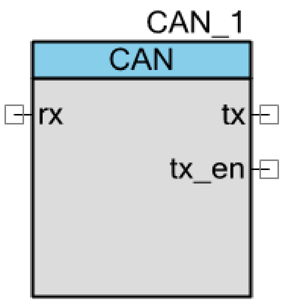
# Controller Area Network (CAN)

2.10

## Features



- CAN2.0A and CAN2.0B protocol implementation, ISO 11898-1 compliant
- Programmable bit rate up to 1 Mbps at 8 MHz (BUS_CLK)
- Two-wire or three-wire interface to external transceiver (Tx, Rx, and Enable)
- Extended hardware message filter that covers Data Byte 1 and Data Byte 2 fields
- Programmable transmit priority: Round Robin and Fixed
- CAN component fully supports PSoC 5LP device, but does not support PSoC 5

## General Description

The Controller Area Network (CAN) controller implements the CAN2.0A and CAN2.0B specifications as defined in the Bosch specification and conforms to the ISO-11898-1 standard.

### When to Use a CAN

The CAN protocol was originally designed for automotive applications with a focus on a high level of fault detection. This ensures high communication reliability at a low cost. Because of its success in automotive applications, CAN is used as a standard communication protocol for motion-oriented machine-control networks (CANOpen) and factory automation applications (DeviceNet). The CAN controller features allow you to efficiently implement higher-level protocols, without affecting the performance of the microcontroller CPU.

## Input/Output Connections

This section describes the various input and output connections for the CAN Component. An asterisk (*) in the list of I/Os indicates that the I/O may be hidden on the symbol under the conditions listed in the description of that I/O.

### rx – Input

CAN bus receive signal (connected to CAN Rx bus of external transceiver).

### tx– Output

CAN bus transmit signal, (connected to CAN Tx bus of external transceiver).

### tx_en – Output *

External transceiver enable signal. This output displays when the **Add Transceiver Enable Signal** option is selected in the **Configure** dialog.

### interrupt – Output *

The interrupt output is driven by the interrupt sources configured in the CAN hardware. All sources are ORed together to create the final output signal. The sources of the interrupt can be:

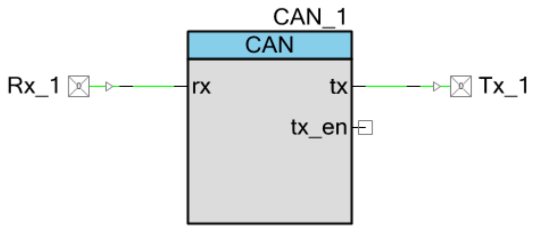
- Message Transmitted
- Message Received
- Receive Buffer Full
- Bus Off State
- CRC Error Detected
- Message Format Error Detected
- Message Acknowledge, Error Detected
- Bit Stuffing Error Detected
- Bit Error Detected
- Overload Frame Received
- Arbitration Lost Detected

This output displays when the **Enable External Interrupt Line** option is selected in the **Advanced Interrupt Configuration…** window of the **Interrupt** tab of the **Configure** dialog.

A failure caused by RX shorted to ground at time zero, before the CAN component is started, cannot be identified and reported to the higher level software by the CAN component. The CAN state machine does reach the idle state unless a falling edge is detected on RX.
It is the responsibility of the higher level software to determine a bus short at time zero prior to initialization of the CAN component.

## Schematic Macro Information

The default CAN in the Component Catalog is a schematic macro using a CAN component with default settings. The CAN component is connected to an Input and an Output Pins component. The Pins components are also configured with default settings, except that Input Synchronized is set to false in the Input Pin component.



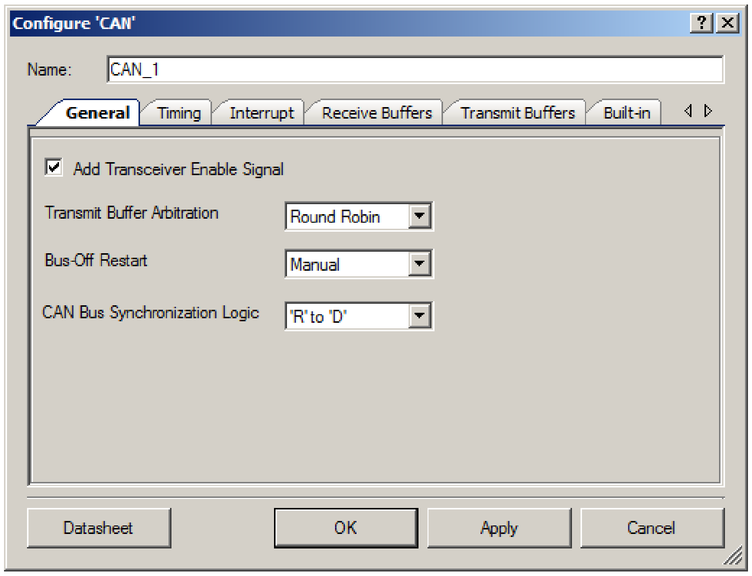
## Component Parameters

Drag a CAN component onto your design and double-click it to open the **Configure** dialog. This dialog has several tabs to guide you through the process of setting up the CAN component.

### Component Update Note

If updating the CAN component from a previous version, many of the parameters have been given a new format and must be converted. To do so, open the **Configure** dialog, change at least one parameter option, and click **OK** to save the change.

### General Tab



Name: CAN_1

☑ Add Transceiver Enable Signal

Transmit Buffer Arbitration: Round Robin

Bus-Off Restart: Manual

CAN Bus Synchronization Logic: 'R' to 'D'

The **General** tab contains the following settings:

### Add Transceiver Enable Signal

Enables or disables the use of the tx_en signal for the external CAN transceiver. Enabled by default.

### Transmit Buffer Arbitration

Defines the message transmission arbitration scheme:

- **Round Robin** (default) – Buffers are served in a defined order: 0-1-2 ... 7-0-1. A particular buffer is only selected if its TxReq flag is set. This scheme guarantees that all buffers receive the same probability to send a message.
- **Fixed priority** – Buffer 0 has the highest priority. This way it is possible to designate buffer 0 as the buffer for error messages and guarantee that they are sent first.

### Bus-Off Restart

Used to configure the reset type:

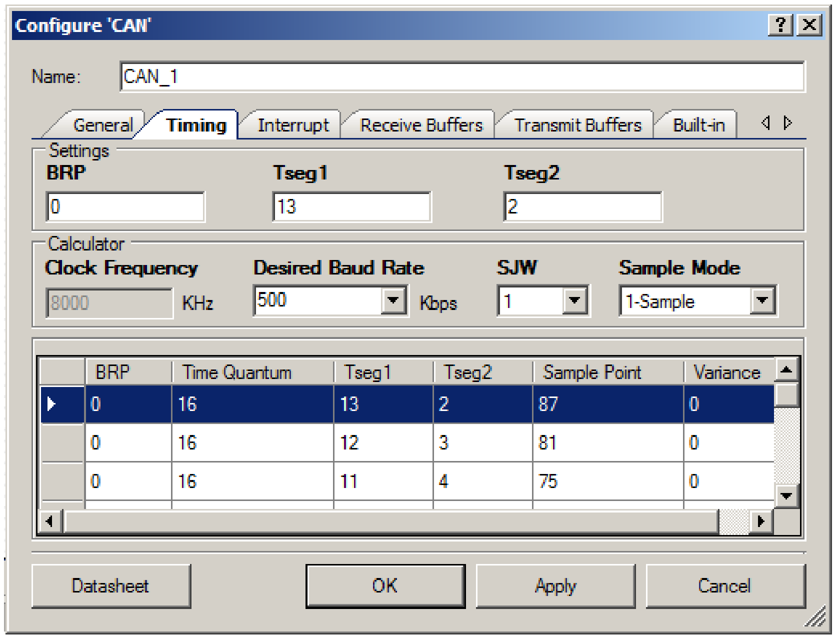
- **Manual** (default) – After the bus is turned off, you must restart the CAN. This is the recommended setting.
- **Automatic** – After the bus is turned off, the CAN controller restarts automatically after 128 groups of 11 recessive bits.

### CAN Bus Synchronization Logic

Used to configure edge synchronization:

- **'R' to 'D'** (default) – Edge from 'R'(recessive) to 'D'(dominant) is used for synchronization
- **Both edges** – Both edges are used for synchronization

## Timing Tab



Name: CAN_1

Settings

BRP: 0 Tseg1: 13 Tseg2: 2

Calculator

Clock Frequency: 8000 KHz Desired Baud Rate: 500 Kbps SJW: 1 Sample Mode: 1-Sample

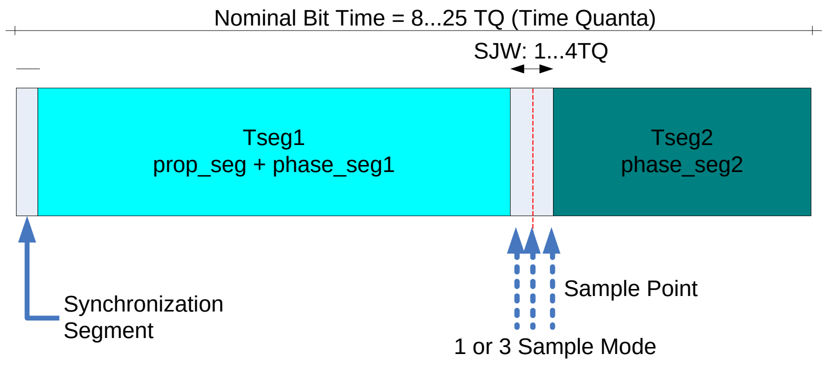
| BRP | Time Quantum | Tseg1 | Tseg2 | Sample Point | Variance |
|---|---|---|---|---|---|
| 0 | 16 | 13 | 2 | 87 | 0 |
| 0 | 16 | 12 | 3 | 81 | 0 |
| 0 | 16 | 11 | 4 | 75 | 0 |

The **Timing** tab contains the following settings:

### Settings

- **BRP** – Bit Rate Prescaler value for generating the time quantum. The bit timing calculator is used to calculate this value. 0 indicates 1 clock; 7FFFh indicates 32768 clock cycles, 15 bits.
- **Tseg1** – Value of time segment 1.
- **Tseg2** – Value of time segment 2. Values 0 and 1 are not allowed; Value 2 is only allowed when **Sample Mode** is set to direct sampling (**1-Sample**).

