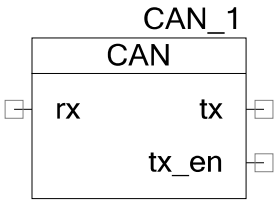
Controller Area Network (CAN)
1.10



Features

- CAN2.0 A/B protocol implementation, ISO 11898 compliant
- Programmable bit rate up to 1Mbps @ 8MHz
- Supports 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 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/O's states that the I/O may be hidden on the symbol under the conditions listed in the description of that I/O.

rx – Input

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

tx– Output

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

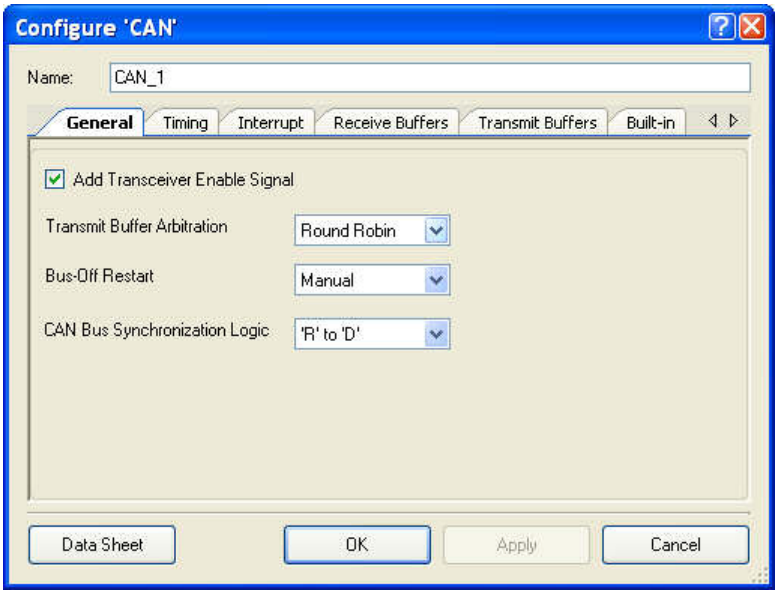
tx_en – Output *

External transceiver enable signal.

Parameters and Setup

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

Configure CAN Dialog – General Tab



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

Add Transceiver Enable Signal

Enables/disables the use of tx_enable signal wire 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:

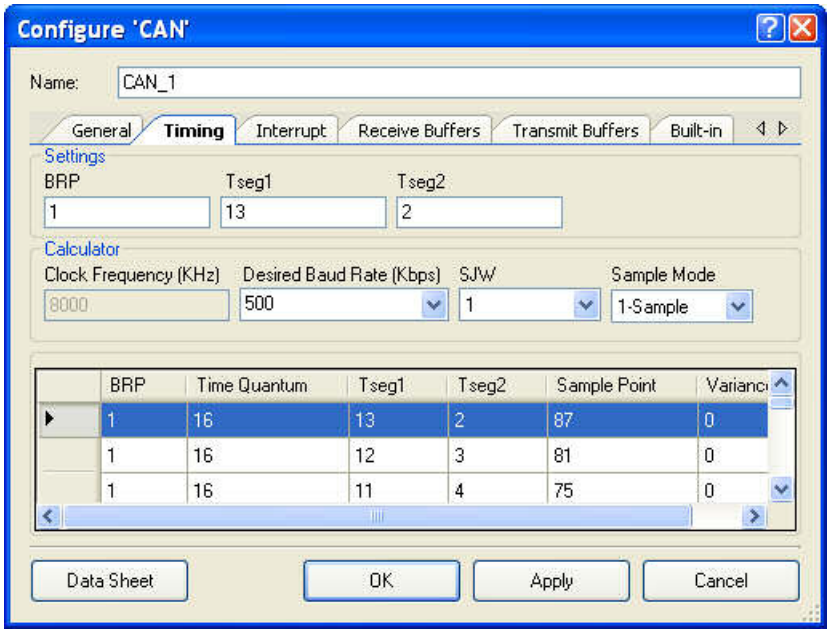
- Manual (default) – After bus-off, the CAN must be restarted by the user. This is the recommended setting.
- Automatic – After bus-off, the CAN controller is restarting automatically after 128 groups of 11 recessive bits.

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

Configure CAN Dialog – 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 cfg_tseg1 = Tseg1– 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 (1Sample). cfg_tseg2 = Tseg2 – 1

Calculator

- Clock Frequency – the CAN Component input clock is equal to BUS_CLK and can be modified in the “Clocks” tab found in the *project_name.cydwr* file.
- Desired Baud Rate in Kbps
- SJW – configuration of synchronization jump width (2 bits). This value must be less then or equal to cfg_tseg2 and cfg_tseg1. User selected value 1, 2, 3, or 4.
- Sample Mode – configuration of sampling mode. User selected value 1–Sample or 3–Sample.

