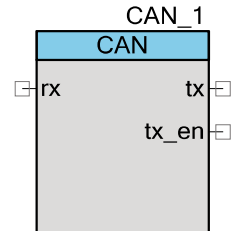


控制器区域网络 (CAN)

2.1

特点

- CAN2.0A/B 协议实现，符合 ISO 11898-1 标准
- 在 8 MHz (BUS_CLK) 时高达 1 Mbps 的可编程比特率
- 两线或三线的接口连接外部收发器 (Tx、Rx 和 Enable (使能))
- 扩展硬件消息过滤器，涵盖“数据字节 1”和“数据字节 2”字段
- 可编程传送优先级：轮循和固定
- 於 PSoC 5 不支持



概述

控制器区域网络 (CAN) 控制器符合 Bosch 规范中定义的 CAN2.0A 和 CAN2.0B 规范，并符合 ISO-11898-1 标准。

何时使用 CAN

CAN 协议最初是针对汽车应用设计的，侧重于高阶的故障检测。能够确保以较低的成本实现高度的通信可靠性。由于在汽车应用中取得了巨大成功，CAN 被用作运动机械控制网络 (CANOpen) 和工厂自动化应用 (DeviceNet) 的标准通信协议。CAN 控制器具有丰富的功能，能够高效实现更高级的协议，而不会影响微控制器 CPU 的性能。

输入/输出连接

本节介绍 CAN 组件的输入和输出连接。I/O 列表中的星号 (*) 表示，在 I/O 说明中列出的情况下，该 I/O 可能不可见。

rx – 输入

CAN 总线接收信号（连接到外部收发器的 CAN Rx 总线）。

tx- 输出

CAN 总线发送信号（连接到外部收发器的 CAN Tx 总线）。

tx_en – 输出 *

外部收发器使能信号。当在 **Configure**（配置）对话框中选择 **Add Transceiver Enable Signal**（添加收发器使能信号）选项时，显示此输出。

中断 – 输出 *

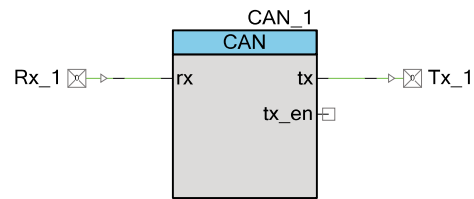
此中断输出由 CAN 硬件中的配置的中断源驱动。所有源一起执行“或”运算以创建最终输出信号。中断的源包括：

- 发送的消息
- 接收的消息
- 已满的接收缓冲区
- 总线关闭状态
- 检测到的 CRC 错误
- 检测到的消息格式错误
- 检测到消息, 确认错误
- 检测到位填充错误
- 检测到误码
- 接收到过载帧
- 检测到仲裁失败

当 **Configure**（配置）对话框中 **Interrupt**（中断）选项卡的 **Advanced Interrupt Configuration...**（高级中断配置...）窗口中选择了 **Enable External Interrupt Line**（启用外部中断线）选项时，显示此输出。

原理图宏信息

组件目录中的默认 CAN 是使用带默认设置的 CAN 组件的原理图宏。CAN 组件连接到输入和输出引脚组件。除在输入引脚组件中“输入同步”设置为假之外，引脚组件也设置为默认设置。



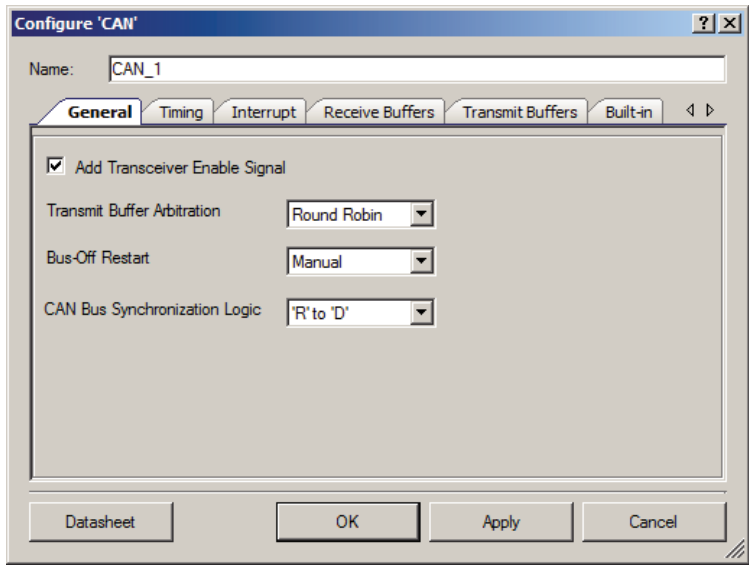
元件参数

将 CAN 组件拖入设计中，双击它以打开 **Configure**（配置）对话框。该对话框有若干选项卡，可引导您完成 CAN 组件的设置过程。

组件更新笔记

如果从之前的版本更新 CAN 组件，您将看到很多参数采用了新的格式。你必须转换它们。为此，打开 **Configure**（配置）对话框，至少更改一个参数选项，并单击 **OK**（确定）以保存更改。

一般选项卡



General（一般）选项卡包含以下设置：

Add Transceiver Enable Signal（添加收发器使能信号）

为外部 CAN 收发器启用或禁用 tx_en 信号的使用。默认启用。



发送缓冲区仲裁

定义消息发送仲裁方案：

- **轮循**（默认）– 缓冲区以所定义的顺序运行：0-1-2 ... 7-0-1。特定的缓冲区仅在设置了其 TxReq 标志时可选择。此方案保证了所有缓冲区具有同等的消息发送概率。
- **固定优先级**– 缓冲区 0 具有最高优先级。这样，可将缓冲区 0 指定为错误消息的缓冲区，保证最先发送错误消息。

总线关闭重新启动

配置复位类型：

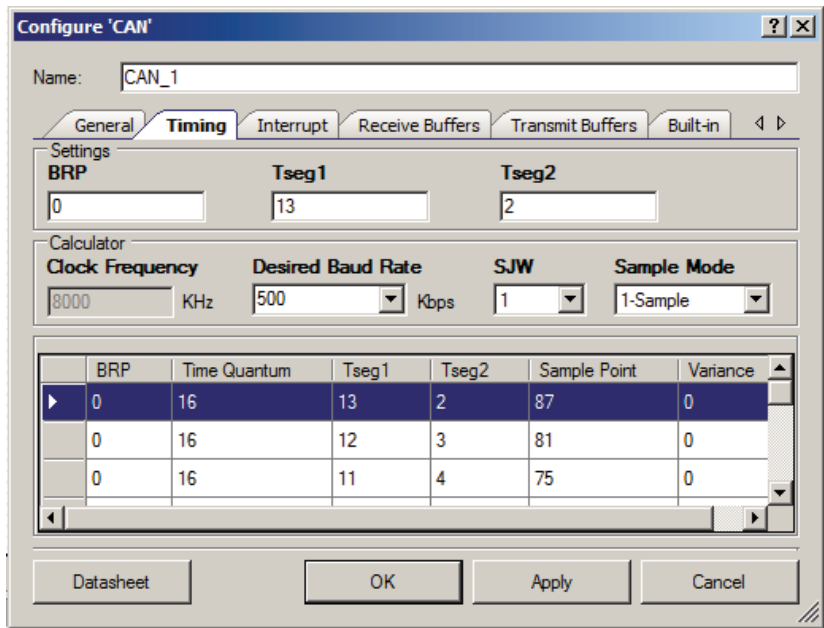
- **手动**（默认）– 总线关闭之后，必须重新启动 CAN。这是推荐设置。
- **自动** – 总线关闭之后，在 128 组 11 个“空闲”位之后，自动重新启动 CAN 控制器。

CAN 总线同步逻辑

配置边沿同步：

- **‘R’ 到 ‘D’**（默认）– 从 ‘R’（空闲）到 ‘D’（占有）的边沿用于进行同步。
- **双边沿** – 双边沿用于进行同步。

定时选项卡

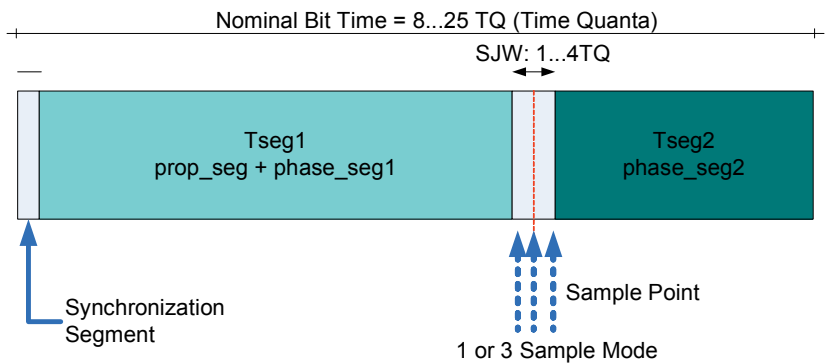


定时选项卡包含以下设置：

设置

- **BRP** – 用于生成时间段的比特率预分频器值。比特计算器用于计算此值。0 表示 1 个时钟周期；7FFFh 表示 32767 个时钟周期，15 个比特。
- **Tseg1** – 时间段 1 的值。
- **Tseg2** – 时间段 2 的值。不允许值 0 和 1；仅在将**样本模式**设置为直接采样 (**1-Sample**) 时才允许值 2。

