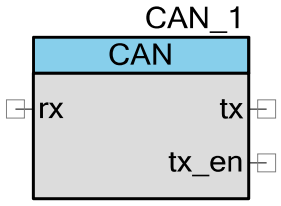
Controller Area Network (CAN)

1.50

Features



- CAN2.0 A/B protocol implementation, ISO 11898-1 compliant
- Programmable bit rate up to 1 Mbps @ 8 MHz (BUS_CLK)
- Two 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

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 the efficient implementation of 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

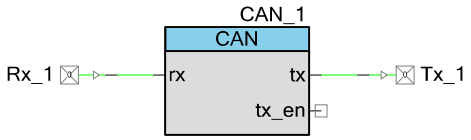
CAN bus transmit signal, (connected to CAN TX bus of external transceiver).

tx_en – Output *

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

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 Output Pin components. The Pins components are also configured with default settings, except Input Synchronized is set to false in the Input Pin component.



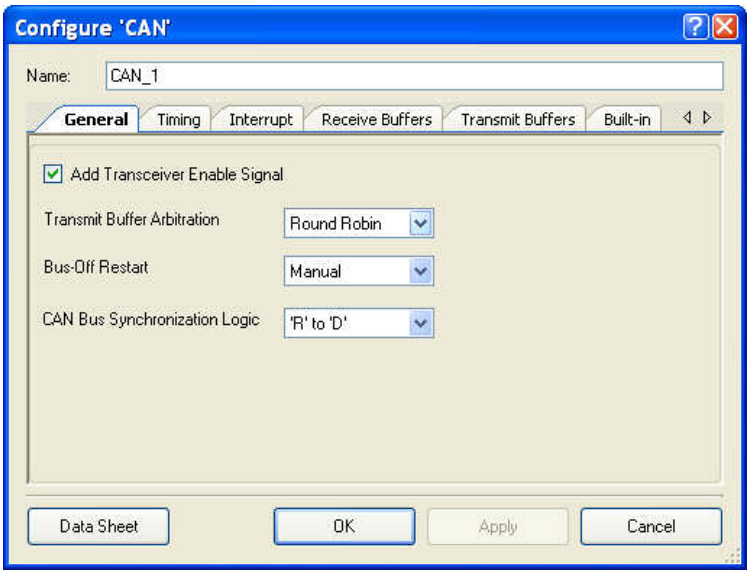
Parameters and Setup

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 were 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/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 it is guaranteed that they are sent first.

Bus-Off Reset

Used to configure the reset type:

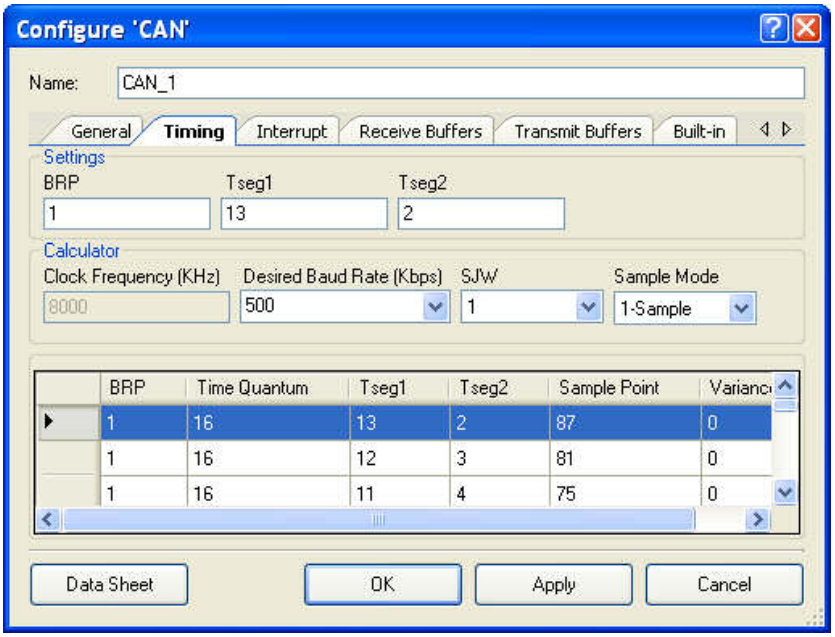
- Manual (default) – After bus-off, the CAN must be restarted by the user. This is the recommended setting.
- Automatic – After bus-off, the CAN controller restarts automatically after 128 groups of 11 recessive bits.

