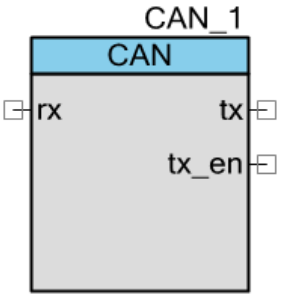
Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

Controller Area Network (CAN)

3.0



Features

- CAN2.0A and CAN2.0B protocol implementation, ISO 11898-1 compliant
- Supports standard 11-bit and extended 29-bit identifiers
- Programmable bit rate up to 1 Mbps
- Up to 16 receive mailboxes with hardware message filtering
- Up to 8 transmit message mailboxes with programmable transmit priority: Round Robin and Fixed
- Two-wire or three-wire interface to external transceiver (tx, rx, and tx enable)
- Supports listen-only mode of operation
- Supports single shot transmission, as well as internal and external loopback modes [1]

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.

The CAN Component is certified by the C&S group GmbH based on the standard protocol and data link layer conformance tests. A complete certification report can be made available on request.

When to Use a CAN

CAN was defined by Bosch and is widely used in Industrial and Automotive applications for high reliability systems. In the Automotive market, it is used in engine control units, sensors, safety systems, etc. and is used as a network connection bus for vehicle body electronics (lamp clusters, electric windows etc.).

1. Not available for PSoC™ 3/PSoC™ 5LP device families

The Component can be used for setting up the CAN communications in order to transmit and/or receive messages over CAN network.

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 *

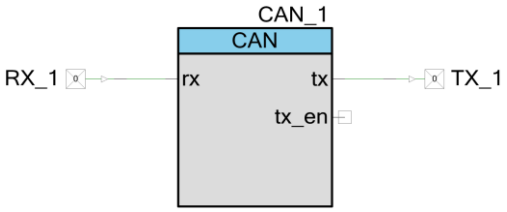
External transceiver enable signal. This output displays when the **Add transceiver enable signal** option is selected in the **Configure** dialog.

interrupt – Output *

External interrupt signal. This output displays when the **Enable external interrupt line** option is selected in the **Configure** dialog.

Schematic Macro Information

The Component Catalog provides a schematic macro for the CAN Component with rx and tx terminals connected to Input and Output Pin Components respectively. The schematic macro has the default configuration settings for the CAN and cy_pins Components.



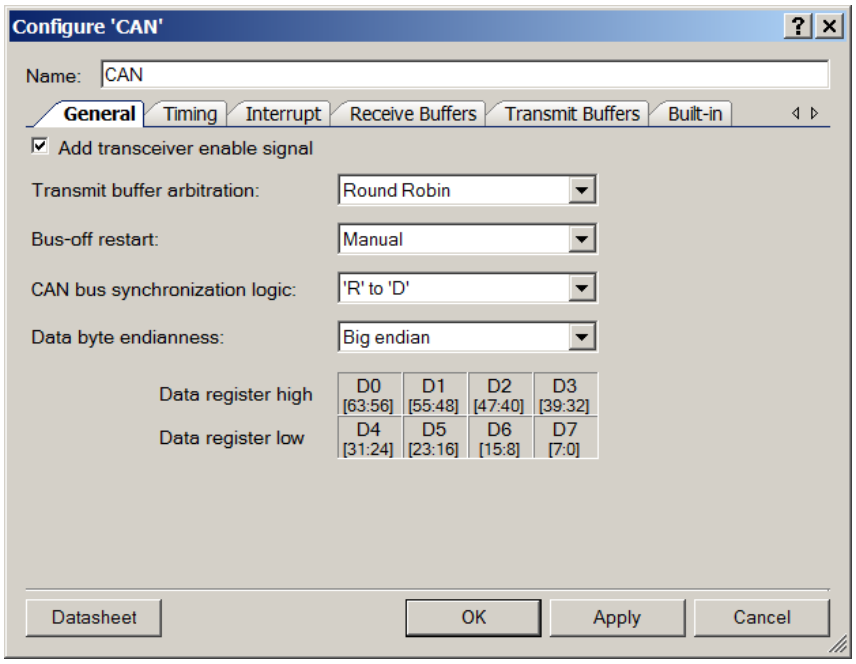
Component Parameters

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



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.

Note The interface between controller and transceiver chip is not standardized as per the CAN specifications. The transceiver enable signal in the PSoC™ CAN Component works well with NXP TJA1050 transceiver logic, which is treated as an industry standard transceiver. If your transceiver is not compliant to this logic, then disable the transceiver enable signal in the Component and use a firmware-controlled Pin Component to enable/disable the transceiver.

Transmit buffer arbitration

Defines the message transmission arbitration scheme:

- **Round Robin** (default) – Mailboxes are served in a defined order: 0-1-2 ... 7-0-1. A particular mailbox is only selected if its TX_REQ flag is set. This scheme guarantees that all mailboxes receive the same probability to send a message.
- **Fixed priority** – Mailbox 0 has the highest priority. This way it is possible to designate mailbox 0 as the mailbox 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

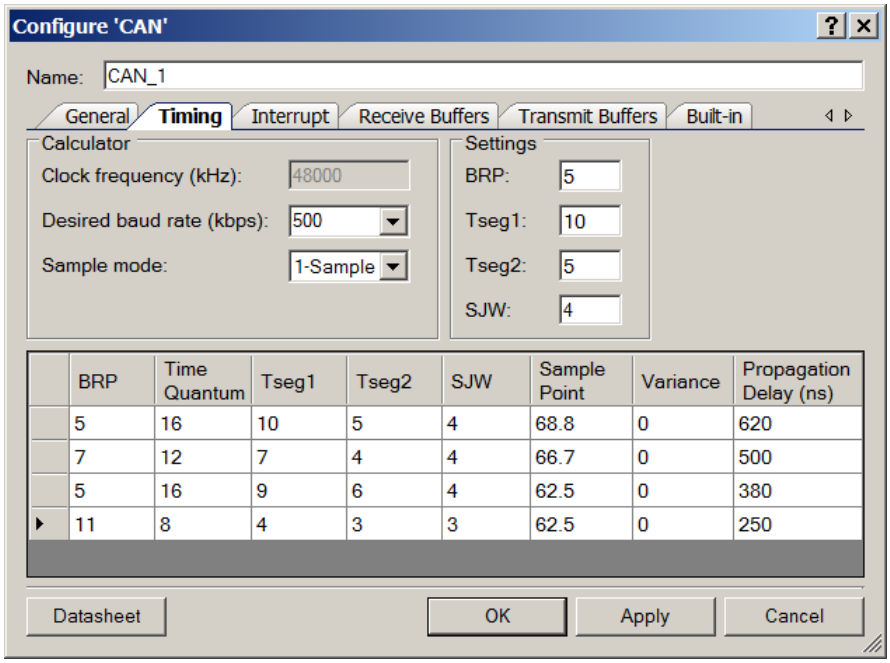
Data byte endianness

This option is not available for PSoC™ 3/PSoC™ 5LP device families. The byte position of the CAN receive and transmit data field endianness can be modified using this setting

- **Big endian** (default) - CAN data byte endianness is not swapped.
- **Little endian** - CAN data byte endianness is swapped during transmission/reception.

