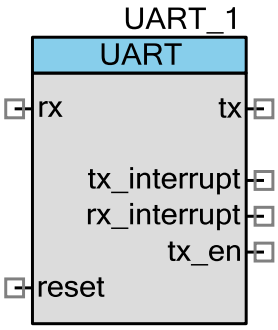
通用异步接收/发送器 (UART)

2.10

特性



- 带有硬件地址检测功能的 9 位寻址模式
- 波特率范围从 110 到 921600 bps，也可任意高达 4 Mbps
- RX 和 TX 缓冲区大小范围有 4 字节 到 65535 字节
- 帧检测、奇偶校验检测和溢出检测
- 可优化的硬件选择，全双工、半双工、仅发送 TX，和仅接收 RX
- 按位 3 取 2 表决
- 中断信号产生和检测
- 8 倍（8x） 或 16 倍（16x） 过采样

概述

UART 提供异步通信，常用串行异步通信设备为 RS232 或 RS485。UART 组件可配置为全双工、半双工、仅 接收 RX 或仅发送 TX 通信方式。所有通信方式都提供相同的基本功能。它们之间的差异仅在于使用的资源量。

为了帮助处理 UART 接收和传送数据，提供了独立大小的可配置缓冲区可映射于 SRAM 中的独立可循环接收发送缓冲区与硬件中的 FIFO 都确保数据不会丢失。这种机制有利于 CPU 利用更多的时间处理关键的实时任务而不是服务于 UART。

在多数应用中，可通过选择波特率、奇偶校验、数据位大小以及起始位数轻松配置 UART。RS232 最常见的配置通常列为“8N1”（全称为八个数据位、无奇偶校验和一个停止位）。这是 UART 组件的默认配置。因此, 多数应用中采用默认配置时, 只需设置波特率。其次，UART 常用于多分支 RS485 网络。UART 组件支持带有硬件地址检测功能的 9 位寻址模式，以及支持在传输过程中启用 TX 收发器的 TX 输出使能信号。

UART 具有悠久的历史，因此随时间推移产生了许多物理层和协议层的接口形式。这些接口形式包括（但不限于）RS423、DMX512、MIDI、LIN 总线、旧终端协议和 IrDa。为了支持常用的 UART 接口形式，UART 组件支持对数据位大小、停止位数、奇偶校验、硬件流控制以及奇偶校验生成和检测的配置。

对于 UART 的硬件编译选项，您可以选择仅在时钟的上升沿输出 UART 数据位的时钟和串行数据流。TX 和 RX 均提供有独立的时钟和数据输出。这些输出目的在于允许通过 CRC 组件与 UART 的连接来自动计算数据 CRC。

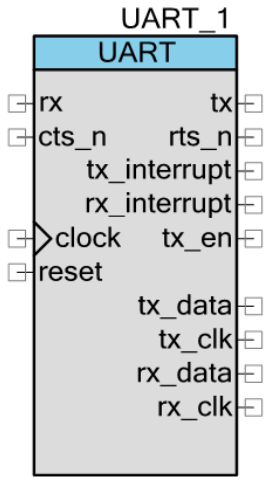
何时使用 UART

每当需要兼容的异步通信接口（尤其是 RS232、RS485 和其他串行设备形式）时都应使用 UART。还可以使用 UART 创建更高级的基于的协议的异步通信，如 DMX512、LIN 和 IrDa 或客户，工业专用协议。

请勿将 UART 用于已创建特定组件以进行协议寻址的情况。例如 DMX512、LIN 或 IrDa 组件，其已具有提供硬件和协议层功能的特定实现。在这种情况下（取决于组件可用性）无需 UART。

输入/输出连接

本节介绍 UART 的各种输入和输出连接。在描述中的某些条件下, 有些 I/O 管脚会不在 UART 组件上显示。



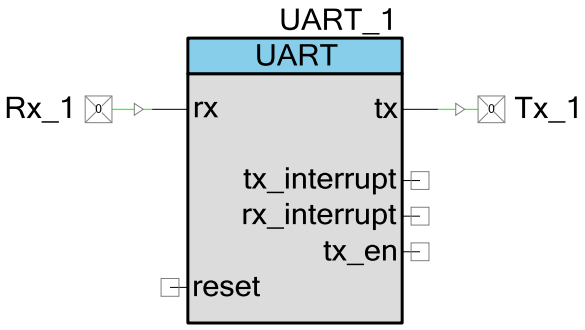
输入	可能已隐藏	说明
clock	Y	输入时钟(**CLOCK**)定义串行通信的波特率（比特率）。根据 **Oversampling Rate**（过采样率）参数，波特率为输入时钟频率的 1/8 或 1/16。该输入时钟频率在 **Clock Selection**（时钟选择）参数设置为 **External Clock**（外部时钟）时可见。如果选择了内部时钟，则您必须在配置过程中定义所需的波特率，并且 PSoC Creator 将解决必要的时钟频率。
复位	N	复位输入将 UART 状态机（RX 和 TX）复位至空闲状态。复位将会丢弃所有正在发送或接收的数据。该复位输入是同步复位，至少需要一个时钟上升沿。复位输入可以保持悬空，不进行外部连接。如果没有任何对象连接到复位线，则组件会向其分配常量逻辑 0。

输入	可能已隐藏	说明
rx	Y	rx 输入携带来自串行总线上另一器件的输入串行数据。该输入在 **Mode**（工作模式）参数设置为 **RX Only**（仅 RX）、**Half Duplex**（半双工）或 **Full UART (RX + TX)**（全双工 UART (RX + TX)）时可见且必须处于连接状态。
cts_n	Y	cts_n 输入显示另一器件准备好接收数据。它是低电平有效输入 (_n)。该输入在 **Flow Control**（流量控制）参数设置为 **Hardware**（硬件）时可见。

输出	可能已隐藏	说明
tx	Y	tx 将输出串行数据传送至串行总线上的另一器件。该输出在 **Mode**（工作模式）参数设置为 **TX Only**（仅 TX）、**Half Duplex**（半双工）或 **Full UART (RX + TX)**（全双工 UART (RX + TX)）时可见。赛普拉斯建议使用外部上拉电阻来保护接收器在运行系统复位过程中免受意外低电平脉冲影响。
rts_n	Y	rts 输出向另一个器件报告您的器件准备好接收数据。此输出是低电平有效 (_n)。当内部 FIFO 和 RX 缓冲区（通过 RX Buffer Size（RX 缓冲区大小）参数分配（当它大于 4 时））已满时，RTS 信号会变为高电平。该输出在 **Flow Control**（流量控制）参数设置为 **Hardware**（硬件）时可见。
tx_en	Y	tx_en 输出主要用于 RS485 通信，以说明您的器件正在总线上发送数据。此输出在发送开始之前变为高电平，在发送完成时变为低电平。这一过程对总线上的其它器件来说, 总线繁忙。该输出在选择了 **Hardware TX Enable**（硬件 TX 启用）参数时可见。
tx_interrupt	Y	tx_interrupt 由一组中断源的逻辑或(OR)产生的。任何一个已使能的中断源触发（true）时，此信号将变为高电平。该输出在 **Mode**（模式）参数设置为 **TX Only**（仅 TX）或 **Full UART (RX + TX)**（全双工 UART (RX + TX)）时可见。
rx_interrupt	Y	rx_interrupt 由一组中断源的逻辑或(OR)组成的。任何一个已使能的中断源触发（true）时，此信号将变为高电平。该输出在 **Mode**（模式）参数设置为 **RX Only**（仅 RX）、**Half Duplex**（半双工）或 **Full UART (RX + TX)**（全双工 UART (RX + TX)）时可见。
tx_data	Y	tx_data 输出用于将 TX 数据移出至 CRC 组件或其他逻辑单元。在选择了 **Enable CRC outputs**（启用 CRC 输出）参数时，该输出可见。
tx_clk	Y	tx_clk的 输出提供用于将 TX 数据移出至 CRC 组件或其他逻辑单元的时钟沿。在选择了 **Enable CRC outputs**（启用 CRC 输出）参数时，该输出可见。
rx_data	Y	rx_data 输出用于将 RX 数据移出至 CRC 组件或其他逻辑单元。在选择了 **Enable CRC outputs**（启用 CRC 输出）参数时，该输出可见。
rx_clk	Y	rx_clk 的输出提供用于将 RX 数据移出至 CRC 组件或其他逻辑单元的时钟沿。在选择了 **Enable CRC outputs**（启用 CRC 输出）参数时，该输出可见。

原理图

组件目录下的默认 UART 模块, 已经是具有默认配置和带有数字输入/输出引脚的 UART 组件。



组件参数

将 UART 组件拖放到您的设计上，然后双击以打开 **Configure**（配置）对话框。

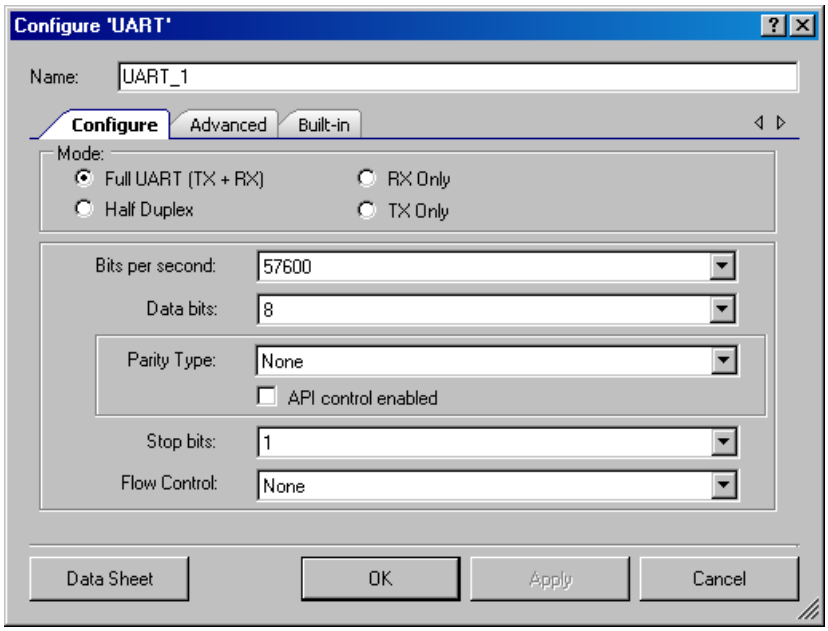
硬件和软件配置选项

硬件配置选项用于更改项目集成和放置在硬件中的方式。如果您对这些选项的任一个进行了更改，则必须重新构建硬件。软件配置选项不影响项目的集成或放置。如果在构建之前设置这些参数，则需要设置其初始值，也可以随时使用提供的 API 更改这些初始值。

下列章节描述 UART 参数以及如何使用对话框配置这些参数。它们还指明了选项是硬件还是软件。

“配置”选项卡

对话框设置看起来像一个超级终端配置窗口，这样可以避免总线两侧配置不正确，因为使用超级终端的 PC 通常为总线的另一侧。



上图中的所有选项均为硬件配置选项。

模式(Mode)

该参数定义了要加入到 UART 组件中的功能模式。这可设置为双向 **Full UART (TX + RX)**（全双工 UART (TX + RX)）（默认）、**Half Duplex**（半双工）UART（使用一半资源）、RS-232 接收器（**RX Only**（仅 RX））或发送器（**TX Only**（仅 TX））。

每秒位数（Bits per second）

该参数定义由时钟产成的硬件的波特率或位宽配置。默认值为 **57600**。

如果使用内部时钟（通过 **Clock Selection**（时钟选择）参数设置），则 PSoC Creator 会生成实现此波特率所需的时钟。

数据位（Data bits）

该参数定义单个 UART 数据传输在开始到停止期间发送的数据位数。有以下几种选项：**5**、**6**、**7**、**8**（默认）或 **9**。

- 八个数据位为默认配置，即每次传输发送一个字节。

- 9 位模式并不发送 9 个数据位；第 9 位会取代奇偶校验位，作为使用 Mark/Space 奇偶校验的地址指示符。如果采用 9 个数据位模式，则应选择 Mark/Space 奇偶校验。

奇偶校验类型（Parity Type）

该参数定义传输中奇偶校验位类型功能。可设置为 **None**（无）（默认）、**Odd**（奇校验）、**Even**（偶校验）或 **Mark/Space**。如果您选择了 9 个数据位，则应选择 **Mark/Space** 作为 **Parity Type**（奇偶校验类型）。

启用 API 控制（API control enabled）

该复选框利用控制寄存器和 UART_WriteControlRegister() 函数来更改奇偶校验。若选中该选项，则将可在不中断 UART 操作的情况下，在字节之间动态更改奇偶校验类型，但因此组件会使用更多的资源。

