

XC2000

Microcontroller Infineon Software Architecture

User's Manual

Lin Driver

V 1.6

2011-01

Edition 2011-01

**Published by
Infineon Technologies AG
81726 Munich, Germany**

**© 2011 Infineon Technologies AG
All Rights Reserved.**

LEGAL DISCLAIMER

THE INFORMATION GIVEN IN THIS DOCUMENT IS GIVEN AS A HINT FOR THE IMPLEMENTATION OF THE INFINEON TECHNOLOGIES COMPONENT ONLY AND SHALL NOT BE REGARDED AS ANY DESCRIPTION OR WARRANTY OF A CERTAIN FUNCTIONALITY, CONDITION OR QUALITY OF THE INFINEON TECHNOLOGIES COMPONENT. THE RECIPIENT OF THIS DOCUMENT MUST VERIFY ANY FUNCTION DESCRIBED HEREIN IN THE REAL APPLICATION. INFINEON TECHNOLOGIES HEREBY DISCLAIMS ANY AND ALL WARRANTIES AND LIABILITIES OF ANY KIND (INCLUDING WITHOUT LIMITATION WARRANTIES OF NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OF ANY THIRD PARTY) WITH RESPECT TO ANY AND ALL INFORMATION GIVEN IN THIS DOCUMENT.

Information

For further information on technology, delivery terms and conditions and prices, please contact the nearest Infineon Technologies Office (www.infineon.com).

Warnings

Due to technical requirements, components may contain dangerous substances. For information on the types in question, please contact the nearest Infineon Technologies Office.

Infineon Technologies components may be used in life-support devices or systems only with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

Document Change History

Date	Version	Changed By	Change Description
2009-12-16	1.0 Draft	Kazmi	Creation
2010-01-28	1.0	Kazmi	Modifications for review comments
2010-04-16	1.1	Puneetha	Included configuration for DAvE
2010-05-19	1.2	Puneetha	Included changes for UTP AI00053852
2010-05-19	1.3	Puneetha	Updated as per Review Comments
2010-07-21	1.4	Kazmi	Included Frame counter API's (5.32, and 5.33) for UTP ID AI00055632
2010-12-02	1.5	Puneetha	Included changes for UTP AI00057690 , AI00056263 , AI00057690
2011-01-12	1.6	Puneetha	Modifications to Lin_TPCallBack explanation

We Listen to Your Comments

Is there any information in this document that you feel is wrong, unclear or missing?
Your feedback will help us to continuously improve the quality of this document.
Please send your proposal (including a reference to this document) to:

mcdocu.comments@infineon.com



Table of Contents	Page
1 Introduction9	9
1.1 Scope9	9
1.2 Abbreviations.....10	10
1.3 References10	10
2 Lin Driver Overview11	11
2.1 SW – HW mapping and HW utilization11	11
2.1.1 HW Communication Channel.....12	12
2.1.2 Timers12	12
2.1.3 Ports12	12
2.1.4 System Clock13	13
3 File Structure13	13
4 Configuration Documentation14	14
4.1 Configuration Concept14	14
4.2 Driver Configuration14	14
4.2.1 LIN Interface configuration14	14
4.2.1.1 Interface Selection14	14
4.2.2 Primary Timer Selection.....14	14
4.2.2.1 Slave Timer14	14
4.3 LLD feature Selection.....15	15
4.3.1 Compliance to specification version.....15	15
4.3.2 Event / Sporadic Frame feature Support15	15
4.3.3 Sleep/Wakeup feature Support.....15	15
4.3.4 Signal Read/Write Support.....15	15
4.3.5 Diagnostic feature Support.....15	15
4.3.6 Frame ID Assignment Feature Support15	15
4.3.7 LLD Full Feature Support.....16	16
4.4 HW Configurations16	16
4.4.1 Peripheral frequency16	16
4.4.2 Port Selection16	16
4.4.3 Port Pin Selection.....16	16
4.4.4 Baud Rate Timer Reload Value17	17
4.4.5 USIC Interrupt Priority17	17
4.4.6 Primary Timer Priority17	17
4.5 Driver Buffer Sizes17	17
4.5.1 Diagnostic Transmit buffer Size17	17
4.5.2 Diagnostic Receive buffer Size17	17
4.5.3 Frame Buffer Size17	17
4.6 Frame related configurations18	18
4.6.1 Number of Frames18	18
4.6.2 Response Error Frame Number.....18	18
4.6.3 Response Error Bit Location18	18
4.6.4 Number of Sporadic Frames18	18
4.6.5 Number of Event Triggered Frames18	18
4.6.6 Number of Unconditional Frames associated with Event Triggered Frames19	19
4.6.7 Number of Frames19	19
4.6.8 LIN frame properties19	19
4.6.9 Event Triggered Frame Definition19	19
4.6.10 Sporadic Frame Definition.....19	19
4.7 Signal Configuration20	20
4.7.1 Signal definition20	20
4.7.2 Number of Signals.....20	20
4.7.3 Signal Prototype Selection20	20
4.8 Node configurations20	20
4.8.1 Node Identification20	20
4.8.1.1 Supplier ID.....20	20
4.8.1.2 Function ID20	20

Table of Contents	Page
4.8.1.3 Variant ID	21
4.8.1.4 NAD	21
4.8.1.5 Transport Layer buffer Management	21
4.8.1.6 Transport Layer Buffer Size	21
4.9 Configuration Using DAVE	22
4.9.1 Creating the configuration From the UI	22
4.9.2 Creating the configuration from LDF File	27
5 API description	29
5.1 l_sys_init.....	29
5.2 l_ifc_init	29
5.3 l_ifc_connect	29
5.4 l_ifc_disconnect.....	29
5.5 l_ifc_read_status	30
5.6 l_ifc_goto_sleep	30
5.7 l_ifc_wake_up.....	30
5.8 l_sys_irq_disable.....	30
5.9 l_sys_irq_restore	31
5.10 l_flg_tst.....	31
5.11 l_flg_clr.....	31
5.12 Signal read APIs	32
5.12.1 Scalar Signal Read API's	32
5.12.2 Bytes Array Signal Read API's.....	32
5.13 Signal write APIs	33
5.13.1 Scalar Signal write API's	33
5.13.2 Bytes Array Signal Write API's	33
5.14 ld_get_raw.....	33
5.15 ld_put_raw.....	34
5.16 ld_raw_rx_status	34
5.17 ld_raw_tx_status	34
5.18 Lin_TpCallBack	34
5.19 ld_send_message	35
5.20 ld_receive_message	35
5.21 ld_tx_status	36
5.22 ld_rx_status	36
5.23 Frmld_FrmCntInit (uint8 u8FrameId)	37
5.24 Frmld_FrmCntRead (uint8 u8FrameId)	37
6 Example code	38
6.1 Creation of slave interface application	38
7 Limitations	39
7.1 Limitation 1	39
7.2 Limitation 2	39

List of Figures**Page**

Figure 1	SW HW Driver Mapping	12
Figure 2	Creating a new project	22
Figure 3	Selecting XC2236N	22
Figure 4	ECU bubble	23
Figure 5	ECU page	23
Figure 6	Interrupts Priority setting	24
Figure 7	LDF Bubble	24
Figure 8	Sample Master Information	25
Figure 9	Node Tab	25
Figure 10	Signal Tab	26
Figure 11	Frame Tab	26
Figure 12	Schedule Tab	27
Figure 13	Sample Master Information	28
Figure 14	Selecting an LDF file	28

List of Tables

Page

Table 1	List of supported microcontrollers	9
Table 2	List of supported peripherals.....	9
Table 3	Abbreviations used in this Document.....	10
Table 4	Supported Port pins and their usage	12
Table 5	File Contents	13

List of Code Listings

Page

Code Listing 1	LIN_Test.c file example code	38
----------------	------------------------------------	----

1 Introduction

This user manual provides information regarding the functionality, configuration parameters and API implementation for the Lin driver.

The Lin driver is built according to the Lin consortium specification, refer [1] and [2]

This user manual is intended to help the user to get acquainted with the driver implementation for 16 bit Infineon microcontroller. It is the documentation that describes how to use the driver.

1.1 Scope

This document addresses the following features of the implementation,

- Microcontroller HW peripherals used to realize the driver
- Driver File structure
- Configurations offered from the driver
- API specification
- Limitations and assumptions

This documentation applies to the Lin driver variant: version 2.0, version 2.1.

This document addresses the driver implementation for the following microcontrollers.

Table 1 List of supported microcontrollers

Microcontroller Family	Microcontroller	Type
XC2000	XC223xN, XC233xB, XC2734X, XC2234L, XC2331D, XC2733X, XC2220U, XC2320S, XC2722X, XC2210U, XC2310S, XC2712X, XC2224L, XC2321D, XC2723X Series of Microcontrollers	XC2000 is Infineon Technologies 16 bit microcontrollers family for automotive and industrial applications

Some of the peripheral units/feature might not be available in all the variants. In such cases, the relevant information should be discarded for the variant where the unit/feature is not applicable. The following table gives the available units/features in the variants of XC2000 Family for driver usage.

