*Application Note*

*AN2432/D*
*Rev. 0, 12/2002*

# *LIN Sample Application for the MC68HC908EY16 Evaluation Board*

**by Peter Topping,**
**Applications Engineering,**
**Freescale, East Kilbride**



## Introduction

The MC68HC908EY16 LIN evaluation board was designed to facilitate the development of LIN (Local Interconnect Network, reference [1]) slave nodes and includes features to ease system development and debug. These features include access to most of the pins on the MC68HC908EY16 MCU, an RS232 ’08 monitor interface and a connector to allow the use of a P&E Cyclone or Multilink interface. Any of these interfaces can be used in conjunction with Metrowerks Codewarrior to constitute a low cost development system allowing the development of MC68HC908EY16 applications including FLASH programming and software debugging. While any kind of application is possible, the board is primarily intended for LIN applications and includes an MC33399 LIN interface chip (reference [4]). The CD accompanying the board includes LIN software drivers which handle the LIN protocol using the EY16’s SCI pins. This allows very easy development of LIN slave nodes.

This application note presents a simple LIN monitor using the MC68HC908EY16 LIN evaluation board. AN2343 (reference [8]) includes a more sophisticated LIN monitor that displays data using an LCD module external to the PCB. The monitor described here does not require any hardware in addition to the evaluation board. The display capability, using only the 5 onboard LEDs, is consequently more limited.

The LIN bus is a low-cost single-wire serial bus ideally suited for use in many industrial and automotive applications. A typical application is in a car door. While a higher bandwidth CAN (Controller Area Network) bus is often used to transfer data to and from the door, within the door a LIN bus with its limited data rate of 20,000 baud is adequate to distribute the information to and from the keypad, mirror, window and lock modules. Each LIN node requires only 3 wires; there is one LIN data line, the other two connections being the positive and negative supplies.

The software for this LIN monitor is listed in appendix I. It is simplified by the use of software drivers (reference [2]) that deal with the serial LIN protocol. These drivers use the MC68HC908EY16's SCI module to interface with the LIN physical interface (MC33399) and allow the application code to use simple Application Program Interface (API) function calls to read status information and data from the LIN messages. This application note, as well as describing the LED LIN monitor, outlines how the drivers can be used to develop other LIN applications.

## Use of the evaluation board without additional hardware

The evaluation board is supplied with the LEDemo application programmed into FLASH memory. It will function as described in this application note by simply applying 8-12 volts to the battery (B1) or LIN (P2) connector. This assumes that three jumpers are fitted: J2, J1 (IRQ-IRQ HIGH) and P4 (pin11-pin13).

The main use of the board is, however, to facilitate the development of LIN applications and for this a programming and development environment needs to be added. The most sophisticated high level language programming and development environment is Metrowerks Codewarrior. A limited version of Codewarrior is available on the WEB (www.metrowerks.com) and also on the CD supplied with the evaluation board. This version is limited to 1K of code but a free 4K license upgrade is available from license_europe@metrowerks.com. Downloading and installation may require full "admin" access to the PC. Accept the suggested paths and defaults (e.g. do not select FlexLM option). For FLASH programming and assembly code development P&E's PROG08SZ and ICS software is suitable. These are available free from www.pemicro.com.

There are 3 main ways of connecting the evaluation board to the development environment.

The simplest in hardware terms is to use the board in monitor mode using an RS232 connection to a PC "COM" port. All that is required is an 8-12 volt supply (at B1 or P2) and a 9-pin straight-through "D" cable to a PC running appropriate software. Use of the P&E tools is very straightforward. In order to get into monitor mode at 9600 baud all that is required is the installation of the 9.8304MHz canned oscillator module and the addition of a link between pins 2 (IRQ) and 3 (VTST) of J1. This provides 9 volts to the IRQ pin. No other links should be installed. The board supplies pull-ups and pull-downs to provide the appropriate levels on pins A1, B3 and B4 for entry into monitor mode.

