8-Bit 串行发送器数据表 TX8 V 3.50

Copyright © 2001-2010 Cypress Semiconductor Corporation. All Rights Reserved.

资源	PSoC® 模块			API 内存 （字节）		引脚 （每个外部 I/O 和时钟）
	数字	模拟 CT	模拟 SC	闪存	RAM	
CY8C29/27/24/22/21xxx、CYWUSB6953、CY8C23x33、CY8CLED02/04/08/16、CY8CLED0xD、CY8CLED0xG、CY8CTST110、CY8CTMG110、CY8CTST120、CY8CTMG120、CY8CTMA120、CY8C21x45、CY8C22x45、CY8CTMA30xx、CY8C28x45、CY8CPLC20、CY8CLED16P01、CY8C28xxx、 CY8C21x12						
	1	0	0	193	0	1
CY8C26/25xxx	1	0	0	203	0	1

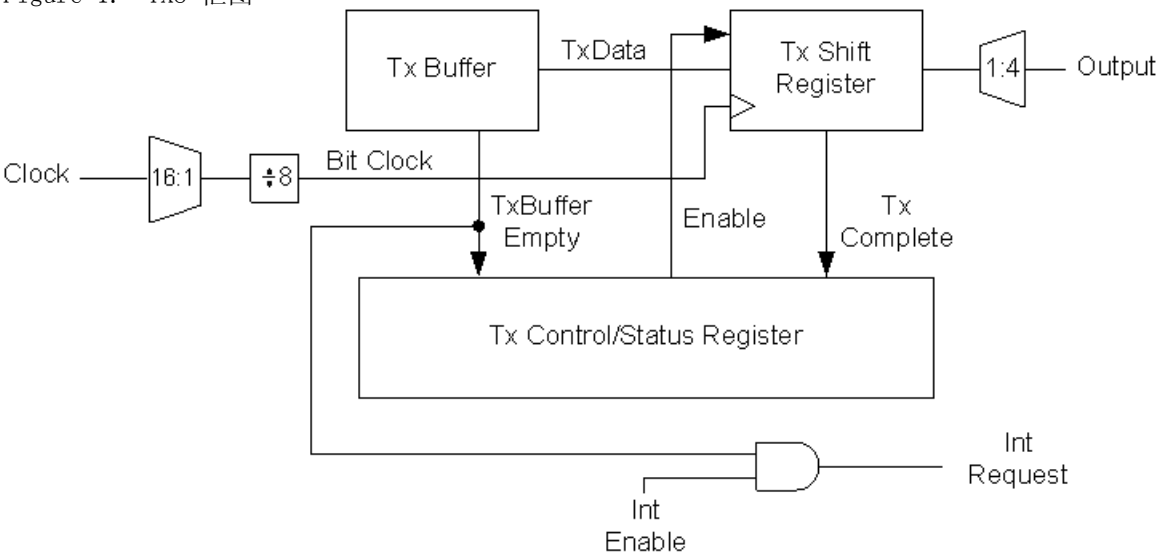
如需一个或多个使用此用户模块的完全配置的功能性示例工程，请转到 www.cypress.com/psocexampleprojects.

特性与概述

- 当 8-bit 串行发送器的可选时钟频率为 48 MHz 时，最大数据速率为 6 Mbit
- 数据成帧包括起始位、可选奇偶校验位和停止位
- RS-232 串行数据符合偶校验、奇校验或无奇偶校验格式
- 发送缓冲区空条件下的可选中断

TX8 用户模块是符合 8-bit RS-232 数据格式的串行发送器，该模块支持可编程的时钟和可选择的中断或轮询的运行方式。 所发送的数据由 1 个起始位、1 个可选奇偶校验位和 1 个停止位成帧。 发送器固件可用于初始化、启动、停止、读取状态和写入数据至 TX8。

Figure 1. TX8 框图



功能说明

TX8 用户模块实现了一种串行发送器。 它使用了一个数字通信型 PSoC 模块的缓冲区、移位和控制寄存器。

控制寄存器通过 TX8 用户模块固件应用程序编程接口 (API) 子程序进行初始化与配置。 当设置控制寄存器内的使能位时，将会生成 1 个内部 8 分频的位时钟。

