8-Bit 串行接收器数据表 RX8 V 3.50

Copyright © 2002-2011 Cypress Semiconductor Corporation. All Rights Reserved.

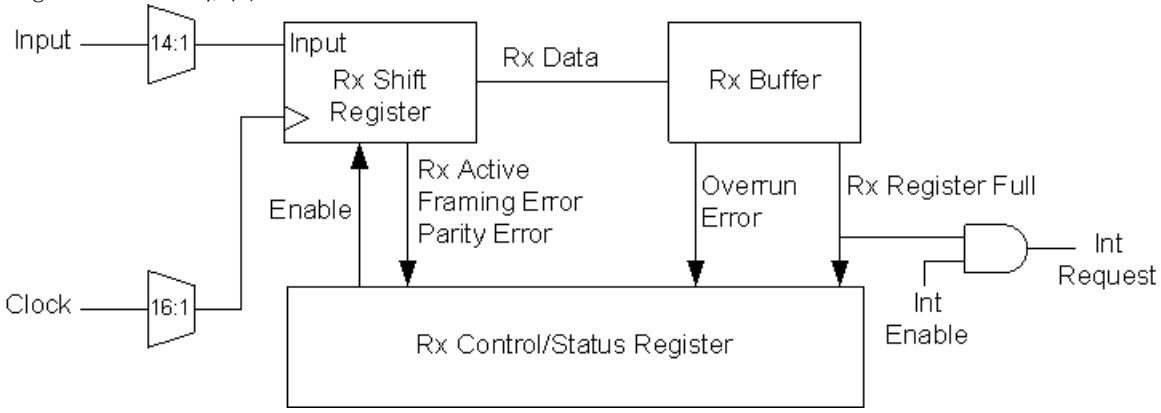
资源	PSoC® 模块			API 内存 （字节）		引脚
	数字	模拟 CT	模拟 SC	闪存	RAM	
CY8C29/27/24/22/21xxx、CY8C23x33、CYWUSB6953、CY8CLED02/04/08/16、CY8CLED0xD、CY8CLED0xG、CY8CTST110、CY8CTMG110、CY8CTST120、CY8CTMG120、CY8CTMA120、CY8C21x45、CY8C22x45、CY8CTST300、CY8CTMG300、CY8CTMA300、CY8CTMA301、CY8CTMA301D、CY8C28x45、CY8CPLC20、CY8CLED16P01、CY8C28xxx、CY8C21x12						
RxCmdBuf 已启用	1	0	0	273	超过 3 个缓冲区	1
RxCmdBuf 已禁用	1	0	0	77	0	1
CY8C26/25xxx						
RxCmdBuf 已启用	1	0	0	283	超过 3 个缓冲区	1
RxCmdBuf 已禁用	1	0	0	87	0	1

特性与概述

- 突发速率高达 6 Mbits/秒
- 符合成帧的 RS-232 数据格式包含起始位、可选奇偶校验位和停止位
- 串行数据格式符合偶校验、奇校验或无奇偶校验
- 接收寄存器满条件下的可选中断
- 自动成帧、过速和奇偶校验错误检测

RX8 用户模块是符合 RS-232 数据格式的 8-bit 串行接收器，支持可编程时钟和可选中断或轮询控制操作。 所接收的数据格式包含起始位、可选奇偶校验位和尾停止位。 接收器固件用于初始化器件、读取接收到的字节以及检测错误条件。

Figure 1. RX8 框图



功能说明

RX8 用户模块实现了一种串行接收器。 RX8 可以在 PSoC Designer 器件编辑器内映射到一个名为 “RX” 的 PSoC 模块。 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。

此控制寄存器采用 RX8 用户模块固件应用程序编程接口 (API) 子程序进行了初始化和配置。 RX8 的初始化包括设置奇偶校验、有选择性地在 Rx 寄存器满条件下允许中断，以及随后启用接收器。

在 RX8 输入上检测到起始位时，将会启动和同步一个 8 分频位时钟，用于对所接收的位的中间部分数据进行采样。 在下一个 8 位时钟的上升沿上，对输入数据进行采样并移位到移位寄存器内。 如果启用了奇偶校验，则下一个位时钟将对奇偶校验位进行采样。 在下一个时钟上，对停止位采样会导致将已接收到的数值字节传输至缓冲区寄存器，并触发下列事件中的一个或多个：

