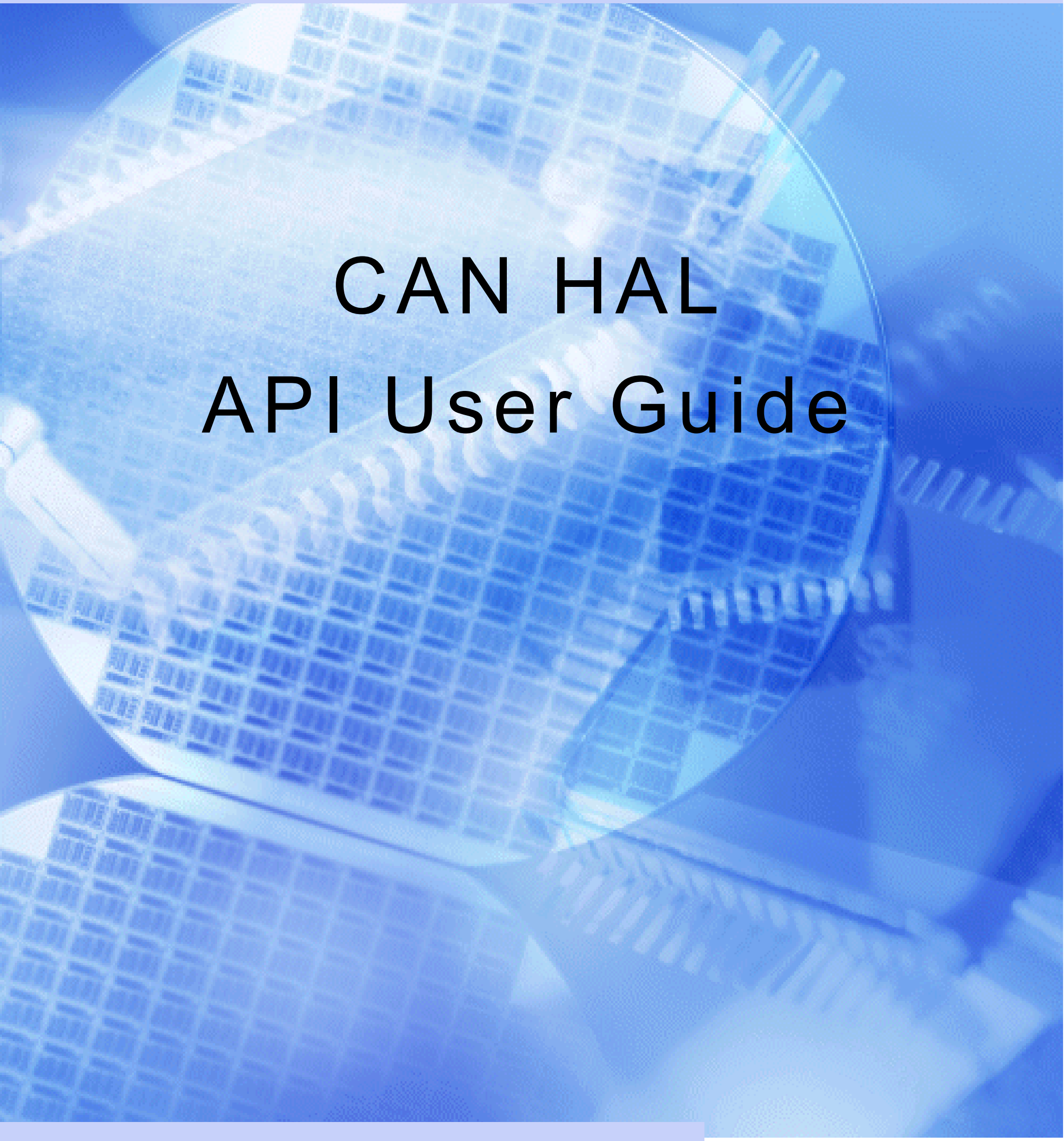
User Guide, v2.4, 17 Feb 2006

CAN HAL API User Guide



Microcontrollers

Never stop thinking.

Edition 17 Feb 2006

Published by Infineon Technologies India Pvt Ltd,
10 Floor Discoveror Building

ITPL, Whitefield Road,
Bangalore - 560 066

User Guide, v2.4, 17 Feb 2006

CAN HAL API User Guide

Microcontrollers

Never stop thinking.

CAN API
Revision History 17 Feb 2006 v2.4

Previous Version: v0.0.0

Page	Subjects (major changes since last revision)
ALL	First release
Section 5	Application Note on Interrupt Service Routine added
Section 6	Application note added to use CAN LLD along DAvE generated drivers.
Section 4	Gateway example added and more comemnts are added
Section 5.3	Added example for elinux driver

Author:
- S.A.Kazmi
- Sriram Mahesh Babu

1 CAN HAL Introduction

The CAN HAL (hardware abstraction layer) forms one part out of a larger system of software modules designed to buffer application software from hardware specific implementation details. The Infineon Technologies modular HAL approach however goes beyond simply providing a standardised API for a type of device, each HAL is designed to work as part of a larger system. System resources are reserved by the HAL's using a system hardware abstraction layer meaning that runtime conflicts are avoided and different peripherals may use the same resources at different points in the application. If the peripheral clock is changeable then the HAL's will normally support on the fly clock changing allowing the clock speed to be changed for power saving etc. Data transfer requests from the application software can be queued so that the application does not need to wait for one transfer to end before the next can be started.

In cases where these features are not desirable, possibly due to runtime efficiency or code size constraints, they can simply be removed by modifying a single configuration file and recompiling the HAL, meaning there is no unnecessary code overhead.

In summary the modular HAL system provides the following advantages:

–No extra hardware specific code is required

–Pre-tested code modules are available

–Hardware can be exchanged with little or no application software modifications

–Power saving features incorporated in the HAL

–System resource conflicts are automatically avoided

–Data transfer requests can be queued.

–HAL's are highly configurable

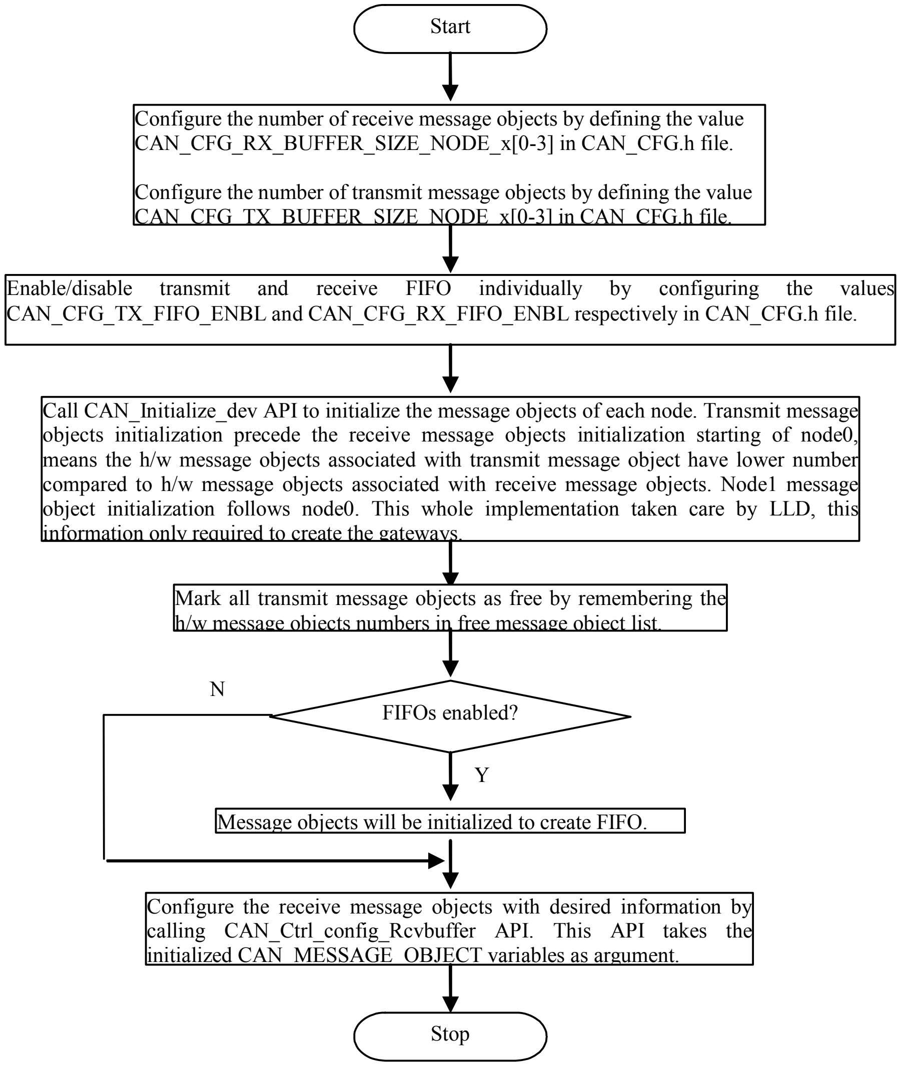
–Porting to different hardware is easy

–Standardised API can be used for development

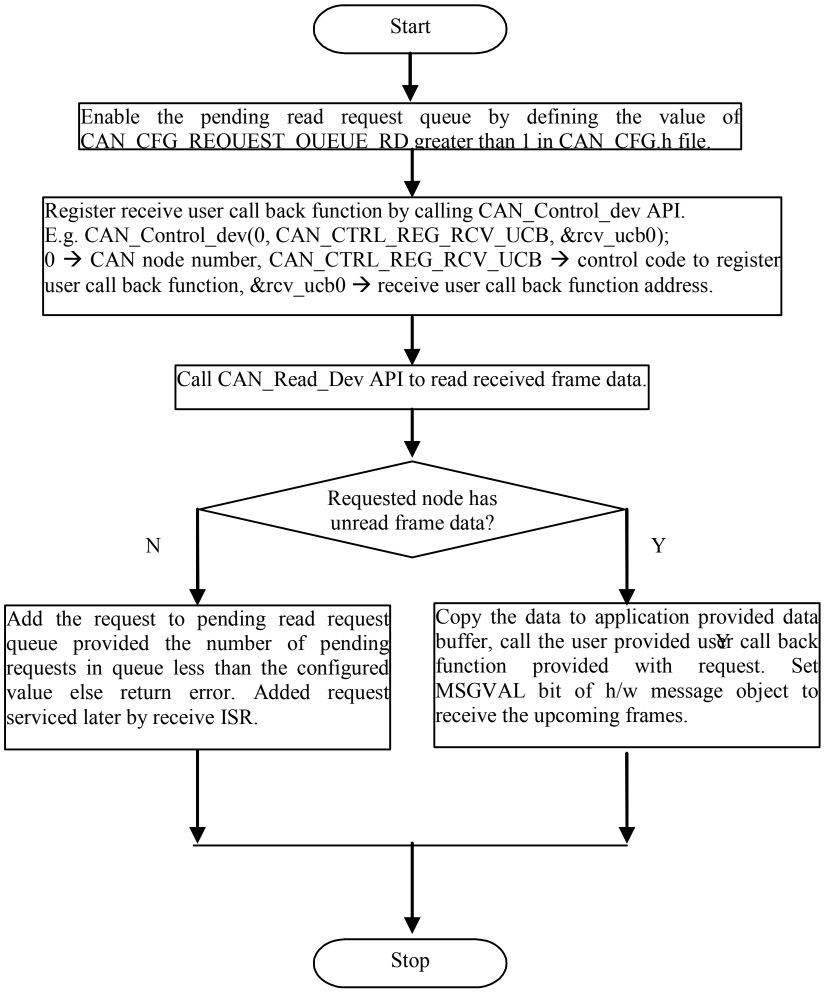
–Application and hardware dependant developments can go in parallel assuming a standard API

2 Design flow diagrams

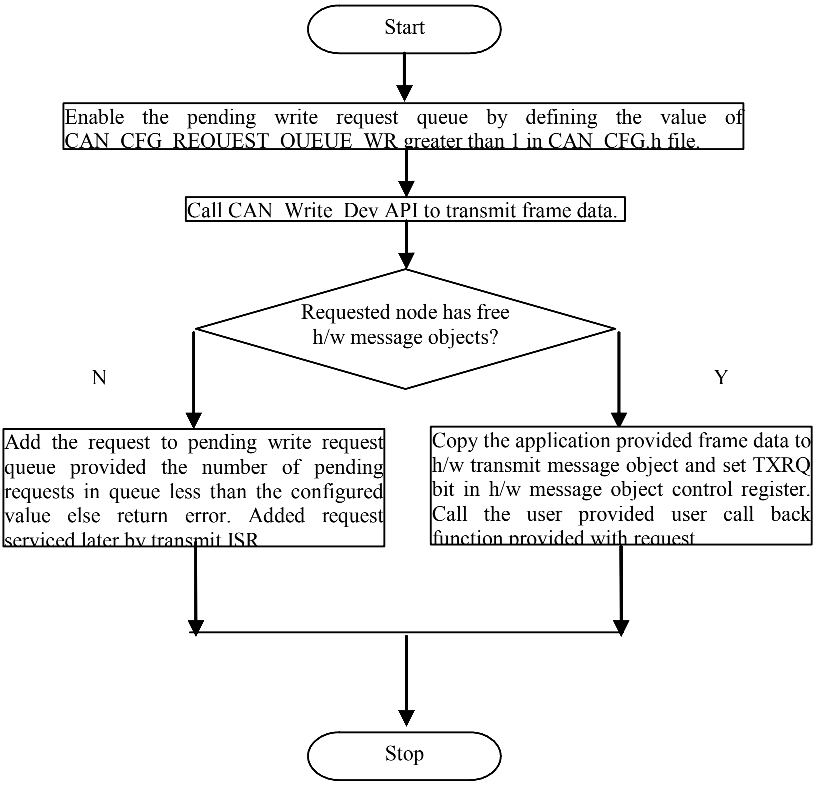
2.1 Initialize/configure message objects for each node


Start
Configure the number of receive message objects by defining the value CAN_CFG_RX_BUFFER_SIZE_NODE_x[0-3] in CAN_CFG.h file.
Configure the number of transmit message objects by defining the value CAN_CFG_TX_BUFFER_SIZE_NODE_x[0-3] in CAN_CFG.h file.
Enable/disable transmit and receive FIFO individually by configuring the values CAN_CFG_TX_FIFO_ENBL and CAN_CFG_RX_FIFO_ENBL respectively in CAN_CFG.h file.
Call CAN_Initialize_dev API to initialize the message objects of each node. Transmit message objects initialization precede the receive message objects initialization starting of node0, means the h/w message objects associated with transmit message object have lower number compared to h/w message objects associated with receive message objects. Node1 message object initialization follows node0. This whole implementation taken care by LLD, this information only required to create the gateways.
Mark all transmit message objects as free by remembering the h/w message objects numbers in free message object list.
N
FIFOs enabled?
Y
Message objects will be initialized to create FIFO.
Configure the receive message objects with desired information by calling CAN_Ctrl_config_Rcvbuffer API. This API takes the initialized CAN_MESSAGE_OBJECT variables as argument.
Stop

