Application Note, V1.0, Feb. 2007

AP16107

LIN Application for XC164CM Using DAvE LIN Configuration tool

Microcontrollers

Never stop thinking

Edition 2007-02-15

Published by
Infineon Technologies AG
81726 München, Germany

AP16107

Revision History:	2007-02	V1.0
Previous Version:	none	

Page	Subjects (major changes since last revision)

Table of Contents **Page**

1 Introduction

In automotive application development, there is a need to have tools which can help you configure the chip to work the way you need it. Such tools can greatly reduce the overhead of application development. This paper deals with an idea to provide introduction and hands-on-experience of Infineon Technologies LIN configuration tool that supports the LIN – a cost effective time triggered automotive protocol. The LIN configuration tool can be used by the developers to make a prototype LIN applications.

This application note gives an overview of the LIN protocol, and describes the operation of the LIN low level driver for Infineon Technologies 16bit microcontrollers. The configuration or implementation of the LIN prototype application is discussed step by step. A LIN application project is explained to build a simple LIN network with XC164CM devices (master node and one slave node).

2 Local Interconnect Network (LIN)

Automotive Communication Requirements

The automotive system is nowadays a complex distributed system with various demands of networking capabilities. One automotive application consists of one or several Electronic Control Units (ECUs). In an automotive system consisting of several automotive applications more than 70 ECUs might need to distribute more than 2500 signals. This makes the automotive system complicated in terms of networking. Adding so many ECUs to the car several different types of communication networks are required to reach cost effective solutions.

A modern car usually contains one or two Controller Area Network (CAN)-networks. CAN is too expensive to use for connecting simple devices such as simple switches and non safety critical functions like indoor lights etc. A low cost communication solution called Local Interconnection Network (LIN) has been developed to solve these problems. The purpose of LIN is not to replace the CAN-bus already running in the vehicles, but to be a low cost complement for connecting simple functions.

LIN

The LIN is a low cost serial asynchronous communication system intended to be used for distributed electronic systems in automotive applications. The communication is based on the SCI (UART) byte-word interface. UART interfaces available as low cost silicon module on almost all micro-controller. The LIN does not support very high bandwidth and long cables as more expensive bus technologies.

The LIN-bus always consists of one master and *n* slaves and can also have an interface to other communication busses. The medium access in a LIN network is controlled by a master node so that no arbitration or collision management in the slave nodes is required, thus giving a guarantee of the worst-case latency times for signal transmission.

A particular feature of LIN is the synchronization mechanism that allows the clock recovery by slave nodes without quartz or ceramics resonator. The maximum transmission speed is 20 kbps.

A node in LIN networks does not make use of any information about the system configuration, except for the denomination of the master node. Nodes can be added to the LIN network without requiring hardware or software changes in other slave nodes. The clock synchronization, the simplicity of UART communication, and the single-wire medium are the major factors for the cost efficiency of LIN.

Applications

Typical applications for the LIN bus are assembly units such as doors, steering wheel, seats, climate regulation, lighting, rain sensor, or alternator. In these units the cost sensitive nature of LIN enables the introduction of mechatronic elements such as smart sensors, actuators, or illumination. They can be easily connected to the car network and become accessible to all types of diagnostics and services.

LIN cluster

The LIN (Local Interconnect Network) is a broadcast bus that operates in speed upto 20 kbps and supports the control of mechatronic nodes in distributed automotive applications. The LIN bus can have an interface to other communication busses, for example CAN.

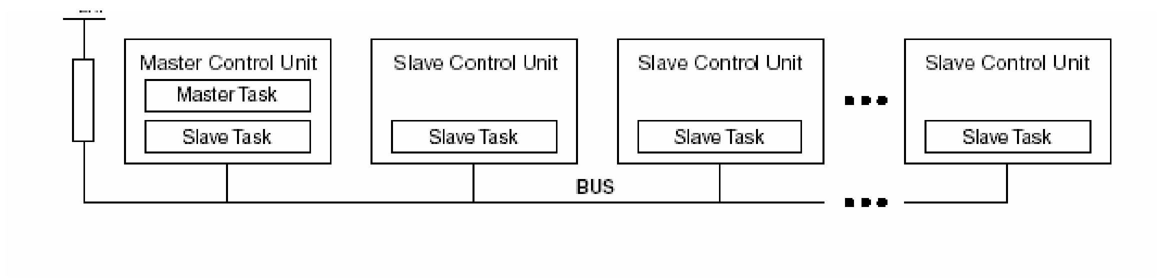


Figure 2.1 A LIN Bus with master and slave nodes

.

Nodes

A node in a LIN cluster interfaces to the physical bus wire using a frame transceiver. The frames are not accessed directly by the application; a signal based interaction layer is added in between.

Master and slave

A LIN cluster consists of one master task and several slave tasks. A master node contains the master task as well as a slave task. All other nodes contain a slave task only.

The LIN is a time triggered master slave network, where the master has a schedule that defines when each frame is to be sent. All frames include a header, which is sent by the master. The header includes information on which node that should send the data frame. If a slave node recognizes its identification it either sends or receives the data of the frame specified in the header.

One node can transmit a frame while all of the other nodes can receive the frame, however every frame is coded to indicate which node is the sender and receiver(s). This enables the receiving nodes to disregard the frames which they do not require, thereby reducing the load on them.

Frames

Each node in the system has a set of frames that should be sent to other nodes in the system. A frame is described by name, size, sender and a deadline. Whether a frame accepted is decided solely by the controller (receiver system). This decision is made using acceptance or message filter. Therefore it is possible for a frame to be accepted by one, several or all controllers for further processing.

Only one frame is transmitted at any given point in time. Consequently, no mechanism is needed to resolve bus collisions, since it is impossible for collisions to occur in a LIN network. All frames to be transmitted are sent within one cycle. The chronological sequence of frames is fixed in a schedule. Schedules may be switched as needed.

Each frame is composed by a HEADER, which is sent by the master task and a RESPONSE that is sent by one of the slave tasks. The header consists of a break and sync pattern followed by an identifier. The identifier uniquely defines the purpose of the frame. The slave task appointed for providing the response associated with the identifier transmits it.

The HEADER consists of a BREAK FIELD, SYNCHRONISATION FIELD and IDENTIFIER FIELD. The RESPONSE consists of three to nine BYTE FIELDs: two, four, or eight DATA FIELDs, and one CHECKSUM FIELD. The BYTE FIELDs are separated by inter-byte spaces, HEADER and RESPONSE are separated by one in-frame-response space.

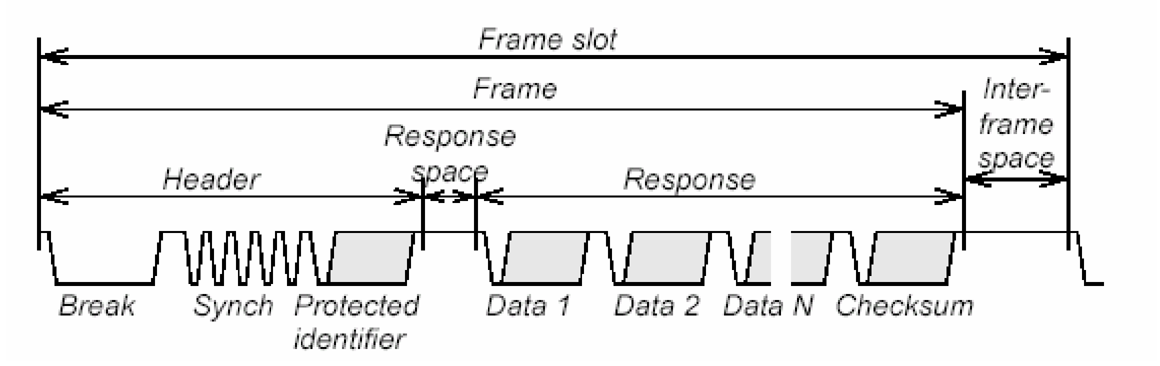


Figure 2.2 LIN FRAME

The number and size of frames are limited to: 32 numbers of 2 byte frames, 16 numbers of 4 byte frames, 16 numbers of 8 byte frames. The slave tasks interested in the data associated with the identifier receives the response, verifies the checksum and uses the data transported. This results in system flexibility, message routing and multicast.

Signals

A signal is transported in the data field of a frame. Several signals can be packed into one frame as long as they do not overlap each other. Each signal has exactly one producer, i.e. it is always written by the same node in the cluster. Zero, one or multiple nodes may subscribe to the signal. Each signal is described by name, size, end-to-end deadline, sender node and receiver node(s). Signals may be represented by 1 to 16 bits. A signal may be visualised as a 'virtual wire', which may either be an input to or an output from a node. As many as hundred signals may be transmitted back and forth over the communication networks.

A signal is either a scalar value or a byte array. A scalar signal is between 1 and 16 bits long. A byte array is an array of between one and eight bytes. A signal is transmitted with the LSB first and the MSB last. The only additional rule for scalar signal packing within a frame is that maximum one byte boundary may be crossed by a scalar signal. Each byte in a byte array shall map to a single frame byte starting with the lowest numbered data byte.

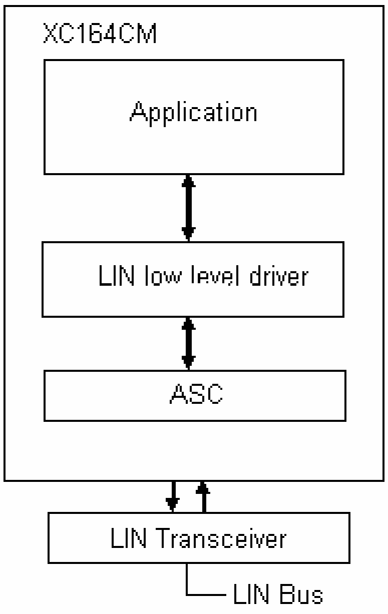
Schedule table

A key property of the LIN protocol is the use of schedule tables. Schedule table makes it possible to assure that the bus will never be overloaded. Deterministic behavior is made possible by the fact that all transfers in a LIN cluster are initiated by the master task.

The master task transmits frame headers based on a schedule table. The schedule table specifies the identifiers for each header and it is the responsibility of the master to assure that all frames are given enough time to be transferred. The master application may use different schedule tables and select among them.

3 LIN Low level driver

The LIN Low level driver is developed for the XC164CM, but the principal for the XC166 family is identical. The LIN bus requires no dedicated on-chip microcontroller communication module. LIN utilizes the standard serial communication interface. That is one major point for the well-balanced cost/performance ratio of this Class A subbus. Data exchange is based on a common hardware peripheral controlled by a dedicated LIN software driver. The LIN LLD handles the basic communication layers and takes care of message transfers, message filtering, and error detection.



. Figure 3.1 LIN Driver and application

The Infineon LIN LLD entirely encapsulates the hardware modules and exclusively handles the on-chip peripherals of the Infineon XC16x microcontrollers, which support LIN. A LIN network based on Infineon Technologies XC16x microcontroller family can be easily realized by using the driver.

LIN Low level driver compliance to the LIN v2.0 protocol specification and developed according to Infineon Technologies HAL (Hardware Abstraction Layer) standard. LLD is certified by C&S (communication & system) group, Germany

Services

The LIN LLD provides several APIs for LIN bus handling. The services of the LIN LLD are Message transmission, Message reception, Message filtering, Connect/disconnect the LIN node to the LIN bus, Sending "*go to sleep mode*" command, Sending "*wake up*" command, Bus timeout detection, Frame monitoring, ID field calculation, Data length extraction, Checksum calculation and LIN message scheduler

Requirements

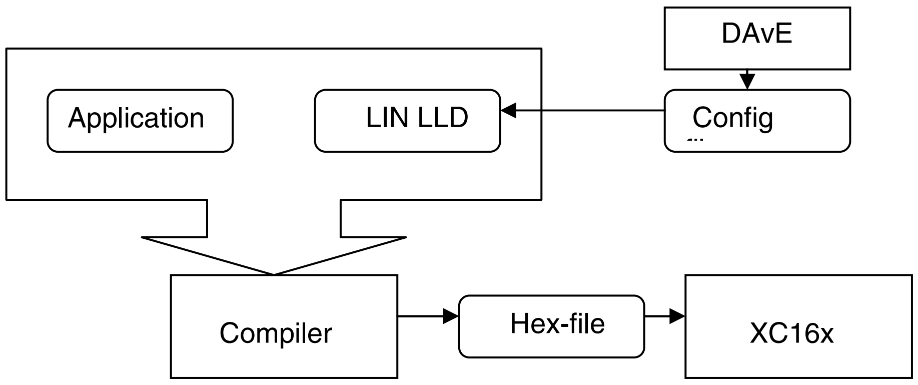
LIN utilizes the serial interface for message frame generation and additionally occupies one timer channel for LIN bus timing monitoring e.g. for detection of not responding slaves or no bus activity. Both peripherals require an interrupt handler, with main processes of the LIN software driver being executed from the interrupt handler of the serial communication channel peripheral.

The software driver occupies less than 3 KBytes of code memory and less than 100 bytes of data memory. These numbers include all the basic LIN software driver functions. Application specific user LIN message buffers require additional memory space.

The user has to configure the system as such, avoiding several slave tasks concurrently responding to the same ID. Moreover, the user is responsible for arranging an application specific network management.

LIN LLD

The application has two software parts, the user application and software driver. The software driver consists of LIN LLD source files and LIN LLD configuration files.



.

Figure 3.2 LIN application development

LIN LLD source files

COMPILER.h
This file provides the data structures to make the LLD compiler independent. It automatically detects the compiler being used and changes the compiler dependent data structures to suit to present compiler. Provides the wrappers to basic data types and ISR handlers.

LIN_IDL.h
Provides the address mapping for the registers used by LIN LLD. To port the LIN LLD to different ECU this file must be updated.

LIN_IIL.h
Provides the bit mapping and mask for the register control bits. This file must be updated to port LIN LLD to different ECU.

SYS_API.h
Provides the definitions for system HAL (SYS HAL). This information shall be used by LIN LLD. To port LIN LLD to different ECU this file must be updated.

LIN_IIL.c
This file contains the implementation of APIs according to LIN protocol specification v2.0.

SYS_IIL.c

This file contain the implementation of system HAL (SYS HAL) related APIs, these APIs can be used by all LLDs of XC164CM.

LDF configuration files

LIN_API.h

Provides the data structures like enums, structures and function prototypes used for LIN LLD.

LIN_LDF.c

This file contains the initialised data for signals, frames and schedule tables. The configured master also act as slave (transmit/receive a frame) simultaneously depend on the configuration for 'LIN_publsh_frm_info' and 'LIN_sbscrb_frm_info' in LIN_LDF.c file.

LIN_LDF.h

Provides data structures for signals, frames and schedules that are used by the LIN LLD.

LIN_CFG.h

Provides configurations for the required functionality of ECU like interface, baud rate, Master or Slave mode, timers and interrupt priorities.

LIN Application file

LIN_test.c

LIN_test.c is a test application source file.