- 控制寄存器内的 “Rx 寄存器满 ”(Rx Register Full) 位被置位，并且如果 RX8 中断已经启用，则会触发相关联的中断。
- 如果在数据流的预期位位置未检测到停止位，则将置位控制寄存器内的 “ 成帧错误 ”(Framing Error) 位。
- 如果缓冲区寄存器没有在当前接收数据的停止位之前被读取，则将置位控制寄存器内的 “ 过速错误 ”(Overrun Error) 位。
- 如果检测到奇偶校验错误，则将置位控制寄存器内的 “ 奇偶校验错误 ”(Parity Error) 位。

对于已完全接收数据字节的轮询检测来说，应当对控制寄存器内的 “Rx 寄存器满 ”(Rx Register Full) 位进行监测。 在下一个字节完全接收之前，必须从缓冲器寄存器内将数据读出，以防止出现过速错误条件。

高层 API

高层 API 能够在基本函数之上添加额外的固件，以提供命令和字符串级的函数而不是字符级的函数。 器件编辑器允许用户设置接收器命令缓冲区的大小、命令终止符、参数分隔符以及低于哪一数值时接收器应忽略字符。

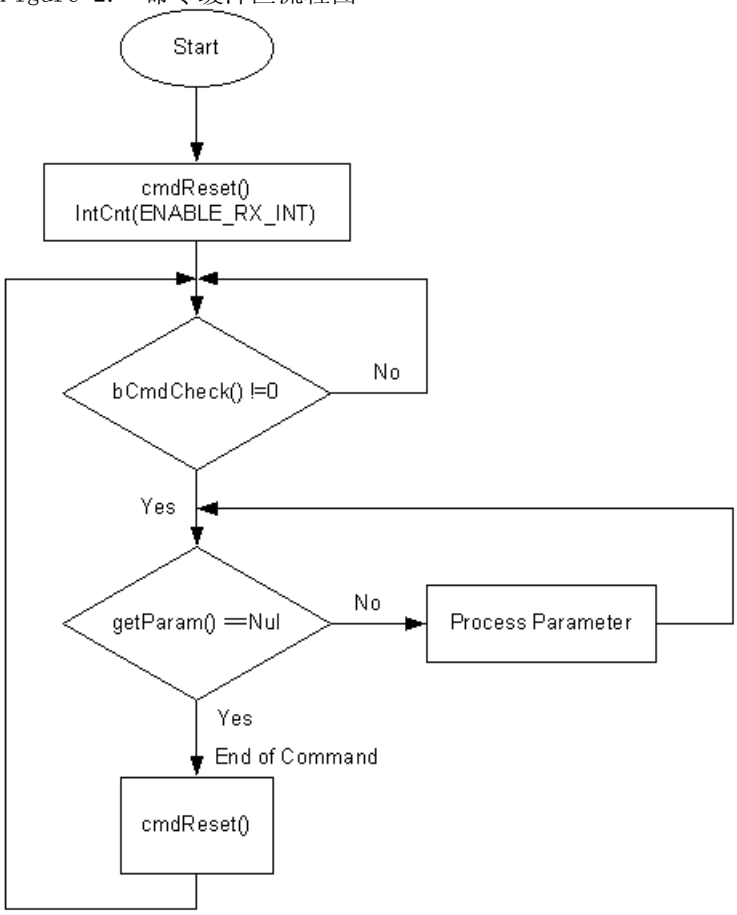
要使用这些高层接收器函数，可进入 “ 器件编辑器 ”(Device Editor) 窗口，选择 “UART”，再为 “RxCmdBuffer” 参数选择 “ 使能 ”(Enable) 选项。 然后，选择一个大小足以容纳最大命令字节数加 1 的 “RxBufferSize” 参数。 选择命令终止符字符 “CommandTerminator”。此参数最经常被设置为回车符 (13) 或换行符 (10)。 如果命令包含两个或更多参数，则选择参数分隔符 “Param_Delimiter.”。常见命令分隔符通常为空格 (32) 或逗号 (44) 字符。 也可以忽略低于选定数值的控制字符。 绝大多数控制字符均处于 0 至 31 的范围内。 设置 “IgnoreCharsBelow” 为 32 以忽略这些字符。 如果所有字符都有效，则将此参数设置为 “1”。 选定作为命令终止器 (CommandTerminator) 的字符不受 “IgnoreCharsBelow” 选项的影响。 以下流程图表明了命令缓冲区函数基本操作的正确顺序。

命令缓冲区用于在 UART RX 中断服务子程序内收集缓冲区内的字符，直至收到命令终止符为止。 此时，将设置一个标志作为命令缓冲区已准备好被读取的信号。 接收缓冲区可以通过读取阵列 INSTANCE_NAME_aRxBuffer 或通过使用 szGetParam() 或 szGetRestOfParams() 函数直接读取。 如果所接收到的字符多于 buffer_size 减 1 个，则随后的字符将被忽略。

