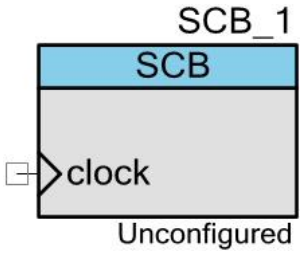
PSoC 4 串行通信模块（SCB）

1.20



特性

- 预配置组件：
  - 工业标准 NXP® I²C 总线接口
  - 支持 Motorola、Texas Instruments 标准 SPI 主设备与从设备功能
  - 支持标准 UART TX 与 RX 功能，支持 SmartCard（ISO7816）读卡器与 IrDA 协议
  - EZ I²C 模式可以模拟通用 I²C EEPROM 接口
- 支持从深度睡眠模式唤醒
- 运行时动态重配置
- 支持 I²C Bootloader 功能

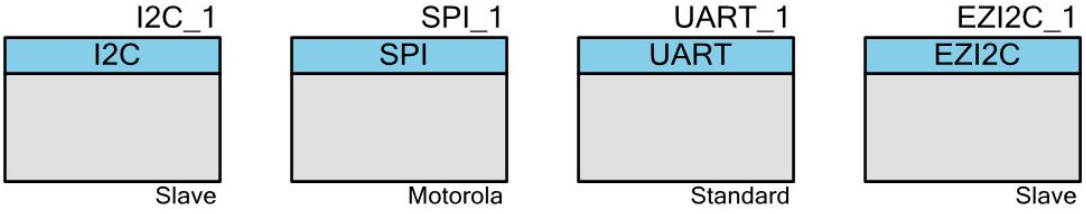
概述

PSoC 4 SCB 组件是一个多功能硬件模块，它能配置成以下组件。在 PSoC Creator 组件目录中，每个组件作为预配置原理图宏，其名称为“SCB Mode”（SCB 模式）。

注意：PSoC 4000 器件只支持 I²C 模式。UART 或 SPI 模式被禁用。

点击下面的链接，可以跳转到相应的章节：

I2C **SPI** **UART** **EZI2C**

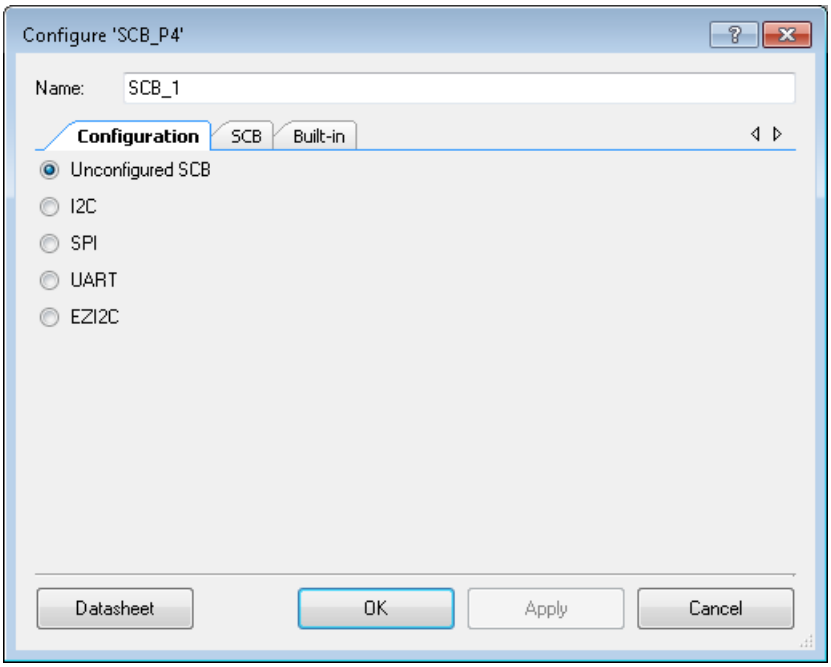


在组件目录中还有未配置的 SCB 组件条目。

何时使用 SCB 组件

SCB 能够配置成不同的工作模式： I2C、SPI、UART 以及 EZI2C。另外，SCB 可在创建时不予以配置，在运行时采用应用编程接口（API） 配置成任意模式。所有可配置相均可在运行时进行设定。

下图显示的是通过组件配置对话框选择相应的工作模式。

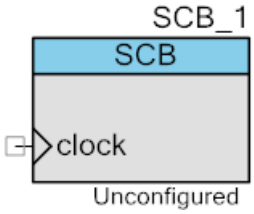


注意：PSoC 4000 器件只支持 I^2C 模式。UART 或 SPI 模式被禁用。

采用对话框配置 SCB 的工作模式是最常用，也是最简便的将 SCB 配置成不同功能组件的配置方法。Unconfigured SCB （未配置的 SCB）选项可用于创建多应用设计， 此时 SCB 需要动态重配置，其具体用途是未知的。

未配置的 SCB

SCB 可在运行时进行配置，从未配置模式跳转到任一其他工作模式。



输入/输出接口

本部分描述了 SCB 组件的各种输入/输出接口。I/O 列表中的星号（*）表示，在 I/O 说明部分中所列出的情况下，该 I/O 可能不可见。

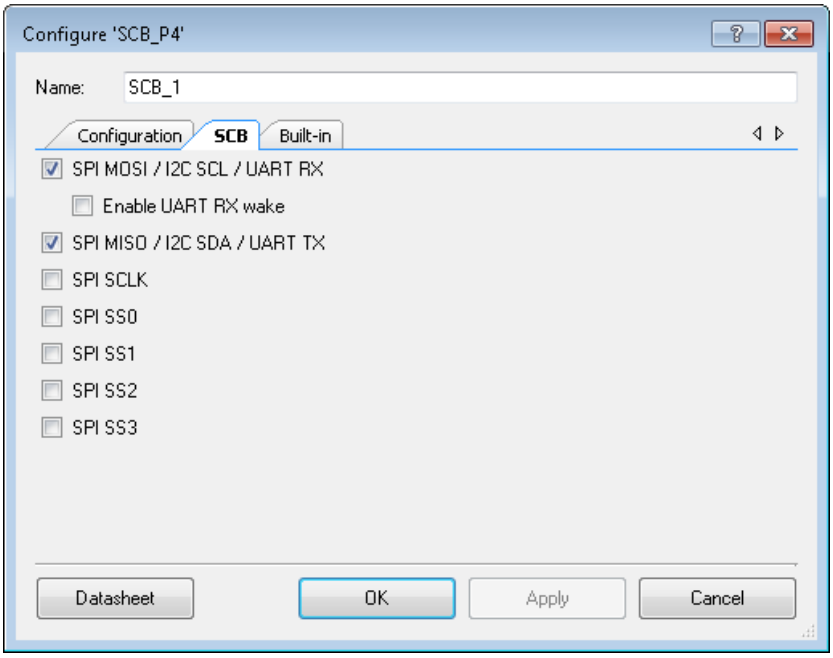
时钟 — 输入

时钟负责运行此模块。在未配置模式，该输入一直可见。对于其他工作模式，该选项为组件提供了使用内部时钟还是外部时钟的选择。

在组件中通信接口（例如 TX/RX，I2C/SDA 等）引脚都被隐藏，因为这些引脚只能连接到特定的 IO，不具备通用信号的可布线性。了解更多信息，请参见芯片*技术参考手册（TRM）*中*“I/O 系统”*部分的内容。

注意：被隐藏输出引脚的输入缓冲器被禁止，以避免在低功耗模式下存在输入漏电流。读取这些引脚的状态返回值始终为零。要获取当前状态，读取状态前必须先使能输入缓冲区。

SCB 选项卡



注意：PSoC 4000 器件只支持 I^2C 模式。这些器件使用了下列引脚名称：

- SPI MOSI / I2C SCL / UART RX 引脚名称为 I2C SCL
- SPI MISO / I2C SDA / UART TX 引脚名称为 I2C SDA
- SPI SCLK、SPI SS0 到 SS3 引脚不可用，“Enable UART RX wake”（使能 UART RX 唤醒）的选项也不可用

SCB 选项卡选项可以选择该模块运行时需要使用的引脚。勾选复选框可以启用被选引脚。

勾选 **Enable UART RX wake** 将向 RX 引脚添加中断，以实现 UART 唤醒功能。 该选项限制了 RX 引脚所在的端口上的其他引脚的中断处理。

未配置操作模式

在未配置模式下执行操作前，请选择组件所能支持的接口。使用 **SCB** 参数选项卡选择需要使用的接口引脚，SPI MOSI / I2C SCL / UART RX 表示当 SCB 被配置为 SPI 组件时该引脚作为 MOSI 使用；配置成 I^2C 组件时它作为 SCL；配置成 UART 组件时它作为 RX。

引脚选择完成后，必须将时钟组件连接到 SCB 时钟输入。该时钟频率和 SCB 过采样配置决定了接口的实际操作速度。时钟频率可以通过 Clock 组件或者该组件提供的 API 来设置。

要想使用 UART 和 I^2C 接口，请选择 **SCB** 选项卡上下面引脚：

- SPI MOSI / I2C SCL / UART RX 引脚

- SPI MISO / I2C SDA / UART TX 引脚

接口数据速率的配置

通过使用下面公式，可以计算得出每个接口的数据速率（其中，f_{SCBCLK} 是连接 SCB 的组件的时钟频率）：

数据速率 = (f_{SCBCLK} / 过采样值)

在**主设备**模式下，过采样因子的值是时钟低相位过采样值和时钟高相位过采样值之和。

f_{SCBCLK} = 数据速率 * 过采样值

为所有接口定义过采样值。请参考 I2C / SPI / UART / EZI2C 参数这一节，了解有关数据速率和过采样的详细信息。

要更改 f_{SCBCLK} 时钟频率，必须更改时钟分频器。时钟组件提供了 API 用以执行该操作。

$Div_{SCBCLK} = f_{HFCLK} / f_{SCBCLK}$

注意： Div_{SCBCLK} 必须为整数值。

下面是为 I^2C 和 UART 计算 Div_{SCBCLK} 值的示例：

设计 f_{HFCLK} 的 HFCLK 配置 = 24 MHz；

需要的 I2C 从设备数据速率 = 100 kbps，并且 UART 波特率 = 115200 bps；

计算 I^2C 从设备的 Div_{SCBCLK} 时，过采样的默认值被选为 16。

f_{SCBCLK} = 数据速率 * 过采样值 = 100000 * 16 = 1.6MHz

$Div_{SCBCLK} = f_{HFCLK} / f_{SCBCLK}$ = 24 MHz / 1.6MHz = 15

计算 UART 从设备的 Div_{SCBCLK} 值时，过采样的默认值被选为 16。

f_{SCBCLK} = 数据速率 * 过采样值 = 115200 * 16 = ~1,843 MHz

$Div_{SCBCLK} = f_{HFCLK} / f_{SCBCLK}$ = 24 MHz / 1, 843 MHz = ~13

对于 UART，f_{SCBCLK} 的准确度非常重要，它能确保 UART 的正常操作，因此通过 f_{HFCLK} 和 Div_{SCBCLK} 来计算 f_{SCBCLK} 的实际值。

f_{SCBCLK} 的实际值 = f_{HFCLK} / Div_{SCBCLK} = 24 MHz / 13 = ~1,846 Hz。

需要计算出预计的 f_{SCBCLK} 实际值： (1,843MHz – 1,846 MHz) / 1,843 MHz = ~0.2%

HFCLK 的准确度为±2%时，总误差为：0.2 + 2 = 2.2%。总误差值小于 5%，这样可确保 UART 的准确操作。

通过计算得到下面的值：

- I2C 从设备的数据速率 = 100 kbps：过采样值 = 16，Div_{SCBCLK} = 15；
- UART 波特率 = 115200 bps：过采样值 = 16，Div_{SCBCLK} = 13；

运行时动态重配置

SCB 组件配置对话框用于配置组件。选择一个接口并设置其配置。如果要在组件运行时实现配置的更改，则需要为每个接口提供配置结构。这些结构提供的多个字段与配置对话框中的选项相匹配。

为选定的接口分配配置结构并对其进行配置。指向该结构的指针会传递到所选接口相关的初始化函数。下面是在 I^2C 模式下用于配置的函数实例：

```
void SCB_I2CInit(SCB_I2C_INIT_STRUCT *config)
```

下面实例提供了下列配置所需的配置结构：

- I2C 从设备，数据速率为 1.6 kbps，从设备地址为 0x08
- 标准 UART RX+TX ，TX 和 RX 的缓冲区大小为 10（采用软件缓冲区）

SCB 组件的实例名称为“SCB”，时钟组件的实例名称为“SCBCLK”。

```
#define SCB_BUFFER_SIZE     (10u)

/* Common buffers  or I2C and UART */
uint8 bufferRx[SCB_BUFFER_SIZE + 1u];/* RX software buffer requires one extra
entry for correct operation in UART mode */
uint8 bufferTx[SCB_BUFFER_SIZE];     /* TX software buffer */

/* Use defines from I2C customizer setup */
const SCB_I2C_INIT_STRUCT configI2C =
{
    SCB_I2C_MODE_SLAVE, /* mode: slave */
    0u,     /* oversampleLow: N/A for slave, SCBCLK provides oversampling */
    0u,     /* oversampleHigh: N/A for slave, SCBCLK provides oversampling */
    1u,     /* enableMedianFilter: enable */
    0x08u, /* slaveAddr: slave address */
    0xFEu, /* slaveAddrMask: signle slave address */
    0u,     /* acceptAddr: disable */
    0u      /* enableWake: disable */
};

const SCB_UART_INIT_STRUCT configUart =
{
```

```
    SCB_UART_MODE_STD, /* mode: Standard */
    SCB_UART_TX_RX,    /* direction: RX + TX */
    8u,     /* dataBits: 8 bits */
    SCB_UART_PARITY_NONE, /* parity: None */
    SCB_UART_STOP_BITS_1, /* stopBits: 1 bit */
    16u,    /* oversample: 16u */
    0u,     /* enableIrdaLowPower: disable */
    1u,     /* enableMedianFilter: enable */
    0u,     /* enableRetryNack: disable */
    0u,     /* enableInvertedRx: disable */
    0u,     /* dropOnParityErr: disable */
    0u,     /* dropOnFrameErr: disable */
    0u,     /* enableWake: disable */
    SCB_BUFFER_SIZE, /* rxBufferSize: software buffer 10 bytes */
    bufferRx, /* rxBuffer: RX software buffer enable */
    SCB_BUFFER_SIZE, /* txBufferSize: software buffer 10 bytes */
    bufferTx, /* txBuffer: TX software buffer enable */
    0u,     /* enableMultiproc: disable */
    0u,     /* multiprocAcceptAddr: disable */
    0u,     /* multiprocAddr: N/A for this configuration */
    0u,     /* multiprocAddrMask: N/A for this configuration */
    1u,     /* enableInterrupt: enable to process software buffer */
    SCB_INTR_RX_NOT_EMPTY,   /* rxInterruptMask: enable NOT_EMPTY for RX software
buffer operations */
    0u, /* rxTriggerLevel: N/A for this configuration */
    0u, /* txInterruptMask: NOT_FULL is enabled when there is data to transmit */
    0u  /* txTriggerLevel: N/A for this configuration */
};
```

在下面的示例中执行了一个函数，该函数根据先前的操作模式更改了 SCB 配置，并返回配置更改的状态。该函数指的是提供给 I^2C 和 UART 的配置结构。

```
/* The clock divider value which is written into the register has to be less for
one from calculated */
#define SCBCLK_I2C_DIVIDER  (14u)   /* I2C Slave: 100 kbps with OVS = 16. Required
SCBCLK = 1.6 MHz, Div = 15 */
#define SCBCLK_UART_DIVIDER (12u)   /* UART: 115200 kbps with OVS = 16. Required
SCBCLK = 1.846 MHz, Div = 13 */
/* Operation mode: I2C slave or UART */
#define OP_MODE_UART    (1u)
#define OP_MODE_I2C     (2u)

cystatus SetScbConfiguration(uint32 opMode)
{
    cystatus status = CYRET_SUCCESS;

    if (OP_MODE_I2C == opMode)
    {
        SCB_Stop(); /* Disable component before configuration change */

        /* Change clock divider */
        SCBCLK_Stop();
        SCBCLK_SetFractionalDividerRegister(SCBCLK_I2C_DIVIDER, 0u);
        SCBCLK_Start();
```

```
            /* Configure to I2C slave operation */
            SCB_I2CSlaveInitReadBuf (bufferTx, SCB_BUFFER_SIZE);
            SCB_I2CSlaveInitWriteBuf(bufferRx, SCB_BUFFER_SIZE);
            SCB_I2CInit(&configI2C);

            SCB_Start(); /* Enable component after configuration change */
        }
        else if (OP_MODE_UART == opMode)
        {
            SCB_Stop(); /* Disable component before configuration change */

            /* Change clock divider */
            SCBCLK_Stop();
            SCBCLK_SetFractionalDividerRegister(SCBCLK_UART_DIVIDER, 0u);
            SCBCLK_Start();

            /* Configure to UART operation */
            SCB_UartInit(&configUart);

            SCB_Start(); /* Enable component after configuration change */
        }
        else
        {
            status = CYRET_BAD_PARAM; /* Uknowns operation mode - no actions */
        }

        return (status);
    }
```

注意：更改配置前，必须先禁止 SCB 组件。

注意：未配置模式下，SCB_Init()函数不会初始化组件。必须调用 SCB_**Mode**Init() API。

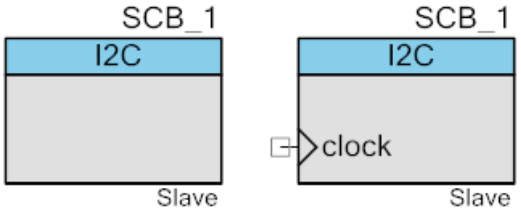
API 名称

某些 API 包含了特定接口前缀作为其名称的一部分。只有将组件配置为采用该接口时，这些 API 才能准确运作。例如，SCB_I2CSlaveStatus()函数属于 I^2C 接口。

对于那些多个接口共用的 API ，API 名称包含了每一个接口。例如，SCB_SpiUartWriteTxData() 属于 SPI 或 UART 接口。

不属于特定接口的 API 不会包含接口作为前缀。例如，SCB_Enable() 或 SCB_EnableInt()。

I2C



I²C 总线是 Philips 基于行业标准开发的两线硬件接口。主设备在 I²C 总线上启动所有通信，并为所有从设备提供时钟。在一个单板或小系统上互联多个设备时，I²C 是一种理想的解决方案。

该组件支持 I²C 从设备、主设备、多主设备与多主从设备配置。

该组件支持高达 1000 kbps 的标准时钟速率。它与 NXP 网站 www.nxp.com 上 NXP I²C 总线规范[1]中所定义的 I²C 标准、快速和超快模式的设备相兼容。组件与其它第三方的 I2C 主从/设备相兼容。

输入/输出接口

本部分描述了 SCB 组件的各种输入/输出接口。I/O 列表中的星号（*）表示，在 I/O 说明部分中所列出的情况下，该 I/O 可能不可见。

时钟 — 输入*

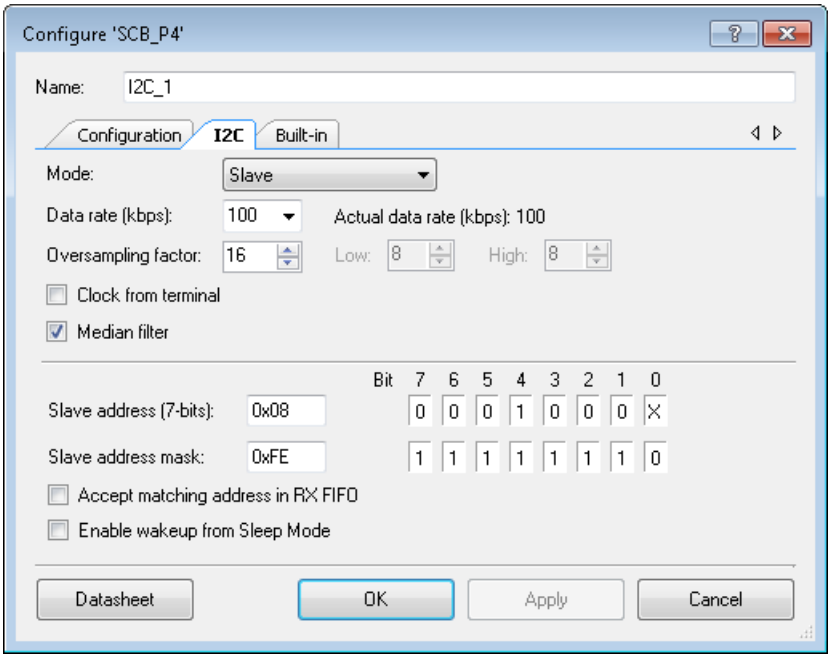
时钟负责运行此模块。只有将 **Clock from terminal**（终端时钟）复选框勾选上时，时钟接口才可见。

在组件中，通信接口引脚（SCL/SDA ）不可见，因为这些引脚只能连接到特定的 IO，不具备可路由性。了解更多信息，请参见芯片*技术参考手册（TRM）*中 *“I/O 系统”* 部分的内容。

1. 请参考 NXP 网站 www.nxp.com 上刊登的 *I²C 总线规格*（2012 年 10 月，修订版 5）

I2C 参数

将 SCB 组件拖放到您的设计上，双击它可打开 **Configure**（配置）对话框。



I2C 选项卡包含下列参数：

Mode（模式）

此选项确定组件支持的模式：从设备、主设备、多主设备或多主从设备。

- **从设备** — 仅限从设备操作（默认）
- **主设备** — 仅限主设备操作
- **多主设备** — 支持总线上多个主设备
- **多主从设备** — 从设备与多主设备同步操作

Data rate（数据速率）

此参数用于设置高达 1000 kbps 的 I^2C 数据速率值（对于 PSoC 4000 系列，该值为 400 kbps）；实际速率可能会基于可用的时钟速度和分频器的分频范围而有所不同。标准数据速率为 50、100（默认值）、400 和 1000 kbps。如果设置了 **Clock from terminal**（终端时钟），则可忽略 **Data rate**（数据速率）参数；输入时钟频率与 **Oversampling factor** （过采样因子）共同决定实际数据速率。

Actual data rate （实际数据速率）

实际数据速率显示组件运行时实际的数据速率。所选数据速率可能和实际数据速率不一样。影响实际数据速率计算的因子包括： 过采样因子、HFCLK 时钟和内部或外部组件时钟的准确度。

Oversampling factor（过采样因子）

此参数定义了 I^2C SCL 时钟的过采样因子，即一个 I^2C SCL 时钟周期内组件时钟的数量(时钟周期比)。**Oversampling factor**（过采样因子）用于计算出获取所定义的 **Data rate** 的过采样量时所需的内部组件时钟频率。

- 在**主设备**模式下，**Oversampling factor**（过采样因子）的值是低和高过采样值之和。这些值均被写入到寄存器内，并用以生成 I^2C 时钟的低和高相位。
- 在**从设备**模式下，只有组件时钟源起着重要的作用。这里需要提供所需的过采样值。

过采样因子的最大值为 32；最小值则取决于 **Median filter**（中值滤波器）的设置。未选取中值滤波器 — 12；选取了中值滤波器 — 14。默认值为 16。

时钟低相位过采样因子

此参数定义了 I^2C SCL 时钟相位中的低过采样因子，即一个低周期 I^2C SCL 时钟周期内组件时钟的数量。它仅使用于 **Master**（主设备）模式。过采样因子的最小值取决于 **Median filter**（中值滤波器）的设置。未选取中值滤波器 — 7； 选取了中值滤波器 — 8。默认值为 8。

时钟高相位过采样因子

此参数定义了 I^2C SCL 时钟相位中的高过采样因子，即一个高周期 I^2C SCL 时钟周期内组件时钟的数量。它仅使用于 **Master**（主设备）模式。过采样因子的最小值取决于 **Median filter**（中值滤波器）的设置。未选取中值滤波器 — 5； 选取了中值滤波器 — 6。默认值为 8。

Clock from terminal (终端时钟)

此参数允许在内部配置时钟和外部配置时钟之间进行选择，以生成数据速率。选中该选项时，组件不对数据速率进行控制，但会基于用户连接的时钟源和组件过采样因子来显示实际数据速率。未选中该选项时，PSoC Creator 将配置所需要的时钟源。组件根据 **Data rate** 参数和过采样因子来计算出时钟源的频率。

注意：当设置数据速率或外部时钟频率值时，请确保 PSoC Creator 可以用当前系统时钟频率提供该值。否则，编译项目时会生成时钟准确度范围警告。警告中包含 PSoC Creator 所设置的实际时钟值。选择需要更改系统时钟还是更改组件时钟以满足时钟设置系统的要求，达到最佳值。

Median filter（中值滤波器）

该参数在 I^2C SDA 输入路径上使用数字 3 抽头中值滤波器。该滤波器可增强信号的抗扰度。然而，过采样因子的最小值将增加。

Slave address （从设备地址（7 位））

这是从设备的 I^2C 识别地址。它仅使用于从设备模式。该地址为右对齐的 7 位从设备地址，它不包括读/写位。可以选择介于 0 至 127 之间的从设备地址，默认值为 **8**。

可以输入十进制或十六进制的值；对于十六进制数值，请在地址前面键入“0x”。同时也提供了二进制的输入格式。

Slave address mask（从设备地址掩码）

在地址匹配程序中，此参数用于屏蔽从设备地址的位。地址掩码中的位 0 对应于读/写方向位，在地址匹配中并不重要。

- 位值为 0 — 该位不进行地址对比。
- 位值为 1 — 该位须与 I^2C 从设备地址的相应位相互匹配。

例如：从设备地址为 0x36，并且从设备地址掩码为 0xDE （地址匹配中，位 0 为读/写位，位 5 为无需关注的位）。匹配的从设备地址为：0x36 和 0x26。

可以输入十进制或十六进制的值；对于十六进制数值，请在地址前面键入“0x”。同时也提供了二进制的输入格式。

Accept matching address in RX FIFO（接受 RX FIFO 中的匹配地址）

此参数确定是否接受 RX FIFO 中的 I^2C 从设备匹配地址。如果需要支持多个 I^2C 地址，那么请使能该项。启用软件地址匹配代码，以确定 RX FIFO 地址是否与所支持的地址范围相匹配。

Accept matching address in RX FIFO（使能从睡眠模式唤醒）

此选项会在从设备地址匹配时将系统从睡眠模式唤醒。只有在从设备或多个主设备-从设备模式下，该选项才有效。

使能该选项时，会添加下述限制（仅限于 PSoC 4100/PSoC 4200 器件）：

- 从设备地址必须保证为偶数（位 0 的值为零）
- 必须禁用中值滤波器

请参考本文档的 I2C 章节中低功耗模式部分的内容，以及*系统参考指南*中的*电源管理 API* 一节，了解更加详细的信息。

I2C API

通过应用编程接口（API），可以使用软件对组件进行配置。下表列出并说明了每个函数的接口。以下各节将更详细地介绍了每个函数。

默认情况下，PSoC Creator 将实例名称“SCB_1”分配给第一个添加到工作区（TopDesign）的 SCB 组件。您可以将该实例重新命名为符合标识符语法规则的唯一一个任意值。实例名称会成为每个全局函数名称、变量和符号常量的前缀。为了便于读取，下表中使用的实例名称为“SCB”。

函数	说明
SCB_Init()	根据自定义程序中定义的参数初始化SCB组件。
SCB_Enable()	使能SCB组件操作。
SCB_Start()	启动SCB组件。
SCB_Stop()	禁用SCB组件。
SCB_Sleep()	准备SCB组件进入深度睡眠模式。
SCB_Wakeup()	使SCB组件退出深度睡眠模式。
SCB_I2CInit()	配置SCB组件，以在I2C模式下运行。
SCB_I2CSlaveStatus()	返回从设备状态标志。
SCB_I2CSlaveClearReadStatus()	返回读取状态标志并清除从设备读取状态标志。
SCB_I2CSlaveClearWriteStatus()	返回写状态并清除从设备写状态标志。
SCB_I2CSlaveSetAddress()	设置从设备地址，该值介于0至127（0x00 至 0x7F）之间。
SCB_I2CSlaveSetAddressMask()	设置从设备地址掩码，该值介于0至254（0x00至0xFE）之间。
SCB_I2CSlaveInitReadBuf()	设置从设备接收数据缓冲区（主设备 <- 从设备）。
SCB_I2CSlaveInitWriteBuf()	设置从设备写缓冲区（主设备 -> 从设备）。
SCB_I2CSlaveGetReadBufSize()	由于调用了SCB_I2CSlaveClearReadBuf()，因此返回主设备读取的字节数量。
SCB_I2CSlaveGetWriteBufSize()	由于调用了SCB_I2CSlaveClearWriteBuf()，因此返回主设备所写的字节数量。
SCB_I2CSlaveClearReadBuf()	将读取缓冲区计数器复位为零。
SCB_I2CSlaveClearWriteBuf()	将写入缓冲区计数器复位为零。
SCB_I2CMasterStatus()	返回主设备状态。
SCB_I2CMasterClearStatus()	返回主设备状态并清除状态标志。
SCB_I2CMasterWriteBuf()	将引用数据缓冲区写入到指定的从设备地址内。
SCB_I2CMasterReadBuf()	从指定的从设备地址读取数据，并将数据放入引用的缓冲区中。

函数	说明
SCB_I2CMasterSendStart()	生成启动条件，并发送指定的从设备地址。
SCB_I2CMasterSendRestart()	生成重启条件并发送指定的从设备地址。
SCB_I2CMasterSendStop()	生成停止条件。
SCB_I2CMasterWriteByte()	写入单个字节。它是一个手动指令，并且只能与 SCB_I2CMasterSendStart() 或 SCB_I2CMasterSendRestart() 函数一同使用。
SCB_I2CMasterReadByte()	读取单个字节。它是一个手动指令，并且只能与 SCB_I2CMasterSendStart() 或 SCB_I2CMasterSendRestart()函数一同使用。
SCB_I2CMasterGetReadBufSize()	返回通过SCB_I2CMasterReadBuf()函数传输的字节数。
SCB_I2CMasterGetWriteBufSize()	返回通过SCB_I2CMasterWriteBuf()函数传输的字节数。
SCB_I2CMasterClearReadBuf()	将读取缓冲区指针复位到缓冲区的起点。
SCB_I2CMasterClearWriteBuf()	将写缓冲区指针复位到缓冲区的起点。

全局变量

正常运行时，不需获得这些变量。

变量	说明
SCB_initVar	SCB_initVar表示SCB组件是否完成了初始化。该变量被初始化为0，并在第一次调用SCB_Start()时设置为1。这样，第一次调用SCB_Start()子程序后，不用重新初始化组件即可重启。 如需重新初始化组件，可在调用SCB_Start()或 SCB_Enable()函数前，先调用SCB_Init()函数。

I2C 函数应用

函数	从设备	主设备	多主设备	多主从设备
SCB_I2CInit()	+	+	+	+
SCB_I2CSlaveStatus()	+	–	–	+
SCB_I2CSlaveClearReadStatus()	+	–	–	+
SCB_I2CSlaveClearWriteStatus()	+	–	–	+
SCB_I2CSlaveSetAddress()	+	–	–	+
SCB_I2CSlaveSetAddressMask()	+	–	–	+
SCB_I2CSlaveInitReadBuf()	+	–	–	+
SCB_I2CSlaveInitWriteBuf()	+	–	–	+
SCB_I2CSlaveGetReadBufSize()	+	–	–	+
SCB_I2CSlaveGetWriteBufSize()	+	–	–	+

函数	从设备	主设备	多主设备	多主从设备
SCB_I2CSlaveClearReadBuf()	+	–	–	+
SCB_I2CSlaveClearWriteBuf()	+	–	–	+
SCB_I2CMasterStatus()	–	+	+	+
SCB_I2CMasterClearStatus()	–	+	+	+
SCB_I2CMasterWriteBuf()	–	+	+	+
SCB_I2CMasterReadBuf()	–	+	+	+
SCB_I2CMasterSendStart()	–	+	+	+
SCB_I2CMasterSendRestart()	–	+	+	+
SCB_I2CMasterSendStop()	–	+	+	+
SCB_I2CMasterWriteByte()	–	+	+	+
SCB_I2CMasterReadByte()	–	+	+	+
SCB_I2CMasterGetReadBufSize()	–	+	+	+
SCB_I2CMasterGetWriteBufSize()	–	+	+	+
SCB_I2CMasterClearReadBuf()	–	+	+	+
SCB_I2CMasterClearWriteBuf()	–	+	+	+

void SCB_Init(void)

说明： 使SCB组件初始化，以便在下面所选的配置之一中运行： I^2C、SPI、UART或EZ I^2C。
当配置被设为“未配置的SCB” 时，该函数不进行任何初始化操作。将使用模式特定的初始化API： SCB_I2CInit、SCB_SpiInit、SCB_UartInit或SCB_EzI 2CInit。

参数： 无

返回值： 无

其他影响： 无

void SCB_Enable(void)

说明： 使能SCB组件。

组件使能时，不应更改SCB配置。禁用组件后，方可更改配置。

当将配置设为“未配置SCB”时，首先必须初始化组件，使其操作于下列某一种配置状态：I^2C、SPI、UART 或 EZ I^2C。否则，该函数不会使能组件。

参数： 无

返回值： 无

其他影响： 无

void SCB_Start(void)

说明： 调用SCB_Init()与SCB_Enable()。调用该函数后，组件启动且准备就绪运行。

当将配置设为“未配置SCB”时，首先必须初始化组件，使其操作于下列某一种配置状态：I^2C、SPI、UART 或 EZ I^2C。否则，该函数将不会使能组件。

参数： 无

返回值： 无

其他影响： 无

void SCB_Stop(void)

说明： 禁用 SCB 组件及其中断。

参数： 无

返回值： 无

其他影响： 无

void SCB_Sleep(void)

说明： 准备组件进入深度睡眠。

“从睡眠模式使能唤醒”的选择操作对该函数的使用产生影响。

在调用CyPmSysDeepSleep()函数之前，先调用SCB_Sleep()函数。有关功耗管理函数的详细信息，请参考PSoC Creator*系统参考指南*。

进入睡眠模式前不得调用该函数。

参数： 无

返回值： 无

其他影响： 无

void SCB_Wakeup(void)