Table 2 List of supported peripherals

Microcontroller	Peripheral used for communication	Remarks
XC223xN, XC233xB, XC2734X,	USIC0, USIC1, USIC2	Universal Serial Interface Channel module, which can be configured as a serial communication channel. Each USIC module provides two channels which can be configured for serial communication required by this driver.
XC2234L, XC2331D, XC2733X, XC2224L, XC2321D, XC2723X	USIC0, USIC1	Universal Serial Interface Channel module, which can be configured

Microcontroller	Peripheral used for communication	Remarks
		as a serial communication channel. Each USIC module provides two channels which can be configured for serial communication required by this driver.
XC2320S, XC2220U, XC2722X, XC2210U, XC2310S, XC2712X	USIC0	Universal Serial Interface Channel module, which can be configured as a serial communication channel. Each USIC module provides two channels which can be configured for serial communication required by this driver.

1.2 Abbreviations

The abbreviations used inside this document are explained in [Table 3](#).

Table 3 Abbreviations used in this Document

Abbreviations	Explanation
μC	Microcontroller
μs	Micro Second
API	Application Programming Interface
ECU	Electronic Control Unit
GPT	General Purpose Timer
LIN	Local Interconnect Network
HW	Hardware
MCU	MicroController Unit
RAM	Random Access Memory
ROM	Read Only Memory
SW	Software
USIC	Universal Asynchronous Receiver Transmitter
USIC	Universal Serial Interface Channel

1.3 References

This section lists down all relevant documents for this User Manual.

Some of the artifacts listed below will not contain the version number. For those artifacts, kindly refer the latest version.

- [1] LIN specification package revision v2.0, 23 SEP 2003, V2.0, <http://www.lin-subbus.org/>
- [2] LIN specification package revision v2.1, 24 November 2006, V2.1, <http://www.lin-subbus.org/>
- [3] XC2237 datasheet, XC223xN_ds.pdf, V1.1, July 2009
- [4] XC2234L datasheet, P11_LE_LQFP64_ds_v0_3.pdf V0.3, June 2010

2 Lin Driver Overview

The LIN driver has been developed according the LIN specification v2.0 / v2.1. It can change its behavior to support the version based upon a compiler switch.

Following features are supported:

- Basic Frames
- Event Triggered Frame
- Sleep / Wakeup Support
- Signal read & writes APIs (static and dynamic)
- Diagnostic Frames
- Class 2 Diagnostic for V2.1
- Node configuration services (Assign NAD, Assign Frame ID Range in V2.1 and Assign Frame ID in V2.0, Conditional Change NAD and Read by ID 0x00)
- Auto Baud rate detecting with the range from 6000bps to 20000bps
- Support of Full Transport Protocol for Class II Diagnostics
- Choice of buffer management for transport Protocol (internal by driver / external)

Implementation of the driver is developed and optimized for Tasking Viper compiler.

2.1 SW – HW mapping and HW utilization

This section introduces the user to the HW features used for implementation. **Figure 1** illustrates the hardware peripherals and their interaction with the software drivers.

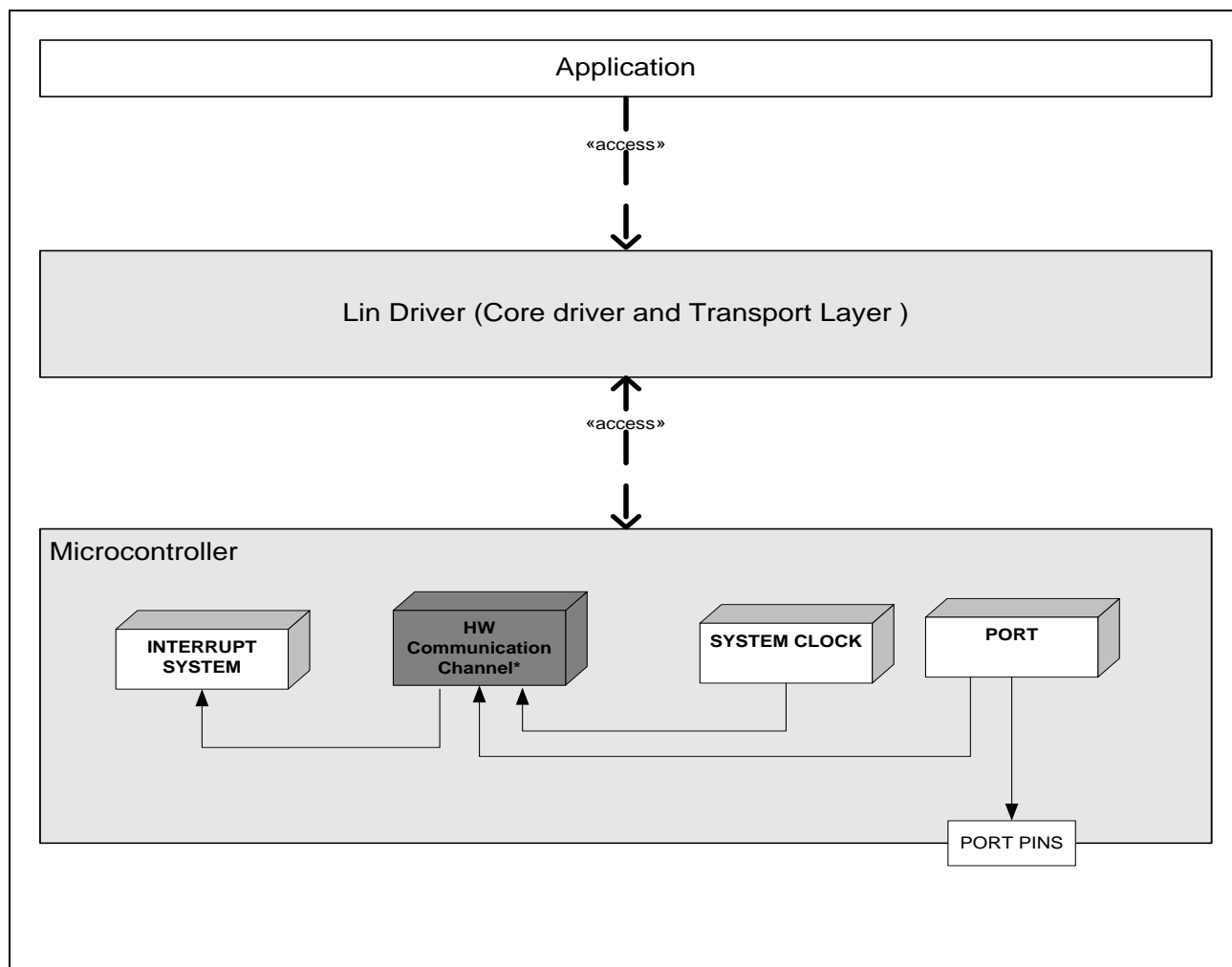


Figure 1 SW HW Driver Mapping

2.1.1 HW Communication Channel

The Lin driver configures and utilizes the underlying microcontroller peripheral in order to transmit / receive the serial data in asynchronous mode. The request to transmit data from the driver is handled within the channel interrupts service routines. Refer Table 2 to identify the communication channel usage based upon the supported HW platform.

2.1.2 Timers

When the driver is configured as a slave, the timer is utilized to fulfill various timeout requirements towards sleep / wakeup functionality and the transport protocol layer functionality.

Timers T2 and T6 of GPTA are available to be configured for this driver.

2.1.3 Ports

Depending upon the selected platform, the Port peripheral controls all access to the pins required by Lin driver. Following table identifies the port pins which are supported via the driver

Table 4 Supported Port pins and their usage

Port Pins	USIC 0 Channel 0	USIC 0 Channel 1	USIC 1 Channel 0	USIC 1 Channel 1	USIC 2 Channel 0	USIC 2 Channel 1
P2.3	Input/ Output	Input				
P2.4	Input	Output				
P2.7			Input			
P2.9		Output				
P2.10		Input/ Output				
P2.13			Input	Input / Output		
P5.8					Input	
P5.10						Input
P6.0				Input		
P6.1				Input / Output		
P10.0	Input	Input/Output		Input		
P10.1	Input/ Output					
P10.5					Output	
P10.6	Input/ Output		Input /Output			
P10.7		Input/ Output				
P10.8						Output
P10.10			Output			
P10.12	Input/ Output		Input/ Output			
P10.13			Input/ Output			
P10.14		Input/ Output				
P10.15		Output	Output			

2.1.4 System Clock

All peripheral clocks and baud rate are derived from system clock. Setting of the system clock is handled outside the Lin driver.

3 File Structure

This section provides details about the driver file structure.