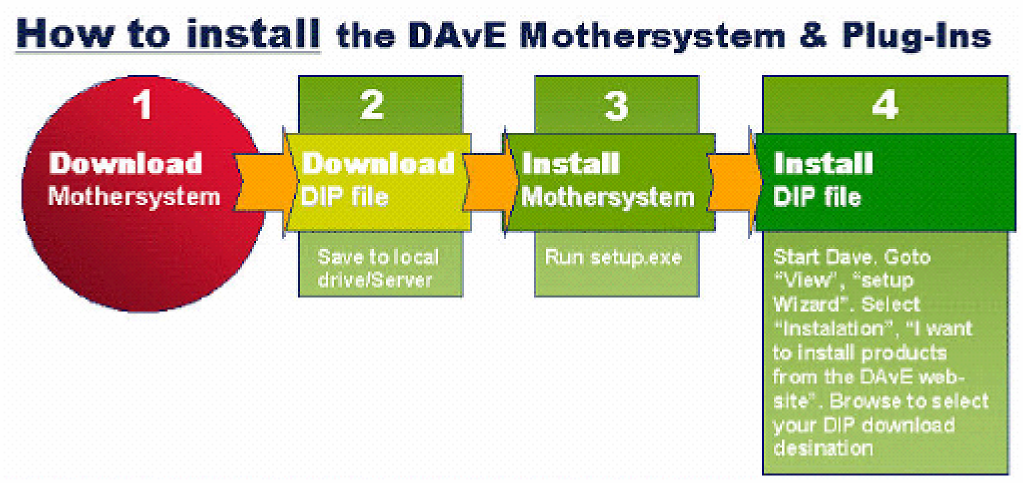
Note : You can contact Infineon Technologies for LIN LLD Source code.

4 LIN Configuration Tool

The LIN Configuration tool is an integral part of DAvE. Digital Application virtual Engineer is configuration and code generation tool for Infineon Technologies Microcontrollers. It provides configuration, initialization and driver code to ease programming of microcontrollers.

Install DAvE XC164CM

- Download and install DAvE Mothersystem from http://www.infineon.com/dave.
- Download and install DAvE XC164CM DIP file.


How to install the DAvE Mothersystem & Plug-Ins
1
Download Mothersystem
2
Download DIP file
Save to local drive/Server
3
Install Mothersystem
Run setup.exe
4
Install DIP file
Start Dave. Goto "View", "setup Wizard". Select "Instalation", "I want to install products from the DAvE web-site". Browse to select your DIP download desination

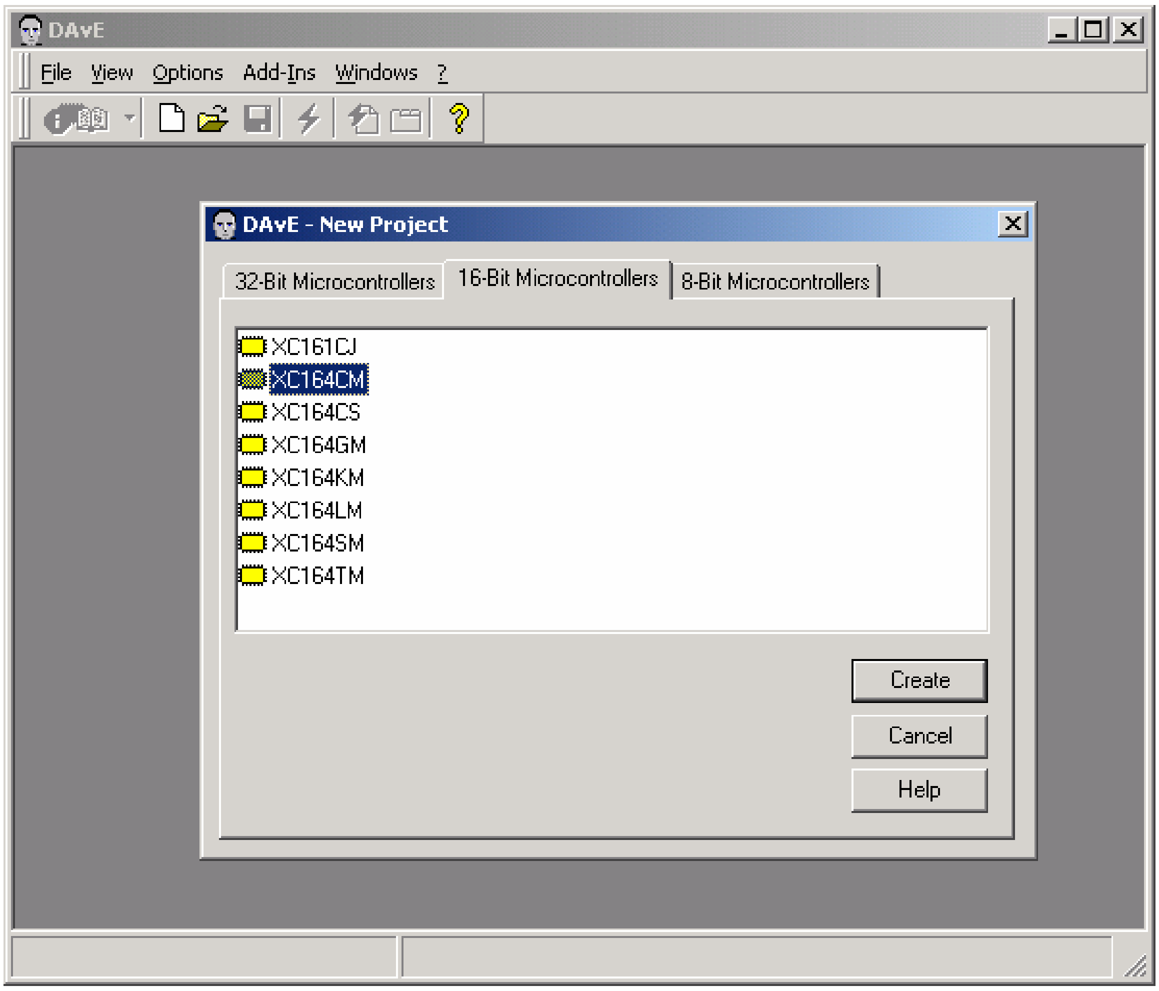
.

Figure 4.1 How to install DAvE

4.1 DAvE XC164CM

Now we need to configure XC164CM. Open DAvE mothersystem to create a new project on XC164CM.

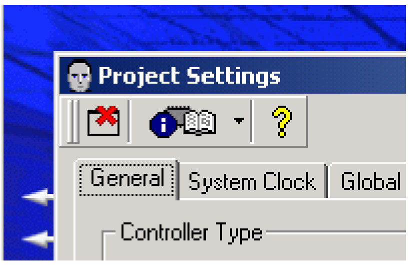
- File -> New -> 16-Bit Microcontrollers -> XC164CM -> Create



Project settings

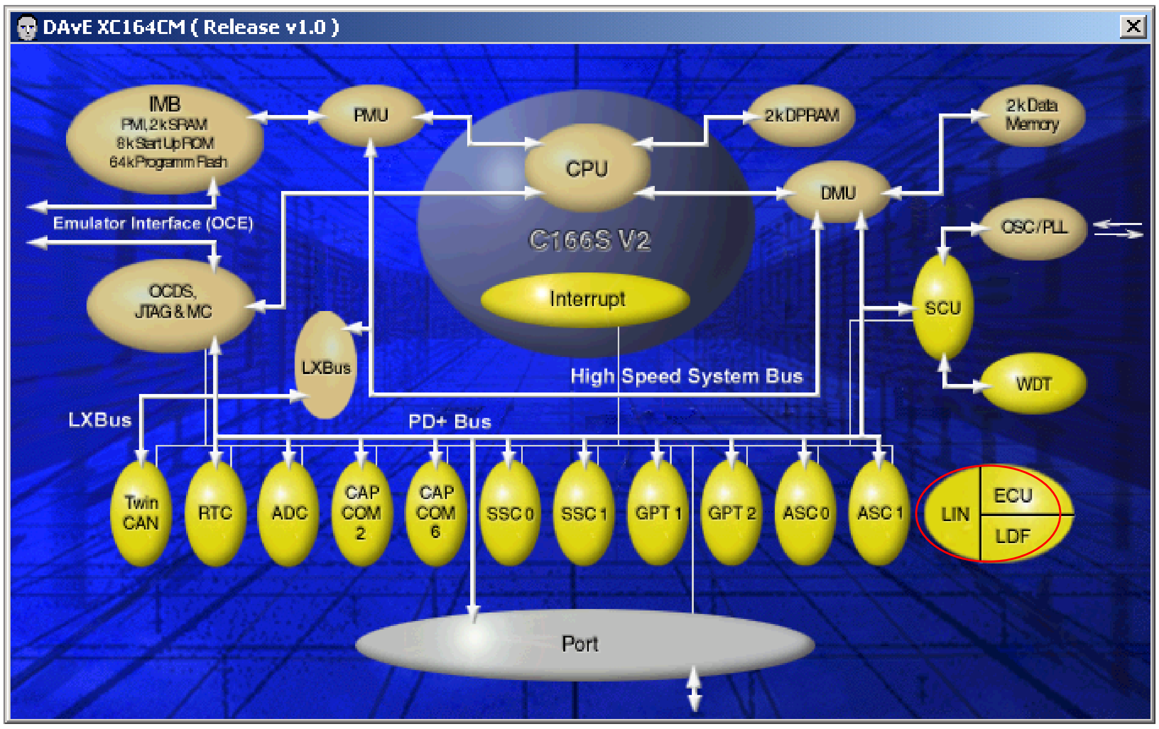
Default project settings have the necessary configurations; so no change in project settings.

Close the project settings page by clicking the close button.



LIN Configurations

The LIN module in DAvE has 2 sub-modules ECU and LDF



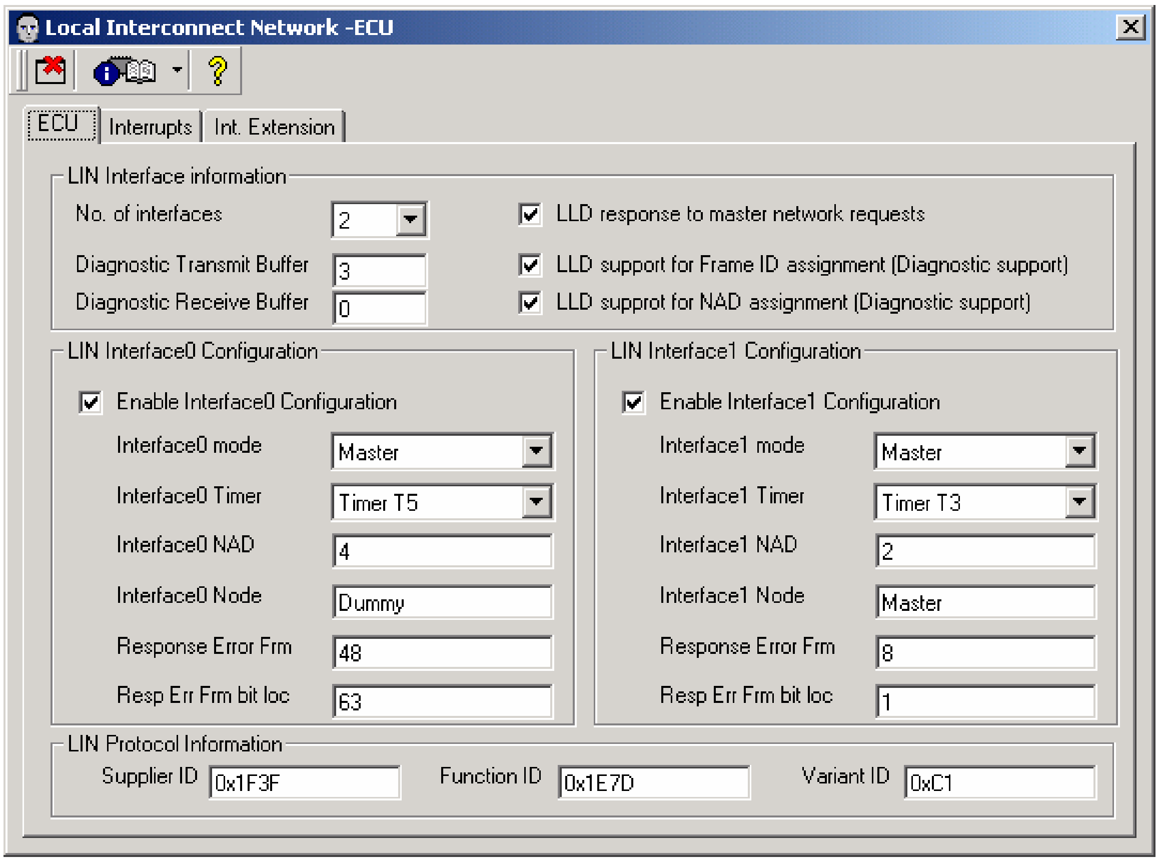
5 LIN Application

Lin application – One master and one slave LIN network using two XC164CMs.

MASTER (XC164CM)	Interface ASC1	Timer T3
SLAVE (XC164CM)	Interface ASC1	Timer T3

5.1 Master Configurations

ECU - configure as done below.



LIN Interface information

No. of interfaces	2
Diagnostic Transmit Buffer	3
Diagnostic Receive Buffer	0

☑ LLD response to master network requests
☑ LLD support for Frame ID assignment (Diagnostic support)
☑ LLD supprot for NAD assignment (Diagnostic support)

LIN Interface0 Configuration

☑ Enable Interface0 Configuration

Interface0 mode	Master
Interface0 Timer	Timer T5
Interface0 NAD	4
Interface0 Node	Dummy
Response Error Frm	48
Resp Err Frm bit loc	63

LIN Interface1 Configuration

☑ Enable Interface1 Configuration

Interface1 mode	Master
Interface1 Timer	Timer T3
Interface1 NAD	2
Interface1 Node	Master
Response Error Frm	8
Resp Err Frm bit loc	1

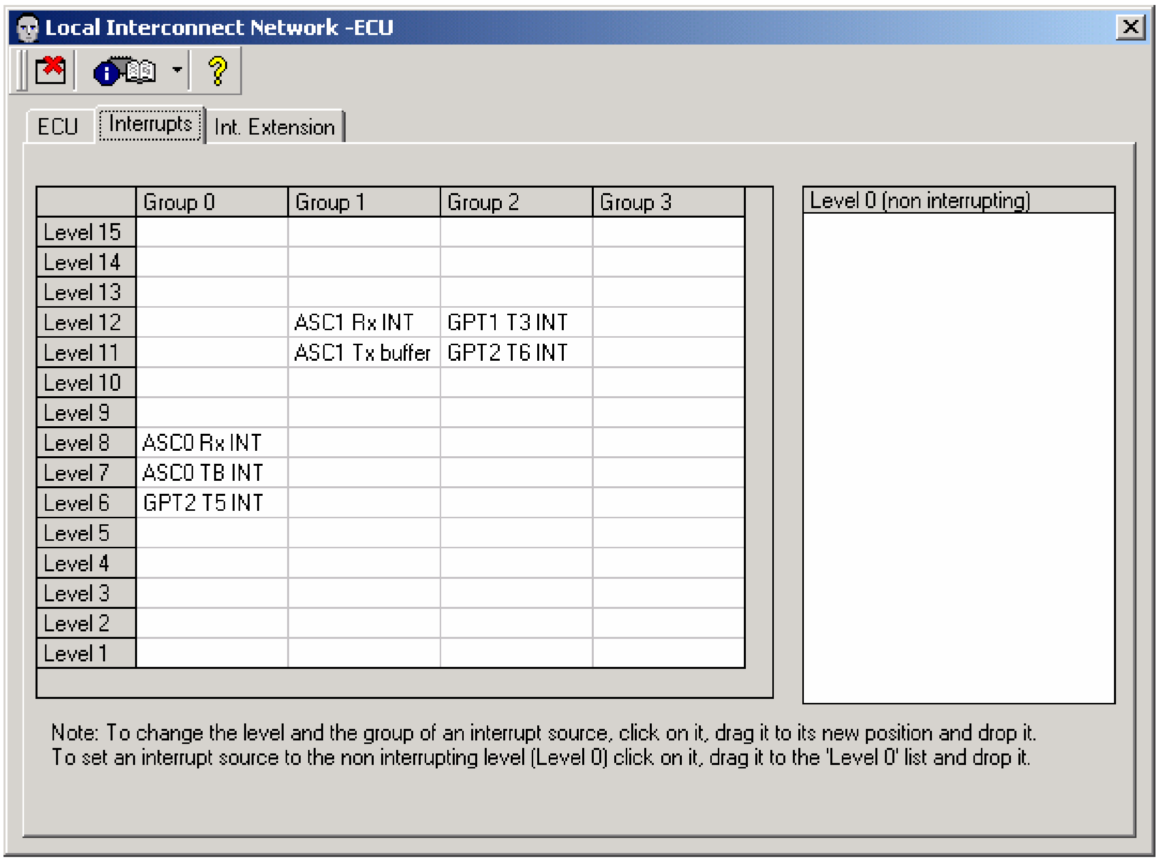
LIN Protocol Information

Supplier ID	0x1F3F
Function ID	0x1E7D
Variant ID	0xC1

Tool restriction: if we want to use only one interface, we have to use interface0 (ASC0). Only when we need two interfaces, we can configure both interface0 (ASC0) & interface1 (ASC1).

