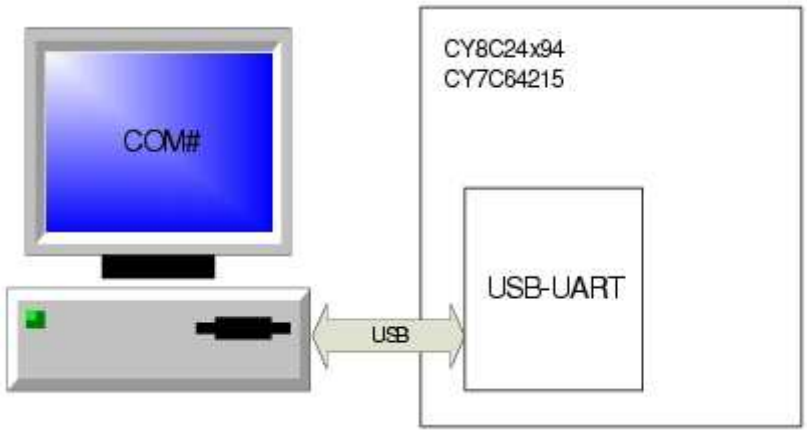
USBUART Device Data Sheet

USBUART vX.Y

资源	PSoC®模块			应用程序接口内存（字节）		引脚（每个外部 I/O）
	数字	模拟 CT	模拟 SC	闪存	RAM	
CY8C24x94, CY8CLED04				1900	60	2
CY7C64215				1900	60	2

特征和概述

- USBUART 设备利用了 1 个 USB 接口来仿真 1 个 COM 端口。
- 在 PSoC 器件侧提供了类似于 UART 的高层函数。



USBUART 设备原理方框图

功能说明

许多内嵌应用均使用 RS-232 接口与诸如 PC 机的外部系统通讯，特别是在调试过程中。但在 PC 机领域，RS-232 COM 端口已经就要从最新生产的计算机中消失了，它们均采用 USB 端口作为串行通讯的替代品。将某种设备移植到 USB 的最简单方式是在 USB 总线上仿真 RS-232 端口。这种方法的主要优点在于，PC 应用程序将 USB 连接当作 RS-232 COM 连接，从而让调试操作非常易于实现。这种方法可以使用包含在从 Windows 98SE 到 Windows XP 的所有版本的 Microsoft® Windows 操作系统中的标准 Windows®驱动程序。

USB 通讯设备类（CDS）规格定义了许多通讯模型，第 3.6.2.1 章内包括了一个在 USB 上实现串行通讯仿真的抽象控制模型。详细内容请参见 CDC 规格的 1.1 版本。Microsoft Windows USB 调制解调器驱动程序 usbser.sys 符合此规格。

在一台新设备第一次连接到一台 Windows PC 时，Windows 将要求用户提供一个驱动程序。在 Windows 2000 及以后版本在安装驱动程序时要求一个 INF 文件。Microsoft Windows 没有为 usbser.sys 驱动程序提供标准的 INF 文件。为了安装一个能够在 USB 上仿真 RS-232 的设备，用户必须提供一个能够将附加的设备映射到 Microsoft CDC 驱动程序的 INF 文件。USBUART 项目的必要 INF 文件将自动生成并且位于项目的 LIB 文件夹内。在提供了 INF 文件后，驱动程序允许 USBI 设备仿真成一个 USB 端口。

在终端应用程序中的设置（波特率、数据位、奇偶校验、停止位和流控制）不会影响数据传输的性能，因为它是台 USB 设备而且使用 USB 协议来控制数据流。但是，除了流控制以外的终端设置值可以采用特定的 API 调用来检索，以便在需要时用于 RS-232 设备。因为 CDC 驱动程序不支持流控制设置值，所以无法检索。

可使用下列 API 调用来检索特定的设置值：

时序

USBUART 用户模块支持在 CY8C24x94 和 CY7C64215 器件上的全速 USB 2.0 运行。

参数

供货商 ID

每个 USB 产品必须拥有一个唯一的供货商 ID（VID）和产品 ID（PID）的组合。这个 2 字节的字符串包含了供货商 ID。供货商 ID 由 USB Implementers Forum 指定。

产品 ID

每个 USB 产品必须拥有一个唯一的供货商 ID（VID）和产品 ID（PID）的组合。这个 2 字节字符串包含了产品 ID。产品 ID 由制造厂商指定而且必须对于产品是唯一的。

一个用于描述产品制造厂商的格式自由的字符串。.在 VendorString（供货商字符串）内不要使用省略符号(`)。

一个用于描述产品的格式自由的字符串。在 ProductString（产品字符串）内不要使用省略符号(`)。

选择序列号的类型。可能的设置值在下表中给出。

参数	说明
无	此设备不拥有序列号。在 SerialNumberString 参数内输入的数值被忽略。
自动	序列号从 PSoC 器件串行化序号自动生成。序列号是 24 个 16 进制的字符。在 SerialNumberString 参数内输入的数值被忽略。
手工	使用输入到 SerialNumberString 参数内的数值。

设置设备的序列号。推荐采用数字数值。只在 SerialNumberType 参数设置为手动时适用。

用于选择设备的电源。设备可以自行供电或从 USB 端口供电。

在设备从 USB 总线供电时，设置从 USB 总线所消耗的功率（以 mA 为单位）。如果设备自行供电，则此参数被忽略。最小值为 1 mA，最大值为 500 mA。通常，对于低功率设备，用户将此值设置为 100 mA，对于高功率设备，用户将此值设置为 500 mA。

应用程序接口

应用编程接口（API）子程序作为本用户模块的一部分进行了提供，它让设计人员能够在更高层次上与本模块打交道。本节具体指明了每个函数的接口以及由“include”文件所提供的常数。