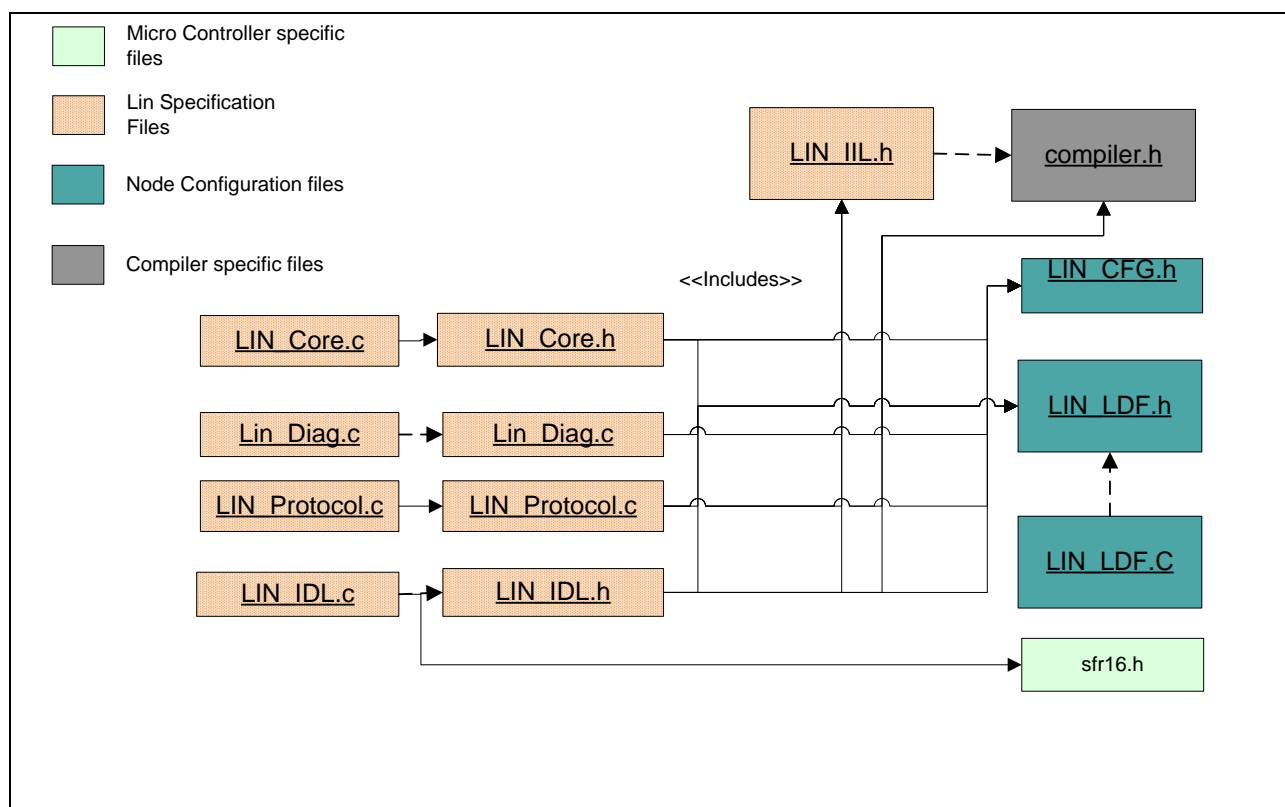


Table 5 File Contents

File Name	File Contents
Lin_CFG.h	Contains hardware configurations. E.g. Primary Timer Number etc.
Compiler.h	Contains compiler specific key words.
Lin_IIL.h	Contains defines / typedefs for LIN_IIL.c and has LIN API definitions. This file to be included in the application files which require LIN functions.
Lin_LDF.h, Lin_LDF.c	Contains LIN Node configuration data
Lin_Core.c /h	Contains LIN Core API.
Lin_IDL.c / h	Contains microcontroller specific functions.
Lin_Protocol.c /h	Contain ISR implementation and Data transfer State Machines
Lin_Diag.c / h	Contains API's for Lin diagnostics.
Sfr.16.h	Contains SFR definitions for Microcontroller
Defines.h	Common Definitions to be used within the Driver (not depicted in figure above)

The user of Lin Driver needs to import Lin_LDF.h, LIN_API.h and Lin_IIL.h file to import Lin functionality.

4 Configuration Documentation

The following chapters summarize

- Configuration concept used for structuring the Driver
- All configuration parameters and their description

4.1 Configuration Concept

The development of Basic Software Drivers involves the following development cycles:

- Compiling,
- Linking and
- Downloading of the executable to ECU memory.

Configuration of parameters can be done in any of these process-steps pre-compile time, link time or even post-build time. This driver is built with a precompile configuration support only, in line with the specifications.

4.2 Driver Configuration

Following parameters are provided in order to configure the driver

4.2.1 LIN Interface configuration

This requirement describes the interface configuration features provided by the driver

4.2.1.1 Interface Selection

4.2.1.1.1 Interface- Slave

This defines the LIN node as Slave.

Type: #define / Macro

Name: "_IFC_SLAVE"

Location: LIN_CFG.h

4.2.2 Primary Timer Selection

4.2.2.1 Slave Timer

This defines the Timer Number when Node is defined as Slave.

Type: #define / Macro

Name: "_IFC_SCND_TMR_NUM"

Location: LIN_CFG.h

Usage : 0 - GPT_T2

1 - GPT_T6

4.3 LLD feature Selection

4.3.1 Compliance to specification version

This defines the LIN specification version number.

Type: #define/ Macro
Name: "LIN_V_2_1"
Location: LIN_CFG.h

4.3.2 Event / Sporadic Frame feature Support

This define enables the support for event triggered and sporadic frames.

Type: #define / Macro
Name: "_EVNT_SPRDC_FRM_SPRT"
Location: LIN_CFG.h.

4.3.3 Sleep/Wakeup feature Support

This define enables the support for sleep, wakeup signals and requests.

Type: #define / Macro
Name: "_WKUP_SLP_SPRT"
Location: LIN_CFG.h.

4.3.4 Signal Read/Write Support

This definition enables the functionality of reading and writing of signal information from/to the frame data buffer (e.g. l_bool_rd, l_bool_wr, l_u8_rd and l_u8_wr etc.)

Type: #define / Macro
Name: "_SIG_RDWR_FUNC_SPRT"
Location: LIN_CFG.h.

4.3.5 Diagnostic feature Support

This definition enables the diagnostic feature (functionality of publishing and subscription to master request and slave response frames).

Type: #define / Macro
Name: "_DGNSTC_FRM_SPRT"
Location: in LIN_CFG.h.

4.3.6 Frame ID Assignment Feature Support

This definition enables the functionality of changing the current frame id to the requested frame ID based on message ID. (used in V2.0)

Type: #define / Macro
Name: "_DGNSTC_ASGN_FRM_SPRT"

Location: in LIN_CFG.h.

4.3.7 LLD Full Feature Support

This enables the slave/master with full functionality supporting all features. Defining this will select the following.

"_DGNSTC_FRM_SPRT",
"_SIG_RDWR_FUNC_SPRT",
"_WKUP_SLP_SPRT", and
"_EVNT_SPRDC_FRM_SPRT"

Type: #define / Macro

Name: "_FEATURES"

Location: LIN_CFG.h

4.4 HW Configurations

4.4.1 Peripheral frequency

This defines the peripheral frequency.

This value is used to derive the baud rate values for USIC and input frequency for timers.

Type: #define / Macro

Name: "PRPL_FREQ"

Location: LIN_CFG.h

4.4.2 Port Selection

Defines the port pins for the usage of receive and transmit by USIC.

Type: #define / Macro

Name: "LIN_PORT_IN" and "LIN_PORT_OUT"

Location: LIN_CFG.h

4.4.3 Port Pin Selection

Defines the port pins for the usage of receive and transmit by USIC.

Type: #define / Macro

Name: "LIN_PIN_IN" and "LIN_PIN_OUT"

Location: LIN_CFG.h

4.4.4 Baud Rate Timer Reload Value_

This defines the BG value used to derive the required baud rate and program BG registers.

Type: #define / Macro

Name: "_PDIV_VAL"

Location: LIN_CFG.h

4.4.5 USIC Interrupt Priority

This defines the USIC interrupt priority.

Type: #define / Macro

Name: "USIC_REQ0_ILVL_PRIO", "USIC_REQ1_ILVL_PRIO" and

Location: LIN_CFG.h.

Range is 0 - 15.

4.4.6 Primary Timer Priority

This defines the primary timer interrupt priority.

Type: #define / Macro

Name: "GPT_REQ_ILVL_PRIO"

Location: LIN_CFG.h.

Range is 0 - 15.

4.5 Driver Buffer Sizes

4.5.1 Diagnostic Transmit buffer Size

This defines the transmit buffer size for diagnostic frames (more specific to v2.0 requirements and for raw diagnostics API).

Type: #define / Macro.

Name: "_DGNSTC_TBUFF_SZ"

Location: LIN_CFG.h.

4.5.2 Diagnostic Receive buffer Size

This defines the transmit buffer size for diagnostic frames (more specific to v2.0 requirements and for raw diagnostics API).

Type: #define / Macro.

Name: "_DGNSTC_RBUFF_SZ"

Location: LIN_CFG.h.

4.5.3 Frame Buffer Size

This defines the total frame buffer size. This is calculated as the sum of the number of bytes used by all LIN frames (from the length of the frames subscriber / publisher).

Type: #define / Macro.
Name: "LIN_SYS_TOTAL_FRM_BUFF_SZ"
Location: LIN_LDF.h.

4.6 Frame related configurations

4.6.1 Number of Frames

This defines the total number of frames.

Type: #define / Macro.
Name: "LIN_SYS_NUM_FRMS"
Location: LIN_LDF.h.

4.6.2 Response Error Frame Number

This defines the response error frame number.

Type: #define / Macro
Name: "_IFC_RESP_ERR_FRM_NUM"
Location: LIN_CFG.h.
Range is 0-59.

4.6.3 Response Error Bit Location

This defines the bit location in the response error frame. LLD will set this bit on a failure in transmission or reception of frame data.

Type: #define / Macro.
Name: "_IFC_RESP_ERR_FRM_BIT_LOC"
Location: LIN_CFG.h.
Range is 0 - 63.