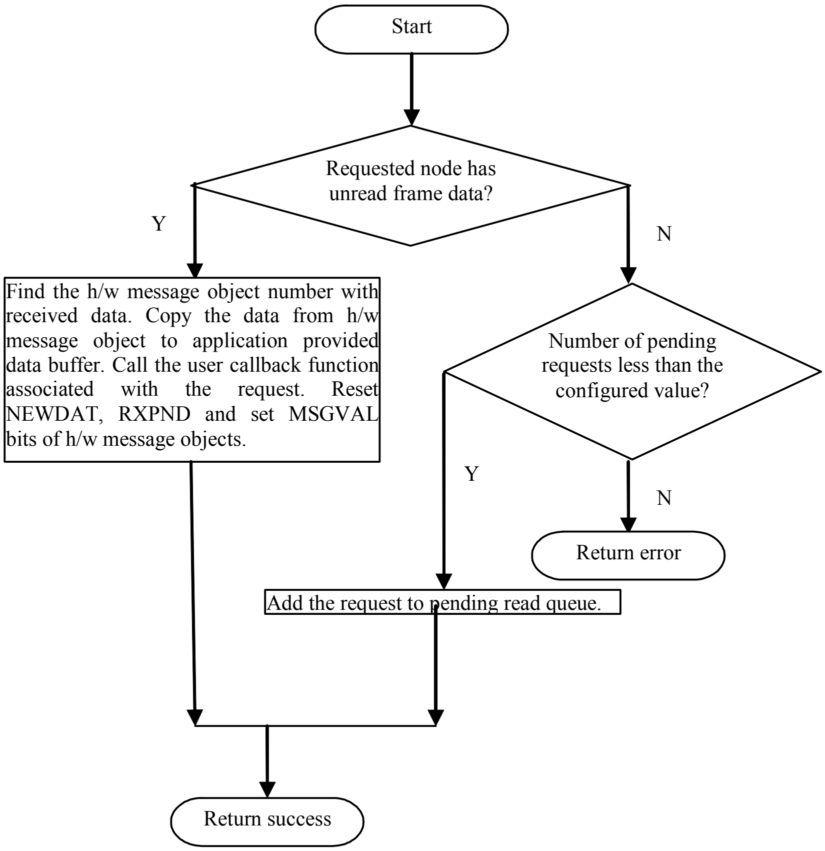
2.2 Read message frame


Start
Enable the pending read request queue by defining the value of CAN_CFG_REQUEST_QUEUE_RD greater than 1 in CAN_CFG.h file.
Register receive user call back function by calling CAN_Control_dev API.
E.g. CAN_Control_dev(0, CAN_CTRL_REG_RCV_UCB, &rcv_ucb0);
0 → CAN node number, CAN_CTRL_REG_RCV_UCB → control code to register user call back function, &rcv_ucb0 → receive user call back function address.
Call CAN_Read_Dev API to read received frame data.
Requested node has unread frame data?
N
Y
Add the request to pending read request queue provided the number of pending requests in queue less than the configured value else return error. Added request serviced later by receive ISR.
Copy the data to application provided data buffer, call the user provided user call back function provided with request. Set MSGVAL bit of h/w message object to receive the upcoming frames.
Stop

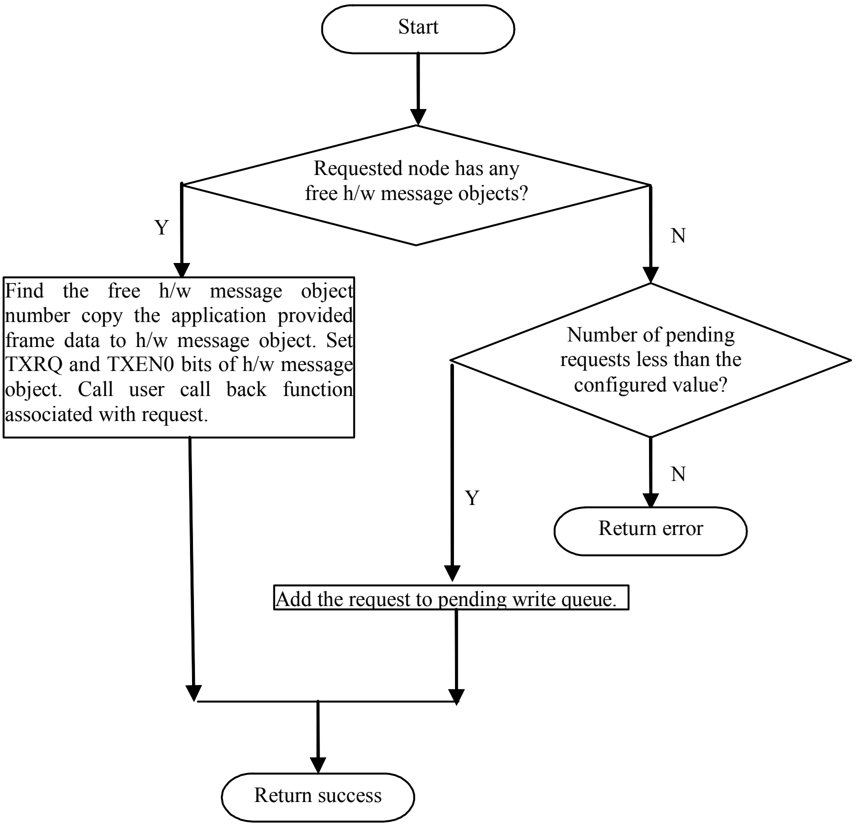
2.3 Transmit message frame


Start
Enable the pending write request queue by defining the value of CAN_CFG_REQUEST_QUEUE_WR greater than 1 in CAN_CFG.h file.
Call CAN_Write_Dev API to transmit frame data.
Requested node has free h/w message objects?
N
Y
Add the request to pending write request queue provided the number of pending requests in queue less than the configured value else return error. Added request serviced later by transmit ISR
Copy the application provided frame data to h/w transmit message object and set TXRQ bit in h/w message object control register. Call the user provided user call back function provided with request
Stop

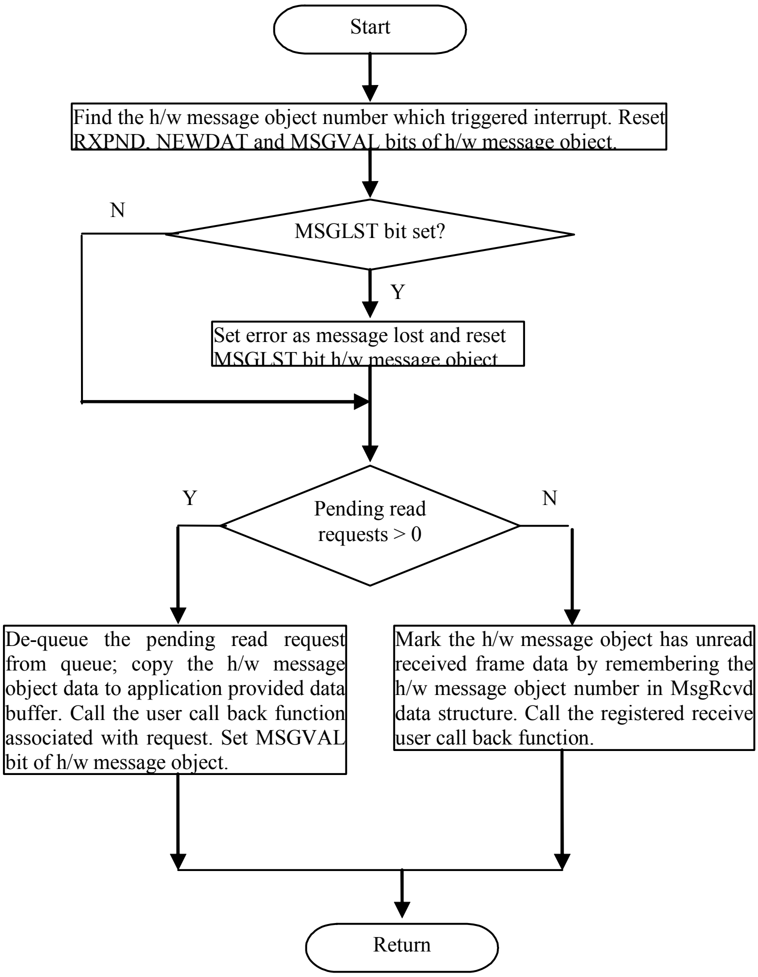
2.4 CAN_Read_Dev API


Start
Requested node has unread frame data?
Y
N
Find the h/w message object number with received data. Copy the data from h/w message object to application provided data buffer. Call the user callback function associated with the request. Reset NEWDAT, RXPND and set MSGVAL bits of h/w message objects.
Number of pending requests less than the configured value?
Y
N
Return error
Add the request to pending read queue.
Return success

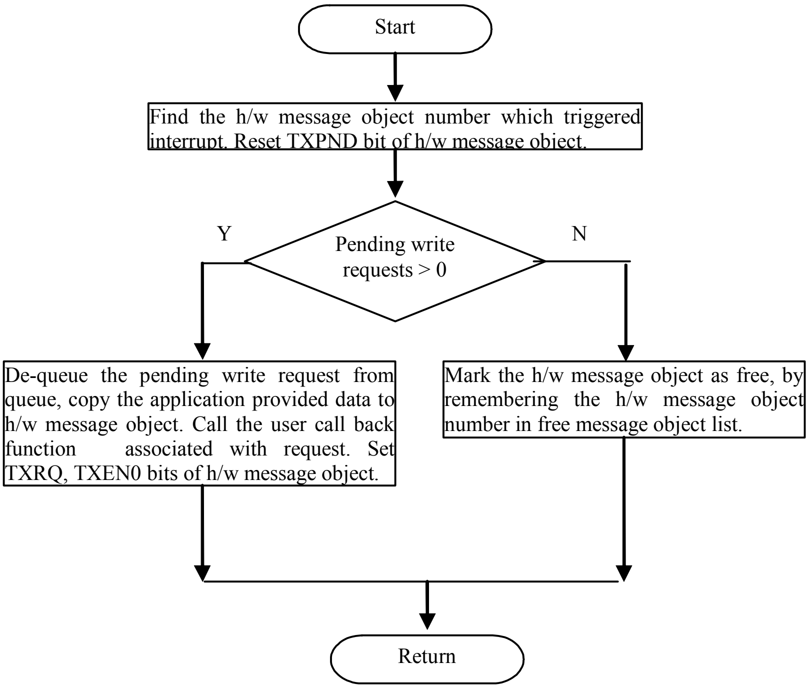
2.5 CAN_Write_Dev API


Start
Requested node has any free h/w message objects?
Y
N
Find the free h/w message object number copy the application provided frame data to h/w message object. Set TXRQ and TXEN0 bits of h/w message object. Call user call back function associated with request.
Number of pending requests less than the configured value?
N
Y
Return error
Add the request to pending write queue.
Return success

2.6 CAN receive message object ISR


Start
Find the h/w message object number which triggered interrupt. Reset RXPND, NEWDAT and MSGVAL bits of h/w message object.
MSGLST bit set?
N
Y
Set error as message lost and reset MSGLST bit h/w message object
Pending read requests > 0
Y
N
De-queue the pending read request from queue; copy the h/w message object data to application provided data buffer. Call the user call back function associated with request. Set MSGVAL bit of h/w message object.
Mark the h/w message object has unread received frame data by remembering the h/w message object number in MsgRcvd data structure. Call the registered receive user call back function.
Return

2.7 CAN transmit message object ISR


Start
Find the h/w message object number which triggered interrupt. Reset TXPND bit of h/w message object.
Pending write requests > 0
Y
N
De-queue the pending write request from queue, copy the application provided data to h/w message object. Call the user call back function associated with request. Set TXRQ, TXEN0 bits of h/w message object.
Mark the h/w message object as free, by remembering the h/w message object number in free message object list.
Return

3 CAN HAL Application Program Interface

This section defines the interface between the peripheral hardware abstraction layer and the application software. It defines the constants, typedef names, function names and limitations of the HAL. The CAN HAL API utilizes a range of constants and typedef names in order to interact in a logical way with the application program. The first section of this chapter will look at these constants and data types.

Please refer to Appendix A - Infineon IFX types for details on the IFX data types.

3.1 API Functions

3.1.1CAN_Initialise_dev

CAN_STATUS CAN_Initialise_dev(CAN_DEVICE *can_device*, **CAN_COM_PARMS*** *CAN_COM_PARMS***)**

CAN driver initialization function, this function initialises the internal data structures of the HAL related to the device selected by CAN_device, allocates any required system resources and configures the peripheral according to the CAN_COM_PARMS structure. The CAN_COM_PARMS structure must be initialised by the user before calling CAN_initialise_dev. This function must be called successfully before any of the other API functions are used and if CAN_terminate_dev is called then CAN_initialise_dev must be called again before using the other API functions. Initialisation of one HAL should run to completion (successfully or otherwise) before the next HAL is initialised. For this reason CAN_initialise should not be called from an ISR or user callback function.

Defined in: CAN_API.H

Return Value

CAN status

CAN_SUCCESS
Initialization is success.

CAN_ERR_NOT_SUPPORTED_HW
Require baud rate and FIFO levels are not with in the device supported limits.

Parameters

can_device
CAN node hardware module identification number, e.g. 0 --> CAN0, 1 --> CAN1.

CAN_COM_PARMS
Driver initialization configuration parameters.

3.1.2CAN_Terminate_dev

CAN_STATUS CAN_Terminate_dev(CAN_DEVICE *can_device***)**

CAN driver termination function; this function sets the peripheral, selected by the CAN_device parameter, into a disabled state and frees any system resources previously allocated in CAN_initialise. After this function has been called CAN_initialise_dev must be called successfully before any of the other API functions are used. CAN_terminate_dev should not be called from an ISR or user callback function.

Defined in: CAN_API.H

Return Value

CAN status

CAN_SUCCESS
  Termination of device is success.

Parameters

can_device
  CAN node hardware module identification number.