Here we like to use interface1 (ASC1). Though we configured two interfaces, we will be using only LIN interface1 (ASC1). We are not using interface0 (ASC0), so interface0 Node configured as Dummy. The reason is, in XC164CM board, ASC0 pin outs are connected to ASC Transceiver, but ASC1 pin outs are connected to LIN Transceiver.

ECU Interrupts



	Group 0	Group 1	Group 2	Group 3
Level 15				
Level 14				
Level 13				
Level 12		ASC1 Rx INT	GPT1 T3 INT	
Level 11		ASC1 Tx buffer	GPT2 T6 INT	
Level 10				
Level 9				
Level 8	ASC0 Rx INT			
Level 7	ASC0 TB INT			
Level 6	GPT2 T5 INT			
Level 5				
Level 4				
Level 3				
Level 2				
Level 1				

Level 0 (non interrupting)

Note: To change the level and the group of an interrupt source, click on it, drag it to its new position and drop it.
To set an interrupt source to the non interrupting level (Level 0) click on it, drag it to the 'Level 0' list and drop it.

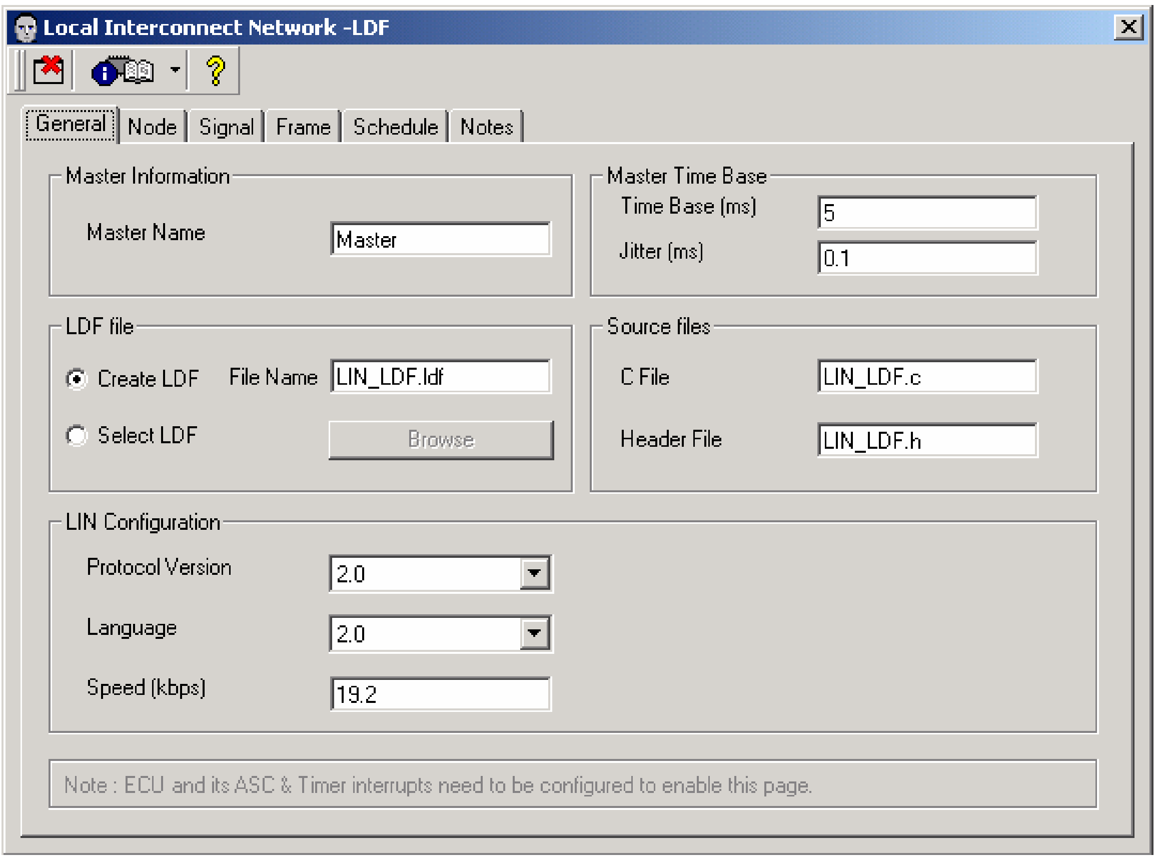
Place the interrupts as shown above

LDF module will not be enabled until all the ECU and interrupts configurations are done.

LDF

General

- No change.



Master Information
Master Name: Master

Master Time Base
Time Base (ms): 5
Jitter (ms): 0.1

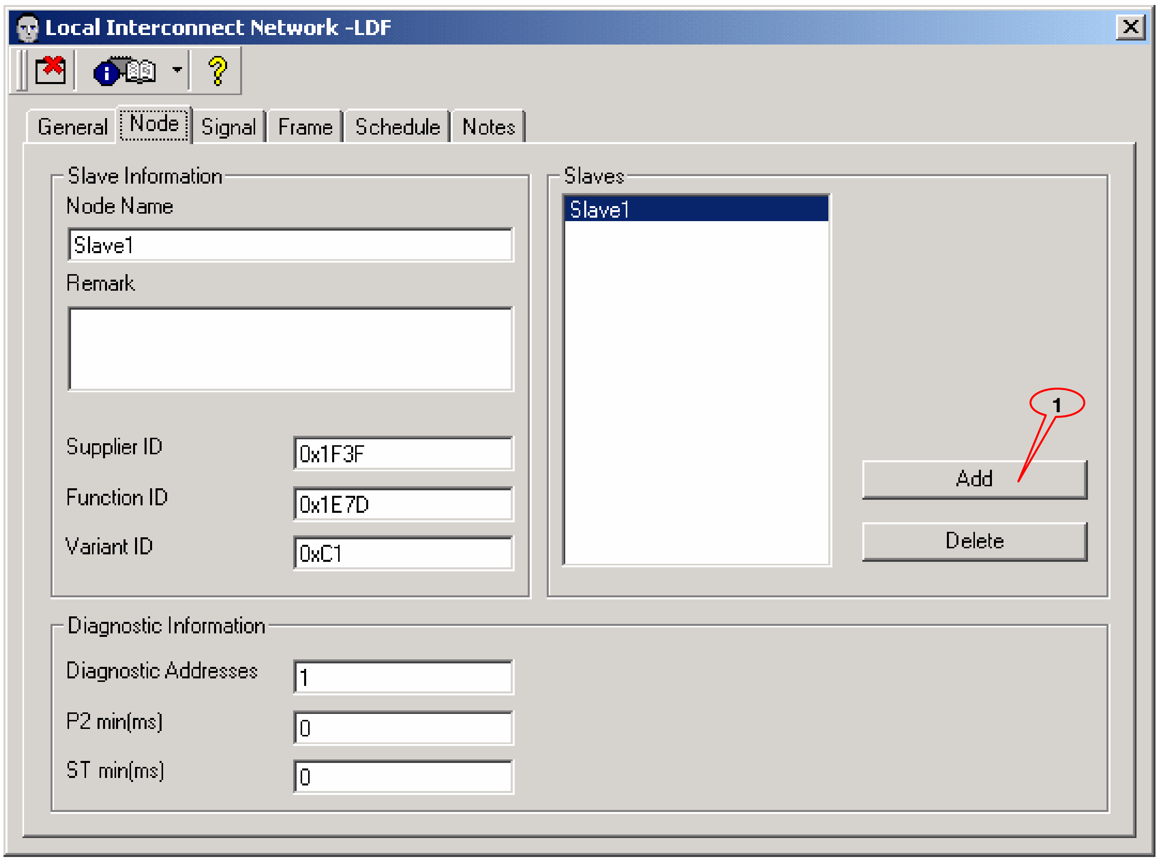
LDF file
[✓ Create LDF] Select LDF
File Name: LIN_LDF.ldf

Source files
C File: LIN_LDF.c
Header File: LIN_LDF.h

LIN Configuration
Protocol Version: 2.0
Language: 2.0
Speed (kbps): 19.2

Note : ECU and its ASC & Timer interrupts need to be configured to enable this page.

Node - configure one slave (Slave1).


Local Interconnect Network -LDF

General | Node | Signal | Frame | Schedule | Notes

Slave Information

Node Name: Slave1

Remark:

Supplier ID	0x1F3F
Function ID	0x1E7D
Variant ID	0xC1

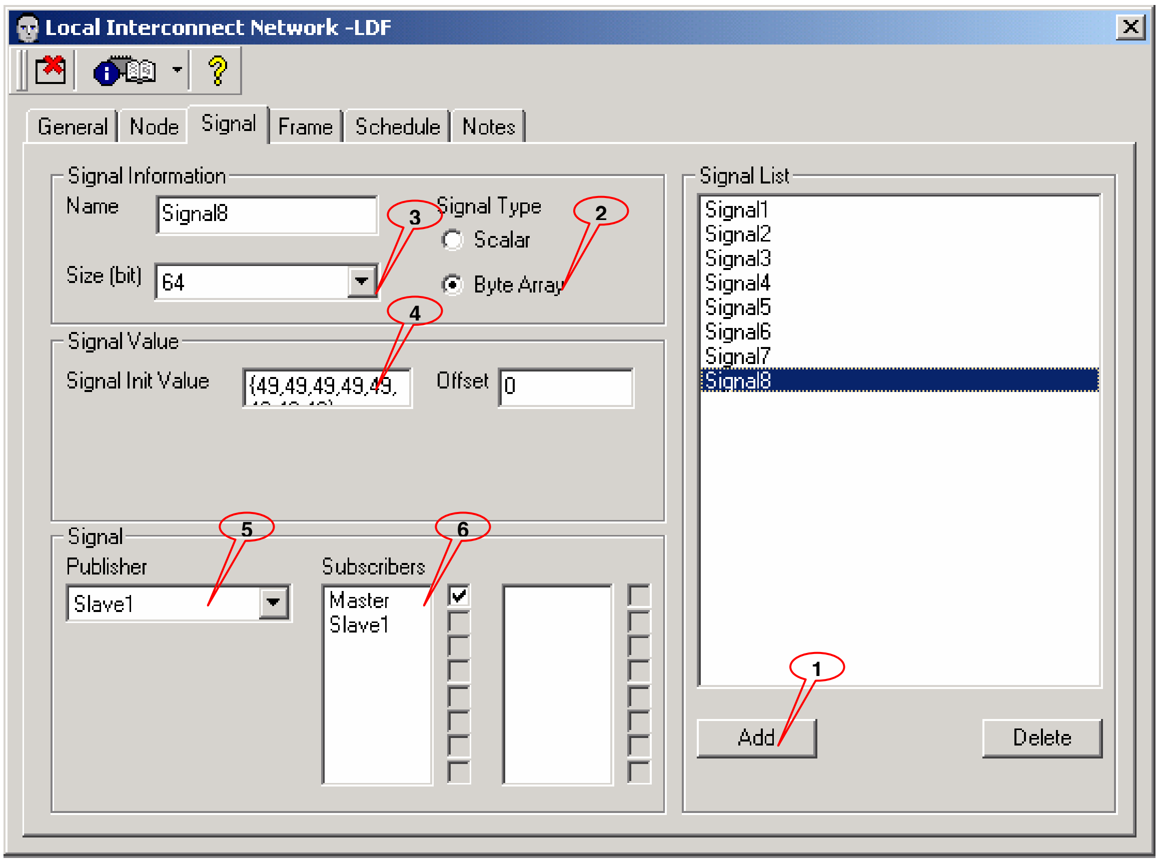
Slaves: Slave1

Diagnostic Information

Diagnostic Addresses	1
P2 min(ms)	0
ST min(ms)	0

Signal - configure 8 signals

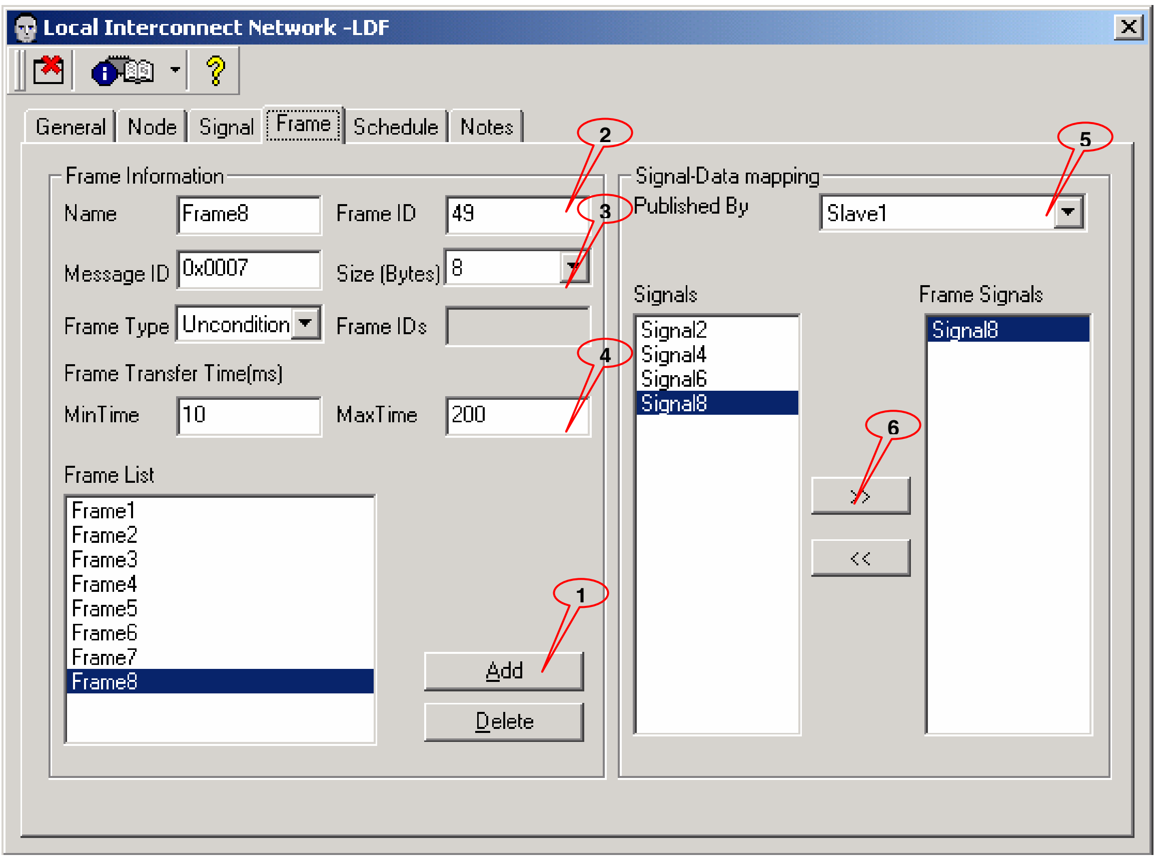
Name	Type	Size	Signal Value	Publisher	Subscriber
Signal1	ByteArray	16	{1,1}	Master	Slave1
Signal2	ByteArray	16	{2,2}	Slave1	Master
Signal3	ByteArray	16	{16,16}	Master	Slave1
Signal4	ByteArray	16	{17,17}	Slave1	Master
Signal5	ByteArray	32	{32,32,32,32}	Master	Slave1
Signal6	ByteArray	32	{33,33,33,33}	Slave1	Master
Signal7	ByteArray	64	{48,48,48,48,48,48,48,48}	Master	Slave1
Signal8	ByteArray	64	{49,49,49,49,49,49,49,49}	Slave1	Master



