AN1099

LIN 2.0 Compliant Driver Using the PIC16FXXX Family Microcontrollers

Authors: *Thorsten Waclawcyzk*
Microchip Technology Inc.

Jin Xu
Microchip Technology Inc.

INTRODUCTION

This application note presents a LIN 2.0-compliant driver for the PIC16XXX family of microcontrollers. This document will focus on the setup and implementation of the driver for a master and/or a slave node(s).

The material in this document is presented with the assumption that the reader is familiar with the previous versions of the LIN specification. Only key additions and modifications of the LIN specification 2.0, the most current specification available to date, will be highlighted.

LIN (Local Interconnect Network) provides a low-cost bus communication for many networks, including automotive and appliance. The LIN protocol provides system development guidelines for the data transmission, the transmission medium, the development tools interfacing and the software programming structures. The key features of the LIN are:

- Low-cost, single-wire implementation
- Single master with multiple slave concept
- Self-synchronization without a resonator in the slave nodes
- Speeds up to 20 Kbits/second
- Signal-based application interaction
- Deterministic signal transmission with computable signal propagation time

LIN 2.0 CHANGES

- Signal groups are replaced with byte arrays that support signal sizes up to eight bytes.
- Automatic bit rate detection is incorporated.
- Unlike the classic checksum that includes only the data bytes, the enhanced checksum is implemented to include the protected identifier and its data bytes.
- Implementation of the sporadic “frame”, which allows the master to update signals that have changed while being transmitted.
- Network management timing is defined in seconds, not in bit times.
- Status management is simplified by standardizing reporting to the network and the application.
- Mandatory node configuration commands are added along with some optional commands.
- Diagnostic API is added and the diagnostic frame is defined.
- LIN product identification for each node is standardized.
- A node capability language specification is added.
- The API is modified to reflect the following changes: byte array, `GOTO` Sleep, wake-up and status reading.
- The API is mandatory for microcontroller-based nodes that are programmed in C.

LIN 2.0 COMPATIBILITY WITH LIN 1.x

The LIN 2.0 specification is a superset of LIN 1.x. Nodes designed to the LIN 2.0 specification and the LIN 1.x specification will communicate with each other with a few exceptions. A LIN 2.0 master node is capable of handling clusters containing both the 1.x and 2.0 slave nodes. The LIN 2.0 master node cannot request the new LIN 2.0 features from a LIN 1.x node:

- Enhanced checksum
- Reconfiguration and diagnostics
- Automatic baud rate detection
- `Response_error` status monitoring

A LIN 2.0 slave node will not work with a LIN 1.x master unless the slave node is reconfigured.

DRIVER RESOURCE USAGE

The driver typically uses 1636 words of program memory and 89 bytes of data memory. The Timer0 interrupt can be used for checking the header and response time-outs. The USART Receive Interrupt is used for receiving and back-to-back transmission bit failure checking. An external interrupt can be used for end of Sync Break check if auto-baud feature is enabled.

PROJECT SETUP

The HI-TECH PICC™ compiler **MUST** be installed prior to using the MPLAB® IDE.

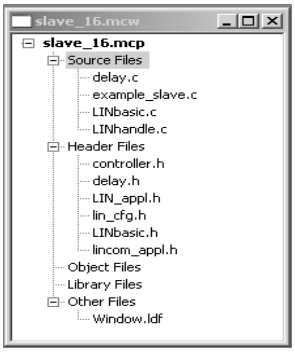
Here are the basic steps required to setup the project:

1. Select *Project>New* to create a new project. Enter the name and directory of the project in the dialog box.
2. Select *Configure>Select Device* to choose the target processor for the project.
3. Select *Project>Select Language Toolsuite* to identify for the MPLAB® IDE which compiler and linker to use. From the dialog box, choose "*PICC Compiler*" as the active tool suite.
4. Add the files to the project. Select the directory and right-click the mouse button to browse for the files. Figure 1 shows the slave final project setup.

The following files will be needed for the project:

- `LINbasic.c` – the driver file, DO NOT MODIFY.
- `LINbasic.h` – the header file for the driver configuration, DO NOT MODIFY.
- `LIN_cfg.h` – the system configuration file.
- `LINhandle.c` – the LIN event handling file.
- `LIN_appl.h` – the node configuration file.
- `Example_slave.c or Example_master.c` – the main file for either a slave node setup or a master setup.

