这部分和编程关系不大，所以略去。

第三部分是 UART 芯片时钟输入电路部分。UART 的工作时钟可以通过在引脚 XTAL1 和 XTAL2 之间连接一个晶体振荡器来产生，也可以通过 XTAL1 直接从外部引入。PC 机则使用了后一种办法，在 XTAL1 引脚上直接输入 1.8432MHz 的时钟信号。UART 发送波特率的 16 倍由引脚 BAUDOUT 输出，而引脚 RCLK 是接收数据的波特率。由于这两者连接在一起，因此 PC 机上发送和接收数据波特率相同。

NS8250 的寄存器

与中断控制芯片 8259A 一样，UART 也是一个可编程的控制芯片。UART 的内部组成框图如下。



NS8250 中 CPU 能够访问的寄存器有 9 个，但是用于选择这些寄存器的地址线 A2~A0 最多能够选择 8 个寄存器。因此 NS8250 就在线路控制寄存器（LCR）中拿出一位（位 7）用作选择除数锁存（divisor latch）寄存器（b7=1时选中） 。位 7 被称作除数锁存访问位（DLAB，Divisor Latch Access Bit）。

因为端口 0x3F8 和 0x3F9是复用的，所以一共有 7 个端口地址。端口地址及寄存器的用途见下表。

串口号 地址范围 中断号

COM1 0x3F8 - 0x3FF IRQ4

COM2 0x2F8 - 0x2FF IRQ3

NS8250 设计有2个中断寄存器（IER-中断允许寄存器， IIR-中断识别寄存器）和4级中断。4级中断的优先权，是按照串行通信过程中事件的紧迫程度安排的，是固定不变的。中断允许寄存器（IER）的低4位控制这4级中断是否被允许：某位为1，则对应的中断被允许；否则，被禁止。



（下表的端口一列，括号里的是COM2的端口地址）

端口 读/写 条件 用途

0x3F8 (0x2F8) 写 DLAB=0 写发送保持寄存器 THR，THR 会把写入的字符发送出去。

读 DLAB=0 读接收缓存寄存器 RBR , RBR 内含有收到的字符。

读/写 DLAB=1 读/写波特率因子的低字节 （LSB )

0x3F9 (0x2F9) 读/写 DLAB=1 读/写波特率因子的高字节 （MSB )

读/写 DLAB=0 IER（Interrupt enable register），中断允许寄存器

[7:4] 全 0 保留不用；

[3] = 1 允许modem状态中断；

[2] = 1 允许接收器线路状态中断；

[1] = 1 允许发送保持寄存器空中断；

[0] = 1 允许接收到数据中断;

0x3FA (0x2FA) 读 IIR（Interrupt identification register），中断识别寄存器。 中断处理程序以此判断此次中断是 4 种中的那一种。

[7:3]全 0 保留不用；

[2:1]

= 11 线路状态改变中断，优先级最高，读线路状态可复位；

= 10 己接收到数据中断， 读接收数据可复位；

= 01 发送保持寄存器空中断，读 IIR 或写发送保持寄存器可复位；

= 00 MODEM 状态改变中断，读 MODEM状态可复位 。

[0] =0,有待处理的中断；=1，无中断

0x3FB (0x2FB) 写 LCR，线路控制寄存器

具体内容见下文。

0x3FC (0x3FC) 写 MCR，MODEM 控制寄存器

[7:5] 全 0 保留 ；

[4] = 1,芯片处于循环反馈诊断操作模式；在这种方式下UART芯片内部自动把输入(SIN) 和输出 (SOUT) 引脚 “短接”，若此时发送的数据和接收到的数据相等，那么就说明 UART 芯片工作正常。注意：但该方式仅用于诊断测试UART芯片的好坏，不能作为一种实际的通信方式使用。

[3] = 1, 辅助用户指定输出2（Auxiliary output 2，OUT2）, 允 许 INTRPT 到系统；

[2] = 1, 辅助用户指定输出1（Auxiliary output 1，OUT1）, PC 机未用；

[1] = 1, 使请求发送 RTS( Request to send) 有效，可以理解为使能发送；

[0] = 1, 使数据终端就绪DTR( Data terminal ready)有效，可以理解为使能接收；

0x3FD (0x2FD) 读 LSR，线路状态寄存器

[7] = 0, 保留；

[6] = 1, 发送移位寄存器为空；

[5] = 1, 发送保持寄存器为空，可以取字符发送；

[4] = 1, 接收到满足间断条件的位序列；

[3] = 1, 帧格式错误；

[2] = 1, 奇偶校验错误；

[1] = 1, 超越覆盖错误；

[0] = 1, 接收器数据准备好，系统可读取。

0x3FE (0x2FE) 读 MSR，MODEM状态寄存器。δ

表示信号或条件发生变化。

[7] = 1, 载波检测(CD)有效；

[6] = 1, 响铃指示(RI)有效；

[5] = 1, 数据设备就绪(DSR)有效；

[4] = 1, 清除发送 (CTS) 有效；

[3] = 1, 检测到 δ

载波 ；

[2] = 1, 检测到响铃信号边沿；

[1] = 1, δ

数据设备就绪(DSR);