将要发送的数据字节由 API 子程序写入至缓冲区寄存器，同时会清空控制寄存器内的 “ 缓冲区空 ” 状态位。 该状态位可用于检测和防止发送溢出错误。

在下一个位时钟的上升沿会把数据传输至移位寄存器，并设置控制寄存器的 “ 缓冲区空 ” 位。 如果中断使能掩码被启用，则将触发一个中断。 此中断可被用来启用下一个字节的排队，以便在当前数据字节传输完成时，新字节将在下一个可用发送时钟发送出去。

在数据字节从缓冲区寄存器传输至移位寄存器的同时，起始位将被发送出去。 接下来的位时钟将串行位流移位至输出。 这个位流由数据字节的每个位（低位在前）、1 个可选的奇偶校验位以及最后的 1 个停止位构成。 在完成停止位的发送时，控制寄存器的 “TX 完成 ” 状态位已经置位。 此位将在被读取之前一直有效。 如果新数据字节已经写入至缓冲区寄存器，则数据字节将传输至移位寄存器而且数据的传输将在位时钟的下一个上升沿开始。

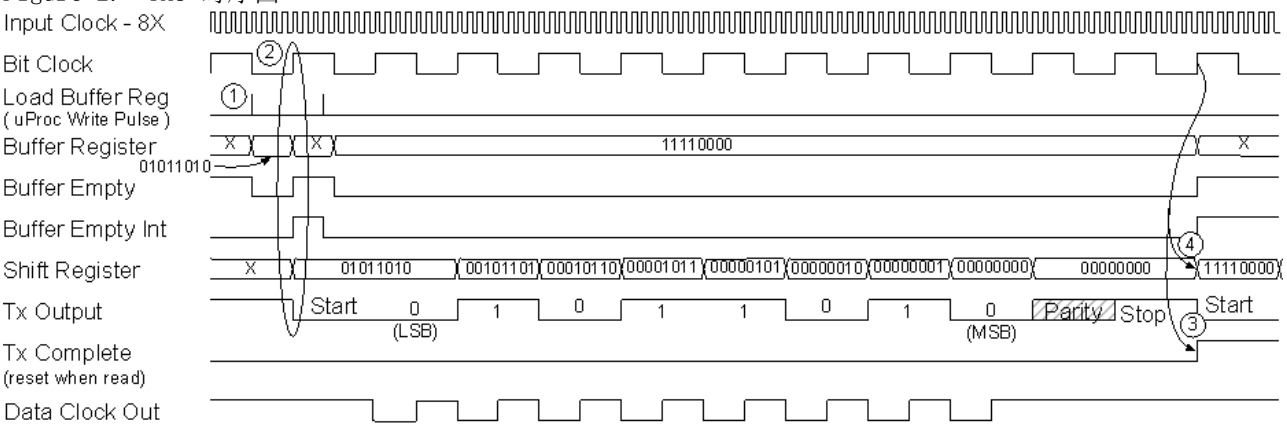
时序

时钟频率必须设置为所期望的位传输速率的 8 倍。

如果启用，将会在出现 “Tx 缓冲区空 ” 事件时发生 TX8 中断。 当传输的数据字节从缓冲区寄存器传输到移位寄存器时将会发生这种情况。 启用和禁用中断由 API 控制。

以下 TX8 时序图显示了 TX8 用户模块的操作。

Figure 2. TX8 时序图



通信系统精度

当使用 PSoC 器件的 SLIMO 模式时，PSoC 系统时钟 (SysClk) 没有足够的精度来保证有效的 UART 通信。 而且，当 PSoC 未连接到 USB 时，PSoC CY8C24x94 系列器件中的 SysClk 没有足够的精度来保证有效的 UART 通信。 系统错误（即通信链路两端的错误总数）必须小于 5%，以便 UART 通信能够正常工作。 有关 SysClk 精度的详细信息，请参见器件数据表。

[+] Feedback

直流和交流电气特性

Table 1. TX8 直流和交流电气特性

参数	条件和注释	典型值	限值	单位
F_{max}	最大传输频率	--	6	Mbits

放置

TX8 用户模块使用名为 “TX” 的单个模块，此模块可自由映射到任意数字通信 PSoC 模块。

参数和资源

时钟

TX8 的时钟来自于 16 个可能来源之一。 此参数通过 PSoC Designer 中的器件编辑器进行设定。 全局 I/O 总线可用于将时钟输入连接至外部引脚或其他 PSoC 模块生成的时钟函数。 当模块使用外部数字时钟时，应将行输入同步关闭，以获得最佳精度以及进行睡眠操作。48 MHz 时钟、CPU_32 kHz 时钟、分频后时钟中的任一个 （24V1 或 24V2）或者另一个 PSoC 模块输出均可以指定为 TX8 时钟输入。