To develop LIN applications in “C”, Metrowerk’s Codewarrior is an appropriate development environment. To use Codewarrior with this simple hardware set-up, use the following procedure:

1. Copy folder “Lin08EY” from CD to an appropriate place on local hard drive.
2. Launch Codewarrior from Start menu: Programs - Metrowerks - CodeWarrior CW08_V2.1 - CodeWarrior IDE (Copy/paste a shortcut for future use).
3. From folder “Lin08EY” drag file: …\sample\ide\EYLEDemo\LEDemo.mcp into Codewarrior window.
4. Switch on the board and run the debugger by clicking on the green arrow. Error messages relating to the limitations of the Codewarrior license may appear, they will not reoccur as long as the rest of this procedure is followed and the debugger is kept open.. If any compilation or linking is required it will happen before the debugger starts up.
5. In “Component” pull-down menu, select “Set target” and then “HC08” and “P&E Target Interface”. Press OK.
6. If prompted by a window headed “Attempting to contact target and pass security”, select “Target Hardware Type” class III and click on “contact target with these settings”. This window may also prompt for any monitor mode security issues.
7. Check “PEDebug” pull-down menu for “device” (EY16) and “mode” (In-circuit debug/programming) and, if required, change to these settings.
8. In “PEDebug” pull-down menu, select “load” and file “slave.abs” (in “bin” directory). This will overwrite the code in FLASH with that contained in “slave.abs” (select yes when prompted for permission to overwrite FLASH).
9. Run program by clicking on green “GO” arrow.
10. Code should now run. If the source code has not been changed then the LED LIN monitor described in this application note will now be running and LED D4 flashing at 1HZ. To obtain full functionality, a LIN master (e.g. the VCT (Volcano Communications Technologies) LINspector) should be added.
11. To develop new LIN applications the source code file LEDemo.c should be removed from the project and replaced with the new application source file(s). This, and a procedure for creating a completely new Codewarrior project, are described in detail in appendix I. Creation of a new project allows development of a different application, with its own LIN characteristics (message IDs etc.), without interfering with the supplied demonstration application.

### Use of the evaluation board with Cyclone, Multilink and MMDS/MMEVS

A second method of using the monitor mode development environment is to connect via the 16-pin P4 connector. This is configured for a Cyclone or Multilink interface. Operation is very similar to that described above, the advantage being that the users PCB does not need to incorporate an RS232 interface or a 9.8304MHz clock source. The high voltage for IRQ and the appropriate levels on pins A1, B3 and B4 are also supplied by the interface so no monitor mode hardware is required on the target hardware as long as these pins are allowed to go to the required levels. Link J1 must be connected between pin 1 (IRQ HIGH) and pin 2 (IRQ). All other links and the 9.8304MHz canned oscillator module should be removed. The following procedure should be used.

1. Copy folder “Lin08EY” from CD to an appropriate place on local hard drive.
2. Launch Codewarrior (Programs - Metrowerks - CodeWarrior CW08_V2.1 – CodeWarrior IDE).
3. From folder “Lin08EY” drag file: …\sample\ide\EYLEDemo\LEDemo.mcp into Codewarrior window.
4. Connected to the Cyclone via its 16-way cable (pin1, marked red, to the corner of the evaluation board) and switch the board on. Run the debugger by clicking on either green arrow. Error messages relating to the limitations of the Codewarrior license may appear, they will not reoccur as long as the rest of this procedure is followed and the debugger is kept open. If any compilation or linking is required it will happen before the debugger starts up.
5. In “Component” pull-down menu, select “Set target” and then “HC08” and “P&E Target Interface”. Press OK.
6. If prompted by a window headed “Attempting to contact target and pass security”, select “Target Hardware Type” class V and click on “contact target with these settings”. This window may also prompt for any monitor mode security issues.
7. Check “PEDebug” pull-down menu for “device” (EY16) and “mode” (In-circuit debug/programming) and, if required, change to these settings.
8. In “PEDebug” pull-down menu, select “load” and file “slave.abs” (in “bin” directory). This will overwrite the code in FLASH with that contained in “slave.abs” (select yes when prompted for permission to overwrite FLASH).
9. Run program by clicking on green “GO” arrow.
10. Code should now run. If the source code has not been changed then the LED LIN monitor described in this application note will now be running and LED D4 flashing at 1HZ. To obtain full functionality, a LIN master

(e.g. the VCT (Volcano Communications Technologies) LINspector) should be added.

11. To develop new LIN applications the source code file LEDemo.c should be removed from the project and replaced with the new application source file(s). This, and a procedure for creating a completely new Codewarrior project, are described in detail in appendix I. Creation of a new project allows development of a different application, with its own LIN characteristics (message IDs etc.), without interfering with the supplied demonstration application.