说明： 退出深度睡眠模式后，为组件进入活动模式做准备。
“从睡眠模式使能唤醒”的选择对该函数的实现产生影响。
退出睡眠模式后不得调用该函数。

参数： 无

返回值： 无

其他影响： 在调用SCB_Sleep()函数前，调用SCB_Wakeup()函数可能会导致意外行为。

void SCB_I2CInit(SCB_I2C_INIT_STRUCT *config)

说明： 配置SCB以运行I^2C 。

该函数**专为**当自定义程序中的SCB配置设为“未配置SCB”时使用。在I2C模式下初始化SCB后，可以使用SCB_Start() 或者 SCB_Enable() 函数启动组件。

该函数为提供配置设置的结构使用了一个指针。该结构所含的信息自定义程序设置同样可提供。

参数： config：针对含有如下字段排序列表的指针。这些字段与自定义程序的选项相匹配。设置的详细说明，请见自定义程序

字段	说明
uint32 mode	I2C的操作模式。可以选择下面各定义： • SCB_I2C_MODE_SLAVE • SCB_I2C_MODE_MASTER • SCB_I2C_MODE_MULTI_MASTER • SCB_I2C_MODE_MULTI_MASTER_SLAVE
uint32 oversampleLow	I2C时钟低相位的过采样因子。从设备模式操作时，该因子将被忽略。过采样因子必须连同时钟率一起选择，以便生成所需的I2C运行速率。
uint32 oversampleHigh	I2C时钟高相位的过采样因子。从设备模式操作时，该因子将被忽略。
uint32 enableMedianFilter	0 – 禁用 1 – 使能
uint32 slaveAddr	7位从设备地址。在非从设备模式下，该地址将被忽略。
uint32 slaveAddrMask	8位从设备地址掩码。位0的值必须为零。在非从设备模式下将被忽略。
uint32 acceptAddr	0 – 禁用 1 – 使能 使能后，Rx FIFO接收匹配地址。
uint32 enableWake	0 – 禁用 1 – 使能 在非从设备模式下，它将被忽略。

返回值： 无

其他影响： 无

uint32 SCB_I2CSlaveStatus(void)

说明： 返回从设备通信的状态。

参数： 无

返回值： uint32：EZ I^2C从设备的当前状态。

此状态包含读取和写入的状态常量。每个常量是一个位字段值。返回值可能包含多个已设置的位，用于指示读取或写入传输的状态。

从设备状态常量	说明
SCB_I2C_SSTAT_RD_CMPLT	从设备读取传输已完成。主设备通过发送一个 “NAK”[2] 表示完成读取时，会置位该位。 必须检查读取错误状态位，以确保读取传输已经完成。
SCB_I2C_SSTAT_RD_BUSY	正在进行从设备读取传输操作。主设备向从设备发送一个读取指令时，将置位该位；则设置RD_CMPLT时会清除它。
SCB_I2C_SSTAT_RD_OVFL	主设备尝试读取超过缓冲区限制的字节。
SCB_I2C_SSTAT_RD_ERR	从设备在进行读取传输时检测到总线错误。错误来源为：启动或停止条件的错误或从设备驱动SDA时仲裁丢失。
SCB_I2C_SSTAT_WR_CMPLT	从设备写入传输已完成。收到一个停止或重新启动条件时，会置位该位。 必须检查写入错误状态位，以确保写入传输已经完成。
SCB_I2C_SSTAT_WR_BUSY	正在进行从设备写入传输操作。当主设备向从设备发送一个写入指令时，将置位该位；则在WR_CMPT置位时可以将该位清零。
SCB_I2C_SSTAT_WR_OVFL	主设备尝试写入的内容已超出了缓冲区末尾。 此时，从设备会连续返回0xFF字节。
SCB_I2C_SSTAT_WR_ERR	从设备写入传输时检测到总线错误。错误来源为：启动或停止条件的错误或从设备驱动SDA时仲裁丢失。 当置位SCB_I2C_SSTAT_WR_ERR位时，对缓冲区进行写操作可能包含无效的字节。在这种情况下，推荐清除写入缓冲区的内容。

其他影响 无

[2] NAK 是“negative acknowledgment”（否定确认）或“not acknowledged”（未确认）的缩写。I^2C 文档中通常用“NACK”来表示，网络上的其它位置则用“NAK”来表示。两者的意思相同。

uint32 SCB_I2CSlaveClearReadStatus(void)

说明： 清除读取状态标志并返回其值。其他状态标志不受影响。

参数： 无

返回值： uint32：从设备的当前读取状态。有关常量的信息，请参见 SCB_I2CSlaveStatus() 函数。

其他影响 该函数并不会清除SCB_I2C_SSTAT_RD_BUSY。

uint32 SCB_I2CSlaveClearWriteStatus(void)

说明： 清除写入状态标志并返回它们的值。其它状态标志不受影响。

参数： 无

返回值： uint32：从设备的当前写入状态。有关常量的信息，请参见SCB_I2CSlaveStatus()函数。

其他影响 该函数不会清除SCB_I2C_SSTAT_WR_BUSY。

void SCB_I2CSlaveSetAddress(uint32 address)

说明： 设置I^2C从设备地址

参数： uint32 address：I^2C从设备地址。该地址为右对齐的7位从设备地址，它不包括读/写位。
该地址的值未就其是否违反I2C规范加以检查。推荐地址介于8到120 （0x08到0x78）之间。

返回值： 无

其他影响： 无

void SCB_I2CSlaveSetAddressMask(uint32 addressMask)

说明： 设置I^2C从设备地址

参数： uint32 addressMask：I^2C从地址掩码。
位值为0 — 该位不进行地址对比。
该比特值为 1 — 该位需要与I2C从设备地址的相应位相互匹配。
该值可为0到254之间的任意值（0x00到0xFE）。地址的LSB为R/W（读/写）位，无论“addressMask”位为何值，该位均被忽略。

返回值： 无

其他影响： 无

void SCB_I2CSlaveInitReadBuf(uint8 * rdBuf, uint32 bufSize)

说明： 设置缓冲区指针以及读取缓冲区的大小。该函数还会复位SCB_I2CSlaveGetReadBufSize()函数所返回的传输计数。

参数： uint8* rdBuf：指向主设备读取的数据缓冲区的指针。

uint32 bufSize：I^2C主设备可读取的缓冲区的大小。

返回值： 无

其他影响： 如果在总线数据操作期间调用该函数，则可能会传输前一个缓冲区位置的数据以及当前缓冲区起始位置的数据。

void SCB_I2CSlaveInitWriteBuf(uint8 * wrBuf, uint32 bufSize)

说明： 设置缓冲区指针并设置写入缓冲区的大小。通过该函数还可以重置由SCB_I2CSlaveGetWriteBufSize()函数返回的传输计数。

参数： uint8* wrBuf：指向主设备写入的数据缓冲区的指针。

uint32 bufSize：I^2C主设备可写入的缓冲区的大小。

返回值： 无

其他影响： 在总线数据操作期间调用该函数时，可接收前一个缓冲区和当前缓冲位置的数据。

uint32 SCB_I2CSlaveGetReadBufSize(void)

说明： 调用SCB_I2CSlaveInitReadBuf()或SCB_I2CSlaveClearReadBuf()函数之后，将返回由I^2C主设备读取的字节数。

最大返回值是读取缓冲区的大小。

参数： 无

返回值： uint32：主设备读取的字节。传输完成前返回的值为零。

其他影响 如果从设备捕获到SCB_I2C_SSTAT_RD_ERR，则该函数会返回无效的值。

uint32 SCB_I2CSlaveGetWriteBufSize(void)

说明： SCB_I2CSlaveInitReadBuf()或SCB_I2CSlaveClearReadBuf()函数被调用之后，返回由I^2C主设备写入的字节数。

最大返回值为写入缓冲区的大小。

参数： 无

返回值： uint32：主设备写入的字节。如果传输尚未完成，它将返回截至目前已传输的字节数。

其他影响 如果从设备捕获到SCB_I2C_SSTAT_WR_ERR，该函数会返回无效的值。

void SCB_I2CSlaveClearReadBuf(void)

说明： 将读取指针复位到读取缓冲区中的第一个字节。主设备读取的下一个字节将成为读取缓冲区的第一个字节。

参数： 无

返回值： 无

其他影响： 无

void SCB_I2CSlaveClearWriteBuf(void)

说明： 将写入指针重置为写入缓冲区中的第一个字节。主设备写入的下一个字节将成为写入缓冲区的第一个字节

参数： 无

返回值： 无

其他影响： 无

uint32 SCB_I2CMasterStatus(void)

说明： 返回主设备的通信状态。

参数： 无

返回值： uint32：I^2C主设备的当前状态。这个状态包含了状态常量。每个常量都是一个位字段值。返回值可能包含多个已设置的位，用于指示读取或写入传输的状态。

主设备状态常量	说明
SCB_I2C_MSTAT_RD_CMPLT	读取传输已完成。 必须检查错误条件状态位，以确保读取传输已经完成。
SCB_I2C_MSTAT_WR_CMPLT	写入传输已完成。 必须检查错误状态位，以确保写入传输已经完成。
SCB_I2C_MSTAT_XFER_INP	正在进行传输。
SCB_I2C_MSTAT_XFER_HALT	传输已停止。I^2C总线等待生成重启或停止条件。
SCB_I2C_MSTAT_ERR_SHORT_XFER	**错误条件**：在传输完所有字节之前，写入传输已经结束。 从设备否认了预期被确认的字节。
SCB_I2C_MSTAT_ERR_ADDR_NAK	**错误条件**：从设备没有确认地址。
SCB_I2C_MSTAT_ERR_ARB_LOST	**错误条件**：主设备在与从设备进行通信时仲裁失败。
SCB_I2C_MSTAT_ERR_BUS_ERROR	**错误条件**：由于总线上启动或停止条件发生错位，主设备传输在总线上发生了错误。
SCB_I2C_MSTAT_ERR_ABORT_XFER	**错误条件**：主设备执行启动条件生成时，从设备由另一台主设备寻址。因此，主设备自动切换到从设备模式并对其响应。主设备数据操作没有发生 该错误条件仅限于多主从设备模式。
SCB_I2C_MSTAT_ERR_XFER	**错误条件**：这是上述所有错误条件的逻辑“或”（OR）的值。

其他影响： 无

uint32 SCB_I2CMasterClearStatus(void)

说明： 清除所有状态标志并返回主设备状态。

参数： 无

返回值： uint32：主设备的当前状态。请参考SCB_I2CMasterStatus()函数的条件。

其他影响： 无

uint32 SCB_I2CMasterWriteBuf(uint32 slaveAddress, uint8 * wrData, uint32 cnt, uint32 mode)

说明： 自动将整个缓冲数据写入到从设备组件中。一旦该函数使能了数据传输，所包含的中断服务子程序（ISR）将处理进一步数据传输。

使能I^2C中断，并清除SCB_ I2C_MSTAT_WR_CMPLT状态位。

参数： uint32 slaveAddress：右对齐的7位从设备地址（有效范围为8至120）。

uint8 wrData：指向发送数据的缓冲区的指针。

uint32 cnt：缓冲区发送的位数。

uint32 mode：传输模式定义：（1）传输开始时是否生成了启动或重起条件，（2）总线生成停止条件之前，传输是否已完成还是被停止。

在传输模式下，可对所有模式常量执行“或”（OR）运算。

传输模式常量	说明
SCB_I2C_MODE_COMPLETE_XFER	执行从起始信号到停止信号的完整传输过程。
SCB_I2C_MODE_REPEAT_START	发送“重启”信号而不是“启动”信号。
SCB_I2C_MODE_NO_STOP	执行传输而无需使用“停止”信号。 预期下一个传输执行重启操作。

返回值： **uint32**：错误状态。有关常量的信息，请参见SCB_I2CMasterSendStart()函数。

其他影响 无

uint32 SCB_I2CMasterReadBuf(uint32 slaveAddress, uint8 * rdData, uint32 cnt, uint32 mode)

说明： 自动读取从设备中的整个缓冲数据。一旦该函数使能了数据传输，所包含的中断服务子程序（ISR）将处理进一步数据传输。

使能I^2C中断并清除SCB_ I2C_MSTAT_RD_CMPLT状态位。

参数： uint32 slaveAddress：右对齐的7位从设备地址（有效范围为8至120）。

uint8 rdData：指向用于放置从设备数据的缓冲区的指针。

uint32 cnt：要读取的缓冲区字节数。

uint32 mode：传输模式用于定义：

(1)在传输开始时是生成启动条件还是重新启动条件，

(2)在总线上生成停止条件之前，该传输是已完成还是被停止。

在传输模式下，可对所有模式常量执行“或”（OR）运算。有关常量的信息，请参见SCB_I2CMasterWriteBuf()函数。

返回值： uint32：错误状态。有关常量的信息，请参见SCB_I2CMasterSendStart()函数。

其他影响 无

uint32 SCB_I2CMasterSendStart(uint32 slaveAddress, uint32 bitRnW)

说明： 生成启动条件，并发送带有读/写位的从器件地址。禁用I^2C中断。

该函数被阻止，并不会返回，直到发出启动条件和地址字节，并且接收到ACK/NACK响应信号或发生错误为止。

参数： uint32 slaveAddress：右对齐的7位从设备地址（有效范围为8至120）。

uint32 bitRnW： 下一个传输的方向它由地址字节中的读/写位定义。

方向常量	说明
SCB_I2C_WRITE_XFER_MODE	设置下一个传输操作需要的写入方向。
SCB_I2C_READ_XFER_MODE	设置下一个传输操作需要的读取方向。

返回值： uint32：错误状态。

错误状态常量	说明
SCB_I2C_MSTR_NO_ERROR	函数完成、无误。
SCB_I2C_MSTR_BUS_BUSY	总线繁忙。总线上不会发送任何数据。需要再次尝试发送数据。
SCB_I2C_MSTR_NOT_READY	主设备不是总线上的活动主设备。可能正在执行从设备操作。总线上不会发送任何数据。需要再次尝试发送数据。
SCB_I2C_MSTR_ERR_LB_NAK	**错误条件：**已否认最后一个字节。
SCB_I2C_MSTR_ERR_ARB_LOST	**错误条件：**主设备仲裁失败
SCB_I2C_MSTR_ERR_BUS_ERR	**错误条件：**主设备发生总线错误。总线错误被错认为开始检测或停止检测。
SCB_I2C_MSTR_ERR_ABORT_START	**错误条件：**由于从设备操作的开始，会中止启动条件的生成。 该错误条件仅限于多主从设备模式。

其他影响： 无

uint32 SCB_I2CMasterSendRestart(uint32 slaveAddress, uint32 bitRnW)

说明： 生成重启条件，并发送带有读/写位的从器件地址。

该函数被阻止，并不会返回，直到发出启动条件和地址，并且接收到ACK/NACK响应信号或发生错误为止。

参数： uint32 slaveAddress：右对齐的7位从设备地址（有效范围为8至120）。

uint32 bitRnW： 下一个传输的方向它由地址字节中的读/写位定义。有关常量的信息，请参见SCB_I2CMasterSendStart()函数。

返回值： uint32：错误状态。有关常量的信息，请参见SCB_I2CMasterSendStart()函数。

其他影响 在调用该函数之前，必须生成有效的“启动”或“重启”条件。如果在调用该函数之前“启动”或“重启”条件失败，则该函数将不执行任何操作。

uint32 SCB_I2CMasterSendStop(void)

说明： 在总线上生成停止条件。

如果已生成带读取方向的启动或重启条件，则至少须读取一个字节。

该函数正在执行阻止操作，且在生成停止条件或发生错误前不会退出。

参数： 无

返回值： uint32：错误状态。有关常量的信息，请参见SCB_MasterSendStart()命令。

其他影响 在调用该函数之前，必须生成有效的“启动”或“重启”条件。如果在调用该函数之前“启动”或“重启”条件失败，则该函数将不执行任何操作。

对于读传输，在生成停止条件前必须至少读取一个字节。

uint32 SCB_I2CMasterWriteByte(uint32 theByte)

说明： 向从设备发送一个字节。

该函数正在执行阻止操作，且在字节传输完成或发生错误之前不会返回。

参数： uint32 theByte：发送到从设备的数据字节。

返回值： uint32：错误状态。

错误状态常量	说明
SCB_I2C_MSTR_NO_ERROR	函数完成、无误。
SCB_I2C_MSTR_NOT_READY	主设备不是总线上的活动主设备。可能正在执行从设备操作。总线上不发送任何数据。需要再次尝试。
SCB_I2C_MSTR_ERR_LB_NAK	**错误条件**：已否认最后一个字节。
SCB_I2C_MSTR_ERR_ARB_LOST	**错误条件**：主设备仲裁失败
SCB_I2C_MSTR_ERR_BUS_ERR	**错误条件**：主设备发生总线错误。总线错误被错认为开始检测或停止检测。

其他影响： 在调用该函数之前，必须生成有效的“启动”或“重启”条件。如果在调用该函数之前“启动”或“重启”条件失败，则该函数将不执行任何操作。

uint32 SCB_I2CMasterReadByte(uint32 ackNack)

说明： 向从设备中读取一个字节，并确认或否认接收到的值。

该函数并不会生成NAK信号。随后调用SCB_I2CMasterSendStop()或SCB_I2CMasterSendRestart()会分别生成NAK和停止或重启条件。

该函数正在执行阻止操作，且在接收到字节或发生错误之前不会返回。

参数： uint32 ackNack：响应已收到的字节。

相应常量	说明
SCB_I2C_ACK_DATA	发出确认（ACK）响应。 主设备通知从设备继续传输，且下一个字节将被读取。
SCB_I2C_NAK_DATA	发出否认（NAK）响应。 主设备通知从设备传输操作将要完成。

返回值： uint32：从从设备中读取的字节。如果发生了错误，返回数据的最高有效位将置为‘1’。

其他影响 在调用该函数之前，必须生成有效的“启动”或“重启”条件。如果在调用该函数之前启动或重启条件失败，则该函数将不执行任何操作并返回一个无效字节。

uint32 SCB_I2CMasterGetReadBufSize(void)

说明： 返回通过SCB_I2CMasterReadBuf()函数传输的字节数。

参数： 无

返回值： uint32：传输的字节数。如果传输尚未完成，它将返回截至目前已传输的字节数。

其他影响 如果在读取传输时发生了SCB_I2C_MSTAT_ERR_ARB_LOST 或 SCB_I2C_MSTAT_ERR_BUS_ERROR，则该函数会返回无效值。

uint32 SCB_I2CMasterGetWriteBufSize(void)

说明： 返回通过SCB_I2CMasterWriteBuf()函数传输的字节数。

参数： 无

返回值： uint32：传输的字节数。如果传输尚未完成，则在传输结束前返回零。

其他影响 如果在写入传输时发生了SCB_I2C_MSTAT_ERR_ARB_LOST 或 SCB_I2C_MSTAT_ERR_BUS_ERROR，则该函数会返回无效值。

void SCB_I2CMasterClearReadBuf(void)

说明： 将读取缓冲区指针重置为缓冲区中的首字节。

参数： 无

返回值： 无

其他影响： 无

void SCB_I2CMasterClearWriteBuf(void)

说明： 将写入缓冲区指针重置为缓冲区中的第一个字节。

参数： 无

返回值： 无

其他影响： 无

Bootloader 支持

SCB 组件可作为 Bootloader 的通信组件使用。SCB 仅在 I2C 模式下方可用作 Bootloader。使用以下配置为外部系统与 Bootloader 之间的通信协议提供支持：

- 配置：I2C

- I2C 模式：从设备或多主从设备
- 数据速率：必须与主机（引导器件）数据速率一致。
- 从设备地址：必须与所选主机（引导器件）的从设备地址保持一致。

更多信息，请参考 Bootloader 组件数据手册。

SCB 组件为使用 Bootloader 提供了一组 API 函数。

函数	说明
SCB_CyBtldrCommStart()	启动I^2C组件并使能其中断。
SCB_CyBtldrCommStop()	禁用I^2C组件并禁用其中断。
SCB_CyBtldrCommReset()	将读取和写入I^2C缓冲区设置为其初始状态，然后重置从设备状态。
SCB_CyBtldrCommWrite()	允许调用程序将数据写入Bootloader主机。该函数将处理轮询以便将数据块完整发送至主机器件。
SCB_CyBtldrCommRead()	允许调用程序读取Bootloader主机中的数据。该函数将处理轮询以便从主机组件完整接收数据块。

void SCB_CyBtldrCommStart(void)

说明： 启动I^2C组件并使能其中断。
每个I^2C写数据操作都将视为Bootloader的指令。
每个输入I^2C读数据操作都将在Bootloader响应已执行的指令前返回0xFF。

参数： 无

返回值： 无

其他影响： 无

void SCB_CyBtldrCommStop(void)

说明： 禁用I^2C组件并禁用其中断。

参数： 无

返回值： 无

其他影响： 无

void SCB_CyBtldrCommReset(void)

说明： 将读取和写入I^2C缓冲区设置为其初始状态，然后重置从设备状态。

参数： 无

返回值： 无

其他影响： 无

cystatus SCB_CyBtldrCommRead(uint8 pData[], uint16 size, uint16 * count, uint8 timeOut)

说明： 允许调用程序读取Bootloader主机中的数据。该函数将处理轮询，以便从主机器件完整接收数据块。

参数： uint8 pData[]：指向发送到器件的数据块的指针。

uint16 size：要写入的字节数

uint16 *count：指向用于写实际写入字节数的变量指针。

uint8 timeOut：等待的单位数（时间为10毫秒），之后会因超时而返回。

返回值： cystatus：如果未遇到任何问题，则返回CYRET_SUCCESS，或返回对该问题描述最准确的值。有关更多信息，请参考*系统参考指*》的“返回代码”一节。

其他影响： 无

cystatus SCB_CyBtldrCommWrite(const uint8 pData[], uint16 size, uint16 * count, uint8 timeOut)

说明： 允许调用程序将数据写入Bootloader主机。该函数将处理轮询以便将数据块完整发送到主机器件。

参数： const pData[]：指向发送到器件的数据块的指针。

uint16 size：要写入的字节数。

uint16 *count：指向用于写实际写入字节数的变量指针。

uint8 timeOut：等待的单位数（时间为10毫秒），之后会因超时而返回。

返回值： cystatus：如果未遇到任何问题，则返回CYRET_SUCCESS，或返回对该问题描述最准确的值。有关更多信息，请参考*系统参考指南*中“返回代码”一节。

其他影响： 无

I2C 功能描述

该组件支持 I²C 从设备、主设备、多主设备和多主从设备等配置。下文简述了如何在这些配置下使用该组件。

由于 I²C 硬件被中断驱动，因此需要为该组件使能全局中断。即使该组件需要中断，也无需向 ISR（中断服务子程序）添加任何代码。该组件将独立于代码之外为所有中断（数据传输）提供服务程序。为此接口分配的存储器缓冲区类似应用程序与 I²C 主设备/从设备之间的简单双端口存储器。

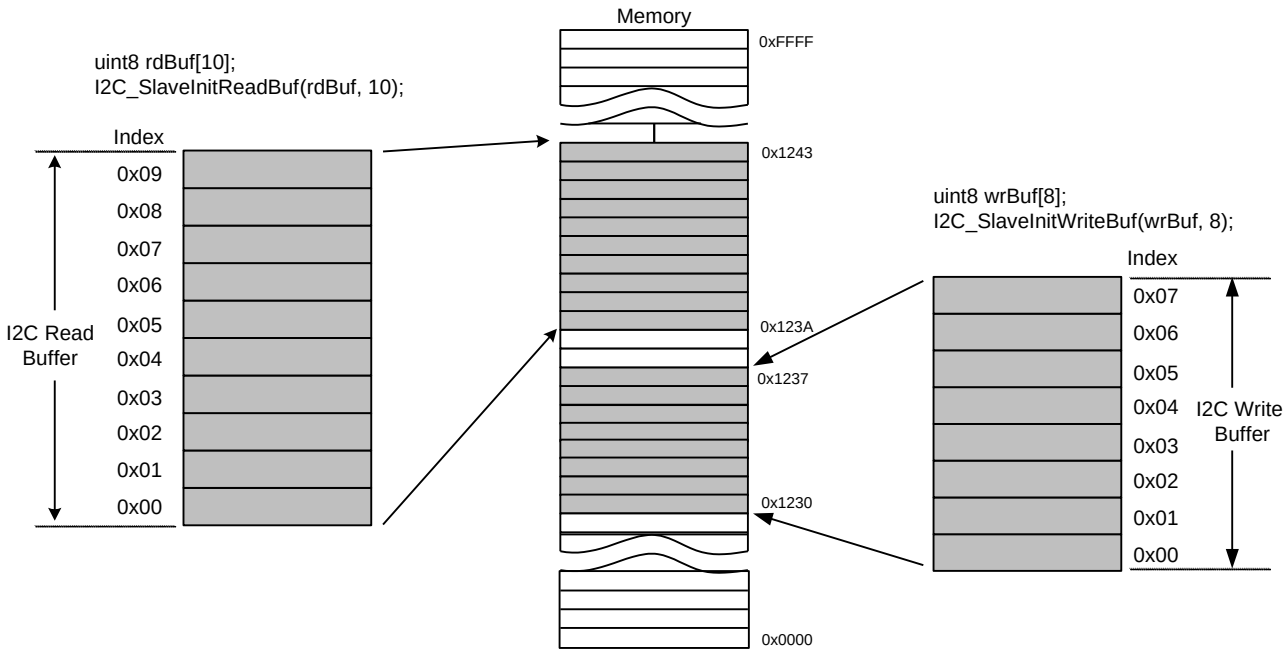
从设备操作

从设备接口由存储器中的两个缓冲区组成。其中一个用于由主设备写入到从设备的数据，另一个则用于主设备从从设备中读取的数据。请牢记，此处的读取和写入均是针对 I²C 主设备的。I²C 从设备的读写缓冲区由下列初始化指令完成设置。这些指令不会分配存储器，而是将阵列指针和大小复制到内部组件变量中。由于组件不会自动生成这些阵列，必须对用于缓冲区的阵列进行实例化。同一缓冲区可同时用于读取和写入缓冲区，但须妥当管理数据。

```
void SCB_I2CSlaveInitReadBuf(uint8 * rdBuf, uint32 bufSize)
void SCB_I2CSlaveInitWriteBuf(uint8 * wrBuf, uint32 bufSize)
```

使用上述功能为读取和写入缓冲区设置一个指针和字节计数。这些函数的“bufSize”可能不大于实际的阵列大小，但也不应大于“rdBuf”或“wrBuf”指针指向的可用存储器大小。

图 1. 从设备缓冲区结构



在调用 SCB_I2CSlaveInitReadBuf()或 SCB_I2CSlaveInitWriteBuf()函数时，内部索引将被设置为“rdBuf”和“wrBuf”分别指向的阵列中的第一个值。当 I²C 主设备读取或写入字节时，该索引会递增，直至偏移小于缓冲区大小。可以随时为读取缓冲区和写入缓冲区分别调用 SCB_I2CSlaveGetReadBufSize()或 SCB_I2CSlaveGetWriteBufSize()以查询已传输的字节数。读取或写入的字节数超过缓冲区中的字节数时，会发生溢出错误。溢出状态将在从设备的状态字节中设置，并可通过 SCB_I2CSlaveStatus() API 读取。

要将索引重置为阵列的起始位置，请使用以下指令。

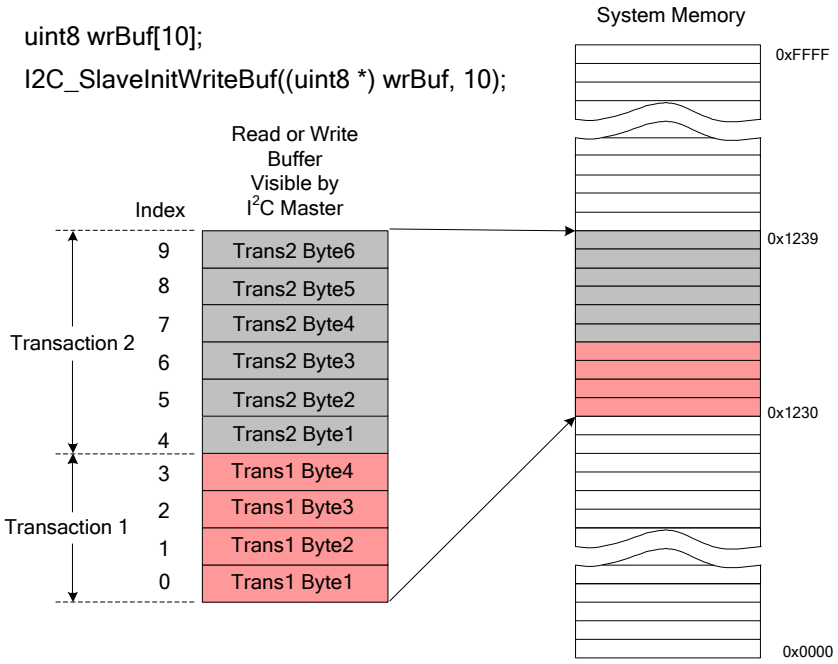
```
void SCB_I2CSlaveClearReadBuf(void)
void SCB_I2CSlaveClearWriteBuf(void)
```

此操作会将索引重置为零。I²C 主设备读取或写入的第二个字节为该阵列的第一个字节。在使用这些清除缓冲区的指令之前，应读取或更新阵列中的数据。

在使用清除缓冲区指令或是阵列索引尝试超过阵列的大小之前，I²C 主设备的多次读取或写入操作将继续使该索引递增。在图 2 的示例中，I²C 主设备已经执行了两次写入数据操作。第一次写入了 4 个字节，第二次写入了 6 个字节。其中，第二次操作中的第 6 个字节已被从设备否认，因此表明已到达缓冲区末尾。如果主设备试图在第二次操作中写入第 7 个字节，或在第三次操作时写入其他字节，则在重置缓冲区之前， 每个字节都会被否认并丢弃。

在第一次操作将索引重置为零并导致第二次操作覆盖第一次操作中的数据后，请使用 SCB_I2CSlaveClearWriteBuf()函数。请务必确保在缓冲区溢出时不会丢失数据。重置缓冲区索引之前，从设备应先处理缓冲区中的数据。

图 2. 系统存储器



读取和写入缓冲区均有四个状态位用以指示传输完成、正在传输以及缓冲区溢出等状态。传输开始时，将设置繁忙标志。传输完成时，将设置传输完成标志并清除繁忙标志。若开始第二次传输，则可能会同时设置繁忙标志和传输完成标志。读写状态标志如下表所示。

从设备状态常数	说明
SCB_I2C_SSTAT_RD_CMPLT	从设备读取传输已完成。
SCB_I2C_SSTAT_RD_BUSY	正在执行从设备读取传输（繁忙）。
SCB_I2C_SSTAT_RD_OVFL	主设备尝试读取超过缓冲区限制的字节。
SCB_I2C_SSTAT_RD_ERR	从设备在进行读取传输时检测到总线错误。
SCB_I2C_SSTAT_WR_CMPLT	从设备写入传输已完成。
SCB_I2C_SSTAT_WR_BUSY	正在执行从设备写入传输（繁忙）。
SCB_I2C_SSTAT_WR_OVFL	主设备尝试写入的内容已超出了缓冲区末尾。
SCB_I2C_SSTAT_WR_ERR	从设备写入传输时检测到总线错误。

以下示例代码对写入缓冲区初始化后等待传输完成。当传输完成后，会将数据复制到工作数组中以处理该数据。在许多应用程序中，不必复制数据到另一个地方，但可在原始缓冲区中进行处理。您可以通过使用读取函数和常量来替换写入函数和常量，创建一个与写入缓冲区示例一样的读取缓冲区示例。处理数据可能意味着新数据将传入到从设备缓冲区中，而不是从从设备缓冲区中传出。

```
uint8 wrBuf[10];
uint8 userArray[10];
uint32 byteCnt;
/* Initialize write buffer before call SCB_Start */
SCB_I2CSlaveInitWriteBuf((uint8 *) wrBuf, 10);

/* Start I2C Slave operation */
SCB_I2CStart();

/* Wait for I2C master to complete a write */

for(;;)  /* loop forever  */
{
   /* Wait for I2C master to complete a write */
   if(0u != (SCB_I2CSlaveStatus() & SCB_I2C_SSTAT_WR_CMPLT))
   {
      byteCnt = SCB_I2CSlaveGetWriteBufSize();
      SCB_I2CSlaveClearWriteStatus();
      for(i=0; i < byteCnt; i++)
      {
         userArray[i] = wrBuf[i]; /* Transfer data */
      }
      SCB_I2CSlaveClearWriteBuf();
   }
}
```

注意： 所有从设备状态和缓冲区操作 API 均不受中断保护，并且可通过 I2C ISR 修改。执行状态和缓冲区操作过程中，建议禁用 I2C 中断。禁用中断可阻止从设备操作并伸展 SCL 时钟。

主设备/多主设备操作

主设备和多主设备操作基本上是相同的，但有两个例外。当您在多主设备模式下操作时，应始终检查总线以查看其是否繁忙。另一个主设备可能已与另一个从设备进行通信。在这种情况下，程序必须等待当前操作完成后才能发出开始数据操作。该程序会查看返回值，如果另一个主设备控制了总线，则该值会设置繁忙状态。

第二个不同之处在于，在多主设备模式下可以同时启动两个主设备。如果发生这种情况，其中一个主设备将仲裁失败。必须在每个字节传输完成后检查是否出现这种情况。组件会自动检查这种情况，并在仲裁失败后发出错误响应。

在执行 I^2C 主设备操作时可进行如下选择：手动和自动。在自动模式中，将创建缓冲区以保持整个传输。在执行写入操作时，缓冲区预填充要发送的数据。如果从从设备读取数据，则至少需要分配一个具有数据包大小的缓冲区。要在自动模式中将字节阵列写入到从设备，请使用以下函数。

```
uint32 SCB_I2CMasterWriteBuf(uint32 slaveAddress, uint8 * wrData, uint32 cnt,
uint32 mode)
```

SlaveAddr 变量是一个右对齐的 7 位从设备地址，其值介于 0 至 127 之间。组件 API 将会自动将写入标志附加到地址字节的 LSb 中。传输的数据阵列将指向第二个参数“xferData”。参数“cnt”为要传输的字节数。最后一个参数，“mode”（模式），将确定传输开始和结束的方式。可以使用重新开始而不是开始来启动数据操作，或在停止序列之前终止数据操作。这些选项将允许执行“背对背”传输，其中最后一个传输不会发送“停止”指令，而下一个传输会发送“重新开始”而不是“开始”指令。

读取操作与写入操作方法几乎一致。会使用带有相同常量的相同参数。

```
uint32 SCB_I2CMasterReadBuf(uint32 slaveAddress, uint8 * rdData, uint32 cnt,
uint32 mode);
```