下面显示了 CAN 比特定时图示：



计算器

- **时钟频率**（单位：MHz） – 系统时钟频率等于 BUS_CLK。
- **需要的波特率**（单位：Kbps） – 选项为：**10、20、62.5、125、250、500、800 或 1000**。
- **SJW** – 同步跳转宽度的配置（2 位）。值不得大于 Tseg1 且不得大于 Tseg2。选项为：**1、2、3 或 4**。
- **样本模式** – 采样模式的配置。选项为：**1-Sample 或 3-Sample**。

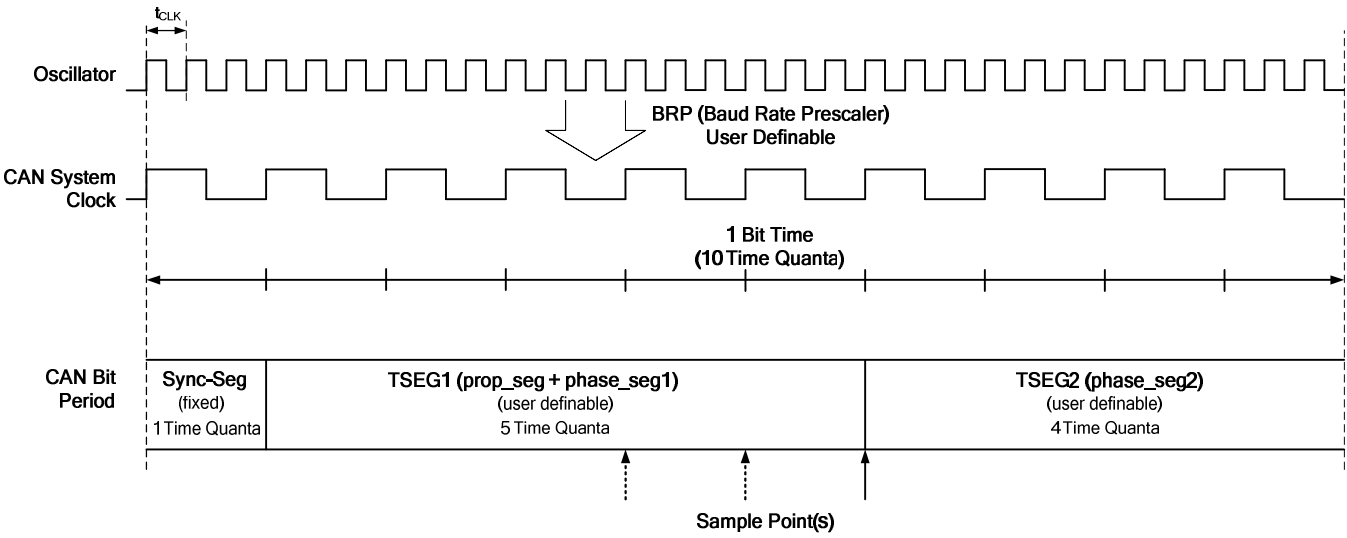
表

在计算了位定时之后，参数表中显示了时间段（Tseg1 和 Tseg2）和 BRP 的推荐寄存器设置。可通过双击相应的行选择要加载的值。选择的值显示在顶部的 **Settings**（设置）输入框。

也可以选择在提供的输入框手动输入 Tseg1、Tseg2 和 BRP 的值。

注意 错误的位定时设置会导致 CAN 控制器保留在错误状态。

下图显示了所有定时是如何衍生自振荡器的示例。



位时间段

SYNC SEG（同步段）

此部分的位时间使总线上的各个节点保持同步。一个边沿预计位于此段中。



PROP_SEG (传输时间段)

此部分的位时间补偿网络中的物理延迟时间。它是总线上信号的传播时间、输入比较器延迟和输出驱动器延迟的总和的两倍。

PHASE_SEG1、PHASE_SEG2 (相位缓冲区间 1/2)

这些相位缓冲区间用于补偿边沿相位错误。可通过重新同步延长或缩短这些区间。

样本点

样本点是总线级别读取并解释为对应位的值的时间点。它位于 PHASE_SEG1 结尾处。

信息处理时间

信息处理时间是始于为计算后续位级别而保留的样本点的时间段。

时间段

时间段是从震荡周期中衍生的固定时间单位。有一个可编程预分频器 (BRP)，整数值范围为 1 到 32767。从最小时间段开始，时间段长度可为

$$\text{时间段} = m \times \text{最小时间段}$$

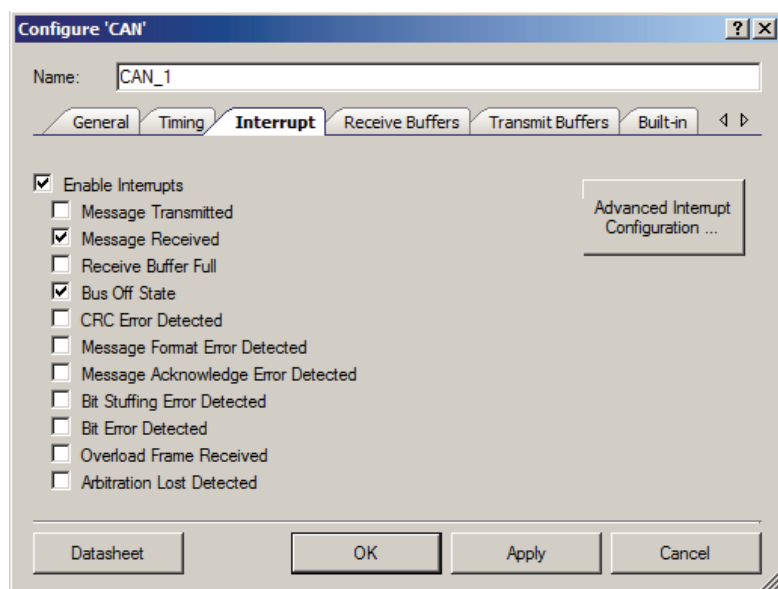
其中 m 是预分频器的值。

时间段长度

- SYNC_SEG 的长度为 1 个时间段。
- PROP_SEG 的长度可编程为 1、2、...、8 个时间段。
- PHASE_SEG1 的长度可编程为 1、2、...、8 个时间段。
- PHASE_SEG2 是 PHASE_SEG1 的最大值和信息处理时间

中断选项卡

基本中断配置



Basic（基本）**Interrupt**（中断）**Configuration**（配置）选项卡包含以下设置：

使能中断

从 CAN 控制器中启用或禁用全局中断。默认启用。

启用 - 在使用 CAN_Start() 启动 CAN 组件时，启用全局中断。

禁用 - 在使用 CAN_Start() 启动 CAN 组件时，不启用全局中断。全局中断启用位设置前不会进入 CAN ISR。您应使用 CAN_GlobalIntEnable() 或 CAN_GlobalIntDisable() 在主代码中启用或禁用全局中断。

发送的消息

启用或禁用消息传输中断，消息传输中断表示已发送了一个消息。默认禁用。禁用消息传输中断选项时，CAN 显示以下消息：**是否禁用所有传输缓冲区中断？**

- **是** - 取消选中 **Message Transmitted**（消息已传输）复选框，并取消选中 **Transmit Buffers**（传输缓冲区）选项卡上的所有单独传输缓冲区中断。
- **否（默认）** - 取消选中 **Message Transmitted**（消息已传输）复选框，并保持 **Transmit Buffers**（传输缓冲区）选项卡上的所有单独传输缓冲区中断不变。
- **取消** - 无更改。

已接收的消息

启用或禁用消息接收中断，消息接收中断表示已接收了一个消息。默认启用。禁用消息接收中断选项时，CAN 显示以下消息：**是否禁用所有接收缓冲区中断？**

- **是**（默认）- 取消选中 **Message Received**（消息已接收）复选框，并取消选中 **Receive Buffers**（接收缓冲区）选项卡上的所有单独接收缓冲区中断。
- **否** - 取消选中 **Message Received**（消息已接收）复选框，并保持 **Receive Buffers**（接收缓冲区）选项卡上的所有单独接收缓冲区中断不变。
- **取消** - 无更改。