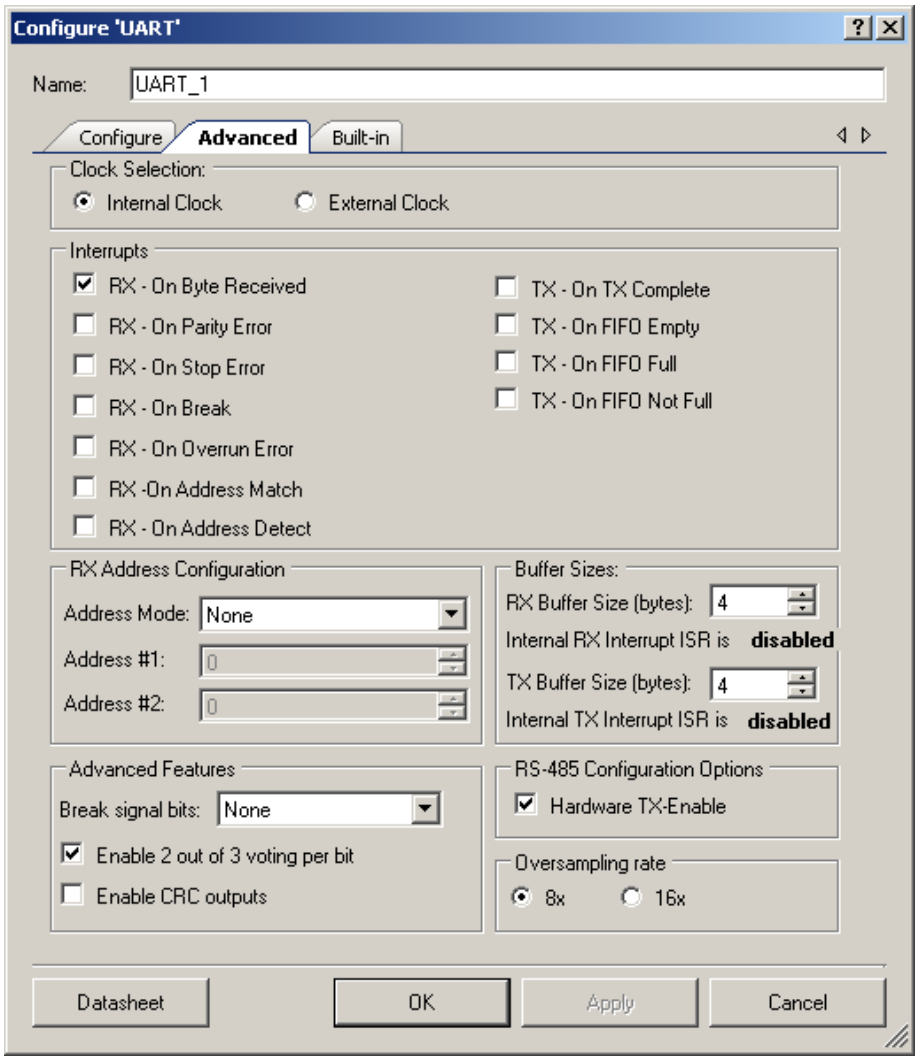
停止位（Stop bits）

该参数定义发送器中实现的停止位数。该参数可设置为 **1**（默认）或 **2** 个数据位。

流量控制（Flow Control）

该参数允许您在 **Hardware**（硬件）或 **None**（无）（默认）之间进行选择。该参数设置为 **Hardware**（硬件）时，CTS 和 RTS 信号管脚可在模块上显示。

“高级”选项卡



硬件配置选项

时钟选择（Clock Selection）

时钟选择参数决定了波特率产生方式，主要有内部配置时钟和外部配置时钟产生方式。设置为 **Internal Clock**（内部时钟）时，所需的时钟频率将由 PSoC Creator 进行计算和配置。在 **External Clock**（外部时钟）模式下，组件不会控制波特率，但可计算预计的波特率。

如果此参数设置为 **Internal Clock**（内部时钟），则时钟输入在宏模块上不可见。

寻址模式（Address Mode）

寻址参数定义了硬件和软件如何交互处理器件地址和数据字节。该参数可设置为以下类型：

- **Software Byte by Byte**（软件逐字节寻址） — 硬件可检测到所接收的每个字节的地址字节。软件必须读取字节地址，并确定该地址是否与 **Address #1**（地址 #1）或 **Address #2**（地址 #2）参数中定义的器件地址或任何其他附加地址相匹配。
- **Software Detect to Buffer**（软件检测缓冲区寻址）— 硬件可检测到地址字节（UART_RX_STS_MRKSPC 状态）。软件（嵌入在 RX ISR 中）会读取地址字节，并确定该地址是否与 **Address #1**（地址 #1）或 **Address #2**（地址 #2）参数中定义的器件地址相匹配（使用 UART_RX_STS_ADDR_MATCH 状态）。它随后将所有寻址数据以及地址字节复制到 RX 缓冲区，RX 缓冲区由 **RX Buffer Size**（RX 缓冲区大小）参数定义。**RX Buffer Size**（RX 缓冲区大小）应手动设置为大于 4 字节。未寻址数据从 FIFO 进行读取，但是不写入缓冲区。
- **Hardware Byte By Byte**（硬件逐字节寻址）— 硬件检测寻址字节，并强制中断（RX - On Byte Received（RX — 接收到字节））将所有数据以及地址从硬件 FIFO 移至数据缓冲区，数据缓冲区由 **RX Buffer Size**（RX 缓冲区大小）定义。硬件不将未寻址字节保存到 FIFO，并且不会为它们生成任何中断。
- **Hardware Detect to Buffer**（硬件检测缓冲区寻址）— 硬件检测寻址字节，并强制中断（RX - 接收到字节）只将数据（不包括地址字节）从硬件 FIFO 移至数据缓冲区，数据缓冲区由 **RX Buffer Size**（RX 缓冲区大小）定义。硬件不将未寻址字节保存到 FIFO，并且不会为它们生成任何中断。
- **None**（无） – 不实现 RX 地址检测。

RX 地址 #1/#2 （RX Address #1/#2）

RX Address（RX 地址）参数表示 UART 可以采用的最多两个器件地址。这些参数都存储在硬件中，用于 **Address Mode**（地址模式）参数中描述的硬件地址检测模式。用到 **RX Address #2**（RX 地址 #2）的硬件未不支持 **Half Duplex**（半双工）模式。对于软件地址模式，这些参数可用于固件。

高级功能（Advanced Features）

- **Break signal bits**（中断信号位）– Break signal bits（中断信号位）参数可启用中断信号产生和检测，并定义传输的逻辑 0 位数量。此选项设置为 **None**（无）可节省资源。
- **Enable 2 out of 3 voting per bit**（启用按位 3 取 2 表决）— **Enable 2 out of 3 voting per bit**（启用按位 3 取 2 表决）参数可启用或禁用错误补偿算法。禁用此选项可节省资源。有关更多信息，请参见本数据手册的功能描述一节。

- **Enable CRC outputs**（启用 CRC 输出）— **Enable CRC outputs**（启用 CRC 输出）参数可启用或禁用 tx_data、tx_clk、rx_data 和 rx_clk 输出。它们用于输出时钟和串行数据流，这些仅在时钟上升沿输出 UART 数据位。这些输出旨在允许 CRC 数据的自动计算。禁用此选项可节省资源。

硬件 TX 使能（Hardware TX Enable）

此参数可启用或禁用 UART 发送（TX UART）中的 TX-Enable 输出的使用。此信号（TX-Enable）用于 RS485 通信。在缓冲区允许条件下，硬件自动提供这个信号输出功能。

过采样率（Oversampling Rate）

此参数可以为波特率的产生选择时钟分频器。

软件配置选项

中断

Interrupt On（启用中断）参数可以用于配置中断源。启用中断(Interrupt On)的所有中断源进行″或(OR)″运算，以提供一组可触发中断的最终事件。中断的一些参数定义了初始配置, 但通过软件配置可以随时重新配置这些中断模式。

- **RX - On Byte Received（RX — 接收字节）** (UART_RX_STS_FIFO_NOTEMPTY)
- **RX - On Parity Error（RX — 奇偶校验错误）** (UART_RX_STS_PAR_ERROR)
- **RX - On Stop Error（RX — 停止位错误）** (UART_RX_STS_STOP_ERROR)
- **RX - On Break（RX — 中断）** (UART_RX_STS_BREAK)
- **RX - On Overrun Error（RX — 溢出错误）** (UART_RX_STS_OVERRUN)
- **RX - On Address Match（RX — 地址匹配）** (UART_RX_STS_ADDR_MATCH)
- **RX - On Address Detect（RX — 地址检测）** (UART_RX_STS_MRKSPC)
- **TX - On TX Complete（TX — TX 完成）** (UART_TX_STS_COMPLETE)
- **TX - On FIFO Empty（TX — FIFO 为空）** (UART_TX_STS_FIFO_EMPTY)
- **TX - On FIFO Full（TX — FIFO 已满）** (UART_TX_STS_FIFO_FULL)
- **TX - On FIFO Not Full（TX — FIFO 未满）** (UART_TX_STS_FIFO_NOT_FULL)

您可使用连接至 tx_interrupt 或 rx_interrupt 输出的外部中断器件处理 中断服务程序（ISR）。中断输出引脚是否可见取决于所选的**模式**参数。根据所选中断状态, 这个中断将输出与内部中断相同的信号。

这些输出随后可用作来自独立于中断的 RX 或 TX 缓冲区的 DMA 请求源，或用作另一个中断（具体取决于所需功能）。

RX 缓冲区大小（字节）

此参数定义分配给 RX 缓冲区的 RAM 字节数。数据从接收寄存器移至该缓冲区。

当所选的缓冲区大小是 4 字节时，硬件 FIFO 的四个字节将用作缓冲区。缓冲区大小超过 4 字节时，需要使用中断将数据从接收 FIFO 移动至此缓冲区。UART_GetChar() 或 UART_ReadRXData() API 函数在不需要对顶层固件进行任何更改的条件下, 可以从正确数据源获取正确数据。

当 RX 缓冲区大小超过 4 字节时，**Internal RX Interrupt ISR**（内部 RX 中断 ISR） 将自动启用，并且 **RX – On Byte Received**（RX – 接收字节）中断源被选中并禁用，因为它会导致错误的处理功能。

TX 缓冲区大小（字节）

此参数定义了分配给 TX 缓冲区的 RAM 字节数。数据通过 UART_PutChar() 和 UART_PutArray() API 命令写入该缓冲区。

当所选的缓冲区大小等于四字节时，硬件 FIFO 的四个字节将用作缓冲区；否则，将分配 RAM 缓冲区。TX 缓冲区大小超过四字节时，需要使用中断将数据从发送缓冲区移至硬件 FIFO，同时不需要对顶层固件进行任何更改。

当 TX 缓冲区大小超过四字节时，**Internal TX Interrupt ISR**（内部 TX 中断 ISR）将自动启用，并且 **TX – On FIFO EMPTY**（TX – FIFO 为空）中断源被选中并禁用，因为它会导致错误的处理功能。

TX 中断在 **Half Duplex**（半双工）模式中不可用；因此在 **Half Duplex**（半双工）模式选中时，**TX Buffer Size**（TX 缓冲区大小）限制为四字节。

内部 RX 中断 ISR

启用 UART 的 RX 部分的中断 ISR。因为需要使用内部 ISR 将数据从 FIFO 传输至 RX 缓冲区，所以该参数根据 **TX Buffer Size**（TX 缓冲区大小）参数自动设置。

内部 TX 中断 ISR

启用 UART 的 TX 部分的中断 ISR。因为需要使用内部 ISR 将数据从 TX 缓冲区传输至 FIFO，所以该参数根据 **TX Buffer Size**（TX 缓冲区大小）参数自动设置。

时钟选择

当选择内部时钟配置时，PSoC Creator 计算所需的频率和时钟源，并产生、实现所需的资源。反之，则必须提供时钟, 并根据输入时钟频率的 1/8 或 1/16 分频计算波特率。

时钟容差最大应为 ±2%。如果时钟不能在此限制内生成，将生成警告。在这种情况下，应在 DWR 中更改主时钟，或是应使用外部基于晶振的时钟。

放置

UART 组件放置于整个 UDB(通用数字阵列）阵列中，并且所有放置信息通过 *cyfitter.h* 文件提供给 API。

资源

资源	资源类型					API 存储器（字节）		引脚（每个外部 I/O）
	数据路径单元	PLD	状态单元	Control/ Count7 单元	中断	闪存	RAM	
全双工 UART	3	27	2	2	2	1695	24	12
全双工 UART*	2	28	2	3	2	1703	24	12
简单 UART	3	6	2	1	0	551	5	4
半双工	1	7	1	2	0	577	4	3
仅用于 RX	1	3	1	1	0	223	3	2
仅用于 TX	2	3	1	0	0	383	4	2
仅用于 TX*	1	3	1	1	0	431	4	2

* 参数 TxBitClkGenDP = false。（要进行切换，请选择 Expression View of Advanced（高级视图）选项卡）。

应用程序编程接口

应用程序编程接口 (API) 程序允许您通过软件配置组件。下表列出了每个函数的接口，并进行了说明。以下各节将更详细地介绍每个函数。

默认情况下，PSoC Creator 将实例名称“UART_1”分配给在指定设计中的组件的第一个实例。您可以将该实例重命名为符合标识符语法规则的任意唯一值。实例名称会成为每个全局函数名称、变量和常量符号的前缀。出于可读性考虑，下表中使用的实例名称为“UART”。

函数	说明
UART_Start()	初始化并启用 UART 操作
UART_Stop()	禁用 UART 操作

函数	说明
UART_ReadControlRegister()	返回控制寄存器的当前值
UART_WriteControlRegister()	在控制寄存器中写入一个 8 位值
UART_EnableRxInt()	启用内部中断 IRQ
UART_DisableRxInt()	禁用内部中断 IRQ
UART_SetRxInterruptMode()	配置已启用的 RX 中断源
UART_ReadRxData()	返回 RX 数据寄存器中的数据
UART_ReadRxStatus()	返回状态寄存器的当前状态
UART_GetChar()	返回已接收数据的下一字节
UART_GetByte()	立即读取 UART RX 缓冲区，返回已接收的字符和错误状况
UART_GetRxBufferSize()	返回RX缓冲区中已接收字节数, 以字节为单位计数
UART_ClearRxBuffer()	清除所有已接收数据的内存阵列
UART_SetRxAddressMode()	设置由软件控制的寻址模式，该模式用于 UART 中的 RX 部分
UART_SetRxAddress1()	设置第一个硬件可检测出的地址（共两个地址）
UART_SetRxAddress2()	设置第二个硬件可检测出的地址（共两个地址）
UART_EnableTxInt()	启用内部中断 IRQ
UART_DisableTxInt()	禁用内部中断 IRQ
UART_SetTxInterruptMode()	配置已启用的 TX 中断源
UART_WriteTxData()	在不检查缓冲区空间或状态的情况下发送一个字节
UART_ReadTxStatus()	读取 UART TX 部分的状态寄存器
UART_PutChar()	向发送缓冲区放置一个字节的数据，在总线可用时发送
UART_PutString()	将字符串中的数据放入存储器缓冲区用于发送
UART_PutArray()	将存储器阵列中的数据放入存储器缓冲区用于发送。
UART_PutCRLF()	向发送缓冲区写入一个字节的数据，并输入回车符和换行符
UART_GetTxBufferSize()	确定 TX 缓冲区中使用的字节数。空缓冲区返回 0
UART_ClearTxBuffer()	清除 TX 缓冲区内的所有数据
UART_SendBreak()	在总线上发送一个中断信号
UART_SetTxAddressMode ()	配置发射器，将下一个字节作为地址或数据发送
UART_LoadRxConfig()	加载接收器配置。半双工 UART 已准备好接收字节
UART_LoadTxConfig()	加载发送器配置。半双工 UART 已准备好发送字节