[+] Feedback

命令缓冲区将采集字符，直到缓冲区满或检测到命令终止符为止。 在满足了这两个条件中任一条件之后，所接收到的任何字符均将被忽略，直到执行 CmdReset 命令为止。 一旦执行了 CmdReset 命令，RX ISR 固件将再次开始采集字符。

Figure 2. 命令缓冲区流程图



时序

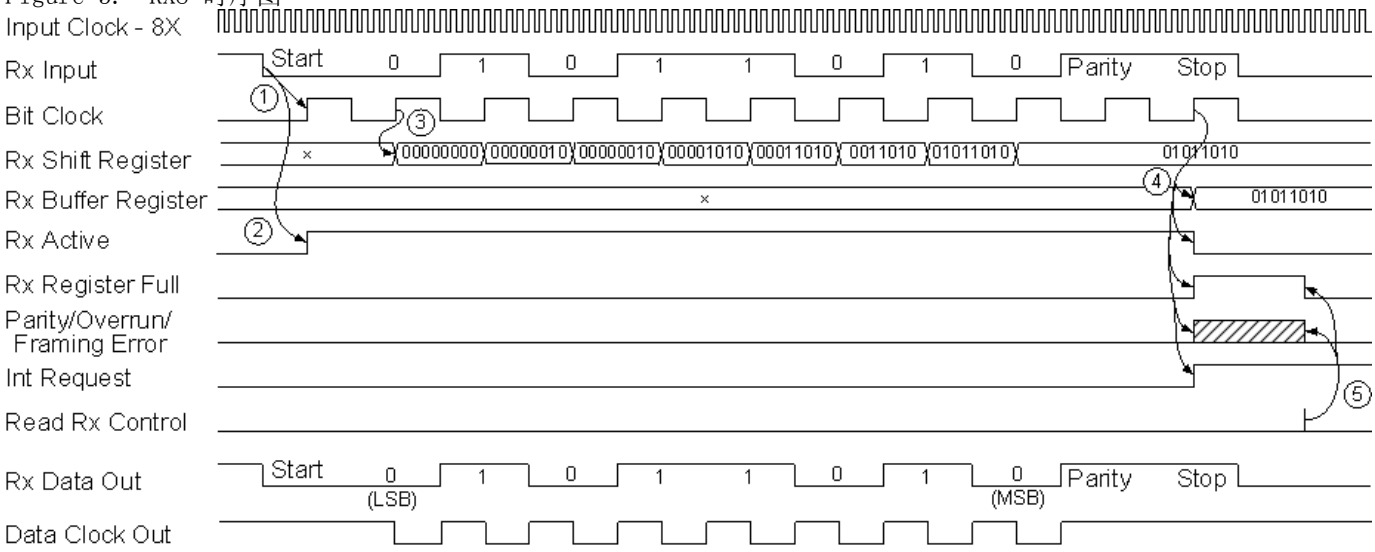
每一个接收到的位都会在一个生成的 8 分频位时钟的上升沿上进行采样，此 8 分频位时钟已与起始位的中间部分同步。

如果已启用，RX8 中断会在每接收到一个字节时发生一次。 启用和禁用中断由 API 控制。

[+] Feedback

下面的 RX8 时序图说明了 RX8 用户模块的操作。

Figure 3. RX8 时序图



图注

1. 检测到起始位，导致了 8 分频位时钟的生成和同步。 此位时钟的上升沿用于对输入位流进行采样。
2. 检测到起始位导致 Rx 活动状态位在 Rx 控制 / 状态寄存器中被置位。
3. 从下一个位时钟上升沿开始以及之后的 8 个时钟，对输入进行采样并移位至 Rx 移位寄存器中。
4. 检测到停止位后，数据会传输至 Rx 缓冲区中，将清空 “Rx 活动 ”(Rx Active)、置位 “Rx 寄存器满 ”(Rx Register Full)、计算奇偶校验 / 过速 / 成帧错误，并且将触发中断请求 （如已启用）。
5. 读取 Rx 控制寄存器可以将状态数据置于数据总线上，并清除所有状态位。 当检测到奇偶校验错误或溢出错误时，固件不需要执行任何操作。 如果检测到成帧错误，成帧器会立即开始寻找下一个数据字节。 假如系统内的 RX 输入有可能长时间卡在逻辑 0 处，则应停止接收器，并且持续查询该信号线直至其返回至逻辑 1 状态，然后才可以再次启用接收器。 如果 RX 输入卡在逻辑 0 处，这样处理可以避免当输入信号线为逻辑 0 时重复检测到 “ 虚假 ” 起始位，从而导致成帧错误 （在应该检测到起始位的位置检测到逻辑 0）。