  The following shows the CAN bit timing representation:



### Calculator

- **Clock Frequency** (in MHz) – The system clock frequency equal to BUS_CLK.
- **Desired Baud Rate** (in Kbps) – Options are: **10**, **20**, **62.5**, **125**, **250**, **500**, **800**, or **1000**.
- **SJW** – Configuration of synchronization jump width (2 bits). The value must be less than or equal to Tseg1 and less than or equal to Tseg2. Options are: **1**, **2**, **3**, or **4**.
- **Sample Mode** – Configuration of sampling mode. Options are: **1-Sample** or **3-Sample**.

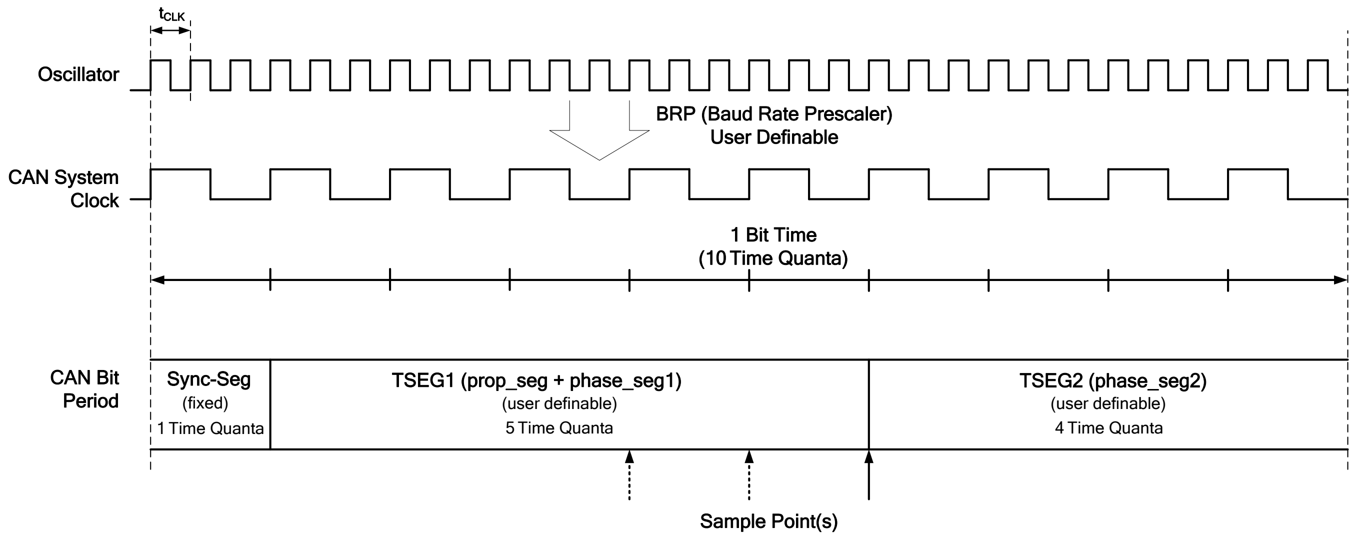
### Table

Bit timing is calculated, and the proposed register settings for time segments (Tseg1 and Tseg2) and BRP are displayed in the parameter table. You can select the values to be loaded by double-clicking the appropriate row. Selected values are displayed in the top **Settings** input boxes.

You may also choose to manually enter values for Tseg1, Tseg2, and BRP in the provided input boxes.

**Note** Incorrect bit timing settings might cause the CAN controller to remain in an error state.

The following diagram shows an example of how all timing is derived from the oscillator.



## Bit Time Segments

### SYNC SEG (Synchronization Segment)

This part of the bit time is used to synchronize the various nodes on the bus. An edge is expected to lie within this segment.

### PROP SEG (Propagation Time Segment)

This part of the bit time is used to compensate for the physical delay times within the network. It is twice the sum of the signal's propagation time on the bus line, the input comparator delay, and the output driver delay.

### PHASE SEG1, PHASE SEG2 (Phase Buffer Segment1/2)

These phase-buffer segments are used to compensate for edge phase errors. These segments can be lengthened or shortened by resynchronization.

### Sample Point

The sample point is the point in time at which the bus level is read and interpreted as the value of that respective bit. It is located at the end of PHASE_SEG1.

### Information Processing Time

The information processing time is the time segment starting with the sample point reserved for calculating the subsequent bit level.

### Time Quantum

The time quantum is a fixed unit of time derived from the oscillator period. There is a programmable prescaler (BRP), with integral values, ranging from 1 to 32768. Starting with the minimum time quantum, the time quantum can have a length of

TIME QUANTUM = m × MINIMUM TIME QUANTUM,

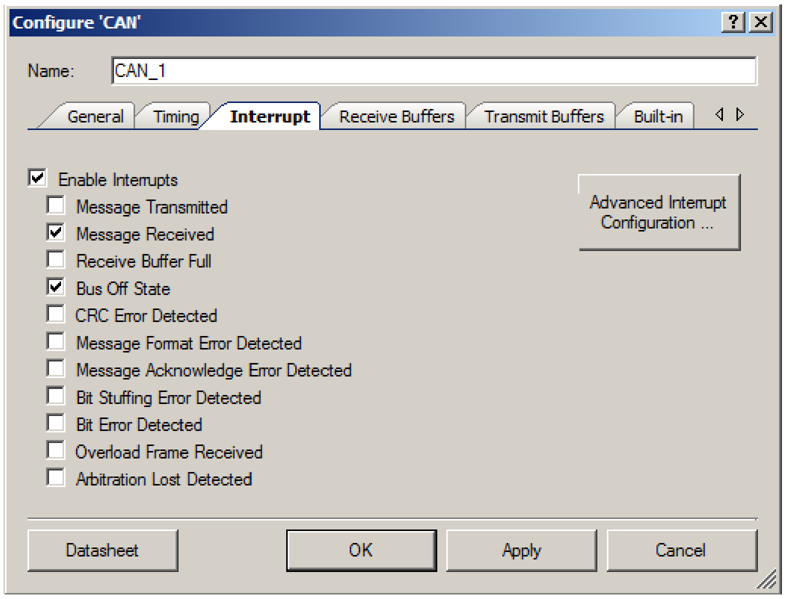
where m is the value of the prescaler.

### Length of Time Segments

- SYNC_SEG is 1 time quantum long.
- PROP_SEG is programmable to be 1, 2, …, 8 time quanta long.
- PHASE_SEG1 is programmable to be 1, 2, ..., 8 time quanta long.
- PHASE_SEG2 is the maximum of PHASE_SEG1 and the information processing time

## Interrupt Tab

### Basic Interrupt Configuration



Name: CAN_1

- [✓] Enable Interrupts
  - [ ] Message Transmitted
  - [✓] Message Received
  - [ ] Receive Buffer Full
  - [✓] Bus Off State
  - [ ] CRC Error Detected
  - [ ] Message Format Error Detected
  - [ ] Message Acknowledge Error Detected
  - [ ] Bit Stuffing Error Detected
  - [ ] Bit Error Detected
  - [ ] Overload Frame Received
  - [ ] Arbitration Lost Detected

The **Basic Interrupt Configuration** tab contains the following settings:

#### Enable Interrupts

Enable or disable global interrupts from the CAN Controller. Enabled by default.

**Enabled** – Global interrupts are enabled when the CAN component is started using CAN_1_Start().

**Disabled** – Global interrupts are not enabled when the CAN component is started using CAN_1_Start(). The CAN ISR is not entered until the global interrupt enable bit is set. It is your responsibility to enable or disable global interrupts in main code, using CAN_1_GlobalIntEnable() or CAN_1_GlobalIntDisable().

#### Message Transmitted

Enable or disable message transmitted interrupts. Disabled by default. Indicates that a message was sent. When disabling the Message Transmitted interrupt, the CAN displays the following message: **Do you wish to disable all Transmit Buffers Interrupts**?

- **Yes** – Uncheck the **Message Transmitted** check box, and uncheck all individual transmit buffer interrupts on the **Transmit Buffers** tab.
- **No** (default) – Uncheck the **Message Transmitted** check box, and keep all individual transmit buffer interrupts on the **Transmit Buffers** tab as they are.
- **Cancel** – No changes are made.

### Message Received

Enable or disable message received interrupts. Enabled by default. Indicates that a message was received. When disabling the Message Received interrupt, the CAN displays the following message: **Do you wish to disable all Receive Buffers Interrupts?**

- **Yes** (default) – Uncheck the **Message Received** check box, and uncheck all individual receive buffer interrupts on the **Receive Buffers** tab.
- **No** – Uncheck the **Message Received** check box, and keep all individual receive buffer interrupts on the **Receive Buffers** tab as they are.
- **Cancel** – No changes are made.

### Receive Buffer Full

Enable or disable message lost interrupt. Indicates that a new message was received when the previous message was not acknowledged. Disabled by default.