A graphical representation of the order of transmitted bytes is provided in the Component customizer based on this setting.

Timing Tab



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

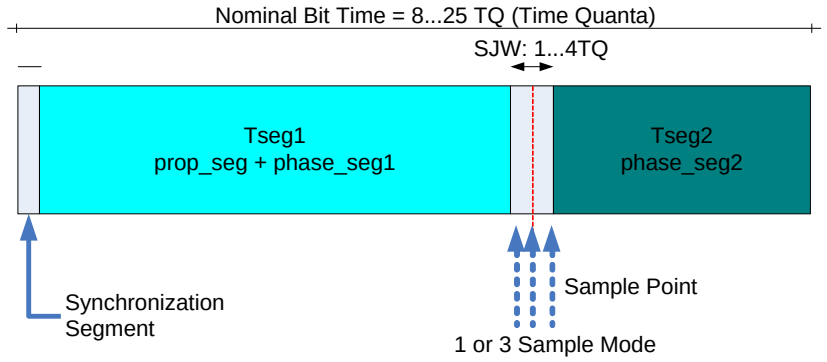
Calculator

- **Clock frequency** (in kHz) – The system clock frequency equal to BUS_CLK (PSoC™ 3/ PSoC™ 5LP) or SYSCLK (PSoC™ 4).
- **Desired baud rate** (in kbps) – Options are: **10**, **20**, **62.5**, **125**, **250**, **500**, **800**, or **1000**.
- **Sample mode** – Configuration of sampling mode. Options are: **1-Sample** or **3-Sample**.

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**).
- **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**.

The following shows the CAN bit timing representation:



Table

Bit timing is calculated as per the ISO specifications, and the proposed register settings for time segments (BRP, Tseg1, Tseg2, and SJW) 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 **Settings** input boxes.

You may also choose to manually enter values for Tseg1, Tseg2, SJW and BRP in the provided input boxes. Incorrect bit timing settings might cause the CAN controller to remain in an error state. So, if the values entered manually do not match any of the values given in the table, the Component displays a warning message: “An invalid set of BRP/Tseg1/Tseg2/SJW is entered. Please select a valid set from the table."

In order for CAN Component to operate properly, clock configuration should be set carefully. It is suggested that clock frequency is set as a multiple of desired baud rate. Also, accuracy of clock used (BUS_CLK for PSoC™ 3/PSoC™ 5LP or SYSCLK for PSoC™ 4) must match the CAN clock tolerance requirement which is:

- 1.58% or better for 125kbps and slower bit-rates
- 0.5% or better for bit-rates faster than 125kbps

Configurations that don’t fit within CAN clock tolerance range are removed from the table. In case when there are no rows provided in the table, warning message is displayed by the Component and user might have to change the clock settings to get a valid configuration in the table.

It is recommended to use an external XTAL or an external clock with higher accuracy as the clock source for the chip, if the internal clock tolerance is less than the desired value.

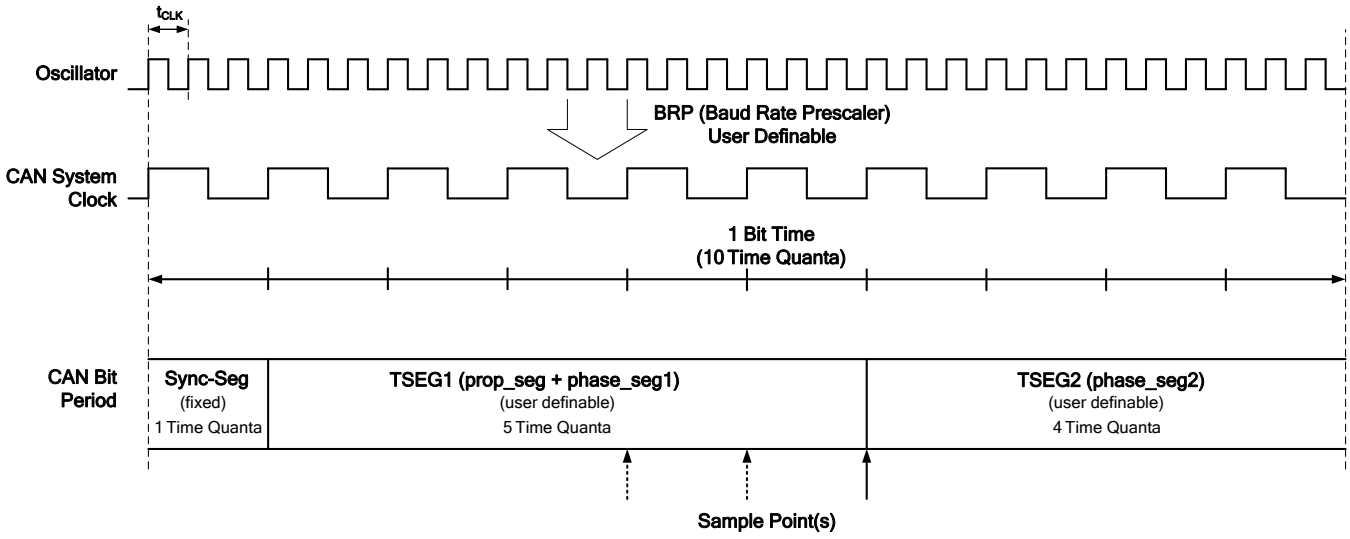
For the PSoC™ 4 device family, it is recommended to enable the WCO PLL lock to achieve the desired clock tolerance for the CAN block. Clock settings can be configured in the PSoC™ Creator Clock Editor (**Clocks** tab in the Design-Wide Resources (*.cydwr) file). For more information, refer to PSoC™ Creator Help.

The calculator computes the variance value for all bit timing configurations and displays it in the **Variance** column (in percent). Variance indicates a deviation of actual baud rate from its nominal value. This deviation is composed of clock accuracy range and rounding error for clock frequency/baud rate ratio. Calculator table is sorted by Variance ascending, and then by Sample point descending, so preferable configurations appear on the top of the table.

The table also displays the maximum allowed propagation delay for the each of the configurations (**Propagation Delay (ns)** column). The worst case propagation delay estimated for the CAN bus must be less than or equal to this value. The length of propagation delay becomes more predominant at higher baud rates.