备注：在这里，就像在所有用户模块 API 中一样，A 和 X 寄存器的数值有可能在调用 API 函数时发生改变。如果在调用时需要使用 A 寄存器和 X 寄存器的数值要，则调用函数有责任在调用前保存 A 寄存器和 X 寄存器内的数值。选择这种“寄存器易变”策略原因在于提高效率，并且自从 PsoC Designer 的 1.0 版本起使用。C 语言编译器会自动处理好这项要求。汇编语言编程者也必须确保自己的代码遵守这一策略。虽然某些用户模块 API 函数可能保持寄存器 A 和 X 不变，但也不保证这些函数以后还能保证如此。

下表列出了 USBUART 模块提供的 API 函数。

USBUART API	
功能	**说明**
void **USBUART_Start**(BYTE bVoltage)	启用此用户模块以配合设备使用。
void **USBUART_Stop**(void)	禁用用户模块。
BOOL **USBUART_Init**(void)	初始化 USBUART 模块。如果 USBUART 成功初始化了，返回一个非零值。
void **USBUART_Write**(BYTE * pData, BYTE bLength)	从 pData 数值向 PC 机发送 bLength 个字节。
void **USBUART_CWrite**(const BYTE * pData, BYTE bLength)	从常数（ROM）pData 数组向 PC 机发送 bLength 个字节。
void **USBUART_PutString**(BYTE * pStr)	向 PC 发送一个以空字符结尾 pStr 的字符串。

void **USBUART_CPutString**(const BYTE * pStr)	向 PC 发送一个常数（ROM）空字符结尾字符串 pStr。
void **USBUART_PutChar**(BYTE bChar)	送 1 个字符至 PC
void **USBUART_PutCRLF**(void)	发送回车符（0x0D）和换行符（0x0A）至 PC 机。
void **USBUART_PutSHexByte**(BYTE bValue)	发送 bValue 的 1 个 2 字符的 16 进制数表示形式至 PC 机。
void **USBUART_PutSHexInt**(INT iValue)	发送 iValue 的 1 个 4 字符的 16 进制数表示形式至 PC 机
BYTE **USBUART_bGetRxCount**(void)	返回当前准备好读取的当前字节个数。
BYTE **USBUART_bTxIsReady**(void)	如果 USBUART 准备好发送数据，则返回一个非零值。
BYTE **USBUART_Read**(BYTE * pData, BYTE bLength)	从端点 RX 缓冲区中读取指定数量的字节，并将数据放置在由 pData 所指定的 RAM 数组中。此函数返回了剩余在 RX 缓冲区内的字节个数以及运行状态。
void **USBUART_ReadAll**(BYTE * pData)	从端点 RX 缓冲区中读取所有可用数据，并将数据放置在由 pData 所指定的 RAM 数组中。
WORD **USBUART_ReadChar**(void)	在这个返回值的最低有效字节（LSB）内返回 1 个来自 RX 缓冲区的字节。此函数还在返回数值的最大有效字节（MSB）内返回了运行状态和剩余在 RX 缓冲区内的字节数量。
BYTE **USBUART_bCheckUSBActivity**(void)	检查和清空总线活动标志位。如果自从上次检查起 USB 曾进入过活动状态则在寄存器 A 内返回 1，否则返回 0。
DWORD ***USBUART_dwGetDTERate**(DWORD * dwDTERate)	返回为此端口设置的以位/秒为单位的数据终端率。
BYTE **USBUART_bGetCharFormat**(void)	返回停止位的数量。
BYTE **USBUART_bGetParityType**(void)	返回奇偶校验类型。
BYTE **USBUART_bGetDataBits**(void)	返回数据位的数量。
BYTE **USBUART_bGetLineControlBitmap**(void)	返回 DTE 和 RTS 信号状态。
void **USBUART_SendStateNotify**(BYTE bState)	发送有关当前 UART 的状态的通知至 PC 机。

USBUART_Start

说明：

执行启动 USBUART 设备用户模块所需要的全部操作。

C 语言：

```
void USBUART_Start(BYTE bVoltage)
```

汇编语言：

```
mov  A, USBUART_5V_OPERATION  ; Select the Voltage level
```

```
call USBUART_Start       ; Call the Start Function
```

参数：

bVoltage 指的是在累加器内传递的芯片的工作电压。此数值决定是否启用调节器以在 5V 下运行，或采用直通模式用于 3V 运行。下表给出了在 C 语言程序和汇编语言程序内提供的符号名称及其相关数值。

屏蔽位	数值	说明
USBUART_3V_OPERATION	0x02	禁用调压器并将 Vcc 直通用于上拉
USBUART_5V_OPERATION	0x03	启用调压器并使用调压器进行上拉

返回值：

无

注意：

寄存器 A 和寄存器 X 均有可能在此函数的本次执行中或后续执行中被修改。在大内存模式下，所有 RAM 页面指针寄存器也会出现这种状况。在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。目前，只有 IDX_PP 和 CUR_PP 页面指针寄存器发生改变。

USBUART_Stop

说明：

执行 USBUART 设备用户模块关闭所需要的全部操作任务。

C 语言：

```
void USBUART_Stop(void)
```

汇编语言：

```
call USBUART_Stop
```

参数：

无

返回值：

无

注意：

寄存器 A 和寄存器 X 均有可能在此函数的本次执行中或后续执行中被修改。在大内存模式下，所有 RAM 页面指针寄存器也会出现这种状况。在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。目前，只有 CUR_PP 页面指针寄存器被修改。