函数	说明
UART_Sleep()	停止 UART 操作并保存用户配置
UART_Wakeup()	恢复并使能用户配置
UART_Init()	初始化随自定义程序提供的默认配置
UART_Enable()	启用 UART 模块操作
UART_SaveConfig()	保存当前用户配置
UART_RestoreConfig()	恢复用户配置

全局变量

变量	说明
UART_initVar	UART_initVar变量指示 UART 是否已初始化。该变量初始化为 0，并在第一次调用 UART_Start() 时设置为 1。这样，第一次调用 UART_Start() 子程序后，器件不用重新初始化即可重启。 若要使组件正确操作，必须先初始化 UART，然后再运行“发送”或“放置”命令。因此，所有写入发送数据的 API 必须用此变量检查确认UART组件已初始化。 如需重新初始化UART组件，可在 UART_Start() 或 UART_Enable() 函数前调用 UART_Init() 函数。
UART_rxBuffer	这是一个 RAM 分配的 RX 缓冲区，其长度由用户定义。当 **RX Buffer Size**（RX 缓冲区大小）参数设置为大于 4 时，此缓冲区由中断用于存储接收的数据。UART_ReadRxData() 和 UART_GetChar() 也用这个缓冲区将数据传递至用户级别固件。
UART_rxBufferWrite	RX 中断使用此变量作为 UART_rxBuffer 的循环索引来写入数据。UART_ReadRxData() 和 UART_GetChar() 函数也使用此变量来识别新数据。由 UART_ClearRxBuffer() 函数清零。
UART_rxBufferRead	UART_ReadRxData() 和 UART_GetChar() 函数使用此变量作为 UART_rxBuffer 的循环索引来读取数据。由 UART_ClearRxBuffer() 函数清零。
UART_rxBufferLoopDetect	当 UART_rxBufferWrite 索引超过 UART_rxBufferRead 索引时，此变量在 RX 中断中设置为 1。这是预过载条件，将会在接收到下一个字节时会影响 UART_rxBufferOverflow。在调用 UART_ReadRxData() 或 UART_GetChar() 函数时设置为零。由 UART_ClearRxBuffer() 函数清零。
UART_rxBufferOverflow	此变量用于表示过载条件。当 UART_rxBuffer 中没有空余空间用于写入新的数据时，此变量在 RX 中断中设置为 1。该条件作为 UART_RX_STS_SOFT_BUFF_OVER 位与 RX 状态寄存器位一起由 UART_ReadRxStatus() 函数返回并清零。由 UART_ClearRxBuffer() 函数清零。

变量	说明
UART_txBuffer	这是 RAM 分配的 TX 缓冲区，其长度由用户定义。当 **TX Buffer Size**（TX 缓冲区大小）参数设置为大于 4 时，此缓冲区由发送 API 用于存储数据以进行发送。TX 中断也用这个缓冲区将数据移至硬件 FIFO。
UART_txBufferWrite	UART_WriteTxData()、UART_PutChar()、UART_PutString()、UART_PutArray() 和 UART_PutCRLF() 函数使用此变量作为 UART_txBuffer 的循环索引来写入数据。TX 中断也用此变量来识别要发送的新数据。由 UART_ClearTxBuffer() 函数清零。
UART_txBufferRead	TX 中断使用此变量作为 UART_txBuffer 的循环索引来读取数据。由 UART_ClearRxBuffer() 函数清零。

void UART_Start(void)

说明： 这是开始执行组件操作的首选方法。UART_Start() 设置 initVar 变量，调用 UART_Init() 函数，然后调用 UART_Enable() 函数。

参数： 无（void）

返回值： 无（void）

特殊情况： 如果已设置 initVar 变量，则该函数仅调用 UART_Enable() 函数。

void UART_Stop(void)

说明： 禁用 UART 操作。

参数： 无（void）

返回值： 无（void）

特殊情况： 无

uint8 UART_ReadControlRegister(void)

说明： 返回控制寄存器的当前值。

参数： 无（void）

返回值： 8位无符号整型（uint8）：控制寄存器的内容。以下定义可用于解析返回值。有关更多信息，请参见接近本数据手册结尾的控制寄存器说明。

值	说明
UART_CTRL_HD_SEND	半双工UART启用时的配置, RX模式配置为0, 或TX模式配置为1。
UART_CTRL_HD_SEND_BREAK	设置此值，允许UART在总线上发送一个中断信号。该位由 UART_SendBreak() 函数写入。
UART_CTRL_MARK	将下一次数据传输（在 Mark/Space 奇偶校验模式）中的奇偶校验位配置为 1 或 0。
UART_CTRL_PARITY_TYPE_MASK	配置下一传输的奇偶校验的两位宽字段（若软件可配置）。以下定义（按 UART_CTRL_PARITY_TYPE0_SHIFT 左移位）可用于识别奇偶校验类型：
UART__B_UART__NONE_REVB	无奇偶校验
UART__B_UART__EVEN_REVB	偶校验
UART__B_UART__ODD_REVB	奇校验
UART__B_UART__MARK_SPACE_REVB	Mark/Space 奇偶校验
UART_CTRL_RXADDR_MODE_MASK	为 UART 接收器配置预期的硬件寻址操作的三位宽字段。以下定义（按 UART_CTRL_RXADDR_MODE0_SHIFT 左移位）可用于识别地址模式：
UART__B_UART__AM_SW_BYTE_BYTE	软件字节寻址检测
UART__B_UART__AM_SW_DETECT_TO_BUFFER	软件检测缓冲区寻址（Software Detect to Buffer）地址检测
UART__B_UART__AM_HW_BYTE_BY_BYTE	硬件字节寻址检测
UART__B_UART__AM_HW_DETECT_TO_BUFFER	硬件检测缓冲区寻址（Hardware Detect to Buffer）地址检测
UART__B_UART__AM_NONE	无地址检测

特殊情况： 无

void UART_WriteControlRegister(uint8 control)

说明： 在控制寄存器中写入一个 8 位值

参数： 8位无符号整型（uint8）控制字：控制寄存器的值

值	说明
UART_ CTRL_HD_SEND	如果启用半双工UART，配置UART为RX模式时为0，为TX模式时为1。可用 UART_LoadTxConfig() 和 UART_LoadRxConfig() 函数进行设置和清除。
UART_ CTRL_HD_SEND_BREAK	设置此值，允许UART在总线上发送一个中断信号。此位最好用 UART_SendBreak() 函数写入。
UART_CTRL_MARK	将下一次数据传输（在 Mark/Space 奇偶校验模式）中的奇偶校验位配置为 1 或 0。
UART_CTRL_PARITY_TYPE_MASK	配置下一传输的奇偶校验的两位宽字段（若软件可配置）。以下定义（按 UART_CTRL_PARITY_TYPE0_SHIFT 左移位）可用于设置奇偶校验类型：
UART__B_UART__NONE_REVB	无奇偶校验
UART__B_UART__EVEN_REVB	偶校验
UART__B_UART__ODD_REVB	奇校验
UART__B_UART__MARK_SPACE_REVB	Mark/Space 奇偶校验
UART_CTRL_RXADDR_MODE_MASK	为 UART 接收器配置预期的硬件寻址操作的三位宽字段。以下定义（按 UART_CTRL_RXADDR_MODE0_SHIFT 左移位）可用于设置地址模式：
UART__B_UART__AM_SW_BYTE_BYTE	软件逐字节寻址检测
UART__B_UART__AM_SW_DETECT_TO_BUFFER	软件检测缓冲区寻址（Software Detect to Buffer）地址检测
UART__B_UART__AM_HW_BYTE_BY_BYTE	硬件逐字节寻址检测
UART__B_UART__AM_HW_DETECT_TO_BUFFER	硬件检测缓冲区寻址（Hardware Detect to Buffer）地址检测
UART__B_UART__AM_NONE	无地址检测

返回值： 无（void）

特殊情况： 无

void UART_EnableRxInt(void)

说明： 启用内部接收器中断。

参数： 无（void）

返回值： 无（void）

特殊情况： 只有在 UART 中选择了 RX 内部中断实现后可用

void UART_DisableRxInt(void)

说明： 禁用内部接收器中断。

参数： 无（void）

返回值： 无 （void）

特殊情况： 只有在 UART 中选择了 RX 内部中断实现后可用

void UART_SetRxInterruptMode(uint8 intSrc)

说明： 将 RX 中断源配置为使能。

参数： 8位无符号整型uint8 intSrc：包含要启用的 RX 中断的位字段。基于状态寄存器的位字段排列。该值必须是下列状态寄存器位掩码的组合：

值	说明
UART_RX_STS_FIFO_NOTEMPTY	接收到字节时中断。
UART_RX_STS_PAR_ERROR	奇偶校验错误时中断。
UART_RX_STS_STOP_ERROR	停止位错误时中断。
UART_RX_STS_BREAK	终止时中断。
UART_RX_STS_OVERRUN	溢出错误时中断。
UART_RX_STS_ADDR_MATCH	地址匹配时中断。
UART_RX_STS_MRKSPC	地址检测时中断。

返回值： 无（void）

特殊情况： 无

uint8 UART_ReadRxData(void)

说明： 返回接收到数据的下一字节。此函数返回数据而不检查状态。必须单独检查状态。

参数： 无（void）

返回值： 8位无符号整型uint8：从 RX 寄存器接收到的数据

特殊情况： 无

uint8 UART_ReadRxStatus(void)

说明： 返回接收器状态寄存器的当前状态以及软件缓冲区溢出状态。

参数： 无（void）

返回值： 8位无符号整型uint8：当前的 RX 状态寄存器值

值	说明
UART_RX_STS_FIFO_NOTEMPTY	若设置，则表示 FIFO 有可用数据。
UART_RX_STS_PAR_ERROR	若设置，则表示检测出一处奇偶校验错误。
UART_RX_STS_STOP_ERROR	若设置，则表示检测出帧错误。当停止位应为（逻辑 1）而 UART 硬件发现逻辑 0 时导致帧错误。
UART_RX_STS_BREAK	若设置，则表示检测出一处终止。
UART_RX_STS_OVERRUN	若设置，则表示 FIFO 缓冲区溢出。
UART_RX_STS_ADDR_MATCH	表示已接收的字节与可用于硬件地址检测的两个地址之一匹配。如果 **Address Mode**（地址模式）设置为 **None**（无），则不实现此功能。在 **Half Duplex**（半双工）模式中，仅为此检测实现 **Address #1**（地址 #1）。
UART_RX_STS_MRKSPC	Mark/Space 奇偶校验位的状态。该位表示在传输的奇偶校验位的位置中是否有标记或空格。如果 **Address Mode**（地址模式）设置为 **None**（无），则不实现此检测。
UART_RX_STS_SOFT_BUFF_OVER	如果设置，则表示 RX 缓冲区溢出。

特殊情况： 所有状态寄存器位均在读取后清除，除 UART_RX_STS_FIFO_NOTEMPTY 以外。
RX 数据寄存器读取后立即清除 UART_RX_STS_FIFO_NOTEMPTY。
请参见本数据手册后面的寄存器一节。

uint8 UART_GetChar(void)

说明： 返回最后接收到的数据字节。UART_GetChar 设计用于 ASCII 字符，并返回8位无符号整型 unit8 数值，其中 1 至 255 是有效字符值，而 0 表示出现错误或数据不存在。

参数： 无（void）

返回值： 8位无符号整型uint8：从 UART RX 缓冲区读取的字符。1 至 255 之间的 ASCII 字符值为有效字符值。返回零则表示出现错误状况或数据不可用。

特殊情况： 无

uint16 UART_GetByte(void)

说明： 立即读取 UART RX 缓冲区数据，返回已接收的字符和错误信息。

参数： 无（void）

返回值： 16位无符号整型uint16：最高有效字节（MSB） 包含状态信息，而 最低有效字节（LSB） 包含 UART RX 数据。若最高有效字节（MSB） 为非零数据，则说明出现了错误。

特殊情况： 无

uint8/uint16 UART_GetRxBufferSize(void)

说明： 返回 RX 缓冲区中已接收字节数。

参数： 无（void）

返回值： 8位或16位无符号整型uint8/uint16：RX 缓冲区内余下字节数的取整计数。返回值类型取决于 RX Buffer Size（RX 缓冲区大小）参数。

特殊情况： 无

void UART_ClearRxBuffer(void)

说明： 清除包含所有接收数据的接收器存储器缓冲区和硬件 RX FIFO。

参数： 无（void）

返回值： 无（void）

特殊情况： 无

void UART_SetRxAddressMode(uint8 addressMode)