3.1.3CAN_Abort

CAN_STATUS CAN_Abort(CAN_DEVICE *can_device***)**

CAN driver abort function, cancels all currently queued data transfers and stops any transfers currently being processed on the peripheral module selected by CAN_device. CAN_initialise_dev need not be called after this function before the other API functions can be used, this function merely clears all current and pending transfers it does not terminate the HAL. New transfers may be requested using CAN_read_Dev and/or CAN_write_Dev immediately after this function returns. All aborted transfers will return a CAN_ERR error code. This function may be used to clear all requests before changing modes etc.

Defined in: CAN_API.H

Return Value

CAN status

CAN_SUCCESS
  Abort of device is success.

Parameters

can_device
  CAN node hardware module identification number.

3.1.4CAN_Status_dev

CAN_STATUS CAN_Status_dev(CAN_DEVICE *can_device*, **CAN_STATUS_INF*** *can_status_inf***)**

CAN driver status function, return the present driver configuration parameters and statistics information.

Defined in: CAN_API.H

Return Value

CAN status

CAN_SUCCESS
Status of device success fully read and returned to application.

Parameters

can_device
CAN node hardware module identification number.

can_status_inf
Users provide data structure to write the current status of the device.

3.1.5CAN_Control_dev

CAN_STATUS CAN_Control_dev(CAN_DEVICE *can_device*, **CAN_CTRL_CODE** *can_ctrl_code*, **void*** *CAN_ctrl_args***)**

CAN driver runtime configuration control function, CAN_control_dev may be used as a single entry point for all the control functions. The user would call the desired control function and provide new configuration parameters through CAN_CTRL_CODE and CAN_ctrl_arg parameters respectively.

Defined in: CAN_API.H

Return Value

CAN status

CAN_SUCCESS
Setting the new configuration parameters is success.

CAN_ERR
The provided CAN_ctrl_code does not match with any of the values defined in CAN_CTRL_CODE.

Parameters

can_device

CAN node hardware module identification number.

can_ctrl_code

CAN control Code to specify the operation to perform.

CAN_ctrl_args

New configuration parameters may not be required by all Control codes.

3.1.6CAN_Read_Dev

CAN_STATUS CAN_Read_Dev(CAN_DEVICE *can_device*, **CAN_TRANSFER*** *can_transfer***)**

CAN driver read function, Received data will be copied to application provided data buffer when s/w LLD receive buffer has any h/w message objects which contain received information. If no h/w message objects associated with s/w LLD receive buffer add it to the pending read request queue provided it is configured (CAN_CFG_REQUEST_QUEUE_RD > 0) by application in CAN_CFG.H file.

Note:

Only one frame can be read per request.

Defined in: CAN_API.H

Return Value

CAN status

CAN_SUCCESS

Successfully read data or add it to pending read request queue.

CAN_ERR_RES

The number of pending requests crosses the user configured CAN_CFG_REQUEST_QUEUE_RD level or user callback function not provided with request.

CAN_ERR_NOT_SUPPORTED_HW

Pending read queues are not configured and s/w LLD receive buffer contain no received h/w message objects.

Parameters

can_device

CAN node hardware module identification number

can_transfer

Read request to read receive message frames

can_transfer->CAN_trans_ucb <> NULL --> unblocked read request

3.1.7CAN_Write_Dev

CAN_STATUS CAN_Write_Dev(CAN_DEVICE *can_device*, **CAN_TRANSFER*** *can_transfer*)

CAN driver write function, Transmit data will be copied to free transmit h/w message object when number of free transmit h/w message objects value greater than zero. Call user callback function provided along with the request.

If no free transmit h/w message objects available add request to the pending write request queue provided it is configured (CAN_CFG_REQUEST_QUEUE_WR > 0) by application in CAN_CFG.H file.

Note:

Only one frame can be read per request.

Defined in: CAN_API.H

Return Value

CAN status

CAN_SUCCESS

Successfully data written to h/w message objects or added to queue.

CAN_ERR_RES

The number of pending requests crosses the user configured CAN_CFG_REQUEST_QUEUE_WR level or callback function not provided with application.

CAN_ERR_NOT_SUPPORTED_HW

Queues are not configured by application.

Parameters

can_device

CAN node hardware module identification number

can_transfer

Write request configuration parameters to send CAN frames.

can_transfer->can_trans_ucb <> NULL --> unblocked write request.

3.1.8CAN_Ctrl_trns_node_act

CAN_STATUS CAN_Ctrl_trns_node_act(CAN_DEVICE *can_device*, **IFX_UINT32 *** *CAN_ctrl_args***)**

CAN driver runtime node activate function, CAN_Ctrl_trns_node_act is used to activate or disable the chosen node at runtime.

If argument value > 0 (zero) --> terminates the participation of this node in the CAN traffic.

Value == 0 (Zero) --> enables the participation of the node in the CAN traffic.

Defined in: CAN_API.H

Return Value

CAN status

CAN_SUCCESS
Operation successfully completed.

CAN_ERR_NOT_SUPPORTED_HW
The requested node does not exist.

CAN_ERR_NOT_SUPPORTED
Not able to support the function.

Parameters

can_device
CAN hardware identification.

CAN_ctrl_args
Configuration parameters

3.1.9CAN_Ctrl_Node_Error_Warn_Level

CAN_STATUS CAN_Ctrl_Node_Error_Warn_Level(CAN_DEVICE *can_device*, **IFX_UINT32 *** *CAN_ctrl_args***)**

CAN driver runtime error warning level configuration control function; this function is used to change the error warning level of the chosen Node. Receive and transmit counters are not effected.

Defined in: CAN_API.H

Return Value

CAN status

CAN_SUCCESS
Level set successfully.

CAN_ERR_NOT_SUPPORTED_HW
: The requested level crosses the hard ware supported limits.

CAN_ERR_NOT_SUPPORTED
: Function not supported

Parameters

can_device
: CAN hardware identification.

CAN_ctrl_args
: Configuration parameters

3.1.10CAN_Ctrl_trns_bit

CAN_STATUS CAN_Ctrl_trns_bit(CAN_DEVICE *can_device*, **NBTR *** *CAN_ctrl_args***)**

CAN driver runtime bit timing configuration control function, CAN_Ctrl_trns_bit used to change the bit timings for the chosen device mainly to change the baud rate. For more details please refer to NBTR structure definition section.

Defined in: CAN_API.H

Return Value

CAN status

CAN_SUCCESS
: Setting the timing segment of device is success.

CAN_ERR_NOT_SUPPORTED_HW
: The requested settings exceed the Hardware limits.

CAN_ERR_NOT_SUPPORTED
: Function not supported.

Parameters

can_device
: CAN hardware identification.

CAN_ctrl_args
: Baud rate configuration parameters

3.1.11CAN_Ctrl_Node_Frame_Count_Counter_Value

CAN_STATUS CAN_Ctrl_Node_Frame_Count_Counter_Value(CAN_DEVICE *can_device*, **IFX_UINT32 *** *CAN_ctrl_args***)**

CAN driver runtime frame counter value control function, CAN_Ctrl_Node_Frame_Count_Counter_Value is used to set the frame count counter value for the chosen node and remaining fields are not affected.

Defined in: CAN_API.H

Return Value

CAN status

CAN_SUCCESS
: Operation completed successfully.

CAN_ERR_NOT_SUPPORTED_HW
: The requested value exceeds the hard ware supported limits.

CAN_ERR_NOT_SUPPORTED
: Function not supported.

Parameters

can_device
: CAN hardware identification.

CAN_ctrl_args
: Configuration parameters

3.1.12CAN_Ctrl_trns_gateway

CAN_STATUS CAN_Ctrl_trns_gateway(CAN_DEVICE *can_device*, **CAN_MESSAGE_OBJECT *** *CAN_ctrl_args***)**

CAN driver runtime gateway configuration control function, CAN_Ctrl_trns_gateway may be used during runtime to create a gateway between two CAN nodes.

CAN_ctrl_args->MoNum --> refers to gateway source object.

CAN_ctrl_args->Mofgpr.CUR --> refers to gateway destination object.

CAN_ctrl_args->Mofgpr.Data_Copy --> data copied from source to destination object.

CAN_ctrl_args->Mofgpr.Data_Frame_Send --> set TXRQ in destination object.

CAN_ctrl_args->Mofgpr.DLC_Copy --> number of data bytes copied to destination object.

CAN_ctrl_args->Mofgpr.ID_Copy --> message object ID copy to destination object.

Defined in: CAN_API.H

Return Value

CAN status

CAN_SUCCESS
: Operation completed successfully.

CAN_ERR_NOT_SUPPORTED_HW
: No receive FIFO configured for this device

CAN_ERR_NOT_SUPPORTED
: Function not supported.

Parameters

can_device
: CAN hardware identification.

CAN_ctrl_args
: Configuration parameters

3.1.13CAN_Ctrl_enable

CAN_STATUS CAN_Ctrl_enable(IFX_UINT32 * *CAN_ctrl_args*)

CAN driver runtime Peripheral enable/Disable function, CAN_Ctrl_enable may be used to enable a CAN peripheral and any IO pins required will be set to node Tx/Rx as required, or may be relinquished to system if peripheral is disabled.

If argument value 1 --> do not request to disable clock to CAN module.

Value 0 --> request to disable clock to CAN module.

Defined in: CAN_API.H

Return Value

CAN status

CAN_SUCCESS
: Operation completed successfully.

CAN_ERR_NOT_SUPPORTED
: Function not supported.

Parameters

CAN_ctrl_args
: Configuration parameters

3.1.14CAN_Ctrl_config_Rcvbuffer

CAN_STATUS CAN_Ctrl_config_Rcvbuffer(CAN_DEVICE *can_device*, **CAN_TRANSFER*** *can_transfer***)**

CAN_Ctrl_config_Rcvbuffer will be used to program the hardware message objects with user provided message frames data which include the following information for both transmit as well as receive data.

1. Accept standard/extended identifier message frames.

2. Message ID and message mask

3. If the message frame to be sent then fill the data with user provided data.

3a. Set the new data, Tx enable fields.

3b. Set MSGVAL field.

4. If the message object to be configured for receive

4a. Clear data fiels.

4b. Reset new data field.

4c. Set MSGVAL field.

Defined in: CAN_API.H

Parameters

can_device

CAN node hardware module identification number

can_transfer

Configuration parameters for hardware message object

3.2 API constants and typedefs

3.2.1CAN_API_VER_MAJ macro

```
#define CAN_API_VER_MAJ
```

Defined in: CAN_API.H

CAN_API_VER_MAJ is defined to the major version of this API which the CAN HAL supports. This version is defined as 0.1 so the following define will be in CAN_API.h:

#define CAN_API_VER_MAJ 0

Application software may check this field to determine if the HAL API version is acceptable. CAN_API_VER_MAJ will be advanced whenever a change is made to this API which will result in it being incompatible with older versions, this will only be done if the API cannot be extended in a way which maintains backwards compatibility

3.2.2CAN_API_VER_MIN macro

```
#define CAN_API_VER_MIN
```

Defined in: CAN_API.H

CAN_API_VER_MIN is defined to the minor version of this API which the CAN HAL supports. This version is defined as 0.1 so the following define will be in CAN_API.h:

#define CAN_API_VER_MIN 1

Application software may check this field to determine if the HAL API version is acceptable. CAN_API_VER_MIN will be advanced whenever an extension is made to this API which does not affect backwards compatibility.

3.2.3CAN_DEVICE typedef

Indicates the Device ID

Defined in: CAN_API.H

Comments

CAN_DEVICE is used in the API wherever a device must be selected. This is required to abstract the number of CAN peripherals implemented in the same system.

3.2.4CAN_STATUS

```
enum CAN_STATUS {
CAN_SUCCESS,
CAN_ERR,
CAN_ERR_RES,
CAN_ERR_RES_INT,
CAN_ERR_RES_MEM,
CAN_ERR_RES_IO,
CAN_ERR_NOT_SUPPORTED,
CAN_ERR_NOT_SUPPORTED_HW,
CAN_ERR_UNKNOWN_DEV,
CAN_ERR_BUSY,
CAN_ERR_NOT_INITIALISED,
CAN_ERR_NET_STUFF,
CAN_ERR_NET_FORM,
CAN_ERR_NET_ACK,
CAN_ERR_NET_CRC,
CAN_ERR_NET_BIT0,
CAN_ERR_NET_BIT1,
CAN_ERR_NET_STATE_ACT,
CAN_ERR_NET_STATE_PASS,
CAN_ERR_NET_STATE_OFF,
CAN_ERR_MSG_NEW,
CAN_ERR_MSG_LOST,
CAN_ERR_RX_OK,
CAN_ERR_TX_OK,
CAN_ERR_FRAMECOUNT_OVERFLOW,
CAN_ERR_ERRORCOUNT_WARNING,
CAN_ERR_NODE_DISABLED
};
```

Defined in: CAN_API.H

Members

CAN_SUCCESS

CAN_SUCCESS indicates that an operation completed successfully.

CAN_ERR

CAN_ERR is used to indicate that an unspecified error was encountered by the HAL.CAN_ERR will only be used as a last resort when the HAL is unable to describe the error using a more specific error code.

CAN_ERR_RES

CAN_ERR_RES is used to indicate that the CAN HAL was unable to reserve a system resource required to carry out the requested operation. This will only be used when the resource is not covered by the other CAN_ERR_RES constants.

CAN_ERR_RES_INT

CAN_ERR_RES_INT is used to indicate that a required interrupt number/ priority is currently unavailable for use by the HAL. This error will be encountered either when an attempt is made to change an interrupt number or priority during run time or when CAN_initialise_dev is called. If interrupt numbers/priorities cannot be dynamically changed due to hardware limitations then CAN_ERR_NOT_SUPPORTED_HW will be returned upon any attempt to use an incompatible number/priority.

CAN_ERR_RES_MEM

CAN_ERR_RES_MEM is used to indicate that the HAL was unable to allocate enough memory to complete the requested operation.

CAN_ERR_RES_IO

CAN_ERR_RES_IO is used to indicate that one or more physical connection lines are unavailable. This may be because a line is shared with another peripheral (and has been reserved) or if it is currently in use as a general purpose I/O line.

CAN_ERR_NOT_SUPPORTED

CAN_ERR_NOT_SUPPORTED is used to indicate that a requested operation cannot be performed because it is not supported in software. This may be because a requiredsoftware module has been compiled out

CAN_ERR_NOT_SUPPORTED_HW

CAN_ERR_NOT_SUPPORTED_HW is used to indicate that a requested operation cannot be performed because a required feature is not supported in hardware.

CAN_ERR_UNKNOWN_DEV

CAN_ERR_UNKNOWN_DEV indicates that a device ID passed to an API function was not valid.