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。

直流和交流电气特性

Table 1. RX8 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}	最高接收频率		6	Mbits/ 秒

放置

RX8 用户模块可以置于任何数字通信模块内。

[+] Feedback

参数和资源

时钟

RX8 的时钟来自于 16 个可能的源之一。 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、分频时钟中的任一个 （24V1 或 24V2）或者另一个 PSoC 模块输出都可指定为 RX8 时钟输入。

时钟频率必须设置为所需位接收频率的 8 倍。

输入

通常，输入经过所需总线选项到异步数据源。 使用全局总线时，输入可以连接到外部引脚之一。

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。 此参数的正确数值应当由下表确定。

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。

RX 输出

此参数允许输入信号路由至行总线之一。 此信号与数据时钟输出 (Data Clock Out) 信号一起可以被用来实现数据验证功能，例如循环冗余校验。

数据时钟输出

此参数允许 SPI 模式 3 的位时钟路由至行总线之一。 位时钟是指 8 分频时钟输入。 数据时钟输出信号的上升沿恰好是数据已经稳定并且可以被采样的时刻。 此信号与 RX 输出信号一起可以被用来实现数据验证功能，例如循环冗余校验。

RxCmdBuffer

此参数用于启用接收命令缓冲区和用于命令处理的固件。 必须启用 UART RX 中断后，命令缓冲区才能运行。

[+] Feedback

RxBufferSize

此参数用于确定为接收缓冲区保留的 RAM 的大小。 可以接收的最大命令长度是所选定的缓冲区大小减 1，因为字符串必须以空字符作为结尾。 此参数只在 RxCmdBuffer 参数启用后以及 UART RX 中断启用后有效。

InvertInput

此参数可以让用户反转 RX 输入信号。

CommandTerminator

此参数用于选择作为命令结束信号的字符。 在接收到此字符时，将对一个标志置位，表示已经接收到了一个完整的命令。 此标志置位后，直到调用 cmdReset() 函数之前，都不会再接收其他字符。

Param_Delimiter

此参数选择用于在命令接收缓冲区内分隔命令和参数的字符。 例如，如果 Param_Delimiter 设置为空格字符 (32)，则每个由空格分隔开的子字符串都将成为一个参数。 假设字符串为 “cmd foo bar c”，则参数将是 “cmd”、“foo”、“bar” 和 “c”。 每次调用 szGetParam() 时均会返回一个指向下一个子字符串的指针，这个子字符串是一个按照从左至右的次序放置的，以空字符结尾的字符串。

IgnoreCharsBelow

此参数将使接收缓冲区忽略低于某一设定数值的字符。 这些字符将被接收，但不会添加到接收缓冲区内。 此参数只在 RxCmdBuffer 参数启用后以及 UART RX 中断处于活动状态下有效。

中断产生控制 (Interrupt Generation Control)

当选中 PSoC Designer 中的 **“ 启用中断产生控制 ”** 复选框时，有两个附加参数将变为可用。 此复选框位于 **“ 项目 ” > “ 设置 ” > “ 芯片编辑器 ” 之下。**. 当拥有多个程序层而且多个用户模块在不同的程序层共享中断时，中断生成控制是非常重要的：

- 中断 API (Interrupt API)
- IntDispatchMode

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。

[+] Feedback

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 RX8 用户模块进行编程控制。下表列出了底层和高层 RX8supplied API 函数。

Table 2. 底层 RX8 API

函数	说明
void RX8_Start(BYTE parity)	启用用户模块并设置奇偶校验。
void RX8_Stop(void)	禁用用户模块。
void RX8_EnableInt(void)	启用中断。
void RX8_DisableInt(void)	禁用中断。
BYTE RX8_bReadRxData(void)	在不检查字符状态是否有效的情况下，返回 RX 数据寄存器中的数据。
BYTE RX8_bReadRxStatus(void)	检查 RX 状态寄存器的状态。