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 – Both edges are used for synchronization

Timing Tab

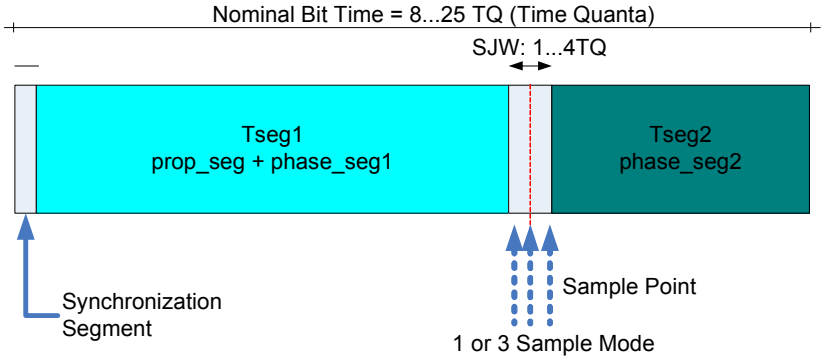


The **Timing** tab contains the following settings:

Settings

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

The following shows the CAN bit timing representation:



Calculator

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

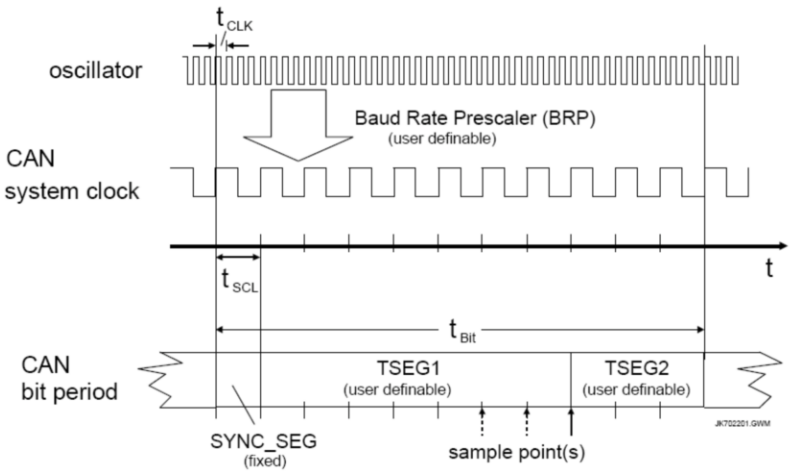
Table

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.

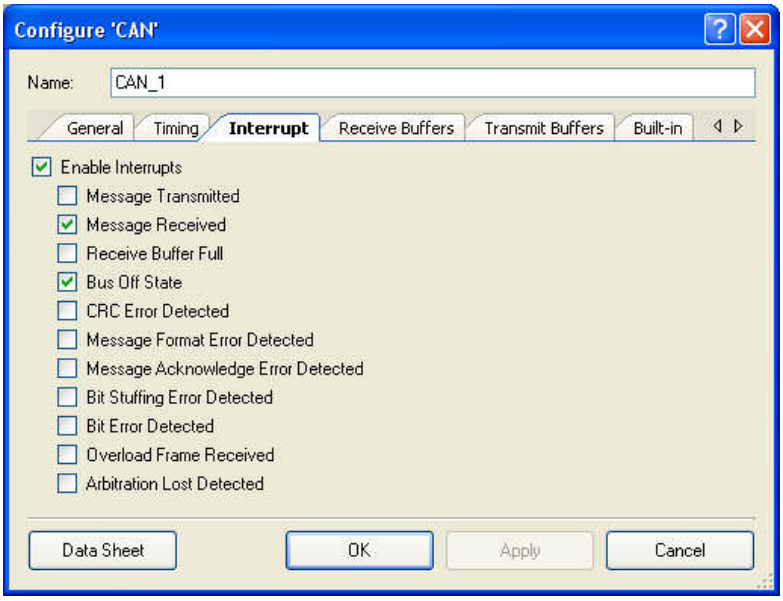
Note Incorrect bit timing settings might cause the CAN controller to remain in an error state.

The following diagram from the Philips Semiconductor Applications Note (AN97046) shows how all timing is derived from the oscillator.



Note: t_{SCL} is the time duration of one Time Quantum (TQ)

Interrupt Tab



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

Enable Interrupts

Enable\Disable Global interrupts from the CAN Controller. Enabled by default.