CAN_ERR_BUSY

CAN_ERR_BUSY is returned if the CAN HAL is already busy performing an operation and the request queue is full or disabled. See Configuring the CAN HAL for information about disabling/enabling request queuing in the CAN HAL.

CAN_ERR_NOT_INITIALISED

CAN_ERR_NOT_INITIALISED is returned if an API function is called before the HAL has been successfully initialised. This checking may be configured out to improve runtime performance, see Configuring the CAN HAL for information

CAN_ERR_NET_STUFF

CAN_ERR_NET_STUFF indicates that 5 equal bits in a sequence have occurred in a part or received message, where this is not allowed.

CAN_ERR_NET_FORM

CAN_ERR_NET_FORM indicates that an error has occurred in a fixed format location of the message received.

CAN_ERR_NET_ACK

CAN_ERR_NET_ACK indicates that the transmission was not acknowledged by any node.

CAN_ERR_NET_CRC

CAN_ERR_NET_CRC indicates that the CRC value contained in the received message is different then the CRC value calculated by the device for the same message.

CAN_ERR_NET_BIT0

CAN_ERR_NET_BIT0: Two different conditions are signaled by this error code.

1) During transmission of a message (or ACK bit, active error flag, or overload flag), the CAN Node tried to send a dominant bit, while the monitored bus value was Recessive.

2) During Bus-off, this code is set each time a sequence of 11 recessive bits have been monitored. The CPU may use this code as indication that the bus is not currently disturbed.

CAN_ERR_NET_BIT1

CAN_ERR_NET_BIT1: It indicates that during transmission, the CAN Node tried to send a recessive level outside the arbitration field and the acknowledge slot, but the monitored bus value was dominant.

CAN_ERR_NET_STATE_ACT
: CAN_ERR_NET_STATE_ACT the node has recovered from Bus-Off and is active now.

CAN_ERR_NET_STATE_PASS
: CAN_ERR_NET_STATE_PASS indicates that the node has gone in passive mode.

CAN_ERR_NET_STATE_OFF
: CAN_ERR_NET_STATE_OFF indicates that the Node has gone into Bus-Off mode.

CAN_ERR_MSG_NEW
: CAN_ERR_MSG_NEW indicates that a new message has been received in the message buffer.

CAN_ERR_MSG_LOST
: CAN_ERR_MSG_LOST indicates that a message has been lost or overwritten in the message buffer.

CAN_ERR_RX_OK
: CAN_ERR_MSG_RX_OK indicates that message reception has successfully completed.

CAN_ERR_TX_OK
: CAN_ERR_MSG_TX_OK indicates that message transmission has successfully completed.

CAN_ERR_FRAMECOUNT_OVERFLOW
: CAN_ERR_FRAMECOUNT_OVERFLOW indicates the frame counter has overflows.

CAN_ERR_ERRORCOUNT_WARNING
: CAN_ERR_ERRORCOUNT_WARNING indicates that the number of errors observed has exceeded the warning level set in the error counter.

CAN_ERR_NODE_DISABLED
: CAN_ERR_NODE_DISABLED indicates the CAN node is disabled, all operations are disabled.

Comments

Many of the HAL API functions return a CAN_STATUS data type. This is a typedef name which is defined as an enumeration, and can be found in CAN_API.h. CAN_STATUS is used by the HAL to return both the initial status of an operation and to communicate any errors encountered during data transmission back to the application via a user call back function.

3.2.5CAN_CTRL_CODE

```
enum CAN_CTRL_CODE {
CAN_CTRL_TRNS_NODE_ACT,
CAN_CTRL_TRNS_NODE_EWRN_LVL,
CAN_CTRL_NODE_FRAME_COUNT_COUNTER_VALUE,
CAN_CTRL_TRNS_GATEWAY,
CAN_CTRL_TRNS_BIT,
CAN_CTRL_DISABLE,
CAN_CTRL_ENABLE,
CAN_CTRL_OPEN,
CAN_CTRL_CFG_RXBUFF,
CAN_CTRL_REG_RCV_UCB,
CAN_CTRL_UNREG_RCV_UCB
};
```

Defined in: CAN_API.H

Members

CAN_CTRL_TRNS_NODE_ACT

This enumeration constant is used with the CAN_control_dev API function. CAN_CTRL_SET_NODE_ACT may be used during runtime to change the CAN node state.

CAN_CTRL_TRNS_NODE_EWRN_LVL

This enumeration constant is used with the CAN_control_dev API function. It may be used during runtime to change the CAN node error warning level.

CAN_CTRL_NODE_FRAME_COUNT_COUNTER_VALUE

This enumeration constant is used with the CAN_control_dev API function. It may be used during runtime to change the CAN node frame count counter value.

CAN_CTRL_TRNS_GATEWAY

This enumeration constant is used with the CAN_control_dev API function. CAN_CTRL_CREATE_GATEWAY may be used during runtime to create a gateway between two CAN nodes.

CAN_CTRL_TRNS_BIT

This enumeration constant is used with the CAN_control_dev API function. CAN_CTRL_SET_BIT may be used during runtime to change the bit timings, segments etc.

CAN_CTRL_DISABLE

This enumeration constant is used with the CAN_control_dev API function. CAN_CTRL_DISABLE may be used to disable a CAN peripheral, any IO pins previously allocated will all be set to inputs.

CAN_CTRL_ENABLE

This enumeration constant is used with the CAN_control_dev API function. CAN_CTRL_ENABLE may be used to enable a CAN peripheral, any IO pins previously allocated will all be set to inputs/outputs as required.

CAN_CTRL_OPEN

This enumeration constant can be mainly used to initialise the CAN node when CAN LLD statically compiled with linux kernel. After opening device the ioctl API called with this argument.

CAN_CTRL_CFG_RXBUFF

This enumeration constant is used to configure the receive message objects dynamically.

CAN_CTRL_REG_RCV_UCB

This enumeration constant used to register the user call back function with CAN LLD. The user call back function will be called by LLD when it receive CAN message frames and no pending read requests available. In user call back function user/application shall call CAN_read_dev API to read the received frame.

User call back function will be called inside the ISR, so it is recommended to return from user call back function immediately to reduce the effect on the other function executions.

CAN_CTRL_UNREG_RCV_UCB

This enumeration constant used to un-register the user call back function with CAN LLD.

Comments

This is a typedef name which is defined as an enumeration. CAN_CTRL_CODE defines a number of enumeration constants which are used to request a specific operation from the CAN_control_dev API function.

3.2.6CAN_MODE Structure

```
typedef struct {
IFX_UINT32 Loop_Back:1;
} CAN_MODE;
```

Defined in: CAN_API.H

Members

Loop_Back:1

0--> the chosen node programmed for loop back mode,

1--> node programmed for non loop back mode

Comments

Used to program the device in loop back or in non loop back mode. It is used by the CAN_COM_PARMS structure to configure the chosen CAN Node at time of initialization.

This is also part of CAN_STATUS_INF, which is used to read the status information of a node.

3.2.7CAN_NODE_STATUS Structure

```
typedef struct {
IFX_UINT32 LEC:3;
IFX_UINT32 TXOK:1;
IFX_UINT32 RXOK:1;
IFX_UINT32 ALERT:1;
IFX_UINT32 EWRN:1;
IFX_UINT32 BUSSOFF:1;
IFX_UINT32 LLE:1;
IFX_UINT32 LOE:1;
IFX_UINT32 SUSACK:1;
IFX_UINT32 CFCVAL:16;
} CAN_NODE_STATUS;
```

Defined in: CAN_API.H

Members

LEC:3

It contains the Last Error code observed in the Node

TXOK:1

It contains the Information about the successful completion of a frame transmission

RXOK:1

It contains the Information about the successful completion of a frame reception

ALERT:1

This bit shows the status of any of these alerts

a) A change in the bit bus off of the CAN NODE STATUS Register

b) A change in the Bit EWRN of the CAN NODE STATUS Register

c) A list Length Error of the Hardware FIFO

d) Bit init has been set by the MultiCan Hardware

EWRN:1

This bit gets set if one of the error counters REC or TEC have reached the warning level

BUSSOFF:1

This bit gets set if CAN Node goes in the Bus off State

LLE:1

This bit gets set if the Hardware list associated with the Node has a Length error

LOE:1

This bit gets set if a hardware message object with wrong list index entry has been detected

SUSACK:1

This bit gets set if the CAN Node is in suspend mode due to OCDS suspend request

CFCVAL:16

These bits contains the frame count/time stamp value of the node Frame counter

Comments

Used to know the status details of a chosen node

E.g.

Frame received/transmitted successfully.

Alerts, bus off status etc...

Part of CAN_STATUS_INF structure, which is used to read the status of chosen node.

3.2.8CAN_NODE_ERROR Structure

```
typedef struct {
IFX_UINT32 REC:8;
IFX_UINT32 TEC:8;
IFX_UINT32 EWRNLVL:8;
IFX_UINT32 LETD:1;
IFX_UINT32 LECIN:1;
} CAN_NODE_ERROR;
```

Defined in: CAN_API.H

Members

REC:8

These bits contain the Receive Error count

TEC:8

These bits contain the Transmit Error count

EWRNLVL:8

These bits contain the Error warning level for the node

LETD:1

This bit indicates the direction of last Error, value 0 indicates receive error and 1 indicates transmit error.

LECIN:1

This bit indicates the value by which last Error count was incremented. Value 0 indicates increment by one and value 1 indicates increment by 8.

Comments

Contains the receive, transmit error counters, error warning level, last error (transmit/ receive) occurred and the value increased during last error.

It is part of CAN_STATUS_INF, which is used to read the status of chosen node.

3.2.9CAN_STAT_INF Structure

```
typedef struct {
CAN_MODE CAN_Mode;
CAN_NODE_STATUS CAN_node_status;
CAN_NODE_ERROR CAN_node_error;
CAN_STATUS CAN_Last_Errcode;
IFX_UINT16 CAN_Node_Pend_Reads;
IFX_UINT16 CAN_Node_Pend_Writes;
} CAN_STAT_INF;
```

Defined in: CAN_API.H

Members

CAN_Mode

The Mode of operation of the current can node, i.e. normal, loop back etc, for details refer CAN_MODE structure definition.

CAN_node_status

The Node Status for details refers CAN_NODE_STATUS structure definition.

CAN_node_error

The Node Error Status, contains the transmit and receive errors, for details refer CAN_NODE_ERROR structure definition.

CAN_Last_Errcode

The Last Error Code, for more details refer CAN_STATUS enumerated definition section.

CAN_Node_Pend_Reads

The Node Pending Reads value, in case the transfer mode is SYS_TRX_MCU

CAN_Node_Pend_Writes

The Node Pending writes value, in case the transfer mode is SYS_TRX_MCU

Comments

It is used by the CAN_status_dev API function to return configuration information about a CAN device, which includes the statistics, provided the value of CAN_CFG_STAT_LOG is 1.

3.2.10MOFGPR Structure

```
typedef struct {
IFX_UINT8 CUR;
} MOFGPR;
```

Defined in: CAN_API.H

Members

CUR

The current object pointer links to the actual target object within a Gateway structure.

Comments

The MOFGPR contains a set of message object link pointer used for gateway functionality. It is a part of CAN_MSG_OBJ structure which is used to configure the receive, transmit and gateway message objects.

All the following information applicable only to Gateway source object.

3.2.11MOFCR Structure

```
typedef struct {
IFX_UINT8 Data_Frame_Send:1;
IFX_UINT8 ID_Copy:1;
IFX_UINT8 DLC_Copy:1;
IFX_UINT8 Data_Copy:1;
} MOFCR;
```

Defined in: CAN_API.H

Members

Data_Frame_Send:1

If value set then TXRQ is set in the Gateway destination object after transfer of data frame from Gateway source to destination, else TXRQ not set in destination object

ID_Copy:1

If set then the identifier of the gateway source object (after storing the received frame in the source) is copied to the gateway destination.

DLC_Copy:1

If set then the data length code of the gateway source object (after storing the received frame in the source) is copied to the gateway destination.

Data_Copy:1

If set then the data field (registers MODATA0 and MODATA4) of the gateway source object (after storing the received frame in the source) is copied to the gateway destination.

Comments

Used to configure Gateway control parameters like ID, data etc... copying from source object to destination object.

All the following information applicable only to Gateway source object.

3.2.12CAN_MESSAGE_OBJECT Structure

```
typedef struct {
IFX_UINT16 Pri;
IFX_UINT16 MOCfg;
IFX_UINT32 ID;
IFX_UINT32 Mask;
MOFGPR Mofgpr;
IFX_UINT8 Data[8];
IFX_UINT8 MoNum;
MOFCR Mofcr;
} CAN_MESSAGE_OBJECT;
```

Defined in: CAN_API.H

Members

Pri

Message Object Priority

MOCfg

Message object configuration register

ID

Standard (11-bit)/extended (29-bit) identifier

Mask

Standard (11-bit)/extended (29-bit) mask

Mofgpr

Message Object FIFO/Gateway Pointer Reg

Data[8]

8 data bytes

MoNum

This indicates the message object number for Gateway source. This element is ignored for purpose other then Gateway creation.

Mofcr

The FIFO / Gateway pointer control configuration

Comments

The following data type serves as a software message object. Each access to a hardware message object has to be made by forward a pointer to a software message object (CAN_MESSAGE_OBJECT). The data type has the following fields:

MOCfg:

This byte contains the "Data Length Code" (DLC), the "Extended Identifier" (IDE) and the "Message Direction" (DIR).

ID:

This field is four bytes long and contains either the 11-bit identifier or the 29-bit identifier

Mask:

This field is four bytes long and contains either the 11-bit mask or the 29-bit mask

Data[8]: 8 bytes containing the data of a frame

3.2.13NECNT Structure

```
typedef struct {
IFX_UINT32 Receive_Count:8;
IFX_UINT32 Transmit_Count:8;
IFX_UINT32 Warn_Level:8;
} NECNT;
```

Defined in: CAN_API.H

Members

Receive_Count:8

This field contains the receive error counter value of the chosen node

Transmit_Count:8

This field contains the transmit error counter value of the chosen node

Warn_Level:8

This field contains the error warning level for the chosen node

Comments

Used to set receive, transmit and warning error levels. Part of CAN_COM_PARMS, which is used to initialise CAN Node.