Table 3. 高层 RX8 API

函数	说明
char RX8_cGetChar(void)	在有效数据可用时，从 RX 数据寄存器内返回字符。在接收到字符之前，此函数将不会返回。
char RX8_cReadChar(void)	立即读取 RX 数据寄存器。如果没有有效的数据，则返回 0，否则返回 1 至 255 之间的 ASCII 字符。
int RX8_ iReadChar(void)	立即读取 RX 数据寄存器。如果没有可用数据或存在错误状况，则在最高有效位 (MSB) 内返回一个错误状态。所接收到的字符在最低有效位 (LSB) 内返回。
void RX8_CmdReset(void)	复位 RX 命令缓冲区。
BYTE RX8_bCmdCheck(void)	如果已经接收到命令终止符，则将返回一个非零值。
BYTE RX8_bCmdLength(void)	返回当前命令长度。
char * RX8_szGetParam(void)	返回指向 RX 缓冲区内下一个参数的指针。
char * RX8_szGetRestOfParams(void)	返回指向剩余参数字符串的指针。
BYTE RX8_bErrCheck(void)	返回命令缓冲区错误状态。

[+] Feedback

RX8_Start

说明：

通过设置控制寄存器的 RX 使能位，设置 RX8 接收器的奇偶校验并启用 RX8 模块。

C 原型：

```
void  RX8_Start(BYTE bParitySetting)
```

汇编：

```
mov   A, RX8_PARITY_NONE
lcall  RX8_Start
```

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
RX8_PARITY_NONE	0x00
RX8_PARITY_EVEN	0x02
RX8_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。 在大内存模式下（CY8C29xxx 和 CY8CLED16），所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

RX8_Stop

说明：

通过清除控制寄存器使能位禁用 RX8 模块。

C 原型：

```
void  RX8_Stop(void)
```

汇编：

```
lcall RX8_Stop
```

参数：

无

返回值：

无

[+] Feedback

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。 在大内存模式下（CY8C29xxx 和 CY8CLED16），所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

RX8_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，在接收寄存器满条件下启用 RX8 中断。

C 原型：

```
void  RX8_EnableInt(void)
```

汇编：

```
lcall  RX8_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。 在大内存模式下（CY8C29xxx 和 CY8CLED16），所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

RX8_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，在接收寄存器满条件下禁用 RX8 中断。

C 原型：

```
void  RX8_DisableInt(void)
```

汇编：

```
lcall  RX8_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。 在大内存模式下（CY8C29xxx 和 CY8CLED16），所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

[+] Feedback

RX8_bReadRxData

说明：

读取从缓冲区寄存器接收的数据字节。

C 原型：

```
BYTE  RX8_bReadRxData(void)
```

汇编：

```
lcall  RX8_bReadRxData
mov   [bRxData],A
```

参数：

无

返回值：

在累加器中返回接收到的数据。

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。 在大内存模式下（CY8C29xxx 和 CY8CLED16），所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

RX8_bReadRxStatus

说明：

读取并返回控制寄存器的状态位。

C 原型：

```
BYTE  RX8_bReadRxStatus(void)
```

汇编：

```
call  RX8_bReadRxStatus
mov   [bRxStatus],A
```

参数：

无

返回值：

返回状态字节读取。 使用下面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 ” 运算结合起来检测组合条件。

RX 状态掩码	值
RX8_RX_ACTIVE	0x10
RX8_RX_COMPLETE	0x08

[+] Feedback

RX 状态掩码	值
RX8_RX_PARITY_ERROR	0x80
RX8_RX_OVERRUN_ERROR	0x40
RX8_RX_FRAMING_ERROR	0x20
RX8_RX_ERROR	0xE0

副作用：

寄存器的读取会清除所有状态位。 应当在抛弃返回数值前谨慎检查所有适用的状态条件。 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。 在大内存模式下 （CY8C29xxx 和 CY8CLED16），所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

RX8_cGetChar

说明：

等待 RX8 内的有效字符并返回其数值。

C 原型：

```
CHAR RX8_cGetChar(void)
```

汇编程序：

```
lcall RX8_cGetChar          ; lcall function to print single character to
                            ; serial port.
mov  [CharBuffer,]A         ; Store retrieved character in buffer
```

参数：

无

返回值：

Char bData： 在累加器中返回从 RX8 中读取的字符。

副作用：

程序流停留在此函数内，直到接收到某个字符或遇到某个以前接收但未读取的字符。 在使用此函数时，应该禁用 RX8 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。 在大内存模式下 （CY8C29xxx 和 CY8CLED16），所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

RX8_cReadChar

说明：

如果数据不可用或存在错误状况，或者返回了零，则立即读取 RX8 端口；否则，读取并返回字符。

C 原型：

```
CHAR RX8_cReadChar(void)
```

汇编程序：

```
lcall RX8_cReadChar          ; lcall function to read a character
```


[+] Feedback

```
cmp   A,0x00                ; Check for error
jz    ProcessError          ; If error, Process the error condition
mov   [CharBuffer],A        ; Store retrieved character in buffer
```