Bit Time Segments

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



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

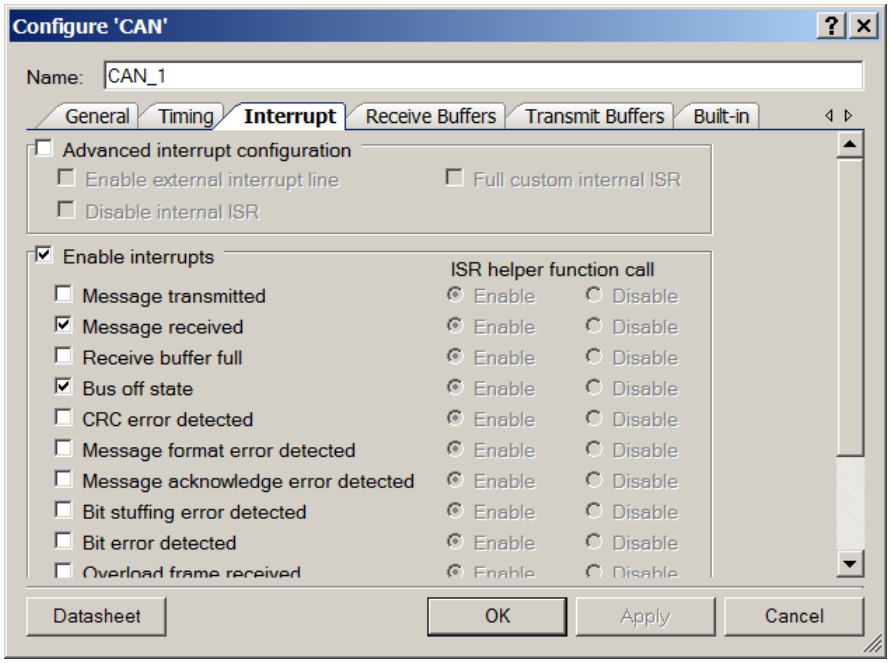
TIME QUANTUM = m × MINIMUM TIME QUANTUM,

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



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().

  When disabling the global interrupts, the CAN Component displays the following message:

  "Interrupts will not be generated by the CAN Component if the Enable interrupts option is unchecked. Do you want to disable all mailbox interrupts on Receive Buffers and Transmit Buffers tabs (IRQ column)?"

  - **Yes** – Uncheck the **Enable interrupts** check box, and uncheck all individual interrupts (IRQ) on the **Transmit Buffers** and **Receive Buffers** tabs. All the interrupt options are grayed out as well.
  - **No** (default) – Uncheck the **Enable interrupts** check box and keep all individual transmit and receive mailbox interrupts as they are.
  - **Cancel** – No changes are made.

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 want to disable all mailbox interrupts on Transmit Buffers tab (IRQ column)?"

- **Yes** – Uncheck the **Message transmitted** check box, and uncheck all individual transmit mailbox interrupts on the **Transmit Buffers** tab.
- **No** (default) – Uncheck the **Message transmitted** check box, and keep all individual transmit mailbox 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 want to disable all mailbox interrupts on Receive buffers tab (IRQ column)?"

- **Yes** (default) – Uncheck the **Message Received** check box, and uncheck all individual receive mailbox interrupts on the **Receive Buffers** tab.
- **No** – Uncheck the **Message Received** check box, and keep all individual receive mailbox 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.

Single shot transmission failure

Enable or disable interrupt for capturing errors during single shot transmission. Indicates that the mailbox set for single shot transmission experienced an arbitration loss or a bus error during transmission. This option is not available for PSoC™ 3/PSoC™ 5LP part families.

Stuck at zero

Enable or disable interrupts to capture errors stuck at dominant errors. Indicates if the rx input remains stuck at 0 (dominant level) for more than 11 consecutive bit times. This option is not available for PSoC™ 3/PSoC™ 5LP part families.

RTR automatic reply sent

Enable or disable interrupt to handle the automatic reply message transmission for an RTR request. Indicates that a RTR auto-reply message was sent. This option is not available for PSoC™ 3/PSoC™ 5LP part families.

Advanced interrupt configuration

Enable advanced 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). In the PSoC™ 4 device family, the internal ISR is disabled automatically if the Enabled External Interrupt Line is selected.

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. For the PSoC™ 4 device family, this option is selected automatically when the Enable external interrupt line option is checked.

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` */
```

Or use *CAN_1_Isr_Callback()* macro callback.

Default is unselected.

The check box is available (not grayed out) only if **Disable Internal ISR** is unchecked.

ISR Helper Function Call

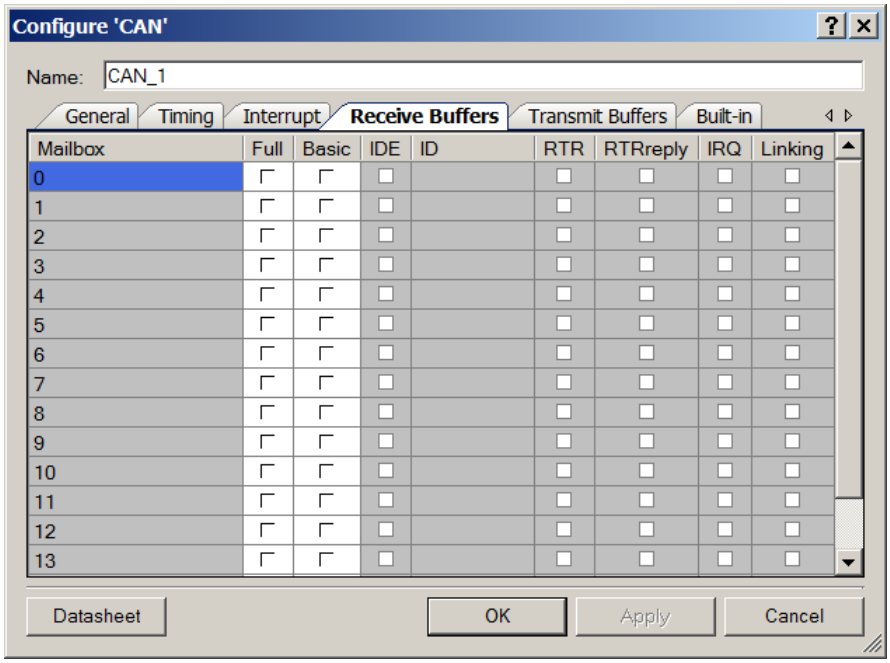
If only basic interrupt settings are used, when an interrupt occurs, the CAN ISR calls relevant user-customizable functions (ISR helpers) based on the enabled interrupts.

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



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** = Checked 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
- **Linking** = Cleared

IDE

When the IDE box is cleared, the identifier is limited to 11 bits. When IDE is selected, the identifier is limited to 29 bits.

ID

The message identifier. The 7 most significant bits of CAN message ID cannot be all recessive, so its range is from 0x000 to 0x7EF for standard 11-bit identifier and to 0x1FBFFFFF for extended 29-bit identifier (IDE).

RTR - Remote Transmission Request

When selected in Full CAN mode, it configures the acceptance filter settings to only allow receipt of messages whose RTR bit is set.

It is not recommended to select this in Basic CAN mode as the receiving node will have to process the message and prepare the response in software within the limited time frame.

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** (default) – Select the **IRQ** check box and select the **Message received** Interrupt check box on the **Interrupt** tab.
- **No** – Select the **IRQ** check box and leave the **Message received** Interrupt check box in the **Interrupt** tab as is.
- **Cancel** – No changes are made.

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

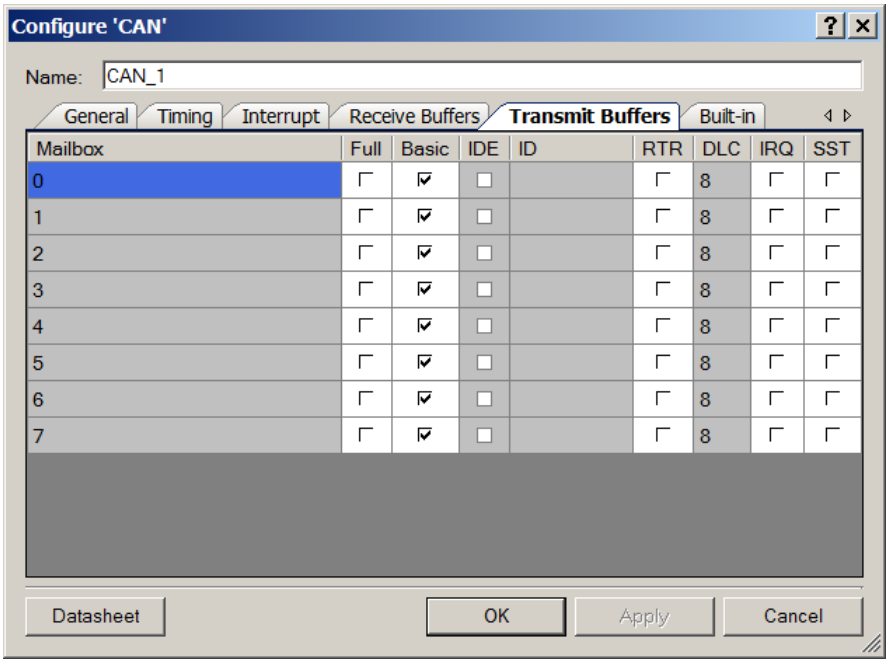
Every Full RX mailbox has a predefined API. The function list is available in the *CAN_1_TX_RX_func.c* project file. These functions are conditionally compiled depending on the receive mailbox setting. Only mailboxes defined as Full have their respective functions compiled.

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.

Transmit Buffers Tab



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**, **SST** 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
- **SST** = Cleared

Basic

By default, the **Basic** check box is selected for all of the mailboxes. If **Basic** is selected, the options **ID**, 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 (unavailable)
- **ID** = Nothing (unavailable)
- **RTR** = Cleared
- **DLC** = 8 (unavailable)
- **IRQ** = Cleared
- **SST** = Cleared

IDE

When the IDE box is cleared, the identifier is limited to 11 bits. When IDE is selected, the identifier is limited to 29 bits.

ID

The message identifier. The 7 most significant bits of CAN message ID cannot be all recessive, so its range is from 0x000 to 0x7EF for standard 11-bit identifier and to 0x1FBFFFFF for extended 29-bit identifier (IDE).

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** (default) – Select **IRQ** and the **Message transmitted** interrupt check box in the **Interrupt** tab.
- **No** – Select **IRQ**. The **Message transmitted** interrupt check box in the **Interrupt** tab remains cleared.
- **Cancel** – No changes are made.

SST

Single Shot Transmission – if enabled retransmission of a CAN message due to an arbitration loss or a bus error is prevented.

Default option is not selected. Option is not available for PSoC™ 3/PSoC™ 5LP.

This feature has to be enabled for all messages in a Time Triggered CAN system.

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 common function provided for all Basic Transmit mailboxes:

```
uint8 CAN_1_SendMsg(const 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 (0x000 to 0x7EF).
  - For an extended message (IDE = 1) identifier limited to 29 bits (0x00000000 to 0x1FBFFFFF)

- 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 Disable, 1 – 0xFF: IRQ Enable)
- SST (0 – SST Disable, 1 – 0xFF: SST Enable). Not available for PSoC™ 3/PSoC™ 5LP device families.
- 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:

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

Clock Selection

The CAN Component is connected to the BUS_CLK (PSoC™ 3/PSoC™ 5LP) or SYSCLK (PSoC™ 4) clock signal. A minimum value of 10 MHz is required to support a maximum CAN baud rate of 1 Mbps. A lower clock frequency can be used to support a lower data rate like 125 Kbps.

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.”

Functions

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/SYSCLK.
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_SetSwapDataEndianness() [2]	Enables or disables the endian swapping of CAN data bytes.
CAN_SetEdgeMode()	Sets Edge mode.
CAN_RXRegisterInit()	Writes only receive CAN registers.
CAN_SetOpMode()	Sets Operation mode.
CAN_SetErrorCaptureRegisterMode() [2]	Sets the error capture register mode to free running or error capture mode.
CAN_ReadErrorCaptureRegister() [2]	This function returns the value of error capture register.
CAN_ArmErrorCaptureRegister() [2]	This function arms the error capture register when the ECR is in error capture mode.
CAN_GetTXErrorFlag()	Returns the flag that indicates if the number of transmit errors equals or exceeds 0x60.

[2] This function is **not** available for PSoC™ 3/PSoC™ 5LP part families.

Function	Description
CAN_GetRXErrorFlag()	Returns the flag that indicates if the number of receive errors equals or 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.
CAN_OvrLdErrorIsr()	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_SSTErrorIsr() [2]	Clears SST error flag and removes the failed message from the transmit mailbox.
CAN_RtrAutoMsgSentIsr() [2]	Clears RTR Auto Message sent flag.
CAN_StuckAtZeroIsr() [2]	Clears StuckAtZeroFlag. 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

Function	Description
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.”

Note When you use functions that return indication of execution – verify return value, to be sure that function call was successful.

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.

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

Value	Description
CYRET_SUCCESS	Function passed successfully.
CAN_FAIL	Function failed.

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.

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

Value	Description
CYRET_SUCCESS	Function passed successfully.
CAN_FAIL	Function failed.

Side Effects: Pending message in the Tx buffer of PSoC™ 3/PSoC™ 5LP will not be aborted on calling the CAN_Stop() API. User has to abort all pending messages before calling the CAN_Stop() function to make sure that the block stops all the message transmission immediately.

uint8 CAN_GlobalIntEnable(void)

Description: This function enables global interrupts from the CAN Component.

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

Value	Description
CYRET_SUCCESS	Function passed successfully.
CAN_FAIL	Function failed.

uint8 CAN_GlobalIntDisable(void)

Description: This function disables global interrupts from the CAN Component.

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

Value	Description
CYRET_SUCCESS	Function passed successfully.
CAN_FAIL	Function failed.

uint8 CAN_SetPreScaler(uint16 bitrate)

Description: This function sets the prescaler for generation of the time quanta from the BUS_CLK/SYSCLK. Values between 0x0 and 0x7FFF are valid.

Parameters: uint16 bitrate: PreScaler value

Time Quantum = (bitrate + 1) clock cycles

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

Value	Description
CYRET_SUCCESS	Function passed successfully.
CAN_FAIL	Function failed.
CAN_OUT_OF_RANGE	Function parameter is out of range

uint8 CAN_SetArbiter(uint8 arbiter)

Description: This function sets the arbitration type for transmit buffers. Types of arbiters are Round Robin and Fixed priority.

Parameters: uint8 arbiter: Type of arbiter

Value	Description
CAN_ROUND_ROBIN	Round robin arbitration.
CAN_FIXED_PRIORITY	Fixed priority arbitration.

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

Value	Description
CYRET_SUCCESS	Function passed successfully.
CAN_FAIL	Function failed.

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: Length of time segment 1, values between 0x2 and 0xF are valid

uint8 cfgTseg2: Length of 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.

Value	Description
CAN_ONE_SAMPLE_POINT	One sampling points is used
CAN_THREE_SAMPLE_POINTS	Three sampling points are used.