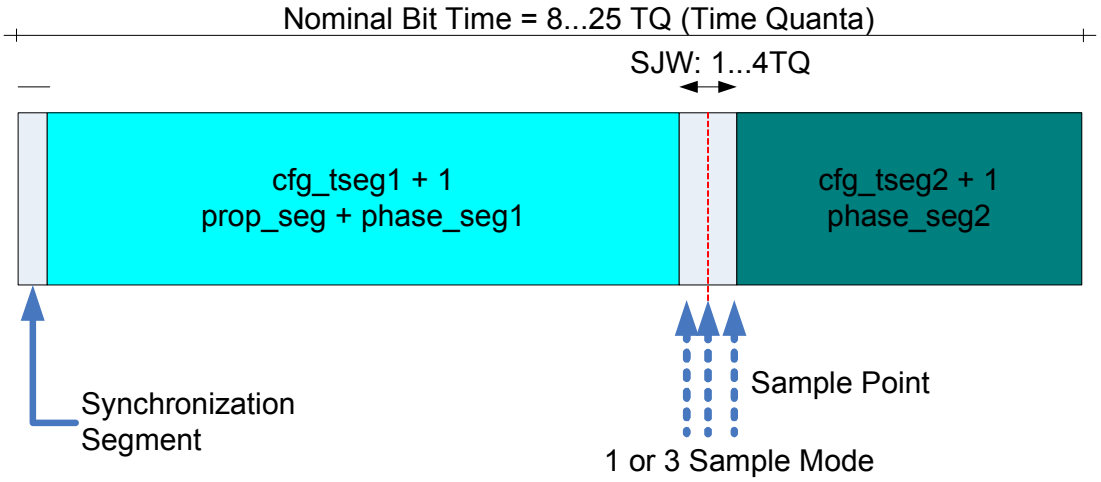
Table

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.

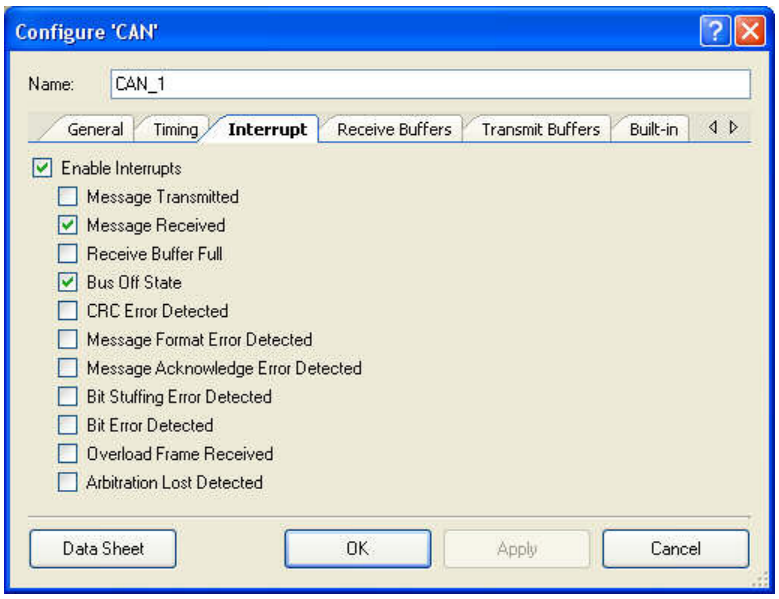
You may also choose to manually enter values for Tseg1, Tseg2 and BRP in the provided input boxes.

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

The following shows the CAN bit timing representation:



Configure CAN Dialog – Interrupt Tab



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

Enable Interrupts

Enable/Disable Global interrupts from CAN Controller. Enabled by default.

Message Transmitted

Enable/Disable message transmitted interrupts. 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” Interrupt box, and uncheck all individual transmit buffer interrupts in the Transmit Tab.
- No (default) – Uncheck the “Message Transmitted” Interrupt box, and keep all individual transmit buffer interrupts in the Transmit Tab as they are.
- Cancel –No changes are made.

Message Received

Enable/Disable message received interrupts. 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” Interrupt box, and uncheck all individual receive buffer interrupts in the receive Tab.

- No – Uncheck the “Message Received” Interrupt box, and keep all individual receive buffer interrupts in the receive 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 has reached the bus off state. Enabled by default.

CRC Error Detected

Enable/Disable of CRC error interrupt. Indicates that the CAN CRC error was detected. Disabled by default.

Message Format Error Detected

Enable/Disable message format error interrupt. Indicates that the CAN message format error was detected. Disabled by default.

Message Acknowledge Error Detected

Enable/Disable message acknowledge error interrupt. Indicates that the CAN message acknowledge error was detected. Disabled by default.

Bit Stuffing Error Detected

Enable/Disable bit stuffing error interrupt. Indicates that 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.