3.2.14NBTR Structure

```
typedef struct {
IFX_UINT32 Baud_Prescaler:6;
IFX_UINT32 Sync_Jump:2;
IFX_UINT32 Timing_Seg1:4;
IFX_UINT32 Timing_Seg2:3;
IFX_UINT32 Divider_Mode:1;
} NBTR;
```

Defined in: CAN_API.H

Members

Baud_Prescaler:6

This field contains the baud rate presale value for the chosen node

Sync_Jump:2

This field contains the synchronization jump width value for the chosen node

Timing_Seg1:4

This field contains the timing segment 1 value for the chosen node

Timing_Seg2:3

This field contains the timing segment 2 value for the chosen node

Divider_Mode:1

If set, the time quantum will last 8 * (BRP + 1) cycles, else time quantum will last (BRP + 1) cycles

Comments

Used to set the baud rate parameters of CAN node like synchronization jump width, Tseg1 and tseg2. Part of CAN_COM_PARMS, which is used to initialise CAN Node.

3.2.15NFCR Structure

```
typedef struct {
IFX_UINT32 Count_value:16;
IFX_UINT32 Frame_Count_select:3;
IFX_UINT32 Frame_Count_mode:2;
} NFCR;
```

Defined in: CAN_API.H

Members

Count_value:16

This field contains the frame count value/ time stamp value for the chosen node

Frame_Count_select:3

The following modes are possible for Frame Count Mode

1) Value = 1:CFC incremented on Remote Frame

2) Value = 2:CFC incremented on correct Reception

3) Value = 4:CFC incremented on correct Transmission

The following modes are possible when the Frame Count Mode is in BIT Timing Mode:

1) Value = 0 whenever a dominant edge (transition from 1 to 0) is monitored on the receive input the time (measured in clock cycles) between this edge and the most recent dominant edge is stored in CFC.

2) Value = 1 whenever a recessive edge (transition from 0 to 1) is monitored on the receive input the time (measured in clock cycles) between this edge and the most recent dominant edge is stored in CFC.

3) Value = 2 whenever a dominant edge is received as a result of a transmitted dominant edge the time (clock cycles) between both edges is stored in CFC.

4) Value = 3 whenever a recessive edge is received as a result of a transmitted recessive edge the time (clock cycles) between both edges is stored in CFC.

5) Value = 4 whenever a dominant edge that qualifies for synchronization is monitored on the receive input the time (measured in clock cycles) between this edge and the most recent sample point is stored in CFC.

6) Value = 5 with each sample point, the time (measured in clock cycles) between the start of the new bit time and the start of the previous bit time is

stored in CFC [11:0]. Additional information is written to CFC [15:12] at each sample point:

CFC[15] : Transmit value of actual bit time

CFC[14] : Receive sample value of actual bit time

CFC[13:12] : CAN bus information

Frame_Count_mode:2

The following modes are possible

1) Value = 0:Frame Count Mode

2) Value = 1:Time Stamp Mode

3) Value = 2:Bit Timing Mode

Comments

Used to configure the frame counting mode and incrementing mode. Part of CAN_COM_PARMS, which is used to initialise CAN Node.

3.2.16CAN_COM_PARMS Structure

```
typedef struct {
CAN_MODE can_mode;
NECNT can_error_counter;
NBTR can_bit_timing;
NFCR can_frame_counter;
} CAN_COM_PARMS;
```

Defined in: CAN_API.H

Members

can_mode

The Node Operation Mode

can_error_counter

Node Error Counter settings for the MultiCan Module

can_bit_timing

Node Bit timing settings for the MultiCan Module

can_frame_counter

Node Frame Counter settings for the MultiCan Module

Comments

It is used by CAN_Initialise_dev API to initialise chosen CAN Node, for more details of each member field refer to the corresponding structure definition section.

3.2.17CAN_TRANSFER Structure

```
typedef struct {
CAN_MESSAGE_OBJECT   * can_buffer;
IFX_UINT8 can_buffer_size;
void (*CAN_trans_ucb)(struct can_trans*, CAN_STATUS);
} CAN_TRANSFER;
```

Defined in: CAN_API.H

Members

can_buffer

Address of the data buffer

can_buffer_size

Number of message objects in the buffer, used only incase CAN_Ctrl_config_Rcvbuffer API.

CAN_trans_ucb

Pointer to user call back function used only incase of CAN_Read_Dev and CAN_Write_Dev APIs,

Comments

CAN_TRANSFER is used by the CAN_read_Dev and CAN_write_Dev functions to provide information regarding the data transfer that is to be performed.

4 Configuring the CAN HAL

Although the CAN HAL can be used immediately without configuring it to suit a particular application it will often be the case that some features written into the HAL will either be unnecessary; degrade performance to an unacceptable level, take up too much memory or only be required for debugging purposes. For this reason the CAN HAL has been designed in such a way that it can be easily configured to remove unused features, a number of optional features may be enabled and/or disabled through the CAN_CFG.h file:

–CAN device number checking

–Initialisation check on API calls

Additionally further options which affect the CAN HAL may be present in the System HAL. Depending on the actual system in question these, or other, options may be available:

–On the fly peripheral clock changing

–Initial interrupts numbers/priorities settings

–CAN physical interface (GPIO) configuration

The System HAL User Guide should be available from the same source as this document, please refer to it for more details on the available settings and features.

4.1 CAN driver HAL configuration parameters

4.1.1CAN_CFG_DEV_CHK macro

```
#define CAN_CFG_DEV_CHK
```

Defined in: CAN_CFG.H

The following define selects whether device ID checking will be performed in the CAN HAL API functions. Disabling this feature will result in less code being generated.

0
: Disable the feature

1
: Enable the feature

4.1.2CAN_CFG_INIT_CHK macro

```
#define CAN_CFG_INIT_CHK
```

Defined in: CAN_CFG.H

The following define selects whether certain CAN HAL API functions will check if the HAL has been initialised before executing. Disabling this feature will result in less code being generated.

0
: Disable the feature

1
: Enable the feature

4.1.3CAN_CFG_FUNC_TERMINATE macro

```
#define CAN_CFG_FUNC_TERMINATE
```

Defined in: CAN_CFG.H

The following define selects whether the HAL API Function CAN_Terminate_dev should be included for compilation. Disabling this feature will result in less code being generated.

0
: Disable the feature

1
: Enable the feature

4.1.4CAN_CFG_FUNC_ABORT macro

```
#define CAN_CFG_FUNC_ABORT
```

Defined in: CAN_CFG.H

The following define selects whether the HAL API Function CAN_Abort should be included for compilation. Disabling this feature will result in less code being generated.

0	Disable the feature
1	Enable the feature

4.1.5CAN_CFG_FUNC_STATUS macro

```
#define CAN_CFG_FUNC_STATUS
```

Defined in: CAN_CFG.H

The following define selects whether the HAL API Function CAN_Status_dev should be included for compilation. Disabling this feature will result in less code being generated.

0	Disable the feature
1	Enable the feature

4.1.6CAN_CFG_FUNC_CONTROL macro

```
#define CAN_CFG_FUNC_CONTROL
```

Defined in: CAN_CFG.H

The following define selects whether the HAL API Function should be included for compilation. Disabling this feature will result in less code being generated.

0	Disable the feature
1	Enable the feature

4.1.7CAN_CFG_FUNC_READ macro

```
#define CAN_CFG_FUNC_READ
```

Defined in: CAN_CFG.H

The following define selects whether the HAL API Function CAN_Control_dev should be included for compilation. Disabling this feature will result in less code being generated.

0
: Disable the feature

1
: Enable the feature

4.1.8CAN_CFG_FUNC_WRITE macro

```
#define CAN_CFG_FUNC_WRITE
```

Defined in: CAN_CFG.H

The following define selects whether the HAL API Function CAN_Write_Dev should be included for compilation. Disabling this feature will result in less code being generated.

0
: Disable the feature

1
: Enable the feature

4.1.9CAN_CFG_FUNC_CTRL_TRNS_NODE_ACT macro

```
#define CAN_CFG_FUNC_CTRL_TRNS_NODE_ACT
```

Defined in: CAN_CFG.H

The following define selects whether the HAL API Function CAN_Ctrl_trns_node_act should be included for compilation. Disabling this feature will result in less code being generated.

0
: Disable the feature

1
: Enable the feature

4.1.10 CAN_CFG_FUNC_CTRL_NODE_EWRN_LVL macro

```
#define CAN_CFG_FUNC_CTRL_NODE_EWRN_LVL
```

Defined in: CAN_CFG.H

The following define selects whether the HAL API Function CAN_Ctrl_Node_Error_Warn_Level should be included for compilation. Disabling this feature will result in less code being generated.

0
: Disable the feature

1
: Enable the feature

4.1.11 CAN_CFG_FUNC_CTRL_TRNS_BIT macro

```
#define CAN_CFG_FUNC_CTRL_TRNS_BIT
```

Defined in: CAN_CFG.H

The following define selects whether the HAL API Function CAN_Ctrl_trns_bit should be included for compilation. Disabling this feature will result in less code being generated.

0
: Disable the feature

1
: Enable the feature

4.1.12 CAN_CFG_FUNC_CTRL_NODE_FRAME_COUNT_COUNTER_VALUE macro

```
#define CAN_CFG_FUNC_CTRL_NODE_FRAME_COUNT_COUNTER_VALUE
```

Defined in: CAN_CFG.H

The following define selects whether the HAL API Function CAN_Ctrl_Node_Frame_Count_Counter_Value should be included for compilation. Disabling this feature will result in less code being generated.

0
: Disable the feature

1
: Enable the feature

4.1.13CAN_CFG_FUNC_CTRL_CREATE_GATEWAY macro

```
#define CAN_CFG_FUNC_CTRL_CREATE_GATEWAY
```

Defined in: CAN_CFG.H

The following define selects whether the HAL API Function CAN_Ctrl_trns_gateway should be included for compilation. Disabling this feature will result in less code being generated.

0	Disable the feature
1	Enable the feature

4.1.14CAN_CFG_FUNC_CTRL_ENABLE macro

```
#define CAN_CFG_FUNC_CTRL_ENABLE
```

Defined in: CAN_CFG.H

The following define selects whether the HAL API Function CAN_Ctrl_enable should be included for compilation. Disabling this feature will result in less code being generated.

0	Disable the feature
1	Enable the feature

4.1.15CAN_CFG_TX_FIFO_ENBL macro

```
#define CAN_CFG_TX_FIFO_ENBL
```

Defined in: CAN_CFG.H

The define decides whether to use FIFO by CAN LLD for a frame transmission. Changing this configuration during runtime has no effect.

0	CAN LLD uses standard message object methodology for transmitting a frame.
1	CAN LLD uses FIFO methodology for transmitting a frame.

4.1.16CAN_CFG_RX_FIFO_ENBL macro

```
#define CAN_CFG_RX_FIFO_ENBL
```

Defined in: CAN_CFG.H

The define decides whether to use FIFO by CAN LLD for a frame reception. Changing this configuration during runtime has no effect.

0	CAN LLD uses standard message object methodology to receive a frame.
1	CAN LLD uses FIFO methodology to receive a frame.

4.1.17CAN_CFG_REQUEST_QUEUE_WR macro

```
#define CAN_CFG_REQUEST_QUEUE_WR
```

Defined in: CAN_CFG.H

The following define sets the maximum number of Write requests that can be supported by the CAN HAL

0	Disable the feature
>0	Enable the feature

4.1.18CAN_CFG_REQUEST_QUEUE_RD macro

```
#define CAN_CFG_REQUEST_QUEUE_RD
```

Defined in: CAN_CFG.H

The following define sets the maximum number of read requests that can be supported by the CAN HAL

0	Disable the feature
>0	Enable the feature

4.1.19CAN_CFG_CLOCK_FDR_STEP macro

```
#define CAN_CFG_CLOCK_FDR_STEP
```

Defined in: CAN_CFG.H

Value will be used to derive CAN clock from system frequency and it ranges from 1 to 1023 in decimal.

4.1.20CAN_CFG_CLOCK_FDR_DIVIDOR_MODE macro

```
#define CAN_CFG_CLOCK_FDR_DIVIDOR_MODE
```

Defined in: CAN_CFG.H

This define select the clock fractional divider mode.

1	Normal
2	Fractional divider mode

4.1.21CAN_CFG_TX_BUFFER_SIZE_NODE_x[0-3] macro

```
#define CAN_CFG_TX_BUFFER_SIZE_NODE_x[0-3]
```

Defined in: CAN_CFG.H

Configure number of transmit message objects required for CAN corresponding to Node0, Node1, Node2 and Node3.

Note: the sum of required transmit and receive message objects of all nodes should not exceed the limit of 128.

4.1.22CAN_CFG_RX_BUFFER_SIZE_NODE_x[0-3] macro

```
#define CAN_CFG_RX_BUFFER_SIZE_NODE_x[0-3]
```

Defined in: CAN_CFG.H

Configure number of receive message objects required for CAN corresponding to Node0, Node1, Node2 and Node3.

Note: the sum of required transmit and receive message objects of all nodes should not exceed the limit of 128.

4.1.23CAN_CFG_NODEx[0 - 3]_INT_TRANSFER_ENABLED macro

```
#define CAN_CFG_NODEx[0 - 3 ]_INT_TRANSFER_ENABLED
```

Defined in: CAN_CFG.H

Transfer interrupt will be triggered for corresponding node when its belonging message object received/transmit message frame. This is true only when the define CAN_CFG_NODEx_INTERRUPT_ENABLED value set to 1.

0	Don't trigger interrupt when message frame received/transmitted.
1	Trigger interrupt when message frame received/transmitted.

4.1.24CAN_CFG_NODEx[0 - 3]_INT_ALERT_ENABLED macro

```
#define CAN_CFG_NODEx[0 - 3]_INT_ALERT_ENABLED
```

Defined in: CAN_CFG.H

Alert interrupt will be triggered for corresponding node when any one of the following conditions happen.

a) Change of BOFF state.

b) Change of error warning level.

c) List length/list object error.

d) INIT bit set.

This is true only when the define CAN_CFG_NODEx_INTERRUPT_ENABLED value set to 1.

0	Don't trigger alert interrupt.
1	Trigger alert interrupt.

4.1.25CAN_CFG_NODEx[0 - 3]_INT_CFC_ENABLED macro

```
#define CAN_CFG_NODEx[0 - 3]_INT_CFC_ENABLED
```

Defined in: CAN_CFG.H

Frame counter overflow interrupt will be triggered for corresponding node when its frame counter about to overflow.

This is true only when the define CAN_CFG_NODEx_INTERRUPT_ENABLED value set to 1.

0
: Don't trigger frame counter overflow interrupt.

1
: Trigger frame counter overflow interrupt.

4.1.26CAN_CFG_NODEx[0 - 3]_INT_LEC_ENABLED macro

```
#define CAN_CFG_NODEx[0 - 3]_INT_LEC_ENABLED
```

Defined in: CAN_CFG.H

Protocol error trigger interrupt for corresponding node when it found any protocol error.

This is true only when the define CAN_CFG_NODEx_INTERRUPT_ENABLED value set to 1.