Name: Signal8
Size (bit): 64
Signal Type: ( ) Scalar [✓ Byte Array]
Signal Init Value: {49,49,49,49,49,
Offset: 0
Publisher: Slave1
Subscribers: Master ✓, Slave1
Signal List: Signal1, Signal2, Signal3, Signal4, Signal5, Signal6, Signal7, Signal8

Frame - configure 8 Unconditional frames

Name	ID	Size	Max Time	Publisher	Frame Signals
Frame1	0	2	200	Master	Signal1
Frame2	1	2	200	Slave1	Signal2
Frame3	16	2	200	Master	Signal3
Frame4	17	2	200	Slave1	Signal4
Frame5	32	4	200	Master	Signal5
Frame6	33	4	200	Slave1	Signal6
Frame7	48	8	200	Master	Signal7
Frame8	49	8	200	Slave1	Signal8



Frame Information
Name: Frame8
Frame ID: 49
Message ID: 0x0007
Size (Bytes): 8
Frame Type: Uncondition
Frame IDs:
Frame Transfer Time(ms)
MinTime: 10
MaxTime: 200

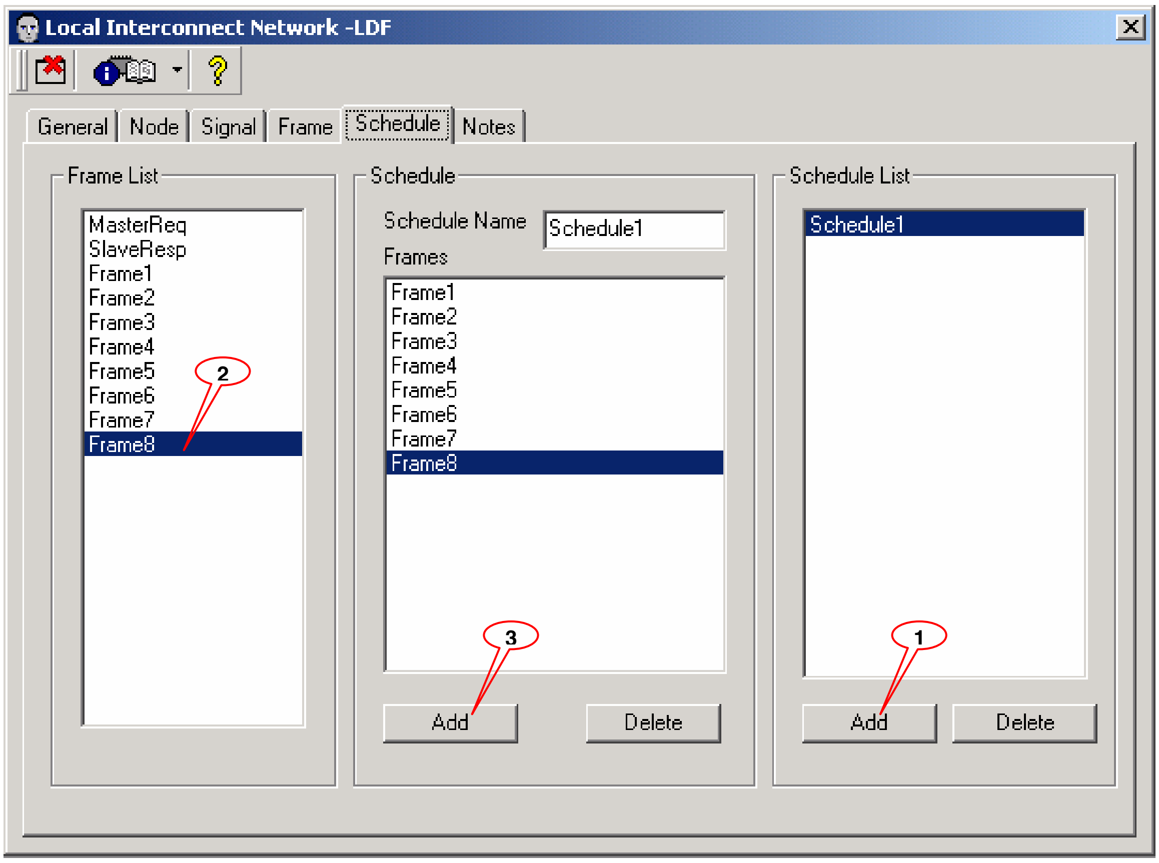
Frame List
Frame1
Frame2
Frame3
Frame4
Frame5
Frame6
Frame7
Frame8

Signal-Data mapping
Published By: Slave1

Signals
Signal2
Signal4
Signal6
Signal8

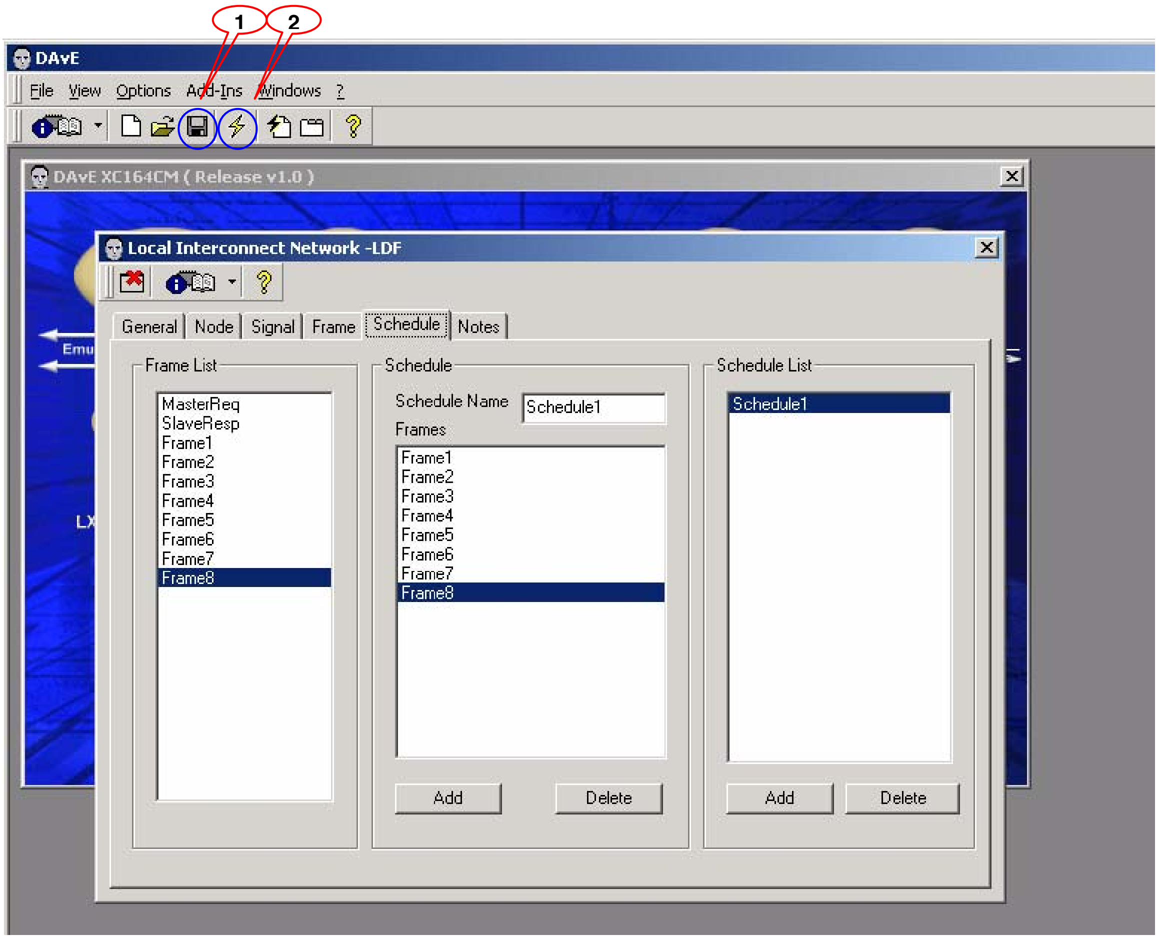
Frame Signals
Signal8

Schedule - configure one schedule (schedule1) with 8 frames.

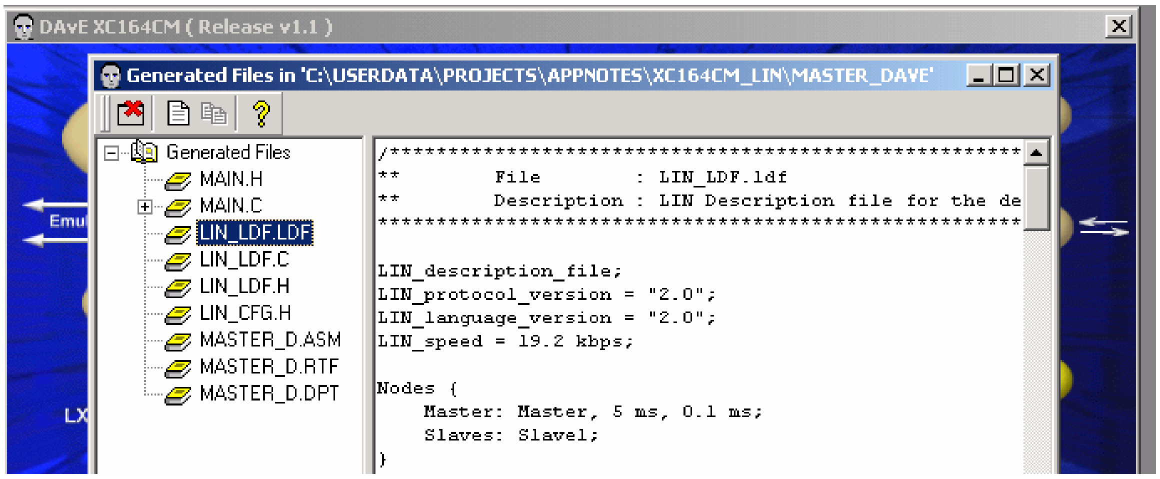


Save & Generate code

(1) Save the configurations as ‘Master_D’ dave project in Master_Dave folder. (2) Generate code. DAvE generates configuration files ()



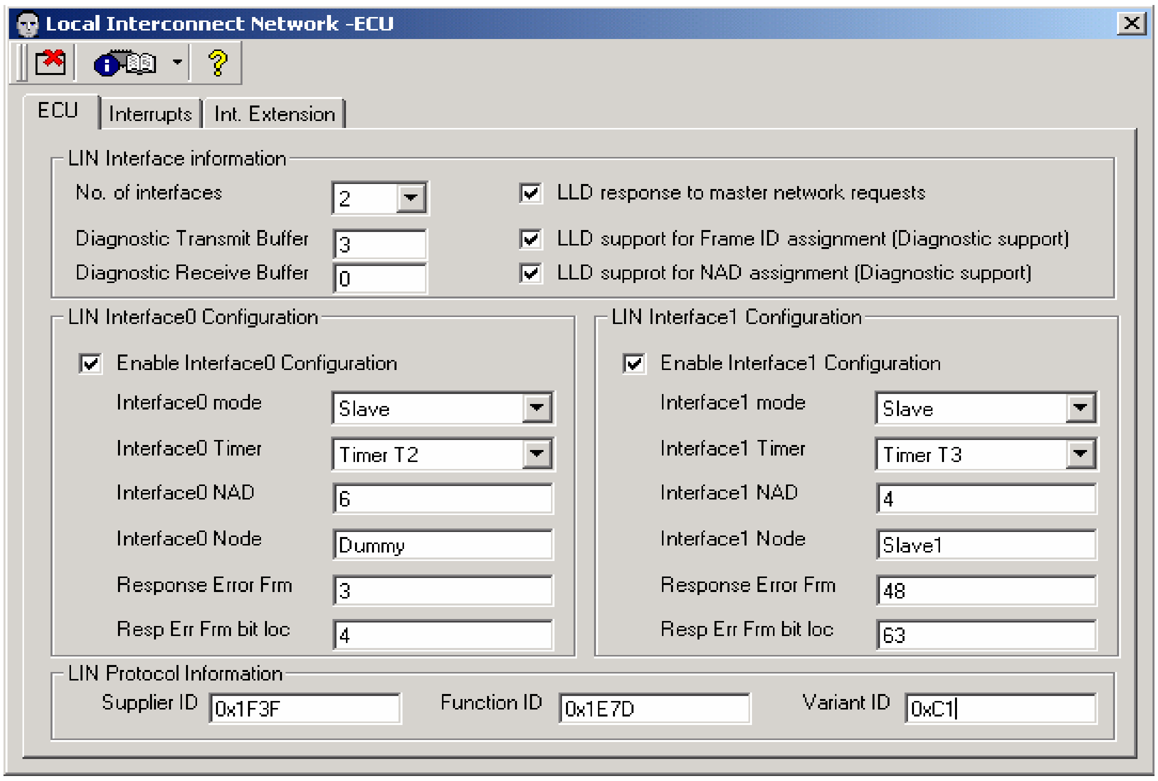
Generated files


DAvE XC164CM (Release v1.1)
Generated Files in 'C:\USERDATA\PROJECTS\APPNOTES\XC164CM_LIN\MASTER_DAVE'
Generated Files
MAIN.H
MAIN.C
LIN_LDF.LDF
LIN_LDF.C
LIN_LDF.H
LIN_CFG.H
MASTER_D.ASM
MASTER_D.RTF
MASTER_D.DPT

```
/*************************************************************
**        File        : LIN_LDF.ldf
**        Description : LIN Description file for the de
*************************************************************

LIN_description_file;
LIN_protocol_version = "2.0";
LIN_language_version = "2.0";
LIN_speed = 19.2 kbps;

Nodes {
    Master: Master, 5 ms, 0.1 ms;
    Slaves: Slave1;
}
```

5.2 Slave Configurations

ECU



LIN Interface information

No. of interfaces: 2
Diagnostic Transmit Buffer: 3
Diagnostic Receive Buffer: 0

☑ LLD response to master network requests
☑ LLD support for Frame ID assignment (Diagnostic support)
☑ LLD supprot for NAD assignment (Diagnostic support)