Message Transmitted

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

CRC Error Detected

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

Message Format Error Detected

Enable\Disable message format error interrupt. Indicates that a CAN message format error was detected. Disabled by default.

Message Acknowledge Error Detected

Enable\Disable message acknowledge error interrupt. Indicates that a CAN message acknowledge error was detected. Disabled by default.

Bit Stuffing Error Detected

Enable\Disable bit stuffing error interrupt. Indicates that a bit stuffing error was detected. Disabled by default.

Bit Error Detected

Enable\Disable bit error interrupt. Indicates that bit error was detected. Disabled by default.

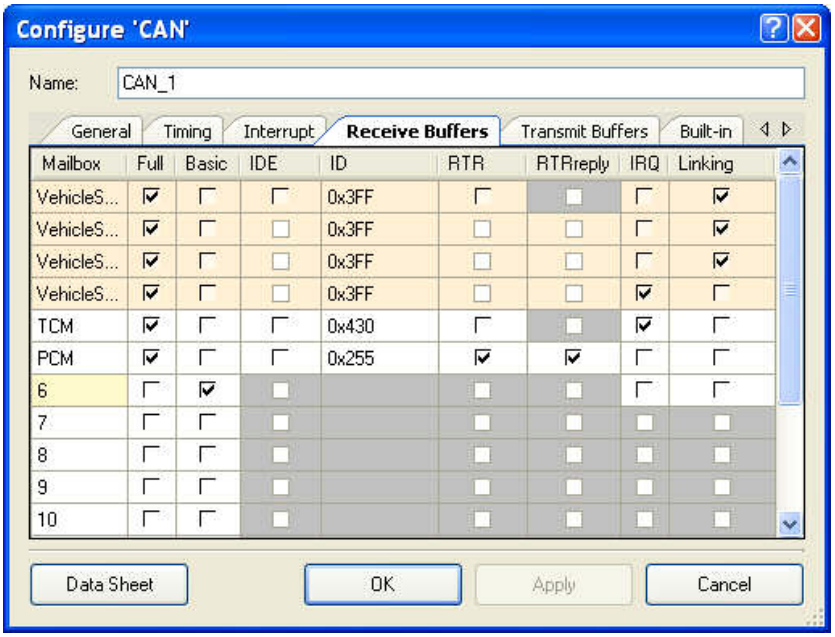
Overload Frame Received

Enable\Disable overload interrupt. Indicates that an overload frame was received. Disabled by default.

Arbitration Lost Detected

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

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, _". If an incorrect name is entered, a message displays and **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 – 15
- IDE = Unchecked
- ID = 0x001
- RTR = Unchecked
- RTRreply = Unchecked and locked (only enable when RTR = Checked)
- IRQ = Checked, only available if Message Received (Interrupt tab) interrupt is checked
- Linking = Unchecked

Basic

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

- IDE = Unchecked(unavailable)
- ID = <All>(unavailable)
- RTR = Unchecked (unavailable)
- RTRreply = Unchecked (unavailable)
- IRQ = Unchecked, only available if Message Received interrupt is checked
- Linking = Unchecked.

IDE

When the IDE box is unchecked the identifier shall be limited to 11 bits (0x001 to 0x7FE). When the IDE box is checked the identified shall be limited to 29 bits (0x00000001 to 0x1FFFFFFE).

RTR - Remote Transmission Request

Only available for mailboxes setup to receive Full CAN messages. When checked will configure 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 setup to receive Full CAN messages, with the RTR bit set. When checked, will automatically reply 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 unchecked 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 can not 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

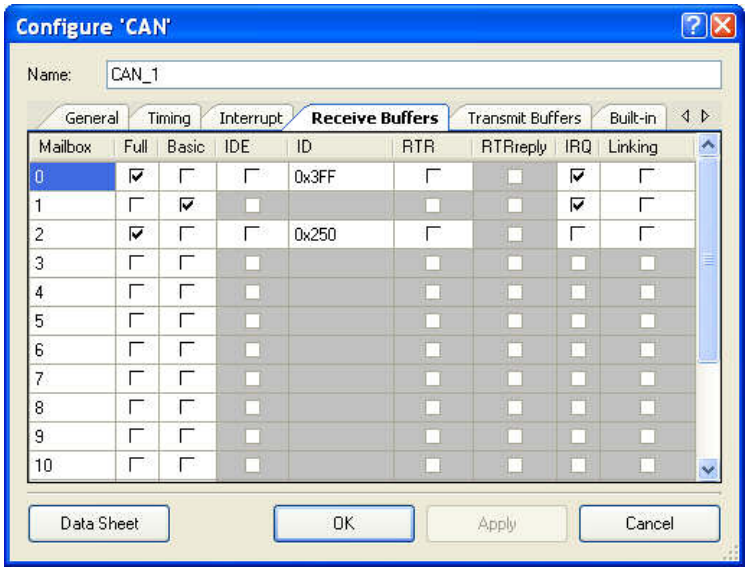
Every "Full" RX mailbox has a predefined API. The function list can be found in the *CAN_1_TX_RX_func.c* project file. These functions are conditionally compiled depending on the receive mailbox setting. Only mailboxes defined as "Full" will have their respective functions compiled.