4.6.4 Number of Sporadic Frames

This defines the total number of sporadic frames.

Type: #define / Macro.
Name: "LIN_SYS_CFG_NUM_OF_SPRDC_FRMS"
Location: LIN_LDF.h.

4.6.5 Number of Event Triggered Frames

This defines the total number of event triggered frames.

Type: #define / Macro.
Name: "LIN_SYS_CFG_NUM_OF_EVNT_FRMS"
Location: LIN_LDF.h.

4.6.6 Number of Unconditional Frames associated with Event Triggered Frames

This defines the maximum number of unconditional frames associated with any of the event triggered frames.

Type: #define / Macro.
Name: "LIN_SYS_CFG_MAX_FRM_PER_EVNT_FRM"
Location: LIN_LDF.h.

4.6.7 Number of Frames

This defines the number of frames. This is used to refer LLD_frm_data_updt_flg, LLD_frm_data_ovwr_flg and LLD_frm_data_usage_flg.

Type: #define / Macro.
Name: "LIN_NUM_FRMS"
Location: LIN_LDF.h.

4.6.8 LIN frame properties

This defines the Frame properties like publisher/subscriber and the frame length information.

Type: structure
Name: "LIN_sys_frm_info"
Location: LIN_LDF.h

4.6.9 Event Triggered Frame Definition

This defines the Event Triggered Frames for LIN cluster.

Type: structure
Name: "LIN_evnt_frm_base"
Location: LIN_LDF.h

4.6.10 Sporadic Frame Definition

This defines the Sporadic Frames for LIN cluster.

Type: structure
Name: "LIN_sprdc_asctd_frm_ids"
Location: LIN_LDF.h

4.7 Signal Configuration

4.7.1 Signal definition

This defines the signal name and other signal details like offset, size.

Type: enum

Name: "LIN_LDF_SIG_INFO"

Location: LIN_LDF.h

4.7.2 Number of Signals

This defines the total number of signals.

Type: #define / Macro.

Name: "LIN_SYS_CFG_NUM_OF_SIGS"

Location: LIN_LDF.h.

4.7.3 Signal Prototype Selection

This defines the total number of signals.

Type: #define / Macro.

Name: "_SIGNAL_PROTOTYPE"

Location: LIN_CFG.h.

4.8 Node configurations

4.8.1 Node Identification

4.8.1.1 Supplier ID

This defines the controller supplier ID.

Type: #define / Macro.

Name: "_SUPPLIER_ID"

Location: LIN_CFG.h.

Range is 0x0000 - 0x7FFF.

4.8.1.2 Function ID

This defines the controller Function ID.

Type: #define / Macro.

Name: "_FUNCTION_ID"

Location: LIN_CFG.h.

Range is 0x0000 - 0xFFFF.

4.8.1.3 Variant ID

This defines the controller Variant ID.

Type: #define / Macro.
Name: "_VARIANT_ID"
Location: LIN_CFG.h. Range is 0x00 - 0xFF.

4.8.1.4 NAD

This defines the NAD for the node.

Type: #define / Macro
Name: "_IFC_NAD"
Location: LIN_CFG.h.
Range is 1 - 27.

4.8.1.5 Transport Layer buffer Management

This defines the Transport Protocol Management feature.

0 - Application provides the buffer and it is responsibility of application to maintain the buffer

1 - Driver defines the buffer and the buffer size is configured by macro _TP_BUFF_SIZE

If buffer management is selected as internal, then the diagnostic buffer is allocated within the driver. The diagnostic transport layer management is done within interrupt context. Upon reception of a UDS service, a callback is issued for the application to process the response. Transport layer fragmentation is supported within interrupt context.

In case the buffer management is selected as external, the allocation of buffer is application's responsibility. The diagnostic transport layer utilizes the API's Id_put_raw, Id_get_raw, Id_send_message and Id_receive_message APIs. Upon reception of a UDS service, a callback is issued for the application to process the response. Transport layer fragmentation is supported within interrupt context

Type: #define / Macro
Name: "_TP_BUFF_MGMT"
Location: LIN_CFG.h.

4.8.1.6 Transport Layer Buffer Size

This defines buffer size in case if driver has to manage the Transport Protocol buffers.

Type: #define / Macro
Name: "_TP_BUFF_SIZE"
Location: LIN_CFG.h.
Range is 0 - 4096

4.9 Configuration Using DAVe

DAVe Configuration Support includes the support to configure the driver parameters defined above, and as an alternative, reading a LDF file to generate the Lin_LDF.c, Lin_LDF.h and Lin.CFG.h in an easy manner.

The sections below explain using this configuration using XC2236N as an example.

4.9.1 Creating the configuration From the UI

- Launch DAVe program. (Start > Programs > DAVe > dave.exe)
- Click “Create a new project”.

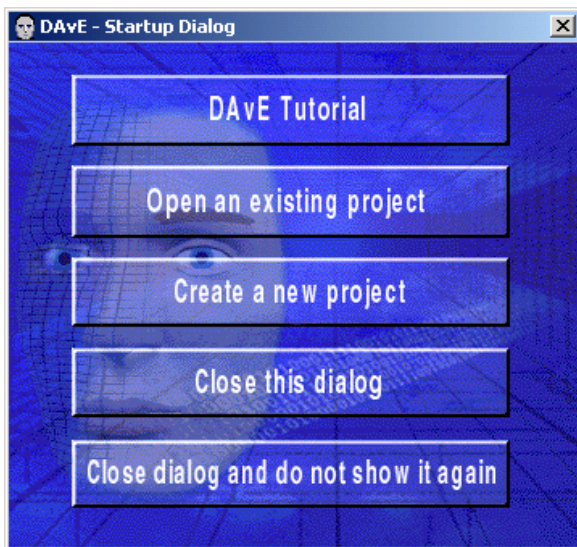


Figure 2 Creating a new project

- From “16-Bit Microcontrollers”, select “XC2236N”, Click “Create”.

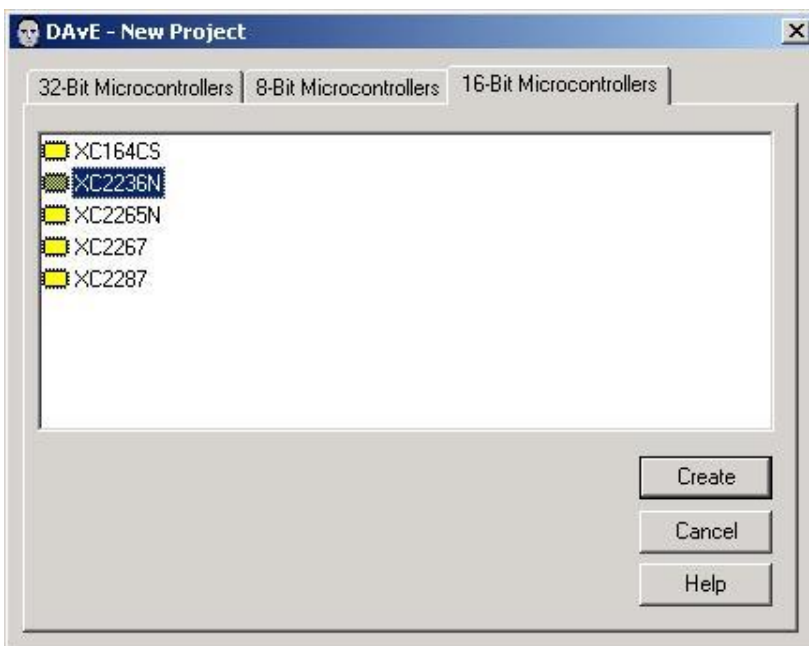


Figure 3 Selecting XC2236N

- Click on “ECU” bubble.

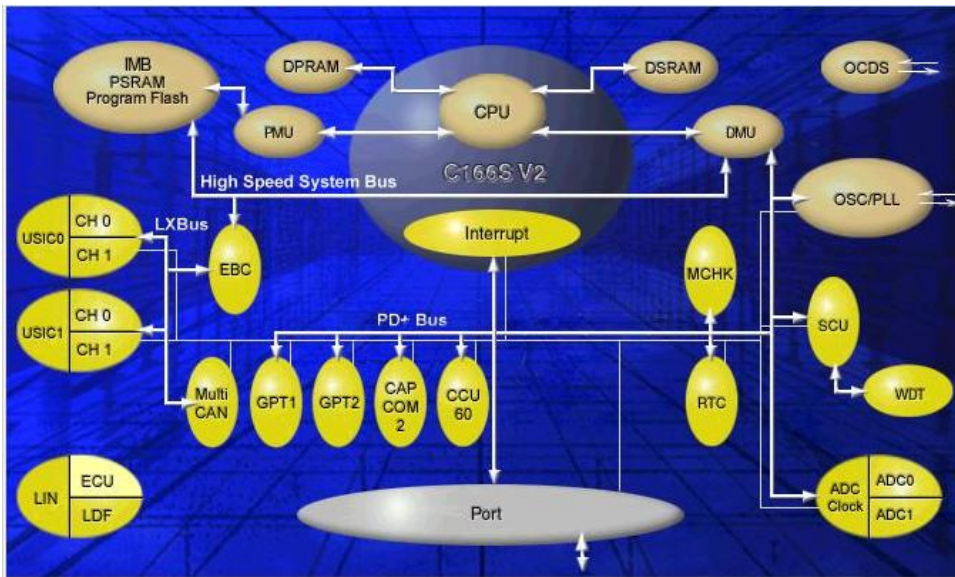


Figure 4 ECU bubble

- Select “ECU page1” tab, and then proceed select the appropriate setting based on the application requirement:
 - Transmit and Receive Pinout
 - Master or Slave Interface Configuration
 - LLD features support
 - Diagnostic functionality support
 - Protocol Information