LIN Interface0 Configuration

☑ Enable Interface0 Configuration

Interface0 mode	Slave
Interface0 Timer	Timer T2
Interface0 NAD	6
Interface0 Node	Dummy
Response Error Frm	3
Resp Err Frm bit loc	4

LIN Interface1 Configuration

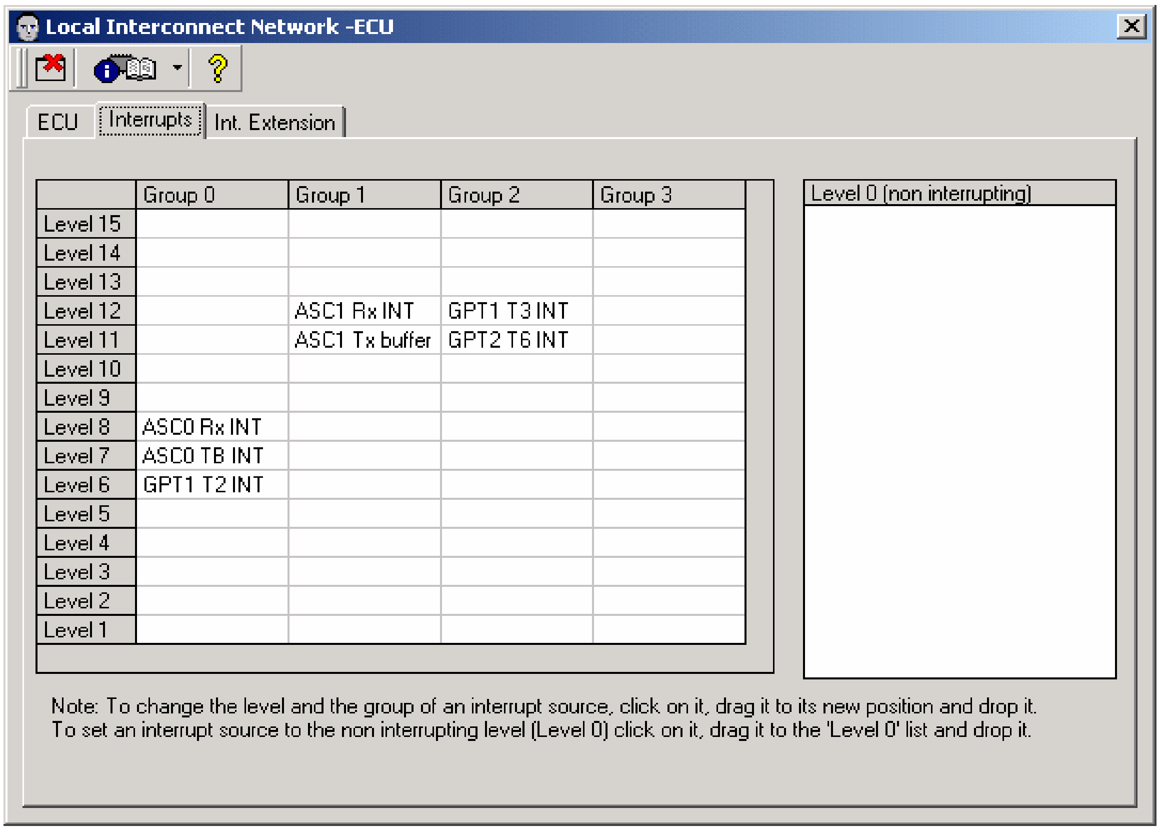
☑ Enable Interface1 Configuration

Interface1 mode	Slave
Interface1 Timer	Timer T3
Interface1 NAD	4
Interface1 Node	Slave1
Response Error Frm	48
Resp Err Frm bit loc	63

LIN Protocol Information

Supplier ID: 0x1F3F
Function ID: 0x1E7D
Variant ID: 0xC1

ECU Interrupts

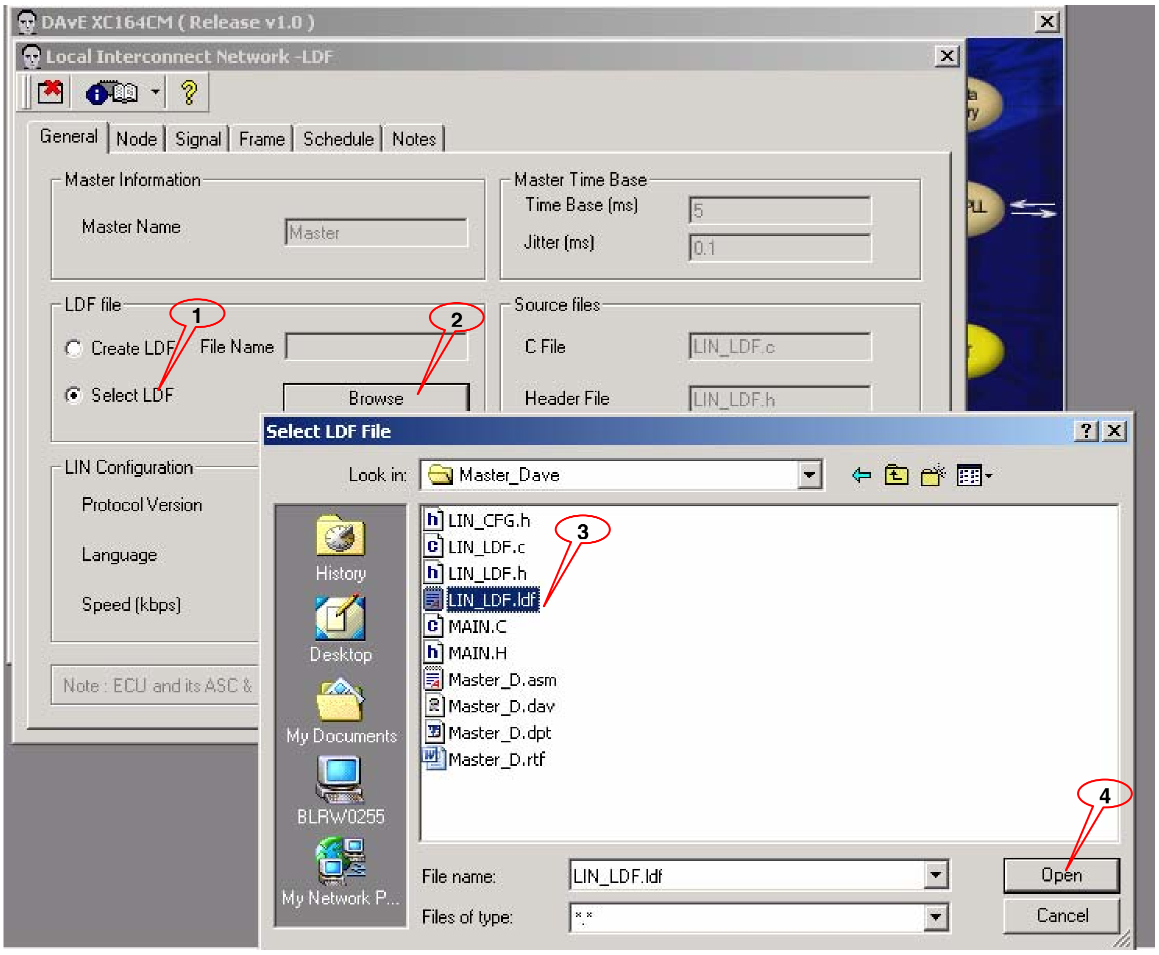


	Group 0	Group 1	Group 2	Group 3
Level 15				
Level 14				
Level 13				
Level 12		ASC1 Rx INT	GPT1 T3 INT	
Level 11		ASC1 Tx buffer	GPT2 T6 INT	
Level 10				
Level 9				
Level 8	ASC0 Rx INT			
Level 7	ASC0 TB INT			
Level 6	GPT1 T2 INT			
Level 5				
Level 4				
Level 3				
Level 2				
Level 1				

Level 0 (non interrupting)

Note: To change the level and the group of an interrupt source, click on it, drag it to its new position and drop it.
To set an interrupt source to the non interrupting level (Level 0) click on it, drag it to the 'Level 0' list and drop it.

LDF



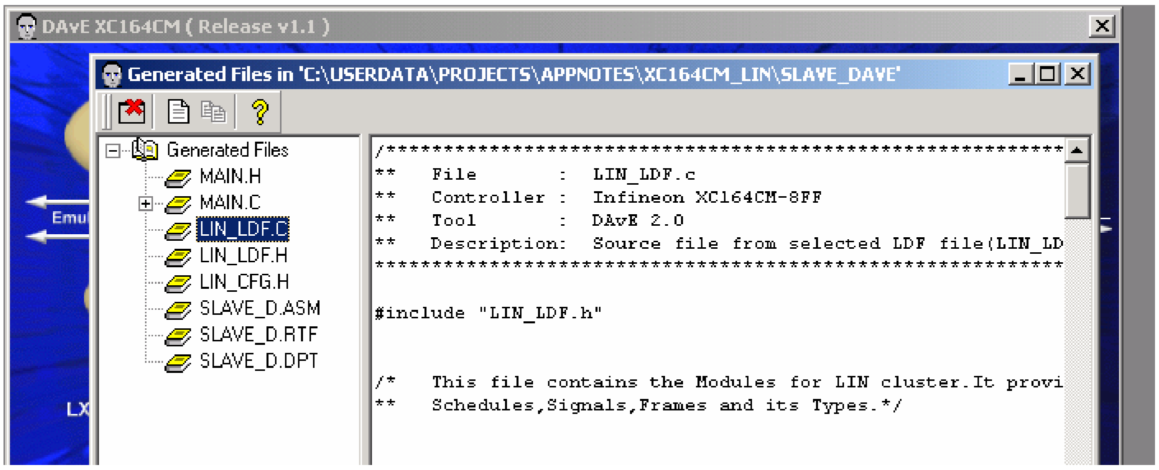
Master Name: Master
Time Base (ms): 5
Jitter (ms): 0.1
File Name:
C File: LIN_LDF.c
Header File: LIN_LDF.h

Look in: Master_Dave

```
LIN_CFG.h
LIN_LDF.c
LIN_LDF.h
LIN_LDF.ldf
MAIN.C
MAIN.H
Master_D.asm
Master_D.dav
Master_D.dpt
Master_D.rtf
```

File name: LIN_LDF.ldf
Files of type: *.*

Generated files


DAvE XC164CM (Release v1.1)
Generated Files in 'C:\USERDATA\PROJECTS\APPNOTES\XC164CM_LIN\SLAVE_DAVE'
Emul
LX

```
Generated Files
    MAIN.H
    MAIN.C
    LIN_LDF.C
    LIN_LDF.H
    LIN_CFG.H
    SLAVE_D.ASM
    SLAVE_D.RTF
    SLAVE_D.DPT
```

```
/*****************************************************************
**    File        :  LIN_LDF.c
**    Controller  :  Infineon XC164CM-8FF
**    Tool        :  DAvE 2.0
**    Description:   Source file from selected LDF file(LIN_LD
*****************************************************************

#include "LIN_LDF.h"


/*    This file contains the Modules for LIN cluster.It provi
**    Schedules,Signals,Frames and its Types.*/
```

5.3 Integration steps – Master Node

1. Create a folder “Source”
2. Put the LIN LLD Source files (Compiler.h, LIN_API.h, LIN_IDL.h, LIN_IIL.c, LIN_IIL.h, SYS_API.h, SYS_IIL.c) in this folder.
3. Create a folder “Master_App”
4. Put the DAvE generated file (LIN_CFG.h, LIN_LDF.h, LIN_LDF.c & LIN_test.c) in this folder from DAvE project ‘Master_D’ which is in folder ‘Master_DAvE’.
5. Create a folder “Master_keil”
6. Create a project “Master_lin” using keil UV3.

How to use keil UV3 IDE

Keil Software development tools for XC16x support embedded software development. The industry-standard Keil C Compiler, Macro Assembler, Debugger, Real-time Kernel, and Single-board Computers support ALL XC16x derivatives.

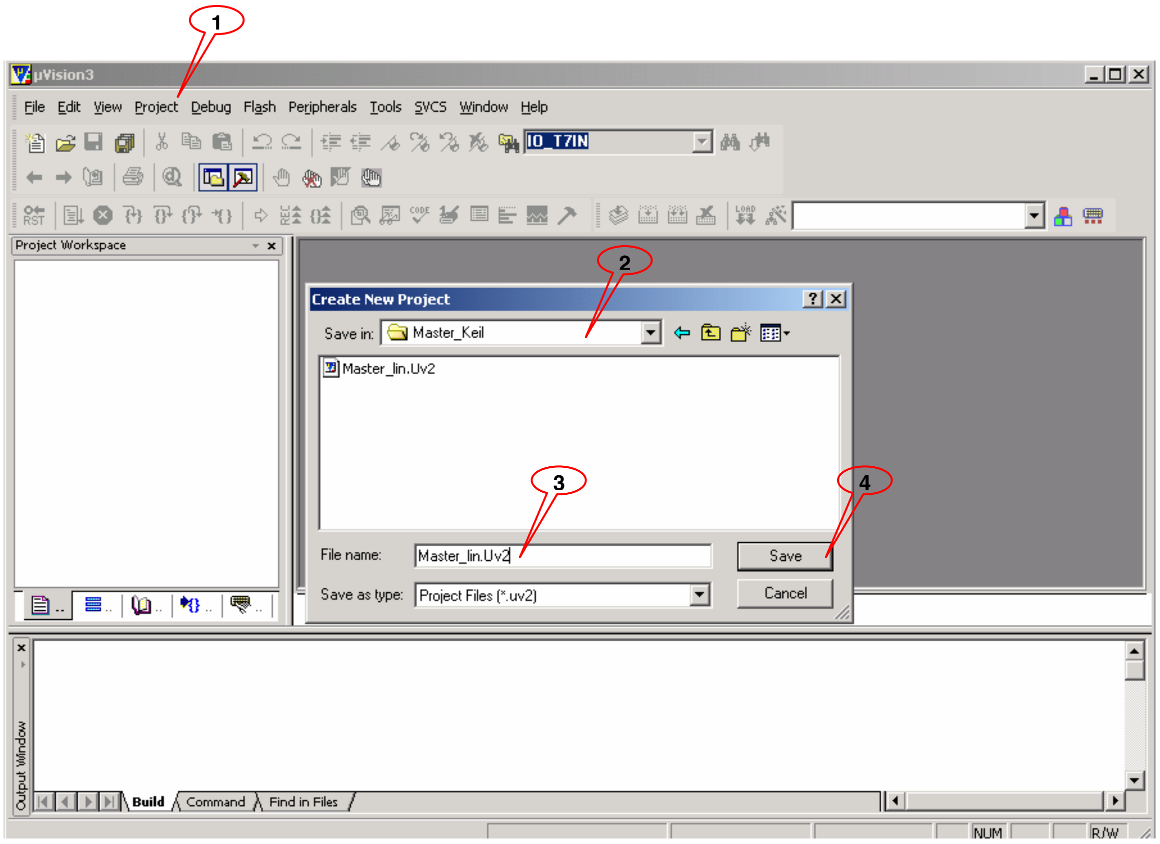
The starter kit package contains a CD for the installation of the evaluation version of keil IDE.
To install keil UV3 IDE, Run the 'Setup' file inside the CD. For more details go to www.keil.com.