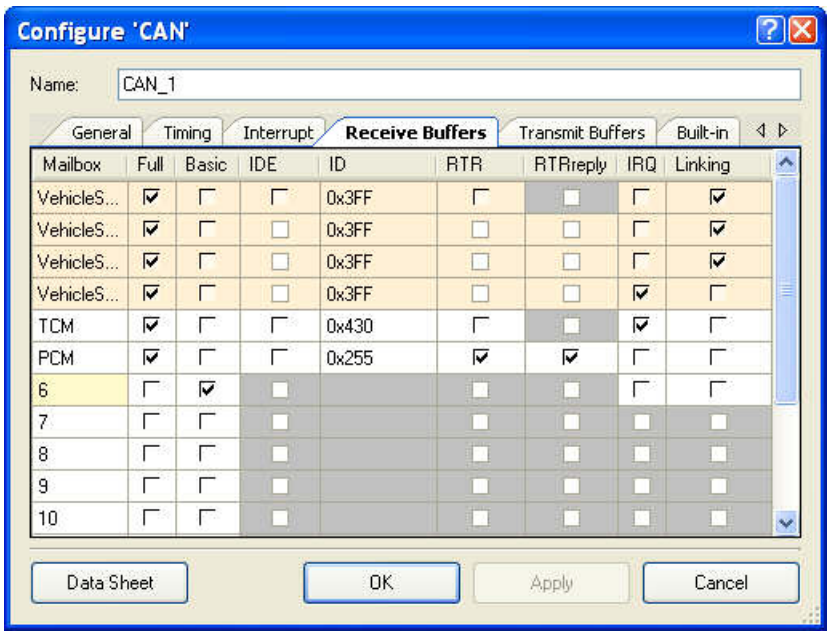
Overload Frame Received

Enable/Disable overload interrupt. Indicates that 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.

Configure CAN Dialog – 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 are locked for all disabled mailboxes.

For “Full” mailboxes, the Mailbox field is editable to allow the user 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, the message Mailbox field returns to the default value.

A full mailbox cannot be setup after a basic mailbox. The customizer limits the setup of full mailboxes once a basic mailbox has been enabled.

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. Functionality linked to the Message Received (Interrupt tab) interrupt status
- Linking = Unchecked

Basic

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

- IDE = Unchecked(unavailable)
- ID = <All>(unavailable)
- RTR = Unchecked (unavailable)
- RTRreply = Unchecked (unavailable)
- IRQ = Unchecked. Functionality linked to the “Message Received” (Interrupt tab) interrupt status
- Linking = Unchecked.

IDE

When the IDE box is unchecked the identifier shall be limited to11bits (0x001 to 0x7FE). When the IDE box is checked the identified shall be limited to 29bits (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 – Check IRQ and check the Message Received Interrupt box in the Interrupt tab.
- No or Cancel – Check IRQ and the Message Received Interrupt box in the Interrupt tab remains unchecked.

Linking

The Linking check box allows the linking of several sequential receive mailboxes to create the receive mailboxes array which acts like a receive FIFO. All mailboxes of the same array must have the same message filter settings (AMR and ACR are identical).

- The last mailbox of an array may not have its link flag set.
- 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 Function

Every “Full” RX mailbox has a predefined API provided to the user. Function list can be found in CAN_1_TX_RX_func.c file included in the project. These functions are conditionally compiled depending on the receive mailbox setting. Only mailboxes defined as “Full” shall have their respective functions compiled.

CAN_1_RX*x*_FUNC_ENABLE – Defines whether function shall be compiled or not. Defines can be found in CAN_1.h project file.

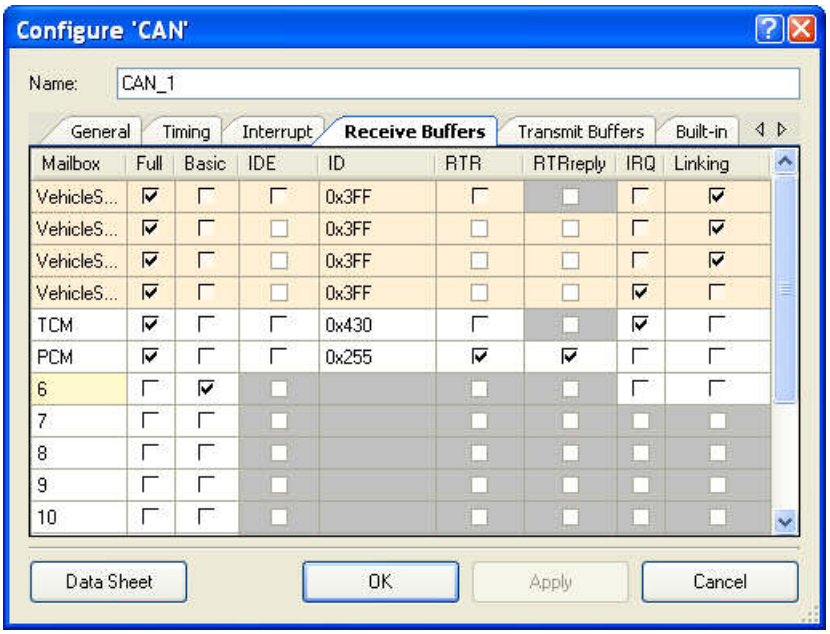
CAN_1_MsgRXIsr(void) loops through all receive mailboxes and checks their respective “Message Available Flag” (MsgAv) and “Interrupt Enable” (IRQ) for a successfully received CAN message.

Upon receipt, the CAN_ReceiveMsg_*x* function is called – where _*x* indicates the “Full“ CAN mailbox number or user defined name.

For all interrupt based “Basic” CAN mailbox, void CAN_ReceiveMsg(uint8 rxreg) function is called – where rxreg indicates the number of the mailbox which received the message.

Below is an example to illustrate the use of the above-described API.