已满的接收缓冲区

启用或禁用消息丢失中断，这表示在上一条消息被否认时，CAN 控制器接收到一条新消息。默认禁用。

总线关闭状态

启用或禁用总线关闭中断，这表示 CAN 节点已达到总线关闭状态。默认启用。

检测到的 CRC 错误

启用或禁用 CRC 错误中断，这表示 CAN 控制器检测到 CAN CRC 错误。默认禁用。

检测到的消息格式错误

启用或禁用消息格式错误中断，这表示 CAN 控制器检测到 CAN 消息格式错误。默认禁用。

检测到的消息确认错误

启用或禁用消息确认错误中断，这表示 CAN 控制器检测到 CAN 消息确认错误。默认禁用。

检测到的位填充错误

启用或禁用位填充错误中断，这表示 CAN 控制器检测到位填充错误。默认禁用。

检测到的误码

启用或禁用位错误中断，这表示 CAN 控制器检测到位错误。默认禁用。

接收的过载帧

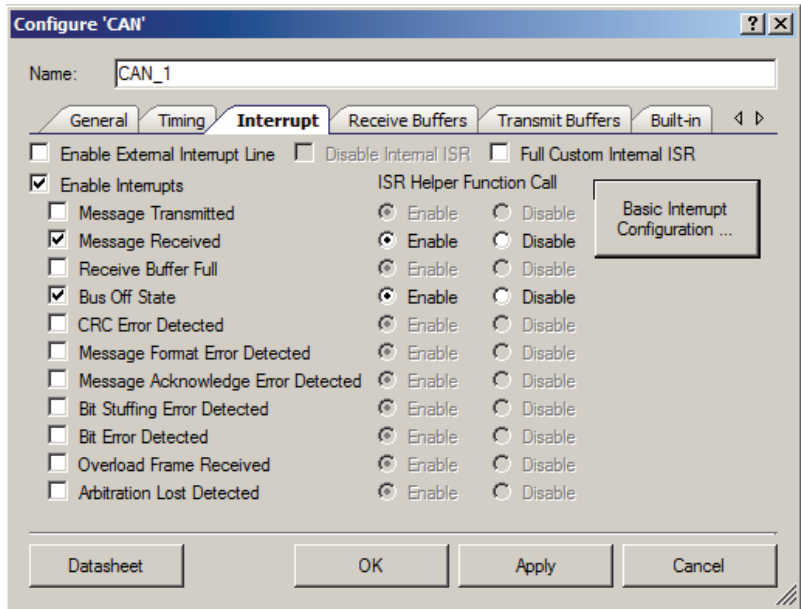
启用或禁用过载中断，这表示 CAN 控制器接收到过载帧。默认禁用。



检测到的仲裁失败

启用或禁用管理已排队消息的仲裁和取消。此中断表示发送消息时丢失仲裁。默认禁用。

高级中断配置



Advanced（高级）**Interrupt Configuration**（中断配置）选项卡包含以下设置：

启用外部中断线

启用 CAN 模块中断线的外部可视性和连接。默认禁用（在 CAN 组件符号实例中，外部中断线不可见）。

禁用内部 ISR

禁用或跳过内部 ISR 组件。如果禁用了内部 ISR，相关 CAN API 不会管理 ISR 启动/停止流程。默认禁用（启用了内部 ISR）。仅在选定了 **Enable External Interrupt Line**（启用外部中断线）时，此复选框才可用（未变灰）。仅在有备用方式（外部中断线）处理中断时，才可禁用内部 ISR。

完全自定义内部 ISR

启用使用带完全自定义代码的内部 ISR。当选定此选项时，CAN_ISR 不含代码只包含可选的 PSoC 3 ES1/ES2 ISR 补丁。将您的自定义代码置于行间：

```
/* Place your Interrupt code here. */
/* `#START CAN_ISR` */

/* `#END` */
```



默认禁用（默认为 CAN v1.50 ISR 处理）。如果选定了 **Disable Internal ISR**（禁用内部 ISR），则复选框不可用（变灰）。

ISR 助手功能调用

如果您仅选择了基本中断设置（例如，如 CAN v1.50 中），当发生中断时，CAN ISR 基于已启用的中断调用相关用户可定制功能（ISR 助手）。

这些选项允许您启用或禁用 ISR 助手调用，从而可在硬件和固件中都可实施对特定中断的自定义处理。默认**启用**。仅在相关中断事件已启用，未选定 **Full Custom Internal ISR**（完全自定义内部 ISR），且未选定 **Disable Internal ISR**（禁用内部 ISR）时，这些选项才可用。

接收缓冲区选项卡

Configure 'CAN'

?

×

Name: CAN_1

General

Timing

Interrupt

Receive Buffers

Transmit Buffers

Built-in

Mailbox	Full	Basic	IDE	ID	RTR	RTRreply	IRQ	Linking
0	☒	☐	☐	0x001	☐	☐	☐	☒
1	☒	☐	☐	0x001	☐	☐	☐	☒
2	☒	☐	☐	0x001	☐	☐	☐	☒
3	☒	☐	☐	0x001	☐	☐	☒	☐
4	☒	☐	☐	0x430	☐	☐	☒	☐
5	☒	☐	☐	0x255	☒	☒	☐	☐
6	☐	☒	☐		☐	☐	☐	☐
7	☐	☐	☐		☐	☐	☐	☐
8	☐	☐	☐		☐	☐	☐	☐
9	☐	☐	☐		☐	☐	☐	☐
10	☐	☐	☐		☐	☐	☐	☐
11	☐	☐	☐		☐	☐	☐	☐

Datasheet

OK

Apply

Cancel

Receive Buffers（接收缓冲区）选项卡包含以下设置：

邮箱

接收邮箱被禁用，直至选择了 **Full**（完全）或 **Basic**（基本）。所有已禁用邮箱的 **IDE**、**ID**、**RTR**、**RTRreply** 和 **IRQ** 字段都被锁定。

对于完全邮箱，**Mailbox**（邮箱）字段可编辑，您可输入唯一的邮箱名称。用于管理各个邮箱的 API 将附加邮箱字符串。接受的符号有：**A–Z**、**a–z**、**0–9** 和 **_**。如果您输入错误的名称，将显示错误消息，且 **Mailbox**（邮箱）字段返回默认值。



完全

选定了 **Full**（完全）时，可修改 **Mailbox**（邮箱）、**IDE**、**ID**、**RTR**、**RTRreply**、**IRQ** 和 **Linking**（链接）字段。这些字段使用以下默认值进行初始化：

- **邮箱** = 邮箱编号 0 到 15
- **IDE** = 已清除
- **ID** = 0x001
- **RTR** = 已清除
- **RTRreply** = 已清除和已锁定（仅在选择了 **RTR** 时启用）
- **IRQ** = 已选定，仅在选择了 **Message Received**（消息已接收）（**中断**选项卡）时可用
- **链接** = 已清除

基本

如果选择了 **基本**，选项 **IDE**、**ID**、**RTR** 和 **RTRreply** 不可用。这些字段使用以下默认值进行初始化：

- **IDE** = 已清除（不可用）
- **ID** = <全部>（不可用）
- **RTR** = 已清除（不可用）
- **RTRreply** = 已清除（不可用）
- **IRQ** = 已清除，仅在选择了 **Message Received**（消息已接收）中断时可用
- **链接** = 已清除

IDE

当 **IDE** 复选框被清除时，标识符长度限制为 11 位（0x001 到 0x7FE）。选择了 **IDE** 时，标识符长度限制为 29 位（0x00000001 到 0x1FFFFFFE）。

RTR - 远程传输请求

仅适用于设置为接收完全 **CAN** 消息的邮箱。选择此选项后，将配置接受过滤设置，以仅允许接收已设置了 **RTR** 位的消息。

RTRreply - 远程传输请求自动回复

仅适用于设置为接收完全 CAN 消息，并设置了 RTR 位的邮箱。选择此选项后，将用接收缓冲区的内容自动回复 RTR 请求。

IRQ

针对邮箱启用 IRQ 时，如果清除了中断选项卡中的 **Message Received Interrupt**（消息接收中断），将显示以下消息：全局“消息接收中断”已禁用。是否启用它？

- **是** - 选择 **IRQ** 复选框，并选择 **Interrupt**（中断）选项卡上的 **Message Received Interrupt**（消息接收中断）复选框。
- **否或取消** - 选择 **IRQ** 复选框，并保持 **Interrupt**（中断）选项卡上的 **Message Received Interrupt**（消息接收中断）复选框不变。

链接

选择 **Linking**（链接）复选框，可链接多个连续的接收邮箱，以创建接收邮箱阵列。此阵列与接收 FIFO 的功能类似。同一阵列的所有邮箱必须具有相同的消息过滤设置，也就是说，验收屏蔽寄存器 (AMR) 和验收编码寄存器 (ACR) 一致。