这两个函数都会返回状态。有关 SCB_I2CMasterStatus()函数返回值的信息，请参见状态表。由于在 I^2C 中断代码中读取和写入传输会在后台完成，因此可以使用 SCB_I2CMasterStatus()函数来确定该传输何时完成。后续代码段显示对从设备的典型写入操作。

```
SCB_I2CMasterClearStatus(); /* Clear any previous status */
SCB_I2CMasterWriteBuf(8u, (uint8 *) wrData, 10u, SCB_I2C_MODE_COMPLETE_XFER);
for(;;)
{
   if(0u != (SCB_I2CMasterStatus() & SCB_I2C_MSTAT_WR_CMPLT))
   {

      /* Transfer complete. Check Master status to make sure that transfer
         completed without errors. */

      break;
```

```
    }
}
```

I²C 主设备可手动操作。在此模式中，将使用单个指令来执行写入数据操作的每个部分。

```
status = SCB_I2CMasterSendStart(8u, SCB_I2C_WRITE_XFER_MODE);
if(SCB_I2C_MSTR_NO_ERROR == status)   /* Check if transfer completed without
errors */
{
   /* Send array of 5 bytes */
   for(i=0; i<5u; i++)
   {
      status = SCB_I2CMasterWriteByte(userArray[i]);
      if(SCB_I2C_MSTR_NO_ERROR != status)
      {
         break;
      }
   }
}
SCB_I2CMasterSendStop();      /* Send Stop */
```

除最后一个字节应该否认外，手动读取数据操作与写入数据操作类似。以下示例显示了一个典型的手动读取数据操作。

```
status = SCB_I2CMasterSendStart(8u, SCB_I2C_READ_XFER_MODE);
if(SCB_I2C_MSTR_NO_ERROR == status)   /* Check if transfer completed without
errors */
{
   /* Read array of 5 bytes */
   for(i=0; i<5u; i++)
   {
      if(i < 4u)
      {
         userArray[i] = SCB_I2CMasterReadByte(SCB_I2C_ACK_DATA);
      }
      else
      {
         userArray[i] = SCB_I2CMasterReadByte(SCB_I2C_NAK_DATA);
      }
   }
}
SCB_I2CMasterSendStop();    /* Send Stop */
```

多主从设备模式操作

在此模式中，可以执行多主设备和从设备操作。可以将组件称为从设备，但固件还会启动主设备模式传输。在此模式中，当主设备在地址字节期间仲裁失败时，从设备硬件将检查获胜主设备是否对其进行寻址。如果地址匹配，从设备转为活动状态。

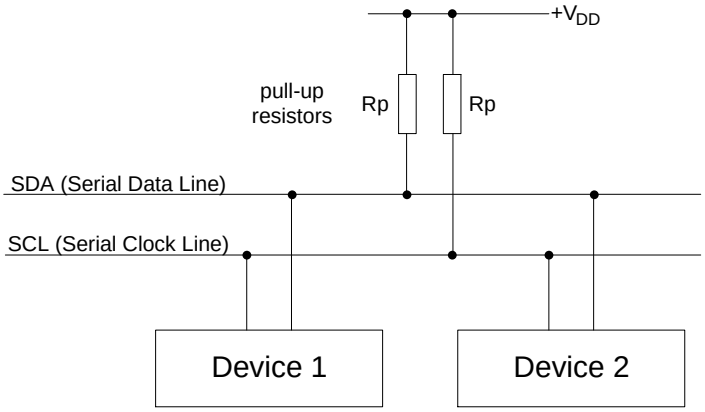
有关主设备和从设备操作实例的信息，请参见从设备操作和主设备操作 一节。

注意：所有从设备状态和缓冲区操作 API 均不受中断保护，并且可通过 I2C ISR 修改。执行状态和缓冲区操作过程中，建议禁用 I2C 中断。禁用中断可阻止从设备操作并伸展 SCL 时钟。

外部电路连接

如图 3 所示，I^2C 总线上要有外部上拉电阻。上拉电阻（R_P）由供电电压、时钟速度和总线电容确定。将输出阶段的任何器件（主设备或从设备）的最小灌电流设置为不超过 3 mA（在 V_{OLmax} = 0.4 V 的条件下）。这会将 5 V 系统的最小上拉电阻值限制在 1.5 kΩ 左右。R_P 的最大值取决于总线电容和时钟频率。对于总线电容为 150 pF 的 5 V 系统，上拉电阻不能超过 6 kΩ。有关上拉电阻和其他物理总线规格的更多信息，请参见 *I^2C 总线规格》*。

图 3. 器件与 I^2C 总线的连接情况



注意：从赛普拉斯或获得许可的其中一个联营公司处购买 I^2C 器件，即可根据 Philips I^2C 专利获得一份使用许可，以在符合 Philips 定义的 I^2C 标准规范的 I^2C 系统中使用这些器件。自 2006 年 10 月 1 日起，Philips 半导体就采用一个新的商标名称 — NXP Semiconductors。

低功耗模式

I^2C 模式下的组件可作为低功耗的睡眠模式和深度睡眠模式的唤醒源。

对于外设来说，睡眠模式便是它的活动模式。不需要对组件进行任何配置更改，也不需要在进入/退出此模式前调用代码。对从设备进行的任何通信都会引起中断并导致唤醒。发生中断的所有主设备活动会导致唤醒。

主设备模式不能作为深度睡眠的唤醒源。此功能仅适用于从设备模式。必须正确配置从设备，以使能该功能。在从设备配置对话框中，必须勾选“Enable wakeup from Sleep mode”（使能睡眠模式中唤醒）项。进入/退出深度睡眠模式前/后，必须调用 SCB_Sleep()和 SCB_Wakeup()函数。

唤醒事件是从设备的地址匹配。外部定时逻辑执行地址匹配。当发生此操作时，将触发唤醒中断。从设备伸展 SCL 线，直到控制权被传递给它的中断子程序以确认该地址为止。

进入深度睡眠模式前，必须完成从设备正在进行的数据操作，因此建议使用下列代码：

```
CyGlobalIntDisable; /* Disables all interrupts to lock the I2C bus state */

/* Checks if slave is busy */
status = (SCB_I2CSlaveStatus() & (SCB_I2C_SSTAT_RD_BUSY |
                                  SCB_I2C_SSTAT_WR_BUSY));

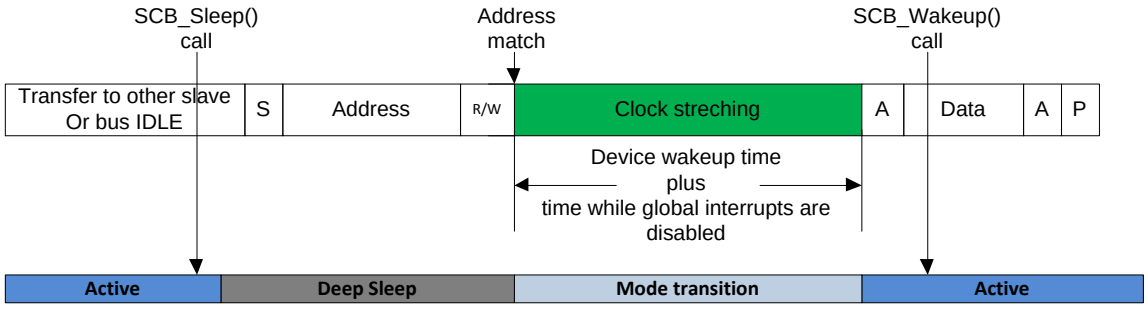
if(0u == status)
{
    SCB_Sleep(); /* Configure the slave to be wakeup source */

    CySysPmDeepSleep();

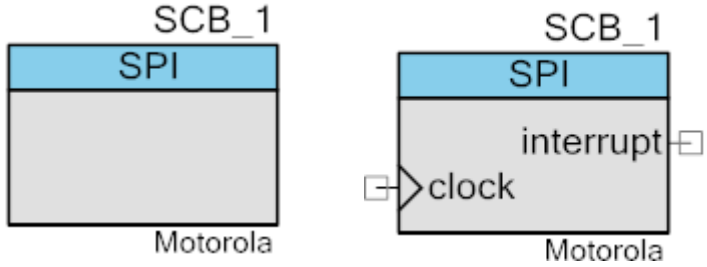
    CyGlobalIntEnable; /* Enable all interrupts to unlock I2C bus state */

    SCB_Wakeup(); /* Configure the slave to active mode operation */
}
else
{
    CyGlobalIntEnable; /* Transaction in progress: do not enter to Deep Sleep */
}
```

图 4. 主设备传输在从设备地址匹配时唤醒器件



SPI



该组件提供了行业标准的 4 线 SPI 接口。原始的 SPI 协议由 Motorola 定义。组件支持三种额外模式，允许同任何 SPI 器件进行通信。除了标准的 8 位字长之外，该组件还支持可配置的 4 至 16 位数据宽度，用于在非标准的 SPI 数据宽度下通信。

输入/输出接口

本部分描述了 SCB 组件的各种输入/输出接口。I/O 列表中的星号（*）表示，在 I/O 说明部分中所列出的情况下，该 I/O 可能不可见。

时钟 — 输入*

时钟负责运行该模块。只有将 **Clock from terminal**（终端时钟）复选框勾选上时，时钟接口才可见。

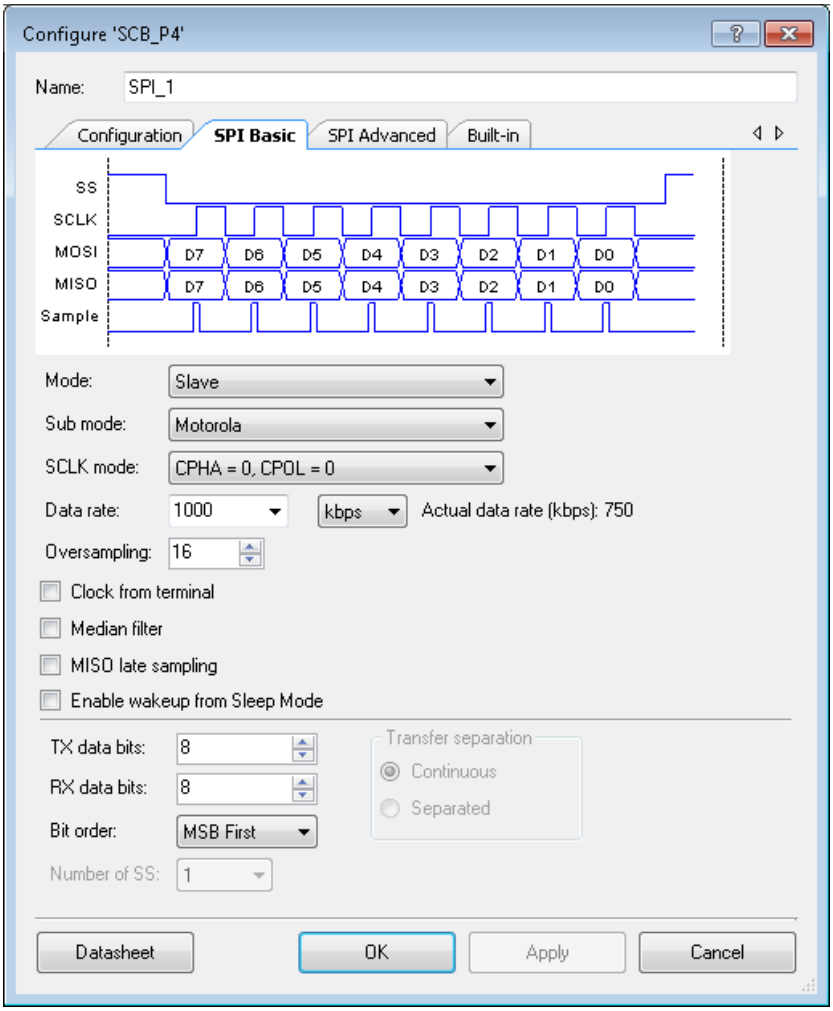
中断 — 输出*

此信号只能与中断组件相连，或者不连接。**Interrupt** 参数的设置，决定该终端的可见/不可见。

通信接口引脚（MOSI/MISO/SSx） 不可见，因为这些引脚只能连接到特定的 IO，不具备可路由性。了解更多信息，请参见芯片*技术参考手册（TRM）*中 *“I/O 系统”* 部分的内容。

注意：被隐藏输出引脚的输入缓冲器被禁止，以避免在低功耗模式下存在输入漏电流。读取这些引脚的状态返回值始终为零。要获取当前状态，读取状态前必须先使能输入缓冲区。

SPI 基本参数



SPI Basic（SPI 基本）选项卡包含下列参数：

Mode（模式）

该选项决定了 SCB 在何种 SPI 模式中操作。

- **从设备** — 仅限从设备操作（默认）
- **主设备** — 仅限主设备操作

Sub mode（子模式）

该选项决定了支持的 SPI 子模式：Motorola、TI（Start Coincides）、TI（Start Precedes）或 National Semiconductor 的 Microwire 协议。

- **Motorola** — 原始 SPI 协议由 Motorola 定义（默认）。

- **TI（Start Coincides）** — Texas Instruments 的 SPI 协议。
- **TI（Start Precedes）** — Texas Instruments 的 SPI 协议。
- **National Semiconductor** — National Semiconductor 的 Microwire 协议。

SCLK mode（SCLK 模式）

该参数定义了您想在通信中运用的时钟相位和时钟极性模式。已提供 CPHA 和 CPO 选择。

- **CPHA = 0，CPOL= 0** — 数据在 SCLK 的下降沿被输出。数据在 SCLK 的上升沿被采样。这是默认模式。
- **CPHA = 0，CPOL= 1**—数据在 SCLK 的上升沿被输出。数据在 SCLK 的下降沿被采样。
- **CPHA = 1，CPOL= 0** —数据在 SCLK 的上升沿被输出。数据在 SCLK 的下降沿被采样。
- **CPHA = 1，CPOL= 1** —数据在 SCLK 的下降沿被输出。数据在 SCLK 的上升沿被采样。

Clock from terminal (终端时钟)

此参数允许在内部配置时钟和外部配置时钟之间进行选择，以生成数据速率。使能该选项时，组件不对数据速率进行控制但会基于用户连接的时钟源显示实际数据速率。考虑到过采样的问题，PSoC Creator 会基于**数据速率**这个参数对要求的时钟频率进行计算和配置。

注意：当设置数据速率或外部时钟频率值时，请确保 PSoC Creator 可以用当前系统时钟频率提供该值。否则，编译项目时会生成时钟精度范围警告。警告中包含 PSoC Creator 所设置的实际时钟值。选择需要更改系统时钟还是更改组件时钟以满足时钟设置系统的要求，达到最佳值。

Data rate（数据速率）

通过此参数设置高达 8000kbps 的 SPI 数据速率值；实际速率可因可用时钟速度和分频器范围而异。标准比特率为 500、1000（默认）、2000、介于 4000 到 8000 且为 2000 kbps 的倍数的值。如果**终端时钟**这个参数被使能，该参数没有作用。

Oversampling（过采样）

此参数定义了 SPI 时钟的过采样因子；一个 SPI 时钟周期内组件时钟的数量（时钟周期比)。**过采样因子**用于计算获取定义的**数据速率**的过采样量时所需的内部组件时钟频率。对于**主设备**模式，将过采样值写入寄存器中，并用于生成 SPI 时钟。偶数**过采样因子**会使 SPI 时钟周期的第一相位与第二相位相同。否则，SPI 时钟周期的第一相位比它的第二相位长一个组件时钟周期。对于**从设备**，重要的组件就是时钟源。该时钟源必须根据要求提供过采样数量。过采样因子的最大值为 32；最小值则取决于组件的设置情况。

对于**主设备**，过采样因子的最小值取决于 **Median filter**（中值滤波器）和 **Enable late MISO sample**（使能延迟 MISO 采样）的设置情况：

- **中值滤波器**未被选中，**使能延迟 MISO 采样**也未被选中 - 6
- **中值滤波器**未被选中，**使能延迟 MISO 采样**被选中 - 3
- **中值滤波器**被选中，**使能延迟 MISO 采样**未被选中 - 8
- **中值滤波器**被选中，**使能延迟 MISO 采样**也被选中 - 4

使能延迟 MISO 采样不适用于**从设备**，并且过采样最小值仅取决于**中值滤波器**。未选中中值滤波器 — 6； 选中中值滤波器 — 8。

默认值为 16。

Median filter（中值滤波器）

该参数在 MISO 输入路径上应用 3 抽头中值滤波器。该滤波器会降低对错误的敏感性。然而，过采样因子的最小值将增加。默认值为**禁用**。

Enable late MISO sample （使能延迟 MISO 采样）

通过该选项可更改用于捕获 MISO 的 SCLK 边沿（仅适用于**主设备**模式）默认值为**禁用**。

使能从睡眠模式唤醒

可使用此选项将系统在从设备选择上从睡眠状态唤醒。

请参考本文档的 SPI 章节中低功耗模式部分的内容，以及*系统参考指南*中的*电源管理 API* 一节，了解更详细的信息。

TX data bits（TX 数据位）

此选项用于定义传输的数据帧中的位宽度。默认位数为单字节（8 位）。4 到 16 之间的任何整数都是有效设置。

RX data bits（RX 数据位）

此选项用于定义接收的数据帧中的位宽度。默认位数为单字节（8 位）。4 到 16 之间的任何整数都是有效设置。

注意：对于 **Motorola** 和 **Texas Instruments** 子模式，**TX 数据位**和 **RX 数据位**的数量设置应相同；对于 **National Semiconductor** 子模式，设置可以不同。

Bit order（位序）

位序参数用于定义串行数据的传输方向。设置为 **MSB 优先**时，首先传输最高有效位。设置为 **LSB 优先**时，首先传输最低有效位。

Number of SS（SS 的数量）

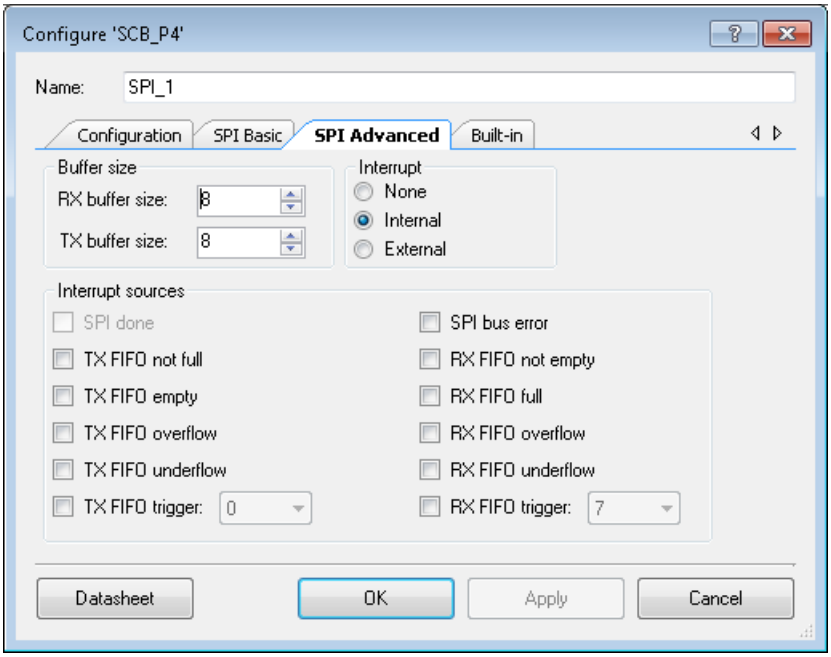
该参数决定 SPI 从设备选择线的数量。线路数量默认值为 1。1 到 4 之间的任何整数都是有效设置。此选项仅在**主设备**模式下有效。

Transfer separation（传输分离）

传输分离参数决定是否通过取消选择从设备选择来分开各个数据传输。下表中定义了这些模式。有关更多信息，请参见下表。

- **Continuous（连续）**—SS 在传输开始时降低，当传输完成时升高（默认）
- **Separated（分离）**—通过每 SCLK 周期发生的 SS 取消选择分离每个数据帧 4 位到 16 位。

高级 SPI 参数



SPI Advanced 选项卡包含下列各参数：

RX buffer size（RX 缓冲区大小）

RX 缓冲区大小参数定义为循环接收数据缓冲区分配的存储器大小（以字节/字为单位）。如果此参数设置为 8，则在硬件中将实现 8 字节/字的 RX FIFO。多达 2^{32} 的其他值会使用该 8 字节/字的 RX FIFO 和由 API 与内部 ISR 控制的软件缓冲区。缓冲区大小限制为可用的存储器。如果 RX 缓冲区大小超过 8 字节/字，中断模式自动设置为“内部”模式。

TX buffer size（TX 缓冲区大小）

TX 缓冲区大小参数定义为循环传输数据缓冲区分配的存储器大小（以字节/字为单位）。如果此参数设置为 8，则在硬件中将实现 8 字节/字的 TX FIFO。多达 2^{32} 的其他值会使用该 8 字节/字的 RX FIFO 和由 API 与内部 ISR 控制的软件缓冲区。缓冲区大小限制为可用的存储器。如果 TX 缓冲区大小超过 8 字节/字，中断模式自动设置“内部”模式。

Interrupt（中断）

此选项用于决定支持的中断模式：“无”、“内部”或“外部”。

- **无** — 禁止内部中断组件
- **内部** — 此选项将保留在 SCB 组件内部的中断组件。预先定义的内部 ISR 连接到中断。可以注册自定义函数，以每次进入 ISR 时均可以调用该函数。**中断源**选项定义了中断源，以触发中断。

- **外部 —** 此选项禁用内部中断并提供中断输出端口。如果需要自定义中断处理程序，可连接中断组件到中断输出端口。**中断源**选项可设置触发中断输出的中断源。

注意：对于大于 8 个字节/字的缓冲区，组件自动使能正确内部软件缓冲区操作所需的内部中断源。此外，必须使能全局中断，才能进行正确的缓冲区处理。

Interrupt sources（中断源）

SPI 支持以下事件中的中断：

- **SPI done** —主设备传输完成事件：TX FIFO 中的所有数据帧被发送，TX FIFO 为空
- **TX FIFO not full** —TX FIFO 未满
- **TX FIFO empty** — TX FIFO 为空
- **TX FIFO overflow** —尝试写入到已满的 TX FIFO
- **TX FIFO underflow** —尝试从空的 TX FIFO 中读取
- **TX FIFO trigger** — 当 TX FIFO 条目少于该字段数量时，生成发送器触发事件
- **SPI bus error** — 在 SPI 传输过程中，意外取消选择 SPI 从设备
- **RX FIFO not empty** — RX FIFO 不为空
- **RX FIFO full** — RX FIFO 已满
- **RX FIFO overflow** — 尝试写入到已满的 RX FIFO
- **RX FIFO underflow** — 尝试从空的 RX FIFO 中读取
- **RX FIFO trigger** — 当 RX FIFO 的条目大于该字段数量时，生成接收器触发事件。

注意：

当 **RX 缓冲区大小**超过 8 个字节/字时，组件保留 **RX FIFO not empty** 中断源，用于内部软件缓冲区操作。

当 **TX 缓冲区大小**超过 8 个字节/字时，组件保留 **TX FIFO not full** 中断源，用于内部软件缓冲区操作。

SPI API

API 允许您利用软件对组件进行配置。下表列出每个函数的接口，并进行了说明。以下各节将更详细地介绍了每个函数。

默认情况下，PSoC Creator 将实例名称“SCB_1”分配给第一个添加到工作区（TopDesign）的 SCB 组件。您可以将该实例重新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、变量和符号常量的前缀。为了便于读取，下表中使用的实例名称为“SCB”。

函数	说明
SCB_Init()	根据自定义程序中定义的参数初始化SCB组件。
SCB_Enable()	使能SCB组件。
SCB_Start()	开始SCB。
SCB_Stop()	禁用SCB组件。
SCB_Sleep()	为组件进入深度睡眠状态做出准备。
SCB_Wakeup()	使组件退出深度睡眠。
SCB_SpiInit()	为SPI操作配置SCB。
SCB_SpiSetActiveSlaveSelect()	选择活动从设备选择线。仅在主设备模式中适用。
SCB_SpiUartWriteTxData()	在传输缓冲区中放置将于下一个可用总线时间发送的数据输入。
SCB_SpiUartPutArray()	将要发送的数据阵列置于传输缓冲区。
SCB_SpiUartGetTxBufferSize()	返回当前传输缓冲区中元素的数量。
SCB_SpiUartClearTxBuffer()	清除传输缓冲区和TX FIFO。
SCB_SpiUartReadRxData()	从接收缓冲区检索下一个数据元素。
SCB_SpiUartGetRxBufferSize()	返回接收缓冲区中接收的数据元素的数量。
SCB_SpiUartClearRxBuffer()	清除接收缓冲区和RX FIFO。

全局变量

在正常运行情况下，无需了解这些变量。

变量	说明
SCB_initVar	SCB_initVar表示SCB组件是否完成了初始化。该变量被初始化为0，并在第一次调用 SCB_Start() 时设置为1。这样，第一次调用SCB_Start()子程序后，不用重新初始化组件即可重启。 如需重新初始化组件，可在调用SCB_Start()或 SCB_Enable()函数前，先调用SCB_Init()函数。
SCB_rxBufferOverflow	当产生内部软件接收缓冲区溢出时，设置 SCB_rxBufferOverflow。

void SCB_Init(void)

说明： 初始化SCB组件，以便在所选的配置之一中运行： I^2C、SPI、UART或EZ I^2C。

设置为“未配置的SCB”时，该函数不进行任何初始化操作。这时，将使用模式特定的初始化API： SCB_I2CInit、SCB_SpiInit、SCB_UartInit或SCB_EzI 2CInit。

参数： 无

返回值： 无

其他影响： 无

void SCB_Enable(void)

说明： 使能SCB组件。

组件使能时，不应更改SCB配置。禁用组件后，方可更改配置。

当将配置设为“未配置SCB”时，首先必须初始化组件，使其操作于下列某一种配置状态：I^2C、SPI、UART 或 EZ I^2C。否则，该函数将不会使能组件。

参数： 无

返回值： 无

其他影响： 无

void SCB_Start(void)

说明： 调用SCB_Init()与SCB_Enable()。调用该函数后，使能组件且准备就绪运行。

当将配置设为“未配置SCB”时，首先必须初始化组件，使其操作于下列某一种配置状态：I^2C、SPI、UART 或 EZ I^2C。否则，该函数将不会使能组件。

参数： 无

返回值： 无

其他影响： 无

void SCB_Stop(void)

说明： 禁用SCB组件及其中断。

参数： 无

返回值： 无

其他影响： 无

void SCB_Sleep(void)

说明： 为组件进入深度睡眠状态做出准备。

“从睡眠模式使能唤醒”的选择操作对该函数的使用产生影响。

在调用CyPmSysDeepSleep()函数之前，先调用SCB_Sleep()函数。有关功耗管理函数的详细信息，请参考*《系统参考指南》*中的“PSoC Creator”一节。

进入睡眠模式前不得调用该函数。

参数： 无

返回值： 无

其他影响： 无

void SCB_Wakeup(void)

说明： 退出深度睡眠模式后，为组件进入活动模式做准备。

“Enable wakeup from Sleep Mode”的选项的选择会影响此函数的实现。

退出睡眠模式后不得调用该函数。

参数： 无

返回值： 无

其他影响： 在调用SCB_Sleep()函数前，调用SCB_Wakeup()函数可能会导致意外行为。

void SCB_SpiInit(SCB_SPI_INIT_STRUCT *config)

说明： 为SPI操作配置SCB。

该函数**专为**当自定义程序中的SCB配置设为 “未配置SCB”时使用。在SPI模式下初始化SCB时，可使用SCB_Start()或SCB_Enable()函数使能组件。

该函数为提供配置设置的结构使用了一个指针。该结构所含的信息自定义程序设置同样可提供。

参数： config：指向含有如下字段排序列表的指针。这些字段与自定义程序的选项相匹配。设置的详细说明，请见自定义程序

字段	说明
uint32 mode	SPI操作模式。可以选择下面各定义： SCB_SPI_SLAVE SCB_SPI_MASTER
uint32 submode	SPI操作子模式。可以选择下面各定义： SCB_SPI_MODE_MOTOROLA SCB_SPI_MODE_TI_COINCIDES SCB_SPI_MODE_TI_PRECEDES SCB_SPI_MODE_NATIONAL
uint32 sclkMode	决定Motorola子模式的sclk关系。在其他子模式中，请忽略它。可以选择下面各定义： SCB_SPI_SCLK_CPHA0_CPOL0 SCB_SPI_SCLK_CPHA0_CPOL1 SCB_SPI_SCLK_CPHA1_CPOL0 SCB_SPI_SCLK_CPHA1_CPOL1
uint32 oversample	SPI时钟的过采样因子。从设备模式操作时，该因子将被忽略。

Parameters (cont.): config (cont.):

字段	说明
uint32 enableMedianFilter	0 – 禁用 1 – 使能
uint32 enableLateSampling	0 – 禁用 1 – 使能 从设备模式时，忽略它。
uint32 enableWake	0 – 禁用 1 – 使能 主设备模式时，将忽略它。
uint32 rxDataBits	RX方向数据位的数量。 对于National子模式，仅允许使用不同的dataBitsRx和dataBitsTx。
uint32 txDataBits	TX 方向数据位位数。 对于National子模式，仅允许使用不同的dataBitsRx和dataBitsTx。
uint32 bitOrder	确定位顺序。可以选择下面各定义： SCB_BITS_ORDER_LSB_FIRST SCB_BITS_ORDER_MSB_FIRST
uint32 transferSeperation	确定是否在SS之间进行连续传输或禁用SS。从设备模式时，忽略它。可以选择下面各定义： SCB_SPI_TRANSFER_CONTINUOUS SCB_SPI_TRANSFER_SEPARATED
uint32 rxBufferSize	字中的RX缓冲区大小： • 数值‘8’代表硬件中缓冲的用途。 • 如果数值超过‘8’将产生软件缓冲区。 必须使能SCB_INTR _RX_NOT_EMPTY中断，用于将数据传输到软件缓冲区内。
uint8* rxBuffer	为RX软件缓冲区提供的缓冲区空间： • 如果软件缓冲有指示，须提供NULL指针。 • 如果软件缓冲做出了说明，则必须提供分配缓冲区的指针。如果dataBitsRx不大于8，则在字节中缓冲区大小为（rxBufferSize + 1）；否则，在字节中缓冲区大小为（2 *（rxBufferSize + 1））。 软件RX缓冲区始终保持一个元素为空。为了能正确地操作，所分配的RX缓冲区必须比预期接收到的最大数据包尺寸大一个元素 。
uint32 txBufferSize	字中的TX缓冲区大小： • 数值‘8’代表硬件中缓冲的用途。 • 如果数值超过‘8’将产生软件缓冲区。

参数 （续）： config (cont.):

字段	说明
	•
uint8* txBuffer	为TX软件缓冲区提供的缓冲区空间： • 如果软件缓冲有指示，须提供NULL指针。 • 如果软件缓冲有指示，须提供分配缓冲区的指针。如果dataBitsTx不大于8，则缓冲区大小为txBufferSize；否则，其大小为（2* rxBufferSize）。
uint32 enableInterrupt	0 – 禁用 1 – 使能 如果使用软件缓冲区，要使能中断。
uint32 rxInterruptMask	要在RX方向中使能的中断源掩码。此掩码的写入不受enableInterrupt字段的设置情况的影响。通过提供以下要使能的所有源的逻辑或（OR）值来使能多个源： • SCB_INTR_RX_TRIGGER • SCB_INTR_RX_NOT_EMPTY • SCB_INTR_RX_FULL • SCB_INTR_RX_OVERFLOW • SCB_INTR_RX_UNDERFLOW • SCB_INTR_SLAVE_SPI_BUS_ERROR
uint32 rxTriggerLevel	RX触发中断的FIFO等级。不论是否使能RX触发中断源，均写入该值。
uint32 txInterruptMask	要在TX方向中使能的中断源掩码。此掩码的写入不受enableInterrupt字段的设置情况的影响。通过提供以下要使能的所有源的逻辑或（OR）值来使能多个源： • SCB_INTR_TX_TRIGGER • SCB_INTR_TX_NOT_FULL • SCB_INTR_TX_EMPTY • SCB_INTR_TX_OVERFLOW • SCB_INTR_TX_UNDERFLOW • SCB_INTR_MASTER_SPI_DONE
uint32 txTriggerLevel	TX触发中断的FIFO等级。不论是否使能TX触发中断源，均写入该值。

返回值： 无

其他影响： 无

void SCB_SpiSetActiveSlaveSelect(uint32 activeSelect)

说明： 选择活动从设备选择线。此函数仅适用于SPI主设备操作模式。组件应处于下列状态之一，以正确改变激活的从设备选择信号源。

- 禁用组件
- 组件已完成所有数据操作（TX FIFO为空，且已设置SpiDone标志）
- 对于此函数，不对状态进行检查。初始化后，活动从设备选择线为0。

参数： uint32 activeSelect：以下四种线路可使用“从设备选择”函数。

激活的从设备选择常量	说明
SCB_SPIM_ACTIVE_SS0	进行以下数据操作时，从设备选择线路0将被激活。
SCB_SPIM_ACTIVE_SS1	进行以下数据操作时，从设备选择线路1将被激活。
SCB_SPIM_ACTIVE_SS2	进行以下数据操作时，从设备选择线路2将被激活。
SCB_SPIM_ACTIVE_SS3	进行以下数据操作时，从设备选择线路3将被激活。

返回值： 无

其他影响： 无

void SCB_SpiUartWriteTxData(uint32 txData)

说明： 在传输缓冲区中放置将于下一个可用总线时间发送的数据输入。

该函数被禁用并等待，直到有可用的空间可放置要求的数据到传输缓冲区中为止。

参数： uint32 txData：所传输的数据。

需要传输的数据位数取决于选择的TX数据位（数据位将从txDataByte的LSB开始计数）。

返回值： 无

其他影响： 无

void SCB_SpiUartPutArray(const uint16/uint8 wrBuf[], uint32 count)

说明： 将需要发送的数据阵列放置于传输缓冲区内。

该函数被禁用，并等待直到传输缓冲区有可用空间就存放所有请求数据。

阵列大小可超过传输缓冲区的大小。

参数： const uint16/uint8 wrBuf[]：指向要放置在传输缓冲区中的数据阵列的指针。需要作为一个阵列条目传输的数据位数取决于选择的TX数据位（在每个阵列条目中，数据位将从LSB开始计数）。

uint32 count：要放置在传输缓冲区中的数据元素的数量。

返回值： 无

其他影响： 无

uint32 SCB_SpiUartGetTxBufferSize(void)