参数：

无

返回值：

CHAR bData：从 RX8 端口读取的字符。1 至 255 之间的 ASCII 字符为有效字符。返回零则表示出现错误状况或数据不可用。

副作用：

函数只接受 1 至 255 之间的有效字符。检测到 0x00（null）字符即认为是一种错误状况。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。在大内存模式下（CY8C29xxx 和 CY8CLED16），所有 RAM 页面指针寄存器也会出现这种状况。在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

RX8_iReadChar

说明：

立即读取 RX8 端口，返回所接收到的字符和错误状况。

C 原型：

```
INT RX8_iReadChar(void)
```

汇编程序：

```
lcall RX8_iReadChar         ; lcall function to read a character
cmp   X,0x00                ; Check for error
jnz   ProcessError          ; If error, Process the error condition
mov   [CharBuffer]A         ; Store retrieved character in buffer
```

参数：

无

返回值：

unsigned int iData：MSB 包含状态，而 LSB 包含 UART RX 数据。如果 MSB 为非零，则发生一项错误。下表显示了可能在 MSB 内返回的错误代码。

错误标志	值	说明
RX8_RX_PARITY_ERROR	0x80	奇偶校验错误
RX8_RX_OVERRUN_ERROR	0x40	缓冲区过速错误
RX8_RX_FRAMING_ERROR	0x20	字符成帧错误
RX8_RX_NO_ERROR	0x0E	没有错误
RX8_RX_NO_DATA	0x01	没有可用数据

[+] Feedback

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。 在大内存模式下（CY8C29xxx 和 CY8CLED16），所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

RX8_CmdReset

说明：

复位命令缓冲区和标志。 这样可以让下一个命令的字符得到接收。

C 原型：

```
void RX8_CmdReset(void)
```

汇编程序：

```
lcall RX8_CmdReset          ; lcall function to reset command buffer.
```

参数：

无

返回值：

无

副作用：

复位 RX8 字符计数并清空接收缓冲区。 仍保留在缓冲区内的所有字符将会丢失。 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。 在大内存模式下 （CY8C29xxx 和 CY8CLED16），所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。 目前，仅修改 CUR_PP 页面指针寄存器。

RX8_bCmdCheck

说明：

检查是否已经接收到命令终止符。

C 原型：

```
BYTE RX8_bCmdCheck(void)
```

汇编程序：

```
lcall RX8_bCmdCheck       ; lcall function to get command complete status.
cmp  A,0x00               ; Check if command complete
jnz  ProcessCmd           ; Process command buffer
```

参数：

无

返回值：

如果已经接收到命令终止符，则将返回一个非零值。

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。 在大内存模式下（CY8C29xxx 和 CY8CLED16），所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。 目前，仅修改 CUR_PP 页面指针寄存器。

[+] Feedback

RX8_bCmdLength

说明：

返回命令缓冲区内的字符长度。 此命令将返回当前命令长度，无论是否已经接收到命令终止符。

C 原型：

```
BYTE RX8_bCmdLength(void)
```

汇编程序：

```
lcall RX8_bCmdLength      ; lcall function to get current command
                          ; Command length is returned in Accumulator
```

参数：

无

返回值：

接收缓冲区内当前字符串的长度。

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。 在大内存模式下（CY8C29xxx 和 CY8CLED16），所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。 目前，仅修改 CUR_PP 页面指针寄存器。

RX8_szGetParam

说明：

从接收缓冲区返回下一个参数，此参数由在器件编辑器内所设置的参数分隔符 (Param_Delimiter) 来分隔。 在所有参数均已经返回后，任何后续调用都将返回一个空指针 (zero)。

C 原型：

```
char * RX8_szGetParam(void)
```

汇编程序：

```
lcall RX8_szGetParam       ; lcall function to return pointer to the
                           ; next parameter.
                           ; Pointer is returned in A and X.
```

参数：

无

返回值：

char * strPtr: 指向参数字符串的指针。

副作用：

在每次调用 szGetParam 时都会修改接收缓冲区。 空字符放置在每个参数后。 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。 在大内存模式下 (CY8C29xxx 和 CY8CLED16)，所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。 目前修改了 CUR_PP 和 IDX_PP 页面指针寄存器。

[+] Feedback

RX8_szGetRestOfParams

说明：

返回一个指向接收缓冲区内尚未由 szGetParam 函数取走的剩余字符串的指针。 如果在 szGetParam 之前调用此函数，则返回一个指向整个接收缓冲区的指针。 如果到达了这个字符串的结尾，则返回一个空字符 (0x00)。