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

Value	Description
CYRET_SUCCESS	Function passed successfully.
CAN_FAIL	Function failed.
CAN_OUT_OF_RANGE	Function parameter is out of range

uint8 CAN_SetRestartType(uint8 reset)

Description: This function sets reset type. Types of reset are Automatic and Manual. Manual reset is the recommended setting.

Parameters: uint8 reset: Reset type

Value	Description
CAN_MANUAL_RESTART	After bus-off, the CAN must be restarted 'by hand'. This is the recommended setting.
CAN_AUTO_RESTART	After bus-off, the CAN is restarting automatically after 128 groups of 11 recessive bits.

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

Value	Description
CYRET_SUCCESS	Function passed successfully.
CAN_FAIL	Function failed.

uint8 CAN_SetSwapDataEndianness(uint8 swap)

Description: This function selects whether the data byte endianness of the CAN receive and transmit data fields has to be swapped or not swapped. This is useful to match the data byte endianness to the endian setting of the processor or the used CAN protocol. This function is not applicable to PSoC™ 3/PSoC™ 5LP part families.

Parameters: uint8 swap: Swap Enable/Disable setting

Value	Description
CAN_SWAP_ENDIANNESS_ENABLE	Endianness of transmitted/received data byte fields are swapped during multibyte data transmission(Little endian).
CAN_SWAP_ENDIANNESS_DISABLE	Endianness of transmitted/received data byte fields are not swapped during multi byte data transmission (Big endian).

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

Value	Description
CYRET_SUCCESS	Function passed successfully.
CAN_FAIL	Function failed.

uint8 CAN_SetEdgeMode(uint8 edge)

Description: This function sets Edge Mode. Modes are 'R' to 'D' (Recessive to Dominant) and Both edges are used.

Parameters: uint8 edge: Edge Mode

Value	Description
CAN_EDGE_R_TO_D	Edge from ‘R’ to ‘D’ is used for synchronization.
CAN_BOTH_EDGES	Both edges are used.

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

Value	Description
CYRET_SUCCESS	Function passed successfully.
CAN_FAIL	Function failed.

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

Value	Description
CYRET_SUCCESS	Function passed successfully.
CAN_FAIL	Function failed.
CAN_OUT_OF_RANGE	Function parameter is out of range

uint8 CAN_SetOpMode(uint8 opMode)

Description: This function sets Operation Mode. Operation Mode values can be one from the following table.

Parameters: uint8 opMode: Operation Mode value

Value	Description
CAN_STOP_MODE	The CAN controller is in the Stop mode.
CAN_ACTIVE_RUN_MODE	The CAN controller is in the Active mode.
CAN_LISTEN_ONLY_MODE	CAN controller in listen only mode. In this mode, the CAN controller receives all CAN bus traffic but doesn't send any information on the CAN bus. The output is held at 'R' level. This feature is useful for automatic bit-rate detection. [3]
CAN_INTERNAL_LOOP_BACK	CAN controller in internal loopback mode. This mode is used for testing purpose. In this mode, the CAN controller receives the sending data. No data is sent to the network and no data is received. Not available for PSoC™ 3 / PSoC™ 5LP device families.
CAN_EXTERNAL_LOOP_BACK	CAN controller in external loopback mode. In this mode the CAN controller participates in the regular CAN transmission and reception. Further, a copy of all sending messages is received. This mode works only if at least one additional CAN node is on the network. Not available for PSoC™ 3 / PSoC™ 5LP device families.

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

Value	Description
CYRET_SUCCESS	Function passed successfully.
CAN_FAIL	Function failed.

Side Effects: For PSoC™ 4 device family, the function re-initializes the CAN registers.

[3] CAN_SetPreScaler() and CAN_SetTsegSample() APIs can be used to change the CAN bit rate. Each bit rate requires set of timing parameters that could be either calculated (refer to Functional Description) or generated by the Component's customizer for each required bitrate (refer to CAN_ARBITER, CAN_CFG_REG_TSEG1, CAN_CFG_REG_TSEG2, CAN_CFG_REG_SJW defines in CAN.h file). If you want to switch between several bit rates then you need to collect sets of timing parameters for each desired bitrate.

uint8 CAN_SetErrorCaptureRegisterMode(uint8 ecrMode)

Description: This function sets the error capture register mode. The 2 modes are possible: free running and error capture mode.

Parameters: uint8 ecrMode: The Error Capture register mode setting

Value	Description
CAN_ECR_FREE_RUNNING	The ECR captures the field and bit position within the current CAN frame.
CAN_ECR_ERROR_CAPTURE	In this mode the ECR register only captures an error event. For successive error captures, the ECR needs to be armed again by writing to the ECR register.

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

Value	Description
CYRET_SUCCESS	Function passed successfully.
CAN_FAIL	Function failed.

uint32 CAN_ReadErrorCaptureRegister(void)

Description: This function returns the value of error capture register.

Return Value: uint32: Returns the value of error capture register

Bits	Name	Description
0	ECR_STATUS	ECR STATUS - 0: ECR register captured an error or it is free running mode 1: ECR register is armed
3:1	ERROR_TYPE	Error type - 000 : Arbitration loss 001 : Bit Error 010 : Bit Stuffing Error 011 : Acknowledge Error 100 : Form Error 101 : CRC Error Others : N/A
4	TX_MODE	TX Mode - 0: No status 1: CAN Controller is transmitter
5	RX_MODE	RX Mode - 0: No status 1: CAN Controller is receiver
11:6	BIT	Bit number inside of Field
16:12	Field	Field - 0x00 : Stopped 0x01 : Synchronize 0x05 : Interframe 0x06 : Bus Idle 0x07 : Start of Frame 0x08 : Arbitration 0x09 : Control 0x0A : Data 0x0B : CRC 0x0C : ACK 0x0D : End of frame 0x10 : Error flag 0x11 : Error echo 0x12 : Error delimiter 0x18 : Overload flag 0x19 : Overload echo 0x1A : Overload delimiter Others : N/A

uint8 CAN_ArmErrorCaptureRegister(void)

Description: This function arms the error capture register when the ECR is in error capture mode, by setting the ECR_STATUS bit in the ECR register.

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

Value	Description
CYRET_SUCCESS	Function passed successfully.
CAN_FAIL	Function failed.

uint8 CAN_GetTXErrorFlag(void)

Description: This function returns the flag that indicates if the number of transmit errors equals or exceeds 0x60.

Return Value: uint8: Indication whether the number of transmit errors equals or exceeds 0x60

uint8 CAN_GetRXErrorFlag(void)

Description: This function returns the flag that indicates if the number of receive errors equals or exceeds 0x60.

Return Value: uint8: Indication whether the number of receive errors equals or exceeds 0x60

uint8 CAN_GetTXErrorCount(void)

Description: This function returns the number of transmit errors.

Return Value: uint8: Number of transmit errors

uint8 CAN_GetRXErrorCount(void)

Description: This function returns the number of receive errors.

Return Value: (uint8) Number of receive errors

uint8 CAN_GetErrorState(void)

Description: This function returns the error status of the CAN Component.

Return Value: uint8: Error status

Value	Description
“00”	Error active (normal operation)
“01”	Error passive
“1x”	Bus off

uint8 CAN_SetIrqMask(uint16 mask)

Description: This function enables or disables particular interrupt sources. Interrupt Mask directly writes to the CAN Interrupt Enable register. A particular interrupt source is enabled by setting its respective flag to 1.