[0] = 1, δ

清除发送(CTS)。

LCR：线路控制寄存器

bit 含义

[1:0] 数据位长度。

00：5位数据位；

01 : 6位数据位；

10 : 7位数据位；

11 : 8位数据位

[2] 停止位长度。

=1：此时停止位长度依赖于数据位长度：若数据位长度是5位 ，则停止位为 1.5 位 ；若数据位长度是 6、7 或 8 位 ，则停止位为 2 位 ；

=0：停止位是1位

[3] =1：允许奇偶校验

=0：无奇偶校验

[4] =1：偶校验；

=0：奇校验

[5] 奇偶校验选择附加标志位。

= 0: 不附加。

= 1: 在校验位和停止位之间附加1位奇偶标志位。若采用偶校验，则这个标志为逻辑0；若采用奇校验，则这个标志为逻辑1。选用这一附加标志位的作用是发送设备把采用何种奇偶校验方式也通过数据流告诉接收设备。显然，在收发双方已经约定奇偶校验方式的情况下，不需要这一附加位。

[6] 中止控制位。指定发送正常信号还是连续发送空号（中止符，逻辑0）。

=0：发送正常信号；

=1：发送端会连续发送空号。若发空号的时间超过一个完整的数据字传送时间，接收端就视发送设备已中止发送。此时，若允许中断，则接收设备向CPU产生中断请求。

[7] =1：除数锁存访问位；

=0：表示访问发送保持寄存器/接收缓存寄存器/中断允许寄存器

看了上面的表格，或许你觉得晕：这么多寄存器，到底怎么配呢？别担心，下文会介绍。

UART 初始化编程方法

当 PC 机上电启动时，系统 RESET 信号通过 NS8250 的 MR 引脚使得 UART 内部寄存器和控制逻辑复位。此后若要使用 UART 就需要对其进行初始化编程操作，比如说设置 UART 的波特率、数据位数以及工作方式等。下面以 PC 上的串行端口 COM1 为例说明对其初始化的步骤。该串口的端口基地址是port = 0x3f8。 UART 芯片中断引脚 INTRPT 被连接至中断控制芯片引脚 IRQ4 上。当然，在初始化之前应该首先在 IDT 表中设置好串行中断处理过程的中断描述符。

设置通信传输波特率

设置通信传输波特率就是设置两个除数锁存寄存器 LSB 和 MSB 的值，即 16 位的波特率因子。由上表可知，若要访问这两个除数锁存寄存器，必须首先设置线路控制寄存器 LCR 的第 8 位 DLAB = 1，即向端口 port+3 (0x3fb) 写入 0x80 。 然后对端口 port (0x3f8) 和 port+1 (0x3f9) 执行写操作。

对于指定的波特率，波特率因子的计算公式为：

波特率因子=UART时钟频率/(波特率∗16)

例如：波特率为2400 Baud，则

波特率因子=UART时钟频率/(波特率∗16)=

1.8432MHz/(2400∗16)=1843200/(2400∗16)=48

48=0x0030，我们需要在 LSB 中写入0x30 , 在 MSB 中写入0x00 。 波特率设置好后，最好复位线路控制寄存器的 DLAB 位。

设置通信传输格式

串行通信传输格式由线路控制寄存器 LCR 中的各个位来定义。其中每位的含义见上表所示。如果我们需要把传输格式设置成无奇偶校验位、8 位数据位和 1 位停止位，那么就需要向LCR 写入值 0x03。

设置 MODEM 控制寄存器

对该寄存器进行写入操作可以设置 UART 的操作方式和控制 MODEM。 UART操作方式有中断和查询两种。还有一种循环反馈方式，但该方式仅用于诊断测试 UART 芯片的好坏，不能作为一种实际的通信方式使用。在 PC 机 ROM BIOS 中使用的是查询方式，但本文是为了搞清楚 Linux-0.11系统而写的，由于 Linux-0.11 采用的是高效率的中断方式，因此我们只介绍中断方式下 UART 的编程方法。

中断方式是指当 MODEM 状态发生变化时、接收出错时、发送保持寄存器空时、或者接收到一个字符时允许UART 通 过 INTRPT 引脚向 CPU 发出中断请求信号。至于允许哪些中断请求则由中断允许寄存器 IER 来确定。但是若要让 UART 的中断请求信号能够送到 8259A 中断控制器，就需要把 MODEM 控制寄存器 MCR 的位3(OUT2) 置位。因为在PC 机中，该位控制着 INTRPT 引脚到 8259A 的电路。

查询方式是指 MODEM 控制寄存器 MCR 位3(OUT2)复位的条件下，程序通过循环查询 UART 寄存器的内容来接收/发送串行数据。当 MCR的位3 = 0 时，虽然在 MODEM 状态发生变化等条件下 UART 仍然能使 INTRPT 引脚产生中断请求信号，并且能根据产生中断的条件设置中断标识寄存器 IIR 的内容，但是中断请求信号并不能被送到8259A 中。因此程序只能通过查询线路状态寄存器 LSR 和中断标识寄存器 IIR 的内容来判断 UART 的当前工作状态并进行数据的接收和发送操作。