说明： 设置由软件控制的寻址模式，该模式由 UART 中的 RX 部分使用。

参数： 8位无符号整型uint8 addressMode：列出的值表示将要实现的 RX 寻址模式。

值	说明
UART__B_UART__AM_SW_BYTE_BYTE	软件逐字节地址检测
UART__B_UART__AM_SW_DETECT_TO_BUFFER	软件检测缓冲区寻址（Software Detect to Buffer）地址检测
UART__B_UART__AM_HW_BYTE_BY_BYTE	硬件逐字节地址检测
UART__B_UART__AM_HW_DETECT_TO_BUFFER	硬件检测缓冲区寻址（Hardware Detect to Buffer）地址检测
UART__B_UART__AM_NONE	无地址检测

返回值： 无（void）

特殊情况： 无

void UART_SetRxAddress1(uint8 address)

说明： 设置第一个可用硬件检测出的接收器地址（共两个地址）。

参数： 8位无符号整型uint8 address：用于硬件地址检测的地址 #1（Address #1）

返回值： 无（void）

特殊情况： 无

void UART_SetRxAddress2(uint8 address)

说明： 设置第二个可用硬件检测出的接收器地址（共两个地址）。

参数： 8位无符号整型uint8 address：用于硬件地址检测的地址 #2（Address #2）

返回值： 无（void）

特殊情况： 无

void UART_EnableTxInt(void)

说明： 启用内部发送器中断。

参数： 无（void）

返回值： 无（void）

特殊情况： 只有在 UART 配置中选择 TX 内部中断实现后才可用。

void UART_DisableTxInt(void)

说明： 禁用内部发送器中断。

参数： 无（void）

返回值： 无（void）

特殊情况： 只有在 UART 配置中选择 TX 内部中断实现后才可用。

void UART_SetTxInterruptMode(uint8 intSrc)

说明： 配置即将启用的 TX 中断源（但不启用中断）。

参数： uint8 intSrc：包含将启用的 TX 中断源的位字段

值	说明
UART_TX_STS_COMPLETE	TX 字节完成时中断
UART_TX_STS_FIFO_EMPTY	TX FIFO 为空时中断
UART_TX_STS_FIFO_FULL	TX FIFO 已满时中断
UART_TX_STS_FIFO_NOT_FULL	TX FIFO 未满时中断

返回值： 无（void）

特殊情况： 无

void UART_WriteTxData(uint8 txDataByte)

说明： 在不检查 TX 状态寄存器的情况下，将一个字节的数据放入发送缓冲区内以备在总线可用时发送。必须单独检查TX状态。

参数： 8位无符号整型uint8 txDataByte：数据字节

返回值： 无（void）

特殊情况： 无

uint8 UART_ReadTxStatus(void)

说明： 读取 UART TX 部分的状态寄存器。

参数： 无（void）

返回值： 8位无符号整型uint8 ：TX 状态寄存器的内容

值	说明
UART_TX_STS_COMPLETE	若设置，则表示已成功发送该字节
UART_TX_STS_FIFO_EMPTY	若设置，则表示 TX FIFO 为空
UART_TX_STS_FIFO_FULL	若设置，则表示 TX FIFO 已满
UART_TX_STS_FIFO_NOT_FULL	若设置，则表示 FIFO 未满

特殊情况： 此函数可读取TX 状态寄存器，并在读取后清除。

void UART_PutChar(uint8 txDataByte)

说明： 将一个字节的数据放入发送缓冲区内以备在总线可用时发送。这是一种可阻塞的 API，该 API 将一直等待直到 TX 缓冲区有空间保存该数据。

参数： 8位无符号整型数uint8 txDataByte：是要传输的数据字节

返回值： 无（void）

特殊情况： 无

void UART_PutString(char* string)

说明： 向 TX 缓冲区发送以空字符结尾的字符串用于传输。

参数： 字符型指针char* string：指向存储在 RAM 或 ROM 中以空字符结尾的字符串阵列的指针

返回值： 无（void）

特殊情况： 若 TX 缓冲区内没有足够的存储空间来存储整个字符串，则此函数将阻塞，直到字符串的最后一个字符载入 TX 缓冲区内。

void UART_PutArray(uint8* string, uint8/uint16 byteCount)

说明： 将存储器阵列中 N 个字节的数据放入 TX 缓冲区内用于传输。

参数： 8位无符号整型指针uint8* string：RAM 或 ROM 中存储的存储器阵列地址

8位或16位无符号整型uint8/uint16 byteCount：即将发送的字节数。参数类型取决于 **TX Buffer Size**（TX 缓冲区大小）参数。

返回值： 无（void）

特殊情况： 若 TX 缓冲区内没有足够的存储空间来存储整个阵列，则此函数将阻塞，直到阵列的最后一个字节载入 TX 缓冲区内。

void UART_PutCRLF(uint8 txDataByte)

说明： 向发送缓冲区写入一个字节的数据后，输入回车符 (0x0D) 和换行符 (0x0A)。

参数： 8位无符号整型uint8 txDataByte：在回车符和换行符前的用于发送的数据字节

返回值： 无（void）

特殊情况： 若 TX 缓冲区内没有足够的存储空间来存储这三个字节，则此函数将阻塞，直到三个字节中的最后一个字节载入 TX 缓冲区内。

uint8/uint16 UART_GetTxBufferSize(void)

说明： 确定 TX 缓冲区中使用的字节数。空缓冲区返回 0。

参数： 无（void）

返回值： 8位/16位无符号整型uint8 /uint16：TX 缓冲区中使用的字节数。返回值类型取决于 **TX Buffer Size**（TX 缓冲区大小）参数。

特殊情况： 无

void UART_ClearTxBuffer(void)

说明： 清除 TX 缓冲区和硬件 FIFO 的所有数据。

参数： 无（void）

返回值： 无（void）

特殊情况： 在发送缓冲区内的等待数据将会被清除而不发送；而当前正在发送的一个字节将完成发送。

void UART_SendBreak(uint8 retMode)

说明： 在总线上发送一个中断信号。

参数： 8位无符号整型uint8 retMode：发送中断返回模式。有关选项，请参见下表。

选项	说明
UART_SEND_BREAK	为中断初始化寄存器，发送中断信号并立即返回
UART_WAIT_FOR_COMPLETE_REINIT	等待直到完成中断传输，将寄存器重新初始化为标准传输模式，然后返回
UART_REINIT	将寄存器重新初始化为标准传输模式，然后返回
UART_SEND_WAIT_REINIT	执行这两个选项：UART_SEND_BREAK 和 UART_WAIT_FOR_COMPLETE_REINIT。建议在多数情况下使用该选项

返回值： 无（void）

特殊情况： UART_SendBreak() 函数对寄存器进行初始化以发送中断信号。中断信号的长度取决于中断信号位的配置。在继续进行标准 8 位通信前，应重新对寄存器配置进行初始化。

void UART_SetTxAddressMode(uint8 addressMode)

说明： 配置发送器，将下一个字节作为地址或数据发送。

参数： 8位无符号整型uint8 addressMode：

选项	说明
UART_SET_SPACE	配置发射器，将下一个字节作为数据发送。
UART_SET_MARK	配置发射器，将下一个字节作为地址发送。

返回值： 无（void）

特殊情况： 此函数可设置并清除控制寄存器中的 UART_CTRL_MARK 位。

void UART_LoadRxConfig(void)

说明： 在半双工模式下加载接收器配置。调用该函数后，UART 即可接收数据。

参数： 无（void）

返回值： 无（void）

特殊情况： 仅在半双工模式下有效。必须确保前一次数据传输已完成且能够安全卸载发送器配置。

void UART_LoadTxConfig(void)

说明： 在半双工模式下加载发送器配置。调用该函数后，UART 即可发送数据。

参数： 无（void）

返回值： 无（void）

特殊情况： 仅在半双工模式下有效。必须确保前一次数据操作已完成且能够安全卸载接收器配置。

void UART_Sleep(void)

说明： 这是准备让组件进入睡眠状态的首选 API 函数。UART_Sleep() API 保存当前器件的状态。然后调用 UART_Stop() 函数，并调用 UART_SaveConfig() 以保存硬件配置。

在调用 CyPmSleep() 或 CyPmHibernate() 函数之前调用 UART_Sleep() 函数。有关电源管理函数的更多信息，请参考 PSoC Creator *System Reference Guide*（《系统参考指南》）。

参数： 无（void）

返回值： 无（void）

特殊情况： 无

void UART_Wakeup(void)

说明： 此 API函数是将UART组件恢复到调用 UART_Sleep() 前的状态。UART_Wakeup() 函数调用 UART_RestoreConfig() 函数以恢复配置。如果组件在调用 UART_Sleep() 函数前已启用，则 UART_Wakeup() 函数也将重新启用组件。

参数： 无（void）

返回值： 无（void）

特殊情况： 该函数可清除 RX 和 TX 软件缓冲区以及硬件 FIFO，但不会复位任何硬件状态机。调用 UART_Wakeup() 函数前未调用 UART_Sleep() 或 UART_SaveConfig() 函数可能会产生意外行为。

void UART_Init(void)

说明： 根据自定义程序“配置”对话框设置来初始化或恢复组件。无需额外调用 UART_Init()，因为 UART_Start() API 会调用该函数，并且UART_Start() API是开始组件操作的首选方法。

参数： 无（void）

返回值： 无（void）

特殊情况： 所有寄存器将设置为自定义程序“配置”对话框中的值。

void UART_Enable(void)

说明： 激活硬件并开始执行组件操作。由于UART_Start() API 会调用UART_Enable()，因此无须额外调用该函数，UART_Start() API是开始组件操作的首选方法。

参数： 无（void）

返回值： 无（void）

特殊情况： 无

void UART_SaveConfig(void)

说明： 此函数会保存组件配置和易失性寄存器。它还保存 Configure（配置）对话框中定义的或通过相应 API 修改的当前组件参数值。该函数由 UART_Sleep() 函数调用。

参数： 无（void）

返回值： 无（void）

特殊情况： 所有易失性寄存器（除了 FIFO）都保存到 RAM。

void UART_RestoreConfig(void)

说明： 恢复易失性寄存器中的用户配置。

参数： 无（void）

返回值： 无（void）

特殊情况： 从RAM中加载所有易失性寄存器的值（除了 FIFO）。只有调用 UART_SaveConfig() 后才能调用此函数，否则错误数据将载入寄存器中。

定义

以下提供的定义仅供参考。定义值由组件自定义程序的设置确定。

定义	说明
UART_INIT_RX_INTERRUPTS_MASK	定义您在配置 GUI 中选择的中断源的初始配置。这是状态寄存器中的位掩码，这些位在配置时已启用为 RX 中断源。
UART_INIT_TX_INTERRUPTS_MASK	定义您在配置 GUI 中选择的中断源的初始配置。这是状态寄存器中的位掩码，这些位在配置时已启用为 TX 中断源。
UART_TXBUFFERSIZE	定义要为 TX 存储器阵列缓冲区分配的存储器大小。这不包括 FIFO 中所包含的四个字节。
UART_RXBUFFERSIZE	定义要为 RX 存储器阵列缓冲区分配的存储器大小。这不包括 FIFO 中所包含的四个字节。

UART_NUMBER_OF_DATA_BITS	定义每次数据传输的位数，用该位数可计算位时钟发生器和位计数器配置寄存器。
UART_BIT_CENTER	基于数据位的数量，该值用于计算 RX 位时钟发生器的中点，该发生器已经在 UART 启动时将载入配置寄存器中。
UART_RXHWADDRESS1	定义在配置 GUI 中选择的初始地址。该地址在 UART 启动时载入相应的硬件寄存器中。
UART_RXHWADDRESS2	定义在配置 GUI 中选择的初始地址。该地址在 UART 启动时载入相应的硬件寄存器中。

固件源代码示例

PSoC Creator 提供了许多包括原理图和示例代码的示例工程，这些工程可以在 Find Example Project（查找示例项目）对话框中找到。要获取组件特定的示例，请打开组件目录中的对话框或原理图中的组件实例。要获取通用的示例，请打开开始页或 **File**（文件）菜单中的对话框。根据需要，使用对话框中的 **Filter Options**（滤波器选项）可缩小可选项目的列表。

有关更多信息，请参考 PSoC Creator 帮助中的“查找示例项目”主题。

功能描述

UART 器件提供同步通讯功能（通常指 RS232 或 RS485）。UART 可配置用于全双工、半双工、仅 RX 或仅 TX 操作。以下部分是关于 UART 组件使用方法的概述。

默认配置

UART 默认配置为无流量控制和奇偶校验的 8 位 UART，且波特率为 57.6 Kbps。

UART 模式：全双工 UART (RX+TX)

全双工模式采用包含异步接收器和发送器的全双工 UART。该模式需要一个单时钟，以确定接收器和发送器的波特率。

UART 模式：半双工

半双工模式采用全双工 UART，但只使用全双工 UART 配置资源的一半。在此配置中，可将 UART 配置为在 RX 模式和 TX 模式中进行转换，但不能同时执行 RX 和 TX 操作。可通过调用 UART_LoadRxConfig() 或 UART_LoadTxConfig() 函数加载 RX 或 TX 配置。

在半双工模式中，**TX – FIFO 未满**状态不可用，但可以采用 **TX – On FIFO 已满**状态。由于此模式下无法使用 TX 中断，所以 TX 缓冲区大小限制在四个字节以内。

在 **Half Duplex**（半双工）模式下，Address2（地址 2）参数不适用于硬件地址匹配状态 (UART_RX_STS_ADDR_MATCH)，但是仍可以由软件使用。