- 阵列的最后一个邮箱不可设置链接标志。
- 最后一个邮箱 15 不可设置链接标志。
- 所有已链接的邮箱用同一种颜色突出显示。
- 链接中只有第一个邮箱可编辑。所有参数自动应用到同一阵列中所有链接的邮箱。
- 针对所有已链接邮箱生成一个函数。

接收消息函数

每个完全 RX 邮箱都有一个预定义的 API。**CAN_1_TX_RX_func.c** 项目文件中有函数列表。这些函数是按照条件编译的，这取决于接收邮箱的设置。只有定义为完全的邮箱才编译各自的函数。

宏观标识符 **CAN_1_RXx_FUNC_ENABLE** 定义是否编译函数。定义列示在 **CAN_1.h** 项目文件中。

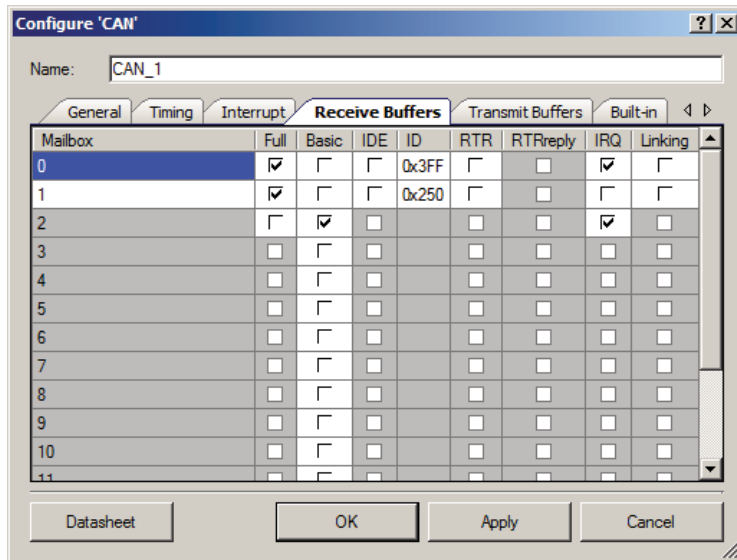
- 当发生消息接收中断时，将调用 **CAN_MsgRXIsr()** 函数。此函数依次通过所有接收邮箱，并检查其各自的“消息可用标志”（MsgAv - 读取：0 无新消息可用；1 有新消息可用）和“中断使能”（接收中断使能：0 禁用中断生成，1 启用中断生成），以成功接收 CAN 消息。
- 如果启用了 **Message Receive**（消息接收）中断，当接收到消息时，将调用 **CAN_ReceiveMsgX()** 函数，其中 X 表示完全 CAN 邮箱编号或用户定义的名称。



- 对于所有基于中断的基本 CAN 邮箱，将调用 `CAN_ReceiveMsg(uint8 rxMailbox)` 函数，其中，`rxMailbox` 参数表示接收到消息的邮箱的编号。

接收缓冲区配置

以下范例说明接收消息 API 的用法。



CAN_1.h 文件:

```
...
#define CAN_1_RX0_FUNC_ENABLE 1
#define CAN_1_RX1_FUNC_ENABLE 0
#define CAN_1_RX2_FUNC_ENABLE 1
...
```

CAN_1_TX_RX_func.c 文件:

```
#if(CAN_1_RX0_FUNC_ENABLE)
void CAN_1_ReceiveMsg0(void)
{
    /* `#START MESSAGE_0_RECEIVED` */

    /* `#END` */

    CAN_1_RX[0].rxcmd.byte[0] |= CAN_1_RX_ACK_MSG;
}
#endif

#if(CAN_1_RX1_FUNC_ENABLE)
void CAN_1_ReceiveMsg1(void)
{

```

```

/* `#START MESSAGE_1_RECEIVED` */

/* `#END` */
CAN_1_RX[1].rxcmd.byte[0] |= CAN_1_RX_ACK_MSG;
}
#endif

```

针对 **CAN** 接收消息 2 调用以下函数，配置为基本 **CAN** 并启用中断。

```

void CAN_ReceiveMsg(uint8 rxMailbox)
{
    if (CAN_1_RX[rxMailbox].rxcmd.byte[0] & CAN_1_RX_ACK_MSG)
    {
        /* `#START MESSAGE_BASIC_RECEIVED` */

        /* `#END` */
        CAN_1_RX[rxMailbox].rxcmd.byte[0] |= CAN_1_RX_ACK_MSG;
    }
}

```

CAN_1_INT.c 文件

```

void CAN_1_MsgRXIsr(void)
{
    ...
    /* RX Full mailboxes handler */
    switch(i)
    {
        case 0 : CAN_ReceiveMsg0();
                break;
        case 1 : CAN_ReceiveMsg1();
                break;
        default:
                break;
    }
    ...
}

```

如果实施了链接，条件编译应用于设置了 **IRQ** 标志的邮箱。所有接收函数通过清除消息可用 (**MsgAv**) 标志来确认消息接收。

如何设置 **AMR** 和 **ACR** 在可接受 ID 范围

以下是 **ACR/AMR** 寄存器示例：

[31:3] – Identifier(ID[31:21] – identifier when IDE = 0, ID[31:3] - identifier when IDE = 1), [2] – IDE, [1] – RTR, [0] – N/A;

验收屏蔽寄存器 (**AMR**) 定义是否针对验收编码寄存器 (**ACR**) 检查输入位。

AMR: ‘0’ 针对各个 **ACR** 检查输入位。当输入位不匹配各自的 **ACR** 位时，不接受消息。
 ‘1’ 输入位无需关注。



例如，要设置邮箱以接收在 0x180-187、IDE = 0（已清除）、RTR = 0（已清除）、邮箱 5 范围内的 ID，执行以下附加操作：

采用 ID 的较低范围，例如 0x180，以及相应的 IDE 和 RTR。对于此 ID、IDE 和 RTR 值，AMR 和 ACR 寄存器设置为以下值：

- | | |
|----------------------|--------------------------------|
| ■ ACR[31:21] = 0x180 | AMR[31:21] = 0x0 |
| ■ ACR[20:3] = 0x0 | AMR[20:3] = 0x3FFFF (all ones) |
| ■ ACR[2] = 0 | AMR[2] = 0 |
| ■ ACR[1] = 0 | AMR[1] = 0 |
| ■ ACR[0] = 0 | AMR[0] = 0 |

定义范围 ID（11 位）的共用部分：

- 0x180 = 0`b001 1000 0000
- 0x187 = 0`b001 1000 0111
- Mask = 0`b001 1000 0XXX
- ACR[31:21] = 0`b000 0000 0111

您必须在 AMR 寄存器中用 1 替代“XXX”及空字符而非共用位。这样，后续值为：

- | | |
|----------------------|---------------------------------|
| ■ ACR[31:21] = 0x180 | AMR[31:21] = 0x7; |
| ■ ACR[20:3] = 0x0 | AMR[20:3] = 0x3FFFF - all ones; |
| ■ ACR[2] = 0 | AMR[2] = 0 |
| ■ ACR[1] = 0 | AMR[1] = 0 |
| ■ ACR[0] = 0 | AMR[0] = 0 |