New Project - Master

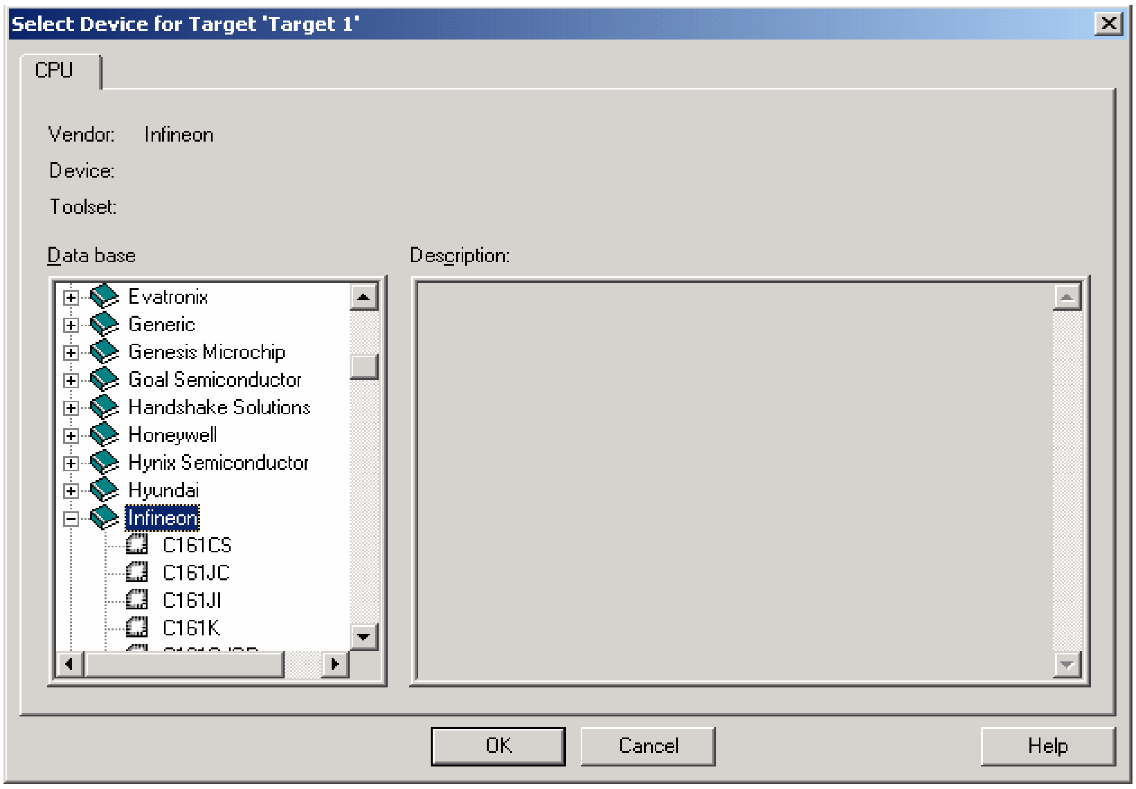
First start the keil UV3 IDE.

Create a new project

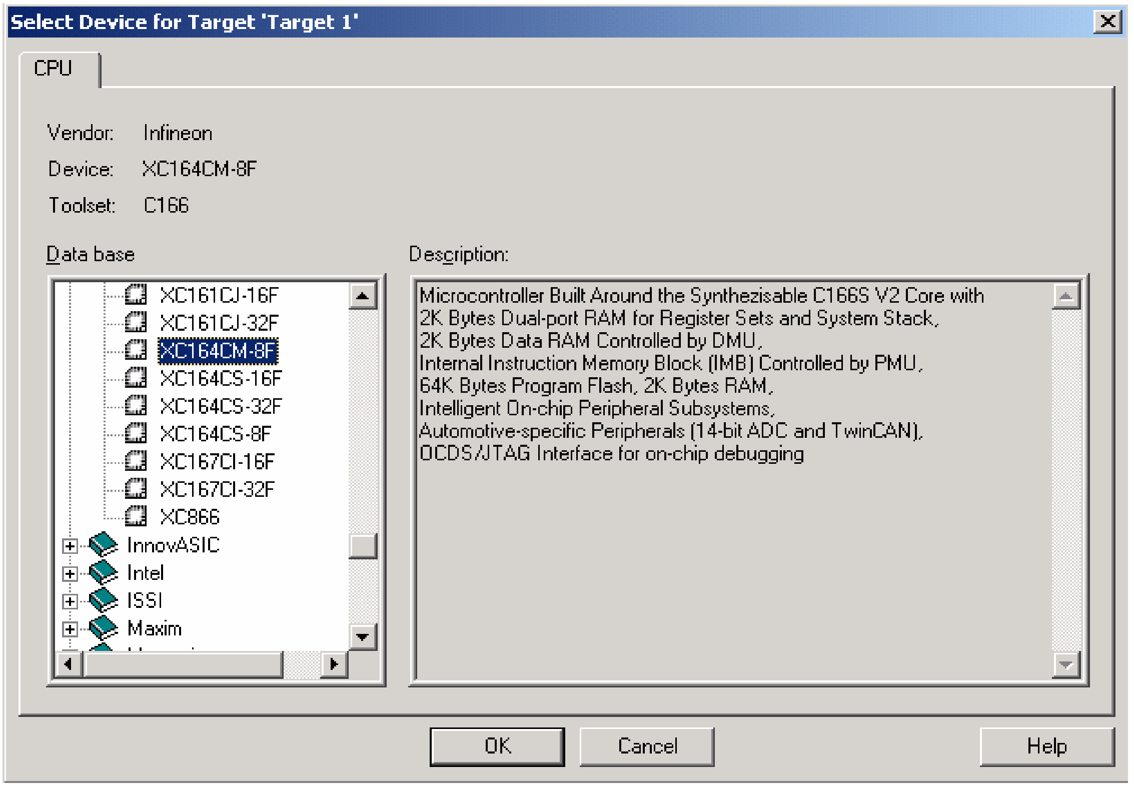
- Project -> New Project
- Select folder "Master_keil" where you want to save your project.
- Name the project "Master_LIN" and click 'Save'.



Select Device for Target -> Infineon -> XC164CM-8F



Vendor: Infineon
Device:
Toolset:

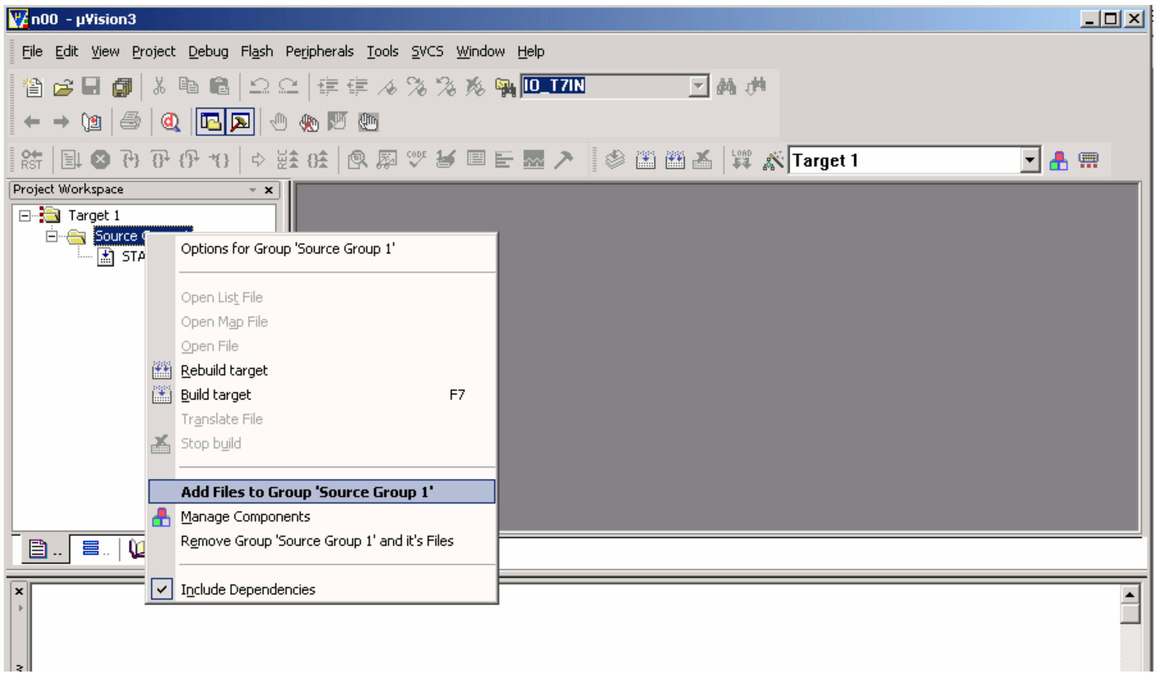


Vendor: Infineon
Device: XC164CM-8F
Toolset: C166

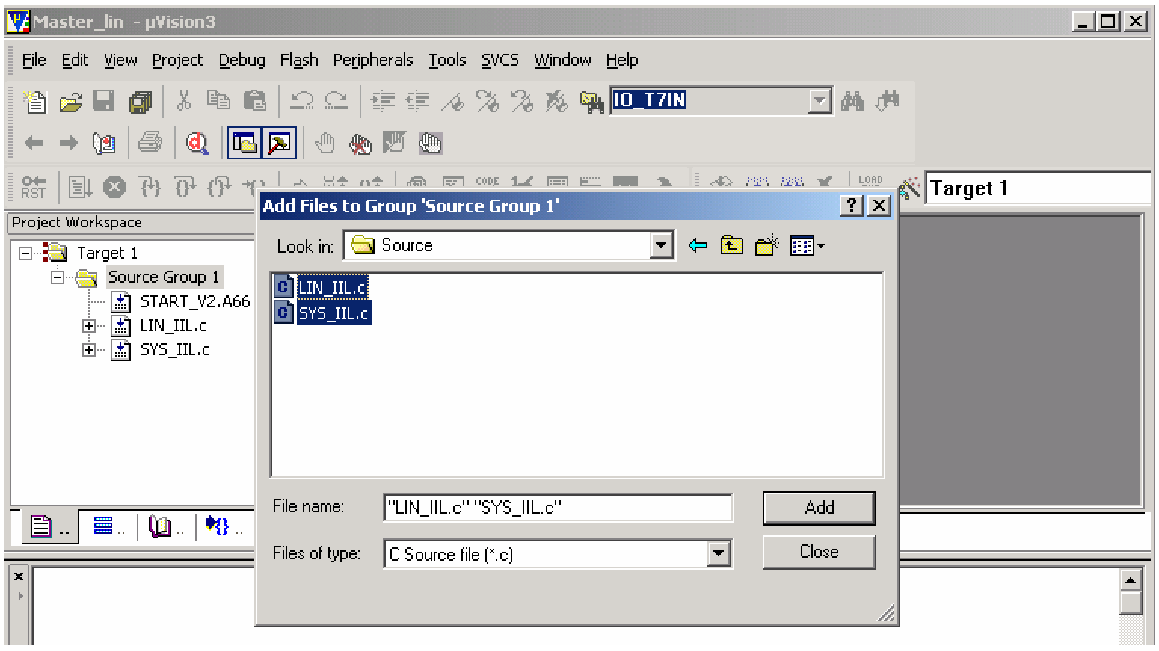
Microcontroller Built Around the Synthezisable C166S V2 Core with
2K Bytes Dual-port RAM for Register Sets and System Stack,
2K Bytes Data RAM Controlled by DMU,
Internal Instruction Memory Block (IMB) Controlled by PMU,
64K Bytes Program Flash, 2K Bytes RAM,
Intelligent On-chip Peripheral Subsystems,
Automotive-specific Peripherals (14-bit ADC and TwinCAN),
OCDS/JTAG Interface for on-chip debugging

Add the files to the project

Expand Target1 -> Right click on Source Group1 -> Add Files to Group ‘Source Group 1’.



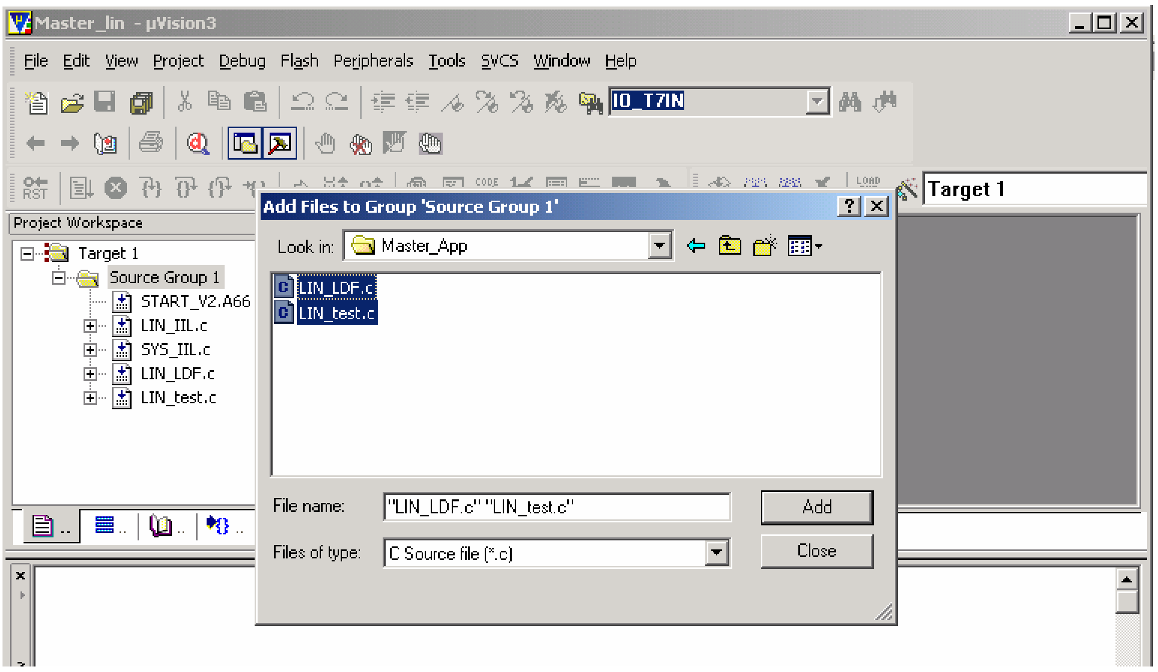
Browse to ‘Source’ folder and select LIN_IIL.c and SYS_IIL.c files to add to the project.