USBUART_Init

说明：

尝试初始化 USBUART 设备和设置与 PC 机的通讯。

C 语言：

```
BOOL USBUART_Init(void)
```

汇编语言：

```
call USBUART_Init
```

参数：

无

返回值：

如果设备成功初始化，可在累加器内返回一个非零值。如果初始化失败要返回 0。用户模块只能在成功初始化后运行。

注意：

寄存器 A 和寄存器 X 均有可能在此函数的本次执行中或后续执行中被修改。在大内存模式下，所有 RAM 页面指针寄存器也会出现这种状况。在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。目前，只有 IDX_PP 和 CUR_PP 页面指针寄存器发生改变。

USBUART_Write

说明：

从（RAM）指针 pData 所指定的位置发送 bLength 个字符至 PC 机。

C 语言：

```
void USBUART_Write(BYTE * pData, BYTE bLength)
```

汇编语言：

```
mov  A,20        ; Load array count
push  A
mov  A,>pData     ; Load MSB part of pointer to RAM string
push  A
```

```
                    mov   A,<pData      ; Load LSB part of pointer to RAM
string

push  A

call  USBUART_Write   ; Make call to function

add   SP,253        ; Reset stack pointer to original position
```

参数：

pData 是一个指向数据数组的指针。这个数据数组的最大长度为 32 个字节。

bLength 是将要从数组传输并发送至 PC 机的字节的数量。有效数值在 0 至 32 之间。

返回值：

无

注意：

寄存器 A 和寄存器 X 均有可能在此函数的本次执行中或后续执行中被修改。在大内存模式下，所有 RAM 页面指针寄存器也会出现这种状况。在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。目前，只有 IDX_PP 和 CUR_PP 页面指针寄存器发生改变。

USBUART_CWrite

说明：

从（ROM）指针 pData 所指定的位置发送 bLength 个字符至 PC 机。

C 语言：

```
void USBUART_CWrite(const BYTE * pData, BYTE bLength)
```

汇编语言：

```
mov   A,20        ; Load array count

push  A

mov   A,>pData      ; Load MSB part of pointer to ROM string

push  A

mov   A,<pData      ; Load LSB part of pointer to ROM string

push  A
```

```
                    call  USBUART_CWrite   ; Make call to function
add   SP,253        ; Reset stack pointer to original position
```

参数：

pData 是一个指向 ROM 内的一个数据数组的指针。这个数据数组的最大长度为 32 个字节。

bLength 是将要从数组传输并发送至 PC 机的字节的数量。有效数值在 0 至 32 之间。

返回值：

无

注意：

寄存器 A 和寄存器 X 均有可能在此函数的本次执行中或后续执行中被修改。在大内存模式下，所有 RAM 页面指针寄存器也会出现这种状况。在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。目前，只有 IDX_PP 和 CUR_PP 页面指针寄存器发生改变。

USBUART_PutString

说明：

发送一个以空字符结尾（RAM）的字符串至 PC 机。

C 语言：

```
void USBUART_PutString(BYTE * pStr)
```

汇编程序：

```
mov  A,>pStr       ; Load MSB part of pointer to RAM based null
                   ; terminated string
mov  X,<pStr       ; Load LSB part of pointer to RAM based null
                   ; terminated string
call USBUART_PutString ; Call function to send string out
```

参数：

pStr：指向将要发送至 PC 机的字符串的指针。最大有效字节（MSB）在累加器内传递，而最小有效字节（LSB）在寄存器 X 内传递。最大字符长度是 32 个字节，包括结尾用的空字符。

返回值：

无

注意：

寄存器 A 和寄存器 X 均有可能在此函数的本次执行中或后续执行中被修改。在大内存模式下，所有 RAM 页面指针寄存器也会出现这种状况。在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。目前，只有 IDX_PP 和 CUR_PP 页面指针寄存器发生改变。

USBUART_CPutString

说明：

发送一个以空字符结尾（ROM）的字符串至 PC 机。

C 语言：

```
void USBUART_CPutString(const BYTE * pStr)
```

汇编程序：

```
mov  A,>pStr       ; Load MSB part of pointer to ROM based null
                   ; terminated string
mov  X,<pStr       ; Load LSB part of pointer to ROM based null
                   ; terminated string
call USBUART_PutString ; Call function to send string out
```

参数：

pStr：指向将要发送至 PC 机的字符串的指针。最大有效字节（MSB）在累加器内传递，而最小有效字节（LSB）在寄存器 X 内传递。最大字符长度是 32 个字节，包括结尾用的空字符。

返回值：

无

注意：

寄存器 A 和寄存器 X 均有可能在此函数的本次执行中或后续执行中被修改。在大内存模式下，所有 RAM 页面指针寄存器也会出现这种状况。在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。目前，只有 IDX_PP 和 CUR_PP 页面指针寄存器发生改变。

USBUART_PutChar

说明：

写入单个字符至 PC 机。

C 语言：

```
void USBUART_PutChar(BYTE bChar)
```

汇编程序：

```
mov  A,0x33       ; Load ASCII character "3" in A
call USBUART_PutChar  ; Call function to send single character to PC
```

参数：

bChar:将要发送至 PC 机的字符。数据在累加器内传递。

返回值：

无

注意：

寄存器 A 和寄存器 X 均有可能在此函数的本次执行中或后续执行中被修改。在大内存模式下，所有 RAM 页面指针寄存器也会出现这种状况。在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。目前，只有 IDX_PP 和 CUR_PP 页面指针寄存器发生改变。

USBUART_PutCRLF

说明：

发送一个回车符（0x0D）和一个换行符（0x0A）至 PC 机。

C 语言：

```
void USBUART_PutCRLF(void)
```

汇编程序：

```
call USBUART_PutCRLF    ; Send a carriage return and line feed out
```