Figure 1 - Receive Buffers Configuration



CAN_1.h file:

```
...
#define CAN_1_RX3_FUNC_ENABLE 1
#define CAN_1_RX4_FUNC_ENABLE 1
#define CAN_1_RX5_FUNC_ENABLE 1
#define CAN_1_RX0_FUNC_ENABLE 0
#define CAN_1_RX1_FUNC_ENABLE 0
#define CAN_1_RX2_FUNC_ENABLE 0
#define CAN_1_RX6_FUNC_ENABLE 0
#define CAN_1_RX7_FUNC_ENABLE 0
#define CAN_1_RX8_FUNC_ENABLE 0
#define CAN_1_RX9_FUNC_ENABLE 0
#define CAN_1_RX10_FUNC_ENABLE 0
#define CAN_1_RX11_FUNC_ENABLE 0
#define CAN_1_RX12_FUNC_ENABLE 0
#define CAN_1_RX13_FUNC_ENABLE 0
#define CAN_1_RX14_FUNC_ENABLE 0
#define CAN_1_RX15_FUNC_ENABLE 0
...
```

CAN_1_TX_RX_func.c file:

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


        /* `#END` */

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

```
}
#endif
#if(CAN_1_RX1_FUNC_ENABLE)
void CAN_1_ReceiveMsg1(void)
{
        /* `#START MESSAGE_2_RECEIVED` */

        /* `#END` */
CAN_1_RX[1].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

#if(CAN_1_RX3_FUNC_ENABLE)
void CAN_1_ReceiveMsgVehicleSpeed(void)
{
        /* `#START MESSAGE_VehicleSpeed_RECEIVED` */

        /* `#END` */
CAN_1_RX[0].rxcmd.byte[0] |= CAN_1_RX_ACK_MSG;
CAN_1_RX[1].rxcmd.byte[0] |= CAN_1_RX_ACK_MSG;
CAN_1_RX[2].rxcmd.byte[0] |= CAN_1_RX_ACK_MSG;
CAN_1_RX[3].rxcmd.byte[0] |= CAN_1_RX_ACK_MSG;
}
#endif

#if(CAN_1_RX4_FUNC_ENABLE)
void CAN_1_ReceiveMsgTCM(void)
{
        /* `#START MESSAGE_TCM_RECEIVED` */

        /* `#END` */
CAN_1_RX[4].rxcmd.byte[0] |= CAN_1_RX_ACK_MSG;
}
#endif
#if(CAN_1_RX5_FUNC_ENABLE)
void CAN_1_ReceiveMsgPCV(void)
{
        /* `#START MESSAGE_PCV_RECEIVED` */

        /* `#END` */
CAN_1_RX[5].rxcmd.byte[0] |= CAN_1_RX_ACK_MSG;
}
#endif
```

```
#if(CAN_1_RX6_FUNC_ENABLE)
void CAN_1_ReceiveMsg6(void)
{
        /* `#START MESSAGE_6_RECEIVED` */

        /* `#END` */
CAN_1_RX[6].rxcmd.byte[0] |= CAN_1_RX_ACK_MSG;
}
#endif

#if(CAN_1_RX7_FUNC_ENABLE)
void CAN_1_ReceiveMsg7(void)
{
        /* `#START MESSAGE_7_RECEIVED` */

        /* `#END` */
CAN_1_RX[7].rxcmd.byte[0] |= CAN_1_RX_ACK_MSG;
}
#endif

#if(CAN_1_RX8_FUNC_ENABLE)
void CAN_1_ReceiveMsg8(void)
{
        /* `#START MESSAGE_8_RECEIVED` */

        /* `#END` */
CAN_1_RX[8].rxcmd.byte[0] |= CAN_1_RX_ACK_MSG;
}
#endif


#if(CAN_1_RX9_FUNC_ENABLE)
void CAN_1_ReceiveMsg9(void)
{
        /* `#START MESSAGE_9_RECEIVED` */

        /* `#END` */
CAN_1_RX[9].rxcmd.byte[0] |= CAN_1_RX_ACK_MSG;
}
#endif

#if(CAN_1_RX10_FUNC_ENABLE)
void CAN_1_ReceiveMsg10(void)
{
        /* `#START MESSAGE_10_RECEIVED` */

        /* `#END` */
CAN_1_RX[10].rxcmd.byte[0] |= CAN_1_RX_ACK_MSG;
}
#endif

#if(CAN_1_RX11_FUNC_ENABLE)
void CAN_1_ReceiveMsg11(void)
{
```

```
        /* `#START MESSAGE_11_RECEIVED` */

        /* `#END` */
CAN_1_RX[11].rxcmd.byte[0] |= CAN_1_RX_ACK_MSG;
}
#endif

#if(CAN_1_RX12_FUNC_ENABLE)
void CAN_1_ReceiveMsg12(void)
{
        /* `#START MESSAGE_12_RECEIVED` */

        /* `#END` */
CAN_1_RX[12].rxcmd.byte[0] |= CAN_1_RX_ACK_MSG;
}
#endif

#if(CAN_1_RX13_FUNC_ENABLE)
void CAN_1_ReceiveMsg13(void)
{
        /* `#START MESSAGE_13_RECEIVED` */

        /* `#END` */
CAN_1_RX[13].rxcmd.byte[0] |= CAN_1_RX_ACK_MSG;
}
#endif

#if(CAN_1_RX14_FUNC_ENABLE)
void CAN_1_ReceiveMsg14(void)
{
        /* `#START MESSAGE_14_RECEIVED` */

        /* `#END` */
CAN_1_RX[14].rxcmd.byte[0] |= CAN_1_RX_ACK_MSG;
}
#endif

#if(CAN_1_RX15_FUNC_ENABLE)
void CAN_1_ReceiveMsg15(void)
{
        /* `#START MESSAGE_15_RECEIVED` */

        /* `#END` */
CAN_1_RX[15].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 rxreg)
{
        if (CAN_1_RX[rxreg].rxcmd.byte[0] & CAN_1_RX_ACK_MSG)
        {
```

```
                /* `#START MESSAGE_BASIC_RECEIVED` */


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

CAN_1.c file