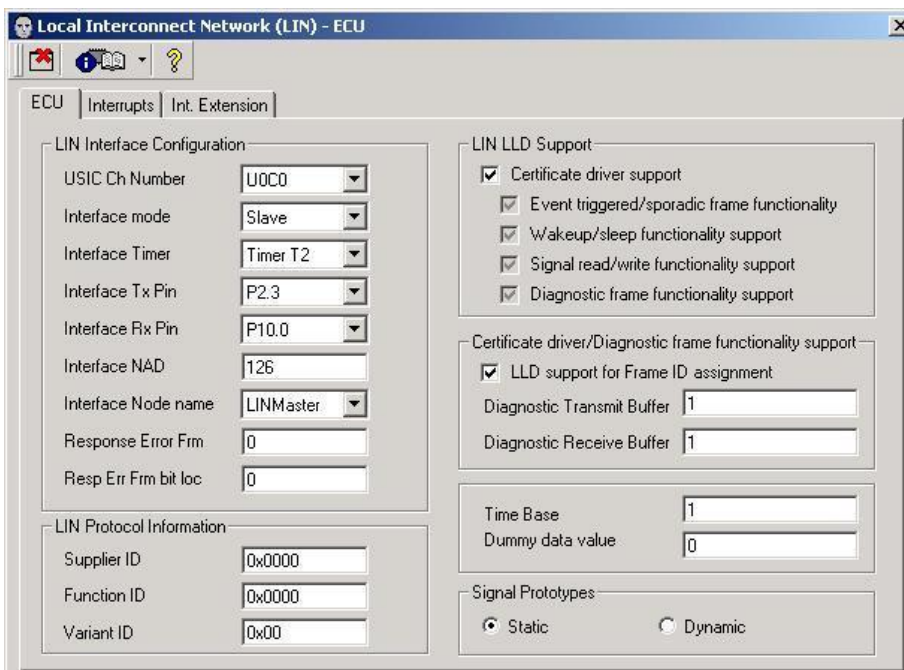


Figure 5 ECU page

- Under “**Interrupts**” tab, select interrupt priority levels for peripherals.

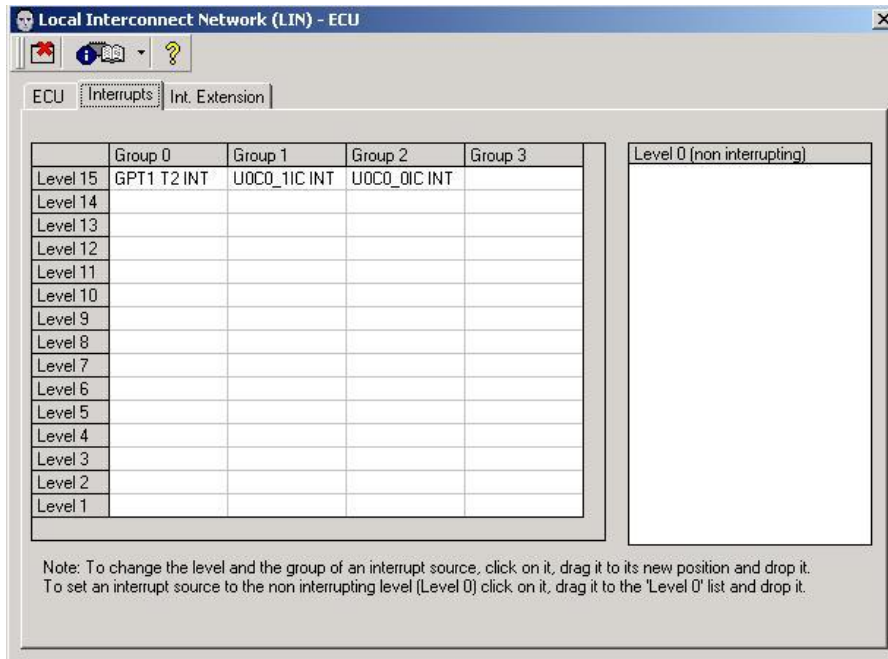


Figure 6 Interrupts Priority setting

- Click on the “**LDF**” bubble.

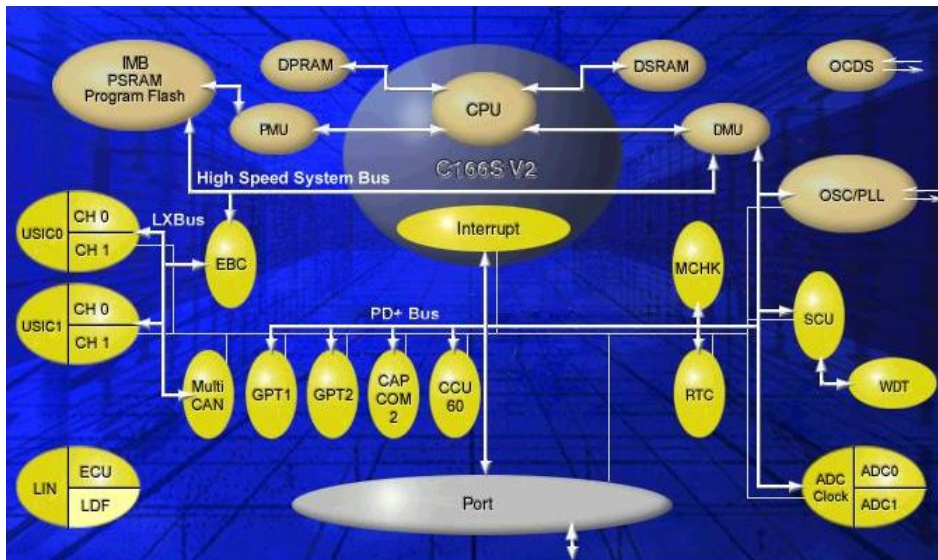
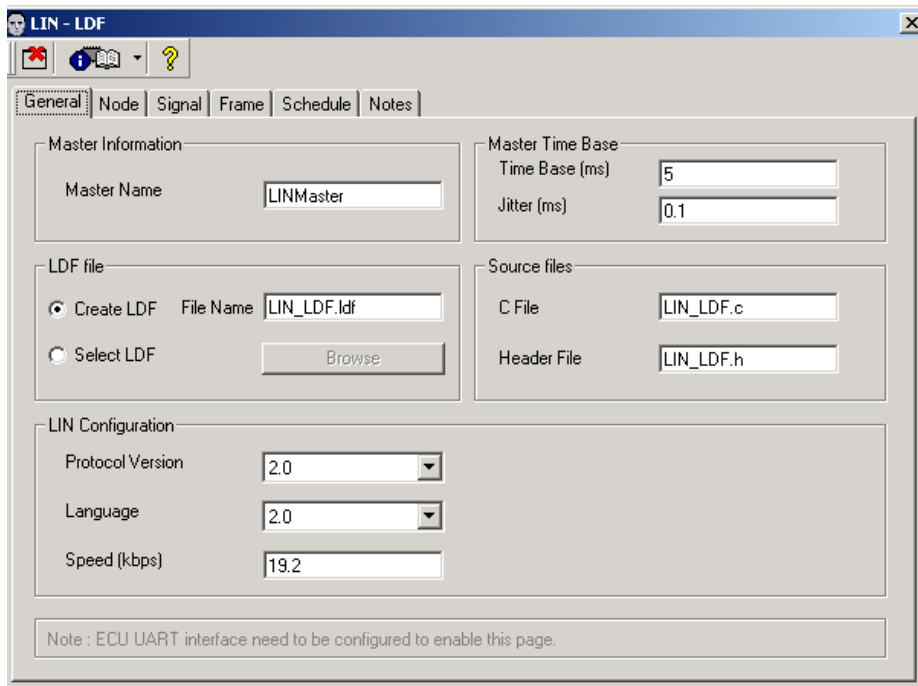


Figure 7 LDF Bubble

- Select “**General**” tab, Select or update Master Information.