参数：

无

返回值：

无

注意：

寄存器A和寄存器X均有可能在此函数的本次执行中或后续执行中被修改。在大内存模式下，所有 RAM 页面指针寄存器也会出现这种状况。在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。目前，只有 IDX_PP 和 CUR_PP 页面指针寄存器发生改变。

USBUART_PutSHexByte

说明：

发送数据的 1 个 2 字节 ASCII 码 16 进制表达方式至 PC 机。

C 语言：

```
void USBUART_PutSHexByte(BYTE bValue)
```

汇编程序：

```
mov  A,0x33         ; Load data to be sent
call USBUART_PutSHexByte ; Call function to output hex representation of
                    ; data. The output for this value would be "33".
```

参数：

bValue：将要转换为 ASCII 字符串的字节（16 进制表示形式）。数据在累加器内传递。

返回值：

无

注意：

寄存器 A 和寄存器 X 均有可能在此函数的本次执行中或后续执行中被修改。在大内存模式下，所有 RAM 页面指针寄存器也会出现这种状况。在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。目前，只有 IDX_PP 和 CUR_PP 页面指针寄存器发生改变。

USBUART_PutSHexInt

说明：

发送数据的 1 个 4 字节 ASCII 码 16 进制表达形式至 PC 机。

C 语言：

```
void USBUART_PutSHexInt(INT iValue)
```

汇编程序：

```
mov  A,0x34       ; Load LSB in A
```

```
                mov  X,0x12       ; Load MSB in X

call UART_PutSHexInt ; Call function to output hex representation of data.

                ; The output for this value would be "1234".
```

参数：

iValue：将要转换为 ASCII 字符串的整数（16 进制表示形式）。最大有效字节（MSB）在寄存器 X 内传递，而最小有效字节（LSB）在累加器内传递。

返回值：

无

注意：

寄存器 A 和寄存器 X 均有可能在此函数的本次执行中或后续执行中被修改。在大内存模式下，所有 RAM 页面指针寄存器也会出现这种状况。在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。目前，只有 IDX_PP 和 CUR_PP 页面指针寄存器发生改变。

USBUART_bGetRxCount

说明：

此函数返回已经从 PC 机接收到字节并且在 RX 缓冲区内等待的字节数量。

C 语言：

```
BYTE USBUART_bGetRxCount(void)
```

汇编语言：

```
call USB_bGetEPCount
```

参数：

无

返回值：

在寄存器 A 内返回当前字节数量。

注意：

寄存器 A 和寄存器 X 均有可能在此函数的本次执行中或后续执行中被修改。在大内存模式下，所有 RAM 页面指针寄存器也会出现这种状况。在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。目前，只有 IDX_PP 页面指针寄存器被修改。

USBUART_bTxIsReady

说明：

如果 TX 缓冲区准备好发送更多数据，则返回一个非零值。否则返回零。

C 语言：

```
BYTE USBUART_bTxIsReady(void)
```

汇编语言：

```
call USBUART_bTxIsready
```

参数：

无

返回值：

如果 TX 缓冲区能够接收数据，则此函数返回一个非零值。否则，返回一个零值。

注意：

寄存器 A 和寄存器 X 均有可能在此函数的本次执行中或后续执行中被修改。在大内存模式下，所有 RAM 页面指针寄存器也会出现这种状况。在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。

USBUART_Read

说明：

从端点 RX 缓冲区中读取 bLength 个字节的收到数据，并将数据放置在由 pData 所指定的数据数组内。

C 语言：

```
BYTE USBUART_Read(BYTE * pData, BYTE bLength)
```

汇编语言：

```
mov A, 25        ; Load count
push A
mov A, >pData      ; Load MSB part of pointer to RAM array
push A
mov A, <pData      ; Load LSB part of pointer to RAM array
```

```
push A
call USBUART_Read
```

参数：

pData 是一个指向数据数组的指针。这个数据数组的最大长度为 32 个字节。

bLength 是将要读取到这个数组的字节数量。有效数值在 0 至 32 之间。

返回值：

使用累加器的 0...6 位返回剩余在 RX 缓冲区内的字节数量，并用累加器的最高有效位（MSb）来指示错误状况。在用户请求超过缓冲区内可用数量更多的字节时，通常会发生错误状况。来自 RX 缓冲区的数据将放置到由 pData 所指定的数据数组。

注意：

寄存器 A 和寄存器 X 均有可能在此函数的本次执行中或后续执行中被修改。在大内存模式下，所有 RAM 页面指针寄存器也会出现这种状况。在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。目前，只有 IDX_PP 和 CUR_PP 页面指针寄存器发生改变。

USBUART_ReadAll

说明：

从端点 RX 缓冲区中读取所有已经接收到的数据，并将数据放置在由 pData 所指定的数据数组内。

C 语言：

```
void USBUART_ReadAll(BYTE * pData)
```

汇编语言：

```
mov  A,>pData      ; Load MSB part of pointer to RAM buffer
mov  X,<pData      ; Load LSB part of pointer to RAM buffer
call USBUART_ReadAll
```

参数：

pData 是一个指向数据数组的指针。最大有效字节（MSB）在累加器内传递，而最小有效字节（LSB）在寄存器 X 内传递。这个数据数组的最大尺寸为 32 个字节。

返回值：

无

注意：

寄存器 A 和寄存器 X 均有可能在此函数的本次执行中或后续执行中被修改。在大内存模式下，所有 RAM 页面指针寄存器也会出现这种状况。在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。目前，只有 IDX_PP 和 CUR_PP 页面指针寄存器发生改变。

USBUART_ReadChar

说明：

从 RX 缓冲器寄存器中读取 1 个字节的已接收数据。

C 语言：

```
WORD USBUART_ReadChar(void)
```

汇编语言：

```
call USBUART_ReadChar
```

参数：

无

返回值：

返回值（累加器）的最高有效字节（MSB）用第 0...6 位内包含了剩余到 RX 缓冲区内的字节数量。而用第 7 位指示错误状态。如果在函数被调用时缓冲区清空时则第 7 位置位。返回数值（X）的最低有效字节 （LSB）包含了 1 个来自缓冲区的字符。