FIGURE 1: PROJECT SETUP

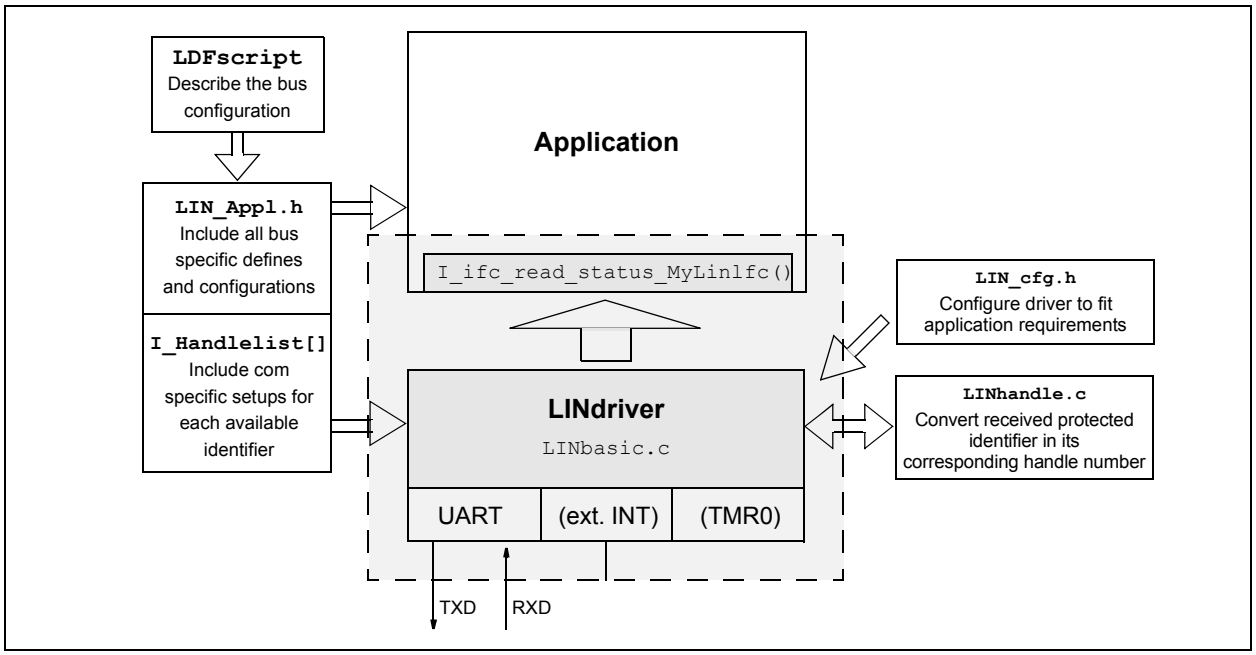


DRIVER SETUP

The driver module is a complete asynchronous self-acting implementation. The user **should not modify** the driver file. Some external files might be needed to configure the driver module based on the network requirements and the application setup.

Figure 2 shows the principal driver setup.

FIGURE 2: DRIVER SETUP



DRIVER FILE DESCRIPTIONS

LINbasic.c

This file is the main driver and **should not be modified**. Functions contained in this file are described further in this section.

Note:	For more details on the LINbasic.c file and its functions, not included in the section, see the LIN API Section [1].

void l_ifc_rx_MyLinIfc(void)

This function is the main driver function call. It handles the complete data exchange including time out, error checking, etc., according to its configurations. The function is interrupt-based for data receptions and transmissions.

void l_ifc_init_MyLinIfc(void)

This function initializes the module by setting up all the variables needed by the driver with default values.

l_u16 l_ifc_read_status_MyLinIfc (void)

This function provides the actual driver status to the application and returns a 16-bit value. See Table 1 for return value of the status.

void l_ifc_wake_up_MyLinIfc(void)

This call transmits a 0xF0 byte (a dominant pulse of 250 μs to 5 ms, depending on the configured bit rate) on the LIN bus to request a wake-up. The wake-up request may be requested by any node in a sleeping cluster.

void l_ifc_goto_sleep_req_MyLinIfc (void)

This call commands all slave nodes in the network to enter Sleep mode by sending a diagnostic master request frame (frame ID = 0x3C) with the first data byte equal to zero. Slave nodes also automatically enter a Sleep mode if the LIN bus is inactive (no transitions between recessive and dominant bit values) for more than 4 seconds.