使用 CAN_RXRegisterInit() 函数写入邮箱编号 5 的 AMR 寄存器：

```
uint8 result = FAIL;
uint16 temp_amr;
uint16 temp_acr;

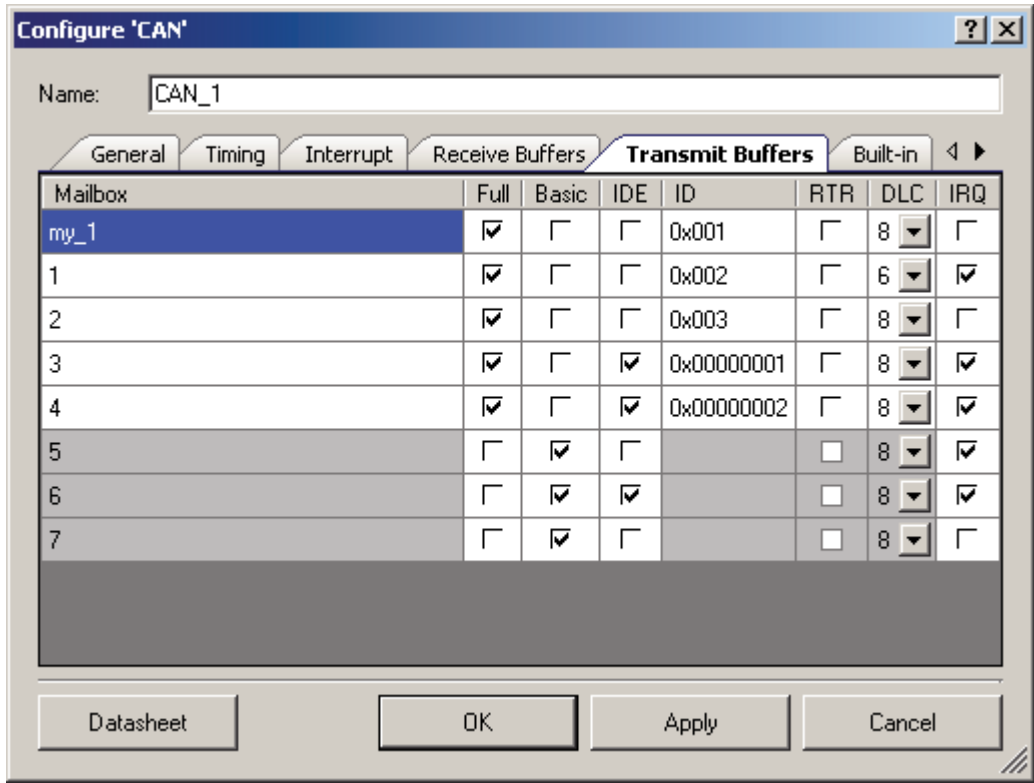
/* Upper address value, so address is shifted */
temp_amr = (0x7 << 21); /* obtain necessary value to put in AMR */
temp_acr = (0x180 << 21); /* obtain necessary value to put in ACR */

if (CAN_RXRegisterInit(&CAN_1_RX[5].rxamr, temp_amr))
{
    if ((CAN_RXRegisterInit(&CAN_1_RX[5].rxacr, temp_acr))
        result = SUCCESS;
```

```
}
return result; /* error - if return no zero value; */
```

有关 AMR 和 ACR 配置的更多详细信息，请参见 [PSoC® 3](#) 和 [PSoC 5® 技术参考手册](#) 中的“控制器区域网络 (CAN)”章节。

传输缓冲区选项卡



Transmit Buffers（传输缓冲区）选项卡包含以下设置：

邮箱

对于完全邮箱，**Mailbox**（邮箱）字段可编辑，您可输入一个唯一的邮箱名称。用于管理各个邮箱的函数将附加邮箱字符串。接受的字符有：**A–Z**、**a–z**、**0–9** 和 **_**。如果输入错误名称，**Mailbox**（邮箱）字段恢复为默认值。

完全

选定了 **Full**（完全）时，可修改 **Mailbox**（邮箱）、**IDE**、**ID**、**RTR**、**RTRreply**、**IRQ** 和 **Linking**（链接）字段。这些字段使用以下默认值进行初始化：

- **邮箱** = 编号 0 到 7
- **IDE** = 已清除



- **ID** = 0x01
- **RTR** = 已清除
- **DLC** = 8
- **IRQ** = 已清除

基本

默认情况下，所有邮箱都选择了 **Basic**（基本）复选框。如果选择了**基本**，选项 **ID**、**RTR** 和 **DLC** 不可用。如果选择了**基本**，使用代码输入所需的 CAN 消息字段。这些字段使用以下默认值进行初始化：

- **IDE** = 已清除
- **ID** = 无（不可用）
- **RTR** = 已清除（不可用）
- **DLC** = 8（不可用）
- **IRQ** = 已清除

IDE

如果 **IDE** 复选框已清除，标识符长度不可超过 11 位（从 0x001 到 0x7FE）。如果选择了 **IDE**，允许使用 29 位标识符（从 0x00000001 到 0x1FFFFFFE）。不可选择 0x000 或 0x7FF（11 位）或 0x1FFFFFFF（29 位）标识符。

ID

消息标识符。

RTR

此消息是返回传输请求消息。

DLC

消息包含的字节数。

IRQ

IRQ 位取决于 **Message Transmitted**（消息已传输）（**中断**选项卡）。

如果 **Message Transmitted**（消息已传输）复选框已清除，选择 **IRQ** 时，将显示以下消息：全局“消息传输中断”已禁用。是否启用它？

- 是 - 选择 **IRQ** 和中断选项卡中的 **Message Transmitted Interrupt**（消息传输中断）复选框。
- 否或取消 - 选择 **IRQ**。中断选项卡中的 **Message Transmitted Interrupt**（消息传输中断）复选框保持已清除状态。

CAN TX 函数

每个完全 TX 邮箱都有一个预定义的 API。*CAN_1_TX_RX_func.c* 项目文件中有函数列表。这些函数是按照条件编译的，这取决于传输邮箱的设置。只有定义为完全的邮箱才编译各自的函数。

宏观标识符 *CAN_1_TXx_FUNC_ENABLE* 定义是否编译函数。定义列示在 *CAN_1.h* 项目文件中。

CAN_SendMsgX() 函数适用于所有配置为全部的 Tx 邮箱，其中 *X* 表示全部 CAN 邮箱编号或用户定义的名称。

传输缓冲区配置

以下范例说明传输 API 的用法。

Configure 'CAN'

Name: CAN_1

GeneralTimingInterruptReceive BuffersTransmit BuffersBuilt-in

Mailbox	Full	Basic	IDE	ID	RTR	DLC	IRQ
0	☒	☐	☐	0x001	☐	8	☐
1	☐	☒	☐		☐	8	☐
2	☐	☒	☐		☐	8	☐
3	☐	☒	☐		☐	8	☐
4	☐	☒	☐		☐	8	☐
5	☐	☒	☐		☐	8	☐
6	☐	☒	☐		☐	8	☐
7	☐	☒	☐		☐	8	☐

DatasheetOKApplyCancel

CAN_1.h 文件

```
#define CAN_1_TX0_FUNC_ENABLE 1
#define CAN_1_TX1_FUNC_ENABLE 0
```



CAN_1_TX_RX_func.c 文件

```

#if(CAN_1_TX0_FUNC_ENABLE)
uint8 CAN_1_SendMsg0(void)
{
    uint8 result = SUCCESS;
    if (CAN_1_TX[0u].txcmd.byte[0u] & CAN_1_TX_REQUEST_PENDING) ==
        CAN_1_TX_REQUEST_PENDING)
    {
        result = CAN_FAIL;
    }
    else
    {
        /* `#START MESSAGE_0_TRANSMITTED` */

        /* `#END` */

        CY_SET_REG32((reg32 *) & CAN_1_TX[0u] |= txcmd CAN_SEND_MESSAGE);
    }

    return result;
}
#endif

```

为所有基本传输邮箱提供的共用函数为:

```
uint8 CAN_SendMsg(CAN_TX_MSG *message)
```

为用于收集 CAN 传输消息所需数据的应用程序定义的一般结构:

- ID - 如果 ID 槽包括以下内容时的限制:
 - 对于标准消息 (IDE = 0), 标识符长度限制为 11 位 (0x001 到 0x7FE)。
 - 对于扩展消息 (IDE = 1), 标识符长度限制为 29 位 (0x00000001 到 0x1FFFFFFE)。
- RTR (0 - 标准消息, 1 - 0xFF: 消息中设置的 RTR 位)
- IDE (0 - 标准消息, 1 - 0xFF: 扩展消息)
- DLC (定义数据字节 0 到 8, 9 到 0xFF 的数量等于 8 个数据字节)
- IRQ (0 - IRQ 使能, 1 - 0xFF: IRQ 禁用)
- DATA_BYTES (指针指向代表传输数据的 8 字节的结构)

调用 CAN_SendMsg() 函数时, 函数遍历被指定为基本 CAN 邮箱的传输消息邮箱, 并寻找第一个可用邮箱:

- 当找到一个闲置的基本 CAN 邮箱时, 通过 CAN_TX_MSG 结构的数据被复制到相应的 CAN 传输邮箱中。当消息进入传输队列时, 将返回“成功”指示到应用程序。

- 如未找到闲置基本邮箱，函数将再进行三次尝试。如果所有尝试都失败了，将返回“失败”指示到应用程序。

CAN_TX_MSG 结构包含传输消息所需的所有信息：

```
typedef struct _CAN_TX_MSG
{
    uint32 id;
    uint8 rtr;
    uint8 ide;
    uint8 dlc;
    uint8 irq;
    CAN_DATA_BYTES_MSG *msg;
} CAN_TX_MSG;
```

在一条消息中，**CAN_DATA_BYTES** 结构包含八个数据字节。

```
typedef struct CAN_DATA_BYTES_MSG
{
    uint8 byte[8u];
} CAN_DATA_BYTES_MSG;
```

时钟选择

CAN 组件连接到 **BUS_CLK** 时钟信号。需要最小 8 MHz 以支持所有高达 1 Mbps 的标准 CAN 波特率。

放置

CAN 组件放置在固定函数 CAN 模块中。

Resources（资源）

Description（说明）	Digital resources （数字资源）		API Memory（API 存储器） （字节）		Pins（引脚） （每个外部 I/O）
	CAN IP 模块	Interrupts （中断）	Flash （闪存）	RAM	
tx_en 信号和外部中断信号都禁用	1*	1	4156	18	2
tx_en 信号启用，外部中断信号禁用	1*	1	4156	18	3
tx_en 信号和外部中断信号都启用	1*	1	4156	18	4