注意：

寄存器 A 和寄存器 X 均有可能在此函数的本次执行中或后续执行中被修改。在大内存模式下，所有 RAM 页面指针寄存器也会出现这种状况。在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。目前，只有 IDX_PP 和 CUR_PP 页面指针寄存器发生改变。

USBUART_bCheckUSBActivity

说明：

检查 USB 上的活动状态。

C 语言：

```
BYTE USBUART_bCheckUSBActivity(void)
```

汇编语言：

```
call USB_bCheckUSBActivity
```

参数：

无

返回值：

如果自从上次检查起 USB 曾进入过活动状态则在累加器内返回 1，否则返回 0。

注意：

寄存器 A 和寄存器 X 均有可能在此函数的本次执行中或后续执行中被修改。在大内存模式下，所有 RAM 页面指针寄存器也会出现这种状况。在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。

USBUART_dwGetDTERate

说明：

返回为此端口设置的以位/秒为单位的数据终端率。向此函数传递一个指向一个 DWORD 的指针。此函数在此指针所引用的位置返回 DTE 速率。

C 语言：

```
DWORD *USBUART_dwGetDTERate(DWORD * dwDTERate)
```

汇编语言：

```
mov  A,>dwDTERate    ; Load MSB part of pointer
mov  X,<dwDTERate    ; Load LSB part of pointer
call USBUART_dwGetDTERate
```

参数：

dwDTERate：一个用于指向函数返回时将会存储 DTE 速率的位置。

返回值：

在由函数所传递的指针所引用的位置存储 DTE 速率 DWORD 数值，并随后返回一个指向此位置的指针。

注意：

寄存器 A 和寄存器 X 均有可能在此函数的本次执行中或后续执行中被修改。在大内存模式下，所有 RAM 页面指针寄存器也会出现这种状况。在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。目前，只有 IDX_PP 和 CUR_PP 页面指针寄存器发生改变。

USBUART_bGetCharFormat

说明：

返回停止位的数量。

C 语言：

```
BYTE USBUART_bGetCharFormat(void)
```

汇编语言：

```
call USBUART_bGetCharFormat
```

参数：

无

返回值：

在累加器内返回停止位的数量。下表给出了在 C 语言程序和汇编语言程序内提供的符号名称及其相关数值。

屏蔽位	数值	说明
USBUART_1_STOPBITS	0x00	1 个停止位
USBUART_1_5_STOPBITS	0x01	1.5 个停止位
USBUART_2_STOPBITS	0x02	2 个停止位

注意：

寄存器 A 和寄存器 X 均有可能在此函数的本次执行中或后续执行中被修改。在大内存模式下，所有 RAM 页面指针寄存器也会出现这种状况。在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。目前，只有 CUR_PP 页面指针寄存器被修改。

USBUART_bGetParityType

说明：

返回奇偶校验类型。

C 语言：

```
BYTE USBUART_bGetParityType(void)
```

汇编语言：

```
call USBUART_bGetParityType
```

参数：

无

返回值

在累加器内返回奇偶校验类型。下表给出了在 C 语言程序和汇编语言程序内提供的符号名称及其相关数值。

屏蔽位	数值	说明
USBUART_PARITY_NONE	0x00	无奇偶校验
USBUART_PARITY_ODD	0x01	奇校验
USBUART_PARITY_EVEN	0x02	偶校验
USBUART_PARITY_MARK	0x03	符号奇偶校验
USBUART_PARITY_SPACE	0x04	空格奇偶校验

注意：

寄存器 A 和寄存器 X 均有可能在此函数的本次执行中或后续执行中被修改。在大内存模式下，所有 RAM 页面指针寄存器也会出现这种状况。在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。目前，只有 CUR_PP 页面指针寄存器被修改。

USBUART_bGetDataBits

说明：

返回数据位的数量。

C 语言：

```
BYTE USBUART_bGetDataBits(void)
```

汇编语言：

```
call USBUART_bGetDataBits
```

参数：

无

返回值：

在累加器内返回数据位的数量。此数量可以是 5、6、7、8 或 16。

注意：

寄存器 A 和寄存器 X 均有可能在此函数的本次执行中或后续执行中被修改。在大内存模式下，所有 RAM 页面指针寄存器也会出现这种状况。在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。目前，只有 CUR_PP 页面指针寄存器被修改。

USBUART_bGetLineControlBitmap

说明：

返回一个表示 RS-232 型控制信号的位图。

C 语言：

```
BYTE USBUART_bGetLineControlBitmap(void)
```

汇编语言：

```
call USBUART_bGetLineControlBitmap
```

参数：

无

返回值：

在累加器内返回 1 个表示控制信号状态的位图。这个位图的每个位都可以独立对待。位 D7..D2 保留不用。下表给出了在 C 语言程序和汇编语言程序内提供的符号名称及其相关数值。

屏蔽位	数值	说明
USBUART_RTS	0x02	RTS (1 – 激活载体； 0 – 停用载体)
USBUART_DTR	0x01	DTR (1 – 存在； 0 – 不存在)

注意：

寄存器 A 和寄存器 X 均有可能在此函数的本次执行中或后续执行中被修改。在大内存模式下，所有 RAM 页面指针寄存器也会出现这种状况。在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。目前，只有 CUR_PP 页面指针寄存器被修改。