说明： 返回传输缓冲区中当前元素的数量。

禁用TX软件缓冲区：返回TX FIFO中用到的条目数量。

使能TX软件缓冲区：返回传输缓冲区中当前用到的元素数量。该数目不包含TX FIFO中用到的条目数量。TX FIFO未满前，传输缓冲区的大小为0。

参数： 无

返回值： uint32：准备传输的数据元素的数量。

其他影响： 无

void SCB_SpiUartClearTxBuffer(void)

说明： 清除传输缓冲区和TX FIFO。

参数： 无

返回值： 无

其他影响： 无

uint32 SCB_SpiUartReadRxData(void)

说明： 从接收缓冲区检索下一个数据元素。

禁用RX软件缓冲区：返回自RX FIFO检索的数据元素。如果RX FIFO为空，将返回未定义的数据。

使能RX软件缓冲区：从软件接收缓冲区返回数据元素。如果软件接收缓冲区为空，将返回数值0。

参数： 无

返回值： uint32：来自接收缓冲区的下一个数据元素。需要接收的数据位数取决于所选择的RX数据位（数据位将从返回值的LSB开始计数）。

其他影响 无

uint32 SCB_SpiUartGetRxBufferSize(void)

说明： 返回接收缓冲区中接收到的数据元素的数量。

使能RX软件缓冲区：返回RX FIFO中用到的条目数量。

使能RX软件缓冲区：返回放置在接收缓冲区中的元素的数量。

参数： 无

返回值： uint32：接收到的数据元素的数量

其他影响： 无

void SCB_SpiUartClearRxBuffer(void)

说明： 清除接收缓冲区和RX FIFO。

参数： 无

返回值： 无

其他影响： 无

SPI 功能描述

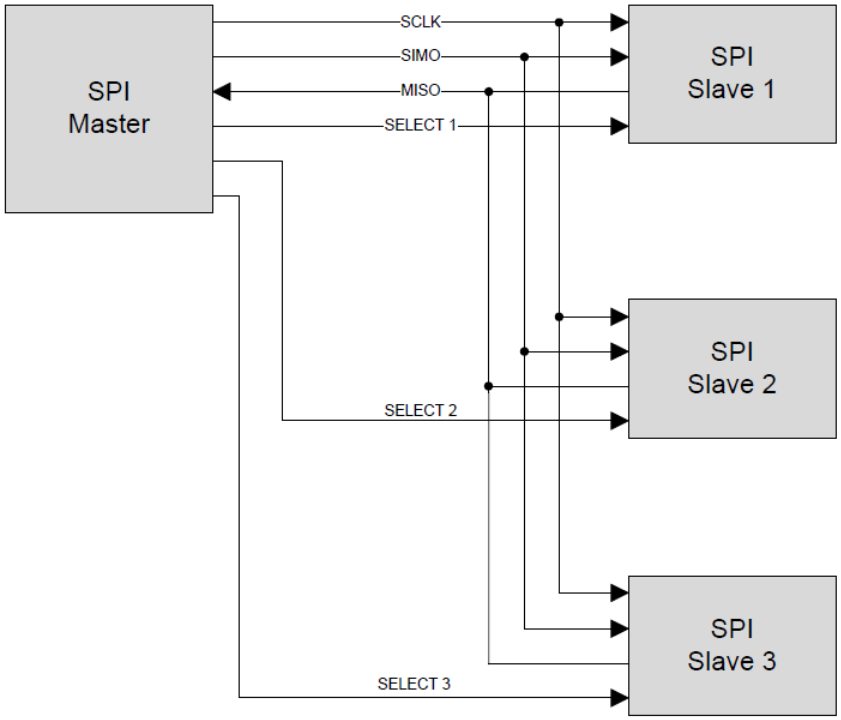
串行外设接口（SPI）协议是同步的串行接口，具有“单主设备多从设备”的拓扑结构。原始 SPI 协议由 Motorola 定义。器件在主设备或从设备模式下运行。主设备启动数据帧传输。多个从设备由各自的从设备选择线路支持。

SPI 接口包括四种信号：

- **SCLK**－串行时钟（从主设备输出，输入到从设备）。
- **MOSI**－主设备输出，从设备输入（从主设备输出，输入到从设备）。

- **MOSI**－主设备输入，从设备输出（输入到主设备，从从设备输出）。
- **SELECT**－从设备选择（主要为有效低电平信号，从主设备输出，输入到从设备）。

图 5. SPI 总线连接示例图



Motorola 子模式操作

原始 SPI 协议由 Motorola 定义。其为全双工协议：传输和接收同时进行。

Motorola SPI 协议具有四种模式，其定义了数据如何在 MOSI 和 MISO 线路上被输出和采样。这些模式由时钟极性（CPOL）和时钟相位（CPHA）决定。

- **CPHA = 0，CPOL= 0** － 数据在 SCLK 的下降沿被输出。数据在 SCLK 的上升沿被采样。
- **CPHA = 0，CPOL= 1**－数据在 SCLK 的上升沿被输出。数据在 SCLK 的下降沿被采样。
- **CPHA = 1，CPOL= 0** －数据在 SCLK 的上升沿被输出。数据在 SCLK 的下降沿被采样。
- **CPHA = 1，CPOL= 1** －数据在 SCLK 的下降沿被输出。数据在 SCLK 的上升沿被采样。

图 6 描述了基于 CPOL 和 CPHA 的 MOSI/MISO 数据输出和采样。

图 6. SPI Motorola 帧格式

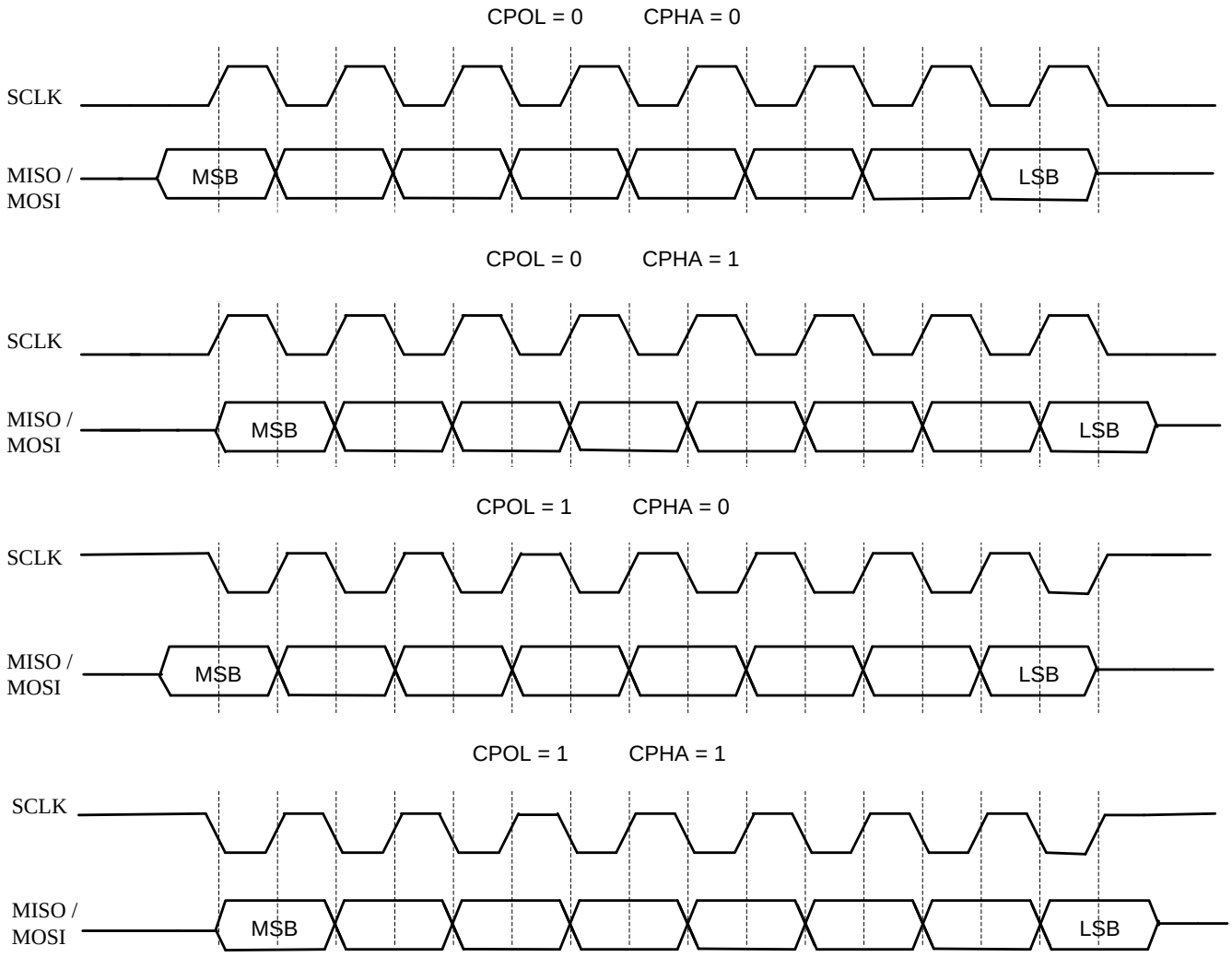
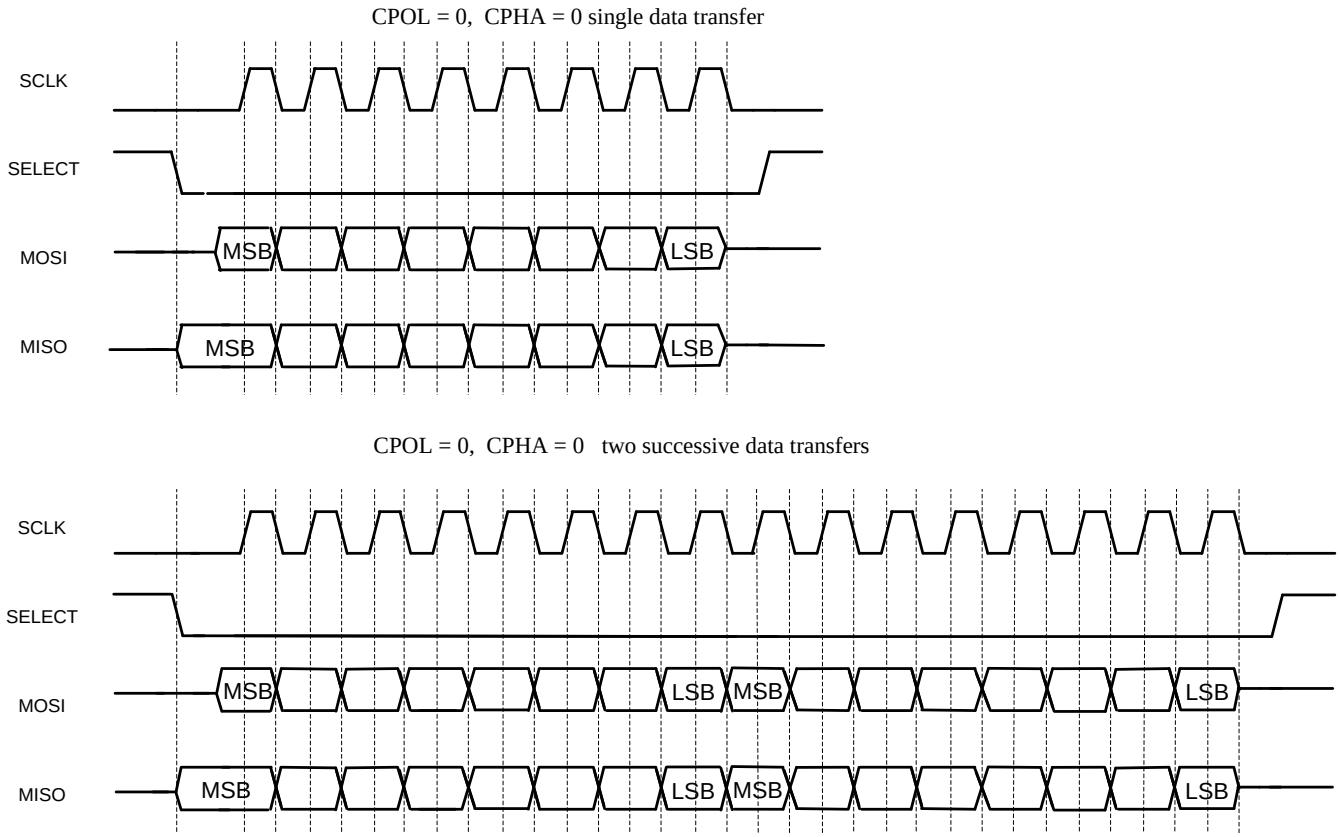


图 7 描述了模式 0（CPOL 为‘0’，CPHA 为‘0’）下的一个 8 位数据传输和两个连续的 8 位数据传输。

图 7. SPI Motorola 数据传输示例图



Texas Instruments 子模式操作

Texas Instruments 的 SPI 协议重新定义了 SS 信号的用途。其使用该信号（而非有效的从设备选择低电平信号）指示数据传输开始。该协议仅支持 CPHA = 1，CPOL= 0。

单个位传输周期中的高电平有效脉冲指示传输开始。在一个 SCLK 周期中，该脉冲传输可能优先于第一个数据帧位传输，或与第一个数据位传输同步发生。传输的 SCLK 时钟是自由运行的时钟。

图 8 描述了一个 8 位数据传输和两个连续的 8 位数据传输。SS 脉冲传输优先于第一个数据位传输。

请注意：第二个数据传输中的 SELECT 脉冲如何与第一个数据传输中的最后数据位同步传输。

图 8. TI（优先）数据传输示例图

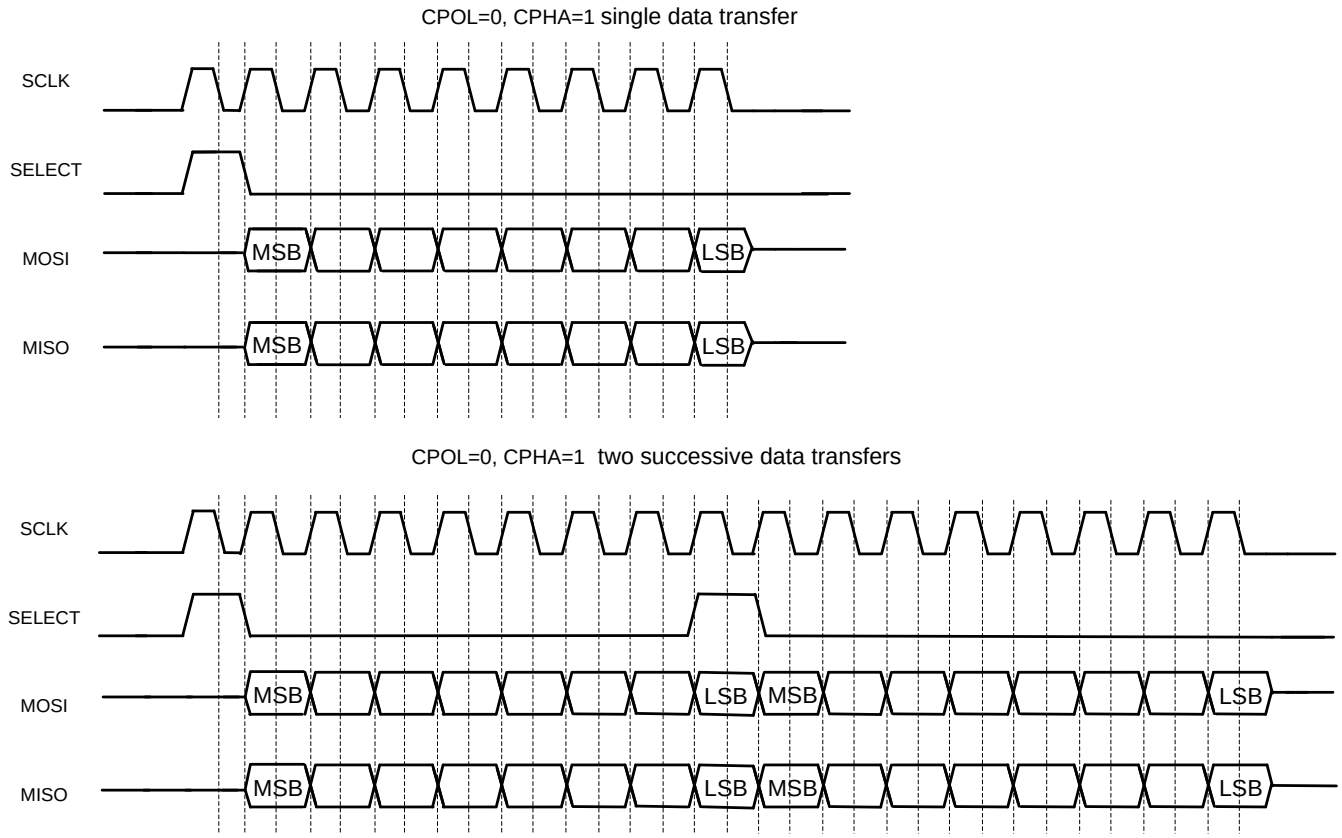
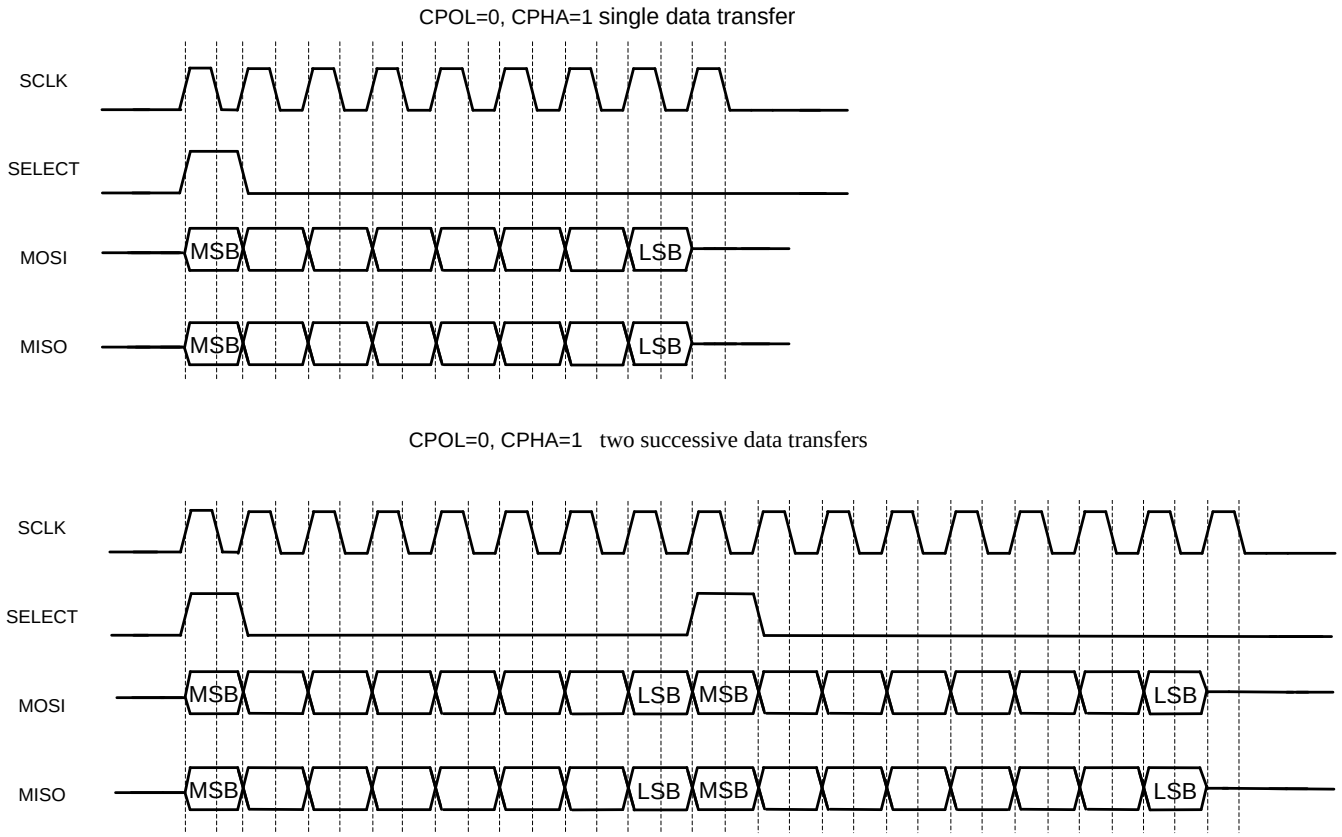


图 9 描述了一个 8 位数据传输和两个连续的 8 位数据传输。SS 脉冲传输优先于第一个数据位传输。

图 9. TI（同步）数据传输示例图



National Semiconductor 的 Microwire 子模式操作

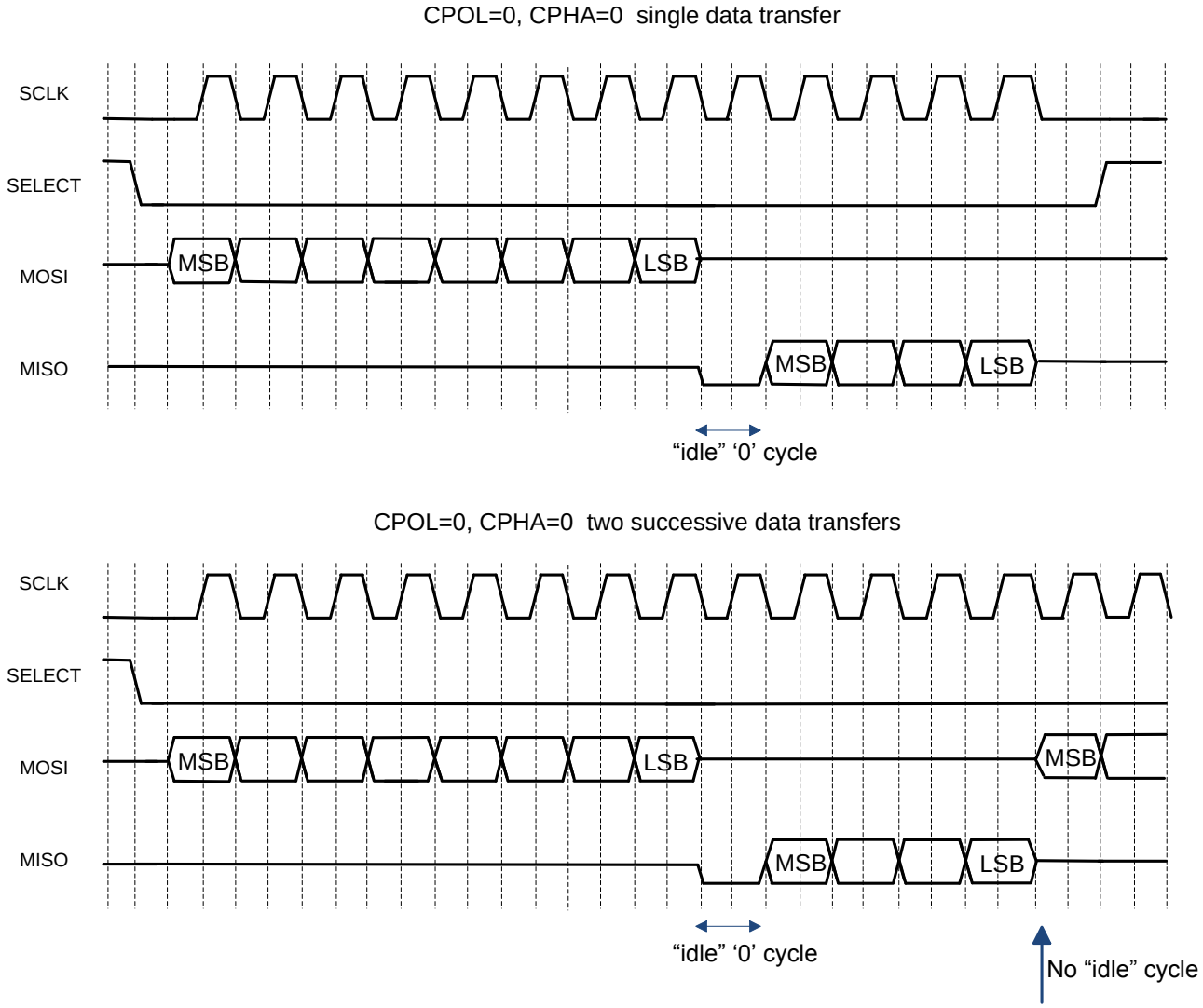
National Semiconductor 的 Microwire 协议为半双工协议。传输和接收交替进行，并非同时进行（传输发生在接收之前）。单个“空闲”位传输周期中，传输和接收分开进行。该协议仅支持 CPHA = 1，CPOL= 0。

注意：在单个“空闲”位传输周期中，连续的数据传输将不再分开进行。

传输数据的传输大小可能与接收数据的传输大小不同。

图 10 描述了一个数据传输和两个连续的数据传输。在这两种情况下，传输数据的传输大小为 8 位，接收数据的传输大小为 4 位.

图 10. National Semiconductor 的 Microwire 数据传输示例图



连续与分开传输

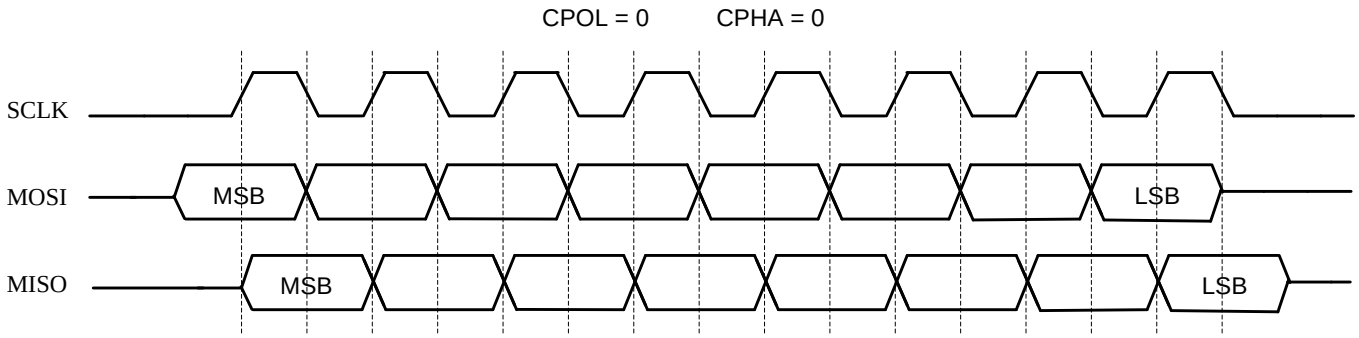
数据分开传输期间，SELECT 线路在分开传输之间始终从‘0’变为‘1’，然后又从‘1’变回‘0’。每个分开的数据传输重复该过程。

多个数据传输可能同时发生，SELECT 线路在分开传输之间无需进行切换。图 11 描述了 SCLK 模式（CPHA=0，CPOL= 0）下两个连续的 8 位数据传输。

MISO late sampling（MISO 延迟采样）

在 SCLK 后半周期，MISO 被采样（仅适用于主设备模式）。延迟采样技术解决了从主设备向从设备发送 SCLK 以及从从设备向主设备发送 MISO 时的往复路径延迟问题。

图 11. MISO 延迟采样示例图



软件缓冲器

SCB 具有 FIFO 存储器，是一个 16 字 x 16 位 SRAM，具备“字节写入使能”功能。在 SPI 模式下，FIFO 被拆分成 TX FIFO 和 RX FIFO。各含 8 个条目，每个条目 16 位。每个条目的 16 位宽度用于调节可配置的数据宽度。

内部中断处理程序用于提供软件和硬件 TX/RX 缓冲间的互动，而不对您的顶级固件进行任何更改。

您也应当考虑到使用软件缓冲会导致被传输的字之间计时间隔更大，因为中断处理程序需要额外的时间来执行（取决于所选 HFCLK 的值）。

中断

当 **RX 缓冲区大小**或 **TX 缓冲区大小**超过 8 字节/字时，**RX FIFO 非空**并且 **TX FIFO 未满**中断源通过内部软件缓冲操作组件保留。**请勿清除或禁用它们**，因为这样做会造成错误的软件缓冲操作。但清除来自其他源的中断事件是用户的责任，因为它们不会被自动清除。那是客户处理程序函数的功能。

假如 **RX 缓冲区大小**或 **TX 缓冲区大小**不大于 8 字节/字，使用的不是软件缓冲，而只是硬件 TX 或 RX FIFO。在**内部**中断模式中，不会自动清除中断。这是用户的责任。在这种情况下，优先选择**外部**或**无**中断。

低功耗模式

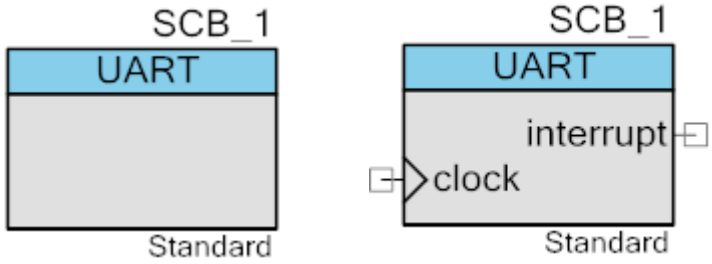
SPI 模式中的组件可作为低功耗睡眠和深度睡眠模式的唤醒源。

对于外设来说，睡眠模式便是它的活动模式。这便表明进入/退出此模式前，不需要对组件或将要调用的代码进行任何配置更改。对从设备进行的任何通信都会引起中断并导致唤醒。发生中断的所有主设备活动都会导致唤醒。

主设备模式不能作为深度睡眠的唤醒源。此功能仅适用于从设备模式。深度睡眠模式要求对从设备进行合适的配置，将其作为唤醒源使用。在从设备配置对话框中，必须勾选“Enable wakeup from Sleep mode”（使能睡眠模式中的唤醒）项。进入/退出深度睡眠模式前/后，必须调用 SCB_Sleep()和 SCB_Wakeup()函数。

在**从设备**运行模式中，器件的唤醒通过从设备的选择进行。唤醒耗时较长，进行中的 SPI 传输被非应答（NACK）— “0xff”字节发送到 MISO 线。器件唤醒时间过后，主设备必须再次巡查组件。

UART



UART 提供异步通信，常用串行异步通信设备为 RS232。支持三种不同的 UART 类串行接口协议：

- UART — 基本型
- SmartCard — 与 UART 类似，但是可以发送非应答（NACK）。
- IrDA — 用于红外线通信的调制方案。

输入/输出接口

本部分描述了 SCB 组件的各种输入/输出接口。I/O 列表中的星号（*）表示，在 I/O 说明部分中所列出的情况下，该 I/O 可能不可见。

时钟 — 输入*

时钟负责运行该模块。只有将 **Clock from terminal**（终端时钟）复选框勾选上时，时钟接口才可见。

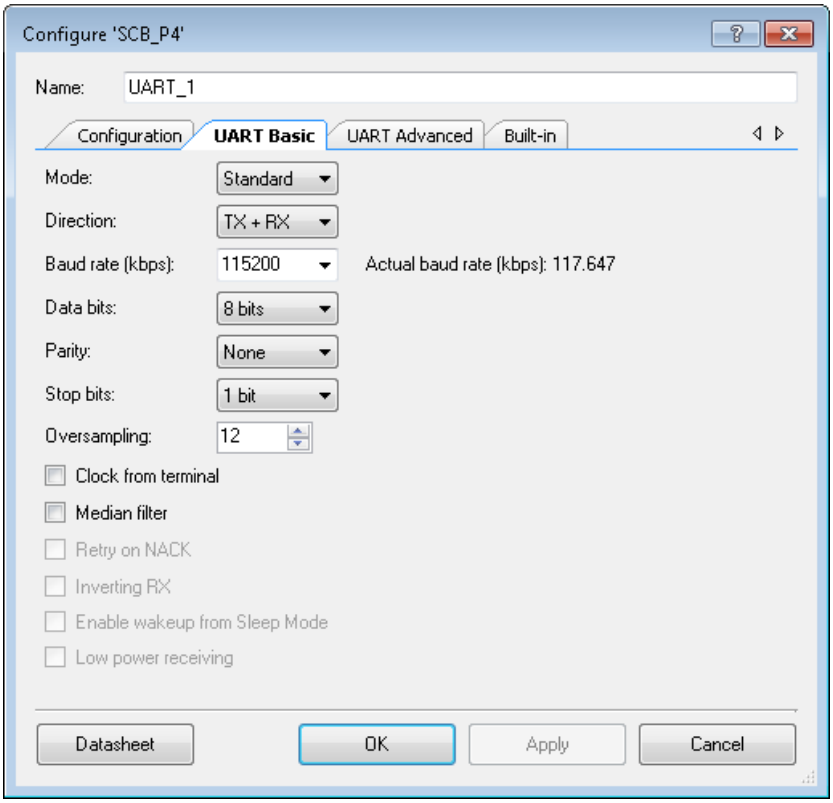
中断 — 输出*

此信号只能与中断组件相连，或者不连接。根据**中断**参数的设置，终端的存在位将各不相同。

通信接口引脚（TX/RX）不可见，因为这些引脚只能连接到特定的 IO，不具备可路由性。了解更多信息，请参见芯片*技术参考手册（TRM）*中*“I/O 系统”*部分的内容。

注意：禁用被隐藏的输出引脚上的输入缓冲区，以避免发生低功耗模式下的电流链锁现象。读取这些引脚的状态返回值始终为零。要获取当前状态，读取状态前必须先使能输入缓冲区。

基本 UART 参数



UART Basic 选项卡包含下列参数：

Mode（模式）

该选项用于确定 UART 的操作模式：Standard、SmartCard 或 IrDA。默认模式为 **Standard**（标准模式）。

Direction（传输方向）

该参数定义了要加入到 UART 组件中的功能模式。可设置为双向 **TX + RX**（默认），接收器（**仅适用于 RX**）或发送器（**仅适用于 TX**）。