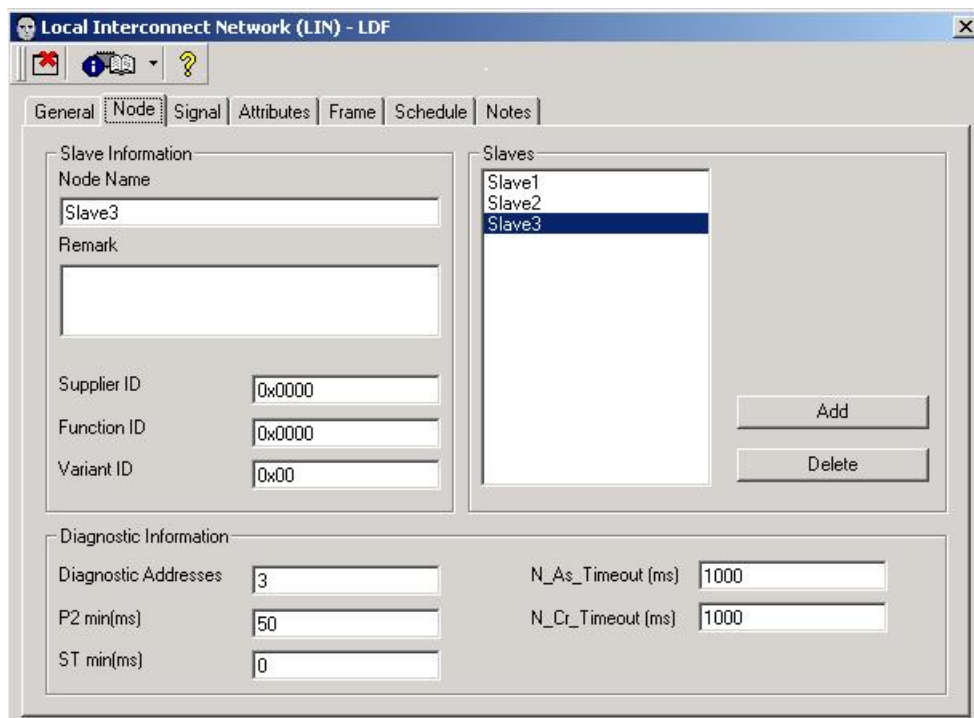
The screenshot shows the 'LIN - LDF' configuration window with the 'General' tab selected. The window contains several sections for configuring the master information:

- Master Information:** Includes a text field for 'Master Name' (set to 'LINMaster').
- Master Time Base:** Includes text fields for 'Time Base (ms)' (set to '5') and 'Jitter (ms)' (set to '0.1').
- LDF file:** Includes radio buttons for 'Create LDF' (selected) and 'Select LDF'. The 'File Name' field is set to 'LIN_LDF.ldf'. A 'Browse' button is available.
- Source files:** Includes text fields for 'C File' (set to 'LIN_LDF.c') and 'Header File' (set to 'LIN_LDF.h').
- LIN Configuration:** Includes dropdown menus for 'Protocol Version' (set to '2.0') and 'Language' (set to '2.0'), and a text field for 'Speed (kbps)' (set to '19.2').

A note at the bottom states: "Note : ECU UART interface need to be configured to enable this page."

Figure 8 Sample Master Information

- Select “**Node**” tab, and enter information for different nodes.



The screenshot shows the 'Local Interconnect Network (LIN) - LDF' configuration window with the 'Node' tab selected. The window contains several sections for configuring node information:

- Slave Information:** Includes text fields for 'Node Name' (set to 'Slave3') and 'Remark'. Below these are text fields for 'Supplier ID' (set to '0x0000'), 'Function ID' (set to '0x0000'), and 'Variant ID' (set to '0x00').
- Slaves:** A list box containing 'Slave1', 'Slave2', and 'Slave3' (selected). Below the list are 'Add' and 'Delete' buttons.
- Diagnostic Information:** Includes text fields for 'Diagnostic Addresses' (set to '3'), 'P2 min(ms)' (set to '50'), 'ST min(ms)' (set to '0'), 'N_As_Timeout (ms)' (set to '1000'), and 'N_Cr_Timeout (ms)' (set to '1000').

Figure 9 Node Tab

- Select “**Signal**” tab, and enter information for different signals.

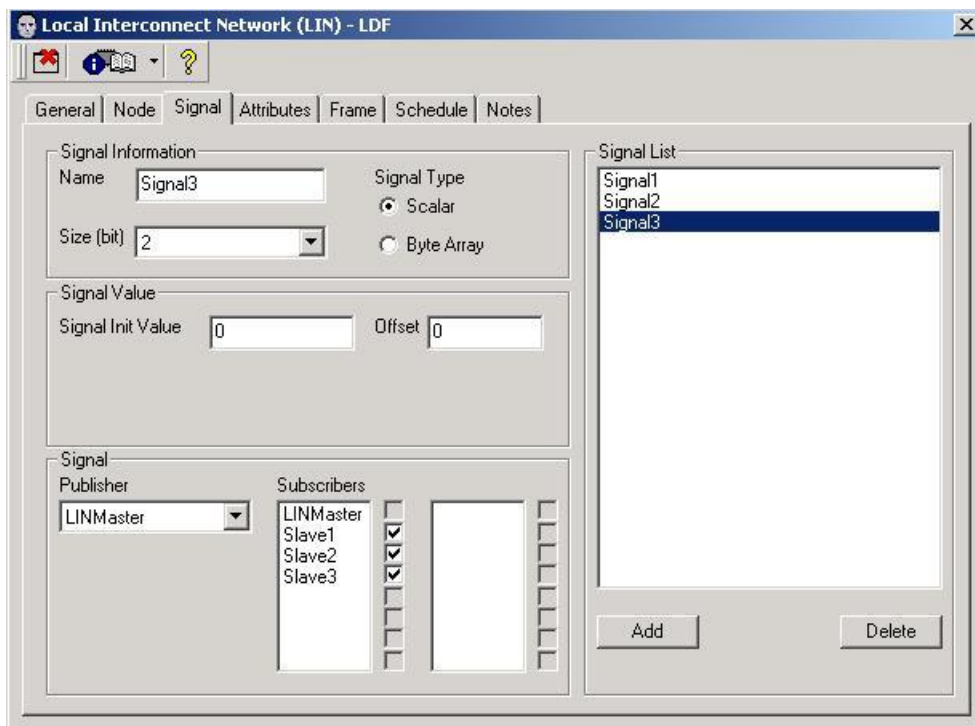


Figure 10 Signal Tab

- Select “**Frame**” tab, and enter information for different frames.

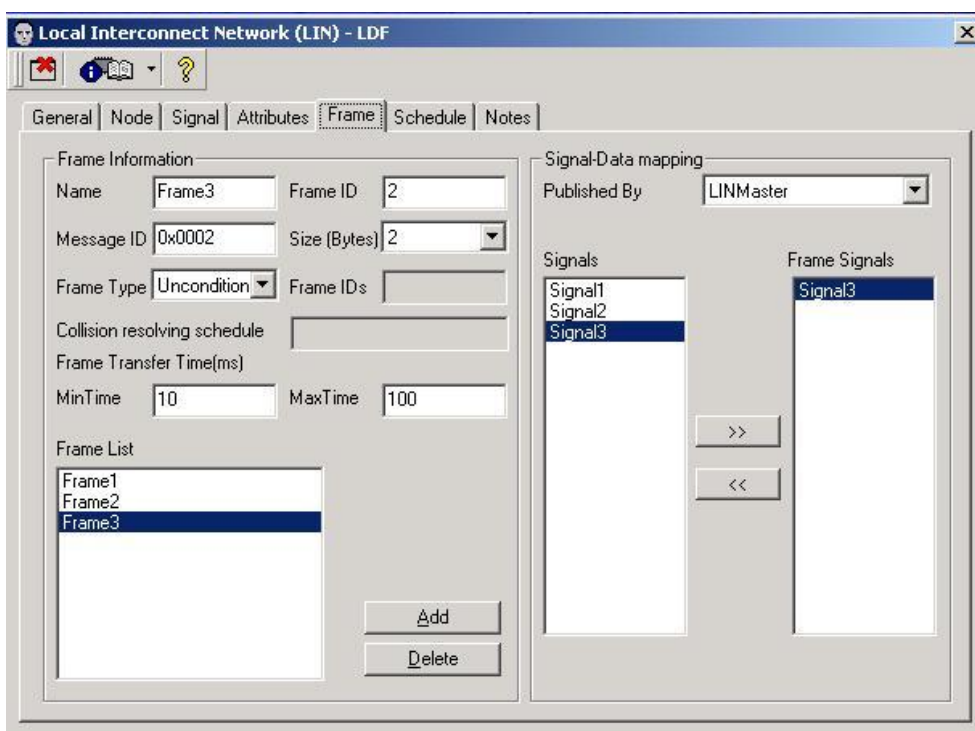


Figure 11 Frame Tab

- Select “**Schedule**” tab, enter information for schedule tables.