The third, and most sophisticated, development possibility is to use an MMDS or MMEVS with an MC68HC908EY16 EM module (M68EML08EY). To use this option, the MCU should be removed from the board and replaced with the appropriate target header and cable to interface with the MMDS/MMEVS. Codewarrior is also suitable for use with this hardware set-up.

## LIN evaluation board hardware

The schematic of the MC68HC908EY16 LIN evaluation board is shown in figure 1. Apart from the MCU itself, two chips are required to facilitate a simple LIN node. These are the LIN interface, in this case the MC33399 and a 5 volt regulator. The regulator used on the PCB is a 3-pin 7805. The board also incorporates an RS232 serial port using an LT1181 interface chip. The port, with a standard 9-way “D” connector, allows easy connection to the serial port of a PC to facilitate FLASH programming and software debugging using Metrowerks (e.g. Codewarrior) or P&E (e.g. ICS and PROG08) utilities. If the MCU is de-soldered, there is sufficient space around its footprint to fit a target header for the MMDS/MMEVS development system if this is preferred.

The MC33399 includes an internal 30kohm LIN pull-up so this does not need to be fitted on the PCB if it is being used as a slave. The capability to add a 1kohm master pull-up is, however, incorporated. The LIN bus is accessible via a 3-pin header that includes $V_{BAT}$ (12 volts) and ground connections. This 12 volt connection and the optional 1kohm pull-up are reverse polarity protected.

Although the MC68HC908EY16 has an on-chip oscillator, an 8MHz crystal is included on the board. This can be used for applications which need the accuracy of a crystal and also allows easier development of those which will eventually use the on-chip oscillator. The intention is that an application can be fully written and debugged using an accurate clock prior to modifying it to incorporate any oscillator trimming of the internal clock that may be required. This will always be necessary for LIN applications which use the internal oscillator. Use of the crystal requires removal of the 9.8304MHz oscillator module and the addition of a link at J2 and between pins 11 and 13 of the Cyclone/Multilink connector (P4).

The board also incorporates a 9.8304MHZ canned oscillator module which is required to run monitor mode using the RS232 PC interface. This module is fitted using a socket so that it does not need to be fitted if it is not required. Cyclone and Multilink interfaces incorporate their own clock source and thus do not require the oscillator module.

There are pull-up resistors on the IRQ and Reset pins. Reset also has a pull-down button and IRQ has a link that allows the connection of a zener stabilised 9 volts supply to enable entry into monitor mode which facilitates debugging and in-circuit programming of the on-chip FLASH memory.

The board includes 5 LEDs driven directly by MCU port pins. These LEDs are used in the monitor code to show that the application is running and to provide some information on LIN bus activity.