Baud rate（波特率）

该参数定义时钟生成硬件的波特率或位宽配置，最高可达 921600。实际波特率可能因可用时钟速度和分频器范围而不同。默认值为 **115200**。

Data bits（数据位）

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**。

- 8 为默认配置，即每次传输发送一个字节。
- 9 位模式并不表示发送 9 个数据位；第 9 位会取代奇偶校验位，作为地址或数据的指示符。

Parity（奇偶校验）

该参数定义传输中奇偶校验位类型功能。可设置为**无**（默认）、**奇校验**或**偶校验**。

Stop bits（停止位）

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）、**1.5** 或 **2** 个数据位。

Oversampling（过采样）

该参数定义了 UART 接口的过采样因子。**过采样因子**用于计算获取定义的**数据速率**的过采样量时所需的内部组件时钟频率。过采样因子的最大值为 32；最小值则取决于组件的设置。

对于**标准和智能卡模式**，过采样因子最小值取决于**中值滤波器**的设置。未选取中值滤波器 — 6；选取中值滤波器 — 8。默认值为 16。

对于 **IrDA** 模式，会预定义过采样值，并始终使能中值滤波器。

默认值为 16。

Clock from terminal（终端时钟）

此参数允许在内部配置时钟和外部配置时钟之间进行选择，以生成数据速率。使能该选项时，组件不对数据速率进行控制但会基于用户连接的时钟源显示实际数据速率。不使能该选项时，考虑到过采样的问题，PSoC Creator 会基于**波特率**参数对所需的时钟频率进行计算和配置。

注意：当设置数据速率或外部时钟频率值时，请确保 PSoC Creator 可以用当前系统时钟频率提供该值。否则，编译项目时会生成时钟准确度范围警告。警告中包含 PSoC Creator 所设置的实际时钟值。选择是应当更改系统时钟还是组件时钟以满足时钟设置系统要求，达到最佳值。

Median filter（中值滤波器）

该参数在 RX 线的输入路径上应用 3 抽头数字中值滤波器。该滤波器会降低对错误的敏感性。然而，过采样因子的最小值将增加。默认值为**禁用**。

Retry on NACK（对 NACK 进行重试）

该参数用于对 NACK 功能进行重试。收到非应答时重新发送数据帧。该选项仅适用于 **SmartCard** 模式。

Inverting RX（反转 RX）

该参数用于对收到的 RX 线信号进行反转。该选项仅适用于 **IrDA** 模式。

Enable wakeup from Sleep Mode（使能从睡眠模式中唤醒）

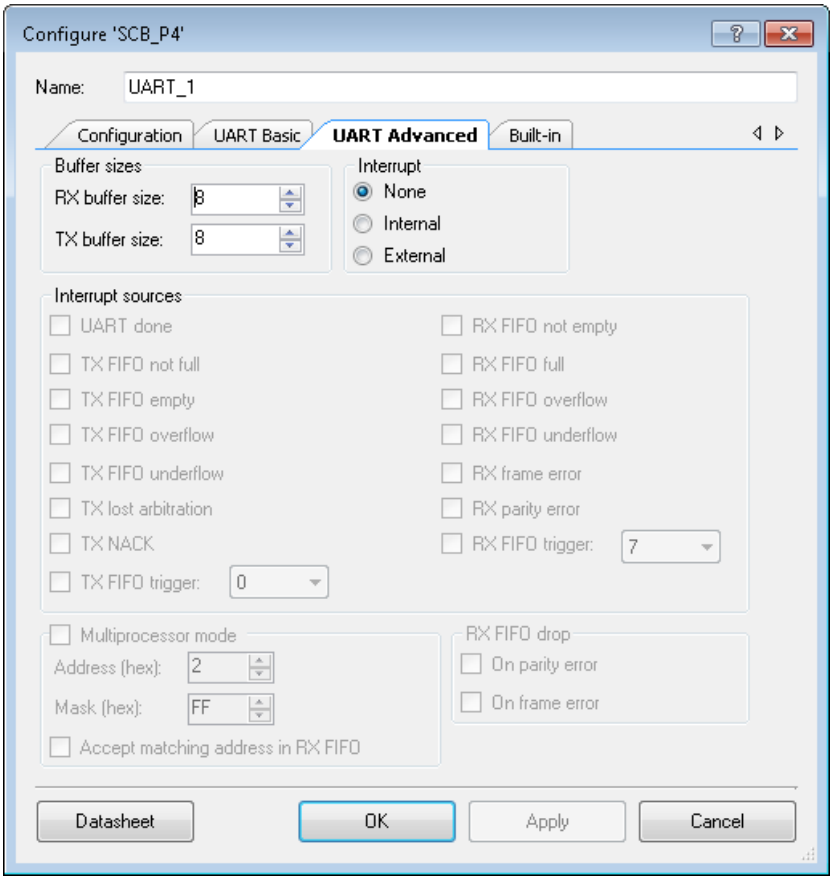
该选项可在开始位将系统从睡眠中唤醒。该选项仅在使能 **RX Direction** 时适用。

请参考本文档的 UART 章节中低功耗模式部分的内容，另请参考*系统参考指南*中的*电源管理 API* 一节，了解更详细的信息。

Low power receiving（低功耗接收）

该参数用于使能 IrDA 低功耗接收模式。该选项仅在使能 **RX Direction** 时适用。

高级 UART 参数



RX buffer size（RX 缓冲区大小）

RX 缓冲区大小参数定义为循环接收数据缓冲区分配的存储器大小（以字节/字为单位）。如果将该参数设置为 8，则在硬件中会实现 RX FIFO 的 8 个字节/字。所有其他值（最大为 2^{32}）使用 8

字节/字 RX FIFO 和由提供的 API 和内部 ISR 控制的软件缓冲区。缓冲区大小受限于可用存储器。如果 RX 缓冲区大小超过 8 字节/字，中断模式自动设置为“内部”模式。

TX buffer size（TX 缓冲区大小）

TX 缓冲区大小参数定义为循环传输数据缓冲区分配的存储器大小（以字节/字为单位）。如果此参数设置为 8，则在硬件中将实现 8 字节/字的 TX FIFO。所有其他值（最大为 2^{32}）使用 8 字节/字 RX FIFO 和由提供的 API 和内部 ISR 控制的软件缓冲区。缓冲区大小受限于可用存储器。如果 TX 缓冲区大小超过 8 字节/字，中断模式自动设置“内部”模式。

Interrupt（中断）

此选项用于确定支持的中断模式：“无”、“内部”或“外部”。

- **无 —** 禁止内部中断组件
- **内部 —** 此选项将保留在 SCB 组件内部的中断组件。预先定义的内部 ISR 连接到中断。可以注册自定义函数，以每次进入 ISR 时均可以调用该函数。**中断源**选项定义了中断源，以触发中断。
- **外部 —** 此选项禁用内部中断并提供中断输出端口。如果需要客户中断处理程序，中断组件可以与之连接。**中断源**选项可设置触发中断输出的中断源。

注意：对于大于 8 个字节/字的缓冲区，组件自动使能正确内部软件缓冲区操作所需的内部中断源。此外，必须使能全局中断，才能进行正确的缓冲区处理。

Interrupt sources（中断源）

SPI 支持以下事件中的中断：

- **UART done** — UART 发送器完成事件：TX FIFO 中的所有数据帧被发送且 TX FIFO 为空
- **TX FIFO not full** — TX FIFO 未满
- **TX FIFO empty** — TX FIFO 为空
- **TX FIFO overflow** —尝试写入到已满的 TX FIFO
- **TX lost arbitration** —UART 仲裁失败：TX 线上的输出值与 RX 线上的观测值不相同。当发送器和接收器共享 TX/RX 线时，该条件事件十分有用。在 SmartCard 模式下，更是如此。
- **TX NACK** —UART 发送器在 SmartCard 模式下收到非应答。
- **TX FIFO underflow** — 尝试从空的 TX FIFO 中读取。

- **TX FIFO trigger** — 当 TX FIFO 的条目少于该字段数量时，生成发送器触发事件。
- **RX FIFO not empty** — RX FIFO 不为空。
- **RX FIFO full** — RX FIFO 已满。
- **RX FIFO overflow** — 尝试写入到已满的 RX FIFO。
- **RX FIFO underflow** — 尝试从空的 RX FIFO 中读取。
- **RX frame error** — 接收数据帧中包含帧错误。该错误可能是启动或停止位错误：
- 启动位错误— 检测到启动位周期开始后（RX 线从‘1’变为‘0’），启动位周期中间采样错误（RX 线为‘1’）。

  注意：启动位错误在接收到数据帧前进行检测。
- 停止位错误：RX 线采样为‘0’，但是期望值为‘1’。

  注意：停止位错误可能会导致接收连续数据帧失败。停止位错误在接收到数据帧后进行检测。

  注意：如果停止位持续时间等于一位，不对帧错误进行跟踪。
- **RX parity error** —收到的数据帧中包含奇偶校验错误。
- **RX FIFO trigger** — 当 RX FIFO 的条目大于该字段数量时，生成接收器触发事件。

注意：

当 **RX 缓冲区的大小**超过 8 字节/字时，组件保留 **RX FIFO not empty** 中断源。

当 **TX 缓冲区的大小**超过 8 字节/字时，组件保留 **TX FIFO not full** 中断。

多处理器模式

该参数用于使能多处理器模式，其中 9 位的第一位表示地址。默认值为**禁用**。要使能该选项须将数据位设置为 9 位。

地址（hex）

从设备地址。用于在使能多处理器模式时进行匹配。默认值为 0x02。

掩码（hex）

从设备地址掩码。在与从设备地址进行匹配时使用这些位。默认值为 **0xFF**。

- 位值为 0 — 该位不进行地址对比。

- 位值为 1 — 该位需要与地址的相应位相互匹配。

接受 RX FIFO 中的匹配地址

该参数确定是否接受 RX FIFO 中的匹配地址。

注意：不匹配的地址不会存入 RX FIFO 中。

RX FIFO 丢弃

为 RX FIFO 提供硬件数据丢弃选项。

- **On parity error**（奇偶校验错误）— 定义奇偶校验失败时的行为。通过奇偶校验时，收到的数据被发送至 RX FIFO。否则，收到的数据丢失。仅适用于**标准**和 **SmartCard** 模式。
- **On frame error**（帧错误）— 定义检测到帧错误时的行为。未捕获帧错误时，收到的数据被发送至 RX FIFO。否则，收到的数据将丢失。

UART API

API 允许您利用软件对组件进行配置。下表列出每个函数的接口，并进行了说明。以下各节将更详细地介绍了每个函数。

默认情况下，PSoC Creator 将实例名称“SCB_1”分配给第一个添加到工作区（TopDesign）的 SCB 组件。您可以将该实例重新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、变量和符号常量的前缀。为了便于读取，下表中使用的实例名称为“SCB”。

函数	说明
SCB_Init()	根据自定义程序中定义的参数初始化SCB组件。
SCB_Enable()	使能SCB组件。
SCB_Start()	开始SCB。
SCB_Stop()	禁用SCB组件。
SCB_Sleep()	为组件进入深度睡眠状态做出准备。
SCB_Wakeup()	为组件退出深度睡眠做准备。
SCB_UartInit()	配置SCB以进行UART操作。
SCB_UartPutChar()	在下一个可用总线时间内放置传输缓冲区中要发送的一字节数据。
SCB_UartPutString()	在下一个可用的总线时间内放置传输缓冲区中要发送的以空字符结尾的字符串。
SCB_UartPutCRLF()	向传输缓冲区放置一个字节的数据后，输入回车符（0x0D）和换行符（0x0A）。
SCB_UartGetChar()	从接收缓冲区检索下一个数据元素。

函数	说明
SCB_UartGetByte()	从接收缓冲区检索下一个数据元素。
SCB_UartSetRxAddress()	在多处理器模式下，为UART设置硬件可检测的接收器地址。
SCB_UartSetRxAddressMask()	在多处理器模式下，为UART设置硬件地址掩码。
SCB_SpiUartWriteTxData()	在传输缓冲区中放置将于下一个可用总线时间发送的数据输入。
SCB_SpiUartPutArray()	将需要发送的数据阵列放置于传输缓冲区内。
SCB_SpiUartGetTxBufferSize()	返回传输缓冲区中当前元素的数量。
SCB_SpiUartClearTxBuffer()	清除传输缓冲区和TX FIFO。
SCB_SpiUartReadRxData()	从接收缓冲区检索下一个数据元素。
SCB_SpiUartGetRxBufferSize()	返回接收缓冲区中接收到的数据元素的数量。
SCB_SpiUartClearRxBuffer()	清除接收缓冲区和RX FIFO。

全局变量

在正常运行情况下，无需了解这些变量。

变量	说明
SCB_initVar	SCB_initVar表示SCB组件是否完成了初始化。该变量被初始化为0，并在第一次调用SCB_Start() 时设置为1。这样，第一次调用SCB_Start()子程序后，不用重新初始化组件即可重启。 如果组件需要重新初始化，则首先调用SCB_Init()函数，然后再调用SCB_Start()或SCB_Enable()函数。
SCB_rxBufferOverflow	当产生内部软件接收缓冲区溢出时，设置SCB_rxBufferOverflow。

void SCB_Init(void)

说明： 初始化SCB组件，以便在所选的配置之一中运行： I^2C、SPI、UART或EZ I^2C。
设置为“未配置的SCB”时，该函数不进行任何初始化操作。这时，将使用模式特定的初始化API： SCB_I2CInit、SCB_SpiInit、SCB_UartInit或SCB_EzI 2CInit。

参数： 无

返回值： 无

其他影响： 无

void SCB_Enable(void)

说明： 使能SCB组件。

组件使能时，不应更改SCB配置。禁用组件后，方可更改配置。

当将配置设为“未配置SCB”时，首先必须初始化组件，使其操作于下列某一种配置状态：I^2C、SPI、UART 或 EZ I^2C。否则，该函数将不会使能组件。

参数： 无

返回值： 无

其他影响： 无

void SCB_Start(void)

说明： 调用SCB_Init()与SCB_Enable()。调用该函数后，使能组件且准备就绪运行。

当将配置设为“未配置SCB”时，首先必须初始化组件，使其操作于下列某一种配置状态：I^2C、SPI、UART 或 EZ I^2C。否则，该函数将不会使能组件。

参数： 无

返回值： 无

其他影响： 无

void SCB_Stop(void)

说明： 禁用SCB组件及其中断。

参数： 无

返回值： 无

其他影响： 无

void SCB_Sleep(void)

说明： 为组件进入深度睡眠状态做出准备。

“从睡眠模式使能唤醒”的选择操作对该函数的使用产生影响。

在调用CyPmSysDeepSleep()函数之前，先调用SCB_Sleep()函数。

有关功耗管理函数的详细信息，请参考*《系统参考指南》*中的“PSoC Creator”一节。

进入睡眠模式前不得调用该函数。

参数： 无

返回值： 无

其他影响： 无

void SCB_Wakeup(void)

说明： 退出深度睡眠模式后，为组件进入活动模式做准备。
“Enable wakeup from Sleep Mode”的选项的选择会影响此函数的实现。
退出睡眠模式后不得调用该函数。

参数： 无

返回值： 无

其他影响： 在调用SCB_Sleep()函数前，调用SCB_Wakeup()函数可能会导致意外行为。

void SCB_UartInit(SCB_UART_INIT_STRUCT *config)

说明： 配置SCB以进行UART操作。

该函数**专为**当自定义程序中的SCB配置设为 “未配置SCB”时使用。在UART模式中初始化SCB后，可通过SCB_Start()或SCB_Enable()函数使能该组件。

该函数为提供配置设置的结构使用了一个指针。该结构所含由自定义程序设置可提供的同样信息。

参数： config：指向含有如下字段排序列表的结构指针。这些字段与自定义程序的选项相匹配。设置的详细说明，请见自定义程序

字段	说明
uint32 mode	UART操作模式。可以选择下面各定义： SCB_UART_MODE_STD SCB_UART_MODE_SMARTCARD SCB_UART_MODE_IRDA
uint32 direction	UART操作方向。可以选择下面各定义： SCB_UART_TX_RX SCB_UART_RX SCB_UART_TX
uint32 dataBits	数据位数量
uint32 parity	确定奇偶校验。可以选择下面各定义： SCB_UART_PARITY_EVEN SCB_UART_PARITY_ODD SCB_UART_PARITY_NONE
uint32 stopBits	确定停止位的数量。可以选择下面各定义： SCB_UART_STOP_BITS_1 SCB_UART_STOP_BITS_1_5 SCB_UART_STOP_BITS_2
uint32 oversample	UART的过采样因子。 **注意**：使能enableIrdaLowPower时，过采样因子值被更改。 SCB_UART_IRDA_LP_OVS16 SCB_UART_IRDA_LP_OVS32 SCB_UART_IRDA_LP_OVS48 SCB_UART_IRDA_LP_OVS96 SCB_UART_IRDA_LP_OVS192 SCB_UART_IRDA_LP_OVS768 SCB_UART_IRDA_LP_OVS1536
uint32 enableIrdaLowPower	使能IrDA低功耗RX模式。 0 – 禁用 1 – 使能 当使能时，TX功能失效。
uint32 enableMedianFilter	0 – 禁用 1 – 使能

uint32 enableRetryNack	0 – 禁用 1 – 使能 除SmartCard外，对于其他模式，它将被忽略。
uint32 enableInvertedRx	0 – 禁用 1 – 使能 除IrDA外，对于其他模式，它被忽略。
uint32 dropOnParityErr	如果检测到奇偶校验错误，会丢失RX FIFO中的数据。 0 – 禁用 1 – 使能
uint32 dropOnFrameErr	如果检测到帧错误，会丢失RX FIFO中的数据。 0 – 禁用 1 – 使能
uint32 enableWake	0 – 禁用 1 – 使能 除标准UART模式外，对于其他模式，该字段均被忽略。须使能RX功能。
uint32 rxBufferSize	字中的RX缓冲区大小： • 数值‘8’代表硬件中缓冲的用途。 • 如果数值超过‘8’将产生软件缓冲区。 必须使能SCB_INTR _RX_NOT_EMPTY中断，以将数据传输到软件缓冲区内。
uint8* rxBuffer	为RX软件缓冲区提供的缓冲区空间： • 如果软件缓冲有指示，须提供NULL指针。 • 如果软件缓冲有指示，须提供分配缓冲区的指针。如果dataBits不大于8，则在字节中缓冲区的大小为（rxBufferSize + 1）；否则，在字节中缓冲区大小为（2 *（rxBufferSize + 1））。 软件RX缓冲区始终保持一个元素为空。为了能正确地操作，所分配的RX缓冲区必须比预期接收到的最大数据包尺寸大一个元素 。
uint32 txBufferSize	字中的TX缓冲区大小： • 数值‘8’代表硬件中缓冲的用途。 • 如果数值超过‘8’将产生软件缓冲区。
uint8* txBuffer	为TX软件缓冲区提供的缓冲区空间： • 如果软件缓冲有指示，须提供NULL指针。 • 如果软件缓冲有指示，须提供分配缓冲区的指针。如果dataBits不大于8，则缓冲区大小为txBufferSize；否则，其大小为 （2* rxBufferSize）。
uint32 enableMultiproc	使能多处理器模式。 0 – 禁用 1 – 使能
uint32 multiprocAcceptAddr	允许接收匹配地址。 0 – 禁用 1 – 使能
uint32 multiprocAddr	要在多处理器模式中匹配的8位地址。对于其他模式，将忽略它。

uint32 multiprocAddrMask	为匹配多处理器地址进行比较的地址位的8位掩码。对于其他模式，将忽略它。
uint32 enableInterrupt	0 – 禁用 1 – 使能 如果使用软件缓冲区，要使能中断。
uint32 rxInterruptMask	要在RX方向中使能的中断源掩码。此掩码的写入不受enableInterrupt字段的设置情况的影响。通过提供以下要使能的所有源的逻辑或（OR）值来使能多个源： SCB_INTR_RX_TRIGGER SCB_INTR_RX_NOT_EMPTY SCB_INTR_RX_FULL SCB_INTR_RX_OVERFLOW SCB_INTR_RX_UNDERFLOW SCB_INTR_RX_FRAME_ERROR SCB_INTR_RX_PARITY_ERROR
uint32 rxTriggerLevel	RX触发中断的FIFO等级。不论是否使能RX触发中断源，均写入该值。
uint32 txInterruptMask	要在TX方向中使能的中断源掩码。此掩码的写入不受enableInterrupt字段的设置情况的影响。通过提供以下要使能的所有源的逻辑或（OR）值来使能多个源： SCB_INTR_TX_TRIGGER SCB_INTR_TX_NOT_FULL SCB_INTR_TX_EMPTY SCB_INTR_TX_OVERFLOW SCB_INTR_TX_UNDERFLOW SCB_INTR_TX_UART_DONE SCB_INTR_TX_UART_NACK SCB_INTR_TX_UART_ARB_LOST
uint32 txTriggerLevel	TX触发中断的FIFO等级。不论是否使能TX触发中断源，均写入该值。

返回值： 无

其他影响： 无

void SCB_UartPutChar(uint32 txDataByte)

说明： 在下一个可用总线时间内放置传输缓冲区中要发送的一字节数据。该函数被禁用并等待直至有可用的空间可放置要求数据到传输缓冲区中为止。

在UART多处理器模式中可使用该函数发送9位数据。使用SCB_UART_MP_MARK添加标志，以便创建地址字节。

参数： uint32 txDataByte：所传输的数据。

返回值： 无

其他影响： 无

void SCB_UartPutString(const char8 string[])

说明： 在下一个可用的总线时间内放置传输缓冲区中要发送的以空字符结尾的字符串。

该函数被禁用并保持等待状态，直到传输缓冲区中有一个可用的空间时会将所有需要的数据放进去。

参数： const char8 string[]：指向要放置在传输缓冲区中的空结尾字符串阵列的指针。

返回值： 无

其他影响： 无

void SCB_UartPutCRLF(uint32 txDataByte)

说明： 向传输缓冲区放置一个字节的数据后，输入回车符（0x0D）和换行符（0x0A）。

该函数被禁用并保持等待状态，直到传输缓冲区中有一个可用的空间时会将所有需要的数据放进去。

参数： uint32 txDataByte：所传输的数据。

返回值： 无

其他影响： 无

uint32 SCB_UartGetChar(void)

说明： 从接收缓冲区检索下一个数据元素。该函数专为ASCII 字符设计并返回一个字符。其中1至255之间的数字均为有效字符，0代表出错或没有显示数据。

禁用RX软件缓冲区：返回从RX FIFO检索的数据元素。

使能RX软件缓冲区：从软件接收缓冲区返回数据元素。

参数： 无

返回值： uint32：来自接收缓冲区的下一个数据元素。1至255之间的ASCII字符值为有效的字符值。返回0则表示出现错误状况或无可用数据。

其他影响： 由于使用RX FIFO和软件缓冲区，错误位可能与读取字符不相符。

使能RX软件缓冲区：内部软件缓冲区溢出现象不做为错误条件。检查SCB_rxBufferOverflow以捕获错误条件。

uint32 SCB_UartGetChar(void)

说明： 从接收缓冲区检索下一个数据元素，并返回接收到的字节和错误条件。

禁用RX软件缓冲区：返回从RX FIFO检索的数据元素。如果RX FIFO为空，将返回未定义数据。

使能RX软件缓冲区：从软件接收缓冲区返回数据元素。

参数： 无

返回值： uint32：位15-8存放状态，位7-0存放来自接收缓冲区的下一个数据元素。如果位15-8均为非零值，则表示已出错。

其他影响： 由于使用RX FIFO和软件缓冲区，错误位可能与读取字符不相符。

禁用RX软件缓冲区：内部软件缓冲区溢出现象不做为错误条件。检查SCB_rxBufferOverflow以捕获错误条件。

void SCB_UartSetRxAddress(uint32 address)

说明： 在多处理器模式下，为UART设置硬件可检测的接收器地址。

参数： uint32 address：用于检测硬件地址的地址。

返回值： 无

其他影响： 无

void SCB_UartSetRxAddressMask(uint32 addressMask)

说明： 在多处理器模式下，为UART设置硬件地址掩码。

参数： uint32 addressMask：地址屏蔽。

位值为0 — 该位不进行地址对比。

位值为1 — 该位需要与地址的相应位相互匹配。

返回值： 无

其他影响： 无

void SCB_SpiUartWriteTxData(uint32 txData)

说明： 在传输缓冲区中放置将于下一个可用总线时间发送的数据输入。数据传输方向为最低有效位 (LSB)。

该函数被禁用，并等待直到传输缓冲区有可用空间时就将请求的数据存放进去。

在UART多处理器模式下，可使用该函数发送9位数据。使用SCB_UART_MP_MARK添加标志，以便创建地址字节。

参数： uint32 txData：所传输的数据。

需要传输的数据位数取决于所选择的数据位（数据位将从txDataByte的LSB开始计数）。

返回值： 无

其他影响： 无

void SCB_SpiUartPutArray(const uint16/uint8 wrBuf[], uint32 count)

说明： 将需要发送的数据阵列放置于传输缓冲区内。

该函数被禁用，并等待直到传输缓冲区有可用空间时就将所有的请求数据存放进去。

阵列大小可超过传输缓冲区的大小。

参数： const uint16/uint8 wrBuf[]：指向要放置在传输缓冲区中的数组的指针。需要作为一个阵列条目传输的数据位数取决于所选择的数据位（在每个阵列条目中，数据位将从LSB开始计数）。

uint32 count：要放置在传输缓冲区中的数据元素的数量。

返回值： 无

其他影响： 无

uint32 SCB_SpiUartGetTxBufferSize(void)

说明： 返回传输缓冲区中当前元素的数量。

禁用TX软件缓冲区：返回TX FIFO中用到的条目数量。

使能TX软件缓冲区：返回传输缓冲区中当前用到的元素数量。该数目不包含TX FIFO中用到的条目数量。在TX FIFO已满之前，传输缓冲区的大小为零。

参数： 无

返回值： uint32：准备传输的数据元素的数量。

其他影响： 无

void SCB_SpiUartClearTxBuffer(void)

说明： 清除传输缓冲区和TX FIFO。

参数： 无

返回值： 无

其他影响： 无

uint32 SCB_SpiUartReadRxData(void)

说明： 从接收缓冲区检索下一个数据元素。

禁用RX软件缓冲区：返回自RX FIFO检索的数据元素。如果RX FIFO为空，将返回未定义的数据。

使能RX软件缓冲区：从软件接收缓冲区返回数据元素。如果接收软件缓冲区为空，将返回零值。

参数： 无

返回值： uint32：来自接收缓冲区的下一个数据元素。

需要接收的数据位数取决于所选择的数据位（数据位将从返回值的LSB开始计数）。

其他影响 无

uint32 SCB_SpiUartGetRxBufferSize(void)

说明： 返回接收缓冲区中接收到的数据元素的数量。

使能RX软件缓冲区：返回RX FIFO中用到的条目数量。

使能RX软件缓冲区：返回放置在接收缓冲区中的元素的数量。

参数： 无

返回值： uint32：接收到的数据元素的数量

其他影响： 无

void SCB_SpiUartClearRxBuffer(void)

说明： 清除接收缓冲区和RX FIFO。

参数： 无

返回值： 无

其他影响： 无

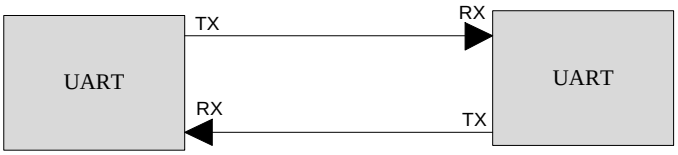
UART 的功能描述

通用异步发送器/接收器（UART）协议为异步串行接口。UART 的发送和接收接口由两种信号组成：

- **TX —** 发送器
- **RX —** 接收器

注意：SCB 不支持与流量控制相关联的 RS232 边带信号，例如 DTR（数据终端准备）和 DCD（数据载波检测）等。

图 12. UART 典型连接



标准操作模式

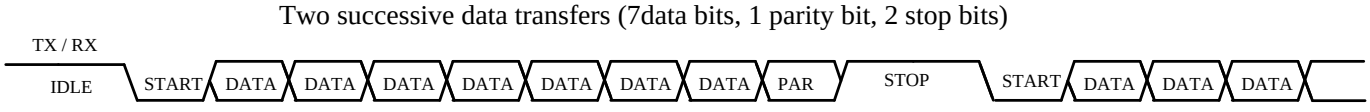
标准 UART 通过“点对点”拓扑结构定义。

典型的 UART 传输位依次由“起始位”、多个“数据位”、可选的“奇偶检验位”和一个或多个“停止位”组成。“起始位”的值始终为‘0’；“数据位”的值取决于传输的数据；“奇偶校验位”的值设置为可保证对“数据位”进行奇数或偶数校验的值；“停止位”的值为‘1’。“奇偶校验位”由发送器生成，可被接收器用来检测单个位传输的错误。不传输数据时，TX 线为‘1’，也就是与“停止位”的值相同。

可通过在 TX 线上将‘1’变为‘0’来实现从“停止位”到“起始位”的切换。接收器可利用这种切换与发送器的时钟保持同步。在每次数据传输开始时进行同步，这样即使在发送器和接收器时钟间出现频率偏移时也可实现无差错的数据传输。要求的时钟准确度取决于数据传输大小。

停止周期或连续数据传输间的“停止位”数量通常在发送器和接收器之间达成一致，通常在 1~3 个传输周期范围内。

图 13. UART 协议



UART 第 9 个数据位的使用

第 9 位被送到奇偶校验位的位置，它常被用于定义发送的数据是否是地址还是标准数据。奇偶校验位中的标记 （1） 表示发送的是地址，奇偶校验位中的空格 （0） 表示发送的是数据。数据流

为“起始位、数据位、奇偶校验、停止位”，与其他奇偶校验模式类似，但该位在传输前必须由用户软件控制，而非根据数据位的值计算得出。

```
tx_data = 0x31;
tx_data |= 0x100; /* Set 9th bit to indicate 'address' */
UART_SpiUartWriteTxData(tx_data)
```

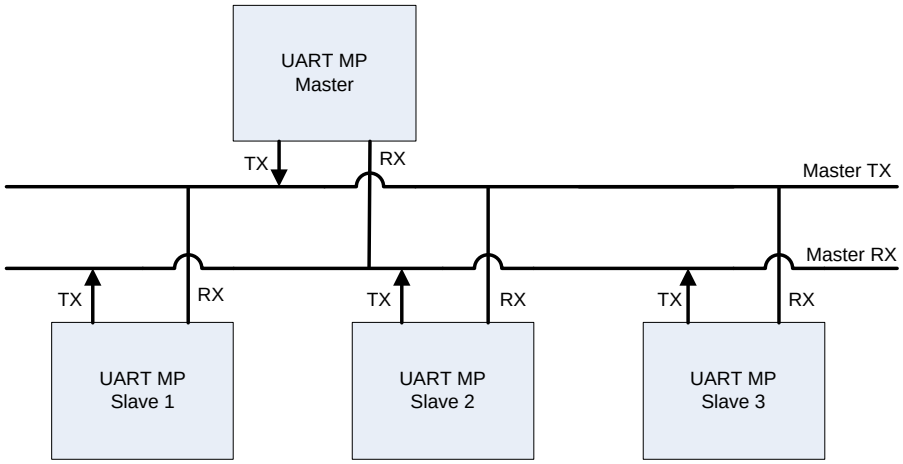
多处理器操作模式

用“单主设备多从设备”拓扑结构对此模式进行定义。多处理器模式又称 UART 9 位协议，标准 UART 协议使用 5 到 8 位数据字段。

多处理器模式具有以下的主要属性：

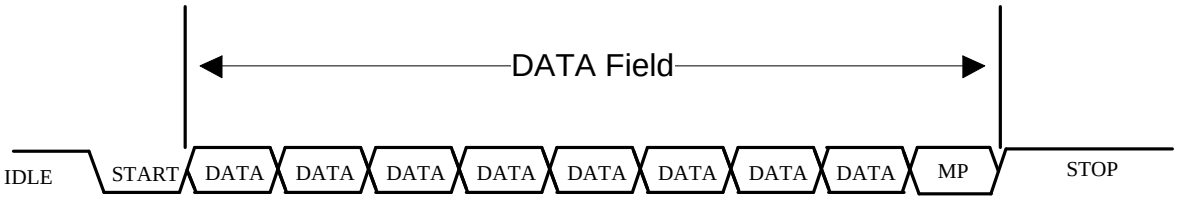
- 单个拥有多个从设备的主设备（多分支网络）
- 每个从设备都由独特的地址识别
- 使用 9 位数据段，把第 9 位（MSB）作为地址/数据标志。设为‘1’时，它表示地址字节；设为‘0’时，它表示数据字节。
- 奇偶校验位被禁用

图 14. 多处理器总线连接



为了使多处理器模式能按下面的选项配置 UART：**模式**：标准模式； **数据位**：9 位；**奇偶校验**：无。

图 15. 多处理器模式中的 UART 数据帧



因为多分支网络的数据链路层是一个用户定义协议，所以它为数据段的组成提供了一种灵活的定义方法。

地址帧中的所有位都能用来表示设备地址。另外，一些位可表示地址，而其他位可表示对从设备的指令，还有一些位可表示以下数据帧中数据的长度。

在多处理器模式中，SCB 可作为主设备或从设备使用。

当 UART 充当从设备时，收到的地址与**地址**和**掩码**匹配。在勾选 **Accept matching address in RX FIFO**（接受 RX FIFO 中的匹配地址）的选项时，匹配后的地址被写入到 RX FIFO 中。如果有匹配，随后收到的数据会被发送到 RX FIF 中 O。如果没有匹配，随后收到的数据会丢失，直到收到下一个地址作对比。

SmartCard (ISO7816) 操作模式

ISO7816 是异步串行接口，使用“单主设备单从设备”拓扑结构进行定义。组件中只支持主设备（读卡器）函数。

SCB 使用异步字符传输提供基础物理层支持，且只提供标准 ISO7816[3]引脚列表中的“I/O”引脚。通过在 TX 和 RX 控制模块间的内部复用，SCB 中 UART 的 TX 线将与 SmartCard 的 I/O 线相连接。