半双工模式示例：

- 本示例假设名为 UART_1 组件已放入的设计中。
- 将 UART 配置为如下所述的 **Mode**（模式）：Half Duplex（半双工），**Bits per second**（波特率）：115200，**Data bits**（数据位）：8，**Parity Type**（奇偶校验类型）：None（无），**Rx Buffer Size**（Rx 缓冲区大小）：4，**Tx Buffer Size**（Tx 缓冲区大小）：4。

```
#include <device.h>

void main()
{

    uint8 recByte;
    uint8 tmpStat;

    CyGlobalIntEnable;                  /* Enable interrupts */

    UART_1_Start();                   /* Start UART */
    UART_1_LoadTxConfig();              /* Configure UART for transmitting */
    UART_1_PutString("Half Duplex Test");   /* Send message */
    /* make sure that data has been transmitted */
    CyDelay(30);    /* Appropriate delay could be used */
                    /* Alternatively, check TX_STS_COMPLETE status bit */
    UART_1_LoadRxConfig(); /* Configure UART for receiving */
    while(1)
    {
        recByte = UART_1_GetChar();     /* Check for receive byte */
        if(recByte > 0)                 /* If byte received */
        {
            UART_1_LoadTxConfig();      /* Configure UART for transmitting */
            UART_1_PutChar(recByte);    /* Send received byte back */
            do                          /* wait until transmission complete */
            {   /* Read Status register */
                tmpStat = UART_1_ReadTxStatus();
                /* Check the TX_STS_COMPLETE status bit */
            }while(~tmpStat & UART_1_TX_STS_COMPLETE);
            UART_1_LoadRxConfig();      /* Configure UART for receiving */
        }
    }
}
```

UART 模式：仅接收 RX

此模式只实现 UART 的接收器部分。该模式需要一个单时钟，以确定接收器的波特率。

UART 模式：仅发送 TX

此模式只采用 UART 中的发送器部分。该模式需要一个单时钟，以确定发送器的波特率。

UART 流量控制：无，硬件流量控制

UART 中的流量控制为现有总线提供单独的 RX 和 TX 状态指示线。若启用硬件流量控制，则在此 UART 和另一个 UART 之间可以使用“请求发送” (RTS) 线和“允许发送” (CTS) 线。CTS 线是 UART 的输入，当可以在总线上发送数据时由系统中另一个 UART 发出这个信号。RTS 线是 UART 的输出，通知总线上的另一个 UART 它已准备就绪，可以接收数据。UART 的 RTS 线与另一个 UART 的 CTS 线相连，反之亦然。这些信号线只在传输开始前有效。若在传输开始后发出或清除信号，则信号的改动只会影响下一次传输。

UART 奇偶校验：无

此设置下，UART 没有奇偶校验位。数据流格式为“启动、数据、停止”。

UART 奇偶校验：奇校验

奇校验开始时的奇偶校验位等于 1。每当在数据流中遇到 1 时，奇偶校验位便进行一次切换。完成数据传输后，则发送奇偶校验位的状态。奇校验可确保 UART 总线始终存在切换。若所有数据均为零，则发送的奇偶校验位将等于 1。数据流格式为“启动、数据、奇偶校验、停止”。奇校验是最常用的奇偶校验类型。

UART 奇偶校验：偶校验

偶校验开始时的奇偶校验位等于 0。每当在数据流中遇到 1 时，奇偶校验位便进行一次切换。完成数据传输后，则发送奇偶校验位的状态。数据流格式为“启动、数据、奇偶校验、停止”。

UART 奇偶校验：Mark/Space，数据位：9

Mark/Space 奇偶校验通常用于判定发送的数据为地址数据还是标准数据。奇偶校验位中的标记 (1) 表示发送的是数据，奇偶校验位中的空格 (0) 表示发送的是地址。在数据传输过程中，标记或空格在奇偶校验位位置发送。数据流格式为“启动、数据、奇偶校验、停止”，与其他奇偶校验模式类似，但该位在传输前由软件进行设置，而非根据数据位的值计算得出。RS485 和类似协议可使用此类奇偶校验。

TX 使用模型

固件可以使用含 UART_SET_MARK 参数的 UART_SetTxAddressMode API 函数为数据包中的第一个地址字节配置发送器。该 API 设置在控制寄存器中的 UART_CTRL_MARK 位。完成 MARK 奇偶校验设置后，发送的第一个字节为地址，其余字节将根据 SPACE 奇偶校验作为数据发送。发

送器会自动在第一个地址字节后发送数据字节。在发送其他数据包之前，控制寄存器中的 UART_CTRL_MARK 位应在至少一个时钟内清除。可通过调用含 UART_SET_SPACE 参数的 UART_SetTxAddressMode API 来完成此项操作。以下代码示例展示了此项操作。

发送定址数据包示例：

- 本示例假设名为 UART_TX 的组件已放入设计中。
- 将 UART 配置为 **Data bits**（数据位）：9，**Parity Type**（奇偶校验类型）：Mark/Space。

```
#include <device.h>

void main()
{
    UART_TX_Start();
    /*Set UART_CTRL_MARK bit in Control register*/
    UART_TX_SetTxAddressMode(UART_TX_SET_MARK);
    /*Send data packet with the address in first byte*/
    /*The address byte is character '1', which is equal to 0x31 in hex format*/
    UART_TX_PutString("1UART TEST\r");

    /*Clear UART_CTRL_MARK bit in Control register*/
    UART_TX_SetTxAddressMode(UART_TX_SET_SPACE);
}
```

RX 使用模型

接收器有四种不同模式。

1. 软件逐字节寻址（Software Byte by Byte）

在需要自定义代码时使用此模式。

状态寄存器中的 UART_RX_STS_MRKSPC 位表示地址或数据字节到达了接收器。

接收可寻址的数据包示例：

- 本示例假设名为 UART_RX 的组件已放入设计中。
- 将 UART 配置为 **Data bits**（数据位）：9，**Parity Type**（奇偶校验类型）：Mark/Space，**Interrupts**（中断）：RX - On Byte Received（RX - 接收到字节），**Address Mode**（寻址模式）：Software Byte by Byte（软件逐字节寻址），**Address#1**（地址 #1）：31。
- 将外部 ISR 连接到名称为“isr_rx”的 rx_interrupt 引脚。

```
#include <device.h>

#define STR_LEN_MAX     60u
char rx_buffer[STR_LEN_MAX];
uint8 packet_receivedRX = 0u;
```

```
void main()
{
    CyGlobalIntEnable;          /* Enable interrupts */
    isr_rx_Start();
    UART_RX_Start();

    if(packet_receivedRX == 1u)
    {
        /* add analyze here */
        packet_receivedRX = 0u;
    }
}
```

ISR 源代码示例

```
uint8 rec_status = 0u;
uint8 rec_data = 0;
static uint8 pointerRX = 0u;
static uint8 address_detected = 0u;

rec_status = UART_RX_RXSTATUS_REG;
if(rec_status & UART_RX_RX_STS_FIFO_NOTEMPTY)
{
    rec_data = UART_RX_RXDATA_REG;
    if(rec_status & UART_RX_RX_STS_MRKSPC)
    {
        if (rec_data == UART_RX_RXHWADDRESS1) /* Use any other address */
        {
            address_detected = 1;
        }
        else
        {
            address_detected = 0;
        }
    }
    else
    {
        if(address_detected)
        {
            if(pointerRX >= STR_LEN_MAX)
            {
                pointerRX = 0u;
            }
            /* Detect end of packet */
            if(rec_data == '\r')
            {   /* write null terminated string */
                rx_buffer[pointerRX++] = 0u;
                pointerRX = 0u;
                paket_receivedRX = 1u;
            }
            else
            {
                rx_buffer[pointerRX++] = rec_data;
            }
```

```
            }
        }
    }
```

2. 软件检测缓冲区寻址（Software Detect to Buffer）

在此模式下，在 RX ISR 中实现所有必需代码。

- 将 UART 配置为 **Data bits**（数据位）：9，**Parity Type**（奇偶校验类型）：Mark/Space，**RX Buffer Size**（RX 缓冲区大小）：20，**Address Mode**（地址模式）：Software Detect to Buffer（软件检测缓冲区寻址），1»Address#1（地址 #1）：31。

接收可寻址的数据包示例：

```
void main()
{
    uint8 rec_data = 0u;

    CyGlobalIntEnable;          /* Enable interrupts */
    UART_RX_Start();
    for(;;)
    {
        rec_data = UART_RX_GetChar();
        if(rec_data > 0u)
        {
            /* add analyze here */
        }
    }
}
```

3. 硬件逐字节寻址（Hardware Byte By Byte）

硬件检测过滤掉未带地址的数据包。此模式的主代码（main）类似于上面的示例。

- 将 UART 配置为 **Data bits**（数据位）：9，**Parity Type**（奇偶校验类型）：Mark/Space，**RX Buffer Size**（RX 缓冲区大小）：20，**Address Mode**（地址模式）：Hardware Byte By Byte（硬件逐字节寻址），**Address#1**（地址 #1）：31。

4. 硬件检测缓冲区寻址（Hardware Detect to Buffer）

这是无需地址字节的项目的首选模式。硬件根据一个字节地址过滤掉没有地址的数据包，主代码（main）只接收一个字节的地址。

- 将 UART 配置为 **Data bits**（数据位）：9，**Parity Type**（奇偶校验类型）：Mark/Space，**RX Buffer Size**（RX 缓冲区大小）：20，**Address Mode**（地址模式）：Hardware Detect to Buffer（硬件检测缓冲区寻址），**Address#1**（地址 #1）：31。

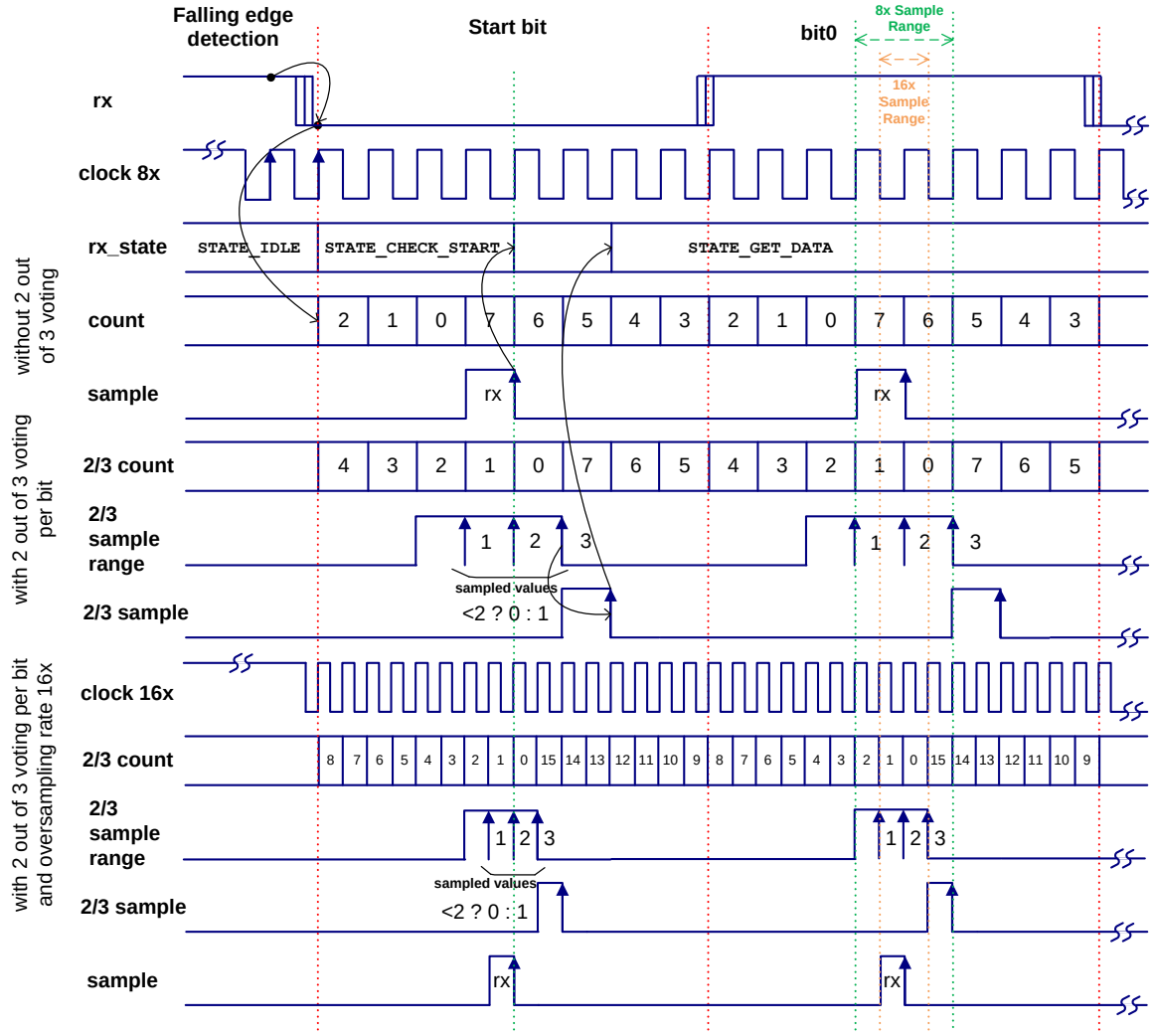
UART 停止位：一、二

停止位的数目可作为同步通信机制。在慢速的系统中，为了使接收方能够在更多数据发出前处理数据，有时停止命令需要占用两个位时间。在发送占两个位宽的停止信号期间，发送器使得接收器有额外时间解析数据字节和奇偶校验。接收器不检查第二个停止位是否有帧错误。数据流格式仍是“启动、数据、[奇偶校验]、停止”，停止位时间可配置为一个或两个位宽度。

3 取 2 表决

“3 取 2 表决”特性启用错误补偿算法。该算法实际上对每个位的中间部分过采样三次并执行多数表决，以决定该位是 0 还是 1。如果未启用“3 取 2 表决”，则每个位的中间部分仅采样一次。

启用时，该参数需要额外的硬件资源，以实现基于三个过采样时钟周期的 RX 输入的 3 位计数器。下图显示了 8 位和 16 位过采样的实现，分别包括启用和未启用“3 取 2 表决”功能的过采样。



下降沿检测的执行来识别起始位。在此检测之后，计数器开始从半位长度到 0 倒数，且接收器切换到 CHECK_START 状态。当计数器数到 0 时，已对 RX 线采样三次。如果 RX 线经验证为低