MCR的位 1 和位 0 分别用于控制 MODEM ， 当这两位置位时，UART的数据终端就绪引脚（DTR）和请求发送引脚（RTS）输出有效。

若要把 UART 设置成中断方式，并且使 DTR 和 RTS 有效，那么就需要向 MODEM 控制寄存器写入0x0b ， 即1011b 。

初始化中断允许寄存器

中断允许寄存器 IER 用来设置可产生中断的条件，即中断来源类型。共有 4 种中断源类型可供选择，对应比特位置1 表示允许该条件产生中断，否则禁止。当中断产生时，具体是哪个中断源产生的中断由中断标识寄存器 IIR 中的 [2:1] 指明，并且读写特定寄存器的内容可以复位 UART 的中断。IER 的位 0 用于确定当前是否有中断，位0=0表示有待处理的中断。

在 Linux 0.11 串行端口初始化函数中，设置允许 3 种中断源产生中断(写入0x0d)，即在 MODEM 状态发生变化时、接收有错时、接收器收到字符时都允许产生中断，但不允许发送保持寄存器空产生中断。因为我们此时还没有数据要发送。当对应串行终端的写队列有数据要发送时，tty\_write()函数会调用rs\_write()函数来置位发送保持寄存器空中断允许标志，从而在该中断源引发的串行中断处理过程中，内核程序就可以开始取出写队列中的字符发送输出到发送保持寄存器中，以让 UART 发送出去。一旦 UART 把该字符发送了出去，发送保持寄存器又会变空而引发中断请求。于是只要写队列中还有字符，系统就会重复这个处理过程，把字符一个一个地发送出去。当写队列中所有字符都发送出去后，写队列变空了，中断处理程序就会把中断允许寄存器中的发送保持寄存器空中断允许标志复位掉，从而再次禁止发送保持寄存器空引发中断请求。此次“循环”发送操作也随之结束。

UART 中断处理程序编程方法

由上文可知，UART 可由 4 种不同的中断源类型产生中断。因此当串行中断处理程序刚开始执行时，仅知道发生了中断，但不知道是哪种情况引起了中断。所以串行中断处理程序的第一个任务就是确定产生中断的具体条件。这需要借助于中断标识寄存器 IIR 来确定产生当前中断的源类型。因此串行中断处理程序可以根据产生中断的源类型使用子程序地址跳转表jmp\_table[]来分别处理，其伪代码如下：

rep\_int:

inb %dx,%al // 读中断标识寄存器 IIR 到 AL

testb $1,%al // 判断有无待处理的中断（位0==0说明有中断）

jne end // 若没有待处理的中断，则跳转到end

// 根据IIR[2:1]的值调用对应的处理函数

call \*jmp\_table(,%eax,2) /\* NOTE! not \*4, bit0 is 0 already \*/

jmp rep\_int // 跳转，继续判断有无待处理中断

end:

movb $0x20,%al //中断退出处理。向中断控制器发送结束中断指令E0I 。

outb %al,$0x20 /\* EOI \*/

...

iret

jmp\_table:

.long modem\_status,write\_char,read\_char,line\_status

//edx中是串口基地址0x3F8

modem\_status:

addl $6,%edx /\* clear intr by reading modem status reg \*/

inb %dx,%al // 读Modem状态寄存器

ret

line\_status:

addl $5,%edx /\* clear intr by reading line status reg. \*/

inb %dx,%al // 读线路状态寄存器

ret

read\_char:

inb %dx,%al // 读取接收缓冲寄存器中的字符

...

ret

write\_char:

// 由于设置了发送保持寄存器允许中断标志而引起此次中断,

// 说明对应串行终端的写字符缓冲队列中有字符需要发送。

写字符队列出队-->al;

outb %al,%dx //写到发送保持寄存器中

if(写字符队列为空)

{

屏蔽发送保持寄存器空中断;

}

ret

在取出 IIR 的内容后，首先根据位 0 判断是否有待处理的中断。若位 0 = 0，则表示有需要处理的中断。再根据位[2:1]使用指针跳转表调用相应的子处理程序。在每个子程序中，会在处理完后复位 UART 的相应中断源。当子处理程序返回后，主处理程序会循环判断是否还有其他中断源（[0] ==0 ? ) 。如果本次中断还有其他中断源，那么 IIR 的位 0 仍然是0。于是中断处理程序会再调用相应中断源的子处理程序……直到引起本次中断的所有中断源都被处理并复位，此时 UART 会自动设置 IIR 的位 0 = 1 ，表示已无待处理的中断，于是中断处理程序即可退出。

---------------------

作者：ARM的程序员敲着诗歌的梦

来源：CSDN

原文：https://blog.csdn.net/longintchar/article/details/79905324

版权声明：本文为博主原创文章，转载请附上博文链接！