USBUART_SendStateNotify

说明：

发送有关 UART 状态的通知至 PC 机。

C 语言：

```
void USBUART_SendStateNotify(BYTE bState)
```

汇编语言：

```
mov A, (USBUART_DCD + USBUART_DSR)
call USBUART_SendStateNotify
```

参数：

在累加器内返回表示控制信号状态的 bState 位图。这个位图的每个位都可以独立对待。下表给出了在 C 语言程序和汇编语言程序内提供的符号名称及其相关数值。

屏蔽位	数值	说明
USBUART_DCD	0x01	RS-232 DCD 信号
USBUART_DSR	0x02	RS-232 DSR 信号
USBUART_BREAK	0x04	中断检测机制的状态
USBUART_RING	0x08	铃声检测信号的状态。
USBUART_FRAMING_ERR	0x10	已经发生了成帧错误。
USBUART_PARITY_ERR	0x20	已经发生了奇偶校验错误。
USBUART_OVERRUN	0x40	由于过速错误，返回数据已经被抛弃。.

返回值：

无

注意：

寄存器 A 和寄存器 X 均有可能在此函数的本次执行中或后续执行中被修改。在大内存模式下，所有 RAM 页面指针寄存器也会出现这种状况。在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。目前，只有 CUR_PP 和 IDX_PP 页面指针寄存器发生改变。

代码范例

以下代码演示了如何在简单的应用中使用 USBUART 用户模块。在一台新设备第一次连接到一台 Windows PC 时，Windows 将要求用户提供一个驱动程序。在 Windows 2000 及以后版本在安装驱动程序时要求一个 INF 文件。Microsoft Windows 没有为 usbser.sys 驱动程序提供 Windows 自带的标准的 INF 文件。为了安装一个能够在 USB 上仿真 RS-232 的设备，用户必须提供一个能够将附加的设备映射到 Microsoft usbser.sys 驱动程序的 INF 文件。USBUART 项目的必要 INF 文件将自动生成并且位于项目的 LIB 文件夹内。要使用这个 INF 文件来安装设备。一旦驱动程序已经安装，此设备将经枚举成为一个 COM 端口，并简单地向所接收到的符号回送至 PC 机。

```
BYTE Len;

BYTE pData[32];

void main()

{

   M8C_EnableGInt;              //Enable Global Interrupts
```

```
                        USBUART_Start(USBUART_5V_OPERATION);  //Start
USBUART 5V operation
   while(!USBUART_Init());         //Wait for Device to initialize
   while(1)
   {
     Len = USBUART_bGetRxCount();    //Get count of ready data
     if (Len)
     {
       USBUART_ReadAll(pData);     //Read all data rom RX
       while (!USBUART_bTxIsReady());//If TX is ready
       USBUART_Write(pData, Len);   //Echo
     }
   }
}
```

采用汇编语言编写的等同代码如下：

```
include "m8c.inc"     ; part specific constants and macros
include "memory.inc"   ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc"  ; PSoC API definitions for all User Modules
AREA bss (RAM, REL)
Len: blk 1
pData: blk 32
export _main

AREA text (ROM, REL)
```

```
                    _main:
M8C_EnableGInt      ; Enable Global Interrupts
MOV  A, USBUART_5V_OPERATION
LCALL USBUART_Start    ; Start USBUART 5V operation
deviceInit:         ; Wait for Device to initialize
LCALL USBUART_Init
CMP  A,0
JZ   deviceInit
mainLoop:
LCALL USBUART_bGetRxCount
MOV  [Len],A        ; Get count of ready data
CMP  [Len],0        ; Check if Len is 0
JZ   mainLoop
mov  A,>pData       ; Load MSB part of pointer to RAM buffer
mov  X,<pData       ; Load LSB part of pointer to RAM buffer
call  USBUART_ReadAll   ; Read all data rom RX
txReady:
LCALL USBUART_bTxIsReady ; Check to see if TX is ready
CMP  A,0
JZ   txReady
             ; Echo data
mov  A,[Len]        ; Load array count
push  A
```

```
                    mov   A,>pData      ; Load MSB part of pointer to RAM
string

push  A

mov   A,<pData      ; Load LSB part of pointer to RAM string

push  A

call  USBUART_Write

add   SP,253        ; Reset stack pointer to original position

jmp   mainLoop
```

寄存器配置

本节描述了 USBUART 设备用户模块所使用或修改的 PSoC 资源寄存器。

资源 EP0_CNTL：Bank 0 reg[D7] 端点 0 控制寄存器								
位	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
数值	Setup Received	IN Received	OUT Received	ACK'd Transaction	模式			

Setup Received：在此位为 1 时，则此位表示已经收到并确认应答了有效的 SETUP 数据包。从 SETUP 事务处理的数据包开始起，此位被强制为高电平，直到 SIE 所返回的确认（ACK）数据包的开始。在这段间隔时间内，已采取措施以防止 CPU 清空此位。在这段间隔时间，此位将在被固件清空之前均保持置位。在此位在设置为 1 时，CPU 无法写入至 EP0_DRx 寄存器。这样可以在固件获得读取 SETUP 数据的机会前，防止固件覆写掉已进入的 SETUP 事务处理。此位可以由任何指向这个寄存器的未锁定写入操作来清空。

IN Received：在此位为 1 时，则此位表示已经收到了有效的 IN 数据包。在主机确认了 IN 数据包之后，此位设置为 1。在此位为零时，既可以表示尚未接收到 IN，也可以表示主机没有通过发送确认握手应答来确认 IN 数据。此位可以由任何指向这个寄存器的未锁定写入操作来清零。

OUT Received：在此位为 1 时，则此位表示已经收到并确认应答了有效的 OUT 数据包。在 OUT 事务处理中的最后一个收到数据包后此位置位为 1。在此位为零时，表示没有收到 OUT 数据包。此位可以由任何指向这个寄存器的未锁定写入操作来清零。