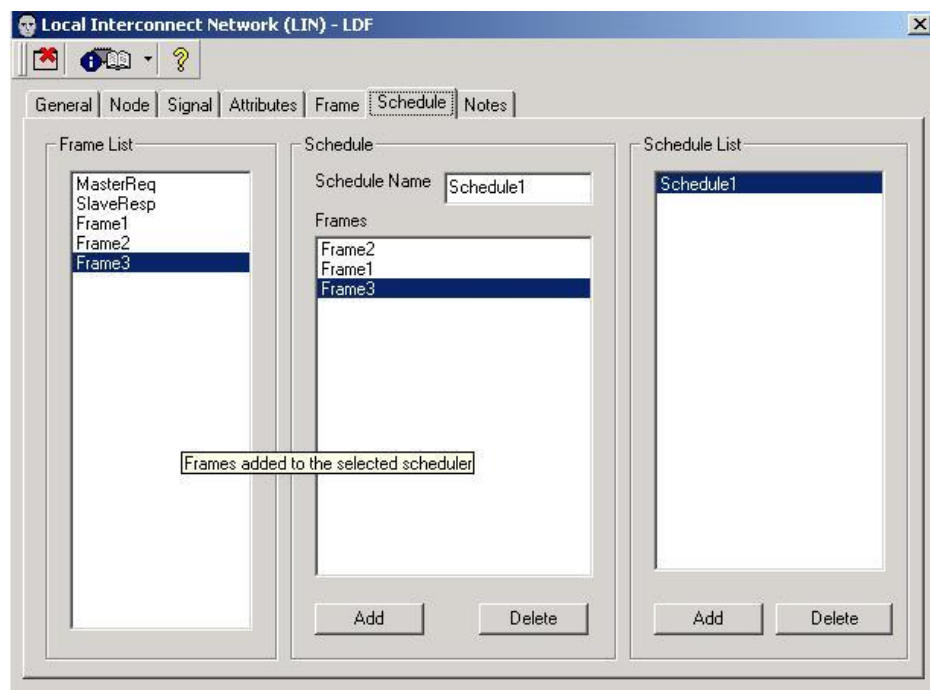


Figure 12 Schedule Tab

- Click on “**Generate Code**” icon in the toolbar to generate corresponding Lin_LDF.c, Lin_LDF.h and Lin_CFG.h

4.9.2 Creating the configuration from LDF File

As an alternative, the configuration supports reading the LDF file, and creating corresponding driver files.

The Sequence is as under:

- Under “**General**” tab, select the “Select LDF” Option.

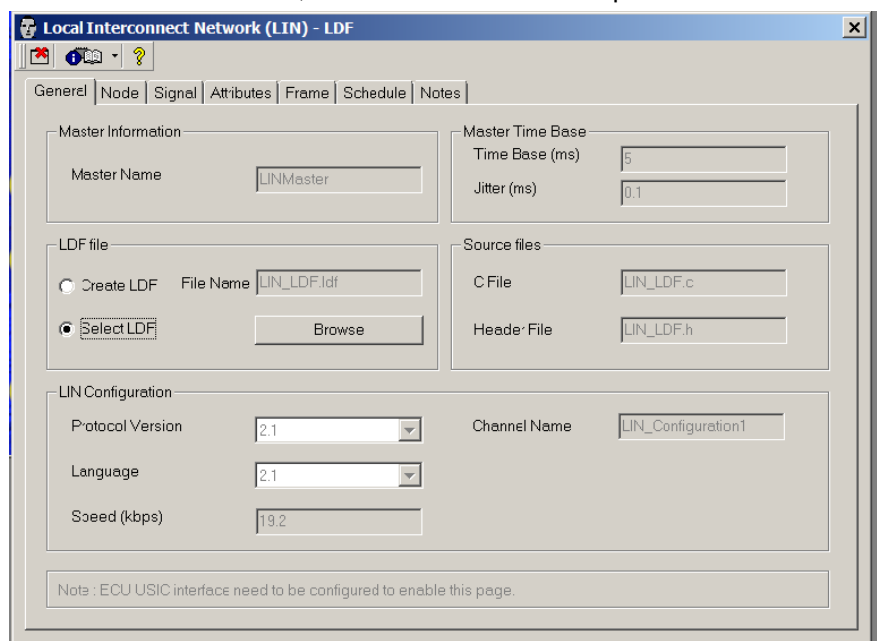


Figure 13 Sample Master Information

- Select Option “Select LDF”, Click on Browse Button, a file dialog appears

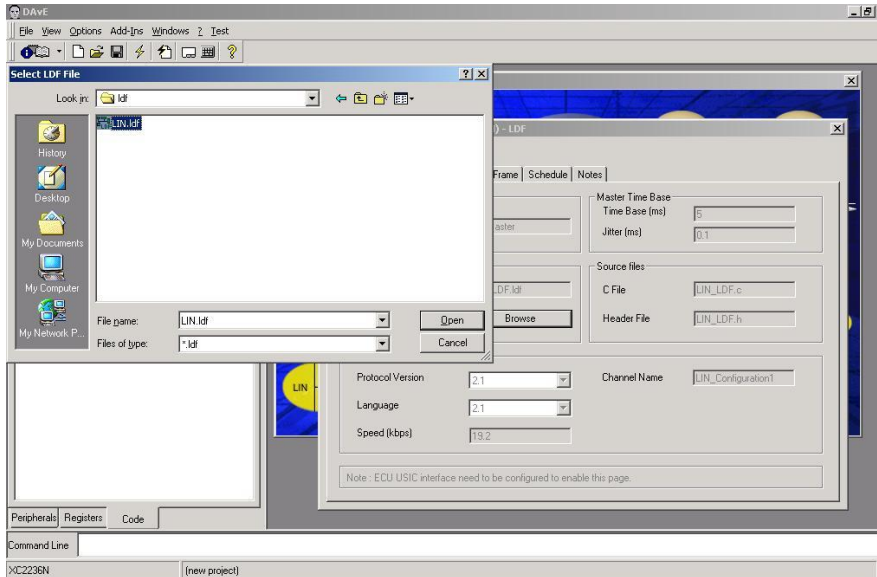


Figure 14 Selecting an LDF file

- Select the desired LDF
- Click on “**Generate Code**” icon in the toolbar to generate corresponding Lin_LDF.c, Lin_LDF.h and Lin_CFG.h

5 API description

5.1 l_sys_init

- Description** : This API is called by application to initialize LIN cluster and this should be the first before using any LIN related APIs.
- Return value** : Boolean
0 → Success
1 → Failure
- Argument** : None
- Example** : `ret_val = l_sys_init();`

5.2 l_ifc_init

- Description** : This API is called by application to become part of LIN cluster and called only once after calling l_sys_init API.
- Return value** : None
- Argument** : Interface number
- Example** : `l_ifc_init(0);`

5.3 l_ifc_connect

- Description** : Application uses this API to connect to LIN cluster and to receive/transmit LIN frames. This API is called after calling l_ifc_init or l_ifc_disconnect APIs. This is contrary to l_ifc_disconnect API.
- Return value** : Boolean
0 → Success
1 → Failure
- Argument** : Interface number
- Example** : `ret_val = l_ifc_connect(0);`

Attention: This API is not available within v2.1 implementation as per the specifications

5.4 l_ifc_disconnect

- Description** : This is used to disconnect the interface from LIN cluster. After successful return of this call interface will not receive/transmit LIN frames. This is contrary to l_ifc_connect API.
- Return value** : Boolean
0 → Success
1 → Failure
- Argument** : Interface number
- Example** : `ret_val = l_ifc_disconnect(0);`

Attention: This API is not available within v2.1 implementation as per the specifications

5.5 l_ifc_read_status

Description	:	Used to know the status of interface.
Return value	:	Status
		Bit 0 → error in response
		Bit 1 → successful transfer of frame data
		Bit 2 → over run (bit 0 or bit 1 are previously set)
		Bit 3 → received sleep command or entered into sleep state because bus is idle
		Bit [8-15] → Frame ID received from master interface recently
Arguments	:	Interface number
Example	:	<code>ret_val = l_ifc_read_status(0);</code>

5.6 l_ifc_goto_sleep

Description	:	Used only by master interface to request all interfaces in LIN cluster to enter into sleep state by sending <i>go to sleep</i> command
Return value	:	None
Arguments	:	Interface number
Example	:	<code>l_ifc_goto_sleep (0);</code>

5.7 l_ifc_wake_up

Description	:	Used by any interface to send wakeup request in sleeping LIN cluster. After receiving this request master starts publishing frame headers.
Return value	:	None
Arguments	:	Interface number
Example	:	<code>l_ifc_wake_up (0);</code>

5.8 l_sys_irq_disable

Description	:	Returns the current status of interrupts and disable all the LIN interface related interrupts except global interrupt
Return value	:	l_irqmask
		Bit 0 → USIC interrupt enable bit status
		Bit 1 → Primary timer interrupt enable bit status
		Bit 2 → Secondary timer interrupt enable bit status, this is applicable only in case of slave interface and sleep/wakeup feature is enabled.
Arguments	:	None
Example	:	<code>irq_status = l_sys_irq_disable ();</code>

5.9 **l_sys_irq_restore**

Description : Set/reset the interrupts enable bit status as defined in the argument.

Return value : None

Arguments : `l_irqmask`

- Bit 0 → USIC interrupt enable bit status
- Bit 1 → Primary timer interrupt enable bit status
- Bit 2 → Secondary timer interrupt enable bit status, this is applicable only in case of slave interface and sleep/wakeup feature is enabled.

Example : `l_sys_irq_restore (irq_status);`

5.10 **l_flg_tst**

Description : Returns Boolean value indicating the current status of frame data

Return value : Boolean

- Bit 0 → Updated data is not available
- Bit 1 → Updated data is available

Arguments : Frame ID, shall not include the parity bits

Example : `l_flg_tst(4);`

5.11 **l_flg_clr**

Description : Reset the particular frame updated data flag to indicate the frame data is invalid

Return value : None

Arguments : Frame ID, shall not include the parity bits

Example : `l_flg_clr (4);`

5.12 Signal read APIs

These APIs are used to read particular signal information by providing signal name as argument. The value of the signal will be returned. Before calling this function it is advised to check the status of the information by calling `l_flg_tst` API.

The signal names are specified in `LIN_sig_base`, this API definition available in `LIN_LDF.c` file. To use these signal names application has to include the `LIN_LDF.h` file.