```
    void CAN_1_MsgRXIsr(void)
    {
    …
    /* RX Full mailboxes handler */
              switch(i)
              {
                    case 3 : CAN_1_ReceiveMsgVehicleSpeed();
                          break;
                    case 4 : CAN_1_ReceiveMsgTCM();
                          break;
                    case 5 : CAN_1_ReceiveMsgPCV();
                          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 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, in this example, it is 0x180 and IDE and RTR accordingly. For this ID, IDE and RTR values and the AMR and ACR registers are set to the following 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
   - AMR[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:

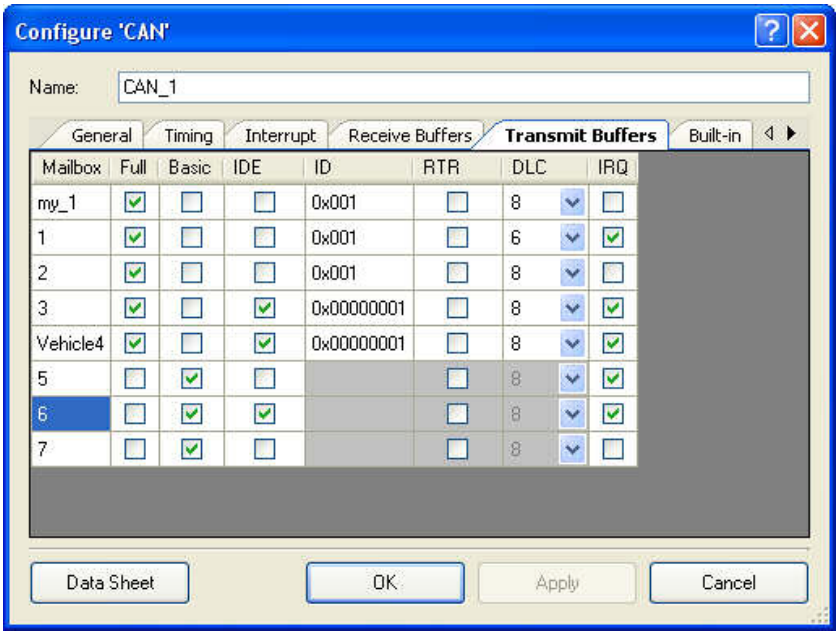
```
uint8 result = FAIL;
uint16 temp_amr;
uint16 temp_acr;

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 Chapter 23 – Controller Area Network (CAN) in the TRM.

Configure CAN Dialog – 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 in 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. 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

ID slot has a check to verify proper identifier length.

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

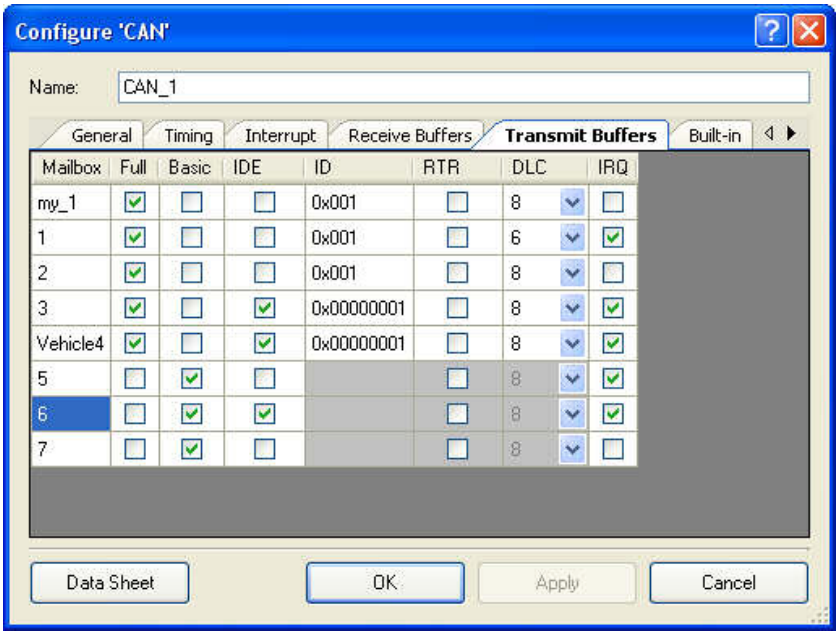
Every “Full” TX mailbox has a predefined API provided to the user. Function list can be found in CAN_1_TX_RX_func.c file included in the project. These functions are conditionally compiled depending on the transmit mailbox setting. Only mailboxes defined as “Full” shall have their respective functions compiled.

CAN_1_TX*x*_FUNC_ENABLE – Defines whether a function shall be compiled or not. Defines can be found in CAN_1.h project file.

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.

Below is an example to illustrate the use of the above-described API.

Figure 2 - Transmit Buffers Configuration



CAN_1.h file