The macro identifier CAN_1_RX*x*_FUNC_ENABLE defines whether a function is compiled or not. Defines can be found 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 upon receipt of a message 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 which received the message.

Receive Buffers Configuration

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



CAN_1.h file:

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

CAN_1_TX_RX_func.c file:

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


    /* `#END` */

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

}
#endif

#if(CAN_1_RX2_FUNC_ENABLE)
void CAN_1_ReceiveMsg2(void)
{
    /* `#START MESSAGE_2_RECEIVED` */

    /* `#END` */
CAN_1_RX[2].rxcmd.byte[0] |= CAN_1_RX_ACK_MSG;
}
#endif
```

The following function will be called for CAN Receive Message 1, configured as Basic CAN with interrupt enabled.

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


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

CAN_1_INT.c file

```
void CAN_1_MsgRXIsr(void)
{
…
/* RX Full mailboxes handler */
    switch(i)
    {
        case 0 : CAN_1_ReceiveMsg0();
            break;
        case 2 : CAN_1_ReceiveMsg2();
            break;
        default:
            break;
    }
…
}
```

In case Linking is implemented, conditional compile applies to the mailbox with the IRQ flag set. All receive functions will 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 respective ACR bit.

‘1’ The incoming bit is doesn’t care.

For example, to setup the mailbox to receive a range of IDs of 0x180 – 187, IDE = 0 (unchecked), RTR = 0 (unchecked), mailbox 5, do the following additional actions:

1. Take low range of ID, for instance is 0x180 and IDE and RTR accordingly. For this ID, IDE and RTR values AMR and ACR register is set to values:
   - ACR[31-21] = 0x180 AMR[31-21] = 0x0;
   - ACR[20-3] = 0x0 AMR[20-3] = 0x3FFFF - all ones;
   - ACR[2] = 0 AMR[2] = 0
   - ACR[1] = 0 AMR[1] = 0
   - ACR[0] = 0 AMR[0] = 0
2. Define common part of range ID (11bit):
   - 0x180 = 0`b001 1000 0000
   - 0x187 = 0`b001 1000 0111
   - Mask = 0`b001 1000 0XXX
   - ACR[31-21] = 0'b000 0000 0111

   You must put 1s instead of "XXX" and nulls instead common bits into AMR register. So next values are:
   - ACR[31-21] = 0x180 AMR[31-21] = 0x7;
   - ACR[20-3] = 0x0 AMR[20-3] = 0x3FFFF - all ones;
   - ACR[2] = 0 AMR[2] = 0
   - ACR[1] = 0 AMR[1] = 0
   - ACR[0] = 0 AMR[0] = 0
3. Use function "CAN_1_RXRegisterInit" to write AMR register of mailbox number 5:

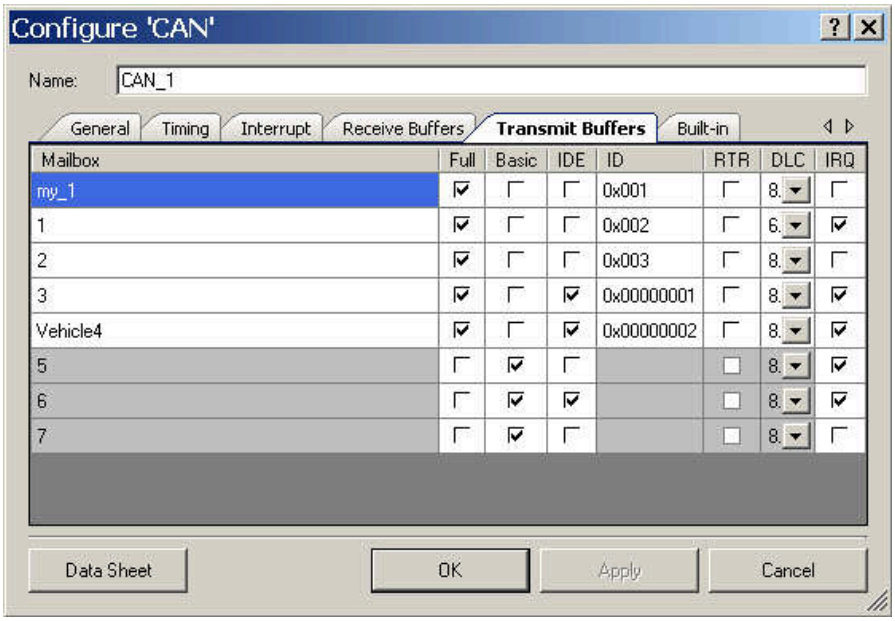
```
uint8 result = FAIL;
uint16 temp_amr;
uint16 temp_acr;