5.12.1 Scalar Signal Read API's

Description : The following APIs are used to read the scalar signal information, the value ranges from 0 to 16 bits

`l_bool_rd`
`l_u8_rd`
`l_u16_rd`

Return value : Value of the signal

Arguments : Signal name

Example : `signal_info = l_u8_rd (signal1);`

5.12.2 Bytes Array Signal Read API's

Description : The following API is used to read byte array signal data, the value ranges from 1 to 8 bytes

`l_bytes_rd`

Return value : None

Arguments : Signal name
Start of the "byte index" into the frame data which includes signal data.
Number of data bytes to read
Pointer to application data buffer

Example : `l_bytes_rd (signal1, 0, 8, *ptr_appl_buf);`

Note: After reading the individual signal data it is up to the application to clear the updated data flag by calling `l_flg_clr` API. These sets of APIs are available only in case `_SIG_RDWR_FUNC_SPRT` or `_FEATURES` set to 1 in `LIN_CFG.h` file.

5.13 Signal write APIs

These APIs are used to write particular signal information by providing signal name as argument along with value.

The signal names are specified in LIN_sig_base, this API definition available in LIN_LDF.c file. To use these signal names application has to include the LIN_LDF.h file.

5.13.1 Scalar Signal write API's

Description	:	The following APIs are used to write the scalar signal information, the value ranges from 0 to 16 bits l_bool_wr l_u8_wr l_u16_wr
Return value	:	Signal name Value of the signal
Arguments	:	None
Example	:	<code>l_u8_wr (signal1, signal_val);</code>

5.13.2 Bytes Array Signal Write API's

Description	:	The following API is used to write byte array signal data, the value ranges from 1 to 8 bytes l_bytes_wr
Return value	:	None
Arguments	:	Signal name Start of the "byte index" into the frame data which includes the signal data. Number of data bytes to read Pointer to application data buffer
Example	:	<code>l_bytes_wr (signal1, 0, 8, *ptr_appl_buf);</code>

Note: These sets of APIs are available only in case of _SIG_RDWR_FUNC_SPRT or _FEATURES set to 1 in LIN_CFG.h file.

5.14 Id_get_raw

Description	:	Copies the oldest received diagnostic frame data from LLD buffer to the application data buffer.
Return value	:	None
Arguments	:	Interface name

Pointer to application data buffer

Example : `ld_get_raw (0, &app_data_buff);`

5.15 **ld_put_raw**

Description : Copy the diagnostic frame data provided by application to the LLD diagnostic transmit buffer.

Return value : None

Arguments : Interface name

Pointer to application data buffer

Example : `ld_put_raw(0, &app_data_buff);`

5.16 **ld_raw_rx_status**

Description : Return the status of diagnostic receive buffer.

Return value : Status

4 → Error occurred during data transfer

8 → Diagnostic data available

Arguments : Interface name

Example : `ld_raw_rx_status(0);`

5.17 **ld_raw_tx_status**

Description : Return the status of diagnostic transmit buffer.

Return value : Status

1 → The transmit queue is full

2 → The transmit queue is empty

8 → Error occurred during data transfer

Arguments : Interface name

Example : `ld_raw_tx_status(0);`

5.18 **Lin_TpCallBack**

Description : Called by The Driver to let the application prepare the response for UDS Services.

Return value : None

0 → The positive / negative response data has been written to diagnostic TX buffer

1 → No response data has been written to diagnostic TX buffer

Arguments : IRxBuff → Pointer to diagnostic RX buffer where command data has been stored

IRxLen → Actual Length of the data received within the Rx buffer

Please note that for a diagnostic frame length is always fixed (8 bytes), which does not necessarily mean that the frame has 8 bytes of valid data. The transport protocol frames carry other information like NAD PCI, SID and data length information as well, which are not reported from the variable.

API description

Ex: if the data bytes are 7, then actual data bytes transmitted would be 16, in which only 7 bytes are valid and the value of variable lRxLen should contain this value. Other bytes are standard command bytes like the NAD, PCI, the unused data bytes and the payload length (SID is anyhow the part of the data bytes).

lTxBuff → Pointer to diagnostic TX buffer where response data should be populated

lTxLen → Length of the Response

Example :

```
uint8 Lin_TpCallBack ( uint8 *lRxBuff, uint16 lRxLen,
                      uint8 *lTxBuff, uint16 *lTxLen)
{
    uint8 retVal = (uint8)0;
    switch(lRxBuff[0])
    {
        case 0xb2:
        {
            /* Populate the response here */
        }
        break; /*0xb2 */
        default:
        break ;
    }/* switch */
    return(retVal);
}
```

5.19 ld_send_message

Description : The call packs the information specified by data and length into one or multiple diagnostic frames. The call is asynchronous. If there is a message in progress, the call will return with no action.

Return value : none

Arguments : interface handles number, number of bytes to be transmitted, Node address, and the buffer to transmit.

Example : `ld_send_message (0, 256, 127, &data1[0][0]);`

5.20 ld_receive_message

Description : The call prepares the LIN diagnostic module to receive one message and store it in the buffer pointed to by data. At the call, length shall specify the maximum length allowed. When the reception has completed, length is changed to the actual length and NAD to the NAD in the message.

Return value : none

Arguments : interface handle number, number of bytes to be read, Node address, and the buffer where data will be copied.

Example : `ld_receive_message (0, 256, 127, &data1[0][0]);`

5.21 ld_tx_status

Description : The call returns the status of the last made call to ld_send_message.

Return value : The following values can be returned.

- LD_IN_PROGRESS → The transmission has not yet completed.
- LD_COMPLETED → The transmission has completed successfully (and you can issue a new ld_send_message call). This value is also returned after initialization of the transport layer.
- LD_FAILED → The transmission ended in an error. The data was only partially sent. The transport layer shall be reinitialized before processing further messages. To find out why a transmission has failed, check the status management function l_read_status.
- LD_N_AS_TIMEOUT → The transmission failed because of N_As timeout.

Arguments : Interface handle number

Example : `status = ld_tx_status (0);`

5.22 ld_rx_status

Description : The call returns the status of the last made call to ld_receive_message.

Return value : The following values can be returned.

- LD_IN_PROGRESS → The reception has not yet completed.
- LD_COMPLETED → The reception has completed successfully (and you can issue a new ld_send_message call). This value is also returned after initialization of the transport layer.
- LD_FAIL → The reception ended in an error. The data was only partially sent. The transport layer shall be reinitialized before processing further messages. To find out why a transmission has failed, check the status management function l_read_status.
- LD_N_CR_TIMEOUT → The transmission failed because of N_Cr timeout.
- LD_WRONG_SN → The reception has failed because of an unexpected sequence number.

Arguments : Interface handle number

Example : `status = ld_rx_status (0);`

Note: The Driver can be configured to have internal / external buffer management for diagnostic uses via the configuration parameter `_TP_BUFF_MANGEMENT`.

Note: These following diagnostic API's are available only in case of `_DGNSTC_FRM_SPRT` is selected or `_FEATURES` is set to 1 in `LIN_CFG.h` file, AND buffer management is not internal.

1. ld_put_raw
2. ld_get_raw
3. ld_send_message
4. ld_receive_message
5. ld_raw_tx_status
6. ld_raw_rx_status

7. Id_tx_status
8. Id_rx_status

5.23 FrmId_FrmCntInit (uint8 u8FrameId)

Description : The call initializes the frame counter for a given frame.

Return value : The following values can be returned.

- 0 → Invalid argument.
- 1 → The Frame counters have been initialized.

Arguments : Frame index for which the counter should be initialized

Example : `status = FrmId_FrmCntInit (FrmId) ;`

5.24 FrmId_FrmCntRead (uint8 u8FrameId)

Description : The call reads the counter for a given frame.

Return value : 16 bit counter value

Arguments : Frame index for which the counter value should be returned

Example : `frame_rcvd_val = FrmId_FrmCntRead (FrmId);`

6 Example code

The following sample pieces of code are used to write the basic application program.

6.1 Creation of slave interface application

Configure the interface type in LIN_CFG.h file at the time of compilation

```
#define LIN_V_2_1 /* Select the Protocol version */  
#define _IFC_SLAVE /* Slave is selected */
```

Configure primary timer and secondary timer in LIN_CFG.h file

```
#define _IFC_SCND_TMR_NUM GPT_T6 /* secondary timer T6 is selected */
```

Configure Lin_FrmInfo and Lin_FrmOffset related APIs in LIN_LDF.c and LIN_LDF.h Files

Code Listing 1 LIN_Test.c file example code

```
001: #include "LIN_API.h"  
002: void main()  
003: {  
004:     l_u8 data1[2];  
005:     l_sys_init(); /*LIN cluster initialization*/  
006:     l_ifc_init(0); /*LIN interface initialization*/  
007:     #ifndef LIN_V_2_1  
008:         l_ifc_connect(0);  
009:     #endif  
010:     l_sys_irq_restore(0x07);  
011:     while(1) {}  
012: }
```

7 Limitations

This chapter highlights major limitations of the LIN driver.

7.1 Limitation 1

For the API `I_ifc_read_status ()` for the Master node, the updating of the “Overrun”, “Successful Transfer” & “Error in response” bit of the Status information occurred before the next frame transmission.

7.2 Limitation 2

The interface number should always be 0.

www.infineon.com