时钟频率必须设置为所期望的位传输速率的 8 倍。 每 10 个输入时钟期间发送一个数据位。

输出

发送器的输出可以被连接至全局输出总线。 全局输出总线可以再连接至外部引脚或其它 PSoC 模块以执行下一步的处理。

TX 中断模式

此选项决定了何时为 TX 模块产生中断。 “TxRegEmpty” 选项能够让中断在数据从数据寄存器传输至移位寄存器后立刻发生。 选择第二个选项 “TxComplete” 将会延迟中断，直到最后位移出移位寄存器。 当有必要知道字符何时被完全传输出去时，第二个选项很有用。 第一个选项 “TxRegEmpty” 最适合用于最大化发送器的输出。 此选项允许在前 1 个字节正在发送时加载 1 个字节。 在中断服务子程序中，TX_CONTROL_REG 应当被读取以启用后续中断。

ClockSync

在 PSoC 器件中，数字模块可以在系统时钟以外提供时钟源。 数字时钟源甚至可以用连锁方式串联起来。 这样就会引起与系统时钟相关的时滞。 这些时滞对于 CY8C29/27/24/22/21xxx 和 CY8CLED04/08/16 PSoC 器件系列具有更为关键的意义，因为其中存在各种数据路径优化，特别是那些应用于系统总线的部分。 此参数可用于控制时钟时滞，确保读取和写入 PSoC 模块寄存器值时进行正确操作。 此参数的正确值应根据下表决定： .

[+] Feedback

时钟同步值	使用说明
同步到 SysClk	此设置值用于任何由 24 MHz (SysClk) 经过 2 分频或更多分频所衍生出来的时钟源。 这样的时钟源包括 VC1、VC2、VC3 （在 VC3 由 SysClk 驱动时）、32KHz 和以 SysClk 作为时钟源的数字 PSoC 模块。 外部生成的时钟源也应使用此值来确保执行正确的同步操作。
Sync 到 SysClk*2	此设置可以适用于任何基于 48 MHz (SysClk*2) 的时钟，除非产生的频率为 48 MHz （换句话说，所有分频器的乘积为 1 时）。
使用 SysClk Direct	需要 24 MHz (SysClk/1) 的时钟时使用。 此项选择并不真正执行同步，但提供了对系统时钟本身的低时滞访问方式。 如果选择此项，则此选项将覆盖之前 “ 时钟 ”(Clock) 参数的设置。 在所有分频器组合起来最终生成了 24 MHz 的输出时，一定要使用此项，而不使用 VC1、VC2、VC3 或数字模块。
Unsynchronized	选中 48 MHz (SysClk*2) 输入时使用。 在需要未同步输入时使用。 一般来说，只有在中断生成是计数器的唯一应用时才推荐使用此选项。

数据时钟输出

数据时钟输出信号是一种对应于时钟输入 8 分频的时钟信号。 该时钟仅在数据位发送过程中有效，在其他所有时间内处于高电平。 时钟的上升沿恰好是数据已经稳定并且可以被采样的时刻。 数据时钟输出信号可用于实现数据校验功能，例如循环冗余校验。

中断产生控制 (Interrupt Generation Control)

以下两项参数 InterruptAPI 和 IntDispatchMode 只能在 PSoC Designer 内将 **“ 中断产生控制 ”(Interrupt generation control)** 复选框选中后进行访问。 此复选框位于 **“ 项目 ” > “ 设置 ” > “ 芯片编辑器 ” 之下。.**

InterruptAPI

InterruptAPI 参数允许有条件地生成一个用户模块的中断处理程序和中断矢量表入口。 选择 “ 启用 ”(Enable) 以生成中断处理程序和中断矢量表条目。 选择 “ 禁用 ”(Disable) 以取消生成中断处理程序和中断矢量表条目。 如果要使用接收命令缓冲区，则 InterruptAPI 参数应当设置为 “ 启用 ”(Enable)。 在那些拥有多个程序层而且有多个程序层使用同一模块资源的项目中，特别推荐要正确地选择是否要生成中断 API。 仅在必要时选择生成中断 API，这样可以避免生成中断调度代码，从而减少开销。