更高级的协议实现留给来自用户级的固件处理。

SmartCard 数据传输

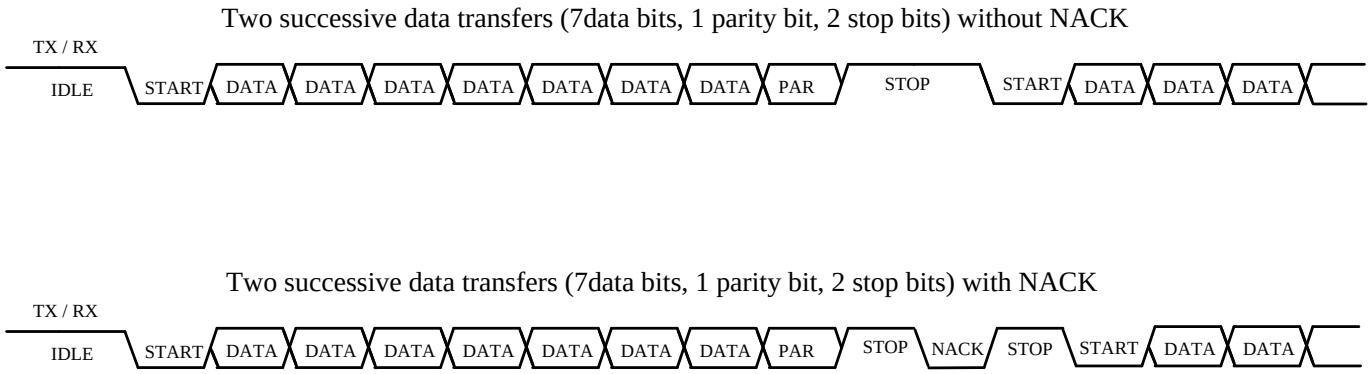
SmartCard 传输与 UART 传输相似，外加可能从接收器传送到发送器的否认(NACK)信号。NACK 总是设为‘0’。发送器和接收器在不同时刻驱动同一条 I/O 线路。图 16 阐释了 SmartCard 协议。

通常，协议的实现使用了带上拉电阻的三态驱动器，这样当线路未被驱动时，其值为‘1’（该值与未传输数据时或“停止位”的值相同）。

---

[3] 请访问 ISO 网站 www.iso.org，查找 *ISO/IEC 7816-3:2006 – 标识卡 – 集成电路卡 – 第 3 部分：带触点的卡 — 电气接口及传输协议*（1997）。

图 16. SmartCard 数据传输示例



SmartCard 传输让发送器驱动“起始位”、“数据位”和“奇偶校验位”。驱动这些位之后，它会通过释放总线来进入停止周期。线路中释放结果为‘1’（即“停止位“的值）。半个位传输期进入停止期后，接收器可能驱动线路上的 NACK （‘0’值）一个到两个位的传输期。发送器观察到此 NACK，通过延长停止期的方式做出反应，延长时间为一个位的传输期。要想让此协议奏效，停止期应长于一个位的传输期。

注意：带 NACK 的数据传输所用的时间比不带 NACK 的数据传输长一个数据传输周期。

协议 T=0 和 T=1

ISO7816 规范中有两种传输协议，T=0（异步字符的半双工传输）和 T=1（异步模块的半双工传输）。在物理层中，T=1 的模块是用异步字符实现的。

复位应答（ATR）

根据定义，复位应答是通过卡（从设备）发送到接口器件（读卡器或主设备）的字节的序列值，作为对复位的回应。在 I/O 线路上，每个字节都在异步字符中传输。

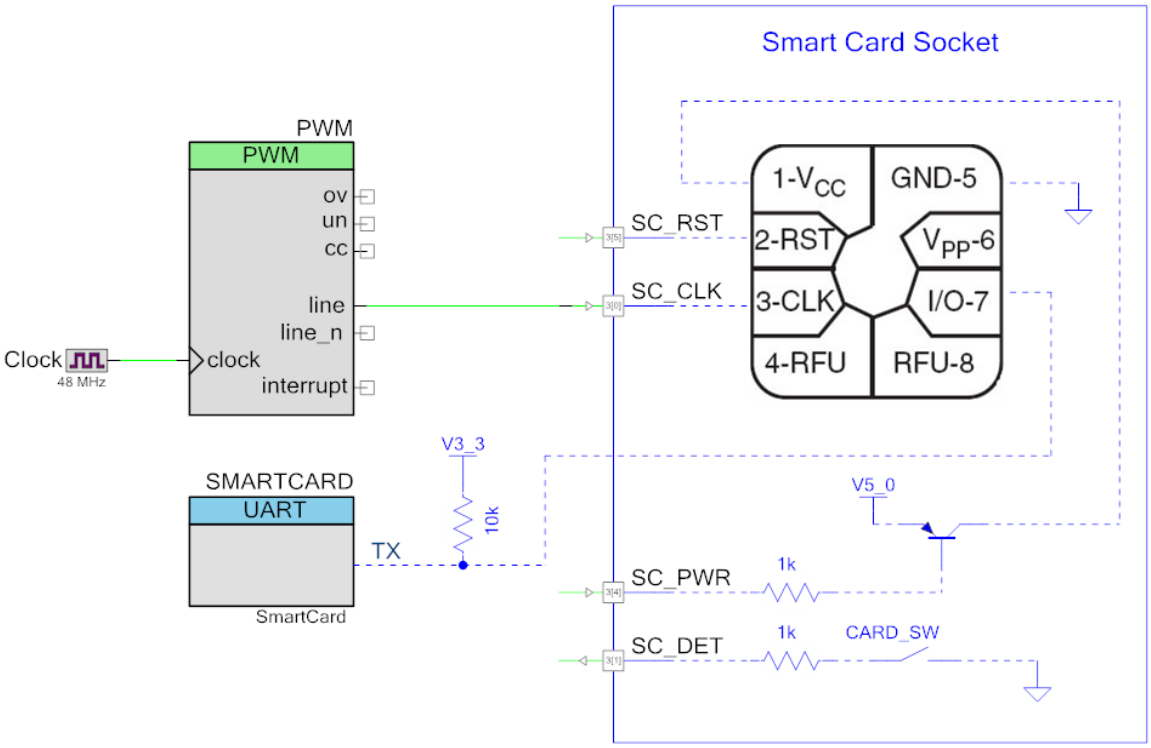
通信设置的过程（ATR，复位应答）、PPS（协议及参数选择）、操作条件类型的选择、操作模式选择和转换、NACK 上的再传输以及其他高级协议的实现都留给用户固件处理。

SmartCard 读卡器实现示例

您必须考虑如何用其他可用的系统资源为“RST”信号、“CLK”信号、卡检测信号和卡电源控制信号实现一个完整的 SmartCard 系统。

图 17 是用 TCPWM 和引脚组件实现 SmartCard 读卡器功能的示例。

图 17. SmartCard 读卡器实现示例



UART 与 I/O 卡触点相连接。必须将上拉电阻连接到此线路。SC_RST、SC_CLK 为标准卡触点。SC_DET 由于卡片插入检测。SC_PWR 用于控制卡电源的开或关。请参考 *ISO7816 规范*，了解更多详细信息。

IrDA 操作模式

用“点对点”拓扑结构定义 IrDA。SCB 仅从基础物理层为 IrDA 提供支持[4]，速率为 1200 bps 到 115200 bps。物理层负责定义数据传输的硬件收发器。更高级的协议实现留给来自用户级的固件处理。

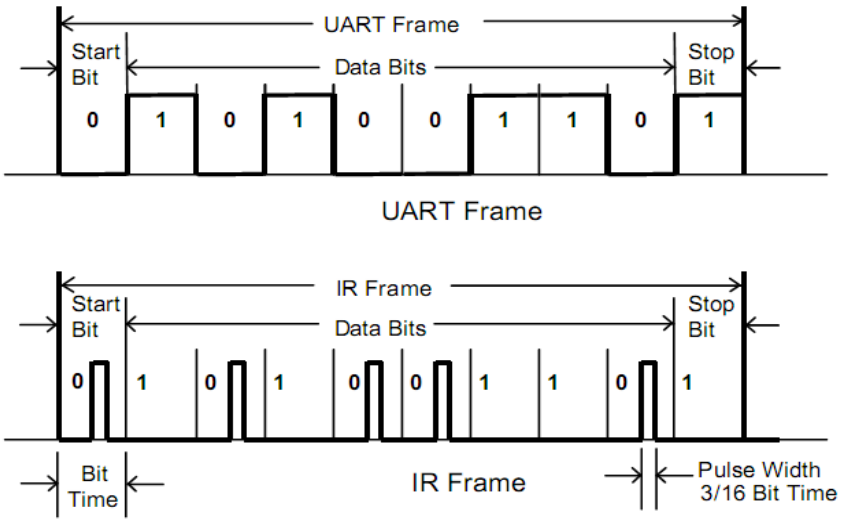
IrDA 对传输速率的最低要求仅为 9600 bps。所有的传输都必须从这个速率开始以确保兼容性。更高的速率是建立链接后端口协商的问题。

IrDA 协议给 UART 信号增加了一种调制方案。在发送器端，对比特进行调制。在接收器端，对比特进行解调。调制方案使用的是反相归零 (RZI) 格式。线路上短的‘1’脉冲表示的比特值‘0’，线路上‘0’脉冲表示的比特值‘1’。IrDA 使用 3/16 RZI 调制。

图 18 展示了 UART 帧和 IR 帧，它们均包含一个起始位和 8 个数据位，没有奇偶校验位，以停止位结束。

[4] 请访问 IrDA 网站 www.irda.org，查找 *IrPHY（IrDA 物理层连接规范）*（修订版 1.4，2001 年 5 月）

图 18. UART 帧和 IR 帧示例



过采样选择

IrDA 使用 3/16 RZI 调制，因此通过配置 **Oversampling**（过采样），采样时钟的频率应设置为所选 **Baud rate**（波特率）的 16 倍。IrDA 的 **Oversampling** 要保持为 16。

正常传输与低功耗传输

IrDA 运行模式有两种：

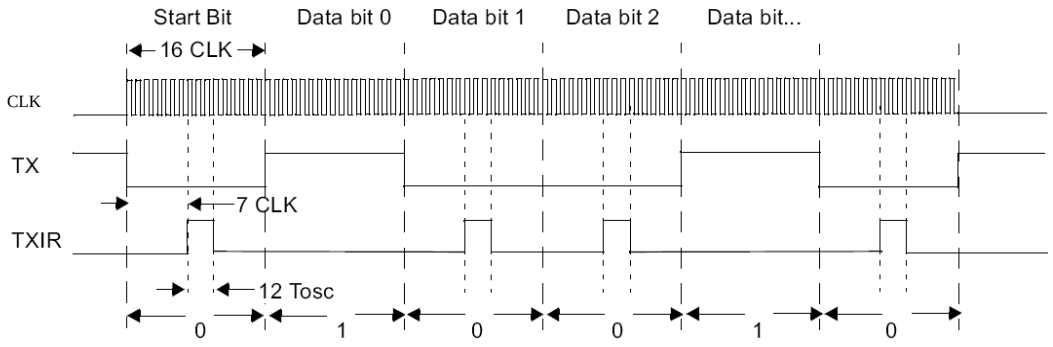
- **Normal transmission**（正常传输） — 脉冲宽度约为位周期的 3/16（所有波特率）
- **Low power transmission**（低功耗传输）— 脉冲宽度可能比位周期的 3/16 更短（通常降至 1.62µs，最小 1.41µs）（针对小于 115200 bps 的波特率）。仅支持 **RX** 方向。

Inverting RX（反转 RX）

此选择可用于支持下述两种解调方案。

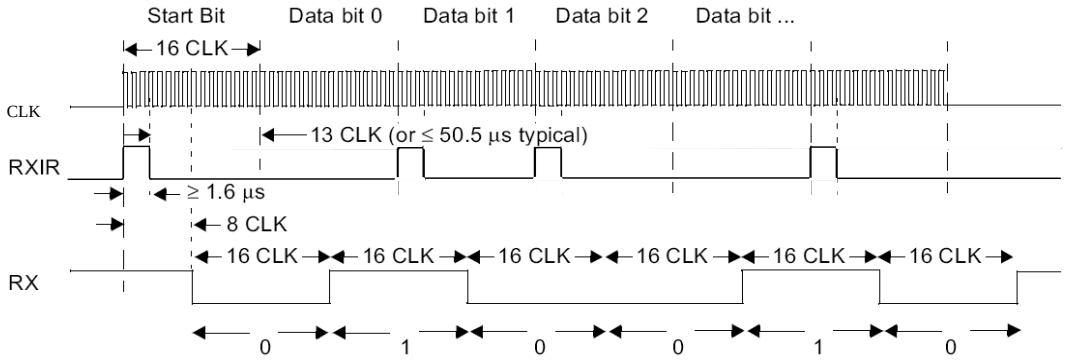
按照 IrPHY 规范，IR 帧调制（编码）方案如图 19 所示。

图 19. IR 帧调制方案



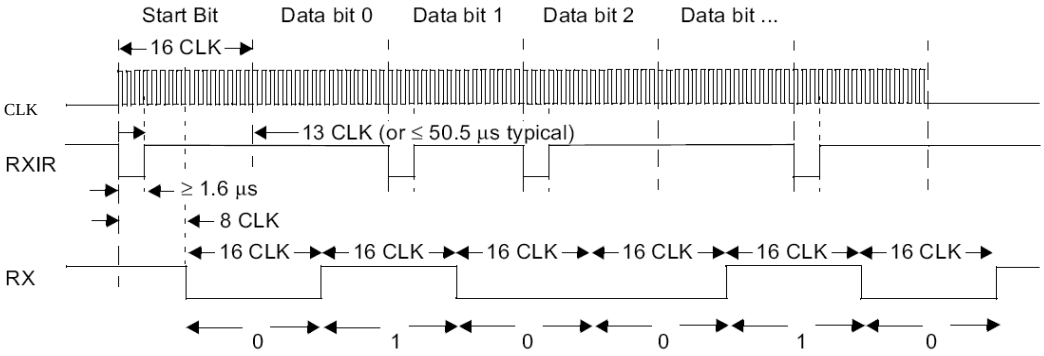
IR 帧解调（解码）方案如图 20 所示。RXIR 线路的电压为低电平预设，高电平有效。

图 20. IR 帧解调方案 1（高电平有效）



在应用中，来自 IrDA 收发器的 RXIR 帧输出通常是上拉式，高电平预设，低电平有效。图 21 显示的是另一种解调方案。

图 21. IR 帧解调方案 2（低电平有效）



软件缓冲区

SCB 包含 FIFO 存储器，FIFO 存储器是一种 16 字 x 16 位的 SRAM，允许写入字节。在 UART 模式中，FIFO 被拆分为 TX FIFO 和 RX FIFO。各含 8 个条目，每个条目 16 位。每个条目的 16 位宽度用于调节可配置的数据宽度。

内部中断处理程序用于提供软件和硬件 TX/RX 缓冲间的互动，而不对您的顶级固件进行任何更改。

也应当考虑到使用软件缓冲会导致被传输的字之间计时间隔更大，因为中断处理程序需要额外的时间来执行（取决于所选 HFCLK 的值）。

中断

当 **RX 缓冲区大小**或 **TX 缓冲区大小**超过 8 字节/字时，**RX FIFO 非空**并且 **TX FIFO 未满**中断源通过内部软件缓冲操作组件保留。**请勿清除或禁用它们**，因为这样做会造成错误的软件缓冲操作。但用户需要清除来自其他源的中断，因为它们不会被自动清除。那是客户处理程序函数的功能。

假如 **RX 缓冲区大小**或 **TX 缓冲区大小**不大于 8 个字节/字，仅使用硬件 TX 或 RX FIFO，而不使用软件缓冲区。在**内部**中断模式下，不会自动清除中断。这是用户的责任。这种情况下应首选**外部**中断模式。

低功耗模式

UART 模式中的组件可以作为低功耗的睡眠和深度睡眠模式的唤醒源。

对于外设来说，睡眠模式便是它的活动模式。这意味着进入/退出此模式之前，不需要对组件或将要调用的代码进行任何配置更改。在 TX 或 RX 方向上发生任何生成中断的 UART 操作均会唤醒器件。

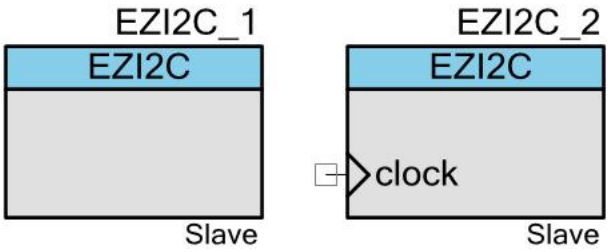
深度睡眠模式要求适当将 UART 配置为唤醒源。在从设备配置对话框中，必须勾选“Enable wakeup from Sleep mode”（使能睡眠模式中的唤醒），并使能 RX 方向。进入/退出深度睡眠模式前/后，必须调用 SCB_Sleep()和 SCB_Wakeup()函数。

通过由引入起始位产生的 RX GPIO 下降沿事件唤醒器件。唤醒受两大限制：

- 唤醒数据操作的第一个数据位必须为‘1’。UART 跳过起始位并在第一个数据位同步。
- 器件的唤醒时间必须小于一个位的持续时间。在其它情况下收到的数据不正确。

注意：RX GPIO 中断限制了使用来自放置 UART rx 引脚的端口的所有 GPIO 中断。

EZI2C



EZI2C 从设备模式实现了基于 I²C 寄存器的从设备。它与 I²C 标准、快速和超快速模式的设备兼容，装置见 NXP I2C 总线规范。I²C 总线是基于 Philips® 开发的行业标准的两线硬件接口。主设备在 I²C 总线上启动所有通信，并为所有从设备提供时钟。EZ I²C 从设备支持高达 1000 kbps 的标准数据速率，且与同一总线上的多个组件兼容。

EZ I²C 从设备是 I²C 从设备的唯一实现，主设备和从设备之间的所有通信都在 ISR（中断服务子程序）中处理，不需要与主程序流交互。该接口表现为主设备与从设备之间的共享存储器。一旦执行了 EZI2C_Start()函数，则几乎不再需要与 API 交互。

输入/输出接口

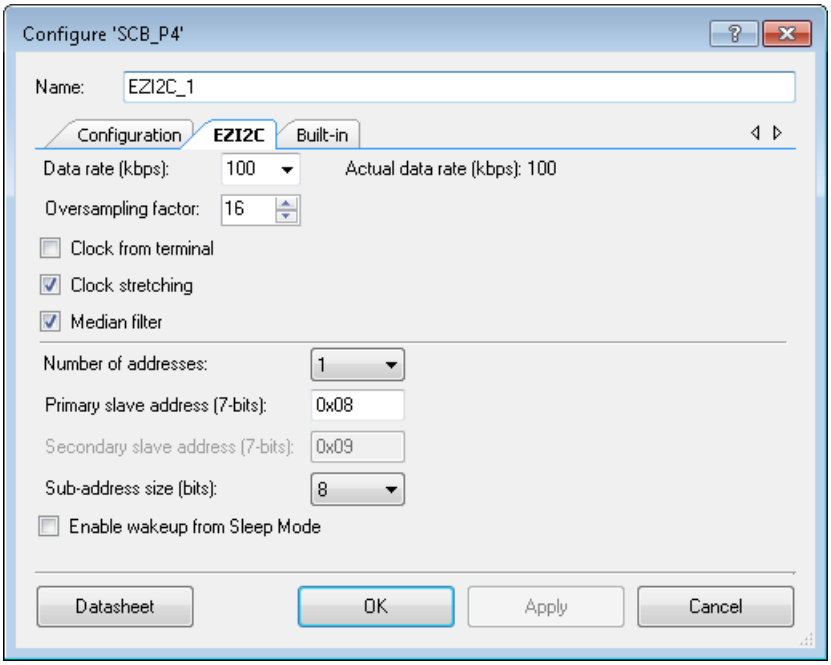
本部分描述了 SCB 组件的各种输入/输出接口。I/O 列表中的星号（*）表示，在 I/O 说明部分中所列出的情况下，该 I/O 可能不可见。

时钟 — 输入*

时钟负责运行该模块。只有将 **Clock from terminal**（终端时钟）复选框勾选上时，时钟接口才可见。

在组件中接口特定引脚都不可见，因为这些引脚只能连接到特定的 IO，不具备通用信号的可布线性。了解更多信息，请参见芯片*技术参考手册（TRM）*中 *“I/O 系统”* 部分的内容。

EZI2C 参数



EZI2C 选项卡具有下列各参数：

Data rate（数据速率）

此参数用于设置高达 1000 kbps 的 I^2C 数据速率值（对于 PSoC 4000 系列，该值为 400 kbps）；实际速率可能会基于可用的时钟速度和分频器的分频范围而有所不同。标准数据速率为 50、100（默认值）、400 和 1000 kbps。如果 **Clock Stretching** 选项被禁用，**数据速率** 的最大值为 400kHz。如果设置了 **Clock from terminal**（终端时钟），则可忽略 **Data rate** 参数；输入时钟频率与 **Oversampling factor**（过采样因子）共同决定实际数据速率。

Actual data rate （实际数据速率）

实际数据速率显示的是将用当前设置运行的组件上的数据速率。所选数据速率可能和实际数据速率不同。影响实际数据速率计算的因子包括： 过采样因子、HFCLK 时钟和内部或外部组件时钟的准确度。

Oversampling factor（过采样因子）

此参数定义了 I^2C SCL 时钟的过采样因子，即一个 I^2C SCL 时钟内组件时钟的数量。**Oversampling factor** 用于计算出获取定义 **Data rate** 的过采样量时所需的内部组件时钟频率。过采样因子的最大值为 32；最小值则取决于 **Median filter**（中值滤波器）的设置。未选取中值滤波器 — 12； 选取了中值滤波器 — 14。默认值为 16。

Clock from terminal（终端时钟）

该参数允许在内部配置时钟和外部配置时钟之间进行选择，以生成数据率。使能该选项时，组件不对数据速率进行控制，但会基于用户连接的时钟源和组件过采样因子来显示实际数据速率。当该选项无效时，PSoC Creator 将自动配置所需要的时钟源。将根据 **Data rate** 参数和过采样因子的组合来计算出时钟频率。

注意：当设置数据速率或外部时钟频率值时，请确保 PSoC Creator 可以用当前系统时钟频率提供该值。否则，编译项目时会生成时钟准确度范围警告。警告中包含 PSoC Creator 所设置的实际时钟值。选择是应当更改系统时钟还是组件时钟以满足时钟设置系统要求，达到最佳值。

Clock Stretching （时钟延展）

如果 EZ I^2C 从设备尚未响应，该参数适用于 SCL 线路上的时钟延展。使能该选项能够确保从设备操作的一致性。因为在发生任何 EZ I^2C 从设备中断延迟时，时钟延展将暂停 I^2C 传输。如果未勾选时钟延展选项，设计需要快速服务 EZ I^2C 从设备中断，以支持从设备的准确操作。

Median filter（中值滤波器）

该参数 I2C SDA 信号输入路径上实现 3 抽头中值滤波器。该滤波器会降低对错误的敏感性。然而，过采样因子的最小值将增加。

Number of addresses（地址数）

此选项确定是识别一个（默认值）还是两个独立 I^2C 从设备地址。如果识别了两个地址，那么将通过软件执行地址检测。当 **Clock Stretching** 选项被**禁用**时，只能选择一个地址数量。

Slave address （从设备地址（7 位））

这是从设备作为主要从设备需要识别的 I^2C 地址。该地址为右对齐的 7 位从设备地址，它不包括读/写位。可以选择介于 0 至 127 之间的从设备地址，默认值为 **8**。

可以输入十进制或十六进制的值；对于十六进制数值，请在地址前面键入“0x”。

Secondary slave address（辅助从设备地址(7 位)）

这是从设备作为辅助从设备需要识别的 I^2C 地址。该地址为右对齐的 7 位从设备地址，它不包括读/写位。可以选择介于 0 至 127 之间的从设备地址，默认值为 **9**。请参见 更好的辅助地址选择。

可以输入十进制或十六进制的值；对于十六进制数值，请在地址前面键入“0x”。

Sub-address Size（辅助地址大小）

该选项确定可以访问的数据范围。可以选择 **8** 位（默认值）或 **16** 位辅助地址。如果使用 8 位的辅助地址大小，则主设备只能访问 0 到 255 之间的数据偏移。您还可以选择 16 位辅助地址大小。这将允许 I^2C 主设备访问最大达 65,536 字节的数据数组。

Enable wakeup from Sleep Mode（使能从睡眠模式唤醒）

此选项会在从设备地址匹配时将系统从睡眠模式唤醒。

使能该选项时，会添加下面的限制（仅限于 PSoC 4100/PSoC 4200 器件）：

- 必须使能时钟延展
- 必须禁用中值滤波器
- 从设备地址必须保证为偶数（位 0 的值为零）

请参考本文档的 EZI2C 章节中低功耗模式部分的内容，另请参考*系统参考指南*中的*电源管理 API* 一节，了解更详细的信息。

EZI2C 的 API

通过应用编程接口（API），可以使用软件对组件进行配置。下表列出并说明了每个函数的接口。以下各节将更详细地介绍了每个函数。

默认情况下，PSoC Creator 将实例名称“SCB_1”分配给第一个添加到工作区（TopDesign）的 SCB 组件。可以将该实例重新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、变量和符号常量的前缀。为了便于读取，下表中使用的实例名称为“SCB”。

函数	说明
SCB_Init()	根据自定义程序中定义的参数初始化SCB组件。
SCB_Enable()	使能SCB组件操作。
SCB_Start()	启动SCB组件。
SCB_Stop()	禁用SCB组件。
SCB_Sleep()	准备SCB组件进入深度睡眠模式。
SCB_Wakeup()	使SCB组件退出深度睡眠模式。
SCB_EzI2CInit()	配置SCB组件，以在EZ I^2C模式中运行。
SCB_EzI2CGetActivity()	返回EZ I^2C从设备状态。
SCB_EzI2CSetAddress1()	设置主EZ I^2C从设备地址。
SCB_EzI2CGetAddress1()	返回主要的EZ I^2C从设备地址。

函数	说明
SCB_EzI2CSetBuffer1()	设置数据缓冲区，在主从设备地址请求中公开给I²C主设备。
SCB_EzI2CSetReadBoundaryBuffer1()	在主要地址请求中，通过I²C主设备设置公开的数据缓冲区的只读边界。
SCB_EzI2CSetAddress2()	设置辅助的 EZ I²C 从设备地址。
SCB_EzI2CGetAddress2()	返回辅助EZ I²C从设备地址。
SCB_EzI2CSetBuffer2()	设置数据缓冲区，在辅助从设备地址请求中公开给I²C主设备。
SCB_EzI2CSetReadBoundaryBuffer2()	在辅助地址请求中，通过I²C主设备设置公开的数据缓冲区的读取边界。

全局变量

正常运行时，不需获得这些变量。

变量	说明
SCB_initVar	SCB_initVar表示SCB组件是否完成了初始化。该变量被初始化为0，并在第一次调用SCB_Start()时设置为1。这样，第一次调用SCB_Start()子程序后，不用重新初始化组件即可重启。 如需重新初始化组件，可在调用SCB_Start()或 SCB_Enable()函数前，先调用SCB_Init()函数。

void SCB_Init(void)

说明： 初始化SCB组件，以便在所选的配置之一中运行： I²C、SPI、UART或EZ I²C。

被设置为“未配置SCB”时，该函数不进行任何初始化操作。这时，将使用模式特定的初始化API： SCB_I2CInit、SCB_SpiInit、SCB_UartInit或SCB_EzI 2CInit。

参数： 无

返回值： 无

其他影响： 无

void SCB_Enable(void)

说明： 使能SCB组件。

组件使能时，不应更改SCB配置。禁用组件后，方可更改配置。

当将配置设为“未配置SCB”时，首先必须初始化组件，使其操作于下列某一种配置状态：I²C、SPI、UART 或EZ I²C。否则，该函数将不会使能组件。

参数： 无

返回值： 无

其他影响： 无

void SCB_Start(void)

说明： 调用SCB_Init() 与 SCB_Enable()。

调用该函数后，组件启动且准备就绪运行。

当将配置设为“未配置SCB”时，首先必须初始化组件，使其操作于下列某一种配置状态：I^2C、SPI、UART 或 EZ I^2C。否则，该函数将不会使能组件。

参数： 无

返回值： 无

其他影响： 无

void SCB_Stop(void)

说明： 禁用SCB组件及其中断。

参数： 无

返回值： 无

其他影响： 无

void SCB_Sleep(void)

说明： 准备组件进入深度睡眠。

“从睡眠模式使能唤醒”的选择操作对该函数的使用产生影响。

在调用CyPmSysDeepSleep()函数之前，先调用SCB_Sleep()函数。有关功耗管理函数的详细信息，请参考PSoC Creator*系统参考指南*。

进入睡眠模式前不得调用此函数。

参数： 无

返回值： 无

其他影响： 无

void SCB_Wakeup(void)

说明： 退出深度睡眠模式后，为组件进入活动模式做准备。

“从睡眠模式使能唤醒”的选择影响了该函数的实现。

退出睡眠模式后不得调用该函数。

参数： 无

返回值： 无

其他影响： 在调用SCB_Sleep()函数前，调用SCB_Wakeup()函数可能会导致意外行为。

void SCB_EzI2CInit(SCB_EZI2C_INIT_STRUCT *config)

说明： 配置SCB以运行EZ I^{2}C。

该函数**专为**当自定义程序中的SCB配置设为 “未配置SCB”时使用。在EZ I^{2}C模式中初始化SCB后，组件可用 SCB_Start()或者SCB_Enable()函数启动。

该函数为提供配置设置的结构使用了一个指针。该结构所含的信息自定义程序设置同样可提供。

参数： config：针对含有该字段排序列表的指针。这些字段与自定义程序的选项相匹配。设置的详细说明，请见自定义程序

字段	说明
uint32 enableClockStretch	0 – 禁用 1 – 使能 使能该字段时，将根据要求延展SCL，以进行正确的操作。
uint32 enableMedianFilter	0 – 禁用 1 – 使能
uint32 numberOfAddresses	受支持的地址的数量： SCB_EZI2C_ONE_ADDRESS SCB_EZI2C_TWO_ADDRESSES
uint32 primarySlaveAddr	主7位从设备地址。
uint32 secondarySlaveAddr	辅助7位从设备地址。
uint32 subAddrSize	辅助地址的大小： SCB_EZI2C_SUB_ADDR8_BITS SCB_EZI2C_SUB_ADDR16_BITS
uint32 enableWake	0 – 禁用 1 – 使能 使能时，匹配的地址将会生成一个唤醒请求。

返回值： 无

其他影响： 无

uint32 SCB_EzI2CGetActivity(void)

说明： 返回EZ I²C从设备状态。

调用该函数后，读、写和错误状态标志均被复位为0。

当完成对EZ I²C从设备的某个数据操作时，繁忙状态标志被清除。

该函数将禁用EZ I²C从设备中断，使从设备能够正确地操作。一旦禁用了时钟延展选项，EZ I²C从设备操作的准确性会显著受到影响。禁用中断的时长应小于EZ I²C从设备中断的延迟。请参考“禁用时钟延展”一节，了解更多的信息。

参数： 无

返回值： uint32：EZ I²C从设备的当前状态。

这个状态包含了状态常量的数量。每个常量是一个位字段值。返回值可能包含多个位，用于指示传输的状态。

从设备状态常数	说明
SCB_EZI2C_STATUS_READ1	读取传输已完成。传输操作使用了主从设备地址。 必须检查错误条件状态位，以确保读取传输已经完成。
SCB_EZI2C_STATUS_WRITE1	写入传输已完成。该缓冲区的内容被修改。传输操作使用了主从设备地址。 必须检查错误条件状态位，以确保写入传输已经完成。
SCB_EZI2C_STATUS_READ2	读取传输已完成。传输操作使用了辅助从设备地址。 必须检查错误条件状态位，以确保读取传输已经完成。
SCB_EZI2C_STATUS_WRITE2	写入传输已完成。该缓冲区的内容被修改。传输操作使用了辅助从设备地址。 必须检查错误条件状态位，以确保写入传输已经完成。
SCB_EZI2C_STATUS_BUSY	正在进行对主地址或辅助地址的传输。当某个地址得到匹配时，将设置状态位。在停止或重启条件下，该位将被清除。
SCB_EZI2C_STATUS_ERR	对主从设备地址或辅助从设备地址进行传输的过程中，发生了一个错误。错误来源为：启动或停止条件的错误或从设备驱动SDA时仲裁丢失。 置位SCB_EZI2C_STATUS_ERR位时，从设备缓冲区可能会包含无效的字节。这种情况下，推荐清除缓冲区的内容。

其他影响： 无

void SCB_EzI2CSetAddress1(uint32 address)

说明： 设置主EZ I²C 从设备地址。

参数： uint32 address：主I²C 从设备地址。

该地址为右对齐的7位从设备地址，它不包括读/写位。

本地址的值未就其是否违反 I²C规范加以检查。推荐地址范围为8至120之间（0x08到0x78）。

返回值： 无

其他影响： 无

uint32 SCB_EzI2CGetAddress1(void)

说明： 返回主EZ I²C从设备地址。

该地址为右对齐的7位从设备地址，它不包括读/写位。

参数： 无

返回值： uint32：主I²C从设备地址。

其他影响： 无

void SCB_EzI2CSetBuffer1(uint32 bufSize, uint32 rwBoundary, volatile uint8 * buffer)

说明： 设置数据缓冲区，在主从设备地址请求中公开给I²C主设备。

参数： uint32 bufSize：数据缓冲区的大小（以字节为单位）。

uint32 rwBoundary：数据字节的数量，起始于读取和写入访问缓冲区的开头。位于偏移rwBoundry或更远位置处的数据字节是只读的。

此值必须小于或等于缓冲区大小。

uint8* buffer：指向数据缓冲区的指针。

返回值： 无

其他影响： 在对主从设备地址进行传输导致意外行为的中间，请调用该函数。

void SCB_EzI2CSetReadBoundaryBuffer1(uint32 rwBoundary)

说明： 设置数据缓冲区中的只读边界，在主从设备地址请求中公开给I^2C主设备。

参数： uint32 rwBoundary：数据字节的数量，起始于读取和写入访问缓冲区的开头。位于偏移rwBoundry或更远位置处的数据字节是只读的。

此值必须小于或等于缓冲区大小。

返回值： 无

其他影响： 在对主从设备地址进行传输导致意外行为的中间，请调用该函数。

void SCB_EzI2CSetAddress2(uint32 address)

说明： 设置辅助EZ I^2C从设备地址。

参数： uint32 address: 辅助I^2C从设备地址。

该地址为右对齐的7位从设备地址，它不包括读/写位。

该地址的值未就其是否违反I2C规范加以检查。推荐地址范围为8至120之间（0x08到0x78）。

返回值： 无

其他影响： 无

uint32 SCB_EzI2CGetAddress2(void)