0
: Don't trigger protocol error interrupt.

1
: Trigger protocol error interrupt.

5 Application Examples

This section presents a number of simple example applications using the CAN HAL which cover the most commonly used CAN HAL API functions.

5.1 Receive and Transmit CAN frmaes when all nodes in loop back mode

```
/*
  Test procedure:
  For all the test cases make sure the following configuration is correct
  in CAN_CFG.h file

  CAN_CFG_TX_BUFFER_SIZE_NODE_0     5
  CAN_CFG_TX_BUFFER_SIZE_NODE_1     5
  CAN_CFG_TX_BUFFER_SIZE_NODE_2    5
  CAN_CFG_TX_BUFFER_SIZE_NODE_3    5

  CAN_CFG_RX_BUFFER_SIZE_NODE_0     5
  CAN_CFG_RX_BUFFER_SIZE_NODE_1     5
  CAN_CFG_RX_BUFFER_SIZE_NODE_2    5
  CAN_CFG_RX_BUFFER_SIZE_NODE_3    5

Test case 0:
a. Test all 4 CAN nodes in loop back mode.
b.  CAN  node  0  receive  5  CAN  message  frames  into  swobjudum[0]  to
swobjudum[4]  transmitted by CAN node 1 with message ID number 0x15555554.
c.  CAN  node  2  receive  5  CAN  message  frames  into  swobjudum[5]  to
swobjudum[9] transmitted by CAN node 3 with message ID number 0x15555555.

Note:
To enable Transmit FIFO configure define CAN_CFG_TX_FIFO_ENBL to value 1
in CAN_CFG.H file.
To enable receive FIFO configure define CAN_CFG_RX_FIFO_ENBL to value 1
in CAN_CFG.H file.

Configure  the  transmit  and  read  request  queue  as  follows  in  CAN_CFG.h
file.
  #define CAN_CFG_REQUEST_QUEUE_WR 20
  #define CAN_CFG_REQUEST_QUEUE_WR 20
*/
```

```
#include "COMPILER.h"
#include "CAN_CFG.h"
#include "CAN_IDL.h"
#include "CAN_IIL.h"
#include "CAN_API.h"

CAN_MESSAGE_OBJECT swobj[10], swobjdum[10] ;
IFX_VUINT8          rcv_msg_frms0 = 0, rcv_msg_frms2 = 0;

CAN_TRANSFER canT,   canT1,   canT2,   canT3,   canT4 ;
CAN_TRANSFER canT5,  canT6,   canT7,   canT8,   canT9 ;

CAN_TRANSFER canR,   canR1,   canR2,   canR3,   canR4 ;
CAN_TRANSFER canR5,  canR6,   canR7,   canR8,   canR9 ;

/*Function prototypes used by application*/
void delay(void); /*delay function*/
void rx_ucb(struct can_trans *t, CAN_STATUS s); /*transfer receive user
callback function*/
void tx_ucb(struct can_trans *t, CAN_STATUS s); /*transfer transmit user
call back function*/
void rcv_ucb0(void); /*registered receive user callback function for
node0*/
void rcv_ucb2(void); /*registered receive user call back function for
node1*/

void main(void)
{
IFX_UINT8 i = 0 ;
CAN_COM_PARMS can_init_parms ;

/*Initialise global system clock frequency*/
SYS_clk_initialise();

/*Init the CAN Mode structure*/
can_init_parms.can_mode.Loop_Back = 1 ; /*Set node in loopback mode*/

/*Baud rate configuration parameters*/
can_init_parms.can_bit_timing.Baud_Prescaler = 0 ;
can_init_parms.can_bit_timing.Divider_Mode = 0 ;
can_init_parms.can_bit_timing.Sync_Jump = 2;
can_init_parms.can_bit_timing.Timing_Seg1 = 5 ;
can_init_parms.can_bit_timing.Timing_Seg2 = 4 ;

/*Error counter configuration parameters*/
can_init_parms.can_error_counter.Receive_Count = 0 ;
can_init_parms.can_error_counter.Transmit_Count = 0 ;
can_init_parms.can_error_counter.Warn_Level = 96 ;

/*Frame counter configuration parameters*/
can_init_parms.can_frame_counter.Count_value = 0 ;
can_init_parms.can_frame_counter.Frame_Count_mode = 0 ;
can_init_parms.can_frame_counter.Frame_Count_select = 7 ;

CAN_Initialise_dev(0, &can_init_parms) ;  /*Initialise CAN node 0*/
CAN_Initialise_dev(1, &can_init_parms) ;  /*Initialise CAN node 1*/
CAN_Initialise_dev(2, &can_init_parms) ;  /*Initialise CAN node 2*/
CAN_Initialise_dev(3, &can_init_parms) ;  /*Initialise CAN node 3*/

/*Enable interrupts globally*/
ENABLE_GLOBAL_INTERRUPT() ;

/*Register user call back with LLD to receive a notification when LLD
  receives a message frame*/
CAN_Control_dev(0, CAN_CTRL_REG_RCV_UCB, &rcv_ucb0);
CAN_Control_dev(2, CAN_CTRL_REG_RCV_UCB, &rcv_ucb2);

/*Clear receive data buffers*/
for(i=0; i<10; i++)
{
  swobjdum[i].Data[0] =  0 ;
  swobjdum[i].Data[1] =  0 ;
  swobjdum[i].Data[2] =  0 ;
  swobjdum[i].Data[3] =  0 ;
  swobjdum[i].Data[4] =  0 ;
  swobjdum[i].Data[5] =  0 ;
  swobjdum[i].Data[6] =  0 ;
  swobjdum[i].Data[7] =  0 ;
}


/*Configure  message  objects  to  receive  message  frames  with  ID  -->
0x15555554 for CAN node 0*/
for(i = 0 ; i < 5 ; i++)
{
  swobj[i].Data[0] = 0 ;
  swobj[i].Data[1] = 0;
  swobj[i].Data[2] = 0;
  swobj[i].Data[3] = 0;
  swobj[i].Data[4] = 0;
  swobj[i].Data[5] = 0;
  swobj[i].Data[6] = 0;
  swobj[i].Data[7] = 0;
  swobj[i].ID  = 0x15555554 ; /*Message frame ID*/
  swobj[i].Mask = 0x1FFFFFFF ;
  swobj[i].MOCfg = 0x0084 ;   /*Extended identifier*/
}

canT.can_buffer_size = 5 ; /*Number of message objects to be configured*/
canT.can_buffer = &swobj[0] ;  /*Adress of application specified first
message object*/

/* The application provided message object configuration parameters
specified in swobj[0..4] or copied to node0 allocated receive hardware
message objects */
if (CAN_Ctrl_config_Rcvbuffer( 0, &canT ) != CAN_SUCCESS)
{
  return;
}

/* The application provided message object configuration parameters
specified in swobj[5..9] or copied to node2 allocated receive hardware
message objects */
for(i = 5 ; i < 10 ; i++)
{
  swobj[i].Data[0] = 0;
  swobj[i].Data[1] = 0;
  swobj[i].Data[2] = 0;
  swobj[i].Data[3] = 0;
  swobj[i].Data[4] = 0;
  swobj[i].Data[5] = 0;
  swobj[i].Data[6] = 0;
  swobj[i].Data[7] = 0;
  swobj[i].ID  = 0x15555555 ; /*Message frame ID*/
  swobj[i].Mask = 0x1FFFFFFF ;
  swobj[i].MOCfg = 0x0084 ;   /*Extended identifier*/
}

canT.can_buffer_size = 5 ; /*Number of message objects to be configured*/
canT.can_buffer = &swobj[5] ;  /*Adress of application specified first
message object*/

/* configure message object for recieve of CAN node2 */
if (CAN_Ctrl_config_Rcvbuffer( 2, &canT ) != CAN_SUCCESS)
{
  return;
}
```

```
/*Configure transmit message objects to send frames over CAN node 1 */
for(i = 0 ; i < 5 ; i++)
{
  swobj[i].Data[0] = 1 + i;
  swobj[i].Data[1] = 1 + i;
  swobj[i].Data[2] = 1 + i;
  swobj[i].Data[3] = 1 + i;
  swobj[i].Data[4] = 1 + i;
  swobj[i].Data[5] = 1 + i;
  swobj[i].Data[6] = 1 + i;
  swobj[i].Data[7] = 1 + i;
  swobj[i].ID   = 0x15555554 ;
  swobj[i].Mask = 0x1FFFFFFF ;
  swobj[i].MOCfg = 0x0084 ;   //DLC
}


/*Configure transmit message objects to send frames over CAN node 3 */
for(i = 5 ; i < 10 ; i++)
{
  swobj[i].Data[0] = 2 + i;
  swobj[i].Data[1] = 2 + i;
  swobj[i].Data[2] = 2 + i;
  swobj[i].Data[3] = 2 + i;
  swobj[i].Data[4] = 2 + i;
  swobj[i].Data[5] = 2 + i;
  swobj[i].Data[6] = 2 + i;
  swobj[i].Data[7] = 2 + i;
  swobj[i].ID   = 0x15555555 ; /*Message object ID*/
  swobj[i].Mask = 0x1FFFFFFF ;/*Message object ID mask*/
  swobj[i].MOCfg = 0x0084 ;   /*Extended identifier*/
}

canT.can_buffer = &swobj[0];/*Application provided data buffers to send
data*/
canT.CAN_trans_ucb   =   tx_ucb;/*Application   provided   user   callback
function*/
if(CAN_Write_Dev(1,&canT) == CAN_SUCCESS);
canT1.can_buffer = &swobj[1] ;
canT1.CAN_trans_ucb = tx_ucb ;
if(CAN_Write_Dev(1,&canT1) == CAN_SUCCESS);
canT2.can_buffer = &swobj[2] ;
canT2.CAN_trans_ucb = tx_ucb ;
if(CAN_Write_Dev(1,&canT2) == CAN_SUCCESS);
canT3.can_buffer = &swobj[3] ;
canT3.CAN_trans_ucb = tx_ucb ;
if(CAN_Write_Dev(1,&canT3) == CAN_SUCCESS);
```

```
canT4.can_buffer = &swobj[4] ;
canT4.CAN_trans_ucb = tx_ucb ;
if(CAN_Write_Dev(1,&canT4) == CAN_SUCCESS);

delay();

canT5.can_buffer = &swobj[5] ;
canT5.CAN_trans_ucb = tx_ucb ;
if(CAN_Write_Dev(3,&canT5) == CAN_SUCCESS);
canT6.can_buffer = &swobj[6] ;
canT6.CAN_trans_ucb = tx_ucb ;
if(CAN_Write_Dev(3,&canT6) == CAN_SUCCESS);
canT7.can_buffer = &swobj[7] ;
canT7.CAN_trans_ucb = tx_ucb ;
if(CAN_Write_Dev(3,&canT7) == CAN_SUCCESS);
canT8.can_buffer = &swobj[8] ;
canT8.CAN_trans_ucb = tx_ucb ;
if(CAN_Write_Dev(3,&canT8) == CAN_SUCCESS);
canT9.can_buffer = &swobj[9] ;
canT9.CAN_trans_ucb = tx_ucb ;
if(CAN_Write_Dev(3,&canT9) == CAN_SUCCESS);

delay();

/*Wait till all transmitted frames received*/
while((rcv_msg_frms2 < 5) && (rcv_msg_frms0 < 5))
{
}

canR5.can_buffer  =  &swobjdum[5];/*Application provided data buffers to
read data*/
canR5.CAN_trans_ucb  =  rx_ucb;/*Application  provided  user  callback
function*/
if(CAN_Read_Dev(2,&canR5) == CAN_SUCCESS );
canR6.can_buffer = &swobjdum[6] ;
canR6.CAN_trans_ucb = rx_ucb ;
if(CAN_Read_Dev(2,&canR6) == CAN_SUCCESS );
canR7.can_buffer = &swobjdum[7] ;
canR7.CAN_trans_ucb = rx_ucb ;
if(CAN_Read_Dev(2,&canR7) == CAN_SUCCESS );
canR8.can_buffer = &swobjdum[8] ;
canR8.CAN_trans_ucb = rx_ucb ;
if(CAN_Read_Dev(2,&canR8) == CAN_SUCCESS );
canR9.can_buffer = &swobjdum[9] ;
canR9.CAN_trans_ucb = rx_ucb ;
if(CAN_Read_Dev(2,&canR9) == CAN_SUCCESS );
```

```
canR.can_buffer = &swobjdum[0] ;
canR.CAN_trans_ucb = rx_ucb ;
if(CAN_Read_Dev(0,&canR) == CAN_SUCCESS );
canR1.can_buffer = &swobjdum[1] ;
canR1.CAN_trans_ucb = rx_ucb ;
if(CAN_Read_Dev(0,&canR1) == CAN_SUCCESS );
canR2.can_buffer = &swobjdum[2] ;
canR2.CAN_trans_ucb = rx_ucb ;
if(CAN_Read_Dev(0,&canR2) == CAN_SUCCESS );
canR3.can_buffer = &swobjdum[3] ;
canR3.CAN_trans_ucb = rx_ucb ;
if(CAN_Read_Dev(0,&canR3) == CAN_SUCCESS );
canR4.can_buffer = &swobjdum[4] ;
canR4.CAN_trans_ucb = rx_ucb ;
if(CAN_Read_Dev(0,&canR4) == CAN_SUCCESS );

   while(1)
   {
   }

}


volatile unsigned int cx = 0;
void delay(void)
{
    volatile int cc = 0;
    cx = 0;
    for (cc = 0 ; cc < 5 ; cc++)
    {
      for (cx = 0; cx < 0xFFFFF ; cx++);
    }
}

IFX_UINT32 tx_s = 0, tx_f = 0, rx_s = 0, rx_f = 0;

void rx_ucb(struct can_trans *t, CAN_STATUS s)
{
  if(s == CAN_SUCCESS)
  {
    rx_s++;
  }
  else
  {
    rx_f++;
  }
}
```

```
void tx_ucb(struct can_trans *t, CAN_STATUS s)
{
  if(s == CAN_SUCCESS)
  {
    tx_s++;
  }
  else
  {
    tx_f++;
  }
}


void rcv_ucb0(void)
{
  rcv_msg_frms0++;
}

void rcv_ucb2(void)
{
  rcv_msg_frms2++;
}
```