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

if (CAN_1_RXRegisterInit(&CAN_1_RX[5].rxamr, temp_amr))
{
    if ((CAN_1_RXRegisterInit(&CAN_1_RX[5].rxacr, temp_acr))
    result = SUCCESS;
}
return result;  /* error – if return no zero value; */
```

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



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, _". If you enter an incorrect name the Mailbox field, it will revert 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 – 7
- IDE = Unchecked
- ID = 0x01
- RTR = Unchecked
- DLC = 8,
- IRQ = Unchecked

Basic

Default "Basic" checkbox is set for all of the mailboxes. If "Basic" is selected then the options ID, RTR, and DLC are unavailable. If Basic is selected, the required CAN message fields should be entered via code. Default selections are placed with the following options:

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

IDE

If the IDE box is unchecked then the identifier cannot be greater then 11bits (from 0x001 to 0x7FE). If the IDE box is checked then 29bit identifier is allowed (from 0x00000001 to 0x1FFFFFFE). The user may not 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 is unchecked and when selecting checking the IRQ, the message text appears: "Global "Message Transmitted Interrupt" is disabled. Do you wish to enable it?" and the selections are Yes – No – Cancel.

- Yes – Check IRQ and check the Message Transmitted Interrupt box in the Interrupt tab.
- No or Cancel – Check IRQ and the Message Transmitted Interrupt box in the Interrupt tab remains unchecked.

CAN TX Functions

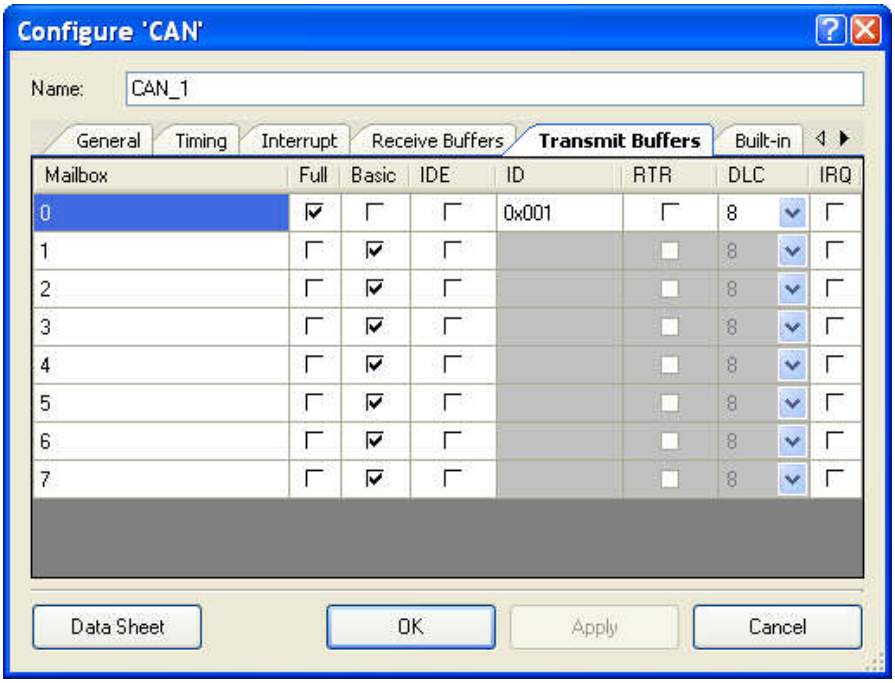
Every "Full" TX mailbox has a predefined API. The function list can be found in the *CAN_1_TX_RX_func.c* project file. These functions are conditionally compiled depending on the transmit mailbox setting. Only mailboxes defined as "Full" will have their respective functions compiled.

The macro identifier CAN_1_TX*x*_FUNC_ENABLE defines whether a function is compiled or not. Defines can be found in the *CAN_1.h* project file.

The CAN_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.



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 = SUCCESS;
    if (CAN_1_TX[0u].txcmd.byte[0u] & CAN_1_TX_REQUEST_PENDING) ==
        CAN_1_TX_REQUEST_PENDING)
```

```
        {
            result = CAN_FAIL;
        }
        else
        {
            /* `#START MESSAGE_0_TRANSMITTED` */

            /* `#END` */

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

    return result;
    }
    #endif
```