IntDispatchMode

IntDispatchMode 参数用于指定如何处理中断请求（当处于同一模块的多个用户模块在不同的程序层共享该中断时）。 选择 “ActiveStatus” 会导致固件在为共享的中断请求提供服务之前测试哪一个程序层正处于活动状态。 这项测试发生在每次请求共享中断的时候。 这样会导致延迟增加并且会产生一个服务于共享中断请求的非确定性的程序，但并不要求任何 RAM 资源。 选择 “OffsetPreCalc” 参数会导致固件只在最初已经有一个程序层运行时计算共享中断请求的来源。 这项计算减少了中断延迟会产生一个服务于共享中断请求的确定性程序，但其代价是 1 个字节的 RAM 资源。

应用程序编程接口

应用程序编程接口 (API) 子程序作为用户模块的一部分提供，使设计人员能够在较高的层级处理模块。本部分具体说明了每个函数对应的接口以及 “include” 文件所提供的相关常量。

Note 在这里，如同所有用户模块 API 中的一样，A 和 X 寄存器的值可能因调用 API 函数而更改。 发起调用 API 的函数有责任在调用之前保留 A 和 X 的值 （如果调用后需要再次用到它们）。 选择这种 “ 寄存器易变 ” 策略是为了提高效率，并且自从 PsoC Designer 的 1.0 版本起使用。 C 编

[+] Feedback

译器自动遵循此要求。 汇编语言程序员也必须确保其代码遵守这一策略。 虽然有些用户模块的 API 函数可能保留 A 和 X 不变，但并不保证在未来也将如此。

API 子程序允许对 TX8 用户模块进行编程控制。 下表列出了 TX8 提供的底层和高层 API 函数。

Table 2. 底层 TX8 API

函数	说明
void TX8_Start(BYTE bParity)	启用用户模块并设置奇偶校验。
void TX8_Stop(void)	禁用用户模块。
void TX8_EnableInt(void)	启用 TX 中断。
void TX8_DisableInt(void)	禁用 TX 中断。
void TX8_SetTxIntMode(BYTE bTxIntMode)	设置 TX 中断的源。
void TX8_SendData(BYTE bTxData)	在不检查 TX 状态的情况下发送字节。
BYTE TX8_bReadTxStatus(void)	返回 TX 状态寄存器的状态。

Table 3. 高层 TX8 API

函数	说明
void TX8_PutString(char * szStr)	从 TX8 端口向外发送以空字符为结尾的字符串。
void TX8_CPutString(const char * azStr)	从 TX8 端口向外发送以空字符为结尾的常数 (ROM) 字符串。
void TX8_PutChar(char bData)	在 TX 寄存器为空时发送字符至 TX8 端口。 在 TX 数据寄存器可以被写入 （并且没有数据过速错误）之前，此函数将不会返回。
void TX8_Write(char * aStr, BYTE bCnt)	从 aStr 数组向 TX8 端口发送 bCnt 字节。
void TX8_CWrite(const char * aStr, int iCnt)	从常数 aStr 数组向 TX8 端口发送 iCnt 字节。
void TX8_PutSHexByte(BYTE bValue)	发送 bValue 的 2 字符 16 进制表示形式的数据至 TX8 端口。
void TX8_PutSHexInt(int iValue)	发送 iValue 的 4 字符 16 进制表示形式的数据至 TX8 端口。
void TX8_PutCRLF(void)	发送回车符 (0x0D) 和换行符 (0x0A) 至 TX8 端口。

TX8_Start

说明：

通过设置控制寄存器的 Tx 使能位，设置 TX8 发送器的奇偶校验并启用 TX8 模块。 一旦启用，将会在写入缓冲区寄存器之后在下个位时钟开始传输。

C 原型：

```
void  TX8_Start(BYTE bParitySetting)
```

汇编：

```
mov   A, TX8_PARITY_NONE
lcall  TX8_Start
```


[+] Feedback

参数：

bParitySetting： 一个用于指定发送奇偶校验的字节。 下表给出了在 C 语言和汇编语言中提供的符号名及其相关值。

符号名	值
TX8_PARITY_NONE	0x00
TX8_PARITY_EVEN	0x02
TX8_PARITY_ODD	0x06

返回值：

无

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 对于大型存储器模型（CY8C27x66、CY8C29xxx 和 CY8CLED16）中的所有 RAM 页指针寄存器，也是如此。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