（例如 3 位中有至少 2 位为 0），则接收器进入 GET_DATA 状态。否则，接收器将返回空闲状态。8x 或 16x 过采样率的起始位检测顺序相同。

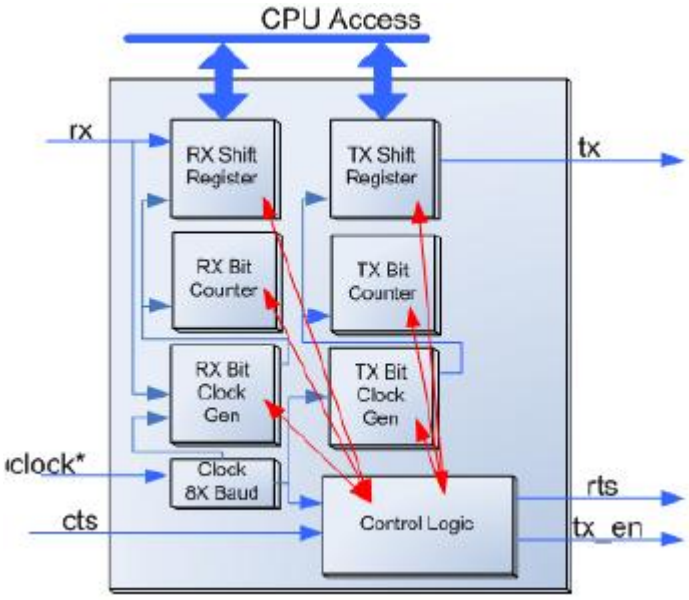
一旦接收器进入 GET_DATA 状态，RX 输入将会载入到计数器中，这个计数器在周期 4 到 6（3 个周期）上启用。该计数器将计算 RX 输入中 1 的数目。如果计数器值为 2 或更大，则该计数器的输出将为 1；否则输出将为 0。在第 7 个时钟沿上，此值将作为 RX 值被采样，送至数据路径。如果未启用表决，则在起始位检测后仅在第 5 个时钟沿上对 RX 输入进行采样，并且此后每到第 8 个时钟上升沿则进行采样。

当启用 16x 的过采样率时，表决算法将在计数器周期 8 到 10 发生，并且在第 11 个周期上，计数器输出将作为 RX 值由数据路径进行采样。如果未启用表决，则在第 9 个时钟沿上对 RX 输入进行采样，并且此后每到第 16 个时钟沿则进行采样。

框图和配置

UART 在通用数字模块（UDB） 中实现，图 1 对其进行了描述。

图 1. UDB 实现



寄存器

之前描述的 API 函数为大多数应用所需的常见运行时函数提供了支持。以下部分为高级用户提供 UART 寄存器的简要描述。

RX 和 TX 状态

状态寄存器（RX 和 TX 有独立的状态寄存器）为只读寄存器，包含为 UART 定义的各种状态位。可使用 UART_ReadRxStatus() 和 UART_ReadTxStatus() 函数访问此类寄存器值。

中断输出信号（tx_interrupt 和 rx_interrupt）是通过每个寄存器内的掩码位(masked bit)字段的 OR 运算生成的。可调用 UART_SetRxInterruptMode() 和 UART_SetTxInterruptMode() 函数设置掩码（masks）。在接收中断时，可通过 UART_GetRxInterruptSource() 和 UART_GetTxInterruptSource() 函数调用读取相应的状态寄存器，检索中断源。状态寄存器在读取后清除，因此在调用 UART_ReadRxStatus() 或 UART_ReadTxStatus() 函数之前，中断源可被保存。状态寄存器上的所有操作必须使用位字段的下列定义，这是因为构建时这些位字段可以在状态寄存器中移动。

状态寄存器中定义的位字段掩码（masks）, 都可以作为中断源包含于工程内，可以用#define 在已生成的头文件（.h）中定义这些位字段。

状态数据寄存于 UART 的输入时钟沿上。其中有些位是读取状态寄存器后就清除。它们用作 UART 中断输出，并被指定在读取后清除。所有其他位均配置为透明并且代表直接来自状态寄存器输入的数据；这些位在读取后不清除。

在以下定义中，所有配置为读取后清除的位用星号 (*) 标记：

RX 状态寄存器

定义	说明
UART_RX_STS_MRKSPC *	Mark/Space 奇偶校验位的状态。该位表示在传输的奇偶校验位中是否有标记或空格。仅当地址模式未设置为 None（无）时才实现这个检测。
UART_RX_STS_BREAK *	表示在传输中检测到中断信号。
UART_RX_STS_PAR_ERROR *	表示在传输中检测到奇偶校验错误。
UART_RX_STS_STOP_ERROR *	此位表示传输中帧错误。当停止位应为（逻辑 1）而 UART 硬件发现逻辑 0 时导致帧错误。
UART_RX_STS_OVERRUN *	表示接收 FIFO 缓冲区溢出错误。
UART_RX_STS_FIFO_NOTEMPTY	表示接收 FIFO 是否不为空。
UART_RX_STS_ADDR_MATCH *	表示已接收的字节与可用于硬件地址检测的两个地址之一匹配。仅当地址模式未设置为 None（无）时才实现这个功能。在 **Half Duplex**（半双工）模式中，仅 **Address #1**（地址 #1）。用于此检测。

TX 状态寄存器

定义	说明
UART_TX_STS_FIFO_FULL	表示发送 FIFO 已满。由于存储器中分配的发送缓冲区的状态不在硬件中表示，所以不应将其与该缓冲区混为一谈，必须在固件中对其进行检查。
UART_TX_STS_FIFO_NOT_FULL**	表示发送 FIFO 未满。
UART_TX_STS_FIFO_EMPTY	表示发送 FIFO 为空。
UART_TX_STS_COMPLETE *	表示最后的字节已从 FIFO 发送。

** — 在半双工模式中不可用

控制

控制寄存器允许您控制 UART 的常规操作。使用 UART_WriteControlRegister() 函数写入此寄存器，使用 UART_ReadControlRegister() 函数读取此寄存器。如果在自定义程序中选择了简单 UART 选项，就不会使用此控制寄存器；有关更多详情，请参见资源。当读取或写入控制寄存器时，必须使用头 (.h) 文件中定义的位字段。控制寄存器的宏定义 (#defines) 如下：

UART_CTRL_HD_SEND

用于在半双工模式中在 RX 和 TX 操作之间进行动态重新配置。该位由 UART_LoadTxConfig() 函数设置并且由 UART_LoadRxConfig() 函数清除。

UART_CTRL_HD_SEND_BREAK

若设置，则会在总线上发送一个中断信号。该位由 UART_SendBreak() 函数写入。

UART_CTRL_MARK

用于控制发送字节的 Mark/Space 奇偶校验操作。若设置，则该位表示在总线上发送的下一字节的奇偶校验位位置上将为 1（标记）。所有后续字节的奇偶校验位位置上将为 0（空格），直至该位由固件清除并复位。

UART_CTRL_PARITY_TYPE_MASK

奇偶校验类型控制是一个 2 位宽度的字段,用于为下次传输定义奇偶校验操作。此位字段是控制寄存器中的两个连续位。此位字段中的所有操作必须使用与可用奇偶校验类型有关的 宏定义（#defines）。它们为：

值	说明
UART__B_UART__NONE_REVB	无奇偶校验
UART__B_UART__EVEN_REVB	偶校验

UART__B_UART__ODD_REVB	奇校验
UART__B_UART__MARK_SPACE_REVB	Mark/Space 奇偶校验

在初始化时，该位字段在 **Parity Type**（奇偶校验类型）配置参数对话框中配置奇偶校验类型，在运行时，可以由 UART_WriteControlRegister() 函数的调用进行修改。

UART_CTRL_RXADDR_MODE_MASK

RX 地址模式控制是一个 3 位字段，用于为 UART 接收器定义预期硬件寻址操作。此位字段是控制寄存器中的三个连续位。此位字段中的所有操作必须使用与可用模式相关联的宏定义（#defines）。它们为：

值	说明
UART__B_UART__AM_SW_BYTE_BYTE	软件逐字节地址检测（Software Byte by Byte address detection）
UART__B_UART__AM_SW_DETECT_TO_BUFFER	软件检测缓冲区寻址（Software Detect to Buffer）地址检测
UART__B_UART__AM_HW_BYTE_BY_BYTE	硬件逐字节寻址地址检测（Hardware Byte by Byte address detection）
UART__B_UART__AM_HW_DETECT_TO_BUFFER	硬件检测缓冲区寻址（Hardware Detect to Buffer）地址检测
UART__B_UART__AM_NONE	无地址检测

此位字段在初始化时在 **Address Mode**（地址模式）配置参数对话框中配置，在运行时可以通过 UART_WriteControlRegister() 函数调用进行修改。

TX 数据（8 位）

TX 数据寄存器包含要发送的数据。该寄存器作为 FIFO 实现。软件状态机可以控制来自发送存储缓冲区的数据，便于处理较大量要发送的数据。为了将数据放置在总线上，所有处理数据发送的函数必须通过该寄存器（FIFO）。如果该寄存器（FIFO）中有数据并且流量控制指示数据可发送，那么数据将会在总线上发送。该寄存器 (FIFO) 一旦为空，总线上就不再发送数据，直至将其添加到 FIFO。当该 FIFO 为空时，可能会建立 DMA 使用头文件中定义的 TX 数据寄存器地址来填充该 FIFO。

值	说明
UART_TXDATA_REG	TX 数据寄存器

RX 数据

RX 数据寄存器包含已接收的数据，并作为 FIFO 实现。软件状态机器将数据从接收 FIFO 移至存储器缓冲区。通常情况下，RX 中断将指示数据已接收，此时可用 CPU 或 DMA 检索到该数据。当 FIFO 不为空时，可能会建立 DMA 通过使用头文件中定义的 RX 数据寄存器地址，在该寄存器（FIFO）中检索数据。

值	说明
UART_RXDATA_REG	RX 数据寄存器

常量

常量一般为状态寄存器和控制寄存器以及某些枚举类型定义的。前面已经为状态和控制寄存器描述了大多数的常量。然而，头文件中包含更多的常量。每个寄存器定义都需要一个指向寄存器数据或寄存器地址的指针。由于编译器具有多个字节顺序，因此 CY_GET_REGX 和 CY_SET_REGX 宏必须用于访问大于 8 位的寄存器。这些宏需要对每个寄存器使用以_PTR 结尾的定义。

必须允许适当的引擎在构建期间内放置和路由控制寄存器和状态寄存器的位。常量用于定义位的放置。对于每个状态寄存器和控制寄存器的位，有一个相关的_SHIFT 值，该值用于定义相关位在寄存器中的偏移。这些位在头文件中定义最终位掩码（the final bit mask）为_MASK 定义（_MASK 扩展仅添加到大于一位的位字段中，所有一位字段不需要_MASK 扩展）。

直流和交流电气特性

下面的值表示了预计性能，它们基于初始特性数据。

时序特性“额定路由的最大值”

当前正在收集数据。该表将在以后的版本中更新。

参数	说明		最小值	典型值	最大值	单位
f_{CLOCK}	组件时钟频率 [1]					
		全双工 UART	–	–	24	MHz
		简单 UART	–	–	44	MHz
		半双工 UART	–	–	50	MHz
		仅用于接收 RX	–	–	66	MHz

[1] 组件时钟频率的最大值取决于所选模式和其他功能。

参数	说明		最小值	典型值	最大值	单位
		仅用于 发送TX	–	–	58	MHz
t_{CLOCK}	时钟周期		$1/f_{CLOCK}$	–	–	ns
f_b	比特率		–	–	f_{CLOCK}/过采样	Mbps
T_{CLOCK}	时钟容差					
		8x 过采样	–	2.6	–	%
		16x 过采样	–	3.2	–	%
$\%_{ERR}$	错误率		–	STA [2]	–	%
t_{RES}	复位脉冲宽度		t_{CLOCK} + 5	–	–	ns
t_{CTS_TX}	CTS_N 非活动状态到 TX_EN 活动状态和 起始位在TX 上的时间		1	–	2	t_{CLOCK}
t_{TX_TXDATA}	从 TX 到 TX_DATA 的延迟		–	1	–	t_{CLOCK}
t_{TX_TXCLK}	从 TX 更改到 TX_CLK 活动状态的延迟					
		8x 过采样	–	5	–	t_{CLOCK}
		16x 过采样	–	9	–	t_{CLOCK}
t_{S_RES}	复位建立时间		5	–	–	ns
t_{RTS_RX}	RTS_N 非活动状态到 RX 数据状态的时间		–	–	STA [3]	ns
t_{RX_RXCLK} t_{RX_RXINT}	从 RX 到 RX_CLK 的延迟					
		8x 过采样	4	–	5	t_{CLOCK}
		16x 过采样	8	–	9	t_{CLOCK}
t_{RXCLK_RTS}	从最后的 RX_CLK 上升沿到 RTS_N 活动状态延迟		–	1	–	t_{CLOCK}
t_{RX_RXDATA}	从 接收（RX） 到 接收到数据（RX_DATA） 的延迟		0	–	1	t_{CLOCK}

[2] 当 PSoC Creator 无法生成精确频率时钟时，系统中会出现错误 $\%_{ERR}$。必须按本数据手册后面所述来计算该值。

[3] t_{RTS_RX} 值取决于静态时序分析结果，必须按本数据手册后面所述进行计算。

时序特性“所有数据路由（All Routing）的最大值[4]”