```
#define CAN_1_TX0_FUNC_ENABLE 1
#define CAN_1_TX1_FUNC_ENABLE 1
#define CAN_1_TX2_FUNC_ENABLE 1
#define CAN_1_TX3_FUNC_ENABLE 1
#define CAN_1_TX4_FUNC_ENABLE 1
#define CAN_1_TX5_FUNC_ENABLE 0
#define CAN_1_TX6_FUNC_ENABLE 0
#define CAN_1_TX7_FUNC_ENABLE 0
```

CAN_1_TX_RX_func.c file

```
#if(CAN_1_TX0_FUNC_ENABLE)
uint8 CAN_1_SendMsgmy_1(void)
{
```

```
uint8 result = SUCCESS;
          if (CAN_1_TX[0].txcmd.byte[0] & CAN_1_TX_REQUEST_PENDING)
          {
                result = FAIL;
          }
          else
          {
                /* `#START MESSAGE_0_TRASMITTED` */

                /* `#END` */
                CY_SET_REG32((reg32 *) &CAN_1_TX[0].txcmd.byte[0] |=
CAN_1_SEND_MESSAGE);
          }

return result;
}
#endif

#if(CAN_1_TX1_FUNC_ENABLE)
uint8 CAN_1_SendMsg1(void)
{
uint8 result = SUCCESS;
          if (CAN_1_TX[1].txcmd.byte[0] & CAN_1_TX_REQUEST_PENDING)
          {
                result = FAIL;
          }
          else
          {
                /* `#START MESSAGE_1_TRASMITTED` */


                /* `#END` */
                CY_SET_REG32((reg32 *) &CAN_1_TX[1].txcmd.byte[0] |=
CAN_1_SEND_MESSAGE);
}

return result;
}
#endif

#if(CAN_1_TX2_FUNC_ENABLE)
uint8 CAN_1_SendMsg2(void)
{
uint8 result = SUCCESS;
          if (CAN_1_TX[2].txcmd.byte[0] & CAN_1_TX_REQUEST_PENDING)
          {
                result = FAIL;
          }
          else
          {
                /* `#START MESSAGE_2_TRASMITTED` */


                /* `#END` */
```

```
                CY_SET_REG32((reg32 *) &CAN_1_TX[2].txcmd.byte[0] |=
CAN_1_SEND_MESSAGE);
}

return result;
}
#endif

#if(CAN_1_TX3_FUNC_ENABLE)
uint8 CAN_1_SendMsg3(void)
{
uint8 result = SUCCESS;
          if (CAN_1_TX[3].txcmd.byte[0] & CAN_1_TX_REQUEST_PENDING)
          {
                result = FAIL;
          }
          else
          {
                /* `#START MESSAGE_3_TRASMITTED` */


                /* `#END` */
                CY_SET_REG32((reg32 *) &CAN_1_TX[3].txcmd.byte[0] |=
CAN_1_SEND_MESSAGE);
}

return result;
}
#endif

#if(CAN_1_TX4_FUNC_ENABLE)
uint8 CAN_1_SendMsgVehicle4(void)
{
uint8 result = SUCCESS;
          if (CAN_1_TX[4].txcmd.byte[0] & CAN_1_TX_REQUEST_PENDING)
          {
                result = FAIL;
          }
          else
          {
                /* `#START MESSAGE_Vehicle4_TRASMITTED` */


                /* `#END` */
                CY_SET_REG32((reg32 *) &CAN_1_TX[4].txcmd.byte[0] |=
CAN_1_SEND_MESSAGE);
}

return result;
}
#endif

#if(CAN_1_TX5_FUNC_ENABLE)
uint8 CAN_1_SendMsg5(void)
{
```

```
uint8 result = SUCCESS;
          if (CAN_1_TX[5].txcmd.byte[0] & CAN_1_TX_REQUEST_PENDING)
          {
                result = FAIL;
          }
          else
          {
                /* `#START MESSAGE_5_TRASMITTED` */


                /* `#END` */
                CY_SET_REG32((reg32 *) &CAN_1_TX[5].txcmd.byte[0] |=
CAN_1_SEND_MESSAGE);
}

return result;
}
#endif

#if(CAN_1_TX6_FUNC_ENABLE)
uint8 CAN_1_SendMsg6(void)
{
uint8 result = SUCCESS;
          if (CAN_1_TX[6].txcmd.byte[0] & CAN_1_TX_REQUEST_PENDING)
          {
                result = FAIL;
          }
          else
          {
                /* `#START MESSAGE_6_TRASMITTED` */


                /* `#END` */
                CY_SET_REG32((reg32 *) &CAN_1_TX[6].txcmd.byte[0] |=
CAN_1_SEND_MESSAGE);
}

return result;
}
#endif

#if(CAN_1_TX7_FUNC_ENABLE)
uint8 CAN_1_SendMsg7(void)
{
uint8 result = SUCCESS;
          if (CAN_1_TX[7].txcmd.byte[0] & CAN_1_TX_REQUEST_PENDING)
          {
                result = FAIL;
          }
          else
          {
                /* `#START MESSAGE_7_TRASMITTED` */


                /* `#END` */
```

```
                CY_SET_REG32((reg32 *) &CAN_1_TX[7].txcmd.byte[0] |=
CAN_1_SEND_MESSAGE);
}

return result;
}
#endif
```

The common function provided for all “Basic” Transmit mailboxes:

```
uint8 CAN_1_SendMsg(CANTXMsg *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 for ID slot is placed bellow;
- 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).