说明： 返回辅助EZ I^2C从设备地址。

该地址为右对齐的7位从设备地址，它不包括读/写位。

参数： 无

返回值： uint32: 辅助I^2C从设备地址。

其他影响： 无

void SCB_EzI2CSetBuffer2(uint32 bufSize, uint32 rwBoundary, volatile uint8 * buffer)

说明： 设置数据缓冲区，在辅助从设备地址请求中公开给I²C主设备。

参数： uint32 bufSize：数据缓冲区的大小（以字节为单位）。

uint32 rwBoundary：数据字节的数量，起始于读取和写入访问缓冲区的开头。位于偏移rwBoundry或更远位置处的数据字节是只读的。

此值必须小于或等于缓冲区大小。

uint8* buffer：数据缓冲区的指针。

返回值： 无

其他影响： 在对辅助从设备地址进行传输导致意外行为的中间，请调用该函数。

void SCB_EzI2CSetReadBoundaryBuffer2(uint32 rwBoundary)

说明： 设置数据缓冲区中的只读边界，在辅助地址请求中公开给I²C 主设备。

参数： uint32 rwBoundary：数据字节的数量，其起始于读取和写入访问缓冲区的开头。。位于偏移rwBoundry或更远位置处的数据字节是只读的。

此值必须小于或等于缓冲区大小。

返回值： 无

其他影响： 打算对辅助从设备地址进行传输导致意外行为的中间，请调用该函数。

EZI2C 的功能说明

该组件支持具有一个或两个 I²C 地址的 I²C 从设备。任一地址都可以访问 RAM 或闪存数据空间中定义的存储器缓冲区。闪存存储器缓冲区是只读的，而 RAM 缓冲区可以是读写的。这些地址右对齐。

由于 I²C 硬件是中断驱动的，因此在使用该组件时，必须使能全局中断。即使该组件需要中断，您也不必向 ISR（中断服务子程序）添加任何代码。该组件将独立于代码之外为所有中断（数据传输）提供服务程序。为此接口分配的存储器缓冲区看上去类似于您的应用与 I²C 主设备之间的简单双端口存储器。

如果需要，可以通过在数据结构中定义信号和指令位置，在主设备与从设备之间创建更高级别接口。

存储器接口

对于 I²C 主设备，该接口看上去非常类似于通用 I²C EEPROM。可将 EZ I²C 缓冲区配置为变量、数组或结构，但它首选作为数组使用。该缓冲区通过 I²C 总线充当您的程序与 I²C 主设备之间共享

的存储器接口。该组件允许读取和写入 I²C 主设备对指定的缓冲区存储器进行访问，防止对缓冲区外的空间进行访问或对只读区域进行写入操作。

例如：如果主从设备地址的缓冲区是使用下面的代码进行配置的。4 到 9 号缓冲区是只读的。

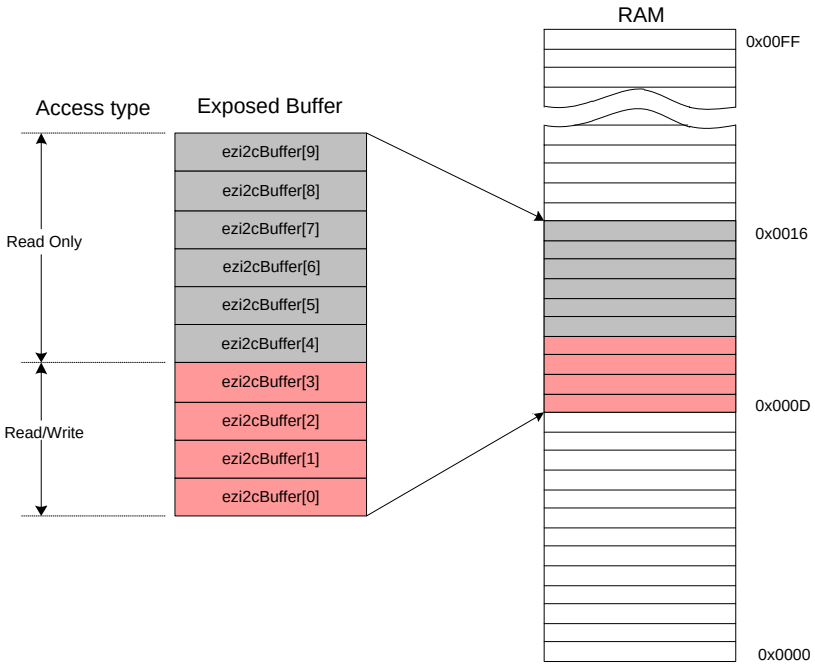
```
#define BUFFER_SIZE             (0x0Au)
#define BUFFER_RW_BOUNDARY      (0x04u)

uint8 ezi2cBuffer[BUFFER_SIZE];

SCB_EzI2CSetBuffer1(BUFFER_SIZE, BUFFER_RW_BOUNDARY, ezi2cBuffer);
```

缓冲区 ezi2cBuffer 按图 22 分配。

图 22. EZ I²C 缓冲区公开给一个 I²C 主设备



配置整个缓冲区为读取和写入访问，缓冲区的大小以及读取/写入边界需要使用相同的数值。例如：

```
SCB_EzI2CSetBuffer1(BUFFER_SIZE, BUFFER_SIZE, ezi2cBuffer);
```

处理字节顺序

可将 EZ I²C 缓冲区设置为一个变量。大于一个字节的变量，则需要了解有关字节顺序方面的知识（低位优先或高位优先）。它将决定在 I²C 总线上的字节顺序。处理合适的字节顺序是 I²C 主设备的责任。

```
SCB_EzI2CSetBuffer1(BUFFER_SIZE, BUFFER_SIZE, ezi2cBuffer);
```

所有 PSoC 4 器件都是低位优先的设备，因此主设备将按照顺序读取这两个字节：0xBB 0xAA。

处理结构

可将 EZ I²C 缓冲区按照结构进行设置。编译器列出了存储器的结构，并且可以添加额外的字节。这个被称为填装。编译器将添加这些字节用于使文件结构对齐，使其满足 Cortex-M0 的要求。该处理器不支持对多字节字段的非对齐访问。当使用某个结构时，该应用必须将这种对齐方式考虑在内。如果需要封装字段，则需要使用字节阵列，而不是某个结构。

处理某个状态字节

要定义一个高级协议，则需要使用 EZ I²C 缓冲区内的一个状态字节。该状态字节应通过 I²C 主设备修改，但编译器不用知道中断例程是否修改了该缓冲区。这样可以使编译器优化实现“while”循环，在该“while”循环中测试状态字节的变化。必须使用易失性关键词通知编译器，即使程序中没有语句显示修改状态字节，但该字节也可能改变状态。

代码示例：

```
#define BUFFER_SIZE        (0x0Au)
#define STATUS_BYTE_POS    (0u)

volatile uint8 ezi2cBuffer[BUFFER_SIZE];

SCB_EzI2CSetBuffer1(BUFFER_SIZE, BUFFER_SIZE, ezi2cBuffer);

ezi2cBuffer[STATUS_BYTE_POS] = 0x01u;
while(0x01u == ezi2cBuffer[STATUS_BYTE_POS])
{
    /* Wait for status byte to be changed by the master */
```

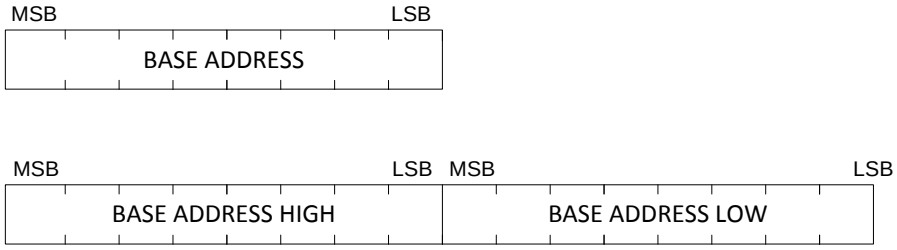
外部主设备可视的接口

EZ I²C 从设备组件支持读写区域的基本读写操作和只读区域的读取操作。两个 I²C 地址接口使用单独基地址寻址的单独数据缓冲区。基地址是一个位于 EZ I²C 缓冲区内的索引，它的大小为从 0 到缓冲区大小 - 1。首先是基地址，然后是数据字节。基地址大小是根据 **Sub-address size（辅助地址大小）** 参数来决定的：一个字节(Sub-address size = 8bits)或两个字节(Sub-address size = 16bits)。8 位的辅助地址用于对 256 字节的缓冲区进行访问；16 位的辅助地址用于对 65535 字节

的缓冲区进行访问。如果在 16 位的数值中，第一个字节地址为高字节，第二个字节地址为低字节， 如图 23 所示。

例如，需要访问的基地址为 0x0201：高字节为 0x02，低字节为 0x01。

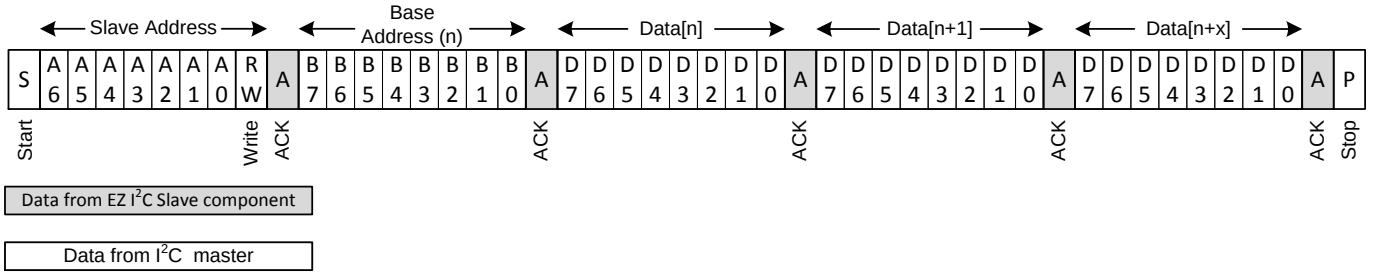
图 23. 8 位和 16 位辅助地址大小



对于写入操作，一般都提供了基地址，并且根据配置情况基地址为一个字节或者两个字节。保留基地址，并且在后面的读取操作中使用。基地址后面是一个字节序列，这些字节序列从基地址的位置开始写入缓冲区。缓冲区索引会根据每个写操作字节而递增，但它不会影响基地址，它仍被保留。写入操作的长度受最大缓冲区读写区域大小的**限制**。当主设备尝试对读写范围外的区域进行写操作，或通过缓冲区的末尾时，根据时钟延展的设置，EZ I²C 从设备在 I²C 总线上的操作有所不同：

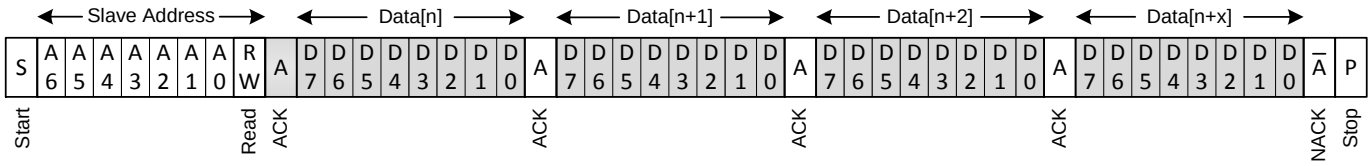
- 使能：字节被从设备否认，主设备停止当前的传输操作。否认的字节被从设备接收。
- 禁用：由从设备确认的所有写入字节，但是这些字节均被丢弃。

图 24. I²C 主设备将 X 个字节写入到 EZ I²C 从设备缓冲区内。



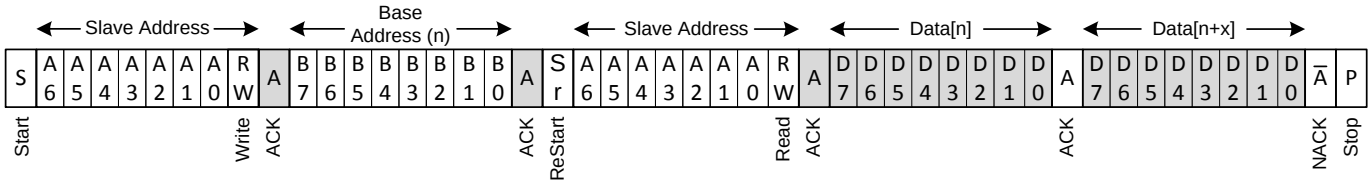
读操作**总是开始**于最新的写操作所设置的基地址。每个读字节的缓冲区索引都是递增的。无论读取了多少个字节，两个连续的读操作都从相同的基地址开始。读取操作的长度**不受**数据缓冲区最大尺寸的限制。如果读操作已经通过了缓冲区的结尾，则 EZ I²C 从设备将返回 0xFF 字节。

图 25. I²C 主设备将从 EZ I²C 从设备缓冲区内读取 X 个字节。



一般情况下，进行某个读操作前需要更新基地址。此时，需要结合写入和读取操作。I²C 主设备可能会使用“重启”或“停止/启动”等条件来结合这两种操作。写入操作只设置基地址，随后的读取操作将从新的基地址开始读取。如果基地址仍保持不变，则不需要进行写操作。

图 26. I²C 主设备将设置基地址，并从 EZ I²C 从设备缓冲区内读取 X 个字节。



NXP 网站上所提供的完整的 I²C 规范中，并且通过参考组件数据手册，您可以获得 I²C 总线的详细说明以及实现方法。

数据一致性

虽然数据缓冲区可以包含大于单字节的数据结构，但是主设备读取或写入操作仍由多个单字节操作组成。这可能会引起数据的一致性问题，因为没有任何机制能够确保多字节读取或写入操作在接口两侧（主设器件和从设备）是同步的。例如，考虑一个包含单一的两字节整数的缓冲区。虽然主设备每次读取 2 字节整数的一个字节，但是从设备可能在主设备读取整数的第一个字节（LSB）到要读取第二个字节（MSB）期间已经更新了整个整数。主设备读取的数据可能无效，因为 LSB 读取自原始数据，而 MSB 读取更新的值。

您必须在主设备、从设备或二者上提供一个机制，以确保在另一方读取或写入数据时主设备或从设备不会进行更新。可以使用 SCB_EzI2CGetActivity()函数开发特定于应用的机制。

注意：当组件被禁用或从设备不繁忙时，用于设置缓冲区的 API 的中断将不受保护，并且必须调用它。

Clock Stretching （时钟延展）

时钟延展可以通过将 SCL 线路保持为低电平来暂停操作。不能继续进行操作，直到 SCL 线得到释放，允许信号再次变高为止。时钟延展的支持是 I^2C 规范的可选性能。因此，EZ I^2C 从设备将提供一个用于使能或禁用该性能的选项。

Clock Stretching Enable（使能时钟延展）

使能时钟延展选项会使从设备能够在字节级别上将一个暂停功能插入到操作中。这样可以保证任何从设备中断延迟的 EZ I^2C 从设备操作都一致。该选项的缺点是主设备也需要支持时钟延展。

Clock Stretching Enable（禁止时钟延展）

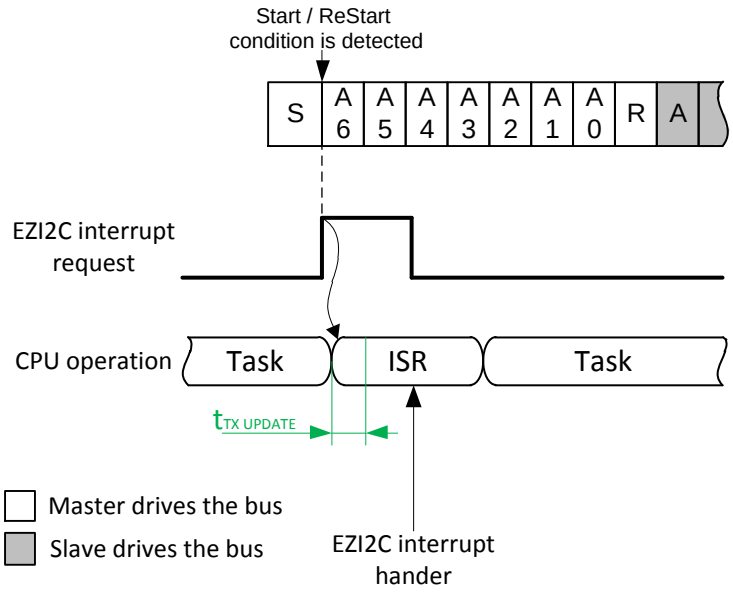
通过禁用时钟延展配置 EZ I^2C 从设备，使之运行于中断服务子程序。这样，EZ I^2C 从设备可以运行而不需要时钟延展。虽然已经优化过，但从设备中断还需要定期服务。从设备中断服务可延迟的最长时间被定义为 EZ I^2C 从设备的最大中断延迟。如果设计不满足所需要的最大 EZ I^2C 从设备中断延迟，则从设备的性能会出现问题。如果该设计没有满足所需要的最大 EZ I^2C 从设备中断延迟，建议使能时钟延展。当选择时钟延展禁用选项时，应注意下列各项：

- 最大的从设备中断延迟
- 通过重新启动与数据操作链路
- 从设备的繁忙管理

最大的从设备中断延迟

在进行数据操作之前，所有主设备都先生成启动条件。从设备硬件将检测该条件并生成一个中断请求，从而启动从设备操作（图 27）。与启动条件相比，重新启动条件的生成对从设备产生相同的影响，但必须设置先前数据操作的完成标志。另外，重新启动条件具有更高的优先级。

图 27. EZ I²C 从设备开始运行



从启动从设备的中断处理器到 TX FIFO 更新第一个字节间的时间被定义为 $t_{TX\ UPDATE}$[5] [6]（图 27）。TX FIFO 更新包括清除 TX FIFO，并从从设备缓冲区内将字节写入到 TX FIFO。在主设备开始读取第一个数据字节前，必须完成 TX FIFO 更新。否则，会发生多个问题，包括：读取旧的 TX FIFO 内容，清除 TX FIFO 时发生时钟延展[7]，或由于在字节传输中间清除了 TX FIFO 而只能读取部分字节。

进行主设备的读取数据操作期间，应用于TX FIFO更新的限制会使从设备的最大中断延迟被定义为最大延迟。可以在从设备硬件检测启动条件到开始执行从设备中断处理器之间添加该延迟（图 28）。

因此，最大中断延迟必须小于 主设备地址字节传输时间（$t_{ADDRESS}$）与从设备ACK位传输时间（t_{ACK}）的总和。但考虑到TX FIFO更新的限制时，最大中断延迟必须小于：

$$t_{MAX\ LATENCY} = (t_{ADDRESS} + t_{ACK}) - t_{TX\ UPDATE} = (8bits / f_{SCL} + 1bit / f_{SCL}) - t_{TX\ UPDATE} = 9 / f_{SCL} - t_{TX\ UPDATE}$$

5 该时间取决于所设计的各种设置如 CPU 时钟、编译器、优化等。

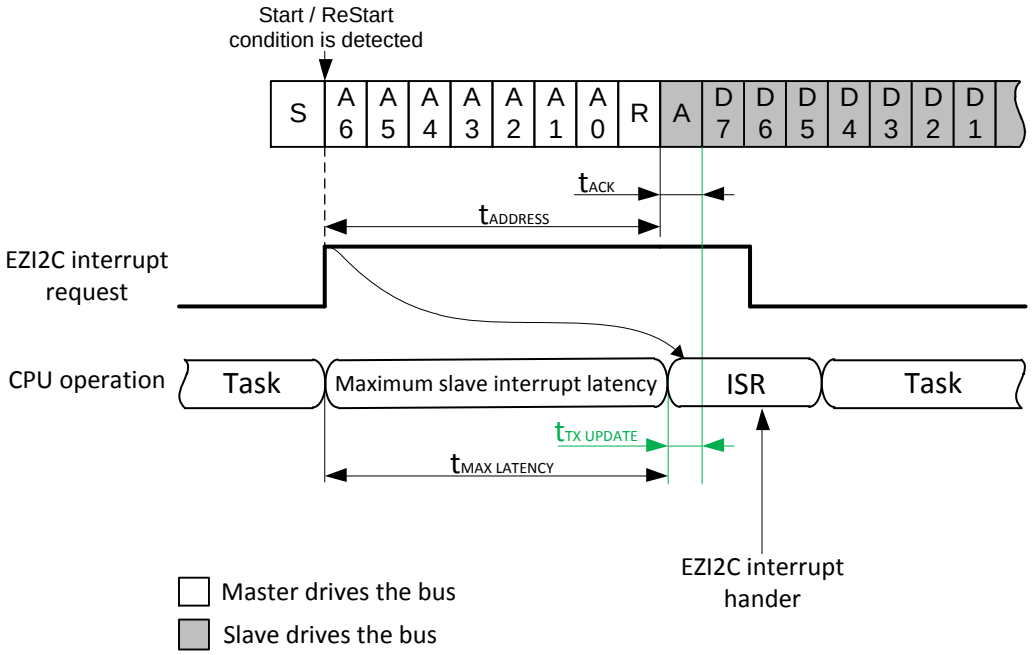
6 它不包含 Cortext-M0 处理器的中断延迟。

7 当 TX FIFO 为空时，从设备硬件将延展时钟。

例如，I^2C 数据速率为 100 kpbs 时，最大中断延迟必须小于：

$$t_{MAX\ LATENCY} = 9 / f_{SCL} – t_{TX\ UPDATE} = 90\ uS – t_{TX\ UPDATE}$$

图 28. EZ I^2C 从设备的最大中断延迟



设计建议：

1. 使用最高的 SYSCLK 频率，以便使 CPU 降低 EZ I^2C 从设备中断的执行时间。
2. 使用编译器的优化选项可降低在 EZ I^2C 从设备中断内执行的指令数量。
3. 将 EZ I^2C 中断设置为设计中优先级最高的中断。如果其他中断具有相同的优先级，请确保它们执行的时间小于 EZ I^2C 最大的中断延迟。
4. 计算设计中每个关键部分持续的时间，然后将得到的值与 EZ I^2C 的最大中断延迟进行比较，以确保该设计满足标准。

通过重新启动与数据操作链路

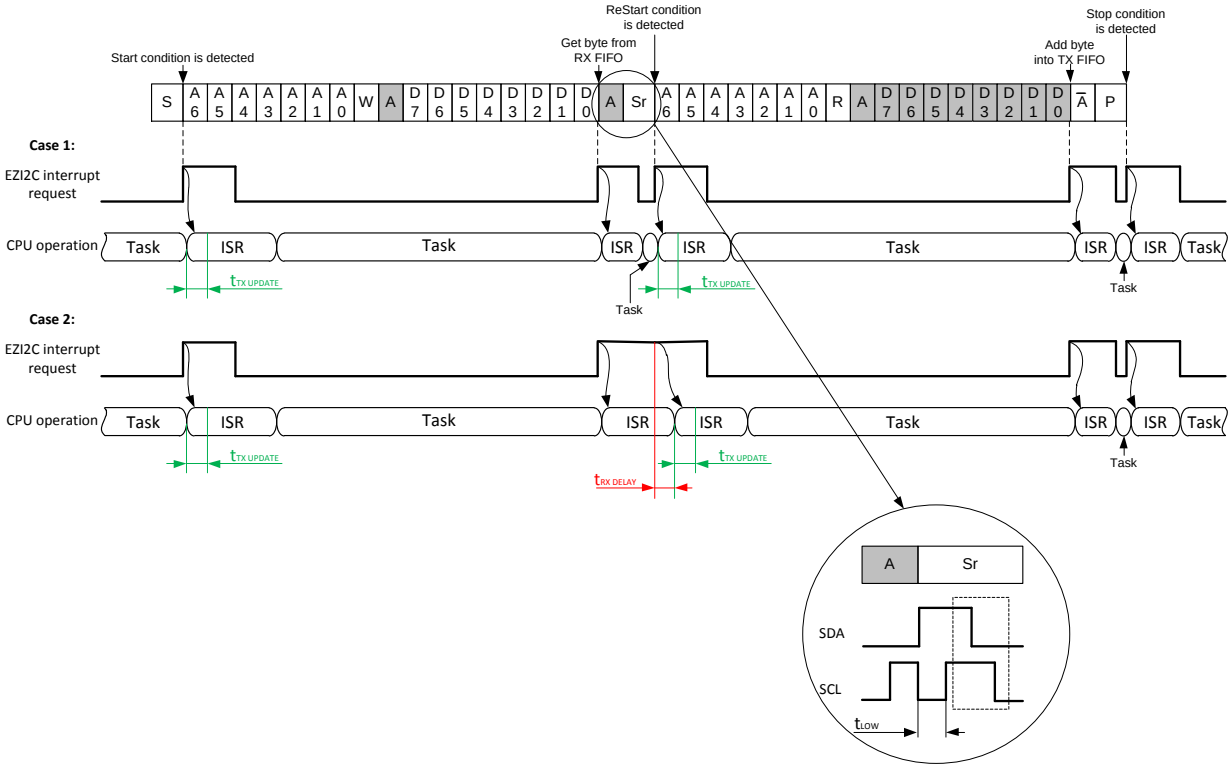
主设备写基本地址和读数据操作通过重新启动链路在一起（图 29）。主设备编写的基本地址（或数据字节）被存储在 RX FIFO 内，并由从设备的中断处理器服务（图 29，情况 1）。与重新启动条件相比，RX FIFO 的优先级更高。这是因为基本地址会通过主设备的写数据操作更新，以用于 TX FIFO 更新。服务 RX FIFO 所需要的时间会对重新启动条件的服务产生影响。如果是（图 29，

情况 2）所示的情况，则生成重新启动条件后，最大中断延迟将以 RX FIFO 服务递增的数值降低：

$$t_{MAX\ LATENCY} = 9 / f_{SCL} – (t_{RX\ DELAY} + t_{TX\ UPDATE})$$

由于 I^2C 规范提供的是最小值，因此必须检查主设备的重新启动时序。在生成重新启动之前，某些主设备延长 t_{LOW} 时间，以准备进行下一个数据操作，但这是器件特定的特性。如果可以控制该时间，则通过在重新启动前一直递增 t_{LOW}，直到 $t_{RX\ DELAY}$ 为 0 为止可以消除 RX FIFO 服务对最大中断延迟产生的影响。

图 29. 主设备设置基本地址并读取数据



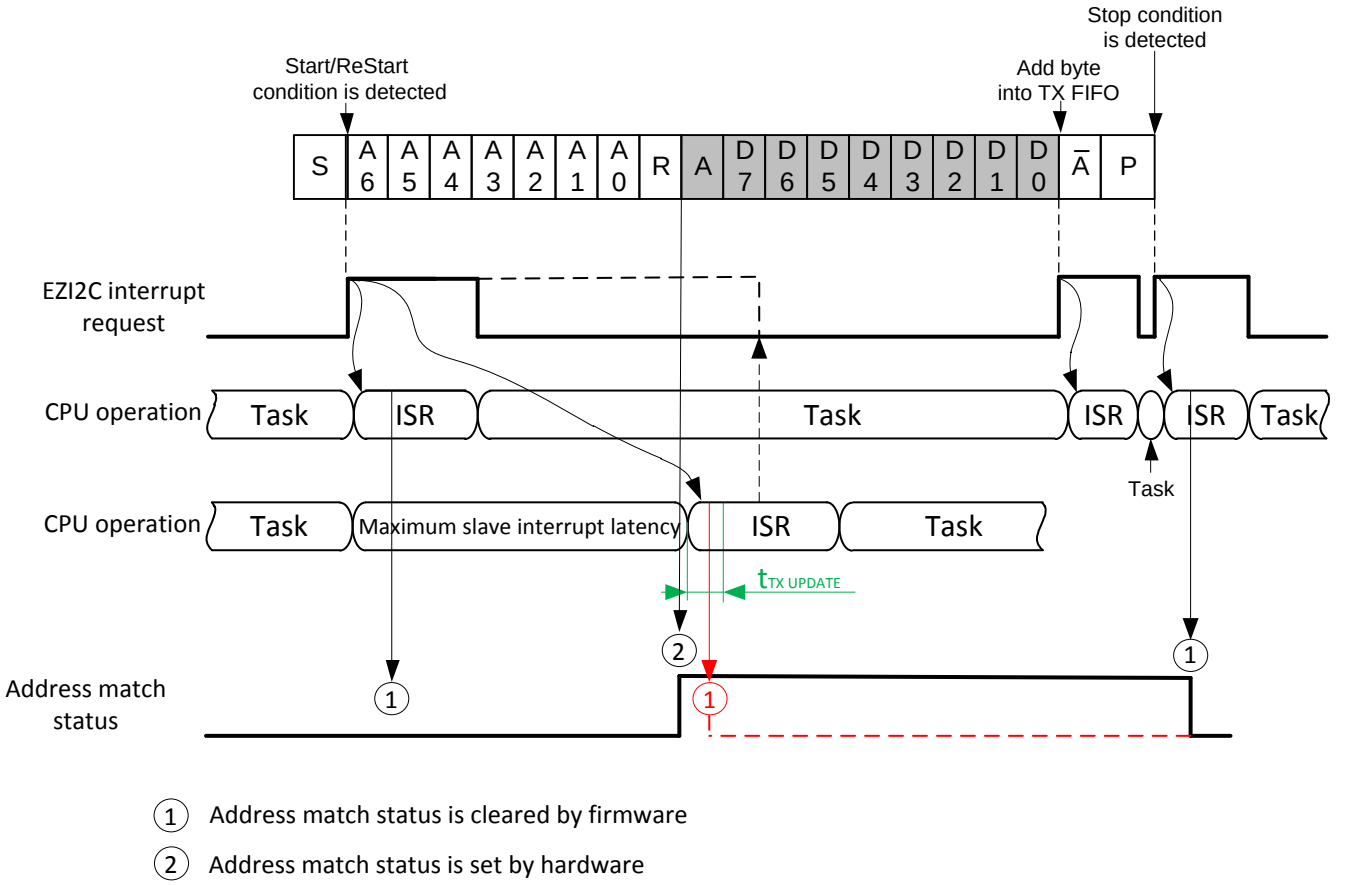
从设备的繁忙管理

SCB_EzI2CGetActivity()和 SCB_Sleep()（仅针对 PSoC 4000 器件）API 使用地址匹配状态来跟踪从设备的繁忙状态。对于 PSoC 4100 /PSoC 4200 器件来说，硬件在地址字节中第 8 个 SCL 的上升沿上触发该事件；而对于 PSoC 4000 器件，硬件在地址字节中第 8 个 SCL 的下降沿上触发该事件（图 30，黑色的第二个周期）。

启动/重新启动或停止条件服务时，固件必须清除地址匹配，使其满足作为从设备的繁忙状态使用（图 30，黑色的第一个周期）。如果启动/重新启动中断服务被延迟，以达到最大中断延迟，那么表示地址匹配状态被清除得过早（图 30，红色的第一个周期）。

这样会使从设备的繁忙记录出错。为了正确记录从设备的繁忙状态，必须将 PSoC 4100 /PSoC 4200 器件的最大中断延迟降低为 1.5 为/ f_{SCL}，并将 PSoC 4000 器件的最大中断延迟降低为 1 位/ f_{SCL}。SCB 硬件总线的繁忙状态可以作为管理总线活动的备用方案。更多有关信息，请参考*技术参考手册（TRM）*中 *SCB_I2C_STATUS* 寄存器中的 *SCB_BUS_BUSY* 位说明。

图 30. 从设备的繁忙管理



外部电气连接

有关 I²C 的信息，请参考外部电路连接中介绍的内容。

更好的辅助地址选择

硬件地址匹配逻辑通过使用地址位掩码来支持两个地址。地址掩码定义了进行某个地址匹配时该地址中被视为无效的位。一个无效位会导致两个匹配地址；两位会匹配四个地址，并以此类推。由于这个原因，优先选择的辅助地址与主要地址有一个位不一样。这种情况下的地址掩码会使某一位无效。如果两个地址有多于一个位不同，这时额外的地址将会通过硬件匹配，并将依赖固件地址匹配以产生一个 NACK。

例如：

- 主要地址 = 0x24，辅助地址 = 0x34，只有一位差异。只有两个地址被硬件匹配。
- 主要地址 = 0x24，辅助地址 = 0x30，有两位差异。四个地址被硬件匹配：**0x24**、0x34、0x20 和 **0x30**。固件要求确认主要和辅助地址 0x24 和 0x30，并否认所有其它的 0x20 和 0x34。

低功耗模式

EZ I²C 模式中的组件可以作为低功耗的睡眠和深度睡眠模式的唤醒源。

对于外设来说，睡眠模式相当于活动模式。不需要对组件进行任何配置修改，也不需要在进入/退出此模式前调用代码。对从设备进行的任何通信都会引起中断并导致唤醒。

深度睡眠模式要求对从设备进行合适的配置，将其作为唤醒源使用。在从设备配置对话框中，必须勾选“Enable wakeup from Sleep mode”（使能从睡眠模式中唤醒）。进入/退出深度睡眠模式之前/之后，必须调用 SCB_Sleep()和 SCB_Wakeup()。

唤醒事件是从设备的地址匹配。外部定时逻辑执行地址匹配。当发生此操作时，将触发唤醒中断。然而，地址匹配完成后，从设备的性能取决于所选择的时钟延展选项。

使能时钟延展。从设备一直延展 SCL 线，直到控制权被传递给从设备的中断子程序以确认该地址为止。

进入深度睡眠模式之前，必须完成从设备正在进行的数据操作，因此建议使用下列代码：

```
CyGlobalIntDisable; /* Disable all interrupts to lock the I2C bus state */

/* Check if a transaction is in process */
status = (SCB_EzI2CGetActivity() & SCB_EZI2C_STATUS_BUSY);

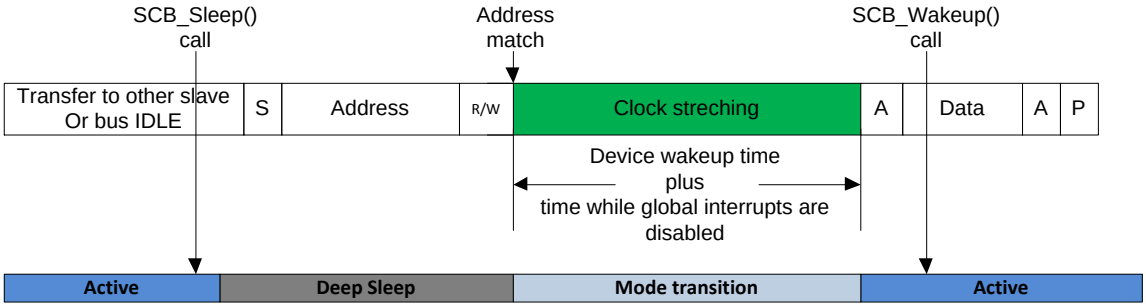
if(0u == status) /* Slave is not addressed */
{
    SCB_Sleep(); /* Prepare for Deep Sleep: enables wakeup interrupt */

    CySysPmDeepSleep();
```

```
        CyGlobalIntEnable; /* Enable all interrupts to unlock I2C bus state */

        SCB_Wakeup(); /* Restore for Active mode: disables wakeup interrupt */
    }
    else
    {
        /* Transaction in progress: do not go to Deep Sleep */
        CyGlobalIntEnable;
    }
```