Reconfiguration Function Calls

The following function calls are used to support LIN 2.0 reconfiguration features using the diagnostic frame IDs, 0x3C and 0x3D.

l_u8 ld_AssignFrameID(l_u8 *l_NAD)

This function is used to set a valid protected identifier to a frame specified by its message identifier. Table 2 shows the structure of the frame.

TABLE 1: RETURN VALUE OF STATUS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Last frame protected identity(1)								0	0	0	0	GOTO Sleep(2)	Overrun(3)	Successful transfer(4)	Error in response(5)

Note 1: The protected identity last detected on the bus and processed in the node.

2: Set if a GOTO Sleep command has been received since the previous call to this function.

3: Set if two or more frames are processed since the last call to this function.

4: Set if one (or multiple) frame response has been processed without any error since the last call of this function.

5: Set if one (or multiple) frame processed by the node had an error in the frame response section since the previous call to this function.

TABLE 2: ASSIGN FRAME ID

NAD	PCI	SID	D1	D2	D3	D4	D5
NAD	0x06	0xb1	Supplier ID LSB	Supplier ID MSB	Message ID LSB	Message ID MSB	Protected ID

Protocol Control Information (PCI)

Protocol Control Information contains the transport layer flow control information. PCI type SF (Single Frame) indicates that the transported message contains a maximum of five data bytes that fit into the single PDU (Packet Data Unit). The value of length is set to the number of used data bytes plus one (for the SID or RSID). Refer to Table 3.

TABLE 3: PCI STRUCTURE

Type	PCI Type				Additional Information			
	B7	B6	B5	B4	B3	B2	B1	B0
SF	0	0	0	0	Length			

Service Identifier (SID)

Service Identifier specifies the tasks that the slave node addressed must perform. 0xB0 to 0xB4 values are used for SID configuration.

A positive response to an assign frame ID request is sent only if the NAD and the supplier ID match. No response is sent for a negative response. The Response Service Identifier (RSID) specifies the contents of the response. The RSID for a positive response is always SID + 0x40. The implementation of a response is optional. See Table 4 for the structure of the positive response.

Note:	For more information, see the LIN Diagnostic and Configuration Section [1].

TABLE 4: POSITIVE ASSIGN FRAME ID RESPONSE

NAD	PCI	RSID	Unused				
NAD	0x01	0xf1*	0xff	0xff	0xff	0xff	0xff

* RSID value is equal to SID + 0x40.

`l_u8 ld_ReadByID()`

This function reads back the supplier identity and other properties from a slave node. Table 5 shows the structure of the frame.

A response is sent only if the address of the slave node (NAD), the supplier ID and the function ID match. Table 6 shows some of the possible positive responses.

If the request fails, then a negative response is sent. Table 7 shows the structure of the negative response.

Note:	For more information, see the LIN Diagnostic and Configuration Section [1].

TABLE 5: READ BY IDENTIFIER

NAD	PCI	SID	D1	D2	D3	D4	D5
NAD	0x06	0xb2	Identifier	Supplier ID LSB	Supplier ID MSB	Function ID LSB	Function ID MSB

TABLE 6: POSITIVE RESPONSES FOR READ BY ID

ID	NAD	PCI	RSID	D1	D2	D3	D4	D5
0	NAD	0x06	0xf2*	Supplier ID LSB	Supplier ID MSB	Function ID LSB	Function ID MSB	Variant
1	NAD	0x05	0xf2*	Serial 0, LSB	Serial 1	Serial 2	Serial 3, MSB	0xff
	Reserved							
16	NAD	0x04	0xf2*	Message ID 1 LSB	Message ID 1 MSB	Protected ID (or FF)	0xff	0xff
17	NAD	0x04	0xf2*	Message ID 2 LSB	Message ID 2 MSB	Protected ID (or FF)	0xff	0xff

* RSID value is equal to SID + 0x40.

TABLE 7: NEGATIVE RESPONSE FOR READ BY ID

NAD	PCI	RSID	D1	D2	Unused		
NAD	0x03	0x7f	Requested SID (= 0xb2)	Error code (= 0x12)	0xff	0xff	0xff

`l_u8 ld_AssignNAD(l_u8 *l_NAD)`

This is an optional function that is used to resolve conflicting node addresses. It should be structured as shown in Table 8.

A response is sent only if the NAD, the supplier ID and function ID match. The implementation of the response is optional. See Table 9 for the structure of a positive response. This request and the response always use the initial NAD to avoid losing the address of the node.