5.2 CAN gateway functionality example

```
/*
  Test procedure:
 Please make sure the follwoing configuration exist for this test case in
  CAN_CFG.h file

  CAN_CFG_TX_BUFFER_SIZE_NODE_0    5
  CAN_CFG_TX_BUFFER_SIZE_NODE_1    5
  CAN_CFG_TX_BUFFER_SIZE_NODE_2   5
  CAN_CFG_TX_BUFFER_SIZE_NODE_3   5

  CAN_CFG_RX_BUFFER_SIZE_NODE_0    5
  CAN_CFG_RX_BUFFER_SIZE_NODE_1    5
  CAN_CFG_RX_BUFFER_SIZE_NODE_2   5
  CAN_CFG_RX_BUFFER_SIZE_NODE_3   5

Test case:
a. This is to test the Gateway functionality of CAN module.
b. Initialise CAN node3, node2 in loopback node1 and node0 in non loopback
mode.
c. Configure node0, node2 receive buffers.
d. Configure node2 receive message objects as gateway source objects and
node1 message objects as gateway destination objects.
e. Connect nod0, node1 externally.
f. node1 transmits receive message (through node2 by using gateway) on
   external bus, which are also received by node0.
g. Compiler, download and  debug application.
h. swobjdum[5-9] message objects contain the data received by node0,
   swobjdum[0-4] contain message objects received by node2.
  Please note that can node1 and node0 are in non loopback mode. The data
   received by can node0 is from can node1, node1 from node2 by using
   gateway.


Note:
CAN hardware message objects are allocated for each node depend on user
configuration            CAN_CFG_TX_BUFFER_SIZE_NODE_x            and
CAN_CFG_RX_BUFFER_SIZE_NODE_x

Configure the transmit and read request queue as follows in CAN_CFG.h
file.
   #define CAN_CFG_REQUEST_QUEUE_WR 20
   #define CAN_CFG_REQUEST_QUEUE_WR 20
*/
```

```
#include "COMPILER.h"
#include "CAN_CFG.h"
#include "CAN_IDL.h"
#include "CAN_IIL.h"
#include "CAN_API.h"

CAN_MESSAGE_OBJECT swobj[10], swobjdum[15] ;
CAN_TRANSFER canT, canT1, canT2, canT3, canT4 ;
CAN_TRANSFER canR, canR1, canR2, canR3, canR4 ;
CAN_TRANSFER canR5, canR6, canR7, canR8, canR9 ;


void delay(void);
void rx_ucb(struct can_trans *t, CAN_STATUS s);
void tx_ucb(struct can_trans *t, CAN_STATUS s);

void main(void)
{
IFX_UINT8 i = 0 ;
CAN_COM_PARMS can_init_parms ;

/*Initialise global system clock frequency*/
SYS_clk_initialise();

/*Init the CAN Mode structure*/
/*Device is initialised in loop back mode*/
can_init_parms.can_mode.Loop_Back = 1 ;

/*Baud rate control configuration parameters*/
can_init_parms.can_bit_timing.Baud_Prescaler = 0 ;
can_init_parms.can_bit_timing.Divider_Mode = 0 ;
can_init_parms.can_bit_timing.Sync_Jump = 1;
can_init_parms.can_bit_timing.Timing_Seg1 = 3 ;
can_init_parms.can_bit_timing.Timing_Seg2 = 2 ;

/*Error counters control configuration parameters*/
can_init_parms.can_error_counter.Receive_Count = 0 ;
can_init_parms.can_error_counter.Transmit_Count = 0 ;
can_init_parms.can_error_counter.Warn_Level = 91 ;

/*Frame counter control configuration parameters*/
can_init_parms.can_frame_counter.Count_value = 0 ;
can_init_parms.can_frame_counter.Frame_Count_mode  =  0  ;  /*Frmae  count
mode*/
can_init_parms.can_frame_counter.Frame_Count_select = 7 ;
```

```
/*Initialise Node2 and Node3 in loopback mode*/
CAN_Initialise_dev(3, &can_init_parms) ;
CAN_Initialise_dev(2, &can_init_parms) ;

can_init_parms.can_mode.Loop_Back = 0 ; /*Non loop back*/

/*Initialise Node1 and Node0 in non loopback mode*/
CAN_Initialise_dev(1, &can_init_parms) ;
CAN_Initialise_dev(0, &can_init_parms) ;

/*Enable interrupts globally*/
ENABLE_GLOBAL_INTERRUPT() ;

/*Reset all application receive data buffers*/
for(i=0;i<15;i++)
{
  swobjdum[i].Data[0] =  0 ;
  swobjdum[i].Data[1] =  0 ;
  swobjdum[i].Data[2] =  0 ;
  swobjdum[i].Data[3] =  0 ;
  swobjdum[i].Data[4] =  0 ;
  swobjdum[i].Data[5] =  0 ;
  swobjdum[i].Data[6] =  0 ;
  swobjdum[i].Data[7] =  0 ;
}

for(i = 0 ; i < 5 ; i++)
{
  swobj[i].Data[0] = 0 ;
  swobj[i].Data[1] = 0;
  swobj[i].Data[2] = 0;
  swobj[i].Data[3] = 0;
  swobj[i].Data[4] = 0;
  swobj[i].Data[5] = 0;
  swobj[i].Data[6] = 0;
  swobj[i].Data[7] = 0;
  swobj[i].Mask = 0x1FFFFFFF ;
  swobj[i].ID  = 0x15555554 ; //extended message frame identifier
  swobj[i].MOCfg = 0x0084 ;  //extended message frame
}

canT.can_buffer_size = 5 ; /*Number of message objects to be configured*/
canT.can_buffer = &swobj[0] ;  /*Adress of application specified first
message object*/
```

```
/*configure message object for recieve for node2*/
if (CAN_Ctrl_config_Rcvbuffer( 2, &canT ) != CAN_SUCCESS)
{
     return;
}

/*configure message object for recieve for node0*/
if (CAN_Ctrl_config_Rcvbuffer( 0, &canT ) != CAN_SUCCESS)
{
      return;
}


/*Create a gateway between node2 and node1. Node2 message objects
  as source and node1 message objects as destination objects*/
for(i = 0 ; i < 5; i++)
{
swobj[i].Mofcr.Data_Copy = 1;    /*data copied from source object to
destination object*/
swobj[i].Mofcr.Data_Frame_Send = 1; /*TXRQ is set in destination object*/
swobj[i].Mofcr.DLC_Copy = 1; /*number of data bytes copied from source
object to destination object*/
swobj[i].Mofcr.ID_Copy = 1;  /*Message object ID copied from source object
to destination object*/
swobj[i].Mofgpr.CUR = 10 + i; /*Destination object number*/
swobj[i].MoNum = 25 + i;/*Source object number*/
  if(CAN_Control_dev(2, CAN_CTRL_TRNS_GATEWAY, &swobj[i]) != CAN_SUCCESS)
  {
    return;
  }
}
/*Configure application transmit data buffers*/
for(i = 0 ; i < 5 ; i++)
{
  swobj[i].Data[0] = 0x11 + i;
  swobj[i].Data[1] = 0x22 + i;
  swobj[i].Data[2] = 0x33 + i;
  swobj[i].Data[3] = 0x44 + i;
  swobj[i].Data[4] = 0x55 + i;
  swobj[i].Data[5] = 0x66 + i;
  swobj[i].Data[6] = 0x77 + i;
  swobj[i].Data[7] = 0x88 + i;
  swobj[i].Mask = 0x1FFFFFFF ;

  swobj[i].ID  = 0x15555554 ; //extended message object identifier
  swobj[i].MOCfg = 0x0084 ;  //extended identifier
}
```

```
/*Application provided data buffers to read data*/
canR.can_buffer     = &swobjdum[0] ;
/*Application provided user callback function*/
canR.CAN_trans_ucb = rx_ucb ;
if(CAN_Read_Dev(2,&canR) == CAN_SUCCESS );
canR1.can_buffer = &swobjdum[1] ;
canR1.CAN_trans_ucb = rx_ucb ;
if(CAN_Read_Dev(2,&canR1) == CAN_SUCCESS );
canR2.can_buffer = &swobjdum[2] ;
canR2.CAN_trans_ucb = rx_ucb ;
if(CAN_Read_Dev(2,&canR2) == CAN_SUCCESS );
canR3.can_buffer = &swobjdum[3] ;
canR3.CAN_trans_ucb = rx_ucb ;
if(CAN_Read_Dev(2,&canR3) == CAN_SUCCESS );
canR4.can_buffer = &swobjdum[4] ;
canR4.CAN_trans_ucb = rx_ucb ;
if(CAN_Read_Dev(2,&canR4) == CAN_SUCCESS );


/*Application provided data buffers to send data*/
canT.can_buffer     = &swobj[0] ;
/*Application provided user callback function*/
canT.CAN_trans_ucb = tx_ucb ;
if(CAN_Write_Dev(3,&canT) == CAN_SUCCESS);
canT1.can_buffer = &swobj[1] ;
canT1.CAN_trans_ucb = tx_ucb ;
if(CAN_Write_Dev(3,&canT1) == CAN_SUCCESS);
canT2.can_buffer = &swobj[2] ;
canT2.CAN_trans_ucb = tx_ucb ;
if(CAN_Write_Dev(3,&canT2) == CAN_SUCCESS);
canT3.can_buffer = &swobj[3] ;
canT3.CAN_trans_ucb = tx_ucb ;
if(CAN_Write_Dev(3,&canT3) == CAN_SUCCESS);
canT4.can_buffer = &swobj[4] ;
canT4.CAN_trans_ucb = tx_ucb ;
if(CAN_Write_Dev(3,&canT4) == CAN_SUCCESS);

canR5.can_buffer = &swobjdum[5] ;
canR5.CAN_trans_ucb = rx_ucb ;
if(CAN_Read_Dev(0,&canR5) == CAN_SUCCESS );
canR6.can_buffer = &swobjdum[6] ;
canR6.CAN_trans_ucb = rx_ucb ;
if(CAN_Read_Dev(0,&canR6) == CAN_SUCCESS );
canR7.can_buffer = &swobjdum[7] ;
canR7.CAN_trans_ucb = rx_ucb ;
if(CAN_Read_Dev(0,&canR7) == CAN_SUCCESS );
canR8.can_buffer = &swobjdum[8] ;
```

```
canR8.CAN_trans_ucb = rx_ucb ;
if(CAN_Read_Dev(0,&canR8) == CAN_SUCCESS );
canR9.can_buffer = &swobjdum[9] ;
canR9.CAN_trans_ucb = rx_ucb ;
if(CAN_Read_Dev(0,&canR9) == CAN_SUCCESS );
  while(1)
  {
  }

}

volatile unsigned int cx = 0;
void delay(void)
{
    volatile int cc = 0;
    cx = 0;
    for (cc = 0 ; cc < 5 ; cc++)
    {
      for (cx = 0; cx < 0xFFFFF ; cx++);
    }
}

IFX_UINT32 tx_s = 0, tx_f = 0, rx_s = 0, rx_f = 0;

void rx_ucb(struct can_trans *t, CAN_STATUS s)
{
  if(s == CAN_SUCCESS)
  {
    rx_s++;
  }
  else
  {
    rx_f++;
  }
}

void tx_ucb(struct can_trans *t, CAN_STATUS s)
{
  if(s == CAN_SUCCESS)
  {
    tx_s++;
  }
  else
  {
    tx_f++;
  }
}
```

5.3 CAN eLinux driver application

```
/*
  This test file used to test CAN node 0 with CAN analyser.
  Transaction include receive and transmit single frame.
  Baud rate        : 50k
  Message ID       : 0x15555554
  Identifier type : Extended

Linux Setup:
  1. Please check the Major number assigned to CAN driver by Linux kernel.
     Major number displayed on console at the time of Linux boot-up.
     Eg., CAN device registered with major number = 254
     Say major number is 254.
     (or)
     Move to /proc directory.
    Type ‘cat devices’ at command prompt to know major number associated
     with each device.
     e.g.
    254 tc1130can; then 254 is the major number associated with CAN eLinux
      driver.


  2. Create a node for CAN node 0 in linux kernel system by using following
     command at console.
     > mknod /dev/tc1130can0 c 254 0
     /dev/tc1130can0 --> Node name for CAN node 0
     c               --> Represents character driver
     254             --> Major number
     0               --> Minor number assigned to CAN node 0.


  3. Configure CAN analyser to recive and transmit CAN message frames.

  4. Start application.

     Note:
    Please strictly follow the following rules to create nodes in Linux
     kernel.

  Node number           Minor number
  -----------           ------------
  Node 0                     0
  Node 1                     1
  Node 2                     2
  Node 3                     3
*/
```

```
/*Include Linux kernel header files*/
#include <stdio.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

/*Include CAN LLD header files*/
#include "SYS_API.h"
#include "CAN_API.h"

/*
  Message objects to receive and tranmsit frames.
*/
CAN_MESSAGE_OBJECT rx_buf[1] ;
CAN_MESSAGE_OBJECT tx_buf[1] ;

/*
  Variables used for trannsfer requests and configuration
  of receive message objects.
*/
CAN_TRANSFER rx0_T, tx0_T, conf;

int main()
{
int can0_fd; /*File descriptor*/
volatile unsigned short i = 0;
CAN_COM_PARMS can_init_parms;

/*Init the CAN Mode structure*/
can_init_parms.can_mode.Loop_Back = 0 ;

/*Node NBTR*/
can_init_parms.can_bit_timing.Baud_Prescaler = 0 ;
can_init_parms.can_bit_timing.Divider_Mode = 0 ;
can_init_parms.can_bit_timing.Sync_Jump = 1;
can_init_parms.can_bit_timing.Timing_Seg1 = 3 ;
can_init_parms.can_bit_timing.Timing_Seg2 = 2 ;

/**Error Counts*/
can_init_parms.can_error_counter.Receive_Count = 0 ;
can_init_parms.can_error_counter.Transmit_Count = 0 ;
can_init_parms.can_error_counter.Warn_Level = 91 ;

can_init_parms.can_frame_counter.Count_value = 0 ;
can_init_parms.can_frame_counter.Frame_Count_mode = 0 ;
can_init_parms.can_frame_counter.Frame_Count_select = 7 ;

/*Open CAN node*/
can0_fd = open("/dev/tc1130can0", 666);
if(can0_fd < 0)
{
  printf("CAN node0 failed\r\n");
}

ioctl(can0_fd, CAN_CTRL_OPEN, &can_init_parms);

for(i = 0 ; i < 1 ; i++)
{
  rx_buf[i].Data[0] = 0 ;
  rx_buf[i].Data[1] = 0;
  rx_buf[i].Data[2] = 0;
  rx_buf[i].Data[3] = 0;
  rx_buf[i].Data[4] = 0;
  rx_buf[i].Data[5] = 0;
  rx_buf[i].Data[6] = 0;
  rx_buf[i].Data[7] = 0;
  rx_buf[i].Mask = 0x1FFFFFFF ;
  rx_buf[i].ID  = 0x15555554 ; //extended identifier
  rx_buf[i].MOCfg = 0x0084 ;  //extended identifier
}

conf.can_buffer = &rx_buf[0];
conf.can_buffer_size = 1;

ioctl(can0_fd, CAN_CTRL_CFG_RXBUFF, &conf); /*test*/

for(i = 0 ; i < 1 ; i++)
{
  tx_buf[i].Data[0] = 1;
  tx_buf[i].Data[1] = 2;
  tx_buf[i].Data[2] = 3;
  tx_buf[i].Data[3] = 4;
  tx_buf[i].Data[4] = 5;
  tx_buf[i].Data[5] = 6;
  tx_buf[i].Data[6] = 7;
  tx_buf[i].Data[7] = 8;
  tx_buf[i].Mask = 0x1FFFFFFF ;
  tx_buf[i].ID  = 0x15555554 ; //extended identifier
  tx_buf[i].MOCfg = 0x0084 ;  //extended identifier
}
```

```
rx0_T.can_buffer = &rx_buf[0] ;
rx0_T.CAN_trans_ucb = 0;

tx0_T.can_buffer = &tx_buf[0] ;
tx0_T.CAN_trans_ucb = 0;

for(;;)
{
  read(can0_fd, &rx0_T, 1);
  write(can0_fd, (const CAN_TRANSFER *)&tx0_T, 1);
}
  return 0;
}
```

6 Application note on the disabling of interrupts in the Interrupt Service Routine

From the hardware perspective, an Interrupt Service Routine(ISR) is entered with the interrupt system globally disabled. The Low Level Driver(LLD) does not enable global interupt in the ISR, as the LLD ISRs are kept short. Most LLD ISRs invoke a callback function that was registered by the application. If required, the application may enable global interrupts (by calling ENABLE_GLOBAL_INTERRUPT()) at the beginning of the ISR callback function.