*CAN 组件使用芯片中的专用 CAN 硬件模块。

应用程序编程接口

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

默认情况下，PSoC Creator 将实例名称“CAN_1”分配给指定设计中组件的第一个实例。您可以将该实例重命名为符合标识语法规则的任意唯一值。实例名称会成为每个全局函数名称、变量和常量符号的前缀。出于可读性考虑，下表中使用的实例名称为“CAN”。

函数	Description（说明）
CAN_Start()	设置 initVar 变量，调用 CAN_Init() 函数，然后调用 CAN_Enable() 函数。
CAN_Stop()	禁用 CAN 组件。
CAN_GlobalIntEnable()	从 CAN 组件中启用全局中断。
CAN_GlobalIntDisable()	从 CAN 组件中禁用全局中断。
CAN_SetPreScaler()	设置从 BUS_CLK 生成时间段的预分频器。
CAN_SetArbiter()	设置传输缓冲区的仲裁类型
CAN_SetTsegSample()	配置：时间段 1、时间段 2、同步跳转宽度和采样模式。
CAN_SetRestartType()	设置复位类型。
CAN_SetEdgeMode()	设置沿模式。
CAN_RXRegisterInit()	仅写入接收 CAN 寄存器。
CAN_SetOpMode()	设置操作模式。

函数	Description (说明)
CAN_GetTXErrorflag()	返回标志, 说明传输错误数是否超过 0x60。
CAN_GetRXErrorflag()	返回标志, 说明接收错误数是否超过 0x60。
CAN_GetTXErrorCount()	返回传输错误数。
CAN_GetRXErrorCount()	返回接收错误数。
CAN_GetErrorState()	返回 CAN 组件的错误状态。
CAN_SetIrqMask()	设置以启用或禁用特定的中断源。
CAN_ArbLostIsrc()	清除仲裁丢失中断标志。
CAN_OvrLdErrorIsrc()	清除过载错误中断标志。
CAN_BitErrorIsrc()	清除位错误中断标志。
CAN_BitStuffErrorIsrc()	清除位填充错误中断标志。
CAN_AckErrorIsrc()	清除确认错误中断标志。
CAN_MsgErrorIsrc()	清除格式错误中断标志。
CAN_CrcErrorIsrc()	清除 CRC 错误中断标志。
CAN_BusOffIsrc()	清除总线关闭中断标志。将 CAN 组件设为停止模式。
CAN_MsgLostIsrc()	清除消息丢失中断标志。
CAN_MsgTXIsrc()	清除传输消息中断标志。
CAN_MsgRXIsrc()	清除接收消息中断标志, 并针对基于基本和完全中断的邮箱调用正确的处理程序。
CAN_RxBufConfig()	为特定邮箱配置所有接收寄存器。
CAN_TxBufConfig()	为特定邮箱配置所有传输寄存器。
CAN_SendMsg()	从一个基本邮箱中发送消息。
CAN_SendMsg0-7()	查看邮箱 0-7 是否有未传输的消息等待仲裁。
CAN_TxCancel()	取消传输已排队等候传输的消息。
CAN_ReceiveMsg0-15()	确认收到新消息。
CAN_ReceiveMsg()	清除接收特定消息中断标志。
CAN_Sleep()	使 CAN 组件做好睡眠准备。
CAN_Wakeup()	使 CAN 组件做好唤醒准备。
CAN_Init()	根据“配置”对话框设置, 初始化或恢复 CAN 组件。
CAN_Enable()	启用 CAN 组件。
CAN_SaveConfig()	保存当前配置。

函数	Description（说明）
CAN_RestoreConfig()	恢复配置。

对于返回执行指示的函数，0 表示“成功”，1 表示“失败”，2 表示“超出范围”。

全局变量

变量	Description（说明）
CAN_initVar	说明 CAN 是否已初始化。变量将初始化为 0，并在第一次调用 CAN_Start() 时设置为 1。这样，在第一次调用 CAN_Start() 子程序后，组件不用重新初始化即可重启。 如需重新初始化组件，可在 CAN_Start() 或 CAN_Enable() 函数前调用 CAN_Init() 函数。

uint8 CAN_Start(void)

说明： 此函数设置 initVar 变量，调用 Init 函数，然后调用 CAN_Enable() 函数。此函数将 CAN 组件设为运行模式，并在轮询邮箱可用时启动计数器。

参数： None（无）

Return Value（返回值）： uint8：说明寄存器是否已写入和验证

Side Effects（副作用）： 如果已设置 initVar 变量，则该函数仅调用 CAN_Enable() 函数。

uint8 CAN_Stop(void)

说明： 此函数将 CAN 组件设为停止模式，并在轮询邮箱可用时停止计数器。

参数： None（无）

Return Value（返回值）： uint8：说明寄存器是否已写入和验证

Side Effects（副作用）： None（无）

uint8 CAN_GlobalIntEnable(void)

说明:	此函数从 CAN 组件中启用全局中断。
参数:	None (无)
Return Value (返回值):	uint8: 说明寄存器是否已写入和验证
Side Effects (副作用):	None (无)

uint8 CAN_GlobalIntDisable(void)

说明:	此函数从 CAN 组件中禁用全局中断。
参数:	None (无)
Return Value (返回值):	uint8: 说明寄存器是否已写入和验证
Side Effects (副作用):	None (无)

uint8 CAN_SetPreScaler(uint16 bitrate)

说明:	此函数设置从 BUS_CLK 生成时间段的预分频器。0x0 和 0x7FFF 之间的值有效。
参数:	uint16 bitrate: 预分频器值
Return Value (返回值):	uint8: 说明寄存器是否已写入和验证
Side Effects (副作用):	None (无)

uint8 CAN_SetArbiter(uint8 arbiter)

说明:	此函数设置传输缓冲区的仲裁类型。仲裁器类型是循环和固定优先级。值 0 和 1 有效。
参数:	uint8 arbiter: 仲裁器类型
Return Value (返回值):	uint8: 说明寄存器是否已写入和验证
Side Effects (副作用):	None (无)



uint8 CAN_SetTsegSample(uint8 cfgTseg1, uint8 cfgTseg2, uint8 sjw, uint8 sm)

说明: 此函数配置：时间段 1、时间段 2、同步跳转宽度和采样模式。

参数: uint8 cfgTseg1: 时间段 1，介于 0x2 和 0xF 之间的值有效

uint8 cfgTseg2: 时间段 2，介于 0x1 和 0x7 之间的值有效

uint8 sjw: 同步跳转宽度，介于 0x0 和 0x3 之间的值有效。

uint8 sm: 采样模式，使用一或三个采样点

Return Value (返回值): uint8: 说明寄存器是否已写入和验证

Side Effects (副作用): None (无)

uint8 CAN_SetRestartType(uint8 reset)

说明: 此函数设置复位类型。复位类型为自动和手动。手动复位为推荐设置。值 0 和 1 有效。

参数: uint8 复位: 复位类型

Return Value (返回值): uint8: 说明寄存器是否已写入和验证

Side Effects (副作用): None (无)

uint8 CAN_SetEdgeMode(uint8 edge)

说明: 此函数设置沿模式。模式为“R”到“D”（空闲到占有），同时使用两种沿。值 0 和 1 有效。

参数: uint8 edge: 沿模式

Return Value (返回值): uint8: 说明寄存器是否已写入和验证

Side Effects (副作用): None (无)

uint8 CAN_RXRegisterInit(uint32 *regAddr, uint32 config)

说明: 此函数仅写入 CAN 接收寄存器。

参数: uint32 * regAddr: 指针指向 CAN 接收寄存器
uint32 configuration: 将写入到寄存器中的值

Return Value (返回值): uint8: 说明寄存器是否已写入和验证

Side Effects (副作用): None (无)

uint8 CAN_SetOpMode(uint8 opMode)

说明: 此函数设置操作模式。操作模式为“活动”或“仅侦听”。值 0 和 1 有效。

参数: uint8 opMode: 操作模式值

Return Value (返回值): uint8: 说明寄存器是否已写入和验证

Side Effects (副作用): None (无)

uint8 CAN_GetTXErrorflag(void)

说明: 此函数返回标志，说明传输错误数是否超过 0x60。

参数: None (无)

Return Value (返回值): uint8: 说明传输错误数是否超过 0x60

Side Effects (副作用): None (无)

uint8 CAN_GetRXErrorflag(void)

说明: 此函数返回标志，说明接收错误数是否超过 0x60。

参数: None (无)