TX8_Stop

说明：

通过清除控制寄存器 Tx 使能位禁用 TX8 模块。

C 原型：

```
void  TX8_Stop(void)
```

汇编：

```
lcall  TX8_Stop
```

参数：

无

返回值：

无

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 对于大型存储器模型（CY8C27x66、CY8C29xxx 和 CY8CLED16）中的所有 RAM 页指针寄存器，也是如此。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

TX8_EnableInt

说明：

通过在数字 PSoC 模块中断屏蔽寄存器中设置合适的使能位，在 “Tx 缓冲区空 ” 条件下启用 TX8 中断。 TX8 的放置位置决定了具体的中断矢量和优先级。 详细信息请参见 PSoC 数据表选取的特定部分。

C 原型：

```
void  TX8_EnableInt(void)
```


[+] Feedback

汇编：

```
lcall  TX8_EnableInt
```

参数：

无

返回值：

无

副作用：

如果某个中断正待处理且调用了该 API，则将立刻触发该中断。 此调用应当在调用 Start() 之前执行。 寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 对于大型存储器模型（CY8C27x66、CY8C29xxx 和 CY8CLED16）中的所有 RAM 页指针寄存器，也是如此。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

TX8_DisableInt

说明：

通过在数字 PSoC 模块中断屏蔽寄存器中清除合适的使能位，在 “Tx 缓冲区空 ” 条件下禁用 TX8 中断。

C 原型：

```
void  TX8_DisableInt(void)
```

汇编：

```
lcall  TX8_DisableInt
```

参数：

无

返回值：

无

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 对于大型存储器模型（CY8C27x66、CY8C29xxx 和 CY8CLED16）中的所有 RAM 页指针寄存器，也是如此。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

TX8_SetTxIntMode

说明：

设置中断的源。

C 原型：

```
void  TX8_SetTxIntMode(BYTE bTxIntMode)
```

汇编：

```
lcall  TX8_SetTxIntMode
```

参数：

bTxIntMode: 用于指定中断模式的 1 个字节。 下表给出了 C 语言和汇编语言中提供的符号名称及其相关数值。

[+] Feedback

TX 中断模式	值
TX8_INT_MODE_TX_REG_EMPTY	0x00
TX8_INT_MODE_TX_COMPLETE	0x01

返回值：

无

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 对于大型存储器模型（CY8C27x66、CY8C29xxx 和 CY8CLED16）中的所有 RAM 页指针寄存器，也是如此。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

TX8_SendData

说明：

通过将指定用于发送的数据加载到缓冲区寄存器来启动数据发送操作。 应对控制寄存器内的 “TX 完成 ” 状态位进行监控，以确保发送操作已经启动。

C 原型：

```
void  TX8_SendData(BYTE bTxData)
```

汇编：

```
mov   A, bTxData
lcall  TX8_SendData
```

参数：

bTxData： 要加载到 TX 缓冲区寄存器的数据。 在累加器内传递。

返回值：

无

副作用：

如果设置了 “Tx 缓冲区为空 ” 条件下的中断，则在 bTxData 从缓冲区寄存器传输到移位寄存器后会发生中断。 寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 对于大型存储器模型 （CY8C27x66、CY8C29xxx 和 CY8CLED16）中的所有 RAM 页指针寄存器，也是如此。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

TX8_bReadTxStatus

说明：

返回控制寄存器的内容。

C 原型：

```
BYTE TX8_bReadTxStatus(void)
```

汇编：

```
lcall TX8_bReadTxStatus
and  A, TX8_TX_COMPLETE
jnz  TxIsComplete
```


[+] Feedback

参数：

无

返回：

返回状态字节读取。 利用定义的掩码来测试特定的状态条件。 请注意，这些掩码可以通过 “ 或 ” 运算结合起来同时检测多个条件。

RX 状态掩码	值
TX8_TX_COMPLETE	0x20
TX8_TX_BUFFER_EMPTY	0x10

副作用：

状态位在执行读取后会被清除。 寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 对于大型存储器模型（CY8C27x66、CY8C29xxx 和 CY8CLED16）中的所有 RAM 页指针寄存器，也是如此。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