7 Application note to use CAN LLD with DAvE generated drivers

Replace MAIN.h file from LLD release package with DAvE generated MAIN.H file

Update SYS_DAVE_GEN_SYS_CLK_FREQ configuration parameter in SYS_CFG.H file with the DAvE generated system clock value (The DAvE generated system clock can be observed in MAIN.c file).

8 Related Documentation

- Infineon Technologies HAL/Device Driver Software Suite Overview
- Ethernet, SSC and ASC HAL User Guide
- elinux_canlld_readme.txt file

9 Open Issues

Under certain circumstances, the MultiCAN module transmits the most significant bit (MSB) of the identifier wrong.

Please refer *multican_maintenance_tx_id_msb1.pdf* document for additional information.

10 Limitations

CAN LLD cannot be used with GNU compiler with optimization level 2 and greater.

11 Appendix A - Infineon IFX types

To overcome the problem of the size of data types changing between different compilers the HAL software modules use IFX types. These are defined in a file called COMPILER.H which is generated for each compiler that is supported. Table 1 presents these IFX types.

Table 1 Table of IFX Data Types

IFX_UINT8	Unsigned 8 bit integer
IFX_UINT16	Unsigned 16 bit integer
IFX_UINT32	Unsigned 32 bit integer
IFX_SINT8	Signed 8 bit integer
IFX_SINT16	Signed 16 bit integer
IFX_SINT32	Signed 32 bit integer
IFX_VUINT8	Unsigned 8 bit volatile integer
IFX_VUINT16	Unsigned 16 bit volatile integer
IFX_VUINT32	Unsigned 32 bit volatile integer
IFX_VSINT8	Signed 8 bit volatile integer
IFX_VSINT16	Signed 16 bit volatile integer
IFX_VSINT32	Signed 32 bit volatile integer
IFX_SFLOAT	Signed flaot
IFX_STINT8	Signed static 8 bit integer
IFX_STINT16	Signed static 16 bit integer
IFX_STINT32	Signed static 32 bit integer
IFX_STUINT8	Unsigned static 8 bit integer
IFX_STUINT16	Unsigned static 16 bit integer
IFX_STUINT32	Unsigned static 32 bit integer

12 Appendix B - The System HAL

This appendix presents a brief description of the CAN related settings and options available in the System HAL.

This section defines the configurable parameters of the System HAL - interrupts, GPIO ports, and the clock. The user may change only the value associated with the macros to suit application requirements. However, the user may NOT change the name of the macro.

12.1 Data Transfer Options

Depending upon the system the HAL is operating in there may be several different options available regarding data transfers. Some systems have a DMA controller available, others have a PCP, some have both of these and some have neither. The system HAL provides enumeration constants which can be used to specify the desired data transfer option, this enumeration is given the typedef name SYS_TRANS_MODE.

Table 2 presents the possible transfer options.

Table 2 Data Transfer Options

SYS_TRNS_DMA	Use the DMA controller to move the data
SYS_TRNS_PCP	Use the PCP to manage the transfer (requires a additional PCP CAN program module)
SYS_TRNS_MCU_INT	Use the microcontroller unit to manage the transfer using interrupts.
SYS_TRNS_MCU	Use the microcontroller unit to manage the transfer by polling the peripheral.

12.2 SYS HAL Configurable Parameters

This section defines the configurable parameters of the SYS HAL - interrupts, GPIO ports, and the clock. The user may change only the value associated with the macros to suit application requirements. However, the user may NOT change the name of the macro.

12.3System Clock Frequency

The clock must be operational before the controller can function. This clock is connected to the peripheral clock control registers, so changing the value of this clock frequency will affect all peripherals. The individual peripherals can scale down this frequency according to their requirements, for more details please refer to the corresponding user guide documents.

12.3.1SYS_CFG_USB_DEVICE_ENABLE macro

```
#define SYS_CFG_USB_DEVICE_ENABLE
```

Defined in: SYS_CFG.H

User needs to configure whether the USB device has been used.

1

Equate this macro to 1 if onchip USB device is used.

0

Equate this macro to 0 if usb device is not used (default).

12.3.2SYS_CFG_USB_ONCHIP_CLK macro

```
#define SYS_CFG_USB_ONCHIP_CLK
```

Defined in: SYS_CFG.H

User can configure the USB clock generation logic whether it internal or external. If clock is external, it will be derived from pin P4.0.

1

Equate this macro to 1 for internal clock generation

0

Equate this macro to 0 for external clock generation

12.3.3SYS_CFG_USB_CLK_DIVISOR macro

```
#define SYS_CFG_USB_CLK_DIVISOR
```

Defined in: SYS_CFG.H

User needs to configure the USB clock ratio based upon the USB clock frequency. Since clock frequency can be either 48 MHZ, 96 or 144 MHZ, the ratio can 1, 2 or 3 respectively.

12.3.4SYS_CFG_OSC_FREQ macro

```
#define SYS_CFG_OSC_FREQ
```

Defined in: SYS_CFG.H

User has to configure this with external applied frequency.

12.3.5SYS_CFG_CLK_MODE macro

```
#define SYS_CFG_CLK_MODE
```

Defined in: SYS_CFG.H

User needs to configure this macro to any one of the following clock operation mode.

0

Direct drive (CPU clock directly derived from external applied frequency, N, P, and K values are not considered).

1

PLL mode (N, P, K values will be considered to derive CPU clock frequency from external frequency)

2

VCO bypass/pre-scalar mode (N value not considered to derive CPU clock from external frequency).

12.3.6SYS_CFG_FREQ_SEL macro

```
#define SYS_CFG_FREQ_SEL
```

Defined in: SYS_CFG.H

This define decide the frequency ration between CPU and system, this is independent from the clock mode selection(SYS_CFG_CLK_MODE).

0	Ratio of fcpu/fsys is 2.
1	Ratio of fcpu/fsys is 1 i.e. fcpu = fsys.

12.3.7SYS_CFG_KDIV macro

```
#define SYS_CFG_KDIV
```

Defined in: SYS_CFG.H

User has to configure this with a value ranges from 1 to 16, used for both PLL and VCO bypass modes.

12.3.8SYS_CFG_PDIV macro

```
#define SYS_CFG_PDIV
```

Defined in: SYS_CFG.H

User has to configure this with a value ranges from 1 to 8, used for both PLL and VCO bypass modes.

12.3.9SYS_CFG_NDIV macro

```
#define SYS_CFG_NDIV
```

Defined in: SYS_CFG.H

User has to configure this with a value ranges from 1 to 128, used only for PLL mode.

12.3.9.1Comments

Advisable value range is 20 to 100.

12.3.10SYS_CFG_FIX_TC1130A_BUG macro

```
#define SYS_CFG_FIX_TC1130A_BUG
```

Defined in: SYS_CFG.H

User can use this definition for software workaround done for TC1130A at system driver and not at module level.

1	Enbale software work-around for hardware bug fixes.
0	Disbale software work-around for hardware bug fixes.

12.4Interrupt priorities configuration

The following priorities are used for interrupts. Corresponding to these priorities ISR code will be placed in Interrupt base Vector Table. The user can edit the priorities according to application requirements. These priorities will be static.

Priorities range from 1 to 255. Each interrupt should have a unique priority. Lowest priority is 1 and the highest priority is 255.

12.4.1SYS_CAN_SRN0 macro

```
#define SYS_CAN_SRN0
```

Defined in: SYS_CFG.H

Service request node 0 interrupt priority of CAN.

12.4.2SYS_CAN_SRN1 macro

```
#define SYS_CAN_SRN1
```

Defined in: SYS_CFG.H

Service request node 1 interrupt priority of CAN [receive message object interrupt priority for node0].

12.4.3SYS_CAN_SRN2 macro

```
#define SYS_CAN_SRN2
```

Defined in: SYS_CFG.H

Service request node 2 interrupt priority of CAN[receive message object interrupt priority for node1].

12.4.4.SYS_CAN_SRN3 macro

```
#define SYS_CAN_SRN3
```

Defined in: SYS_CFG.H

Service request node 3 interrupt priority of CAN[receive message object interrupt priority for node2].

.

12.4.5SYS_CAN_SRN4 macro

```
#define SYS_CAN_SRN4
```

Defined in: SYS_CFG.H

Service request node 4 interrupt priority of CAN[receive message object interrupt priority for node3].

.

12.4.6SYS_CAN_SRN5 macro

```
#define SYS_CAN_SRN5
```

Defined in: SYS_CFG.H

Service request node 5 interrupt priority of CAN[transmit message object interrupt priority for node0].

.

12.4.7SYS_CAN_SRN6 macro

```
#define SYS_CAN_SRN6
```

Defined in: SYS_CFG.H

Service request node 6 interrupt priority of CAN[transmit message object interrupt priority for node1].

.

12.4.8SYS_CAN_SRN7 macro

```
#define SYS_CAN_SRN7
```

Defined in: SYS_CFG.H

Service request node 7 interrupt priority of CAN[transmit message object interrupt priority for node2].

12.4.9.SYS_CAN_SRN8 macro

```
#define SYS_CAN_SRN8
```

Defined in: SYS_CFG.H

Service request node 8 interrupt priority of CAN[transmit message object interrupt priority for node3].

.

12.4.10SYS_CAN_SRN9 macro

```
#define SYS_CAN_SRN9
```

Defined in: SYS_CFG.H

Service request node 9 interrupt priority of CAN.

12.4.11SYS_CAN_SRN10 macro

```
#define SYS_CAN_SRN10
```

Defined in: SYS_CFG.H

Service request node 10 interrupt priority of CAN.

12.4.12SYS_CAN_SRN11 macro

```
#define SYS_CAN_SRN11
```

Defined in: SYS_CFG.H

Service request node 11 interrupt priority of CAN.

12.4.13SYS_CAN_SRN12 macro

```
#define SYS_CAN_SRN12
```

Defined in: SYS_CFG.H

Service request node 12 interrupt priority of CAN.

12.4.14SYS_CAN_SRN13 macro

```
#define SYS_CAN_SRN13
```

Defined in: SYS_CFG.H

Service request node 13 interrupt priority of CAN.

12.4.15SYS_CAN_SRN14 macro

```
#define SYS_CAN_SRN14
```

Defined in: SYS_CFG.H

Service request node 14 interrupt priority of CAN.

12.4.16SYS_CAN_SRN15 macro

```
#define SYS_CAN_SRN15
```

Defined in: SYS_CFG.H

Service request node 15 interrupt priority of CAN.

12.5GPIO Port Configuration Parameters

This section defines the configurable port settings of the peripherals. These macros define the following parameters:

Peripheral Module
- Name of the macro which includes the name of the peripheral and the port line (Transmit/Receive).

Port
- Port Number.

Pin
- Bit Number in the Port.

Dir
- Value of the bit in the Dir register.

Alt0
- Value of the bit in the Altsel0 register.

Alt1
- Value of the bit in the Altsel1 register.

Od
- Value of the bit in the Open Drain register.

Pullsel
- Value of the bit in the Pull up/Pull down selection register.

Pullen
- Value of the bit in the Pull up/Pull down enable register.

Note: User may use -1, to indicate an unused (or don't care) value.

These macros should be defined has a set of values in above sequence and separated by commas (,).

E.g. #define SYS_GPIO_ASC0_TX 1, 7, 1, 1, -1, -1, -1, -1

12.5.1SYS_GPIO_CAN0_RX macro

```
#define SYS_GPIO_CAN0_RX
```

Defined in: SYS_CFG.H

Port configuration used for CAN0 receive transfer line.

12.5.2SYS_GPIO_CAN0_TX macro

```
#define SYS_GPIO_CAN0_TX
```

Defined in: SYS_CFG.H

Port configuration used for CAN0 transmit transfer line.

12.5.3SYS_GPIO_CAN1_RX macro

```
#define SYS_GPIO_CAN1_RX
```

Defined in: SYS_CFG.H

Port configuration used for CAN1 receive transfer line.

12.5.4SYS_GPIO_CAN1_TX macro

```
#define SYS_GPIO_CAN1_TX
```

Defined in: SYS_CFG.H

Port configuration used for CAN1 transmit transfer line.

12.5.5SYS_GPIO_CAN2_RX macro

```
#define SYS_GPIO_CAN2_RX
```

Defined in: SYS_CFG.H

Port configuration used for CAN2 receive transfer line.

12.5.6SYS_GPIO_CAN2_TX macro

```
#define SYS_GPIO_CAN2_TX
```

Defined in: SYS_CFG.H

Port configuration used for CAN2 transmit transfer line.

12.5.7SYS_GPIO_CAN3_RX macro

```
#define SYS_GPIO_CAN3_RX
Defined in: SYS_CFG.H
```

Port configuration used for CAN3 receive transfer line.

12.5.8SYS_GPIO_CAN3_TX macro

```
#define SYS_GPIO_CAN3_TX
```

Defined in: SYS_CFG.H

Port configuration used for CAN3 transmit transfer line.