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

Value	Description
CAN_GLOBAL_INT_ENABLE	Global Interrupt Enable Flag.
CAN_ARBITRATION_LOST_ENABLE	Arbitration Loss Interrupt Enable.
CAN_OVERLOAD_ERROR_ENABLE	Overload Interrupt Enable.
CAN_BIT_ERROR_ENABLE	Bit Error Interrupt Enable.
CAN_STUFF_ERROR_ENABLE	Stuff Error Interrupt Enable.
CAN_ACK_ERROR_ENABLE	Ack Error Interrupt Enable.
CAN_FORM_ERROR_ENABLE	Form Error Interrupt Enable.
CAN_CRC_ERROR_ENABLE	CRC Error Interrupt Enable.
CAN_BUS_OFF_ENABLE	Busoff State Interrupt Enable.
CAN_BUS_STUCK_AT_ZERO_ENABLE*	Stuck at zero Interrupt Enable.
CAN_BUS_SST_ERROR_ENABLE*	Single shot transmission failure Interrupt Enable.
CAN_BUS_RTR_AUTO_REPLY_ENABLE*	RTR auto reply sent interrupt enable.
CAN_RX_MSG_LOST_ENABLE	Rx Msg Loss Interrupt Enable.
CAN_TX_MESSAGE_ENABLE	Tx Msg Sent Interrupt Enable.
CAN_RX_MESSAGE_ENABLE	Msg Recived Interrupt Enable.

* These interrupt sources are **not** available for PSoC™ 3/PSoC™ 5LP part families.

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

Value	Description
CYRET_SUCCESS	Function passed successfully.
CAN_FAIL	Function failed.

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.

void CAN_OvrLdErrorIsr(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.

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.

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.

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.

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.

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.

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.

Side Effects: Stops CAN Component operation.

void CAN_SSTErrorIsr(void)

Description: This function is the entry point to the single shot transmission error Interrupt. It is only generated if the single shot transmission is enabled. Generated when the mailbox set for single shot transmission experienced an arbitration loss or a bus error during transmission.

Side Effects: Removes message that failed SST transmission from the Tx mailbox.

void CAN_RtrAutoMsgSentIsr(void)

Description: This function is the entry point to the RTR automatic message sent Interrupt. It is only generated if RTR message sent interrupt parameter is enabled.

void CAN_StuckAtZeroIsr(void)

Description: This function is the entry point to the stuck at dominant bit Interrupt. It is only generated if Stuck at zero interrupt parameter is enabled. Enabling this interrupt is recommended.

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.

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.

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.

uint8 CAN_RxBufConfig(const CAN_RX_CFG *rxConfig)

Description: This function configures all receive registers for a particular mailbox. The mailbox number contains CAN_RX_CFG structure.

Parameters: const 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

Value	Description
CYRET_SUCCESS	Function passed successfully.
CAN_FAIL	Function failed.

uint8 CAN_TxBufConfig(const CAN_TX_CFG *txConfig)

Description: This function configures all transmit registers for a particular mailbox. The mailbox number contains CAN_TX_CFG structure.

Parameters: const 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

Value	Description
CYRET_SUCCESS	Function passed successfully.
CAN_FAIL	Function failed.

uint8 CAN_SendMsg(const 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: const CAN_TX_MSG * message: Pointer to structure containing required data to send message

Return Value: uint8: Indication if message has been sent

Value	Description
CYRET_SUCCESS	Function passed successfully.
CAN_FAIL	Function failed.

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.

Return Value: uint8: Indication if Message has been sent

Value	Description
CYRET_SUCCESS	Function passed successfully.
CAN_FAIL	Function failed.

void CAN_TxCancel(uint8 bufferId)

Description: This function cancels transmission of a message that has been queued for transmission. Values between 0 and 7 are valid.

Parameters: uint8 bufferId: Number of Tx mailbox.

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.

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

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.

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.

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. The function restores CAN Rx and Tx buffer control register configurations provided by the customizer. If the Component was enabled before the CAN_Sleep() function was called, the CAN_Wakeup() function will also re-enable the Component.

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.

Return Value: uint8: Indication whether the configuration has been accepted or rejected

Value	Description
CYRET_SUCCESS	Function passed successfully.
CAN_FAIL	Function failed.

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.

Return Value: uint8: Indication whether the configuration has been accepted or rejected

Value	Description
CYRET_SUCCESS	Function passed successfully.
CAN_FAIL	Function failed.

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. This function is applicable only for PSoC™ 3/PSoC™ 5LP part families.

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. This function is applicable only for PSoC™ 3/PSoC™ 5LP part families.

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.

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. If re-initialization of the Component is required, then the CAN_Init() function can be called before the CAN_Start() or CAN_Enable() function.

Macros

- Set/Clear bits macro for CAN_RX mailbox (i)
  - CAN_RX_ACK_MESSAGE(i) – acknowledges a new message
  - CAN_RX_RTR_ABORT_MESSAGE(i) – requests removal of a pending RTR message reply
  - CAN_RX_BUF_ENABLE (i) [4] – enables RX Buffer
  - CAN_RX_BUF_DISABLE (i) [4] – disables RX Buffer
  - CAN_SET_RX_RTRREPLY (i) [4] – enables automatic RTR message handling
  - CAN_CLEAR_RX_RTRREPLY (i) [4] – disables automatic RTR message handling
  - CAN_RX_INT_ENABLE (i) [4] – enables Interrupt generation
  - CAN_RX_INT_DISABLE (i) [4] – disables Interrupt generation
  - CAN_SET_RX_LINKING (i) [4] – links this buffer with the next buffer
  - CAN_CLEAR_RX_LINKING (i) [4] – removes linking with the next buffer
  - CAN_SET_RX_WNPL (i) – sets the Write Protect bit, which lets Bits[6:3] of the command register become modified
  - CAN_CLEAR_RX_WNPL (i) – clears the Write Protect bit, which makes Bits[6:3] of the command register unchanged
  - CAN_SET_RX_WNPH (i) – sets the Write Protect bit
  - CAN_CLEAR_RX_WNPH(i) – clears the Write Protect bit
  - CAN_GET_DLC(mailbox) – gets Data Length Code
  - “i” – is the mailbox number.