Note:	For more information, see the LIN Diagnostic and Configuration Section [1].

Time-Out Implementation

The time-out feature is user selectable and a dependent of the baud rate chosen in the lin_cfg.h file. If enabled, Timer1 is used to track the calculated maximum response time allowed.

The LIN specification 2.0 described the frame slot time allocation requirements as the following:

The nominal value for transmission of a frame matches the number of bits transmitted excluding any response space, byte spaces or inter-frame space.

$T_{Header\_Nominal} = 34 * T_{Bit}$

$T_{Response\_Nominal} = 10*(N_{Data}+1)*T_{Bit}$

$T_{Frame\_Nominal} = T_{Header\_Nominal} + T_{Response\_Nominal}$

The maximum value between the bytes is an additional 40% of time allowance compared to the nominal value.

$T_{Header\_Maximum} = 1.4 * T_{Header\_Nominal}$

$T_{Response\_Maximum} = 1.4 * T_{Response\_Nominal}$

$T_{Frame\_Maximum} = T_{Header\_Maximum} + T_{Response\_Maximum}$

- T_{Bit} is the time required to transmit one bit

- N_{Data} is the number of data bytes in the frame

When using time out, the timing starts with the max header time calculation that begins after receiving the first 10 T_{Bits} of the Sync Break. After the Sync Break is verified by checking for 0x00 in the receiver buffer and the frame error flag bit, Timer1 is enabled and written with the value of ($T_{Header\_Maximum}$ - 10*T_{Bit}). The header time-out count ends after the identifier has been received or an time-out event.

The identifier provides information about the length of the response. Therefore response time-out value can be calculated by first multiplying the total number of bytes (data plus one byte of checksum) and T_{Bit}, then add the remaining time left that was not used from the header time out.

Void SetupTimeoutIDLE(Void)

This function is the default routine called after each interruptive-based communication event. The function stores all errors, resets the error flags and changes the LIN bus Communication mode to wait for a Sync Break.

Note:	For more information, see the LIN Protocol Specification, Frame Transfer, section 2.2 for more timing related specification [1].

TABLE 8: ASSIGN NAD FRAME

NAD	PCI	SID	D1	D2	D3	D4	D5
Initial NAD	0x06	0xb0	Supplier ID LSB	Supplier ID MSB	Function ID LSB	Function ID MSB	New NAD

TABLE 9: POSITIVE ASSIGN ID RESPONSE

NAD	PCI	RSID	Unused				
Initial NAD	0x01	0xf0	0xff	0xff	0xff	0xff	0xff

`LINbasic.h`

This is the header file for the LIN driver. This file defines the variables and values used by the driver. It will cause a compiler error if the configuration settings are not correct. Users **should not modify** this file.

`LIN_cfg.h`

This header file is used to configure the driver for the applications. The following items are defined in this file:

- System clock
- Nominal baud rate
- LIN version
- TX/RX/CS pins
- External interrupt, if needed for auto-baud detection
- Timer1, if time-out handling is enabled

The LIN product identification (Supplier ID, Function ID and Variant) and the serial number are required to be defined in this file. The LIN consortium assigns a unique supplier ID. The list of IDs can be found at **http://www.lin-subbus.org**.

LINhandle.c

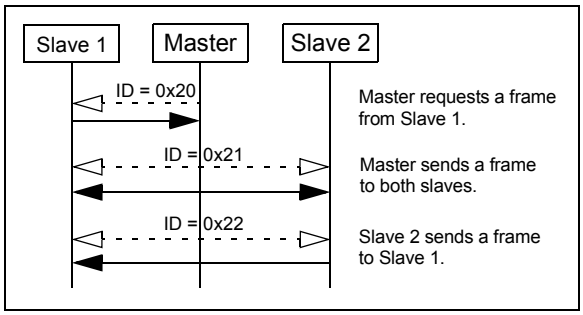
l_u8 l_ifc_pid_to_handle(void)

This function matches the received PID to the list of handles (user defined) and sets up the driver for the frame. The configuration for each handle is stored in the variable `l_HandleList[]` which is defined in the file `LIN_appl.h`. If no match is found for a PID, then a zero is returned to reset the driver and wait for the next Sync Break while the rest of the response is ignored. This function name should not change since it is called from the driver.