TX8_PutString

说明：

向 TX8 端口发送以空字符结尾的 (RAM) 字符串。

C 原型：

```
void TX8_PutString(char * szRamString)
```

汇编程序：

```
mov  A,>szRamString     ; Load MSB part of pointer to RAM based null
                        ; terminated string.
mov  X,<szRamString     ; Load LSB part of pointer to RAM based null
                        ; terminated string.
lcall TX8_PutString      ; Call function to send string out TX8 port
```

参数：

char * aRamString: 指向将要发送至 TX8 端口的字符串的指针。 MSB 在累加器内传递，而 LSB 在 X 寄存器内传递。

返回值：

无

副作用：

程序执行保留在此函数内，直到最后 1 个字符加载到 TX8 发送缓冲区之内。 寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 对于大型存储器模型（CY8C27x66、CY8C29xxx 和 CY8CLED16）中的所有 RAM 页指针寄存器，也是如此。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。 目前，仅修改 IDX_PP 页面指针寄存器。

TX8_CPutString

说明：

从 TX8 端口向外发送空字符结尾的常数 (ROM) 字符串。

[+] Feedback

C 原型：

```
void TX8_CPutString(const char * szROMString)
```

汇编程序：

```
mov  A,>szRomString   ; Load MSB part of pointer to ROM based null
                      ; terminated string.
mov  X,<szRomString   ; Load LSB part of pointer to ROM based null
                      ; terminated string.
lcall TX8_CPutString   ; Call function to send constant string out
                      ; TX8 port
```

参数：

const char * szROMString 指向将要发送至 TX8 端口的字符串的指针。 字符串指针的 MSB 在累加器内传递，指针的 LSB 在 X 寄存器内传递。

返回值：

无

副作用：

程序执行停留在此函数内，直到最后 1 个字符加载到 TX8 发送缓冲区之内。 寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 对于大型存储器模型（CY8C27x66、CY8C29xxx 和 CY8CLED16）中的所有 RAM 页指针寄存器，也是如此。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

TX8_PutChar

说明：

在端口缓冲区为空时，写入单个字符至 TX8 端口。

C 原型：

```
void TX8_PutChar(CHAR cData)
```

汇编程序：

```
mov  A,0x33            ; Load ASCII character “3” in A
lcall TX8_PutChar       ; Call function to send single character to
                      ; TX8 port.
```

参数：

CHAR cData： 将要发送至 TX8 端口的字符。 数据在累加器内传递。

返回值：

无

副作用：

在数据可以写入至 TX8 缓冲区之前，程序执行将停留在此函数之内。 寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 对于大型存储器模型 （CY8C27x66、CY8C29xxx 和 CY8CLED16）中的所有 RAM 页指针寄存器，也是如此。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

TX8_Write

说明：

向 TX8 端口发送 “n” 个字符 (RAM)。

[+] Feedback

C 原型：

```
void TX8_Write(char * szRamString, BYTE bCount)
```

汇编程序：

```
mov   A,20              ; Load string/array count
push  A
mov   A,>szRamString    ; Load MSB part of pointer to RAM string
push  A
mov   A,<szRamString    ; Load LSB part of pointer to RAM string
push  A
mov   X,SP              ; Set X register to point to variables
dec   X
lcall  TX8_Write         ; Make call to function
add   SP,253            ; Reset stack pointer to original position
```

参数：

CHAR * szRamString： 指向将要发送至 TX8 端口的字符串的指针。

BYTE bCount： 将要发送至 TX8 的字符数。

返回值：

无

副作用：

程序执行保留在此函数内，直到最后 1 个字符加载到 TX8 缓冲区之内。 寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 对于大型存储器模型 （CY8C27x66、CY8C29xxx 和 CY8CLED16）中的所有 RAM 页指针寄存器，也是如此。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。 目前，仅修改 IDX_PP 页面指针寄存器。