Return Value (返回值): uint8: 说明接收错误数是否超过 0x60

Side Effects (副作用): None (无)



uint8 CAN_GetTXErrorCount(void)

说明: 此函数返回传输错误数。

参数: None (无)

Return Value
(返回值): uint8: 传输错误数

Side Effects
(副作用): None (无)

uint8 CAN_GetRXErrorCount(void)

说明: 此函数返回接收错误数。

参数: None (无)

Return Value
(返回值): uint8: 接收错误数

Side Effects
(副作用): None (无)

uint8 CAN_GetErrorState(void)

说明: 此函数返回 CAN 组件的错误状态。

参数: None (无)

Return Value
(返回值): uint8: 错误状态

Side Effects
(副作用): None (无)

uint8 CAN_SetIrqMask(uint16 mask)

说明: 此函数启用或禁用特定中断源。中断屏蔽直接写入到 CAN 中断使能寄存器。

参数: uint8 request: 中断使能或禁用请求。每个中断源一位元

Return Value
(返回值): uint8: 说明寄存器是否已写入和验证

Side Effects
(副作用): None (无)

void CAN_ArbLostIsr(void)

说明: 此函数是仲裁丢失中断的入口点。它清除仲裁丢失中断标志。仅在仲裁丢失中断参数启用时才生成此函数。

参数: None (无)

Return Value (返回值): None (无)

Side Effects (副作用): None (无)

void CAN_OvrLdErrorIsr(void)

说明: 此函数是过载错误中断的入口点。它清除过载错误中断标志。仅在过载错误中断参数启用时才生成此函数。

参数: None (无)

Return Value (返回值): None (无)

Side Effects (副作用): None (无)

void CAN_BitErrorIsr(void)

说明: 此函数是位错误中断的入口点。它清除位错误中断标志。仅在位错误中断参数启用时才生成此函数。

参数: None (无)

Return Value (返回值): None (无)

Side Effects (副作用): None (无)



void CAN_BitStuffErrorIsr(void)

说明: 此函数是位填充错误中断的入口点。它清除位填充错误中断标志。仅在位填充错误中断参数启用时才生成此函数。

参数: None (无)

Return Value
(返回值): None (无)

Side Effects
(副作用): None (无)

void CAN_AckErrorIsr(void)

说明: 此函数是确认错误中断的入口点。它清除确认错误中断标志。仅在确认错误中断参数启用时才生成此函数。

参数: None (无)

Return Value
(返回值): None (无)

Side Effects
(副作用): None (无)

void CAN_MsgErrorIsr(void)

说明: 此函数是格式错误中断的入口点。它清除格式错误中断标志。仅在格式错误中断参数启用时才生成此函数。

参数: None (无)

Return Value
(返回值): None (无)

Side Effects
(副作用): None (无)

void CAN_CrcErrorIsr(void)

说明: 此函数是 CRC 错误中断的入口点。它清除 CRC 错误中断标志。仅在 CRC 错误中断参数启用时才生成此函数。

参数: None (无)

Return Value
(返回值): None (无)

Side Effects
(副作用): None (无)

void CAN_BusOffIsr(void)

说明: 此函数是总线关闭中断的入口点。它将 CAN 组件设为停止模式。仅在总线关闭中断参数启用时才生成此函数。建议启用此中断。

参数: None (无)

Return Value
(返回值): None (无)

Side Effects
(副作用): 停止 CAN 组件操作

void CAN_MsgLostIsr(void)

说明: 此函数是消息丢失中断的入口点。它清除消息丢失中断标志。仅在消息丢失中断参数启用时才生成此函数。

参数: None (无)

Return Value
(返回值): None (无)

Side Effects
(副作用): None (无)



void CAN_MsgTXIsr(void)

说明:	此函数是传输消息中断的入口点。它清除传输消息中断标志。仅在传输消息中断参数启用时才生成此函数。
参数:	None (无)
Return Value (返回值):	None (无)
Side Effects (副作用):	None (无)

void CAN_MsgRXIsr(void)

说明:	此函数是接收消息中断的入口点。它清除接收消息中断标志，并针对基本和完全中断邮箱调用正确的处理程序。仅在接收消息中断参数启用时才生成此函数。建议启用此中断。
参数:	None (无)
Return Value (返回值):	None (无)
Side Effects (副作用):	None (无)

uint8 CAN_RxBufConfig(CAN_RX_CFG *rxConfig)

说明:	此函数为特定邮箱配置所有接收寄存器。邮箱编号包含 CAN_RX_CFG 结构。
参数:	CAN_RX_CFG * rxConfig: 指向某结构的指针，该结构包含为特定邮箱配置所有接收寄存器所需的所有值
Return Value (返回值):	uint8: 说明是否已接收或拒绝了特定的接收寄存器配置
Side Effects (副作用):	None (无)

uint8 CAN_TxBufConfig(CAN_TX_CFG *txConfig)

- 说明:** 此函数为特定邮箱配置所有传输寄存器。邮箱编号包含 CAN_TX_CFG 结构。
- 参数:** CAN_TX_CFG * txConfig: 指向某结构的指针, 该结构包含为特定邮箱配置所有传输寄存器所需的所有值
- Return Value (返回值):** uint8: 说明是否已接受或拒绝了特定的传输寄存器配置
- Side Effects (副作用):** None (无)

uint8 CAN_SendMsg(CANTXMsg *message)

- 说明:** 此函数遍历所有指定为基本 CAN 邮箱的传输消息邮箱。它查找第一个闲置可用邮箱, 并从该邮箱发送消息。如未找到闲置基本邮箱, 将再次尝试最多三次。
- 参数:** CAN_TX_MSG * message: 指针, 指向包含发送消息所需的数据的结构
- Return Value (返回值):** uint8: 说明是否已发送消息
- Side Effects (副作用):** None (无)

uint8 CAN_SendMsg0-7(void)

- 说明:** 这些函数是传输消息 0-7 的入口点。它们查看邮箱 07 是否有未传输的消息等待仲裁。如果有, 它们启动消息传输。仅针对被指定为“完全”的传输邮箱生成这些函数。
- 参数:** None (无)
- Return Value (返回值):** uint8: 说明是否已发送消息
- Side Effects (副作用):** None (无)

void CAN_TxCancel(uint8 bufferId)

- 说明:** 此函数取消已排队等候传输的消息的传输。有效值范围介于 0 和 15 之间。
- 参数:** uint8 bufferId: Tx 邮箱数量
- Return Value (返回值):** None (无)
- Side Effects (副作用):** None (无)



void CAN_ReceiveMsg0-15(void)

说明:	此函数是接收消息 0-15 中断的入口点。它们清除接收消息 0-15 中断标志。仅针对指定为“完全”中断的接收邮箱生成这些函数。
参数:	None (无)
Return Value (返回值):	None (无)
Side Effects (副作用):	None (无)

void CAN_ReceiveMsg(uint8 rxMailbox)

说明:	此函数是基本邮箱的接收消息中断的入口点。它清除接收特定消息中断标志。仅在一个接收邮箱被指定为“基本”时，才生成此函数。
参数:	uint8 rxMailbox: 触发接收消息中断的邮箱编号
Return Value (返回值):	None (无)
Side Effects (副作用):	None (无)

void CAN_Sleep(void)

说明:	这是准备组件睡眠的首选子程序。CAN_Sleep() 子程序保存当前组件的状态。然后它调用 CAN_Stop() 函数，并调用 CAN_SaveConfig() 函数保存硬件配置。 在调用 CyPmSleep() 或 CyPmHibernate() 函数之前调用 CAN_Sleep() 函数。有关电源管理函数的更多信息，请参考 PSoC Creator <i>System Reference Guide</i> (《系统参考指南》)。
参数:	None (无)
Return Value (返回值):	None (无)
Side Effects (副作用):	None (无)

void CAN_Wakeup(void)

- 说明:** 该函数是将组件恢复到调用 CAN_Sleep() 时状态的首选子程序。CAN_Wakeup() 函数调用 CAN_RestoreConfig() 函数以恢复用户配置。如果组件在调用 CAN_Sleep() 函数前已启用, 则 CAN_Wakeup() 函数还将重新启用组件。
- 参数:** None (无)
- Return Value (返回值):** None (无)
- Side Effects (副作用):** 调用 CAN_Wakeup() 函数前未调用 CAN_Sleep() 或 CAN_SaveConfig() 函数可能会产生不可预计行为。

uint8 CAN_Init(void)

- 说明:** 此函数根据自定义程序 Configure (配置) 对话框设置来初始化或恢复组件。无需调用 CAN_Init(), 因为 CAN_Start() 子程序会调用该函数并是开始组件操作的首选方法。
- 参数:** None (无)
- Return Value (返回值):** uint8: 说明是否已接受或拒绝配置
- Side Effects (副作用):** 所有寄存器将复位至初始值。这将重新初始化组件, 不同的是, 它不清除邮箱中的数据。
打开 CAN 内核的电源

uint8 CAN_Enable(void)