### Bus Off State

Enable or disable Bus Off interrupt. Indicates that the CAN node has reached the Bus Off state. Enabled by default.

### CRC Error Detected

Enable or Disable CRC error interrupt. Indicates that a CAN CRC error was detected. Disabled by default.

### Message Format Error Detected

Enable or disable message format error interrupt. Indicates that a CAN message format error was detected. Disabled by default.

### Message Acknowledge Error Detected

Enable or disable message acknowledge error interrupt. Indicates that a CAN message acknowledge error was detected. Disabled by default.

### Bit Stuffing Error Detected

Enable or disable bit stuffing error interrupt. Indicates that a bit stuffing error was detected. Disabled by default.

### Bit Error Detected

Enable or disable bit error interrupt. Indicates that a bit error was detected. Disabled by default.

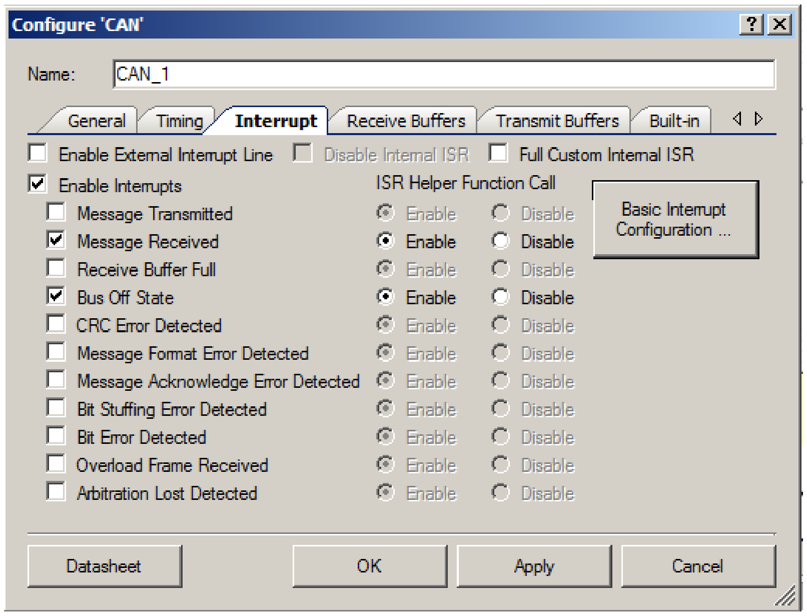
## Overload Frame Received

Enable or disable overload interrupt. Indicates that an overload frame was received. Disabled by default.

## Arbitration Lost Detected

Enable or disable managing arbitration and cancellation of queued messages. Indicates that the arbitration was lost while sending a message. Disabled by default.

## Advanced Interrupt Configuration



Name: CAN_1

☐ Enable External Interrupt Line ☐ Disable Internal ISR ☐ Full Custom Internal ISR

☑ Enable Interrupts

| | ISR Helper Function Call | |
|---|---|---|
| ☐ Message Transmitted | ● Enable | ○ Disable |
| ☑ Message Received | ● Enable | ○ Disable |
| ☐ Receive Buffer Full | ● Enable | ○ Disable |
| ☑ Bus Off State | ● Enable | ○ Disable |
| ☐ CRC Error Detected | ● Enable | ○ Disable |
| ☐ Message Format Error Detected | ● Enable | ○ Disable |
| ☐ Message Acknowledge Error Detected | ● Enable | ○ Disable |
| ☐ Bit Stuffing Error Detected | ● Enable | ○ Disable |
| ☐ Bit Error Detected | ● Enable | ○ Disable |
| ☐ Overload Frame Received | ● Enable | ○ Disable |
| ☐ Arbitration Lost Detected | ● Enable | ○ Disable |

The **Advanced Interrupt Configuration** tab contains the following settings:

## Enable External Interrupt Line

Enable external visibility and connectivity of the CAN block interrupt line. Default is cleared (external interrupt line not visible in the CAN component symbol instance).

## Disable Internal ISR

Disable or bypass internal ISR component. If the internal ISR is disabled, the relevant CAN APIs do not handle the ISR start/stop processes. Default is cleared (internal ISR is enabled). The check box is available (not grayed out) only if **Enable External Interrupt Line** is selected. You can disable the internal ISR only if there is an alternate provision (external interrupt line) to handle interrupts.

### Full Custom Internal ISR

Enable the use of the internal ISR with fully custom code. When this option is selected, the CAN_1_ISR contains no code. Put your custom code between the lines:

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

/* `#END` */
```

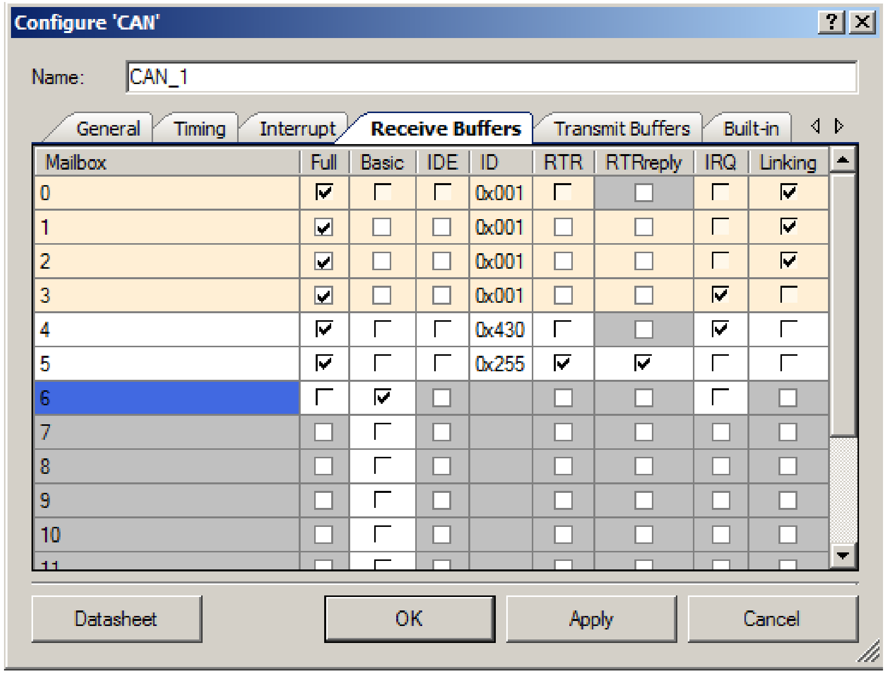
Default is unselected (default to CAN v1.50 ISR handling). The check box is available (not grayed out) only if **Disable Internal ISR** is not selected.

### ISR Helper Function Call

If only basic interrupt settings are used (for example, as in CAN v1.50), when an interrupt occurs, the CAN ISR calls relevant user-customizable functions (ISR helpers) based on the enabled interrupts.

These options give you the opportunity to enable or disable ISR helper calls, so that custom handling of specific interrupts can be implemented both in hardware and firmware. Default is **Enable**. These options are available (not grayed out) if the relevant interrupt event is enabled AND **Full Custom Internal ISR** is not checked AND **Disable Internal ISR** is not checked.

## Receive Buffers Tab



Name: CAN_1

| 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 | ☐ | ☐ | ☐ |  | ☐ | ☐ | ☐ | ☐ |

The **Receive Buffers** tab contains the following settings:

### Mailbox

A receive mailbox is disabled until **Full** or **Basic** is selected. The **IDE**, **ID**, **RTR**, **RTRreply** and **IRQ** fields are locked for all disabled mailboxes.

For Full mailboxes, the **Mailbox** field is editable to enter a unique message name. The API provided for handling each mailbox will have the mailbox string appended. Accepted symbols are: A–Z, a–z, 0–9, and _. If you enter an incorrect name, an error message displays and the **Mailbox** field returns to the default value.

### Full

When **Full** is selected, you can modify the **Mailbox**, **IDE**, **ID**, **RTR**, **RTRreply**, **IRQ** and **Linking** fields. Default selections are placed with the following options:

- **Mailbox** = Mailbox number 0 to 15
- **IDE** = Cleared
- **ID** = 0x001
- **RTR** = Cleared
- **RTRreply** = Cleared and locked (only enable when **RTR** is selected)

- **IRQ** = Selected, only available if **Message Received** (**Interrupt** tab) interrupt is selected
- **Linking** = Cleared

### Basic

If **Basic** is selected, the options **IDE**, **ID**, **RTR**, **RTRreply** are unavailable. Default selections are placed with the following options:

- **IDE** = Cleared (unavailable)
- **ID** = <All> (unavailable)
- **RTR** = Cleared (unavailable)
- **RTRreply** = Cleared (unavailable)
- **IRQ** = Cleared, only available if **Message Received** interrupt is selected
- **Linking** = Cleared

### IDE

When the **IDE** box is cleared the identifier is limited to 11 bits (0x001 to 0x7FE). When **IDE** is selected the identifier is limited to 29 bits (0x00000001 to 0x1FFFFFFE).

### RTR - Remote Transmission Request

Only available for mailboxes set up to receive Full CAN messages. When selected, it configures the acceptance filter settings to only allow receipt of messages whose RTR bit is set.

### RTRreply - Remote Transmission Request Auto Reply

Only available for mailboxes set up to receive Full CAN messages, with the RTR bit set. When checked, it automatically replies to an RTR request with the content of the receive buffer.

### IRQ

When enabling the IRQ for a mailbox, if the **Message Received Interrupt** in the **Interrupt** tab is cleared, the following message is displayed: **Global "Message Received Interrupt" is disabled. Do you wish to enable it?**

- **Yes** – Select the **IRQ** check box and select the **Message Received Interrupt** check box on the **Interrupt** tab.
- **No** or **Cancel** – Select the **IRQ** check box and leave the **Message Received Interrupt** check box in the **Interrupt** tab as is.

### Linking

The **Linking** check box allows the linking of several sequential receive mailboxes to create an array of receive mailboxes. This array acts like a receive FIFO. All mailboxes of the same array must have the same message filter settings; that is, the acceptance mask register (AMR) and acceptance code register (ACR) are identical.

- The last mailbox of an array may not have its linking flag set.
- The last mailbox 15 cannot have its linking flag set.
- All linked mailboxes are highlighted with the same color.
- Only the first mailbox in the linked array is editable. All parameters are automatically applied to all linked mailboxes within the same array.
- One function is generated for all linked mailboxes.

### Receive Message Functions

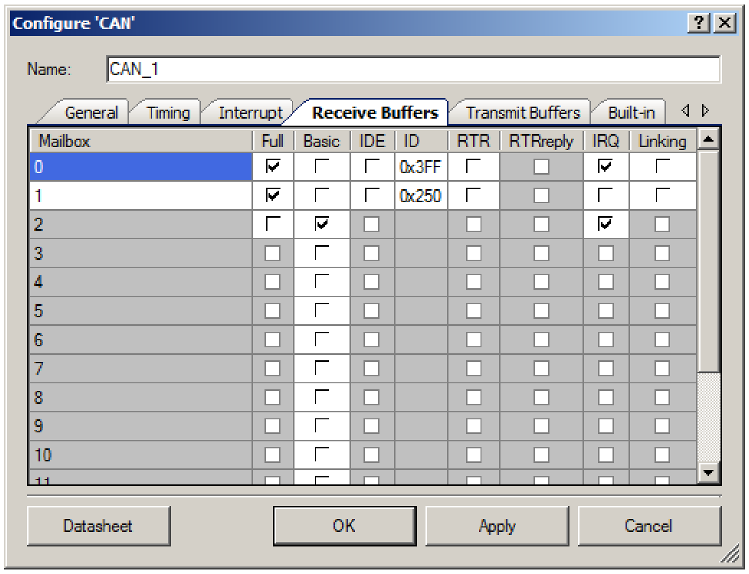
Every Full RX mailbox has a predefined API. The function list is available in the *CAN_1_TX_RX_func.c* project file. These functions are conditionally compiled depending on the receive mailbox setting. Only mailboxes defined as Full have their respective functions compiled.

The macro identifier CAN_1_RX*x*_FUNC_ENABLE defines whether a function is compiled. Defines are listed in the *CAN_1.h* project file.

- When a message received interrupt occurs, the CAN_1_MsgRXIsr() function is called. This function loops through all receive mailboxes and checks their respective “Message Available Flag” (MsgAv – Read: 0 No new message available; 1 New message available) and “Interrupt Enable” (Receive Interrupt Enable: 0 Interrupt generation is disabled, 1 Interrupt generation is enabled) for successful receipt of a CAN message.
- If the **Message Receive** interrupt is enabled, then when a message is received the CAN_1_ReceiveMsg*X()* function is called, where *X* indicates the Full CAN mailbox number or user-defined name.
- For all interrupt-based Basic CAN mailboxes, the CAN_1_ReceiveMsg(uint8 rxMailbox) function is called, where the rxMailbox parameter indicates the number of the mailbox that received the message.

## Receive Buffers Configuration

The following is an example to illustrate the use of the receive message APIs.



Name: CAN_1

| Mailbox | Full | Basic | IDE | ID | RTR | RTRreply | IRQ | Linking |
|---|---|---|---|---|---|---|---|---|
| 0 | ☑ | ☐ | ☐ | 0x3FF | ☐ | ☐ | ☑ | ☐ |
| 1 | ☑ | ☐ | ☐ | 0x250 | ☐ | ☐ | ☐ | ☐ |
| 2 | ☐ | ☑ | ☐ | | ☐ | ☐ | ☑ | ☐ |
| 3 | ☐ | ☐ | ☐ | | ☐ | ☐ | ☐ | ☐ |
| 4 | ☐ | ☐ | ☐ | | ☐ | ☐ | ☐ | ☐ |
| 5 | ☐ | ☐ | ☐ | | ☐ | ☐ | ☐ | ☐ |
| 6 | ☐ | ☐ | ☐ | | ☐ | ☐ | ☐ | ☐ |
| 7 | ☐ | ☐ | ☐ | | ☐ | ☐ | ☐ | ☐ |
| 8 | ☐ | ☐ | ☐ | | ☐ | ☐ | ☐ | ☐ |
| 9 | ☐ | ☐ | ☐ | | ☐ | ☐ | ☐ | ☐ |
| 10 | ☐ | ☐ | ☐ | | ☐ | ☐ | ☐ | ☐ |

### *CAN_1.h* file:

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

### *CAN_1_TX_RX_func.c* file:

```
#if (CAN_1_RX0_FUNC_ENABLE)

    /* ... */
    void CAN_1_ReceiveMsg0(void)
    {
        /* `#START MESSAGE_0_RECEIVED` */

        /* `#END` */

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

#endif /* CAN_1_RX0_FUNC_ENABLE */


#if (CAN_1_RX1_FUNC_ENABLE)

    /* ... */
    void CAN_1_ReceiveMsg1(void)
    {
        /* `#START MESSAGE_1_RECEIVED` */
```

```
            /* `#END` */

            CAN_1_RX[1u].rxcmd.byte[0u] |= CAN_1_RX_ACK_MSG;
        }

   #endif /* CAN_1_RX1_FUNC_ENABLE */
```

The following function will be called for CAN Receive Message 2, configured as Basic CAN with interrupt enabled.

```
void CAN_1_ReceiveMsg(uint8 rxMailbox)
{
    if ((CAN_1_RX[rxMailbox].rxcmd.byte[0u] & CAN_1_RX_ACK_MSG) == CAN_1_RX_ACK_MSG)
    {
        /* `#START MESSAGE_BASIC_RECEIVED` */

        /* `#END` */

        CAN_1_RX[rxMailbox].rxcmd.byte[0u] |= CAN_1_RX_ACK_MSG;
    }
}
```

### *CAN_1_INT.c* file

```
   void CAN_1_MsgRXIsr(void)
   {
   ...
       /* RX Full mailboxes handler */
       switch(i)
       {
           case 0 : CAN_1_ReceiveMsg0();
           break;
           case 1 : CAN_1_ReceiveMsg1();
           break;
           default:
           break;
       }
   ...
   }
```

If Linking is implemented, conditional compile applies to the mailbox with the IRQ flag set. All receive functions acknowledge message receipt by clearing the Message Available (MsgAv) flag.

## How to Set AMR and ACR to Accept Range of IDs

The following is the ACR/AMR register representation:

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

The acceptance mask register (AMR) defines whether the incoming bit is checked against acceptance code register (ACR).

|  |  |  |
|---|---|---|
| AMR: | ‘0’ | The incoming bit is checked against the respective ACR. The message is not accepted when the incoming bit doesn’t match the respective ACR bit. |
|  | ‘1’ | The incoming bit is doesn’t care. |

For example, to set up the mailbox to receive a range of IDs of 0x180–187, IDE = 0 (cleared), RTR = 0 (cleared), mailbox 5, perform the following additional actions:

Take the low range of ID, for instance 0x180, and IDE and RTR accordingly. For this ID, IDE, and RTR values, AMR and ACR registers are set to values:

|  |  |
|---|---|
| ■ ACR[31:21] = 0x180 | AMR[31:21] = 0x0 |
| ■ ACR[20:3] = 0x0 (don't care) | AMR[20:3] = 0x3FFFF (all ones) |
| ■ ACR[2] = 0 | AMR[2] = 0 |
| ■ ACR[1] = 0 | AMR[1] = 0 |
| ■ ACR[0] = 0 | AMR[0] = 0 |

Define common part of range ID (11bit):

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

You must put 1s instead of “XXX” and nulls instead of common bits into the AMR register. So, the next values are:

|  |  |
|---|---|
| ■ ACR[31:21] = 0x180 | AMR[31:21] = 0x7 |
| ■ ACR[20:3] = 0x0 (don't care) | AMR[20:3] = 0x3FFFF (all ones) |
| ■ ACR[2] = 0 | AMR[2] = 0 |
| ■ ACR[1] = 0 | AMR[1] = 0 |
| ■ ACR[0] = 0 | AMR[0] = 0 |

Use the CAN_1_RXRegisterInit() function to write AMR register of mailbox number 5:

```
uint8 result = CAN_1_FAIL;
uint32 temp_amr;
uint32 temp_acr;