The common function provided for all "Basic" Transmit mailboxes:

```
uint8 CAN_1_SendMsg(CAN_TX_MSG *message)
```

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

- ID, the restriction if the ID slot includes: 1) for a standard message (IDE = 0) identifier limited to 11 bits (0x001 to 0x7FE). 2) 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 – 8, 9 – 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 via the CAN_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 3). When all retries fail, an indication of "FAIL" is returned to the application.

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

```
typedef struct _CAN_TX_MSG
{
    uint32 id;
    uint8 rtr;
    uint8 ide;
    uint8 dlc;
```

```
    uint8 irq;
    CAN_DATA_BYTES_MSG *msg;
} CAN_TX_MSG;
```

The CAN_DATA_BYTES structure contains 8 bytes of data in a message.

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

Placement

The CAN component will be placed in the fixed function CAN block.

Resources

Resources	Digital Blocks					API Memory (Bytes)		Pins (per External I/O)
	Datapaths	Macro cells	Status Registers	Control Registers	Counter7	Flash	RAM	
CAN fixed HW *	N/A	N/A	N/A	N/A	N/A	2930	6	3

*The CAN component utilizes the dedicated CAN hardware block in the silicon.

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, CyDesigner 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
uint8 CAN_Start(void)	Sets the initVar variable, calls the Init function, and then calls the Enable function.
uint8 CAN_Stop(void)	Disables the CAN.
uint8 CAN_GlobalIntEnable(void)	Enables Global Interrupts from CAN component.
uint8 CAN_GlobalIntDisable(void)	Disables Global Interrupts from CAN component.
uint8 CAN_SetPreScaler (uint16 bitrate)	Sets PreScaler for generation of the time quanta from the BUS_CLK.
uint8 CAN_SetArbiter (uint8 arbiter)	Sets arbitration type for transmit buffers
uint8 CAN_SetTsegSample (uint8 cfgTseg1, uint8 cfgTseg2, uint8 sjw, uint8 sm)	Configures: Time segment 1, Time segment 2, Synchronization Jump Width, and Sampling Mode.
uint8 CAN_SetRestartType (uint8 reset)	Sets Reset type.
uint8 CAN_SetEdgeMode (uint8 edge)	Sets Edge Mode.
uint8 CAN_RXRegisterInit (reg32 * regAddr, uint32 config)	Writes only receive CAN registers.
uint8 CAN_SetOpMode (uint8 opMode)	Sets Operation Mode.
uint8 CAN_GetTXErrorflag (void)	Returns the flag that indicates if the number of TX errors exceeds 0x60.
uint8 CAN_GetRXErrorflag (void)	Returns the flag that indicates if the number of RX errors exceeds 0x60.
uint8 CAN_GetTXErrorCount (void)	Returns the number of Transmit Errors.
uint8 CAN_GetRXErrorCount (void)	Returns the number of Receive Errors.
uint8 CAN_GetErrorState (void)	Returns error status of the CAN component.
uint8 CAN_SetIrqMask (uint16 mask)	Sets to enable/disable particular interrupt sources.
void CAN_ArbLostIsr(void)	Clears Arbitration Lost interrupt flag.
void CAN_OvrLdErrrorIsr(void)	Clears Overload Error interrupt flag.
void CAN_BitErrorIsr(void)	Clears Bit Error interrupt flag.
void CAN_BitStuffErrorIsr(void)	Clears Bit Stuff Error interrupt flag.
void CAN_AckErrorIsr(void)	Clears Acknowledge Error interrupt flag.
void CAN_MsgErrorIsr(void)	Clears Form Error interrupt flag.
void CAN_CrcErrorIsr(void)	Clears CRC Error interrupt flag.
void CAN_BusOffIsr(void)	Places CAN Component to Stop mode.