Three types of response frames are supported by this function:

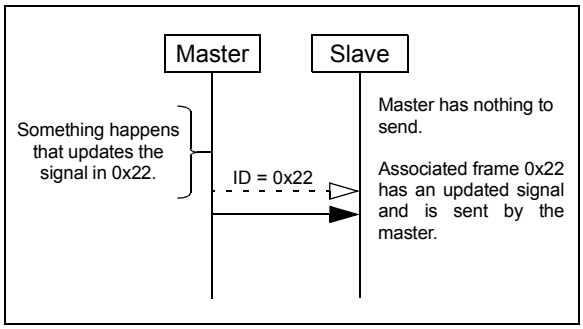
1. Unconditional Frames: Standard frames that answer to any identifier that matches. These frames always carry signals and the identifiers are in the range of 0-59 (0x3B). Figure 3 shows a sequence of three unconditional frames.

FIGURE 3: UNCONDITIONAL FRAME



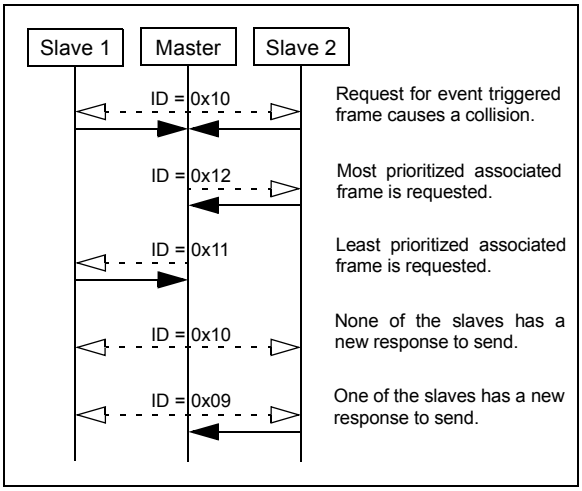
2. Sporadic frames: Frames sent by the master node to update the information in an associated frame sent immediately before. If multiple sporadic frames are associated with the same frame slot, then the most prioritized of the sporadic frames will be transferred in the same slot. If no data has been updated, then the slot is left empty. Figure 4 shows an example of a sporadic frame.

FIGURE 4: SPORADIC FRAME



3. Event-triggered Frame: This type of frame allows the master to poll multiple slave nodes in the LIN cluster without assigning too much of the bus bandwidth. The master will send out the header to all slaves, but the slave nodes will answer only if the data in the related frames has changed and the identifiers matched. If more than one slave node responds to the header during the same frame slot, a collision will occur. The master will request the associated frames one at a time based on priorities after a collision. The value of the first data byte of the frame is always the same as its protected identifier (PID). Figure 5 is an example of an event trigger frame sequence.

FIGURE 5: EVENT TRIGGERED FRAME



LIN_appl.h

This file is an extract of the LIN Description File (LDF) that represents the node configuration for both in and out of the network.

The file is structured as follows:

- Frame Definition
- Frame Structure Definition
- Frame Union Definition
- Reserved Buffer Size For Data Exchange
- Alias and API Calls For Structured Frames Definition
- Setup for Node Relevant Handle List
- Additional Items for the Master Node

Frame Definition

`#define k<Frame_Name>_handle` value

Defines the handle value. Value is a user-defined incremental integer.

`#define k<Frame_Name>_id` value

Defines the frame identifier. Value is in the range of 0 to 63(0x3F)

`#define k<Frame_Name>_pid` value

Defines the protected frame identifier. See Appendix 7.2, located in "*LIN Specification Package Revision 2.0*" [1].

`#define k<Frame_Name>_len` value

Defines the response frame length without the checksum.

`#define k<Frame_Name>_mode` value

Defines one of the two possible states seen by the slave node: RECEIVE or RESPOND.

`_LINAC_l_u8<Frame_Name>_pid`

Reserve memory location if frame is defined as reconfigurable. For more information, see the Configuration Language Section [1].

EXAMPLE 1: FRAME DEFINITIONS

```
#define kWindow_handle1
#define kWindow_id5
#define kWindow_pid133
#define kWindow_len2
#define kWindow_modeRECEIVE
EXTERN bank1 l_u8 Window_pid;
```

Frame Structure Definition

`typedef struct { }_c_<Frame_Name>_msgType`

This file declares the position and size of each signal in the frame. The value could be a single bit (`l_bool`), a char (`l_u8`), an integer (`l_u16`) or a byte array.