ACK'd Transaction：无论何时 SIE 投入一个指向用确认数据包完成寄存器的端点的事务处理时，此位为 1。此位可以由任何指向这个寄存器的未锁定写入操作来设置为零。

模式（Mode）：模式位控制了 USB SIE 如何对流量做出响应，以及 USB SIE 如何根据主机数据包发至端点的结果而改变端点的模式。

模式	说明	模式	说明
1h	否认（NAK）IN/OUT 接受否认 否认 否认 IN 和 OUT 信令。	9h	确认（ACK）OUT（停顿=0） 忽略 忽略 确认。 在向 OUT 发出确认握手信号后，此模式由 SIE 改变成模式 8h。
2h	状态 OUT Only 接受 STALL 检查控制端点，停顿（STALL）IN 和确认零字节 OUT。	Ah	保留 忽略 忽略 忽略
3h	状态 IN/ OUT 接受停顿（STALL）控制端点停顿停顿 IN 和 OUT 信令。	Bh	确认 OUT – 状态 IN 接受 TX 0 字节 确认 确认 OUT 信令或发送零字节数据用于 IN 信令。
4h	保留 忽略 忽略 忽略	Ch	否认 IN 忽略 否认忽略 发送否认握手用于 IN 信令。
5h	ISO OUT 忽略 忽略 总是等时 OUT。	Dh	确认 IN（停顿=0） 忽略 TX 控制 忽略 此模式由 SIE 在接收发给 IN 数据的确认握手信号后改变成模式 Ch
6h	状态 IN Only 接受 TX 0 字节 控制端点停顿，STALL OUT 和发送零字节数据用于 IN 信令	Eh	保留 忽略 忽略 忽略
7h	ISO IN 忽略 TX 计数 忽略 等时 IN。	Fh	确认 IN – 状态 OUT 接受 TX 计数检查 对 IN 数据或状态 OUT 做出响应。
8h	否认 OUT 忽略 忽略否认 发送否认握手至 OUT 信令。		

资源 EPx_CNTL：Bank 1 reg[C4-C7] 端点 1-端点 4 控制寄存器

位	7	6	5	4	3	2	1	0
数值	Stall0	Reserved	NakIntEnable	ACK'd Transaction	模式			

Stall（停顿）：在此位为 1 时，如果模式位设置为 ACK-OUT，则 SIE 让一个 OUT 数据包停顿。如果模式位设置为 ACK-IN，则 SIE 让一个 IN 数据包停顿。其它所有模式下，此位必须为零。

NakIntEnable：在设置为 1 时，即使传输采用否认（NAK）应答来结束，此位也会引起一个端点中断的生成。

ACK'd Transaction：无论何时 SIE 投入一个指向用确认（ACK）数据包完成的寄存器端点的事务处理时，此位为 1。在任何对此寄存器的写入操作后，此位为零。

模式（Mode）：与上面的 EP0_CNTL 相同。

资源 EP0_CNT：Bank 0 reg[D7] 端点 0 控制寄存器

位	7	6	5	4	3	2	1	0

数值	Data Toggle	Data Valid	保留	字节计数

Data Toggle（数据切换）：此位用于选择 DATA 数据包的切换状态。对于 IN 事务处理，固件必须将此位置位。对于 OUT 或 SETUP 事务处理来说，SIE 硬件设置此位为接收的“Data Toggle（数据切换）“位的状态。

Data Valid（数据有效）：此位仅用于 OUT 事务处理。如是 CRC 校验、位填充或 PID 错误发生了，则此位清空。对于某些端点模式设置来说，此位不会更新。此位可以在寄存器不锁存时通过写入零至此位来清空。

Byte Count（字节计数）：这些位指示了一次事务处理中的数据字节数量。对于 IN 事务处理来说，固件将要发送的字节数量的计数加载到来自端点 FIFO 的主机。有效数值为 0 至 8。对于 OUT 或 SETUP 事务处理来说，此计数位由硬件更新成所接收到的数据字节数量加 2 个用于 CRC 的字节。有效数值为 2 至 10。

资源 EPx_CNT0:Bank 0 reg[4F,51,53,55] 端点 1 – 端点 4 Count0 寄存器

位	7	6	5	4	3	2	1	0
数值	Count LSb							

这些位是 9 位计数器的 8 个最低有效位。最高有效位（MSb）就是 EPx_CNT1 寄存器的计数 MSb。

这个 9 位计数指示了一次事务处理中的数据字节数量。对于 IN 事务处理来说，固件将要发送的字节数量的计数加载到主机。有效数值为 0 至 256。

端点计数的低 8 位还设置了 OUT 事务处理所要接收到的字节数量的极限值。在用于端点的 OUT 事务处理得到接收前，此计数值必须设置为能够接收到的最大字节数量，其中 0x01 即为 1 字节，0xff 既为 255 字节。如果此计数值设置为一个大于所接收字节数量的数值，则来自 USB 数据包和 2 个字节 CRC 的数据均将写入至 USB 的专用 SRAM。

如果计数值小于所接收的数据字节的数量，SIE 将标明此数据包为无效而且不会产生中断。例如，1 个 8 个字节的数据包将试图写入 8 个数据字节和 2 个 CRC 字节。8 及其以上的计数值将让 1 个良好的数据包来产生 1 个中断。7 或 7 以上的计数值将导致 SIE 将此数据包标记为坏。