C 原型：

```
char * RX8_szGetRestOfParams(void)
```

汇编程序：

```
lcall RX8_szGetRestOfParams    ; lcall function to return pointer to the
                               ; remainder of the receive buffer.
                               ; Pointer is returned in A and X.
```

参数：

无

返回值：

char * strPtr： 指向接收字符串剩余部分的指针。

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。 在大内存模式下（CY8C29xxx 和 CY8CLED16），所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。 目前，仅修改 CUR_PP 页面指针寄存器。

RX8_bErrCheck

说明：

检查自从上一次调用 bErrCheck 函数后的命令错误。

C 原型：

```
BYTE RX8_bErrCheck(void)
```

汇编程序：

```
lcall RX8_bErrCheck             ; lcall function to return error status
cmp  A,0x00                     ; Check for error
jnz  ProcessError               ; If error, Process the error condition
```

参数：

无

返回值：

BYTE bErr： MSB 包含了自从上一次调用本函数以来所发生的任何错误状况的状态。

[+] Feedback

错误标志	值	说明
RX8_RX_PARITY_ERROR	0x80	奇偶校验错误
RX8_RX_OVERRUN_ERROR	0x40	缓冲区过速错误
RX8_RX_FRAMING_ERROR	0x20	字符成帧错误
RX8_RX_BUF_OVERRUN	0x10	软件 RX 缓冲区过速

副作用：

清除错误状态。 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。 在大内存模式下 （CY8C29xxx 和 CY8CLED16），所有 RAM 页面指针寄存器也会出现这种状况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。 目前，仅修改 CUR_PP 页面指针寄存器。

固件源代码示例

下面的示例固件展示了如何使用 API 函数来创建一个等待 RX8 接收器接收一字节数据的子程序。

```
;-------------------------------------------------------------------------
; Calling routine in Assembly:
;
;    call fWaitToReceiveByte    ; status of RX data returned in A
;    jnz  RX_DATA_NO_ERROR      ; no error found - data in bRxData
;    jmp  RX_DATA_WITH_ERROR    ; error found - status flags in bRxData
;
; Description: Waits for a byte of data to be received by an RX8 receiver.
; Byte of data is returned in a common storage location, bRxData. Value of
; bRxData depends on status of received byte. If there was no error on
; received data then bRxData contains received data byte, else bRxData
; contains the status flags from RX8 control register.
;
; Return Value: In A register:
;   TRUE - data successfully received, bRxData contains received data byte.
;   FALSE - data received with error, bRxData contains status flags.
;
;-------------------------------------------------------------------------

include “rx8.inc”
;;;;;;;;;;;;;;;;;;;;;;;;;;
; exports
;;;;;;;;;;;;;;;;;;;;;;;;;;
; routine name
export fWaitToReceiveByte    ; assembly routine label
export _fWaitToReceiveByte   ; C code label

; Rx data storage
export bRxData               ; assembly routine label
export _bRxData              ; C code label

;;;;;;;;;;;;;;;;;;;;;;;;;;
; data storage
;;;;;;;;;;;;;;;;;;;;;;;;;;
area bss(ram,con,rel)
```


[+] Feedback

```
bRxData:                    ; Rx data storage area
_bRxData:
    blk   1

area text(rom,con,rel)

;;;;;;;;;;;;;;;;;;;;;;;;;
; equates
;;;;;;;;;;;;;;;;;;;;;;;;;
TRUE:   equ   1
FALSE:  equ   0

;;;;;;;;;;;;;;;;;;;;;;;;;
; Routine Code
;;;;;;;;;;;;;;;;;;;;;;;;;
 fWaitToReceiveByte::
_fWaitToReceiveByte::
; Wait for byte to be received
.WAIT_FOR_RX_COMPLETE:
    call  RX8_bReadRxStatus
    push  A
    and   A, RX8_RX_COMPLETE
    jnz   .CHECK_RX_ERRORS
    pop   A
    jmp   .WAIT_FOR_RX_COMPLETE

; Data completely received now check for errors
.CHECK_RX_ERRORS:
    pop   A                     ; Restore status register state
    and   A, RX8_RX_NO_ERROR    ; mask off non-status bits
    jz    .DATA_RX_WITH_NO_ERRORS ; data is valid - no error detected

; Errors detected in received data - return with error condition
;   1) A is set to FALSE indicating error condition
;   2) bRxData contains the RX status flags for further processing
.RX_ERRORS_FOUND:
    mov   [bRxData], A          ; bRxData contains the status flags
    call  RX8_bReadRxData       ; Read RxData reg to prevent future
                                ; overrun error
    mov   A, FALSE              ; Set A to FALSE condition
    ret

; No error detected in received data - return with data
;   1) A is set to TRUE indicating NO error condition
;   2) bRxData contains the received data byte
.DATA_RX_WITH_NO_ERRORS:
    call  RX8_bReadRxData       ; get the received data in A
    mov   [bRxData], A          ; bRxData contains received
                                ; data byte
    mov   A, TRUE               ; set a to NO error condition
    ret

END_fWaitToReceiveByte:
```