---

[4] When using this macro, use the CAN_SET_RX_WNPL() and CAN_CLEAR_RX_WNPL() macros to set/clear the Write Protect bit of the command register. The following example shows the suggested sequence:

```
CAN_SET_RX_WNPL();
CAN_RX_BUF_ENABLE();
CAN_CLEAR_RX_WNPL();
```

- Set/Clear bits macro for CAN_TX mailbox (i)
  - CAN_TX_TRANSMIT_MESSAGE (i) – requests Message Transmit
  - CAN_TX_ABORT_MESSAGE (i) – requests removal of a pending message
  - CAN_TX_INT_ENABLE (i) – enables Transmit Interrupt
  - CAN_TX_INT_DISABLE (i) – disables Transmit Interrupt
  - CAN_SET_TX_WNPL (i) – sets Write Protect Bit 1
  - CAN_CLEAR_TX_WNPL (i) – sets Write Protect Bit 1
  - CAN_SET_TX_IDE (i) – sets an Extended format message
  - CAN_CLEAR_TX_IDE(i) – sets a Standard format message
  - CAN_SET_TX_RTR(i) – sets a RTR format message
  - CAN_CLEAR_TX_RTR(i) – sets a Standard format message
  - CAN_SET_TX_WNPH (i) – sets Write Protect Bit 2
  - CAN_CLEAR_TX_WNPH (i) – clears Write Protect Bit 2
  - “i” – is the mailbox number
- Other macros
  - CAN_GET_RX_IDE(i) – gets the message format
  - CAN_GET_RX_ID(i) – gets the message ID
  - CAN_RX_DATA_BYTEx(i) – gets the received byte
  - CAN_TX_DATA_BYTEx(i) – sets a byte to transmit; applicable for the PSoC™ 3 /PSoC™ 5LP part families
  - “i” – is the mailbox number; “x” – byte number
  - CAN_TX_DATA_BYTEx(i, byte) – sets a byte to transmit; applicable for the PSoC™ 4 part family
  - “i” – is the mailbox number; “x” – is the byte number; “byte” – is the byte to transmit
  - CAN_RX_DATA_BYTE(mailbox, i) – gets the received byte
  - CAN_TX_DATA_BYTE(mailbox, i) – sets a byte to transmit
  - “mailbox” – is the mailbox number; “i” – is the byte number [0..7]; applicable for the PSoC™ 3/PSoC™ 5LP part families
  - CAN_TX_DATA_BYTE(mailbox, i, byte) – sets a byte to transmit; applicable for the PSoC™ 4 part family
  - “mailbox” – is the mailbox number; “i” – is the byte number [0..7]; “byte” is the byte to transmit

- CAN_SET_TX_ID_STANDARD_MSG(i, id) – sets Tx Msg Standart Indentifier in the CAN_TX_ID register.
- CAN_SET_TX_ID_EXTENDED_MSG(i, id) – sets Tx Msg Extended Identifier in the CAN_TX_ID register
- “i” – is the mailbox number; “id” – is the Identifier

User Macro Callbacks

Macro callbacks allow a user to execute code from the API files that are automatically generated by PSoC™ Creator. A callback requires the user to complete the following:

- Define a macro to signal the presence of a callback (in *CyAPICallbacks.h*)
- Write the function declaration (in *CyAPICallbacks.h*)
- Write the function implementation (in any user file)

Macro callbacks [5]	Associated Macro	Description
CAN_ISR_InterruptCallback()	CAN_ISR_INTERRUPT_CALLBACK	Used in the CAN_ISR() interrupt handler to perform additional application-specific actions.
CAN_ArbLostIsr_Callback()	CAN_ARB_LOST_ISR_CALLBACK	Uses in CAN_ArbLostIsr() interrupt handler to perform additional application specific actions.
CAN_OvrLdErrorIsr_Callback()	CAN_OVR_LD_ERROR_ISR_CALLBACK	Uses in CAN_OvrLdErrorIsr() interrupt handler to perform additional application specific actions.
CAN_BitErrorIsr_Callback()	CAN_BIT_ERROR_ISR_CALLBACK	Uses in CAN_BitErrorIsr() interrupt handler to perform additional application specific actions.
CAN_BitStuffErrorIsr_Callback()	CAN_BIT_STUFF_ERROR_ISR_CALLBACK	Uses in CAN_BitStuffErrorIsr() interrupt handler to perform additional application specific actions.
CAN_AckErrorIsr_Callback()	CAN_ACK_ERROR_ISR_CALLBACK	Uses in CAN_AckErrorIsr() interrupt handler to perform additional application specific actions.
CAN_MsgErrorIsr_Callback()	CAN_MSG_ERROR_ISR_CALLBACK	Uses in CAN_MsgErrorIsr() interrupt handler to perform additional application specific actions.
CAN_CrcErrorIsr_Callback()	CAN_CRC_ERROR_ISR_CALLBACK	Uses in CAN_CrcErrorIsr() interrupt handler to perform additional application specific actions.
CAN_BusOffIsr_Callback();	CAN_BUS_OFF_ISR_CALLBACK	Uses in CAN_BusOffIsr() interrupt handler to perform additional application specific actions before Component stop.
CAN_MsgLostIsr_Callback()	CAN_MSG_LOST_ISR_CALLBACK	Uses in CAN_MsgLostIsr() interrupt handler to perform additional application specific actions.

5 The macro callbacks name is contracted by the Component function name optionally appended by short explanation and “Callback” suffix.

Macro callbacks [5]	Associated Macro	Description
CAN_MsgTXIsr_Callback()	CAN_MSG_TX_ISR_CALLBACK	Uses in CAN_MsgTXIsr() interrupt handler to perform additional application specific actions.
CAN_MsgRXIsr_Callback()	CAN_MSG_RX_ISR_CALLBACK	Uses in CAN_MsgRXIsr() interrupt handler to perform additional application specific actions prior to handling Receive Message.
CAN_RtrAutoMsgSentIsr_Callback()	CAN_RTR_AUTO_MSG_SENT_ISR_CALLBACK	Uses in CAN_RtrAutoMsgSentIsr() interrupt handler to perform additional application specific actions.
CAN_StuckAtZeroIsr_Callback()	CAN_STUCK_AT_ZERO_ISR_CALLBACK	Uses in CAN_StuckAtZeroIsr() interrupt handler to perform additional application specific actions before Component stop.
CAN_SSTErrorIsr_Pre_Callback()	CAN_SST_ERROR_ISR_PRE_CALLBACK	Uses in CAN_SSTErrorIsr() interrupt handler to perform additional application specific actions before clearing the transmit mailbox.
CAN_SSTErrorIsr_Post_Callback()	CAN_SST_ERROR_ISR_POST_CALLBACK	Uses in CAN_SSTErrorIsr() interrupt handler to perform additional application specific actions after clearing the transmit mailbox.
CAN_SendMsg_0-7_Callback()	CAN_SEND_MSG_0-7_CALLBACK	Uses in CAN_SendMsg0-7() function to perform additional application specific actions.
CAN_ReceiveMsg_Callback()	CAN_RECEIVE_MSG_CALLBACK	Uses in CAN_ReceiveMsg() function to perform additional application specific actions.
CAN_ReceiveMsg_0-15_Callback()	CAN_RECEIVE_MSG_0-15_CALLBACK	Uses in CAN_ReceiveMsg0-15() function to perform additional application specific actions.

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.

MISRA Compliance

This section describes the MISRA-C: 2004 compliance and deviations for the Component. There are two types of deviations defined:

- project deviations – deviations that are applicable for all PSoC™ Creator Components
- specific deviations – deviations that are applicable only for this Component

This section provides information on Component-specific deviations. Project deviations are described in the MISRA Compliance section of the *System Reference Guide* along with information on the MISRA compliance verification environment.