Function	Description
void CAN_MsgLostIsr(void)	Clears Message Lost interrupt flag.
void CAN_MsgTXIsr(void)	Clears Transmit Message interrupt flag.
void CAN_MsgRXIsr(void)	Clears Receive Message interrupt flag and call appropriate handlers for Basic and Full interrupt based mailboxes.
uint8 CAN_RxBufConfig (CAN_RX_CFG * rxConfig)	Configures all receive registers for particular mailbox.
uint8 CAN_TxBufConfig (CAN_TX_CFG * txConfig)	Configures all transmit registers for particular mailbox.
uint8 CAN_SendMsg (CAN_TX_MSG * message)	Send Message from one of Basic mailboxes.
uint8 CAN_SendMsg0-7(void)	Function checks if mailbox 0-7 doesn't already have un-transmitted messages waiting for arbitration.
void CAN_TxCancel (uint8 bufferId)	Cancel transmission of a message that has been queued for transmitted.
void CAN_ReceiveMsg0-15(void)	Acknowledges receipt of new message.
void CAN_ReceiveMsg (uint8 rxMailbox)	Clears Receive particular Message interrupt flag.
void CAN_Sleep(void)	Prepare CAN component to go to sleep
void CAN_Wakeup(void)	Prepare CAN component to wake up
uint8 CAN_Init(void)	Initializes or restores the CAN per the Configure dialog settings.
void CAN_Enable(void)	Enables the CAN.
void CAN_SaveConfig(void)	Save the current configuration.
void CAN_RestoreConfig(void)	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 reinitialization after the first call to the CAN_Start() routine. If reinitialization 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: This is the preferred method to begin component operation. 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 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 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 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. Value 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 arbitration type for transmit buffers. Types of arbiters are Round Robin and Fixed priority. Value 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, value between 0x2 and 0xF are valid

(uint8) cfgTseg2: Time segment 2, value between 0x1 and 0x7 are valid

(uint8) sjw: Synchronization Jump Width, value 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 recommended setting. Value 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. Value 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. Operations Modes are Active of Listen Only. Value 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 TX errors exceeds 0x60.

Parameters: None

Return Value: (uint8) Indication whether the number of TX errors exceeds 0x60.

Side Effects: None

uint8 CAN_GetRXErrorflag(void)

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

Parameters: None

Return Value: (uint8) Indication whether the number of TX 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 error status of the CAN component.

Parameters: None

Return Value: (uint8) Error status.

Side Effects: None

uint8 CAN_SetIrqMask(uint16 mask)

Description: This function sets to enable/disable particular interrupt sources. Interrupt Mask directly writes to the CAN Interrupt Enable register.

Parameters: (uint8) request: Interrupt enable/disable request. 1 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 entry point to Arbitration Lost Interrupt. Clears Arbitration Lost interrupt flag. Only generated if Arbitration Lost Interrupt parameter is enabled.

Parameters: None

Return Value: None

Side Effects: None

void CAN_OvrLdErrorIsr(void)

Description: This function is entry point to Overload Error Interrupt. Clears Overload Error interrupt flag. Only generated if Overload Error Interrupt parameter is enabled.

Parameters: None

Return Value: None

Side Effects: None

void CAN_BitErrorIsr(void)

Description: This function is entry point to Bit Error Interrupt. Clears Bit Error interrupt flag. Only generated if Bit Error Interrupt parameter is enabled.

Parameters: None

Return Value: None

Side Effects: None

void CAN_BitStuffErrorIsr(void)

Description: This function is entry point to Bit Stuff Error Interrupt. Clears Bit Stuff Error interrupt flag. Only generated if Bit Stuff Error Interrupt parameter is enabled.

Parameters: None

Return Value: None

Side Effects: None

void CAN_AckErrorIsr(void)

Description: This function is entry point to Acknowledge Error Interrupt. Clears Acknowledge Error interrupt flag. Only generated if Acknowledge Error Interrupt parameter is enabled.

Parameters: None

Return Value: None

Side Effects: None

void CAN_MsgErrorIsr(void)

Description: This function is entry point to Form Error Interrupt. Clears Form Error interrupt flag. Only generated if Form Error Interrupt parameter is enabled.