[+] Feedback

以下为使用 C 语言编写的示例项目：

```
//-------------------------------------------------------------------------
//
// Prototype:
// BOOL fWaitToReceiveByte(void);
// Usage:
// if ( fWaitToReceiveByte() )
// {
// /* data valid - process it here! - data in bRxData */
// }
// Return Value:
//    TRUE -  data successfully received, bRxData contains
//            received data byte.
//    FALSE - data received with error, bRxData contains
//            status flags.
//
//-------------------------------------------------------------------------


#include "psocapi.h"
#include "m8c.h"
#include "RX8.h"


/* Global bRxData - saves code - ptrs are expensive */
BYTE bRxData;

BOOL fWaitToReceiveByte(void)
{
    BYTE bRxStatus;

    /* Wait to receive full byte*/
    while ( !( bRxStatus=RX8_bReadRxStatus() & RX8_RX_COMPLETE ) )
    {
        /* might want to sleep or keep track of time */
    }

    /* data received, now check for errors */
    if (( bRxStatus & RX8_RX_NO_ERROR ) == 0 )
    {
        /* no error detected */
        bRxData = RX8_bReadRxData();
        return( TRUE );
    }
    else
    {
        /* error detected */
        bRxData = bRxStatus;
        return( FALSE );
    }
}
```

配置寄存器

用于配置 RX8 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. RX 模块、寄存器： 函数

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	1	0	1

该寄存器定义将此数字通信模块配置为 RX8 用户模块。

Table 5. RX 模块、寄存器： 输入

位	7	6	5	4	3	2	1	0
值	输入源				时钟源			

输入源选择 RX8 输入源。 “ 时钟源 ”(Clock Source) 选择驱动接收器时序的时钟。

Table 6. RX 模块、寄存器： 输出

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 7. RX 模块、数据移位寄存器： DR0

位	7	6	5	4	3	2	1	0
值	RX8 移位寄存器							

RX8 移位寄存器： 在输入端检测到起始位时，RX8 状态机硬件生成一个用于将数据移位到该寄存器的 8 分频位时钟。

Table 8. RX 模块、数据寄存器： DR1

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 9. RX 模块、数据缓冲区寄存器： DR2

位	7	6	5	4	3	2	1	0
值	RX8 缓冲区寄存器							

RX8 缓冲区寄存器： 在采样到停止位后，从 RX8 移位寄存器传送数据。

Table 10. RX 模块、控制 / 状态寄存器： CR0

位	7	6	5	4	3	2	1	0
值	奇偶校验错误	过速错误	成帧错误	RX 活动	RX 寄存器满	奇偶校验类型	奇偶校验启用	RX 启用

[+] Feedback

“ 奇偶校验错误 ”(Parity Error) 是用于指示所接收的数据字节的奇偶校验计算结果的标志。

“ 过速错误 ”(Overrun Error) 是用于指示 RX 缓冲区寄存器数据已被覆写的标志。

“ 成帧错误 ”(Framing Error) 是用于指示已经正确接收停止位的标志。

“RX 活动 ”(Rx Active) 是用于指示是否正在主动接收数据字节的标志。

“RX 寄存器满 ”(Rx Reg Full) 是用于指示某个数据字节已经完整接收，此数据字节已经传输至 RX 缓冲区寄存器及错误状况有效的标志。

“ 奇偶校验类型 ”(Parity Type) 是要计算的奇偶校验类型。 如果奇偶校验未启用则无需关注此位。

“ 奇偶校验启用 ”(Parity Enable) 用于启用或禁用所接收奇偶校验位的计算。 通过设置奇偶校验类型位选择奇偶校验。

“RX 启用 ”(Rx Enable) 用于启用或禁用 RX8 接收器。

版本历史记录

版本	创作者	说明
3.4	DHA	添加了大内存模式芯片的兼容性。 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。

[+] Feedback

Copyright © 2002-2011 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.