File name: "LIN_IIL.c" "SYS_IIL.c"

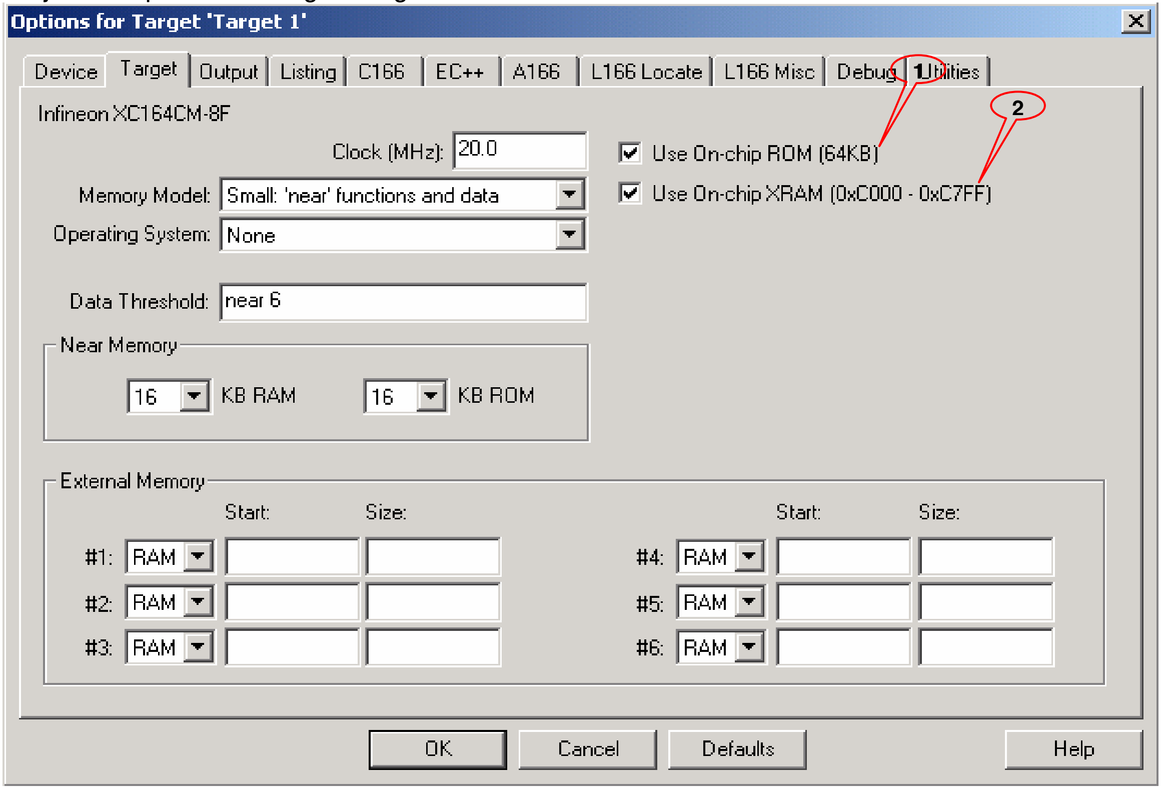
Files of type: C Source file (*.c)

Browse to 'Master_App' folder and select LIN_LDF.c and LIN_test.c files to add to the project



Options for Target

Project -> Options for Target 'Target 1'



Infineon XC164CM-8F

Clock (MHz): 20.0

Memory Model: Small: 'near' functions and data

Operating System: None

Data Threshold: near 6

Near Memory: 16 KB RAM, 16 KB ROM

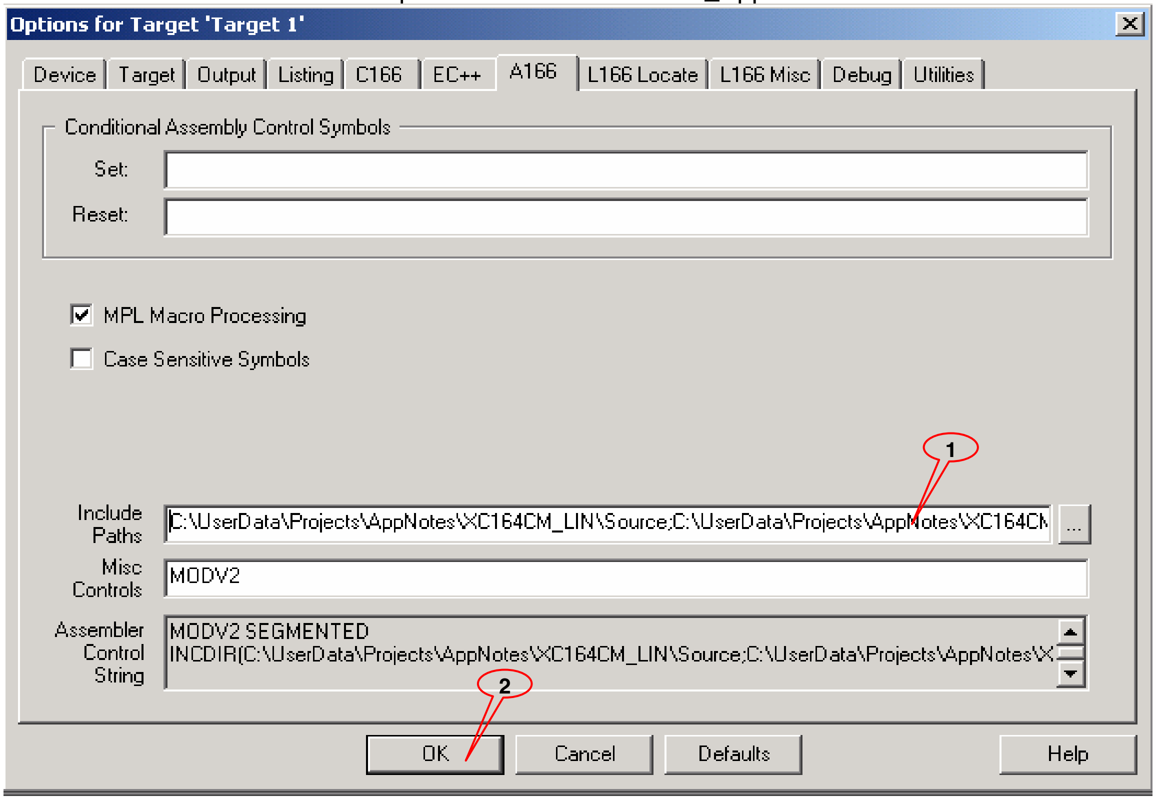
External Memory: #1: RAM, #2: RAM, #3: RAM, #4: RAM, #5: RAM, #6: RAM (Start: / Size: empty)

☑ Use On-chip ROM (64KB)

☑ Use On-chip XRAM (0xC000 - 0xC7FF)

Include Paths

Go to Tab 'A166' and include the paths for source & Master_App folder as shown.



Conditional Assembly Control Symbols

Set:

Reset:

☑ MPL Macro Processing

☐ Case Sensitive Symbols

Include Paths: C:\UserData\Projects\AppNotes\XC164CM_LIN\Source;C:\UserData\Projects\AppNotes\XC164CM

Misc Controls: MODV2

Assembler Control String: MODV2 SEGMENTED INCDIR(C:\UserData\Projects\AppNotes\XC164CM_LIN\Source;C:\UserData\Projects\AppNotes\X

5.4 Master Test application

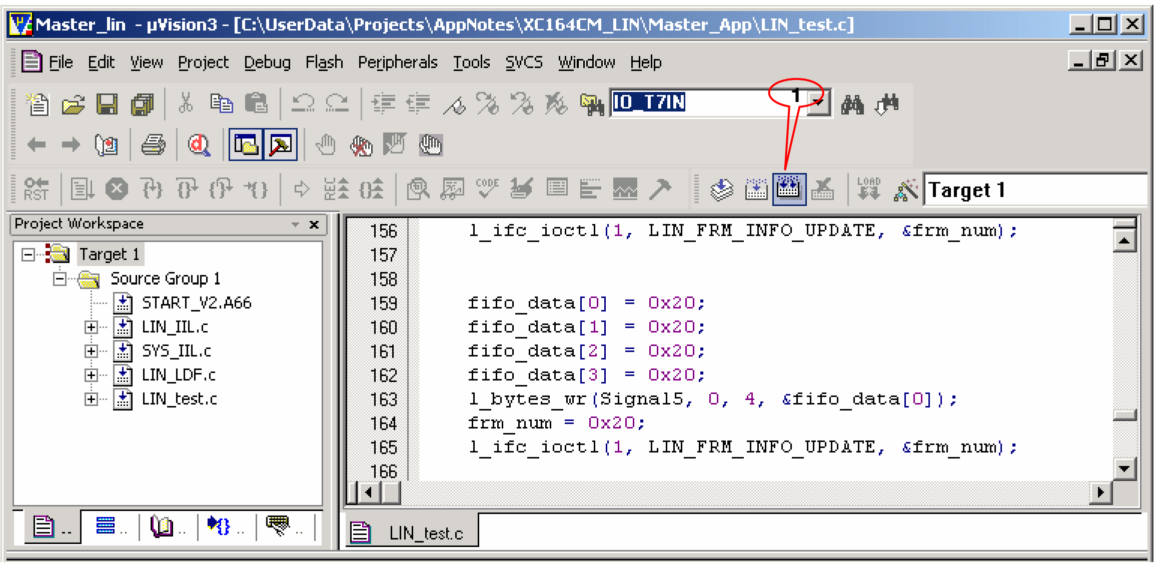
The LIN_test.c file has the master test application. This initializes the master interface, connects to the LIN cluster, initializes the system timer and sends the wake up signal to slave. This transmits four frames and receives four frames from salve.

```
//  Include LLD defined header files

#include "COMPILER.h"
#include "LIN_CFG.h"
#include "LIN_IDL.h"
#include "LIN_IIL.h"
#include "LIN_API.h"
#include "SYS_API.h"

l_u8 frm_num = 0;
l_u8 data[8] = {0};


l_u16 main (void)
{
 // Initialize Master interface

  LIN_calc_freq();         // calculates the system frequency.
  l_ifc_init(1);           // Initialize the interface ASC1 for LIN.
  l_ifc_connect(1);        // connect and make it ready to transmit & receive frames.
  l_sys_init();            // Initialize system timer, used to schedule frames.
  LIN_SET(SYS_HW_PSW, SYS_HW_PSW_IEN);       //Enable global interrupt
  l_ifc_wake_up(1);        // Send wakeup signal to wakeup the sleeping slave.
  l_sch_set(1,0,0);        //  Initialize the schedule table.

// Transmit and receive data

  while(1)
  {
        data[0] = 0x01;                                     // Initial value to data
        data[1] = 0x01;
        l_bytes_wr(Signal1, 0, 2, &data[0]);                // update the signal with data.
        frm_num = 0;                                        // frame number has new data.
        l_ifc_ioctl(1, LIN_FRM_INFO_UPDATE, &frm_num);      // indicates the updated frame.

        data[0] = 0x10;                                     // Frame (0x10) with 2 bytes
        data[1] = 0x10;
        l_bytes_wr(Signal3, 0, 2, &data[0]);
        frm_num = 0x10;
        l_ifc_ioctl(1, LIN_FRM_INFO_UPDATE, &frm_num);

        data[0] = 0x20;                                     // Frame (0x20) with 4 bytes
        data[1] = 0x20;
        data[2] = 0x20;
        data[3] = 0x20;
        l_bytes_wr(Signal5, 0, 4, &data[0]);
        frm_num = 0x20;
        l_ifc_ioctl(1, LIN_FRM_INFO_UPDATE, &frm_num);
```

```
            data[0] = 0x30;                             // Frame (0x30) with 8 bytes
            data[1] = 0x30;
            data[2] = 0x30;
            data[3] = 0x30;
            data[4] = 0x30;
            data[5] = 0x30;
            data[6] = 0x30;
            data[7] = 0x30;
            l_bytes_wr(Signal7, 0, 8, &data[0]);
            frm_num = 0x30;
            l_ifc_ioctl(1, LIN_FRM_INFO_UPDATE, &frm_num);
  }
}

l_flg_clr(0x11);  \\ clears the status flag of the frame (0x11)
l_flg_tst(0x11);  \\ shows the status flag of the frame (0x11)
```

These two functions can be used to monitor the transmission and reception of the frames in the LIN. Use the l_flg_clr(frame no) before the transmission from either side and check the status flag using l_flg_tst(frame no) for the successful transmission and reception.

Generate executable 'Master_LIN'


Master_lin - µVision3 - [C:\UserData\Projects\AppNotes\XC164CM_LIN\Master_App\LIN_test.c]
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
IO_T7IN
1
Target 1
Project Workspace
Target 1
Source Group 1
START_V2.A66
LIN_IIL.c
SYS_IIL.c
LIN_LDF.c
LIN_test.c

```
    l_ifc_ioctl(1, LIN_FRM_INFO_UPDATE, &frm_num);


    fifo_data[0]  = 0x20;
    fifo_data[1]  = 0x20;
    fifo_data[2]  = 0x20;
    fifo_data[3]  = 0x20;
    l_bytes_wr(Signal5, 0, 4, &fifo_data[0]);
    frm_num = 0x20;
    l_ifc_ioctl(1, LIN_FRM_INFO_UPDATE, &frm_num);
```

5.5 Integration steps – Slave Node

Create a folder 'Slave_App'

Put the DAvE generated file (LIN_CFG.h, LIN_LDF.h, LIN_LDF.c & LIN_test.c) in this folder from DAvE project 'Slave_D' which is in folder 'Slave_DAvE'.

Create a folder 'Slave_keil'

Create a project 'Slave_lin' using keil UV3. Follow the same steps shown for Master setup.
Select the LIN_LDF.c and LIN_test.c files from 'Slave_App' folder.

5.6 Slave Test application

The LIN_test.c file has the slave test application.

```
//  Include LLD defined header files

#include "COMPILER.h"
#include "LIN_CFG.h"
#include "LIN_IDL.h"
#include "LIN_IIL.h"
#include "LIN_API.h"
#include "SYS_API.h"

l_u8 frm_num = 0;
l_u8 data[8] = {0};

l_u16 main (void)
{
 // Initialize Master interface

  LIN_calc_freq();          // calculates the system frequency.
  l_ifc_init(1);            // Initialize the interface ASC1 for LIN.
  l_ifc_connect(1);         // connect and make it ready to transmit & receive frames.
  l_sys_init();             // Initialize system timer, used to schedule frames.
  LIN_SET(SYS_HW_PSW, SYS_HW_PSW_IEN);        //Enable global interrupt

// receive and Transmit frames

 while(1)
  {
        data[0] = 0x02;
        data[1] = 0x02;
        l_bytes_wr(Signal2, 0, 2, &data[0]);
        frm_num = 1;
        l_ifc_ioctl(0, LIN_FRM_INFO_UPDATE, &frm_num);

        data[0] = 0x11;
        data[1] = 0x11;
        l_bytes_wr(Signal4, 0, 2, &data[0]);
        frm_num = 0x11;
        l_ifc_ioctl(0, LIN_FRM_INFO_UPDATE, &frm_num);
```

```
            data[0] = 0x21;
            data[1] = 0x21;
            data[2] = 0x21;
            data[3] = 0x21;
            l_bytes_wr(Signal6, 0, 4, &data[0]);
            frm_num = 0x21;
            l_ifc_ioctl(0, LIN_FRM_INFO_UPDATE, &frm_num);

            data[0] = 0x31;
            data[1] = 0x31;
            data[2] = 0x31;
            data[3] = 0x31;
            data[4] = 0x31;
            data[5] = 0x31;
            data[6] = 0x31;
            data[7] = 0x31;
            l_bytes_wr(Signal8, 0, 8, &fifo_data[0]);
            frm_num = 0x31;
            l_ifc_ioctl(0, LIN_FRM_INFO_UPDATE, &frm_num);

  }
}
```

```
l_flg_clr(0x10);  \\ clears the status flag of the frame (0x10)
l_flg_tst(0x10);  \\ shows the status flag of the frame (0x10)
```

These two functions can be used to monitor the transmission and reception of the frames in the LIN. Use the l_flg_clr(frame no) before the transmission from either side and check the status flag using l_flg_tst(frame no) for the successful transmission and reception.

Generate the executable ‘Slave_LIN’

Use pls UAD to download the code to XC164CM.