EXAMPLE 2: typedef struct

```
typedef struct
{
    l_bool  UP : 1;
    l_bool  DOWN : 1;
    l_u8  :6;
} _c_Window_msgType;
```

Frame Union Definition

`typedef union {l_u8_c[size];.....}_c_<Frame_Name>_msgBuf`

Declare a single memory location that is shared by two or more variables of the same or different types and sizes.

EXAMPLE 3: FRAME UNION

```
typedef union
{
     l_u8  _c[kWindow_len];
     _c_Window_msgType window;
} _c_Window_msgBuf;
```

Reserved Buffer Size For Data Exchange

`union _c_<Buffer_Name>_msgBuf <Buffer_Name>`

EXAMPLE 4: RESERVED BUFFER SIZE FOR DATA EXCHANGE

```
EXTERN union
{
  _c_MasterReq_msgBuf MasterReq;
  _c_SlaveResp_msgBuf SlaveResp;
}_u_Buffer;
```

Alias and API Calls For Structured Frames Definition

Alias allows the user to assign any name to a call or command.

EXAMPLE 5: ASSIGN NAME (ALIAS)

```
#define UP Window.window.UP
#define DOWN Window.window.DOWN
#define ButtonInfo Window._c[0]
```

API call assigns a command to a call that conforms to the API format.

EXAMPLE 6: API CALL FOR API FORMAT

```
#define l_bool_rd_UP()    (UP)
#define l_bool_wr_UP(a)   UP = a
#define l_bool_rd_DOWN()   (DOWN)
#define l_bool_wr_DOWN(a) DOWN = a
```

Setup for Node Relevant Handle List

This array defines how a response frame should respond to a specific protected identifier in a node. There are two types of structures for this array (see Examples 7 and 8), depending on the LIN specification revision.

EXAMPLE 7: LIN SPECIFICATION 2.0

```
typedef struct
    {
        l_u8 l_target_mode;
        l_u8 * l_target_addr;
        l_u8 l_target_len;
        l_u16 l_target_MID;
        l_u8 *l_id_addr;
    }l_table_s;
```

EXAMPLE 8: LIN SPECIFICATION 1.X

```
typedef struct
    {
        l_u8 l_target_mode;
        l_u8 * l_target_addr;
        l_u8 l_target_len;
    }l_table_s;
```

The code in the examples above are further defined below:

`l_target_mode`: The direction of data flow, RECEIVE or RESPOND.

`l_target_addr`: Pointer to reserved memory location for frame data.

`l_target_len`: Length of response excluding the checksum.

`l_target_MID`: Message ID needed for reconfiguration using the `AssignFrameID()`. If no MID available, set to 0xFFFF.

`l_id_addr`: Pointer to reserved memory location of reconfigurable frame. If frame is not configurable, set pointer to 0xFFFF.

The array contains at least one constant setup so if function `l_ifc_pid_to_handle()` in `LINhandle.c` returns with '`0`', the driver is reconfigured to wait for the next Sync Break.

Master Node Setup

Note:	The PIC16FXXX family device is not typically used as the master node. Please refer to Microchip application note, AN1009 for the master node setup.

REFERENCES

[1] "*LIN Specification Package Revision 2.0*", LIN Consortium, http://www.lin-subbus.org.

[2] "*AN1009, LIN 2.0 Compliant Driver Using the PIC18XXXX Family Microcontrollers*".

[3] "AN944, Using the EUSART on the PIC16F688".

CONFORMANCE TESTING

This driver has passed the LIN 2.0 Conformance testing performed by IHR.

APPENDIX: SOURCE CODE

The generic source code files and the slave example driver project file can be downloaded from **www.microchip.com**.

Slave example contains the following files:

- `example_slave.c`
- `LINbasic.c`
- `LINhandle.c`
- `delay.c`
- `LINbasic.h`
- `LIN_appl.h`
- `delay.h`
- `lin_cfg.h`
- `controller.h`
- `lincom_appl.h`

NOTES:

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Trademarks

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2002

Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://support.microchip.com
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Fuzhou
Tel: 86-591-8750-3506
Fax: 86-591-8750-3521

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Shunde
Tel: 86-757-2839-5507
Fax: 86-757-2839-5571

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7250
Fax: 86-29-8833-7256

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-4182-8400
Fax: 91-80-4182-8422

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Gumi
Tel: 82-54-473-4301
Fax: 82-54-473-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Penang
Tel: 60-4-646-8870
Fax: 60-4-646-5086

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-572-9526
Fax: 886-3-572-6459

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

12/08/06