当前正在收集数据。该表将在以后的版本中更新。

参数	说明		最小值	典型值	最大值	单位
f_{CLOCK}	组件时钟频率[5]					
		全双工 UART	–	–	12	MHz
		简单 UART	–	–	22	MHz
		半双工 UART	–	–	25	MHz
		仅用于接收 RX	–	–	33	MHz
		仅用于 发送TX	–	–	29	MHz
t_{CLOCK}	时钟周期		$1/f_{CLOCK}$	–	–	ns
f_b	比特率		–	–	f_{CLOCK}/过采样	Mbps
T_{CLOCK}	时钟容差					
		8x 过采样	–	2.6	–	%
		16x 过采样	–	3.2	–	%
$\%_{ERR}$	错误		–	STA [6]	–	%
t_{RES}	复位脉冲宽度		t_{CLOCK} + 5	–	–	ns
t_{CTS_TX}	从CTS_N 非活动状态到 TX_EN 活动状态和 起始位在TX 上的时间		1	–	2	t_{CLOCK}
t_{TX_TXDATA}	从 TX 到 TX_DATA 的延迟			1	–	t_{CLOCK}
t_{TX_TXCLK}	从 TX 更改到 TX_CLK 活动状态的延迟					
		8x 过采样	–	5	–	t_{CLOCK}
		16x 过采样	–	9	–	t_{CLOCK}
t_{S_RES}	复位建立时间		5	–	–	ns
t_{RTS_RX}	RTS_N 非活动状态到 RX 数据		–	–	STA [7]	ns

[4] “所有数据路由的最大值”的时序值通过将“额定路由”时序值按系数 2 降额来进行计算。如果组件实例以这些速度或更低速运行，则对于此组件不应遇到时序问题。

[5] 最大器件时钟频率取决于所选模式和其他功能。

[6] 当 PSoC Creator 无法生成精确频率时钟时，系统中会出现错误 $\%_{ERR}$。必须按本数据手册后面所述来计算该值。

[7] t_{RTS_RX} 值取决于静态时序分析结果，必须按本数据手册后面所述进行计算。

参数	说明		最小值	典型值	最大值	单位
t_{RX_RXCLK} t_{RX_RXINT}	从 RX 到 RX_CLK 的延迟					
		8x 过采样	4	–	5	t_{CLOCK}
		16x 过采样	8	–	9	t_{CLOCK}
t_{RXCLK_RTS}	从最后的 RX_CLK 时钟上升沿到 RTS_N 活动状态延迟的时间		–	1	–	t_{CLOCK}
t_{RX_RXDATA}	从 RX 到 RX_DATA 的延迟		0	–	1	t_{CLOCK}

全双工 UART 选项：

模式： 全双工 UART
奇偶校验： 偶校验
API 控制已启用： 启用
流量控制： 硬件（引脚）
寻址模式： 软件逐字节寻址（Software Byte by Byte）
RX 缓冲区大小（字节） 5
TX 缓冲区大小（字节） 5
中断信号位： 13
3 取 2 表决： 启用
CRC 输出： 启用（输出引脚）
硬件 TX： 启用（输出引脚）
过采样率： 16x
复位： 输入引脚

简单 UART 选项：

模式： 全双工 UART
奇偶校验： 无
API 控制已启用： 禁用
流量控制： 无
寻址模式： 无
RX 缓冲区大小（字节） 4
TX 缓冲区大小（字节） 4
中断信号位： 无
3 取 2 表决： 禁用
CRC 输出： 禁用
硬件 TX： 禁用
过采样率： 8x
复位： 无

半双工 UART 选项：

模式：	半双工
其他所有选项与简单 UART 相同	

仅用于 RX 的选项：

模式：	仅用于接收 RX
其他所有选项与简单 UART 相同的	

仅用于 TX 的选项：

模式：	仅用于发送 TX
TxBitClkGenDP	False（要进行切换，请选择 Expression View of Advanced（高级表达式查看）选项卡）。
其他所有选项与简单 UART 相同	

图 2. TX 模式时序框图

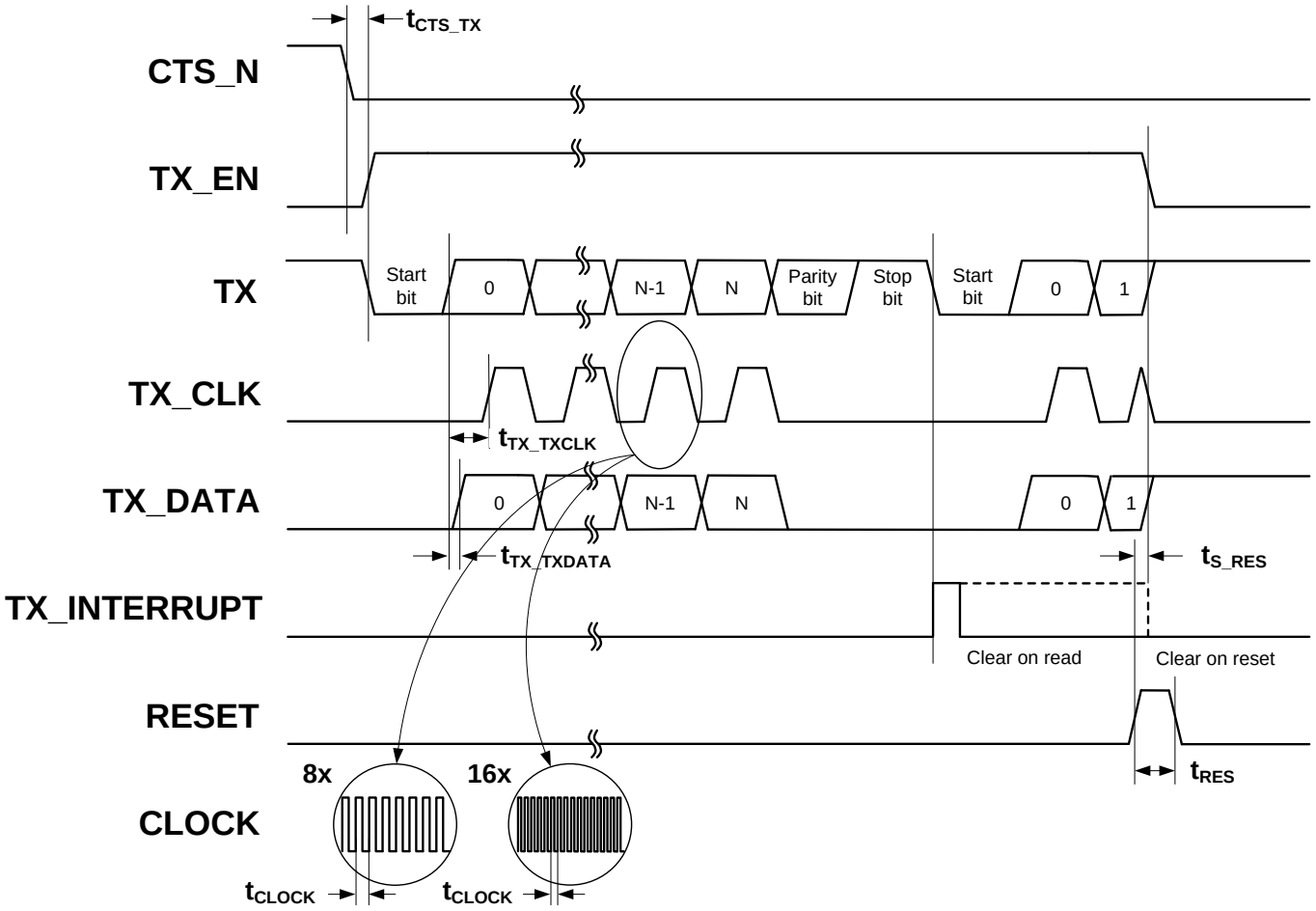
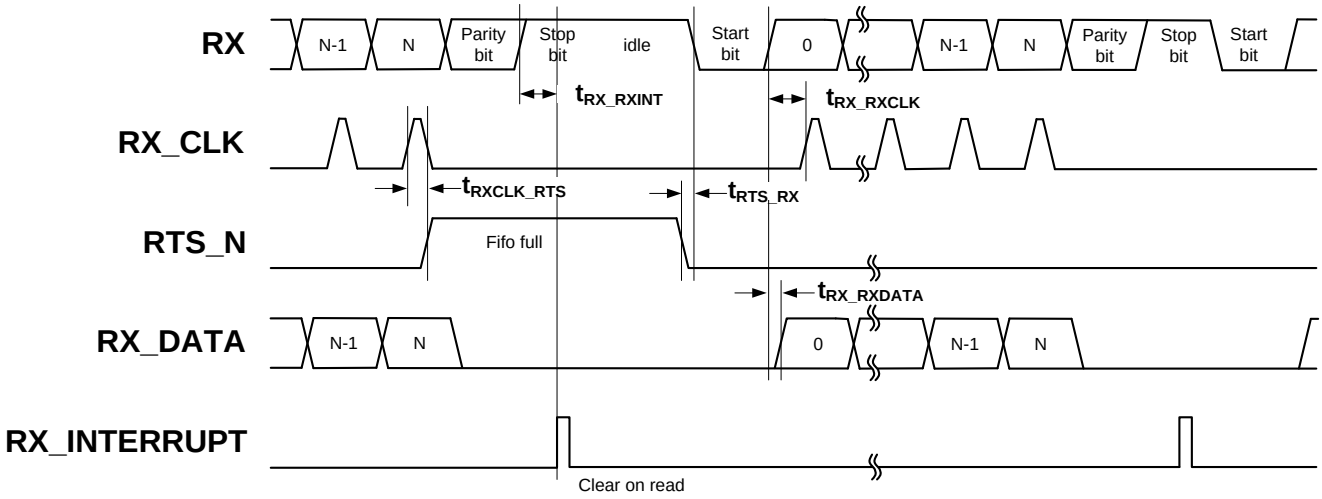


图 3. RX 模式时序框图



如何通过 STA 结果得到特性数据

额定路由最大值是通过使用静态时序分析 (STA) 进行多次测试通过而确定的。您可以用下列方法，使用 STA 结果计算设计的最大值：

f_{CLOCK} 组件时钟频率最大值作为内部时钟（如果已选择内部时钟）或命名的外部时钟显示在时钟汇总的时序结果中。下图显示了*_timing.html* 中的内部时钟限制示例。

- Clock Summary Section

Clock	Type	Nominal Frequency (MHz)	Required Frequency (MHz)	Maximum Frequency (MHz)	Violation
BUS_CLK	Sync	66.000	66.000	N/A	
ClockBlock/clk_bus	Async	66.000	66.000	N/A	
ClockBlock/dclk_0	Async	0.917	0.917	N/A	
ILO	Async	0.001	0.001	N/A	
IMO	Async	3.000	3.000	N/A	
MASTER_CLK	Sync	66.000	66.000	N/A	
PLL_OUT	Async	66.000	66.000	N/A	
UART_1_IntClock	Sync	0.917	0.917	39.588	

t_{CLOCK} 用下面的公式计算时钟周期：

$$t_{CLOCK} = \frac{1}{f_{CLOCK}}$$

f_b 比特率等于时钟频率 (f_{CLOCK}) 除以过采样率（Oversampling）。用过采样率 8x 计算最大的波特率，如以下公式所示：

$$f_b = \frac{f_{CLOCK}}{Oversampling}$$

T_{CLOCK} 使用以下方法计算时钟容差：

假设 UART 配置为 8x 过采样、禁用 3 取 2 表决、8 个数据位、无奇偶校验以及一个停止位。接收器在每个位的第五个时钟处对 RX 线采样。在低电平有效起始位的下降沿开始处识别到新帧。接收 UART 在该下降沿复位其计数器，并期望开始位的中点在 4 个时钟周期后出现，并且每个后续位的中点每 8 个时钟周期出现一次。如果 UART 时钟没有误差，则采样刚好发生在停止位的中点。但是，由于 UART 时钟一定存在误差，所以采样发生在每个位的中点之前或之后。该误差持续累加并造成停止位的最大误差。如果太早或太晚对一个位的 ½ 位周期（8 ÷ 2 = ±4 个时钟）采样，则将会在位跃变时采样并获得错误数据。对于正常的信号质量，位跃变时间等于位时间的 25%。停止位中点允许的误差将等于 UART 时钟的 ±3 个周期。

包括在该允许误差中的另一个误差是检测到开始位下降沿时的同步误差。检测到开始位后，UART 在其 8x 时钟的下一个上升沿启动。由于 8x 时钟和所接收的数据流是异步的，所以开始位的下降沿可能刚好出现在 8x 时钟上升沿之后。这意味着 UART 在同步点上本身就具有 ±1 的时钟误差。因此，允许误差减少至 ±2 个周期。

从开始位的下降沿至停止位中点的总时钟周期为 9.5 × 8 = 76。总时钟容差为 ±2 ÷ 76 × 100% = ±2.6%。

16x 过采样的时钟容差是：(16 ÷ 2 × (1 – 0.25) – 1) ÷ (9.5 × 16) × 100% = ±3.2%

该总容差必须以任一比例在接收器和发送器之间进行拆分。例如，如果 UART 总线一侧（微控制器或 PC）的器件在标准 100ppm 晶体振荡器上运行，则另一侧的器件可使用几乎所有容差预算。

$\%_{ERR}$ PSoC Creator 因 PLL 时钟频率和分频器值而无法生成 UART 所需的精确频率时钟时，系统上将显示该误差。可以将设计范围资源 (DWR) 中的差值视为 CharComp_clock 的所需频率和额定频率。该误差可通过以下等式计算得出：

$$\%_{ERR} = \frac{f_{des} - f_{nom}}{f_{des}} * 100\%$$

Type	Name	Domain	Desired Frequency	Nominal Frequency	Accuracy (%)	Tolerance (%)	Divider	Start on Reset	Source Clock
System	USB_CLK	DIGITAL	48.000 MHz	? MHz	±0	-	1	☐	IMOx2
System	Digital_Signal	DIGITAL	? MHz	? MHz	±0	-	0	☐	
System	XTAL_32KHZ	DIGITAL	32.768 kHz	? MHz	±0	-	0	☐	
System	XTAL	DIGITAL	25.000 MHz	? MHz	±0	-	0	☐	
System	ILO	DIGITAL	? MHz	1.000 kHz	-50, +100	-	0	☑	
System	IMO	DIGITAL	3.000 MHz	3.000 MHz	±1	-	0	☑	
System	BUS_CLK (CPU)	DIGITAL	? MHz	66.000 MHz	±1	-	1	☑	MASTER_CLK
System	MASTER_CLK	DIGITAL	? MHz	66.000 MHz	±1	-	1	☑	PLL_OUT
System	PLL_OUT	DIGITAL	66.000 MHz	66.000 MHz	±1	-	0	☑	IMO
Local	UART_1_IntClock	DIGITAL	921.600 kHz	916.667 kHz	±1	±5	72	☑	Auto: MASTER_CLK