TX8_CWrite

说明：

向 TX8 端口发送 n 个字符 (ROM)。

C 原型：

```
void TX8_CWrite(const char * szRomString, INT iCount)
```

汇编程序：

```
mov   A,0               ; Load MSB of string/array count
push  A
mov   A,20              ; Load LSB of string/array count
push  A
mov   A,>szRomString    ; Load MSB part of pointer to ROM string
push  A
mov   A,<szRomString    ; Load LSB part of pointer to ROM string
push  A
mov   X,SP              ; Set X register to point to variables
dec   X
lcall  TX8_CWrite         ; Make call to function
add   SP,252            ; Reset stack pointer to original position
```

参数：

const char * szRomString： 指向将要发送至 TX8 端口的字符串的指针。

int iCount： 将要发送至 TX8 端口的字符数。

[+] Feedback

返回值：

无

副作用：

程序执行保留在此函数内，直到最后 1 个字符加载到 TX8 缓冲区之内。 寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 对于大型存储器模型 （CY8C27x66、CY8C29xxx 和 CY8CLED16）中的所有 RAM 页指针寄存器，也是如此。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

TX8_PutSHexByte

说明：

发送 2 个字节的 ASCII 16 进制表示形式的数据至 TX8 端口。

C 原型：

```
void TX8_PutSHexByte(BYTE bData)
```

汇编程序：

```
mov  A,0x33               ; Load data to be sent to TX8
lcall TX8_PutSHexByte      ; Call function to output hex
                          ; representation of data.  The output
                          ; for this value would be “33”.
```

参数：

BYTE bData： 将要转换为 ASCII 字符串的字节。

返回值：

无

副作用：

程序流停留在此函数内，直到最后一个字符加载到 TX8 缓冲区之内。 寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 对于大型存储器模型 （CY8C27x66、CY8C29xxx 和 CY8CLED16）中的所有 RAM 页指针寄存器，也是如此。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

TX8_PutSHexInt

说明：

发送 4 个字节的 ASCII 16 进制表示形式的数据至 TX8 端口。

C 原型：

```
void TX8_PutSHexInt(INT iData)
```

汇编程序：

```
mov  A,0x34           ; Load LSB in A
mov  X,0x12           ; Load MSB in X

lcall TX8_PutSHexInt   ; Call function to output hex
                      ; representation of data.  The output
                      ; for this value would be “1234”.
```

参数：

int iData： 将要转换为 ASCII 字符串的整数。

[+] Feedback

返回值：

无

副作用：

程序流停留在此函数内，直到最后一个字符加载到 TX8 缓冲区之内。 寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 对于大型存储器模型 （CY8C27x66、CY8C29xxx 和 CY8CLED16）中的所有 RAM 页指针寄存器，也是如此。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

TX8_PutCRLF

说明：

从 TX8 端口向外发送回车符 (0x0D) 和换行符 (0x0A) 的函数。

C 原型：

```
void TX8_PutCRLF(void)
```

汇编程序：

```
lcall TX8_PutCRLF          ; Send a carriage return and line feed out TX
```

参数：

无

返回值：

无

副作用：

程序执行会停留在此函数中，直到所有字符均已写入 TX8 缓冲区。 寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 对于大型存储器模型 （CY8C27x66、CY8C29xxx 和 CY8CLED16）中的所有 RAM 页指针寄存器，也是如此。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

[+] Feedback

固件源代码示例

下面的代码是一个函数示例，使用 TX8 模块传送常驻内存的以零结尾的字符串。 用汇编语言编写的代码示例：

```
;*****************************************************************
;  Name:   TxZeroTerminatedRamString
;
;  Description:
;    Transmits a zero terminated RAM-based string.
;
;  Parameters:
;    BYTE * pbStrPtr - pointer to the string - passed in
;    the X register.
;
;  Return:
;    None
;
;  Note:
;     TX8_Start should be called prior to calling this function.
;
;*****************************************************************
include "psocapi.inc"
export TxZeroTerminatedRamString

TxZeroTerminatedRamString:
.TxNextByte:
; check to see if end has been reached.
    mov   A, [X]
    jz    .AllDone

; transmit the next data byte
    call  TX8_SendData

.WaitForTxStart:
    ; wait for data to start transmitting
    call  TX8_bReadTxStatus
    and   A, TX8_TX_BUFFER_EMPTY
    jz    .WaitForTxStart
    ; increment the pointer to the next byte in the string
    inc   X
    ; data byte transmitted - send the next one
    jmp   .TxNextByte

.AllDone:
    ; string completely transmitted!
    ret
```