/* Upper address value, so address is shifted */
temp_amr = ((uint32)0x7u << 21u) | ((uint32)0x3FFFFu << 3u);    /* obtain necessary
value to put in AMR */
```

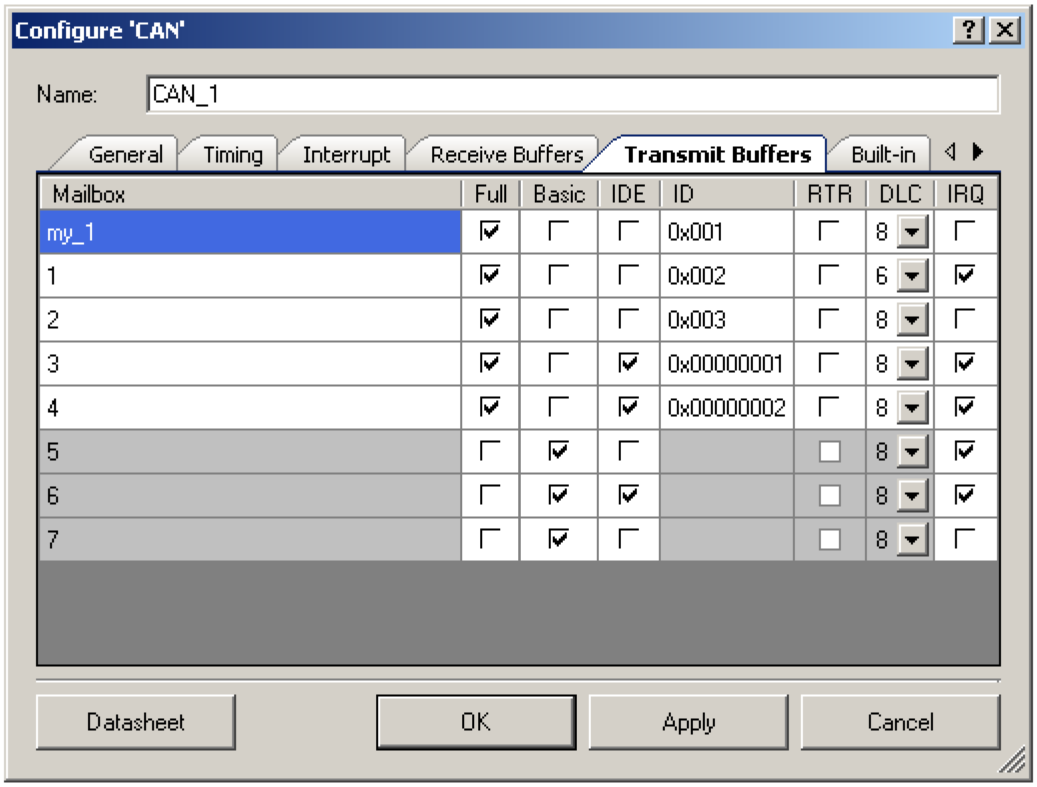
```
temp_acr = ((uint32)0x180u << 21u) | ((uint32)0x3FFFFu << 3u);    /* obtain necessary
value to put in ACR */

if (CAN_1_RXRegisterInit((reg32 *)&CAN_1_RX[5].rxamr, temp_amr) == CYRET_SUCCESS)
{
    if (CAN_1_RXRegisterInit((reg32 *)&CAN_1_RX[5].rxacr, temp_acr) == CYRET_SUCCESS)
    {
        result = CYRET_SUCCESS;
    }
}

if (result == CAN_1_FAIL)
{
    /* error */
}
```

For additional details on AMR and ACR configuration, refer to the Controller Area Network (CAN) chapter in the *PSoC® 3 and PSoC 5® Technical Reference Manual*.

## Transmit Buffers Tab



| Mailbox | Full | Basic | IDE | ID | RTR | DLC | IRQ |
|---|---|---|---|---|---|---|---|
| my_1 | ✓ | | | 0x001 | | 8 | |
| 1 | ✓ | | | 0x002 | | 6 | ✓ |
| 2 | ✓ | | | 0x003 | | 8 | |
| 3 | ✓ | | ✓ | 0x00000001 | | 8 | ✓ |
| 4 | ✓ | | ✓ | 0x00000002 | | 8 | ✓ |
| 5 | | ✓ | | | | 8 | ✓ |
| 6 | | ✓ | ✓ | | | 8 | ✓ |
| 7 | | ✓ | | | | 8 | |

The **Transmit Buffers** tab contains the following settings:

### Mailbox

For Full mailboxes, the **Mailbox** field is editable and you can enter a unique name for a mailbox. The function for handling this mailbox will also have a unique name. The accepted characters are: A–Z, a–z, 0–9, and _. If you enter an incorrect name the **Mailbox** field, it reverts to the default value.

### Full

When **Full** is selected, you can modify the **Mailbox**, **IDE**, **ID**, **RTR**, **RTRreply**, **IRQ**, and **Linking** fields. Default selections are placed with the following options:

- **Mailbox** = Number 0 to 7
- **IDE** = Cleared
- **ID** = 0x01
- **RTR** = Cleared
- **DLC** = 8
- **IRQ** = Cleared

### Basic

By default, the **Basic** check box is selected for all of the mailboxes. If **Basic** is selected, the options **ID**, **RTR**, and **DLC** are unavailable. If **Basic** is selected, the required CAN message fields should be entered using code. Default selections are placed with the following options:

- **IDE** = Cleared
- **ID** = Nothing (unavailable)
- **RTR** = Cleared (unavailable)
- **DLC** = 8 (unavailable)
- **IRQ** = Cleared

### IDE

If the IDE check box is cleared, the identifier cannot be greater than 11 bits (from 0x001 to 0x7FE). If IDE is selected, a 29-bit identifier is allowed (from 0x00000001 to 0x1FFFFFFE). You cannot choose identifiers of 0x000 or 0x7FF (11-bit) or 0x1FFFFFFF (29 bit).

### ID

The message identifier.

### RTR

The message is a Return Transmission Request Message.

### DLC

The number of bytes the message contains.

### IRQ

The IRQ bit depends on **Message Transmitted** (**Interrupt** tab).

If the **Message Transmitted** check box is cleared, when selecting the **IRQ**, the message appears: **Global "Message Transmitted Interrupt" is disabled. Do you wish to enable it?**

- **Yes** – Select **IRQ** and the **Message Transmitted Interrupt** check box in the **Interrupt** tab.
- **No** or **Cancel** – Select **IRQ**. The **Message Transmitted Interrupt** check box in the **Interrupt** tab remains cleared.

### CAN TX Functions

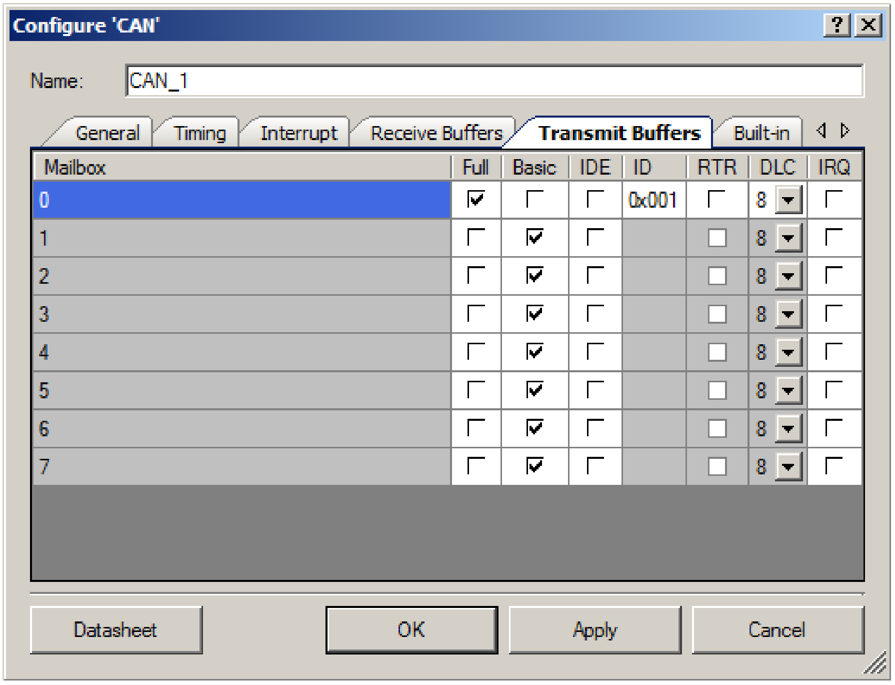
Every Full TX mailbox has a predefined API. The function list is available in the *CAN_1_TX_RX_func.c* project file. These functions are conditionally compiled depending on the transmit mailbox setting. Only mailboxes defined as Full will have their respective functions compiled.

The macro identifier CAN_1_TX*x*_FUNC_ENABLE defines whether a function is compiled. Defines are listed in the *CAN_1.h* project file.

The CAN_1_SendMsg*X*() function is provided for all Tx Mailboxes configured as Full, where *X* indicates the Full CAN mailbox number or user-defined name.

### Transmit Buffers Configuration

The following is an example to illustrate the use of the transmit APIs.



Name: CAN_1

| 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 | |

## *CAN_1.h* file

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

## *CAN_1_TX_RX_func.c* file

```
#if (CAN_1_TX0_FUNC_ENABLE)

    /* ... */
    uint8 CAN_1_SendMsg0(void)
    {
        uint8 result = CYRET_SUCCESS;

        if ((CAN_1_TX[0u].txcmd.byte[0u] & CAN_1_TX_REQUEST_PENDING) ==
            CAN_1_TX_REQUEST_PENDING)
        {
            result = CAN_1_FAIL;
        }
        else
        {
            /* `#START MESSAGE_0_TRASMITTED` */

            /* `#END` */

            CY_SET_REG32((reg32 *) &CAN_1_TX[0u].txcmd, CAN_1_SEND_MESSAGE);
        }

        return(result);
    }

#endif /* CAN_1_TX0_FUNC_ENABLE */
```

The common function provided for all Basic Transmit mailboxes:

```
uint8 CAN_1_SendMsg(CAN_1_TX_MSG *message)
```

A generic structure is defined for the application used to assemble the required data for a CAN transmit message:

- ID – the restriction if the ID slot includes:
    - ❑ For a standard message (IDE = 0) identifier limited to 11 bits (0x001 to 0x7FE).
    - ❑ For an extended message (IDE = 1) identifier limited to 29 bits (0x00000001 to 0x1FFFFFFE)
- RTR (0 – Standard message, 1 – 0xFF: RTR bit set in the message)
- IDE (0 – Standard message, 1 – 0xFF: Extended message)
- DLC (Defines number of data bytes 0 to 8, 9 to 0xFF equal 8 data bytes)
- IRQ (0 – IRQ Enable, 1 – 0xFF: IRQ Disable)
- DATA_BYTES (Pointer to structure of 8 bytes that represent transmit data)

When called, the CAN_1_SendMsg() function loops through the transmit message mailboxes that are designated as Basic CAN mailboxes and looks for the first available mailbox:

- When a free Basic CAN mailbox is found, the data passed through the CAN_1_TX_MSG structure is copied to the appropriate CAN transmit mailbox. When the message is put into transmit queue, an indication of “SUCCESS” is returned to the application.
- When no free Basic mailbox is found, the function tries again for a limited number of retries (up to three). When all retries fail, an indication of “FAIL” is returned to the application.

The CAN_1_TX_MSG structure contains all information required to transmit a message:

```
/* Stuct for BASIC CAN mailbox to send messages */
typedef struct _CAN_1_txMsg
{
    uint32 id;
    uint8 rtr;
    uint8 ide;
    uint8 dlc;
    uint8 irq;
    CAN_1_DATA_BYTES_MSG *msg;
} CAN_1_TX_MSG;
```

The CAN_1_DATA_BYTES structure contains eight bytes of data in a message.