Function is looping through the transmit message mailboxes that are designated as “Basic” CAN mailboxes and look for the first available mailbox:

- When a free “Basic” CAN mailbox has been found. The data passed via the CANTXMsg structure are copied to the appropriate CAN transmit mailbox. When message put into transmit queue indication of “SUCCESS” is returned to the application.
- When no free “Basic” mailbox is found. The function try again limited number of retries. When all retries fail an indication of “FAIL” is returned to the application.

Structure `CANTXMsg` contains all required information to transmit message:

```
typedef struct _CANTXMsg
{
        uint32 id;
        uint8 rtr;
        uint8 ide;
        uint8 dlc;
        uint8 irq;
        DATA_BYTES *msg;
} CANTXMsg;
```

Structure `DATA_BYTES` contains 8 bytes of data in message.

```
typedef struct _DATA_BYTES
{
        uint8 byte[8];
```

```
} DATA_BYTES;
```

Clock Selection

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

Placement

The CAN component will be placed in any available Fixed Function CAN block when multiple blocks exist on the chip.

Resources

The CAN component is a fixed hardware block and does not require any additional resources on the chip.

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
void CAN_Init(void)	Configures CAN component.
uint8 CAN_Start(void)	Sets CAN Component into Run mode and start counter if polling mailboxes available.
uint8 CAN_Stop(void)	Sets CAN Component into Stop mode and stop counter if polling mailboxes available.
uint8 CAN_GlobalIntEnable(void)	This function Enables Global Interrupts from CAN Component.
uint8 CAN_GlobalIntDisable(void)	This function Disables Global Interrupts from CAN Component.

Function	Description
uint8 CAN_SetPreScaler (uint16 bitrate)	Sets PreScaler for generation the time quantum which defines the time quanta.
uint8 CAN_SetArbiter (uint8 arbiter)	Sets arbitration type for transmit buffers
uint8 CAN_SetTsegSample (uint8 cfg_tseg1, uint8 cfg_tseg2, uint8 sm, uint8 sjw)	Configures: Time segment 1, Time segment 2, Sampling Mode and Synchronization Jump Width.
uint8 CAN_SetRestartType (uint8 reset)	Sets Reset type.
uint8 CAN_SetEdgeMode (uint8 edge)	Sets Edge Mode.
uint8 CAN_RXRegisterInit (uint32 *reg, uint32 configuration)	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_GetRXErrorStat (void)	Returns error status of CAN Component.
uint8 CAN_SetIrqMask (uint16 request)	Sets to enable/disable particular interrupt sources. Interrupt Mask directly write to CAN Interrupt Enable register.
void CAN_ArbLostIsr(void)	Entry point to Arbitration Lost Interrupt. Clears Arbitration Lost interrupt flag. Only generated if Arbitration Lost Interrupt parameter is enabled.
void CAN_OvrLdErrorIsr(void)	Entry point to Overload Error Interrupt. Clears Overload Error interrupt flag. Only generated if Overload Error Interrupt parameter is enabled.
void CAN_BitErrorIsr(void)	Entry point to Bit Error Interrupt. Clears Bit Error interrupt flag. Only generated if Bit Error Interrupt parameter is enabled.
void CAN_BitStuffErrorIsr(void)	Entry point to Bit Stuff Error Interrupt. Clears Bit Stuff Error interrupt flag. Only generated if Bit Stuff Error Interrupt parameter is enabled.
void CAN_AckErrorIsr(void)	Entry point to Acknowledge Error Interrupt. Clears Acknowledge Error interrupt flag. Only generated if Acknowledge Error Interrupt parameter is enabled.
void CAN_MsgErrorIsr(void)	Entry point to Form Error Interrupt. Clears Form Error interrupt flag. Only generated if Form Error Interrupt parameter is enabled.

Function	Description
void CAN_CrcErrorIsr(void)	Entry point to CRC Error Interrupt. Clears CRC Error interrupt flag. Only generated if CRC Error Interrupt parameter is enabled.
void CAN_BusOffIsr(void)	Entry point to Bus Off Interrupt. Places CAN Component to Stop mode. Only generated if Bus Off Interrupt parameter is enabled. Recommended setting to enable this interrupt.
void CAN_MsgLostIsr(void)	Entry point to Message Lost Interrupt. Clears Message Lost interrupt flag. Only generated if Message Lost Interrupt parameter is enabled.
void CAN_MsgTXIsr(void)	Entry point to Transmit Message Interrupt. Clears Transmit Message interrupt flag. Only generated if Transmit Message Interrupt parameter is enabled.
void CAN_MsgRXIsr(void)	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.
uint8 CAN_RxBufConfig (CANRXcfg *rxconfig)	Configures all receive registers for particular mailbox. Mailbox number contains CANRXcfg structure.
uint8 CAN_TxBufConfig (CANTXcfg * txconfig)	Configures all transmit registers for particular mailbox. Mailbox number contains CANTXcfg structure.
uint8 CAN_SendMsg (CANTXMsg *message)	Send Message from one of Basic mailboxes. Function loop through the transmit message buffer designed as Basic CAN maiboxes for first free available and send from it. The number of retries is limited.
void CAN_TxCancel (uint8 bufferId)	Cancel transmission of a message that has been queued for transmitted. Values between 0 and 15 are valid.
void CAN_ReceiveMsg0-15(void)	Entry point to Receive Message 0-15 Interrupt. Clears Receive Message 0-15 interrupt flag. Only generated Receive mailbox designed as Full interrupt based.
void CAN_ReceiveMsg (uint8 rxreg)	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.
uint8 CAN_SendMsg0-7(void)	Entry point to Transmit Message 0-7. Function check if mailbox 0-7 doesn't already have an un-transmitted messages waiting for arbitration. If not initiate transmission of the message. Only generated Transmit mailbox designed as Full.