The CAN Component has the following specific deviations:

MISRA-C:2004 Rule	Rule Class (Required/ Advisory)	Rule Description	Description of Deviation(s)
11.4	A	A cast should not be performed between a pointer to object type and a different pointer to object type.	Cast to pointer from 8-bit register to pointer to 16-bit register is performed to access consecutive bytes in hardware.
13.7	R	Boolean operations whose results are invariant shall not be permitted.	The value of the controlling expressions can always be invariant in some Component's configurations.
17.4	R	Array indexing shall be the only allowed form of pointer arithmetic.	Array subscripting applied to an object of pointer type. CAN Component uses pointers to structures that are mapped to the hardware registers.
19.7	A	A function should be used in preference to a function-like macro.	This is caused by a macro statements used for an applying binary mask, which can be replaced by a function, but it will result lower performance.

This Component has the following embedded Components: Clock, Interrupt. Refer to the corresponding Component datasheet for information on their MISRA compliance and specific deviations.

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™ 5LP (GCC)		PSoC™ 4 (GCC)	
	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes
Default	3737	18	2100	21	2292	2
Additional full receive buffer usage	+ 18	–	+ 24	–	+ 24	–
Additional interrupt handler enabled	+ 17	–	+ 8	–	+ 12	–

Interrupt Service Routines

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. There are several CAN Component interrupt sources:

- Message transmitted – the queued message was sent.
- Message received – a message was received.
- Receive buffer full – a new message arrived but there was nowhere to put it.
- Bus off state – CAN has reached the bus off state.
- CRC error detected – CAN CRC error was detected.
- Message format error detected – CAN message format error was detected.
- Message acknowledge error detected – CAN message acknowledge error was detected.
- Bit stuffing error detected - a bit stuffing error was detected.
- Bit error detected – a bit error was detected.
- Overload frame received – an overload frame was received.
- Arbitration lost detected – the arbitration was lost while sending a message.
- Single shot transmission failure – a buffer set for single shot transmission experienced an arbitration loss or a bus error during transmission. Not available for PSoC™ 3/PSoC™ 5LP part families.
- Stuck at zero Error – stuck at dominant error was detected. Not available for PSoC™ 3/ PSoC™ 5LP part families.
- RTR automatic reply sent – RTR auto-reply message was sent. Not available for PSoC™ 3/ PSoC™ 5LP part families.

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.

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.

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.

For PSoC™ 4200M family hardware, the interrupt line can be routed only to pin or to ISR Component.

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 mailbox with new data, without CPU intervention.

It is not possible to use DMA with PSoC™ 4200M family, as the DMA mux does not have a CAN block interrupt as its input.

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.

For the PSoC™ 4200M family, only one ISR (internal or external) could be used at once.

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.

For the PSoC™ 4 family, the Component will disable the internal interrupt if external ISR option is selected.

Functional Description

For a complete description, refer to the Controller Area Network (CAN) chapter in the appropriate device *Technical Reference Manual.*

Block Diagram and Configuration

For complete block diagram and configuration information, refer to the Controller Area Network (CAN) chapter in the appropriate device *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*

Component Debug Window

PSoC™ Creator allows you to view debug information about Components in your design. Each Component window lists the memory and registers for the instance. For detailed hardware registers descriptions, refer to the appropriate device technical reference manual.

To open the Component Debug window:

1. Make sure the debugger is running or in break mode.
2. Choose **Windows > Components…** from the **Debug** menu.
3. In the Component Window Selector dialog, select the Component instances to view and click **OK**.

The selected Component Debug window(s) will open within the debugger framework. Refer to the "Component Debug Window" topic in the PSoC™ Creator Help for more information.

Refer to the appropriate device technical reference manual for a detailed description of each register. The following registers are displayed in the CAN Component debug window.

Register Name	Description
CAN_INT_SR	CAN Interrupt Status register.
CAN_BUF_SR	CAN Rx and Tx message buffer status register.
CAN_ERR_SR	CAN Error Status register.
CAN_TX[0..7]_ID	CAN Tx message buffer identifier register.
CAN_TX[0..7]_DH	CAN Tx message buffer data high register.
CAN_TX[0..7]_DL	CAN Tx message buffer data low register.
CAN_RX[0..15]_ID	CAN Rx message buffer identifier register.
CAN_RX[0..15]_DH	CAN Rx message buffer data high register.
CAN_RX[0..15]_DL	CAN Rx message buffer data low register.
CAN_ECR *	CAN Error Capture register.

* Not available for PSoC™ 3/PSoC™ 5LP device families.

Resources

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

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 10 MHz clock	–	–	1	Mbit

Component Errata

This section lists known problems with the component.

Cypress ID	Component Version	Problem	Workaround
DRV-5877	v3.0	The macro CAN_ERROR_BUS_OFF value 0x10 is incorrect. The correct value is 0x02.	Define a new macro with mask 0x02 to use with the CAN_GetErrorState() function.

Component Changes

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

Current Version	Description of Changes	Reason for Changes / Impact
3.0.f	Updated datasheet.	Added errata to document that the CAN_ERROR_BUS_OFF macro value is incorrect.
3.0.e	Updated datasheet.	Corrected parameters names in the CAN_SetOpMode() API.
3.0.d	Updated datasheet.	Removed note that PSoC™ 4200L characterization data is preliminary.
3.0.c	Updated datasheet.	Added note that PSoC™ 4200L characterization data is preliminary.
3.0.b	Updated datasheet.	Added certification statement.
3.0.a	Updated datasheet.	Updated Macro Callbacks section.
3.0	Added PSoC™ 4200M device support.	Added Option to select Endianness of Data Bytes in the customizer. Added option to select single shot transmission of Tx messages. Added 3 additional interrupt enable options and ISRs for their handling: SST Failure, Stuck at 0, RTR Auto Reply Message sent. Added functions to set/arm/read the Error Capture Register. Modified CAN_SetOperatingMode() function to include options for internal and external loopback.
	Improved the timing calculations to take into account tolerance of source clock.	
	Added Macros to link mailbox names with mailbox numbers.	
	Added User Macro Callbacks support.	

Current Version	Description of Changes	Reason for Changes / Impact
	Changed required minimum clock value for support maximum CAN baud rate to 10 MHz.	Disabled invalid bit timing configuration.
	Enabled RTR fields for Basic CAN.	Enabled RTR option for basic CAN mailboxes to enable implementation of RTR messages with Basic CAN mailboxes.
	Removed restriction on message IDs in the Component GUI.	Removing ID restriction 0x000 from the Component GUI allows users to use the messages with the highest priority.
2.30.a	Updated the datasheet.	Added Component Errata section to document that the Component was changed, but there is no impact to designs.
		Removed references to obsolete PSoC™ 5 device.
2.30	Added MISRA Compliance section.	The Component has specific deviations described.
	Updated API functions parameters descriptions.	Incomplete API functions parameters descriptions.
2.20	CAN_Start() function internal update.	CAN_Start() function was not enabling the Component if it was previously initialized and stopped.
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. 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 PSoC™ 3 CAN Component

Current Version	Description of Changes	Reason for Changes / Impact
	Added ConnectExtInterruptLine, IntISRDisable, FullCastomIntISR, AdvancedInterruptTab parameters and ISR helper parameters in the Component symbol	Updated Marketing Requirements Document MRD for the PSoC™ 3 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	
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.

© Cypress Semiconductor Corporation, 2015-2022. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC (“Cypress”). This document, including any software or firmware included or referenced in this document (“Software”), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress’s patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical Components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage (“Unintended Uses”). A critical Component is any Component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.