```
/* Stuct for DATA of BASIC CAN mailbox */
typedef struct _CAN_1_dataBytesMsg
{
    uint8 byte[8u];
} CAN_1_DATA_BYTES_MSG;
```

## Clock Selection

The CAN component is connected to the BUS_CLK clock signal. A minimum value of 8 MHz is required to support all standard CAN baud rates up to 1 Mbps.

## Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. The following table lists and describes the interface to each function. The subsequent sections cover each function in more detail.

By default, PSoC Creator assigns the instance name "CAN_1" to the first instance of a component in a given design. You can rename the instance to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is "CAN."

| Function | Description |
|---|---|
| CAN_Start() | Sets the initVar variable, calls the CAN_Init() function, and then calls the CAN_Enable() function. |
| CAN_Stop() | Disables the CAN. |
| CAN_GlobalIntEnable() | Enables global interrupts from CAN component. |
| CAN_GlobalIntDisable() | Disables global interrupts from CAN component. |
| CAN_SetPreScaler() | Sets prescaler for generation of the time quanta from the BUS_CLK. |
| CAN_SetArbiter() | Sets arbitration type for transmit buffers |
| CAN_SetTsegSample() | Configures: Time segment 1, Time segment 2, Synchronization Jump Width, and Sampling Mode. |
| CAN_SetRestartType() | Sets reset type. |
| CAN_SetEdgeMode() | Sets Edge mode. |
| CAN_RXRegisterInit() | Writes only receive CAN registers. |
| CAN_SetOpMode() | Sets Operation mode. |
| CAN_GetTXErrorflag() | Returns the flag that indicates if the number of transmit errors exceeds 0x60. |
| CAN_GetRXErrorflag() | Returns the flag that indicates if the number of receive errors exceeds 0x60. |
| CAN_GetTXErrorCount() | Returns the number of transmit errors. |
| CAN_GetRXErrorCount() | Returns the number of receive errors. |
| CAN_GetErrorState() | Returns error status of the CAN component. |
| CAN_SetIrqMask() | Sets to enable or disable particular interrupt sources. |
| CAN_ArbLostIsr() | Clears Arbitration Lost interrupt flag. |

| Function | Description |
|---|---|
| CAN_OvrLdErrrorIsr() | Clears Overload Error interrupt flag. |
| CAN_BitErrorIsr() | Clears Bit Error interrupt flag. |
| CAN_BitStuffErrorIsr() | Clears Bit Stuff Error interrupt flag. |
| CAN_AckErrorIsr() | Clears Acknowledge Error interrupt flag. |
| CAN_MsgErrorIsr() | Clears Form Error interrupt flag. |
| CAN_CrcErrorIsr() | Clears CRC Error interrupt flag. |
| CAN_BusOffIsr() | Clears Bus Off interrupt flag. Places CAN Component to Stop mode. |
| CAN_MsgLostIsr() | Clears Message Lost interrupt flag. |
| CAN_MsgTXIsr() | Clears Transmit Message interrupt flag. |
| CAN_MsgRXIsr() | Clears Receive Message interrupt flag and call appropriate handlers for Basic and Full interrupt based mailboxes. |
| CAN_RxBufConfig() | Configures all receive registers for particular mailbox. |
| CAN_TxBufConfig() | Configures all transmit registers for particular mailbox. |
| CAN_SendMsg() | Sends an message from one of the Basic mailboxes. |
| CAN_SendMsg0-7() | Checks if mailbox 0-7 has untransmitted messages waiting for arbitration. |
| CAN_TxCancel() | Cancels transmission of a message that has been queued for transmission. |
| CAN_ReceiveMsg0-15() | Acknowledges receipt of new message. |
| CAN_ReceiveMsg() | Clears Receive particular Message interrupt flag. |
| CAN_Sleep() | Prepares CAN component to go to sleep |
| CAN_Wakeup() | Prepares CAN component to wake up |
| CAN_Init() | Initializes or restores the CAN per the Configure dialog settings. |
| CAN_Enable() | Enables the CAN. |
| CAN_SaveConfig() | Saves the current configuration. |
| CAN_RestoreConfig() | Restores the configuration. |

For functions that return indication of execution: 0 is “SUCCESS,” 1 is “FAIL,” and 2 is “OUT_OF_RANGE.”

## Global Variables

| Variable | Description |
|---|---|
| CAN_initVar | Indicates whether the CAN has been initialized. The variable is initialized to 0 and set to 1 the first time CAN_Start() is called. This allows the component to restart without re-initialization after the first call to the CAN_Start() routine.<br>If re-initialization of the component is required, then the CAN_Init() function can be called before the CAN_Start() or CAN_Enable() function. |

## uint8 CAN_Start(void)

**Description:** Sets the initVar variable, calls the CAN_Init() function, and then calls the CAN_Enable() function. This function sets the CAN component into run mode and starts the counter if polling mailboxes available.

**Parameters:** None

**Return Value:** uint8: Indication whether register is written and verified

**Side Effects:** If the initVar variable is already set, this function only calls the CAN_Enable() function.

## uint8 CAN_Stop(void)

**Description:** This function sets the CAN component into Stop mode and stops the counter if polling mailboxes available.

**Parameters:** None

**Return Value:** uint8: Indication whether register is written and verified

**Side Effects:** None

## uint8 CAN_GlobalIntEnable(void)

**Description:** This function enables global interrupts from the CAN component.

**Parameters:** None

**Return Value:** uint8: Indication whether register is written and verified

**Side Effects:** None

## uint8 CAN_GlobalIntDisable(void)

| | |
|---|---|
| **Description:** | This function disables global interrupts from the CAN component. |
| **Parameters:** | None |
| **Return Value:** | uint8: Indication whether register is written and verified |
| **Side Effects:** | None |

## uint8 CAN_SetPreScaler(uint16 bitrate)

| | |
|---|---|
| **Description:** | This function sets the prescaler for generation of the time quanta from the BUS_CLK. Values between 0x0 and 0x7FFF are valid. |
| **Parameters:** | uint16 bitrate: PreScaler value |
| **Return Value:** | uint8: Indication whether register is written and verified |
| **Side Effects:** | None |

## uint8 CAN_SetArbiter(uint8 arbiter)

| | |
|---|---|
| **Description:** | This function sets the arbitration type for transmit buffers. Types of arbiters are Round Robin and Fixed priority. Values 0 and 1 are valid. |
| **Parameters:** | uint8 arbiter: Type of arbiter |
| **Return Value:** | uint8: Indication whether register is written and verified |
| **Side Effects:** | None |

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

| | |
|---|---|
| **Description:** | This function configures: Time segment 1, Time segment 2, Synchronization Jump Width, and Sampling Mode. |
| **Parameters:** | uint8 cfgTseg1: Time segment 1, values between 0x2 and 0xF are valid |
| | uint8 cfgTseg2: Time segment 2, values between 0x1 and 0x7 are valid |
| | uint8 sjw: Synchronization Jump Width, values between 0x0 and 0x3 are valid. |
| | uint8 sm: Sampling Mode, one or three sampling points are used |
| **Return Value:** | uint8: Indication whether register is written and verified |
| **Side Effects:** | None |

## uint8 CAN_SetRestartType(uint8 reset)

| | |
|---|---|
| **Description:** | This function sets reset type. Types of reset are Automatic and Manual. Manual reset is the recommended setting. Values 0 and 1 are valid. |
| **Parameters:** | uint8 reset: Reset type |
| **Return Value:** | uint8: Indication whether register is written and verified |
| **Side Effects:** | None |

## uint8 CAN_SetEdgeMode(uint8 edge)

| | |
|---|---|
| **Description:** | This function sets Edge Mode. Modes are 'R' to 'D' (Recessive to Dominant) and Both edges are used. Values 0 and 1 are valid. |
| **Parameters:** | uint8 edge: Edge Mode |
| **Return Value:** | uint8: Indication whether register is written and verified |
| **Side Effects:** | None |

## uint8 CAN_RXRegisterInit(uint32 *regAddr, uint32 config)

| | |
|---|---|
| **Description:** | This function writes CAN receive registers only. |
| **Parameters:** | uint32 * regAddr: Pointer to CAN receive register |
| | uint32 configuration: Value that will be written in register |
| **Return Value:** | uint8: Indication whether register is written and verified |
| **Side Effects:** | None |

## uint8 CAN_SetOpMode(uint8 opMode)

| | |
|---|---|
| **Description:** | This function sets Operation Mode. Operation modes are Active or Listen Only. Values 0 and 1 are valid. |
| **Parameters:** | uint8 opMode: Operation Mode value |
| **Return Value:** | uint8: Indication whether register is written and verified |
| **Side Effects:** | None |

## uint8 CAN_GetTXErrorflag(void)

| | |
|---|---|
| **Description:** | This function returns the flag that indicates if the number of transmit errors exceeds 0x60. |
| **Parameters:** | None |
| **Return Value:** | uint8: Indication whether the number of transmit errors exceeds 0x60 |
| **Side Effects:** | None |

## uint8 CAN_GetRXErrorflag(void)

| | |
|---|---|
| **Description:** | This function returns the flag that indicates if the number of receive errors exceeds 0x60. |
| **Parameters:** | None |
| **Return Value:** | uint8: Indication whether the number of receive errors exceeds 0x60 |
| **Side Effects:** | None |

## uint8 CAN_GetTXErrorCount(void)

| | |
|---|---|
| **Description:** | This function returns the number of transmit errors. |
| **Parameters:** | None |
| **Return Value:** | uint8: Number of transmit errors |
| **Side Effects:** | None |

## uint8 CAN_GetRXErrorCount(void)

| | |
|---|---|
| **Description:** | This function returns the number of receive errors. |
| **Parameters:** | None |
| **Return Value:** | (uint8) Number of receive errors |
| **Side Effects:** | None |

## uint8 CAN_GetErrorState(void)

| | |
|---|---|
| **Description:** | This function returns the error status of the CAN component. |
| **Parameters:** | None |
| **Return Value:** | uint8: Error status |
| **Side Effects:** | None |