[+] Feedback

以 C 语言编写的相同代码：

```
//
// TX8_Start() should be called prior to calling this function.
//
#include "psocapi.h"
#include "m8c.h"


void TxZeroTerminatedRamString( BYTE *pbStrPtr )
{
    /* check for the end condition, before sending the next byte */
    while( *pbStrPtr != 0 )
    {
        /* send the next byte */
        TX8_SendData( *pbStrPtr );

        /* Wait for the data to start transmitting */
        while( !( TX8_bReadTxStatus() & TX8_TX_BUFFER_EMPTY ) );

        pbStrPtr++;
    }
}
```

配置寄存器

用于配置 TX8 用户模块的 A 型数字通信 PSoC 模块寄存器描述如下。 仅解释参数化符号。

Table 4. TX 模块、寄存器： 函数

位	7	6	5	4	3	2	1	0
值	0	0	0	0	1	1	0	1

此寄存器定义将此 “A” 型数字通信模块配置为 TX8 用户模块。

Table 5. TX 模块、寄存器： 输入

位	7	6	5	4	3	2	1	0
值	0	0	0	0	时钟			

时钟 (Clock) 用于选择驱动发送器时序的时钟。 在参数选择过程中使用器件编辑器进行设置。

Table 6. TX 模块、寄存器： 输出

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	TX 输出总线		

“ 输出启用和选择 ”(TX Output Enable and Select) 用于指定 TX8 的输出。 在参数选择过程中使用器件编辑器进行设置。

[+] Feedback

Table 7. TX 模块、数据移位寄存器： DR0

位	7	6	5	4	3	2	1	0
值	TX8 移位寄存器							

TX8 移位寄存器由 TX8 状态机硬件控制，用于移位内容以及发送最低有效位。 数据由 TX8 状态机硬件从 DR1 数据寄存器加载到此寄存器内。

Table 8. TX 模块、数据缓冲区寄存器： DR1

位	7	6	5	4	3	2	1	0
值	TX8 缓冲区寄存器							

TX 缓冲区寄存器用于发送已经由用户模块 API 写入到此寄存器的数据。 加载到该寄存器内的数据通过 TX 状态机传输至 TX 移位寄存器。

Table 9. TX 模块、数据寄存器： DR2

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

未使用该寄存器。

Table 10. TX 模块、控制 / 状态寄存器： CR0

位	7	6	5	4	3	2	1	0
值	Rsvd	Rsvd	TX 完成	TX 寄存器空	Rsvd	奇偶校验类型	奇偶校验启用	TX 启用

“TX 完成 ”(Tx Complete) 是用于指示某个数据字节是否正在传送过程中的标志。 该位在读取寄存器时复位。 可以通过一个 API 函数进行访问。 “TX 寄存器空 ”(Tx Reg Empty) 是用于指示缓冲区寄存器为空的标志。 可以通过一个 API 函数进行访问。 “ 奇偶校验类型 ”(Parity Type) 表示将要计算的奇偶校验的类型。 如果奇偶校验未启用则无需关注此位。可以通过一个 API 函数或者在器件编辑器中进行设置。 “ 奇偶校验启用 ”(Parity Enable) 用于启用或禁用数据字节的奇偶校验位的计算和发送。 通过设置奇偶校验类型位选择奇偶校验。 可以通过一个 API 函数或者在器件编辑器中进行设置。 “TX 启用 ”(Tx Enable) 用于启用或禁用 TX 发送器。 可以通过一个 API 函数进行设置。

[+] Feedback

版本历史记录

版本	创作者	说明
3.3	TDU	更新了时钟说明，内容包括： 在模块中使用外部数字时钟时，为达到最高精度并使用睡眠操作应关闭行输入同步。
3.4	DHA	添加了 DRC，以通知用户除非使用了外部晶振，否则不要使用 32 kHz 选项。
3.50	DHA	添加了对 CY8C21x12 器件的支持。

Note PSoC Designer 5.1 在所有用户模块数据表中引入了版本历史。 此部分记录了当前用户模块版本和以前用户模块版本之间区别的高级描述。

Copyright © 2001-2010 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer™ and Programmable System-on-Chip™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.