5.7 Download the executable code to the XC164CM

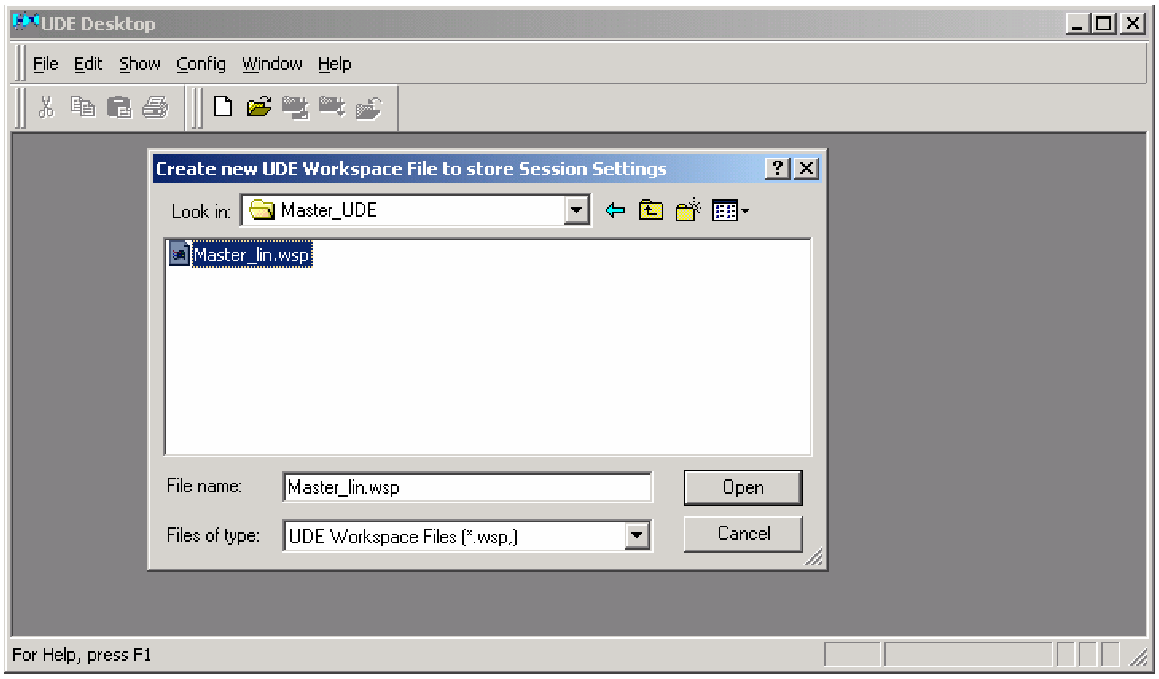
The pls provides UAD (Universal access device) along with UDE (Universal Debug Engine). Install the UDE, using setup given in the pls UAE CD.

Use the parallel port to connect the PC and UAD and use JTAG to connect UAD and XC164CM board.

New workspace – Master Node

Start the UDE by double clicking on the UDE Desktop

File -> New workspace



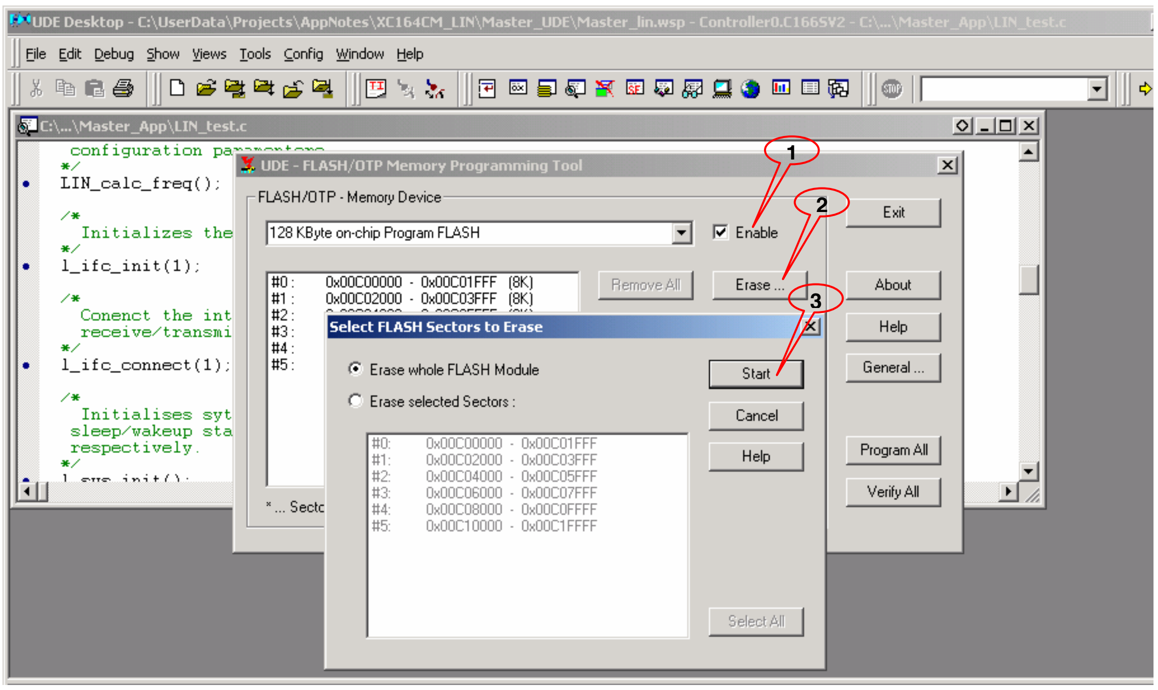
File name: Master_lin.wsp

Files of type: UDE Workspace Files (*.wsp.)

Name the workspace "Master_LIN" and click on open.

Program the flash

Tools -> Flash Programming



UDE Desktop - C:\UserData\Projects\AppNotes\XC164CM_LIN\Master_UDE\Master_lin.wsp - Controller0.C166SV2 - C:\...\Master_App\LIN_test.c

C:\...\Master_App\LIN_test.c

```
  configuration parameters
 */
 LIN_calc_freq();

 /*
  Initializes the
 */
 l_ifc_init(1);

 /*
  Conenct the int
  receive/transmi
 */
 l_ifc_connect(1);

 /*
  Initialises syt
  sleep/wakeup sta
  respectively.
 */
```

UDE - FLASH/OTP Memory Programming Tool

FLASH/OTP - Memory Device: 128 KByte on-chip Program FLASH [✓ Enable]

```
#0 :  0x00C00000 - 0x00C01FFF  (8K)
#1 :  0x00C02000 - 0x00C03FFF  (8K)
#2 :
#3 :
#4 :
#5 :
```

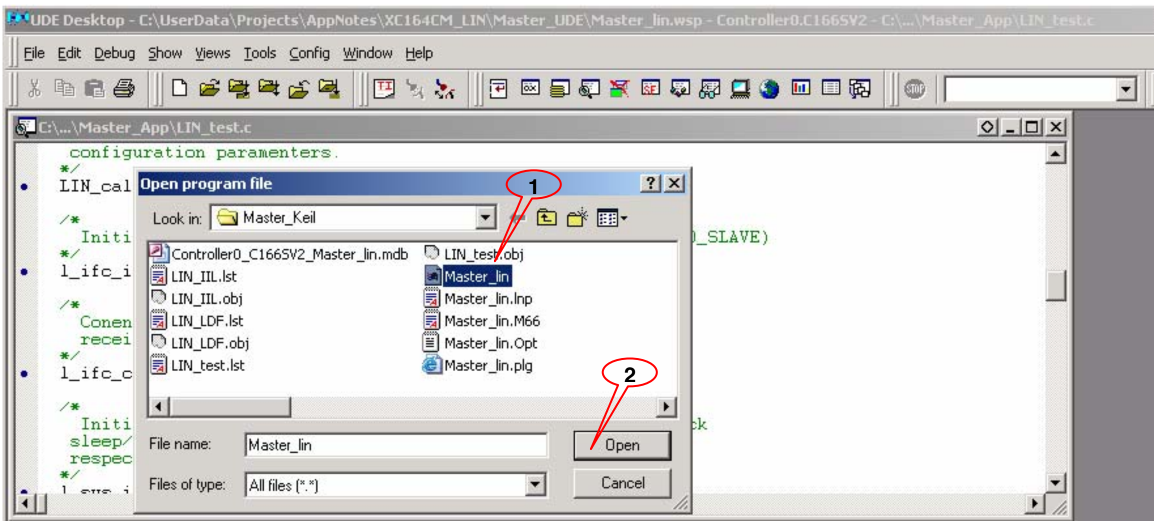
Select FLASH Sectors to Erase

[✓ Erase whole FLASH Module]
Erase selected Sectors :

```
#0:  0x00C00000 - 0x00C01FFF
#1:  0x00C02000 - 0x00C03FFF
#2:  0x00C04000 - 0x00C05FFF
#3:  0x00C06000 - 0x00C07FFF
#4:  0x00C08000 - 0x00C0FFFF
#5:  0x00C10000 - 0x00C1FFFF
```

Load Program

File -> Flash Program



UDE Desktop - C:\UserData\Projects\AppNotes\XC164CM_LIN\Master_UDE\Master_lin.wsp - Controller0.C166SV2 - C:\...\Master_App\LIN_test.c

Open program file

Look in: Master_Keil

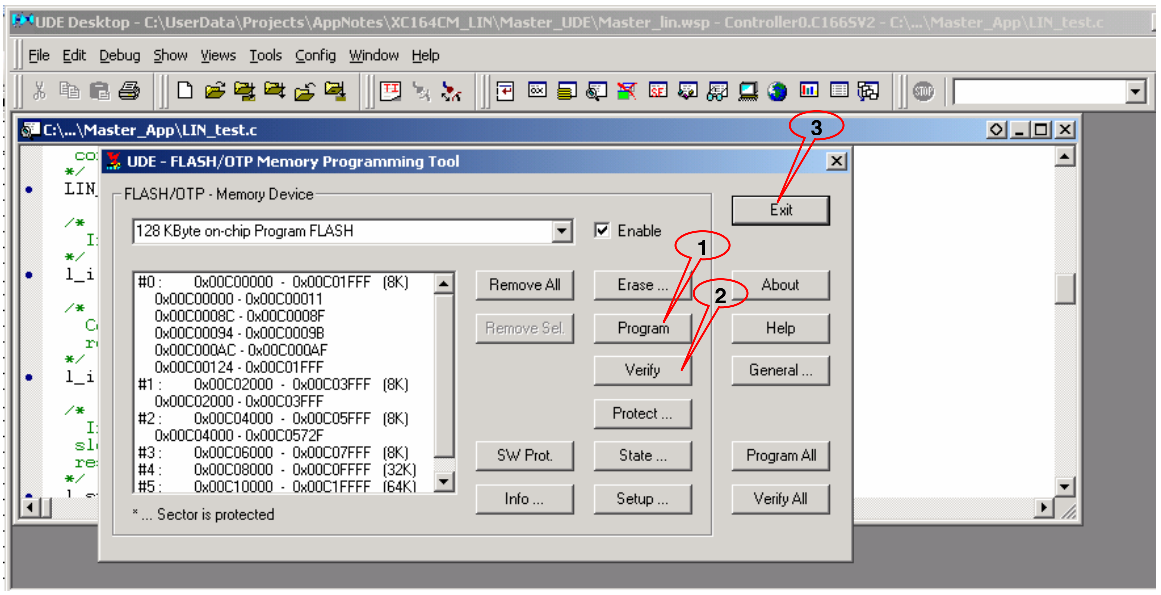
```
Controller0_C166SV2_Master_lin.mdb   LIN_test.obj
LIN_IIL.lst                          Master_lin
LIN_IIL.obj                          Master_lin.lnp
LIN_LDF.lst                          Master_lin.M66
LIN_LDF.obj                          Master_lin.Opt
LIN_test.lst                         Master_lin.plg
```

File name: Master_lin

Files of type: All files (*.*)

Select the executable file ‘Master_LIN’ from folder ‘Master_keil’ and click on open.

Memory programming



UDE Desktop - C:\UserData\Projects\AppNotes\XC164CM_LIN\Master_UDE\Master_lin.wsp - Controller0.C166SV2 - C:\...\Master_App\LIN_test.c

C:\...\Master_App\LIN_test.c

UDE - FLASH/OTP Memory Programming Tool

FLASH/OTP - Memory Device: 128 KByte on-chip Program FLASH
Enable: ✓

```
#0 :    0x00C00000  -  0x00C01FFF   (8K)
    0x00C00000 - 0x00C00011
    0x00C0008C - 0x00C0008F
    0x00C00094 - 0x00C0009B
    0x00C000AC - 0x00C000AF
    0x00C00124 - 0x00C01FFF
#1 :    0x00C02000  -  0x00C03FFF   (8K)
    0x00C02000 - 0x00C03FFF
#2 :    0x00C04000  -  0x00C05FFF   (8K)
    0x00C04000 - 0x00C0572F
#3 :    0x00C06000  -  0x00C07FFF   (8K)
#4 :    0x00C08000  -  0x00C0FFFF   (32K)
#5 :    0x00C10000  -  0x00C1FFFF   (64K)
```

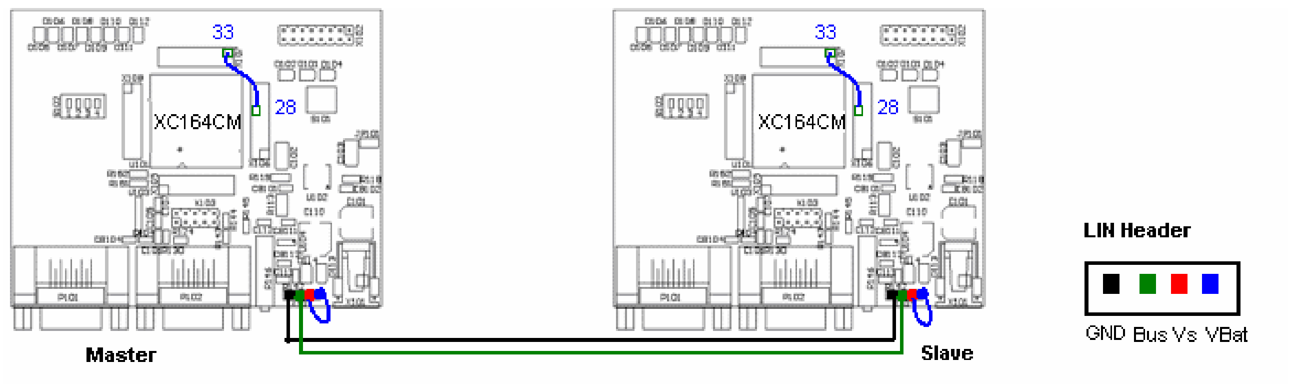
* ... Sector is protected

Slave node – code downloading

Follow the same steps shown above for downloading the slave test application to second XC164CM board. Load the 'Slave_LIN' executable to the slave board.

LIN NETWORK

External connection: the LIN network is designed as one master and one slave, both are using ASC1 and Timer T3 for the LIN. Connect ASC1 RxD1 to T3IN of T3 in both the boards. Connect the LIN bus from Master board to slave and same way the ground. Reset both the boards now the nodes start the LIN communication. This can be monitored in LED by assigning the status of each frame to each P1L port bit value of XC164CM.



6 Conclusion

The LIN configuration tool can be used by the developers to make a prototype LIN applications. This application note explained about features of the tool and how it helps to configure each entity like nodes, signals, frames and schedules in a LIN cluster and automatically generates C-level prototype applications. By introducing this tool, Infineon Technologies now offers all the elements required for developing a LIN network, the powerful 16bit XC16x microcontrollers, LIN Transceivers, LIN LLD and LIN Configuration tool.