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

uint8 CAN_Init(void)

Description: This function configures CAN component. The parameters are passed from the Configure dialog. This function should call once in code. Reconfiguration on the fly only possible through direct register writes.

Parameters: None

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

Side Effects: None

uint8 CAN_Start(void)

Description: This function sets CAN Component into run mode and start counter if polling mailboxes available.

Parameters: None

Return Value: (uint8) Indication whether register is written and verified.

Side Effects: None

uint8 CAN_Stop(void)

Description: This function sets CAN Component into Stop mode and stop 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 PreScaler for generation the time quantum which defines the time quanta. 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 cfg_tseg1, uint8 cfg_tseg2, uint8 sm, uint8 sjw)

Description: This function configures: Time segment 1, Time segment 2, Sampling Mode and Synchronization Jump Width.

Parameters: (uint8) cfg_tseg1: Time segment 1, value between 0x2 and 0xF are valid

(uint8) cfg_tseg2: Time segment 2, value between 0x1 and 0x7 are valid

(uint8) sm: Sampling Mode, one or three sampling points are used

(uint8) sjw: Synchronization Jump Width, value between 0x0 and 0x3 are valid.

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 *reg, uint32 configuration)

Description: This function writes CAN receive registers only.

Parameters: (uint32 *) reg: 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_GetRXErrorStat(void)

Description: This function returns error status of CAN component.

Parameters: None

Return Value: (uint8) Error status.

Side Effects: None

uint8 CAN_SetIrqMask(uint16 request)

Description: This function sets to enable/disable particular interrupt sources. Interrupt Mask directly write to 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_OvrLdErrrorIsr(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(CANRXcfg *rxconfig)

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

Parameters: (CANRXcfg *) 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(CANTXcfg *txconfig)

Description: This function configures all transmit registers for particular mailbox. Mailbox number contains CANTXcfg structure.

Parameters: (CANTXcfg *) 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 Send Message from one of Basic mailboxes. Function loop through the transmit message buffer designed as Basic CAN mailboxes for first free available and send from it. The number of retries is limited.

Parameters: (CANTXMsg *) message: Pointer to structure that contain all required data to send message.

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 interrupt flag. Only generated Receive mailbox designed as Full interrupt based.

Parameters: None

Return Value: None

Side Effects: None

void CAN_ReceiveMsg(uint8 rxreg)

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) rxreg: Mailbox number that trig Receive Message Interrupt.

Return Value: None

Side Effects: None

uint8 CAN_SendMsg0-7(void)

Description: These functions are entry point to Transmit Message 0-7 . Function check if mailbox 0-7 doesn't already have an un-transmitted messages waiting for arbitration. If not initiate transmission of the message. Only generated for Transmit mailbox designed as Full.

Parameters: None

Return Value: (uint8) Indication if Message has been sent.

Side Effects: None

Sample Firmware Source Code

The following is a C language example demonstrating the basic functionality of the CAN component. This example assumes the component has been placed in a design with the default name "CAN_1."

Note If you rename your component you must also edit the example code as appropriate to match the component name you specify.

```
#include <device.h>

 main()
{
    CAN_1_Init();
    CAN_1_Start();
}
```

Interrupt Service Routines

There are 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;
- Buss Off – CAN has reach the buss off state;
- Message Lost – is set when new message arrives but the not place to put this message;
- Transmit Message – indicates that message was sent;
- Receive Message - indicates that message was received;

All this interrupts sources have entry points (functions) so you can place code there, the user sections are placed for this. This functions are conditionally compiled depends of CyDesigner Customizer.

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

Functional Description

For a completed description, refer to Chapter 23 – Controller Area Network (CAN) in the TRM.

Block Diagram and Configuration

For a complete block diagram and configuration information, refer to Chapter 23 – Controller Area Network (CAN) in the TRM.

References

Not applicable

DC and AC Electrical Characteristics

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

5.0V/3.3V DC and AC Electrical Characteristics

Parameter	Typical	Min	Max	Units	Conditions and Notes
Input					
Input Voltage Range	---		Vss to Vdd	V	
Input Capacitance	---		---	pF	
Input Impedance	---		---	Ω	
Maximum Clock Rate	---		67	MHz	

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
1.10.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.
1.10.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.

PRELIMINARY