Pins | Clocks | Interrupts | DMA | System | Directives | Flash Security

例如，对于配置为 115200 位/秒和 8x 过采样的 UART，系统需要 921.6-kHz 的时钟。PLL 配置为 66 MHz 时，DWR 使用 72 分频并生成 66000 ÷ 72 = 916,667-kHz 的时钟。该示例中的误差是：

(921.6 – 916,667) ÷ 921.6 × 100 = ~0.5%

该误差与时钟精度误差之和不应超过时钟容差 (T_{CLOCK})，否则数据将出现误差。

时钟精度取决于所选的 IMO 时钟。对于 3-MHz 来说，时钟精度等于 ±1%。总误差为：0.5 + 1 = 1.5%，小于 8x 过采样的最小时钟容差 (2.6%)

其他 IMO 时钟设置具有较大的精度误差，不建议应用于 UART。

PSoC 5 芯片的最小 IMO 时钟精度为±5%；因此，在这种情况下应使用外部晶振时钟。

t_{CTS_TX} 该参数的特性基于 UART 实现分析（ UART implementation analysis）。与 f_{CLOCK} 时钟同步的状态机检查 CTS_N 信号的下降沿 并在一个时钟延迟后将 TX_EN 设置为高。TX_EN 信号在输出上具有其他同步，以消除可能的短时脉冲。这增加了一个时钟延迟。移位寄存器在 TX_EN 信号变为高电平的同时开始将 TX 数据移出。

t_{TX_TXCLK} 根据 UART 实现分析（ UART implementation analysis），从 TX 输出至 TX_CLK 的延迟时间等于半个位的长度，并在 TX_DATA 信号的中点延迟一个时钟。

$$t_{TX_TXCLK} = t_{CLOCK} * \left(\frac{\text{Oversampling}}{2} + 1 \right)$$

t_{TX_TXDATA} 该参数的特性基于 UART 实现分析（ UART implementation analysis）。TX 信号还与 TX_DATA 输出上的 f_{CLOCK} 同步，因此，在这些信号之间存在一个时钟延迟。

t_{RES} 该参数的特性基于 UART 实现分析（ UART implementation analysis）和 STA 结果。复位输入是同步的，需要至少一个组件时钟的上升沿。应添加建立时间以保证不丢失复位信号。

$$t_{RES} = t_{CLOCK} + t_{S_RES}$$

t_{S_RES} RESET 激活时间是内部寄存器引脚的路由延迟时间加上时钟输出的延迟时间。这由 STA 结果提供，如下所示：

- **Register to Register Section**

  - **Setup Subsection**

    - **Source Clock : BUS_CLK : Positive edge(Required Frequency 33 MHz)**

      - **Destination Clock : UART_1_IntClock : Positive edge(Required Frequency 33 MHz)**

Path Delay Requirement : 30.303ns(33 MHz)

Source	Destination	FMax (MHz)	Delay (ns)	Slack (ns)	Violation
RESET(0)/fb	\UART 1:BUART:reset reg\/main 0	63.800	15.674	14.629	

- **Clock To Output Section**

- **UART_1_IntClock**

Source	Destination	Delay (ns)
\UART 1:BUART:reset reg\/q	TX INT(0) PAD	69.350
\UART 1:BUART:reset reg\/q	RX INT(0) PAD	56.266
\UART 1:BUART:reset reg\/q	RTS N(0) PAD	40.453
Net 4/q	TX(0) PAD	29.383

t_{RX_RXCLK} 基于 UART 实现分析（UART implementation analysis），从 RX 至 RX_CLK 的延迟时间等于半个位的长度，并在 RX_DATA 信号的中点最多延迟一个时钟。

$$t_{RX_RXCLK} = t_{CLOCK} * \left(\frac{\text{Oversampling}}{2} + 1 \right)$$

t_{RX_RXINT} 当停止位在 RX_CLK 时被接收， RX_INTERRUPT 信号生成

t_{RX_RXDATA} RX 信号还与 RX_DATA 输出上的 **f_{CLOCK}** 同步，因此，在这些信号之间最多存在一个时钟延迟。

t_{RXCLK_RTS} 从最后一个 RX_CLK 的上升沿到 RTS_N 活动状态的延迟时间。4 位 FIFO 已满时，发生这种情况。FIFO 一旦已满，硬件就会立即自动设置 RTS_N 信号。从最后的 RX_CLK 上升沿开始的一个组件时钟周期延迟内，加载 FIFO 。

t_{RTS_RX} RTS_N 非活动状态至 RX 数据的延迟时间等于：

$$t_{RTS_RX} = t_{PD_RTS} + RTS_{PD_PCB} + t_{CTS_TX\text{（发送器）}} + RX_{PD_PCB} + t_{S_RX}]$$

其中：

t_{PD_RTS} RTS_N 至引脚的路径延迟。这由 STA 结果时钟至输出部分提供，如下所示。

- **Clock To Output Section**

- **UART_1_IntClock**

Source	Destination	Delay (ns)
\UART 1:BUART:rx state stop1 reg\/q	RX INT(0) PAD	39.902
\UART 1:BUART:sTX:TxShifter:u0\/f0 blk stat comb	TX INT(0) PAD	37.275
\UART 1:BUART:sRX:RxShifter:u0\/f0 blk stat comb	RTS N(0) PAD	27.550
Net 16/q	TX EN(0) PAD	24.813
Net 21/q	TX CLK(0) PAD	24.799
Net 4/q	TX(0) PAD	24.625
Net 20/q	TX DATA(0) PAD	23.648
Net 22/q	RX DATA(0) PAD	23.186
Net 23/q	RX CLK(0) PAD	22.961

RTS_{PD_PCB} 是从接收器组件的 RTS_N 引脚至发送器器件 CTS_N 引脚的 PCB 路径延迟。

发送器的 t_{CTS_TX} 的值必须来自“发送器数据手册”。

RX_{PD_PCB} 是从发送器器件的 TX 引脚至接收器组件的 RX 引脚的 PCB 路径延迟。

t_{S_RX} 是 RX 信号的路径延迟时间。这由 STA 结果寄存器至寄存器部分提供，如下所示。

- **Register to Register Section**
  - **Setup Subsection**
    - **Source Clock : BUS_CLK : Positive edge(Required Frequency 33 MHz)**
      - **Destination Clock : UART_1_IntClock : Positive edge(Required Frequency 16.5 MHz)**

Path Delay Requirement : 30.303ns(33 MHz)

Source	Destination	FMax (MHz)	Delay (ns)	Slack (ns)	Violation
RX(0)/fb	\UART_1:BUART:rx_load_fifo\/main_11	39.584	25.263	5.040	
RX(0)/fb	\UART_1:BUART:rx_state_2\/main_1	40.780	24.522	5.781	
RX(0)/fb	\UART_1:BUART:rx_markspace_pre\/main_4	41.750	23.952	6.351	
RX(0)/fb	\UART_1:BUART:rx_state_3\/main_7	43.303	23.093	7.210	
RX(0)/fb	\UART_1:BUART:rx_state_2\/main_2	44.377	22.534	7.769	
RX(0)/fb	\UART_1:BUART:rx_state_2\/main_0	45.652	21.905	8.398	
RX(0)/fb	\UART_1:BUART:rx_load_fifo\/main_10	46.955	21.297	9.006	
RX(0)/fb	\UART_1:BUART:rx_break_detect\/main_0	54.702	18.281	12.022	
RX(0)/fb	\UART_1:BUART:rx_last\/main_0	54.702	18.281	12.022	
RX(0)/fb	\UART_1:BUART:rx_markspace_pre\/main_0	54.702	18.281	12.022	

组件更改

本节介绍组件与以前版本相比的主要更改。

版本	更改说明	更改/影响原因
2.10	将 UART_PutString() API 的参数类型从 uint8* 更改为 char*。	此 API 的常见用法是与嵌入式字符串一起用作参数：UART_PutString(“Hello World”)。应将“char”类型用于此用途，而不会出现编译器警告。
	UART_ClearRxBuffer()/ UART_ClearTxBuffer() API 还用于清除硬件 FIFO。	需要清除硬件 FIFO，以保证没有更多数据等待进行接收/传送。
	修复了 UART_SendBreak() API 的参数拼写错误。UART_WAIT_FOR_COMLETE_REINT 更改为 UART_WAIT_FOR_COMPLETE_REINT。	兼容错误修复。
	对所有 UART API 添加了 CYREENTRANT 关键词（当它们包含在 .cyre 文件中时）。	并非所有 API 都是真正重新进入。组件 API 源文件中的注释指出了可作为候选的函数。 需要此更改为采用安全方式使用（通过标志或关键节防止并发调用）并且不是重新进入的函数消除编译器警告。
	更新了地址模式功能。	升级这些模式是为了自动跳过未寻址的数据包。
	修复了使用内部 RX 缓冲区时的硬件流量控制模式。在内部缓冲区溢出时，RX ISR 中的代码停止从 FIFO 读取数据。因此，RTS 信号保持发送器 UART。	数据从硬件 FIFO 读取，并移动到 s/w 缓冲区（无论 s/w 缓冲区是否溢出）。
	更新组件内部时钟至 cy_clock_v1_60 。	时钟 v1_60 是最新组件版本。
	将 RX 和 TX 缓冲区大小最小值限制为 4 Byte。	UART 始终使用 4 字节 FIFO 作为缓冲区。
	对数据手册进行了少量编辑和更新	
2.0.a	对数据手册进行了少量编辑和更新	

版本	更改说明	更改/影响原因
2.0	tx_en 输出已寄存（tx_en output registered）	任何组合输出都可能出现短时脉冲，具体取决于放置和信号之间的延迟。 要消除短时脉冲，应将输出寄存。
	复位输入已寄存（Reset input registered）。	使用复位输入时，寄存可提高最大波特率。
	向数据手册中添加了特性数据	
	对数据手册进行了少量编辑和更新	
1.50	添加了睡眠/唤醒和初始化/启用 API 函数。	为支持低功耗模式并提供常用接口，以单独控制大多数组件的初始化和启用。
	中断信号具有长度选择性（11 到 14 位），并且可以向 SendBreak 函数添加参数。	由于未指定 UART 的中断信号长度，因此提供了 11 到 14 位中断信号长度选择。
	添加了 16x 过采样模式。	16x 过采样模式减少了较高速度时对误差的抖动影响。
	从 **Parity Type**（奇偶校验类型）选择删除了软件选项，改为添加了 **API control enabled**（启用 API 控制）复选框。	这样便可以在需要奇偶校验 API 控制时选择默认值。 如果在选择了此选项的情况下从 1.20 版 UART 组件更新，则建议在 1.50 版中选择“None（无）”奇偶校验选项。

© 赛普拉斯半导体公司，2012。此处所包含的信息可能会随时更改，恕不另行通知。除赛普拉斯产品的内嵌电路之外，赛普拉斯半导体公司不对任何其他电路的使用承担任何责任。也不根据专利或其他权利以明示或暗示的方式授予任何许可。除非与赛普拉斯签订明确的书面协议，否则赛普拉斯产品不保证能够用于或适用于医疗、生命支持、救生、关键控制或安全应用领域。此外，对于可能发生运转异常和故障并对用户造成严重伤害的生命支持系统，赛普拉斯不授权将其产品用作此类系统的关键组件。若将赛普拉斯产品用于生命支持系统中，则表示制造商将承担因此类使用而招致的所有风险，并确保赛普拉斯免于因此而受到任何指控。

PSoC® 是赛普拉斯半导体公司的注册商标，PSoC Creator™ 和 Programmable System-on-Chip™ 是赛普拉斯半导体公司的商标。此处引用的所有其他商标或注册商标归其各自所有者所有。

所有源代码（软件和/或固件）均归赛普拉斯半导体公司（赛普拉斯）所有，并受全球专利法规（美国和美国以外的专利法规）、美国版权法以及国际条约规定的保护和约束。赛普拉斯据此向获许可者授予适用于个人的、非独占性、不可转让的许可，用以复制、使用、修改、创建赛普拉斯源代码的派生作品、编译赛普拉斯源代码和派生作品，并且其目的只能是创建自定义软件和/或固件，以支持获许可者仅将其获得的产品依照适用协议规定的方式与赛普拉斯集成电路配合使用。除上述指定的用途之外，未经赛普拉斯的明确书面许可，不得对此类源代码进行任何复制、修改、转换、编译或演示。

免责声明：赛普拉斯不针对此材料提供任何类型的明示或暗示保证，包括（但不仅限于）针对特定用途的适销性和适用性的暗示保证。赛普拉斯保留在不做出通知的情况下对此处所述材料进行更改的权利。赛普拉斯不对此处所述之任何产品或电路的应用或使用承担任何责任。对于可能发生运转异常和故障并对用户造成严重伤害的生命支持系统，赛普拉斯不授权将其产品用作此类系统的关键组件。若将赛普拉斯产品用于生命支持系统中，则表示制造商将承担因此类使用而招致的所有风险，并确保赛普拉斯免于因此而受到任何指控。

产品使用可能受适用的赛普拉斯软件许可协议限制。