- 说明:** 此函数激活硬件并开始执行组件操作。无需调用 CAN_Enable(), 因为 CAN_Start() 子程序会调用该函数, 这是开始组件操作的首选方法。
- 参数:** None (无)
- Return Value (返回值):** None (无)
- Side Effects (副作用):** None (无)



void CAN_SaveConfig(void)

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

参数: None（无）

Return Value (返回值): None（无）

Side Effects (副作用): None（无）

void CAN_RestoreConfig(void)

说明: 此函数会恢复组件配置和非保留寄存器。它还将组件参数值恢复为在调用 **CAN_Sleep()** 函数之前的值。

参数: None（无）

Return Value (返回值): None（无）

Side Effects (副作用): 调用此函数前未调用 **CAN_Sleep()** 或 **CAN_SaveConfig()** 函数可能会产生不可预计行为。以下寄存器将恢复为默认值: **CAN_INT_SR**、**CAN_INT_EN**、**CAN_CMD** 和 **CAN_CFG**。

固件源代码示例

PSoC Creator 在“查找示例项目”对话框中提供了很多包括原理图和代码示例的示例项目。要获取组件特定的示例，请打开组件目录中的对话框或原理图中的组件实例。要获取通用的示例，请打开 **Start Page**（开始页）或 **File**（文件）菜单中的对话框。根据需要，使用对话框中的 **Filter Options**（过滤选项）可缩小可选项目的列表。

有关更多信息，请参见 PSoC Creator 帮助中的“Find Example Project（查找示例项目）”主题。

中断服务子程序

有多个 CAN 组件中断源：

- 仲裁丢失检测 - 发送消息时仲裁丢失。
- 过载错误 - CAN 控制器收到过载帧。
- 位错误 - CAN 控制器检测到位错误。
- 位填充错误 - CAN 控制器检测到位填充错误。

- 确认错误 - CAN 控制器检测到 CAN 消息确认错误。
- 格式错误 - CAN 控制器检测到 CAN 消息格式错误。
- CRC 错误 - CAN 控制器检测到 CAN CRC 错误。
- 总线关闭 - CAN 控制器达到总线关闭状态
- 消息丢失 - 新消息到达，但无处放置。
- 传输消息 - 队列中的消息被发送。
- 接收消息 - 接收到消息。

所有这些中断源都有入口点（函数），所以可将代码置于其中。这些函数是按照条件编译的，这取决于定制器。

接收消息中断有一个特殊处理程序，该程序针对完全和基本邮箱调用相应的函数。

中断输出使用案例

以下是 CAN 组件中硬件中断输出线路的示范用例。

中断事件上的逻辑的硬件控制

硬件中断线可用于执行简单的任务，例如评估 CAN 总线负载。通过在 CAN 组件定制器中启用“消息已传输”和“消息已接收”中断，并将中断线连接到计数器，可评估特定时间间隔中总线上的消息数。同时，如果消息速率超过特定值，可直接在硬件中采取措施。

中断输出与 DMA 的互动

CAN 组件内部不支持 DMA 操作，但您可将 DMA 组件连接到外部中断线（如果已启用）。您负责进行 DMA 配置和操作。同时，您应记住必须管理代码中的一些日常工作（例如，确认消息和清除中断标志），以正确处理 CAN 中断。

利用硬件 DMA 触发器，您可在发生消息接收中断时处理寄存器和数据传输，而无需 CPU 中的任何固件进行操作。处理 RTR 消息时此触发器也很有用。消息传输中断可用于触发 DMA 传输，以用新数据重新加载消息缓冲区，而无需 CPU 干预。

自定义外部中断服务子程序

自定义外部 ISR 可用于辅助或替代内部 ISR。当外部 ISR 用于辅助内部 ISR 时，可设置中断优先级以确定哪个 ISR 先执行（内部或外部），从而在内部 ISR 中编码的操作之前或之后执行操作。当外部 ISR 用于替代内部 ISR 时，您承担正确处理 CAN 寄存器和事件的所有责任。



中断输出与中断子系统的互动

CAN 组件中断输出设置允许您执行以下操作：

- 启用或禁用外部中断线（定制器选项）
- 禁用或旁路内部 ISR（定制器选项）
- 完全自定义内部 ISR（定制器选项）
- 启用或禁用当启用相关事件中断时，内部 ISR 中的特定中断处理函数调用（定制器选项）。可在 CAN 组件定制器中启用或禁用个别中断（消息已传输、消息已接收、接收缓冲区满、总线关闭状态等）。一旦启用，将在内部 CAN_ISR 中执行相关函数调用。这样，您可禁用（移除）此类函数调用。

只有在定制器中启用了外部中断线时，外部中断线才可见。

如果连接了外部中断组件，外部中断组件不作为 CAN_Start() API 的一部分启动，且必须在该子程序外启动。

如果连接了外部中断组件，且内部 ISR 未禁用或跳过，两个中断组件会连接到同一线路。在这种情况下，您有两个单独的中断组件处理同一中断事件。这是一个特定的情况，通常也是不受欢迎的情况。

如果内部 ISR 被禁用或跳过（使用定制器选项）时，内部中断组件将在构建过程中被移除。

若选择在内部中断子程序中禁用个别中断函数调用（针对已启用的中断事件，通过使用定制器选项），CAN 模块中断将触发（当相关事件发生时），但内部 CAN_ISR 子程序中不执行任何内部函数调用。一个示范用例是，当您要通过不同的途径而非标准用户函数调用（例如，通过 DMA）处理特定事件时。

如果选择完全自定义内部 ISR（使用自定义选项），CAN_ISR 函数将不包含除可选 PSoC 3 ES1/ES2 ISR 补丁之外的任何函数调用。

功能描述

有关完整的说明，请参见 *PSoC® 3 和 PSoC® 5 技术参考手册* 中的“控制器区域网络 (CAN)”章节。

框图和配置

有关框图和配置信息，请参见 *PSoC® 3 和 PSoC 5® 技术参考手册* 中的“控制器区域网络 (CAN)”章节。

参考

1. *ISO-11898: 公路车辆 -- 控制器区域网络 (CAN):*
 - ☐ 第 1 部分: 数据链路层和物理信令
 - ☐ 第 2 部分: 高速媒体访问单元
 - ☐ 第 3 部分: 低速、容错、媒体相关接口
 - ☐ 第 4 部分: 时间触发的通信
 - ☐ 第 5 部分: 高速媒体访问单元, 具有低功耗模式
2. *CAN 规范版本 2 BOSCH*
3. *Inicore CANmodule-III-AHB 数据表*

直流和交流电气特性

下面的值表示了预计性能, 它们基于初始特性数据。

CAN 直流规范

参数	Description (说明)	条件	最小值	典型值	最大值	单位
	模块电流消耗	500 kbps	--	--	285	μA
		1 Mbps	--	--	330	μA

CAN 交流规范

参数	Description (说明)	条件	最小值	典型值	最大值	单位
	比特率	最低 8 MHz 时钟	--	--	1	Mbit

组件更改

本节介绍组件与以前版本相比的主要更改。

当前版本	更改说明	更改/影响原因
2.1	已添加所有包括在 .cyre 文件中的带 CYREENTRANT 关键词的 CAN API。	并非所有 API 都是真正可重入的。组件 API 源文件中的注释指出了候选函数。 需要此更改为采用安全方式使用 (通过标志或关键节防止并发调用) 并且不是可重入函数消除编译器警告。



当前版本	更改说明	更改/影响原因
2.0.a	已替换时序图并向数据表添加了描述性文本	
	对数据表进行了少量编辑和更新	
2.0	已向组件符号添加了中断输出	已更新 PSoC 3 CAN 组件的营销要求文档 MRD
	已在组件符号中添加了 ConnectExtInterruptLine、IntISRDisable、FullCustomIntISR、AdvancedInterruptTab 参数和 ISR 助手参数	已更新 PSoC 3 CAN 组件的营销要求文档 MRD
	已在 CAN_Init() 函数开头添加了停止语句	要确保 CAN 在初始化时停止
	已更新了中断处理程序函数。在所有案例中，在用户代码之前清除中断标志。	要以同样的方式清除中断标志
	已移除过期定义	
1.50.a	向数据手册添加了特性数据	
	数据表文本编辑	
1.50	已添加睡眠/唤醒 API。	这些 API 支持低功耗模式。
	已添加 CAN_Init() 和 CAN_Enable() API。	为了符合公司标准，并提供 API 以便无需启动组件即可初始化/恢复组件。

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

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

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

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

产品使用可能受适用的赛普拉斯软件许可协议限制。