一旦 OUT 事务处理完成，全部 9 位计数值将由 SIE 更新成 SIE 所接收到的实际数量加上 2 个用于数据包的 CRC。有效数值为 2 至 258。

资源 EPx_CNT1：Bank 0 reg[4E,50,52,54] 端点 1 – 端点 4 Count1 寄存器

位	7	6	5	4	3	2	1	0
数值	Data Toggle	Data Valid	保留					Count MSb

Data Toggle（数据切换）：此位用于选择 DATA 数据包的切换状态。对于 IN 事务处理，固件必须将此位设置成预期的姿态。对于 OUT 事务处理来说，硬件设置此位为所接收的“Data Toggle（数据切换）”位的状态。

Data Valid（数据有效）：此位仅用于 OUT 事务处理且为只读位。如是 CRC 校验、位填充或 PID 错误发生，则此位为零。对于某些端点模式设置来说，此位不会更新。

Count MSb：此位是 9 位计数器的最高有效位。最低有效位是 EPx_CNT 寄存器的 EPx Count[7:0]位。参见 EPx_CNTx 寄存器有关更多信息。

资源 EP0_DRx：Bank 0 reg[57-5F] 端点 0 数据寄存器 0-7

位	7	6	5	4	3	2	1	0
数值	数据							

这些寄存器用于读取和写入数据至 USB 控制端点。这些寄存器为发送和接收操作所共享。EP0_CNT 寄存器内的计数决定了所接收到的或将要传输的字节的数量。

资源 USB_CNTL0：Bank 0 reg[4A] USB 控制寄存器 0

位	7	6	5	4	3	2	1	0
数值	USB 启用	设备地址						

USB 启用：在此位为 1 时，启用 SIE 用于 USB 流量和 USB 接收器。如果此位为零，则此设备不对 USB 流量做出响应。

设备地址：SIE 将向由这些位所指定的 USB 设备地址做出响应。此地址必须由固件进行设置并由系统在 USB 枚举过程中采用 SETUP 指令来进行指定。

资源 USB_CNTL1：Bank 1 reg[C1] USB 控制寄存器 1

位	7	6	5	4	3	2	1	0
数值	保留					总线活动（Bus Activity）	启用锁定（EnableLock）	RegEnable

总线活动（Bus Activity）：这是一个粘着位，用于检测曾经发生在 USB 总线上的任何非空闲 USB 事件。一旦由 SIE 设置为高电平以指示总线活动，此位将在固件清空它之前保留在逻辑高电平数值下。写入 1 个零至此位以对其进行清空；写入 1 个 1 来保留其数值。

启用锁定（EnableLock）：设置此位为 1 可开启内部振荡器用于 USB 流量的自动频率锁定。除非提供了一个外部时钟，此位应当保持在设置为 1 的状态下以确保正确的 USB 运行。

RegEnable：此位控制着内部 USB 调压器的运行方式。对于 PSoC 供电电压在 5V 范围内的应用来说，设置此位为高可以启用内部调压器。对于处于 3.3V 范围的设备供电电压来说，清空此位可将收发器直接连接到电源。

资源 USBIO_CNTL0：Bank 0 reg[4B] USB IO 控制寄存器 0								
位	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
数值	TEN	TSE0	TD	保留				RD

TEN：置位此位可让 USB 输出采用手动驱动方式。通常情况下，TEN 位保持为低电平，这样内部硬件可以自动控制通讯流。一个用于手动 USB 模式的应用程序会驱动一个恢复信号（USB“K”）以将系统从 USB 挂起状态下唤醒。

TSE0：此位用于手工传输一个单端零（ D+和 D-均为低电平）到 USB 引脚上。如果 TEN = 0，则此位没有影响。

TD：此位用于手动驱动 USB J 或 K 状态到 USB 引脚上。如果 TEN=0，则此位没有影响，而且 TSE0 会超越此位。

RD：这个只读位给出了来自不同接收器的 USB 已接收数据的状态。USB_CR0 寄存器内的 USB 启用位必须置位才能接收数据。如果 USB 启用位未置位，则此位将读取零值。

资源 USBIO_CNTL1：Bank 0 reg[4C] USB IO 控制寄存器 1								
位	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
数值	IO 模式	驱动模式	DPI	DMI	PS2PUEN	USBPUEN	DPO	DMO

IO 模式：此位能够让 D+和 D-引脚在配置后既可用于 USB 模式，也可用于“bit banged”模式。如果此位为 1，则 DMI 和 DPI 位用于驱动 D-和 D+引脚。

驱动模式：如果 IO 模式位为 1，则此位用于配置 D-和 D+引脚用于 CMOS 驱动方式或漏极开路驱动方式。如果 IOMode 为零，则此位不受影响。请注意，在漏极开路模式下，5 k 上拉电阻可以内部连接至 PS2PUEN 位。

DPI：如果 IO 模式（IOMode）位=1，则此位用于驱动 D+引脚。

DMI：如果 IO 模式（IOMode）位=1，则此位用于驱动 D-引脚。

PS2PUEN：此位控制着 2 个内部 5 k 上拉电阻至 D+和 D-引脚的连接。

USBPUEN：此位控制着内部 1.5 k 上拉电阻在 D+引脚上的连接。

DPO：这个只读位给出了 D+引脚的状态。

DMO：这个只读位给出了 D-引脚的状态。

资源 USB_SOFx：Bank 0 reg[48-49] USB 帧起始寄存器 0 和 1								
位	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**

USB_SOF0	帧编号	
USB_SOF1	保留	帧编号

帧编号：USB 帧起始寄存器（USB_SOF0 和 USB_SOF1）提供了对于 11 位 SOF 帧编号的访问。USB_SOF0 寄存器存有 SOF 帧编号的低 8 位[7:0]USB_SOF1 寄存器存有 SOF 帧编号的高 3 位[10:8]。

赛普拉斯半导体公司
公司地址：198 Champion Court
San Jose, CA 95134-1709
电话：408-943-2600
传真：408-943-4730
应用支持热线：425.787.4814
网址：http://www.cypress.com