## uint8 CAN_SetIrqMask(uint16 mask)

**Description:** This function enables or disables particular interrupt sources. Interrupt Mask directly writes to the CAN Interrupt Enable register.

**Parameters:** uint8 request: Interrupt enable or disable request. One bit per interrupt source

**Return Value:** uint8: Indication whether register is written and verified

**Side Effects:** None

## void CAN_ArbLostIsr(void)

**Description:** This function is the entry point to the Arbitration Lost Interrupt. It clears the Arbitration Lost interrupt flag. It is only generated if the Arbitration Lost Interrupt parameter is enabled.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

## void CAN_OvrLdErrrorIsr(void)

**Description:** This function is the entry point to the Overload Error Interrupt. It clears the Overload Error interrupt flag. It is only generated if the Overload Error Interrupt parameter is enabled.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

## void CAN_BitErrorIsr(void)

**Description:** This function is the entry point to the Bit Error Interrupt. It clears Bit Error interrupt flag. It is only generated if the Bit Error Interrupt parameter is enabled.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

## void CAN_BitStuffErrorIsr(void)

**Description:** This function is the entry point to the Bit Stuff Error Interrupt. It clears the Bit Stuff Error interrupt flag. It is only generated if the Bit Stuff Error Interrupt parameter is enabled.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

## void CAN_AckErrorIsr(void)

**Description:** This function is the entry point to the Acknowledge Error Interrupt. It clears the Acknowledge Error interrupt flag. It is only generated if the Acknowledge Error Interrupt parameter is enabled.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

## void CAN_MsgErrorIsr(void)

**Description:** This function is the entry point to the Form Error Interrupt. It clears the Form Error interrupt flag. It is only generated if the Form Error Interrupt parameter is enabled.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

## void CAN_CrcErrorIsr(void)

**Description:** This function is the entry point to the CRC Error Interrupt. It clears the CRC Error interrupt flag. It is only generated if the CRC Error Interrupt parameter is enabled.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

## void CAN_BusOffIsr(void)

| | |
|---|---|
| **Description:** | This function is the entry point to the Bus Off Interrupt. It puts the CAN component in Stop mode. It is only generated if the Bus Off Interrupt parameter is enabled. Enabling this interrupt is recommended. |
| **Parameters:** | None |
| **Return Value:** | None |
| **Side Effects:** | Stops CAN component operation |

## void CAN_MsgLostIsr(void)

| | |
|---|---|
| **Description:** | This function is the entry point to the Message Lost Interrupt. It clears the Message Lost interrupt flag. It is only generated if the Message Lost Interrupt parameter is enabled. |
| **Parameters:** | None |
| **Return Value:** | None |
| **Side Effects:** | None |

## void CAN_MsgTXIsr(void)

| | |
|---|---|
| **Description:** | This function is the entry point to the Transmit Message Interrupt. It clears the Transmit Message interrupt flag. It is only generated if the Transmit Message Interrupt parameter is enabled. |
| **Parameters:** | None |
| **Return Value:** | None |
| **Side Effects:** | None |

## void CAN_MsgRXIsr(void)

| | |
|---|---|
| **Description:** | This function is the entry point to the Receive Message Interrupt. It clears the Receive Message interrupt flag and calls the appropriate handlers for Basic and Full interrupt based mailboxes. It is only generated if the Receive Message Interrupt parameter is enabled. Enabling this interrupt is recommended. |
| **Parameters:** | None |
| **Return Value:** | None |
| **Side Effects:** | None |

## uint8 CAN_RxBufConfig(CAN_RX_CFG *rxConfig)

| | |
|---|---|
| **Description:** | This function configures all receive registers for a particular mailbox. The mailbox number contains CAN_RX_CFG structure. |
| **Parameters:** | CAN_RX_CFG * rxConfig: Pointer to structure that contains all required values to configure all receive registers for a particular mailbox |
| **Return Value:** | uint8: Indication if particular configuration of has been accepted or rejected |
| **Side Effects:** | None |

## uint8 CAN_TxBufConfig(CAN_TX_CFG *txConfig)

| | |
|---|---|
| **Description:** | This function configures all transmit registers for a particular mailbox. The mailbox number contains CAN_TX_CFG structure. |
| **Parameters:** | CAN_TX_CFG * txConfig: Pointer to structure that contains all required values to configure all transmit registers for a particular mailbox |
| **Return Value:** | uint8: Indication if particular configuration of has been accepted or rejected |
| **Side Effects:** | None |

## uint8 CAN_SendMsg(CANTXMsg *message)

| | |
|---|---|
| **Description:** | This function sends a message from one of the Basic mailboxes. The function loops through the transmit message buffer designed as Basic CAN mailboxes. It looks for the first free available mailbox and sends from it. There can only be three retries. |
| **Parameters:** | CAN_TX_MSG * message: Pointer to structure containing required data to send message |
| **Return Value:** | uint8: Indication if message has been sent |
| **Side Effects:** | None |

## uint8 CAN_SendMsg0-7(void)

| | |
|---|---|
| **Description:** | These functions are the entry point to Transmit Message 0-7. This function checks if mailbox 0-7 already has untransmitted messages waiting for arbitration. If so, it initiates transmission of the message. Only generated for Transmit mailboxes designed as Full. |
| **Parameters:** | None |
| **Return Value:** | uint8: Indication if Message has been sent |
| **Side Effects:** | None |

## void CAN_TxCancel(uint8 bufferId)

| | |
|---|---|
| **Description:** | This function cancels transmission of a message that has been queued for transmission. Values between 0 and 15 are valid. |
| **Parameters:** | uint8 bufferId: Number of Tx mailbox. |
| **Return Value:** | None |
| **Side Effects:** | None |

## void CAN_ReceiveMsg0-15(void)

| | |
|---|---|
| **Description:** | These functions are the entry point to the Receive Message 0-15 Interrupt. They clear Receive Message 0 - 15 interrupt flags. They are only generated for Receive mailboxes designed as Full interrupt based. |
| **Parameters:** | None |
| **Return Value:** | None |
| **Side Effects:** | None |

## void CAN_ReceiveMsg(uint8 rxMailbox)

| | |
|---|---|
| **Description:** | This function is the entry point to the Receive Message Interrupt for Basic mailboxes. It clears the Receive particular Message interrupt flag. It is only generated if one of the Receive mailboxes is designed as Basic. |
| **Parameters:** | uint8 rxMailbox: Mailbox number that triggers Receive Message Interrupt |
| **Return Value:** | None |
| **Side Effects:** | None |

## void CAN_Sleep(void)

| | |
|---|---|
| **Description:** | This is the preferred routine to prepare the component for sleep. The CAN_Sleep() routine saves the current component state. Then it calls the CAN_Stop() function and calls CAN_SaveConfig() to save the hardware configuration.<br>Call the CAN_Sleep() function before calling the CyPmSleep() or the CyPmHibernate() function. Refer to the PSoC Creator *System Reference Guide* for more information about power management functions. |
| **Parameters:** | None |
| **Return Value:** | None |
| **Side Effects:** | None |

## void CAN_Wakeup(void)

| | |
|---|---|
| **Description:** | This is the preferred routine to restore the component to the state when CAN_Sleep() was called. The CAN_Wakeup() function calls the CAN_RestoreConfig() function to restore the configuration. If the component was enabled before the CAN_Sleep() function was called, the CAN_Wakeup() function will also re-enable the component. |
| **Parameters:** | None |
| **Return Value:** | None |
| **Side Effects:** | Calling the CAN_Wakeup() function without first calling the CAN_Sleep() or CAN_SaveConfig() function may produce unexpected behavior. |

## uint8 CAN_Init(void)

| | |
|---|---|
| **Description:** | Initializes or restores the component according to the customizer Configure dialog settings. It is not necessary to call CAN_Init() because the CAN_Start() routine calls this function and is the preferred method to begin component operation. |
| **Parameters:** | None |
| **Return Value:** | uint8: Indication whether the configuration has been accepted or rejected |
| **Side Effects:** | All registers will be reset to their initial values. This reinitializes the component with the following exception: it will not clear data from the mailboxes. Enables power to the CAN Core. |

## uint8 CAN_Enable(void)

| | |
|---|---|
| **Description:** | Activates the hardware and begins component operation. It is not necessary to call CAN_Enable() because the CAN_Start() routine calls this function, which is the preferred method to begin component operation. |
| **Parameters:** | None |
| **Return Value:** | None |
| **Side Effects:** | None |

## void CAN_SaveConfig(void)

| | |
|---|---|
| **Description:** | This function saves the component configuration and non-retention registers. This function also saves the current component parameter values, as defined in the Configure dialog or as modified by appropriate APIs. This function is called by the CAN_Sleep() function. |
| **Parameters:** | None |
| **Return Value:** | None |
| **Side Effects:** | None |

### void CAN_RestoreConfig(void)

**Description:** This function restores the component configuration and non-retention registers. This function also restores the component parameter values to what they were prior to calling the CAN_Sleep() function.

**Parameters:** None

**Return Value:** None

**Side Effects:** Calling this function without first calling the CAN_Sleep() or CAN_SaveConfig() function may produce unexpected behavior. The following registers will revert to default values: CAN_INT_SR, CAN_INT_EN, CAN_CMD, and CAN_CFG.

## Sample Firmware Source Code

PSoC Creator provides many example projects that include schematics and example code in the Find Example Project dialog. For component-specific examples, open the dialog from the Component Catalog or an instance of the component in a schematic. For general examples, open the dialog from the Start Page or **File** menu. As needed, use the **Filter Options** in the dialog to narrow the list of projects available to select.