图 31. 主设备传输在从设备地址匹配时唤醒器件（使能时钟延展）



注意：主要地址和辅助地址对匹配地址的范围产生影响。辅助地址首选是同主要地址只有一位的差别。如果它们相差大于一位，那么仍会有一些不适用于该组件的操作将该组件从深度睡眠状态唤醒。在这种情况下，可取消确认地址。

禁用时钟延展：在器件唤醒时间内，从设备取消确认匹配地址并跟踪匹配地址。

进入深度睡眠模式前，必须完成从设备正在进行的数据操作。等待循环会在 SCB_Sleep()函数中实现。此函数被阻止并一直保持等待状态，直到从设备将它配置为唤醒源为止。在重新配置后，将启动地址匹配事件的示例，并且器件会有时间进入深度睡眠模式。从设备配置必须由 SCB_Wakeup()恢复，这样才能在活动模式下正常运行。总之，从设备和主设备之间的协议是唤醒后不对从设备进行访问，直到执行 SCB_Wakeup()为止。建议使用下列代码：

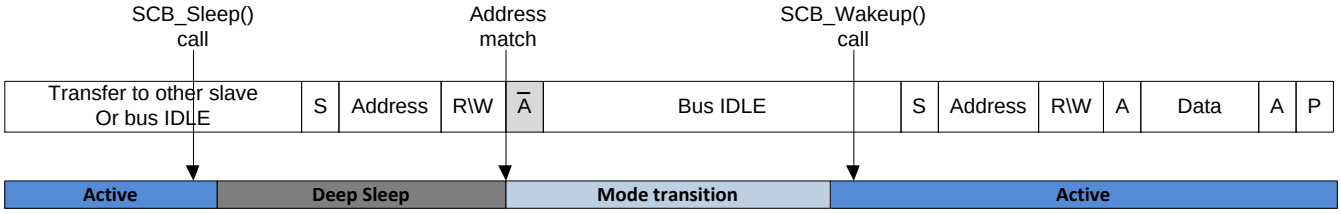
```
SCB_Sleep(); /* Wait for the slave to be free and configures it to be wakeup
source */

CySysPmDeepSleep();

SCB_Wakeup(); /* Configure the slave to active mode operation */
```

注意：在调用 SCB_Sleep()函数前，必须使能用于从设备操作的各中断和全局中断。

图 32. 主设备传输在从设备地址匹配时唤醒器件（禁用时钟延展）



常用 SCB 组件信息

中断 API

这些函数是大多数 SCB 模式的常用函数。

默认情况下，PSoC Creator 将实例名称“SCB_1”分配给第一个添加到工作区（TopDesign）的 SCB 组件。您可以将其重新命名为任何一个符合标识符语法规则的值。实例名称会作为每个与该组件相关的全局函数名称、变量和常量符号的前缀。为便于阅读，下表使用的实例名称为“SCB”。

函数	说明
SCB_EnableInt()	使能NVIC的中断（当使用内部中断时）。
SCB_DisableInt()	禁用NVIC的中断（当使用内部中断时）。
SCB_GetInterruptCause()	返回显示当前触发中断源的位掩码。
SCB_SetCustomInterruptHandler()	注册一个由内部中断处理程序调用的函数。
SCB_SetTxInterruptMode()	配置触发中断事件的TX中断请求寄存器的位。
SCB_GetTxInterruptMode()	返回 TX 中断掩码
SCB_GetTxInterruptSourceMasked()	返回由中断掩码屏蔽的 TX 中断请求寄存器
SCB_GetTxInterruptSource()	返回待处理 TX 中断源的位掩码
SCB_ClearTxInterruptSource()	清除待处理 TX 中断源的位掩码
SCB_SetTxInterrupt()	从 TX 中断源的位掩码生成中断事件
SCB_SetRxInterruptMode()	配置哪些 RX 中断请求寄存器的比特位才可触发中断事件
SCB_GetRxInterruptMode()	返回 RX 中断掩码
SCB_GetRxInterruptSourceMasked()	返回由中断掩码屏蔽的 RX 中断请求寄存器
SCB_GetRxInterruptSource()	返回待处理 RX 中断源的位掩码
SCB_ClearRxInterruptSource()	清除待处理 RX 中断源的位掩码
SCB_SetRxInterrupt()	从 RX 中断源的位掩码生成中断事件
SCB_SetMasterInterruptMode()	配置多少位主设备中断请求寄存器才可触发中断事件。
SCB_GetMasterInterruptMode()	返回主设备中断掩码
SCB_GetMasterInterruptSourceMasked()	返回由中断掩码屏蔽的主设备中断请求寄存器
SCB_GetMasterInterruptSource()	返回待处理主设备中断源的位掩码
SCB_ClearMasterInterruptSource()	清除待处理主设备中断源的位掩码

函数	说明
SCB_SetMasterInterrupt()	从主设备中断源的位掩码生成中断事件
SCB_SetSlaveInterruptMode()	配置多少位的从设备中断请求寄存器才可触发中断事件
SCB_GetSlaveInterruptMode()	返回从设备中断掩码
SCB_GetSlaveInterruptSourceMasked()	返回被中断掩码屏蔽的从设备中断请求寄存器
SCB_GetSlaveInterruptSource()	返回待处理从设备中断源的位掩码
SCB_ClearSlaveInterruptSource()	清除待处理从设备中断源的位掩码
SCB_SetSlaveInterrupt()	用组件中断源的位掩码生成中断事件

中断函数应用

函数	I2C	SPI	UART	EZI2C
SCB_EnableInt()	+	+	+	+
SCB_DisableInt()	+	+	+	+
SCB_GetInterruptCause()	+	+	+	+
SCB_SetCustomInterruptHandler()	+	+	+	+
SCB_SetTxInterruptMode()	+	+	+	+
SCB_GetTxInterruptMode()	+	+	+	+
SCB_GetTxInterruptSourceMasked()	+	+	+	+
SCB_GetTxInterruptSource()	+	+	+	+
SCB_ClearTxInterruptSource()	+	+	+	+
SCB_SetTxInterrupt()	+	+	+	+
SCB_SetRxInterruptMode()	+	+	+	+
SCB_GetRxInterruptMode()	+	+	+	+
SCB_GetRxInterruptSourceMasked()	+	+	+	+
SCB_GetRxInterruptSource()	+	+	+	+
SCB_ClearRxInterruptSource()	+	+	+	+
SCB_SetRxInterrupt()	+	+	+	+
SCB_SetMasterInterruptMode()	+	+	_	_
SCB_GetMasterInterruptMode()	+	+	_	_
SCB_GetMasterInterruptSourceMasked()	+	+	_	_
SCB_GetMasterInterruptSource()	+	+	_	_

函数	I2C	SPI	UART	EZI2C
SCB_ClearMasterInterruptSource()	+	+	_	_
SCB_SetMasterInterrupt()	+	+	_	_
SCB_SetSlaveInterruptMode()	+	+	_	+
SCB_GetSlaveInterruptMode()	+	+	_	+
SCB_GetSlaveInterruptSourceMasked()	+	+	_	+
SCB_GetSlaveInterruptSource()	+	+	_	+
SCB_ClearSlaveInterruptSource()	+	+	_	+
SCB_SetSlaveInterrupt()	+	+	_	+

void SCB_EnableInt(void)

说明： 当使用内部中断时，将使能NVIC的中断。当使用外部中断时，必须用中断组件的API来启用中断。

参数： 无

返回值： 无

其他影响： 无

void SCB_DisableInt(void)

说明： 当使用内部中断时，将禁用NVIC的中断。当使用外部中断时，必须用中断组件的API来禁用中断。

参数： 无

返回值： 无

其他影响： 无

uint32 SCB_GetInterruptCause(void)

说明： 返回显示当前触发中断源的位掩码。根据多个中断寄存器的条件生成中断对该操作模式有用。

参数： 无

返回值： uint32: 已触发包含下列条件OR的掩码：

中断原因常量	说明
SCB_INTR_CAUSE_MASTER	来自主设备的中断
SCB_INTR_CAUSE_SLAVE	来自从设备的中断
SCB_INTR_CAUSE_TX	来自TX的中断
SCB_INTR_CAUSE_RX	来自RX的中断

其他影响： 无

void SCB_SetCustomInterruptHandler(void (*func) (void))

说明： 注册一个由内部中断处理程序调用的函数。首先调用已注册函数，然后内部中断处理程序在中断返回之前执行软件缓存区管理等操作。用户不可中断软件缓存区操作。仅支持一个是自定义处理程序，即：最近期调用提供的函数。初始化期间，未注册任何自定义处理程序。

参数： Func：指向要寄存的函数的指针。NULL （空）值表示删除当前自定义中断处理程序。

返回值： 无

其他影响： 无

void SCB_SetTxInterruptMode(uint32 interruptMask)

说明： 配置触发中断事件的TX中断请求寄存器的位。

参数： uint32 interruptMask：将被启用的 TX 中断源的位掩码。

TX中断源	说明
SCB_INTR_TX_TRIGGER	发送器FIFO中的输入条目少于触发电平指定的值
SCB_INTR_TX_NOT_FULL	发送器FIFO未满。
SCB_INTR_TX_EMPTY	发送器FIFO为空。
SCB_INTR_TX_OVERFLOW	尝试写入到一个已满的发送器FIFO。
SCB_INTR_TX_UNDERFLOW	尝试从一个空的发送器FIFO中读取。
SCB_INTR_TX_UART_NACK	UART接收到SmartCard模式的NACK。
SCB_INTR_TX_UART_DONE	UART传输已完成并且TX FIFO为空。
SCB_INTR_TX_UART_ARB_LOST	UART的TX线路上的值与RX线路上的值不匹配。

返回值： 无

其他影响： 无

uint32 SCB_GetTxInterruptMode(void)

说明： 返回TX中断掩码。

参数： 无

返回值： uint32：已使能的TX中断源的掩码（返回值请参阅 SCB_SetTxInterruptMode()函数）。

其他影响 无

uint32 SCB_GetTxInterruptSourceMasked(void)

说明： 返回由中断掩码屏蔽的TX中断请求寄存器。

参数： 无

返回值： uint32：仅已使能的TX中断源的状态（返回值请参阅 SCB_SetTxInterruptMode()函数）。

其他影响 无

uint32 SCB_GetTxInterruptSource(void)

说明： 返回待处理TX中断源的位掩码。

参数： 无

返回值： uint32：TX中断源的状态（返回值请参阅 SCB_SetTxInterruptMode()函数）。

其他影响 无

void SCB_ClearTxInterruptSource(uint32 interruptMask)

说明： 清除待处理TX中断源的位掩码。

参数： uint32 interruptMask：将被清除的待处理TX中断源的位掩码（返回值请参阅 SCB_SetTxInterruptMode()）。

返回值： 无

其他影响 对于受影响的每个中断源，其他影响如下表所示。

TX中断源	说明
SCB_INTR_TX_TRIGGER	触发事件有效时，不会清除中断源。
SCB_INTR_TX_NOT_FULL	发送器FIFO有空输入时，不会清除中断源。
SCB_INTR_TX_EMPTY	发送器FIFO为空时，将不清除中断源。
SCB_INTR_TX_UNDERFLOW	发送器FIFO为空时，将不清除中断源，并且具有时钟延展功能的I2C模式被选定。在清除数据前，请将这些数据存储在发送器FIFO内。 该性能只针对PSoC 4100/PSoC 4200器件。

void SCB_SetTxInterrupt(uint32 interruptMask)

说明： 从TX中断源的位掩码生成中断事件。

参数： uint32 interruptMask：将生成中断事件的TX中断源的位掩码（返回值请参阅 SCB_SetTxInterruptMode()）。

返回值： 无

其他影响： 无

void SCB_SetRxInterruptMode(uint32 interruptMask)

说明： 配置哪些RX中断请求寄存器的比特位才可触发中断事件。

参数： uint32 interruptMask：将被启用的 RX 中断源的位掩码。

RX中断源	说明
SCB_INTR_RX_TRIGGER	接收器FIFO中的输入多于触发电平指定的值。
SCB_INTR_RX_NOT_EMPTY	接收器FIFO不为空。
SCB_INTR_RX_FULL	接收器FIFO已满。
SCB_INTR_RX_OVERFLOW	尝试写入到一个已满的接收器FIFO。
SCB_INTR_RX_UNDERFLOW	尝试从一个空的接收器FIFO中读取。
SCB_INTR_RX_FRAME_ERROR	检测到UART成帧错误。
SCB_INTR_RX_PARITY_ERROR	检测到UART奇偶校验错误。

返回值： 无

其他影响： 无

uint32 SCB_GetRxInterruptMode(void)

说明： 返回RX中断掩码。

参数： 无

返回值： uint32:已使能的RX中断源掩码（返回值请参阅SCB_SetTxInterruptMode()函数）。

其他影响 无

uint32 SCB_GetRxInterruptSourceMasked(void)

说明： 返回由中断掩码屏蔽的RX中断请求寄存器。

参数： 无

返回值： uint32：仅已使能的RX中断源的状态（返回值请参阅SCB_SetRxInterruptMode()函数）。

其他影响 无

uint32 SCB_GetRxInterruptSource(void)

说明： 返回待处理RX中断源的位掩码。

参数： 无

返回值： uint32：RX中断源的状态（返回值请参阅SCB_SetRxInterruptMode()函数）。

其他影响 无

void SCB_ClearRxInterruptSource(uint32 interruptMask)

说明： 清除待处理RX中断源的位掩码。

参数： uint32 interruptMask：将被清除的待处理RX中断源的位掩码（返回值请参阅SCB_SetRxInterruptMode()）

返回值： 无

其他影响 对于受影响的每个中断源，其他影响如下表所示。

RX中断源	说明
SCB_INTR_RX_TRIGGER	触发事件有效时，将不清除中断源。
SCB_INTR_RX_NOT_EMPTY	接收器FIFO为空时，将不清除中断源。
SCB_INTR_RX_FULL	接收器FIFO为满时，将不清除中断源。

void SCB_SetRxInterrupt(uint32 interruptMask)

说明： 从RX中断源的位掩码生成中断事件。

参数： uint32 interruptMask：将生成中断事件的RX中断源的位掩码（返回值请参阅SCB_SetRxInterruptMode()）。

返回值： 无

其他影响： 无

void SCB_SetMasterInterruptMode(uint32 interruptMask)

说明： 配置多少位主设备中断请求寄存器才可触发中断事件。

参数： uint32 interruptMask：将被启用的 RX 中断源的位掩码。

主设备的中断源	说明
SCB_INTR_MASTER_SPI_DONE	SPI主设备传输已完成并且TX FIFO为空。
SCB_INTR_MASTER_I2C_ARB_LOST	I2C主设备仲裁失败。
SCB_INTR_MASTER_I2C_NACK	I2C主设备收到否定确认（NAK）。
SCB_INTR_MASTER_I2C_ACK	I2C主设备收到确认。
SCB_INTR_MASTER_I2C_STOP	I2C主设备生成“停止”。
SCB_INTR_MASTER_I2C_BUS_ERROR	I2C主设备的总线错误（检测意外启动和停止条件）。

返回值： 无

其他影响： 无

uint32 SCB_GetMasterInterruptMode(void)

说明： 返回主设备中断掩码

参数： 无

返回值： uint32：已使能的主设备中断源的掩码（返回值请参阅SCB_SetMasterInterruptMode()函数）。

其他影响 无

uint32 SCB_GetMasterInterruptSourceMasked(void)

说明： 返回由中断掩码屏蔽的主设备中断请求寄存器。

参数： 无

返回值： uint32：仅已使能的主设备中断源的状态（返回值请参阅SCB_SetMasterInterruptMode()函数）。

其他影响 无

uint32 SCB_GetMasterInterruptSource(void)

说明： 返回待处理主设备中断源的位掩码。

参数： 无

返回值： uint32：主设备中断源的状态（返回值请参阅SCB_SetMasterInterruptMode()函数）。

其他影响 无

void SCB_ClearMasterInterruptSource(uint32 interruptMask)

说明： 清除待处理主设备中断源的位掩码。

参数： uint32 interruptMask：将被清除的待处理主设备中断源的位掩码（返回值请参阅SCB_SetMasterInterruptMode()）。

返回值： 无

其他影响： 无

void SCB_SetMasterInterrupt(uint32 interruptMask)

说明： 从主设备中断源的位掩码生成中断事件。

参数： uint32 interruptMask：将生成中断事件的主设备中断源的位掩码（返回值请参阅SCB_SetMasterInterruptMode()）。

返回值： 无

其他影响： 无

void SCB_SetSlaveInterruptMode(uint32 interruptMask)

说明： 配置多少位的从设备中断请求寄存器才可触发中断事件。

参数： uint32 interruptMask：将被启用的从设备中断源的位掩码。

从设备中断源	说明
INTR_SLAVE_I2C_ARB_LOST	I2C从设备仲裁失败: SDA线路驱动值不同于SDA线路观察值。
INTR_SLAVE_I2C_NACK	I2C从设备收到了否定确认（NAK）。
INTR_SLAVE_I2C_ACK	I2C从设备收到了确认（ACK）。
INTR_SLAVE_I2C_WRITE_STOP	本从设备写入传输的“停止”或“重复启动”事件（进行了地址匹配）。
INTR_SLAVE_I2C_STOP	用于本从设备（读取或写入）传输的“停止”或“重复启动” 事件（进行了地址匹配）。
INTR_SLAVE_I2C_START	I2C从设备收到了“启动”条件 。
INTR_SLAVE_I2C_ADDR_MATCH	I2C从设备收到了匹配地址。
INTR_SLAVE_I2C_GENERAL	I2C从设备收到了通用调用地址。
INTR_SLAVE_I2C_BUS_ERROR	I2C从设备的总线错误（检测意外启动和停止条件）。
INTR_SLAVE_SPI_BUS_ERROR	在 SPI 传输中的预期时间取消选中 SPI 从设备。

返回值： 无

其他影响： 无

uint32 SCB_GetSlaveInterruptMode(void)

说明： 返回从设备中断掩码。

参数： 无

返回值： uint32：已使能的从设备中断源掩码（返回值请参阅SCB_SetSlaveInterruptMode()函数）。

其他影响 无

uint32 SCB_GetSlaveInterruptSourceMasked(void)

说明： 由中断掩码屏蔽的从设备中断请求寄存器。

参数： 无

返回值： uint32：仅已使能的从设备中断源的状态（返回值请参见SCB_SetSlaveInterruptMode() 函数）。

其他影响 无

uint32 SCB_GetSlaveInterruptSource(void)

说明： 返回待处理从设备中断源的位掩码

参数： 无

返回值： uint32：从设备中断源的状态（返回值请参见 SCB_SetSlaveInterruptMode() 函数）

其他影响： 无

void SCB_ClearSlaveInterruptSource(uint32 interruptMask)

说明： 清除待处理从设备中断源的位掩码。

参数： uint32 interruptMask：将被清除的待处理从设备中断源的位掩码（返回值请参见 SCB_SetSlaveInterruptMode()函数）。

返回值： 无

其他影响： 无

void SCB_SetSlaveInterrupt(uint32 interruptMask)

说明： 用组件中断源的位掩码生成中断事件。

参数： uint32 interruptMask：将生成中断事件的从设备中断源的位掩码（返回值请参见 SCB_SetSlaveInterruptMode()函数）。

返回值： 无

其他影响： 无

时钟选择

SCB 由一个专用时钟连接计时。根据操作模式，组件可按自定义配置计算该时钟的频率，或由外部提供。

由于“未配置”模式定制器不知道终端运行模式，在这种情况下，时钟必须外部提供。

MISRA 合规性

本节介绍了 MISRA-C:2004 合规性和本组件的偏差情况。定义了下面两种类型的偏差：

- 项目偏差 — 适用于所有 PSoC Creator 器件的偏差
- 特定偏差 — 仅适用于该组件的偏差

本节介绍了有关组件特定偏差的信息。*系统参考指南*的“MISRA 合规性”章节中介绍项目偏差以及有关 MISRA 合规性验证环境的信息。

SCB 组件具有以下特定偏差：

MISRA-C: 2004规则	规则类[8]	规则说明	偏差说明
1.1	R	此规则规定，代码必须符合C C ISO/IEC 9899:1990标准。	控制结构嵌套（说明）超过15—程序未严格达到ISO:C90。 实际上，大多数编译器支持更自由的嵌套限制，所以只有当要求严格操作时才涉及此限制。通过比较，ISO:C99 规定模块的限制为 127 " 嵌套级。 支持的编译器（GCC 4.1.1、RVDS 和 MDK）支持更多的控制结构嵌套。
17.4	R	数组索引是唯一允许的指针运算形式。	组件使用阵列索引操作访问缓冲区。访问前，先检查缓冲区大小。这是一项安全操作，以防用户提供的缓冲区大小有错误。
19.7	A	函数应该优先于类似于函数的宏。	偏差由于函数宏用于允许更高效的代码。

此组件有以下嵌入式组件：引脚和中断。MISRA 合规性与特定偏差的相关信息，请参见相应组件数据手册。

样例固件源代码

PSoC Creator 在“Find Example Project”（查找示例项目）对话框中提供了多个包括原理图和代码示例的示例项目。要查看特定组件实例，请打开“Component Catalog”中的对话框或原理图中的组件示例。要查看通用示例，请打开“Start Page”或 **File** 菜单中的对话框。根据要求，可以通过使用对话框中的 **Filter Options** 选项来限定可选的项目列表。

更多有关信息，请参考《PSoC Creator 帮助》部分中主题为“查找示例项目”中的内容。

[8] （R）必须/（A）建议

中断服务子程序

SCB 支持多种事件上的中断，取决于操作模式。所有的中断事件一起执行“或”运算，然后发送至中断控制器，因此在任何给定的时间，SCB 只能生成一个中断请求。任何已启用的中断源被触发时，此信号将变为高电平。

如“输入/输出连接”章节所述，当内部操作不需要此信号时，有些模式将此信号显示为终端信号。如果内部操作需要此信号，终端信号就不存在。

通过使用各种中断 API，软件可在单个中断服务子程序中服务多个中断事件。

PSoC Creator 生成必要的中断服务子程序，以处理内部操作。但是，可以使用 *SCB_SetCustomInterruptHandler()*函数寄存一个自定义函数。在内部中断处理程序执行诸如软件缓冲管理功能的任何操作时，将首先调用该用户函数。只支持一个自定义处理程序。

注：用户管理的“中断源”不会被自动清除。这是用户的责任。将‘1’写入到相应的位位置，可清除中断源。清除中断源的首选方法是使用 API （例如：SCB_ClearRxInterruptSource()）。

```
void CustomInterruptHandler(void);
void main()
{
    /* Register custom function */
    SCB_SetCustomInterruptHandler(&CustomInterruptHandler);

    /* Initialize SCB component in UART mode.
    * The SCB_INTR_RX_PARITY_ERROR is already enabled in GUI:
    * UART Advanced Tab.
    */
    SCB_Start();
    CyGlobalIntEnable; /* Enable global interrupts. */
    for(;;)
    {
        /* Place your application code here. */
    }
}

/* User interrupt handler to insert into SCB interrupt handler.
* Note: SCB interrupt set to Internal in GUI.
*/
void CustomInterruptHandler(void)
{
    if(0u != (SCB_GetRxInterruptSourceMasked() & SCB_INTR_RX_PARITY_ERROR))
    {
        /* Interrupt sources does not clear automatically if it is managed by
        * user. The interrupt sources clearing becomes user responsibility.
        */
        SCB_ClearRxInterruptSource(SCB_INTR_RX_PARITY_ERROR);

        /*
        * Add user interrupt code to manage SCB_INTR_RX_PARITY_ERROR.
        */
    }
}
```

TX FIFO 中断源

下述中断源具有特定的行为：TX FIFO 为空，TX FIFO 未满并且 TX FIFO 触发。这些中断源触发 TX FIFO 的当前状态，并保持到清除操作为止。当 SCB 组件被禁用时，TX FIFO 变为空并且 TX 中断源触发其状态。如果 TX FIFO 为空，清除这些中断源没有任何意义，因为它们将自行恢复。恢复操作需要一个时钟周期，因此在此期间清除中断源。当填充 TX FIFO 时，最好监测其中的条目数量，而不是在每个字节输入到 TX FIFO 后尝试清除未满的 TX FIFO 或触发中断源。恢复时间导致误未满 TX FIFO 被误清除或触发中断源。若要启动 TX FIFO 中断源处理，建议采用如下流程：在 TX FIFO 中填入数据，清除已触发的“旧”中断源并启用它。

为了清除中断源，将‘1’写入到相应的位位置。

注意：TX FIFO 触发中断源行为取决于触发电平值。

RX FIFO 中断源

下述中断源具有特定的行为：RX FIFO 为空，RX FIFO 已满并且 RX FIFO 触发。这些中断源触发 RX FIFO 的电流状态并保持到清除操作为止。当 SCB 组件被禁用时，RX FIFO 变为空，且可以清除触发中断源。当 RX FIFO 为空，清除这些中断源没有任何意义，因为他们将得到恢复。恢复操作需要一个时钟周期，因此在此期间清除中断源。当从 RX FIFO 读取数据时，最好监测其中的条目数量，而不是在每个读取字节后尝试清除未满的 RX FIFO 或触发中断源。恢复时间导致误清除不为空的 RX FIFO 或触发中断源。若要启动 RX FIFO 中断源处理，建议采用如下流程：清除触发的“旧”中断源并启用它。在大多数情况下，不需要清除操作。

为了清除中断源，将‘1’写入到相应的位位置。

注意：RX FIFO 触发中断源行为取决于触发电平值。

放置

SCB 按固定功能模块放置，且所有的放置信息将通过 *cyfitter.h* 文件提供给 API。

寄存器

有关寄存器的详细信息，请参见芯片的*《技术参考手册》（TRM）*。

资源

USB 作为固定功能模块实现。

模式		资源类型		
		SCB固定模块		中断
未配置的SCB		1		1
I^2C	从设备	1		1
	主设备	1		1
	多主设备	1		1
	多主要从设备	1		1
SPI	从设备	硬件缓冲区	1	_
		软件缓冲区	1	1
	主设备	硬件缓冲区	1	_
		软件缓冲区	1	1
UART	标准	硬件缓冲区	1	_
		软件缓冲区	1	1
	标准（多处理器模式）	硬件缓冲区	1	_
		软件缓冲区	1	1
	SmartCard	硬件缓冲区	1	_
		软件缓冲区	1	1
	IrDA	硬件缓冲区	1	_
		软件缓冲区	1	1
EZ I^2C	使能时钟延展	一个地址	1	1
		两个地址	1	1
	禁止时钟延展	一个地址	1	1

API 存储器的使用情况

根据编译器、器件、所使用的 API 数量以及组件的配置情况的不同，组件所用的存储器大小也不一样。下表提供了在某一器件配置中的所有 API 使用的存储器大小。

通过使用“Release”模式中的相应编译器，可以进行测量操作。在该模式下，存储器的大小得到优化。对于特定的设计，分析编译器生成的映射文件后可以确定存储器的使用情况。

配置			PSoC 4 （GCC）			
			PSoC 4000		PSoC 4100/PSoC 4200	
			闪存	RAM	闪存	RAM
未配置的 SCB			7006	154	10036	171
I^2C	从设备		1458	42	1530	42
	主设备		2504	46	2748	46
	多主设备		2504	46	2748	46
	多主从设备		3672	79	3904	79
SPI	从设备	硬件缓冲区 [9]	N/A	N/A	438	9
		软件缓冲区 [10]	N/A	N/A	998	50
	主设备	硬件缓冲区 [9]	N/A	N/A	442	9
		软件缓冲区 [10]	N/A	N/A	1002	50
UART	标准	硬件缓冲区 [9]	N/A	N/A	640	9
		软件缓冲区 [10]	N/A	N/A	1152	50
	标准（多处理器模式）	硬件缓冲区 [9]	N/A	N/A	652	9
		软件缓冲区 [10]	N/A	N/A	1224	70
	SmartCard	硬件缓冲区 [9]	N/A	N/A	644	9
		软件缓冲区 [10]	N/A	N/A	1200	50
	IrDA	硬件缓冲区 [9]	N/A	N/A	648	9
		软件缓冲区 [10]	N/A	N/A	1204	50
EZ I^2C	使能时钟延展	一个从设备地址	1240	26	1284	26
		两个从设备地址	1636	50	1664	50
	禁止时钟延展 [11]	一个从设备地址	1216	23	1044	24

[9] 硬件缓冲区－RX 与 TX 缓冲区大小等于 8 字节，仅使用了硬件 FIFO。中断模式为“无”

[10] RX 和 TX 缓冲区大小为 10 字节，内部 RAM 中 10 字节的缓冲区也被用作硬件 FIFO。在这种情况下，内部中断被自动启用。

直流和交流电气特性

除非另有说明，否则这些规范的适用条件是：-40°C ≤ T_A ≤ 85 °C 且 T_J ≤ 100 °C。除非另有说明，否则这些规范的适用范围为 1.71 V 到 5.5 V。

PSoC 4000

I^2C 直流规范

参数	说明	最小值	典型值	最大值	单位	条件
I_{I2C1}	在100 KHz下，模块电流消耗	–	–	10.5	µA	
I_{I2C2}	在 400 KHz 下，模块电流消耗	–	–	135	µA	
I_{I2C4}	I^2C在深度睡眠模式下使能	–	–	2.5	µA	

I^2C AC 规范

参数	说明	最小值	典型值	最大值	单位	条件
F_{I2C1}	比特率	–	–	400	Kbps	

PSoC 4100/PSoC 4200

I^2C 直流规范

参数	说明	最小值	典型值	最大值	单位	条件
I_{I2C1}	在100 KHz下，模块电流消耗	–	–	10.5	µA	
I_{I2C2}	在 400 KHz 下，模块电流消耗	–	–	135	µA	
I_{I2C3}	在1 Mbps下，模块电流消耗	–	–	310	µA	
I_{I2C4}	I^2C在深度睡眠模式下使能	–	–	1.4	µA	

I^2C AC 规范

参数	说明	最小值	典型值	最大值	单位	条件
F_{I2C1}	比特率	–	–	1	Mbps	

[11] 对于 PSoC 4000 器件，“使能从睡眠模式唤醒”被启用。

UART DC 规范

参数	说明	最小值	典型值	最大值	单位	条件
I_{UART1}	100 Kbits/秒时的模块耗电量	–	–	9	µA	
I_{UART2}	1000 Kbits/秒时的模块耗电量	–	–	312	µA	

UART AC 规范

参数	说明	最小值	典型值	最大值	单位	条件
F_{UART}	比特率	–	–	1	Mbps	

SPI DC 规范

参数	说明	最小值	典型值	最大值	单位	条件
I_{SPI1}	在1 Mbits/秒时的模块电流消耗	–	–	360	µA	
I_{SPI2}	4 Mbits/秒时的模块耗电量	–	–	560	µA	
I_{SPI3}	8 Mbits/秒时的模块耗电量	–	–	600	µA	

SPI AC 规范

参数	说明	最小值	典型值	最大值	单位	条件
F_{SPI}	SPI 操作频率（主设备；6X 过采样））	–	–	8	MHz	

SPI 主设备交流规范

参数	说明	最小值	典型值	最大值	单位	条件
T_{DMO}	Sclock驱动沿后的MOSI有效	–	–	15	ns	
T_{DSI}	Sclock捕获沿前的MISO有效。全时钟、MISO推迟采样	20	–	–	ns	
T_{HMO}	关于从设备捕获沿的先前MOSI数据保持时间	0	–	–	ns	

SPI 从设备交流规范

参数	说明	最小值	典型值	最大值	单位	条件
T_{DMI}	Sclock捕获沿前的MOSI有效	40	–	–	ns	
T_{DSO}	Sclock驱动沿后的MISO有效	–	–s	42 + 3 × FCPU	ns	
T_{DSO_ext}	在外部时钟中的Sclock驱动沿后的MISO有效。时钟模式	–	–	48	ns	
T_{HSO}	先前的MISO数据保持时间	0	–	–	ns	
$T_{SSELSCK}$	到第一个SCK有效沿的SSEL有效	100	–	–	ns	

组件更改

本节列出了该组件各版本中的主要更改内容。

版本	更改说明	更改原因/影响
1.20	选择SCB未配置模式时，纠正了带有时钟延展操作的EZ I^2C。	编译时，编译器发出警报。
	当主设备写入乘以8的字节数量并从乘以8的基本地址启动时，纠正了带有时钟延展的EZ I^2C模式的缓冲区更新。	EZ I^2C从设备完成传输过早，并且最后8个字节被保留在RX FIFO内。该缓冲区未被正确更新。
	对于大小为256个字节的缓冲区，纠正了EZ I^2C的当前地址性能。	发生缓冲区的读或写操作溢出时，当前地址包括了第一个元素，而并非指向缓冲区外部。
	提高了带有时钟延展的EZ I^2C模式的中断处理时序。	
	添加了PSoC 4000器件的支持。	
1.10	添加了EZ I^2C模式。	
	将SPI/UART内部中断源从TX_EMPTY修改为TX_NOT_FULL。该中断源用于将数据从内部软件缓冲区转移到TX FIFO内。	当软件缓冲区内的数据可用时，该修改会使 TX_FIFO开始保持为满的状态。在传输过程中，由于一个长的中断响应时间，这样会降低FIFO变空的可能性。
1.0.a	编辑组件数据手册以匹配GUI。	
1.0	SCB组件第一次释放	

© 赛普拉斯半导体公司，2014。此处所包含的信息可能会随时更改，恕不另行通知。除赛普拉斯产品内嵌的电路外，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。也不会根据专利权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了明确的书面协议，否则赛普拉斯产品不保证产品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，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。该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。除上述指定用途外，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：赛普拉斯不针对该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。赛普拉斯保留在不另行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。对于合理预计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，则表示制造商将承担因此类使用而导致的所有风险，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。