Parameters: None

Return Value: None

Side Effects: None

void CAN_CrcErrorIsr(void)

Description: This function is entry point to CRC Error Interrupt. Clears CRC Error interrupt flag. Only generated if CRC Error Interrupt parameter is enabled.

Parameters: None

Return Value: None

Side Effects: None

void CAN_BusOffIsr(void)

Description: This function is entry point to Buss Off Interrupt. Places CAN Component to Stop mode. Only generated if Bus Off Interrupt parameter is enabled. Recommended setting to enable this interrupt.

Parameters: None

Return Value: None

Side Effects: Stop operation of CAN component.

void CAN_MsgLostIsr(void)

Description: This function is entry point to Message Lost Interrupt. Clears Message Lost interrupt flag. Only generated if Message Lost Interrupt parameter is enabled.

Parameters: None

Return Value: None

Side Effects: None

void CAN_MsgTXIsr(void)

Description: This function is entry point to Transmit Message Interrupt. Clears Transmit Message interrupt flag. Only generated if Transmit Message Interrupt parameter is enabled.

Parameters: None

Return Value: None

Side Effects: None

void CAN_MsgRXIsr(void)

Description: This function is entry point to Receive Message Interrupt. Clears Receive Message interrupt flag and call appropriate handlers for Basic and Full interrupt based mailboxes. Only generated if Receive Message Interrupt parameter is enabled. Recommended setting to enable this interrupt.

Parameters: None

Return Value: None

Side Effects: None

uint8 CAN_RxBufConfig(CAN_RX_CFG *rxConfig)

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

Parameters: (CAN_RX_CFG *) rxConfig: Pointer to structure that contain all required values to configure all receive registers for 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 particular mailbox. Mailbox number contains CAN_TX_CFG structure.

Parameters: (CAN_TX_CFG *) txConfig: Pointer to structure that contain all required values to configure all transmit registers for 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 for the first free available and sends from it. The number of retries is limited to 3.

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 entry point to Transmit Message 0-7. This function checks if mailbox 0-7 doesn't already have un-transmitted messages waiting for arbitration. If not, 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 cancel transmission of a message that has been queued for transmitted. 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 entry point to Receive Message 0-15 Interrupt. Clears Receive Message 0 - 15 interrupt flag. Only generated Receive mailbox designed as Full interrupt based.

Parameters: None

Return Value: None

Side Effects: None

void CAN_ReceiveMsg(uint8 rxMailbox)

Description: This function is entry point to Receive Message Interrupt for Basic mailboxes. Clears Receive particular Message interrupt flag. Only generated if one of Receive mailboxes 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.
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 will re-initialize the component with the following exceptions. It will not clear data from the mailboxes.
Enable 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. This will save non-retention registers. This function will also save 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. This will restore non-retention registers. This function will also restore 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 numerous 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 no where to put it
- Transmit Message – the queued message was sent
- Receive Message – a message was received

All 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.

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

DC and AC Electrical Characteristics

The following values indicate expected performance and are based on initial characterization data.

CAN DC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
	Block current consumption	500 kbps	--	--	285	µA
		1 Mbps	--	--	330	µA

CAN 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
1.50.a	Added characterization data to datasheet	
	Minor datasheet edits and updates	
1.50	Added CAN_Sleep() / CAN_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.
1.30.b	Added information to the component that advertizes its compatibility with silicon revisions.	The tool reports an error/warning if the component is used on incompatible silicon. If this happens, update to a revision that supports your target device.
1.30.a	Moved local parameters to formal parameter list.	To address a defect that existed in PSoC Creator v1.0 Beta 4.1 and earlier, the component was updated so that it could continue to be used in newer versions of the tool. This component used local parameters, which are not exposed to the user, to do background calculations on user input. These parameters have been changed to formal parameters which are visible, but un-editable. There are no functional changes to the component but the affected parameters are now visible in the “expression view” of the customizer dialog.
1.30	Updated the Configure dialog. Removed functions from ISR component.	Fixed an issue with the Configure CAN dialog not refreshing after an edit operation, as well as a formatting issue. Changed the CAN component to use the library functions for set interrupt vector and priority from the *CyLib.h* file APIs instead of functions from the ISR component. Functions from the ISR component are no longer generated.

© Cypress Semiconductor Corporation, 2010. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC® is a registered trademark, and PSoC Creator™ and Programmable System-on-Chip™ are trademarks of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress’ product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.