Refer to the “Find Example Project” topic in the PSoC Creator Help for more information.

## Interrupt Service Routines

There are several CAN component interrupt sources:

- Arbitration Lost Detection – The arbitration was lost while sending a message
- Overload Error – An overload frame was received
- Bit Error – A bit error was detected
- Bit Stuff Error - A bit stuffing error was detected
- Acknowledge Error – CAN message acknowledge error was detected
- Form Error – CAN message format error was detected
- CRC Error – CAN CRC error was detected
- Bus Off – CAN has reached the bus off state
- Message Lost – A new message arrived but there was nowhere to put it
- Transmit Message – The queued message was sent
- Receive Message – A message was received

All of these interrupt sources have entry points (functions) so you can place code in them. These functions are conditionally compiled depending on the customizer.

The Receive Message interrupt has a special handler that calls appropriate functions for Full and Basic mailboxes.

## Interrupt Output Use Cases

The following are example use cases of the hardware interrupt output line in the CAN component:

### Hardware Control of Logic on Interrupt Events

The hardware interrupt line can be used to perform simple tasks such as estimating the CAN bus load. By enabling the Message Transmitted and Message Received interrupts in the CAN component customizer, and connecting the interrupt line to a counter, the number of messages that are on the bus during a specific time interval can be evaluated. Also, actions can be taken directly in hardware if the message rate is above a certain value.

### Interrupt Output Interaction with DMA

The CAN component doesn't support DMA operation internally, but you can connect the DMA component to the external interrupt line (if it is enabled). You are responsible for the DMA configuration and operation. Also, you should keep in mind that it is necessary to handle some housekeeping tasks (for example, acknowledging the message and clearing the interrupt flags) in code for proper handling of CAN interrupts.

With a hardware DMA trigger you can handle registers and data transfers when a Message Received interrupt occurs, without any firmware executing in the CPU. This is also useful when handling RTR messages. The Message Transmitted interrupt can be used to trigger a DMA transfer to reload the message buffer with new data, without CPU intervention.

### Custom External Interrupt Service Routine

Custom external ISRs can be used in addition to or as a replacement to the internal ISR. When the external ISR is used in addition to the internal ISR, the Interrupt priority can be set to determine which ISR should execute first (internal or external), thus forcing actions before or after those coded in the internal ISR. When the external ISR is used as replacement for the internal ISR, you assume all responsibility for proper handling of CAN registers and events.

## Interrupt Output Interaction with the Interrupt Subsystem

The CAN component Interrupt Output settings allow you to:

- Enable or disable an external interrupt line (customizer option)
- Disable or bypass the internal ISR (customizer option)

- Fully customize the internal ISR (customizer option)
- Enable or disable specific interrupts handling function calls in the internal ISR, when the relevant event interrupts are enabled (customizer option). Individual interrupts (message transmitted, message received, receive buffer full, bus off state, and so on) can be enabled or disabled in the CAN component customizer. Once enabled the relevant function call is executed in the internal CAN_ISR. This allows you to disable (remove) such function calls.

The external interrupt line is visible only if enabled in the customizer.

If an external Interrupt component is connected, then the external Interrupt component is not started as part of the CAN_Start() API, and will have to be started outside that routine.

If an external Interrupt component is connected and the internal ISR is not disabled or bypassed, then two Interrupt components are connected to the same line. And in this case you will have two separate Interrupt components that will handle the same interrupt events. This is a specific, and in most cases undesirable, situation.

If the internal ISR is disabled or bypassed (using a customizer option) the internal Interrupt component will be removed during the build process.

If you choose to disable an individual interrupt function call in the internal interrupt routine (for an enabled interrupt event, by using a customizer option), the CAN block interrupt triggers (when the relevant event occurs), but no internal function call is executed in the internal CAN_ISR routine. An example use case is when you want to handle a specific event (for example, message received) through a different path, other than the standard user function call (for example, through DMA).

If you choose to fully customize the internal ISR (via customizer option) the CAN_ISR function will not contain any function call.

## Functional Description

For a complete description, refer to the Controller Area Network (CAN) chapter in the *PSoC® 3 and PSoC® 5 Technical Reference Manual*.

## Block Diagram and Configuration

For complete block diagram and configuration information, refer to the Controller Area Network (CAN) chapter in the *PSoC® 3 and PSoC® 5 Technical Reference Manual*.

## References

1. *ISO-11898: Road vehicles* -- Controller area network (CAN):
    - ❑ Part 1: Data link layer and physical signaling
    - ❑ Part 2: High-speed medium access unit

- ❑ Part 3: Low-speed, fault-tolerant, medium-dependent interface
- ❑ Part 4: Time-triggered communication
- ❑ Part 5: High-speed medium access unit with low-power mode

2. *CAN Specification Version 2 BOSCH*
3. *Inicore CANmodule-III-AHB Datasheet*

## Resources

The CAN component uses the dedicated CAN hardware block in the silicon.

## API Memory Usage

The component memory usage varies significantly, depending on the compiler, device, number of APIs used and component configuration. The following table provides the memory usage for all APIs available in the given component configuration.

The measurements have been done with the associated compiler configured in Release mode with optimization set for Size. For a specific design the map file generated by the compiler can be analyzed to determine the memory usage.

| **Configuration** | **PSoC 3 (Keil_PK51)** | | **PSoC 5 (GCC)** | | **PSoC 5LP (GCC)** | |
|---|---|---|---|---|---|---|
| | **Flash Bytes** | **SRAM Bytes** | **Flash Bytes** | **SRAM Bytes** | **Flash Bytes** | **SRAM Bytes** |
| Default | 3348 | 18 | N/A | N/A | 2044 | 21 |
| Additional receive buffer usage | + 8 | – | N/A | N/A | + 16 | – |
| Additional interrupt handler enabled | + 7 | – | N/A | N/A | + 12 | – |

## DC and AC Electrical Characteristics

Specifications are valid for –40 °C ≤ TA ≤ 85 °C and TJ ≤ 100 °C, except where noted. Specifications are valid for 1.71 V to 5.5 V, except where noted.

### DC Specifications

| Parameter | Description | Conditions | Min | Typ | Max | Units |
|---|---|---|---|---|---|---|
| $I_{DD}$ | Block current consumption | | – | – | 200 | µA |

## AC Specifications

| Parameter | Description | Conditions | Min | Typ | Max | Units |
|---|---|---|---|---|---|---|
| | Bit rate | Minimum 8 MHz clock | – | – | 1 | Mbit |

# Component Changes

This section lists the major changes in the component from the previous version.

| Current Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| 2.10.a | Minor datasheet edit. | |
| 2.10 | Added PSoC 5LP device support. | |
| | Added all CAN APIs with CYREENTRANT keyword when they included in .cyre file. | Not all APIs are truly reentrant. Comments in the component API source files indicate which functions are candidates.<br>This change is required to eliminate compiler warnings for functions that are not reentrant used in a safe way: protected from concurrent calls by flags or Critical Sections. |
| | Changed name of the component to “CAN_1” and updated snippets of code in examples for illustration the usage of receive and transmit message APIs. | To meet consistency. |
| | Updated DC and AC Electrical Characteristics section. | |
| 2.0.a | Replaced timing diagram and added descriptive text to datasheet | |
| | Minor datasheet edits and updates | |
| 2.0 | Added interrupt output to the component symbol | Updated Marketing Requirements Document MRD for the PSoC3 CAN Component |
| | Added ConnectExtInterruptLine, IntISRDisable, FullCastomIntISR, AdvancedInterruptTab parameters and ISR helper parameters in the component symbol | Updated Marketing Requirements Document MRD for the PSoC3 CAN Component |
| | Added stop statement at the beginning of the CAN_Init() function. | To ensure that CAN is stopped at initialization |
| | Updated interrupt handlers functions. In all cases the interrupt flag is cleared before the user code. | To clear interrupt flags in the same manner |
| | Removed obsolete defines | |

| Current Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| 1.50.a | Added characterization data to datasheet | |
| | Datasheet text edits | |
| 1.50 | Added Sleep / Wakeup APIs. | These APIs provide support for low-power modes. |
| | Added CAN_Init() and CAN_Enable() APIs. | To comply with corporate standard and provide an API to initialize/restore the component without starting it. |