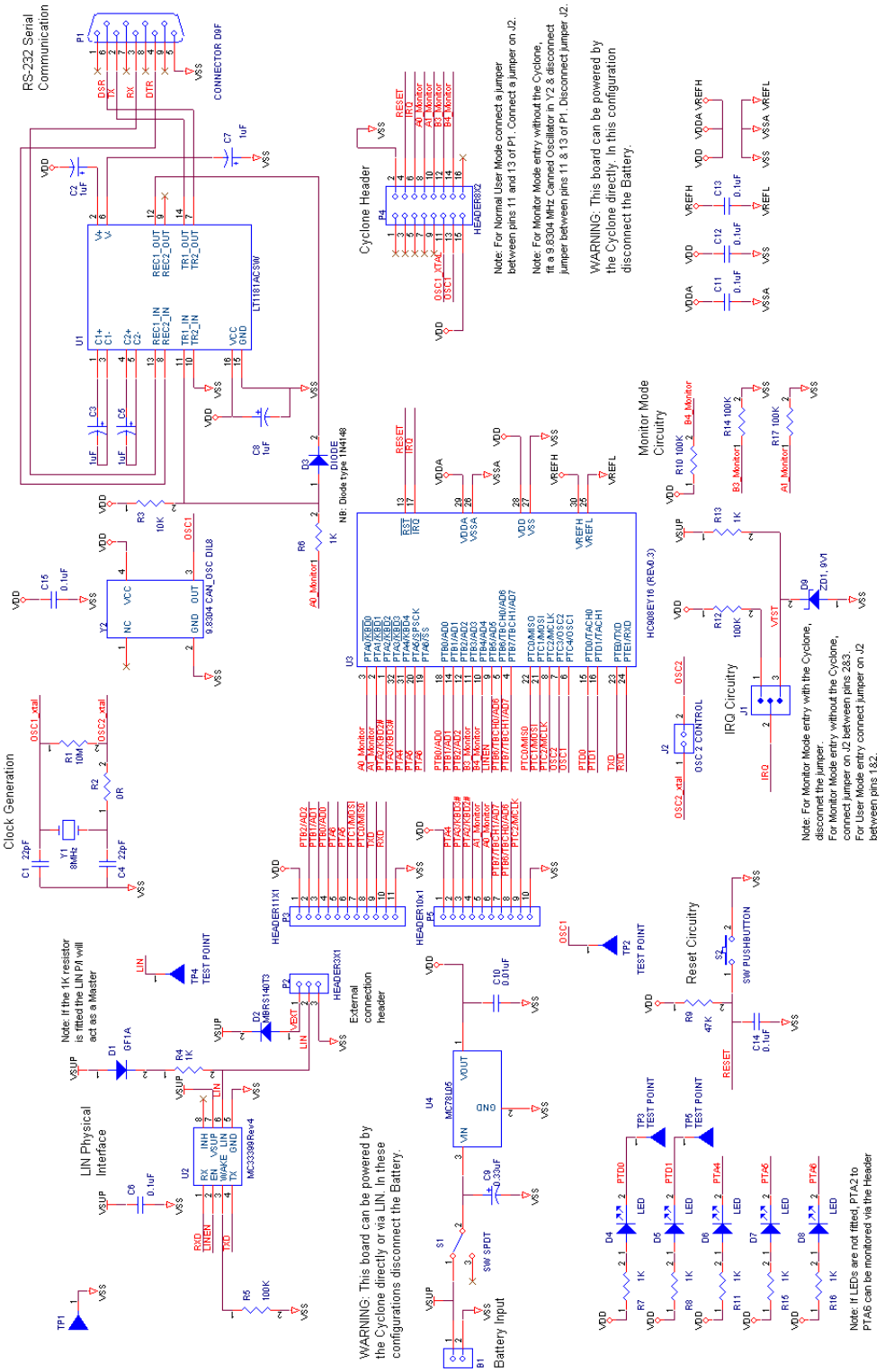

**Figure 1. MC68HC908EY16 evaluation board circuit diagram**

## Software

The LIN LED monitor uses the Freescale/Metrowerks LIN drivers to handle the LIN I/O protocol. Access to the drivers is via the API described in the LIN08 Driver User's Manual (reference [2]). This manual is included on the CD accompanying the evaluation board. An example of the use of this API is the function call "LIN_GetMsg(0x20, Data_buf)". This is all that is required to get the data from the LIN message with an ID of $20 and put it into the array Data_buf. This use of the LIN drivers results in the very simple application software listed in section 7.

In order to respond to a master request command frame (ID 0x3C), the user code has to include the function "LIN_Command()". This is, for instance, how the master would request all slave nodes to go into their low-power standby or "sleep" mode. In this application, sleep mode is entered when there is no bus activity and this function is just a dummy while(1).

The main software flow chart is shown in figure 2. Once the variables have been declared, the CONFIG and I/O registers are initialised. The CONFIG1 value of 0x01 disables the COP while the CONFIG2 value of 0x29 configures the MCU to use the crystal oscillator. In this configuration, the external canned oscillator can also be used. At 9.8304MHz, it is appropriate for monitor mode entry and is also the same frequency as the Cyclone/Multilink clock. By initially using this frequency to develop the application, no software changes are required between software debug and stand-alone running of the LIN node. If a different target frequency is required then care should be taken when the switch is made. The file slave.cfg (appendix IIc) shows the required settings to configure the LIN drivers for a 9600 baud rate using 9.8304MHz and 8.0 MHz oscillator frequencies.

Interrupts are enabled so that the LIN drivers, once initialised by "LIN_init()", can function properly. The main while loop uses the Time Base Module (TBM) to facilitate a 244Hz repetition rate based on an 8MHz crystal. Once every 4.1ms the TBM overflow flag is set and the main loop is executed. The flag is cleared and a counter used to flash an LED (D4) at 244/256 Hz to indicate that the board is running correctly

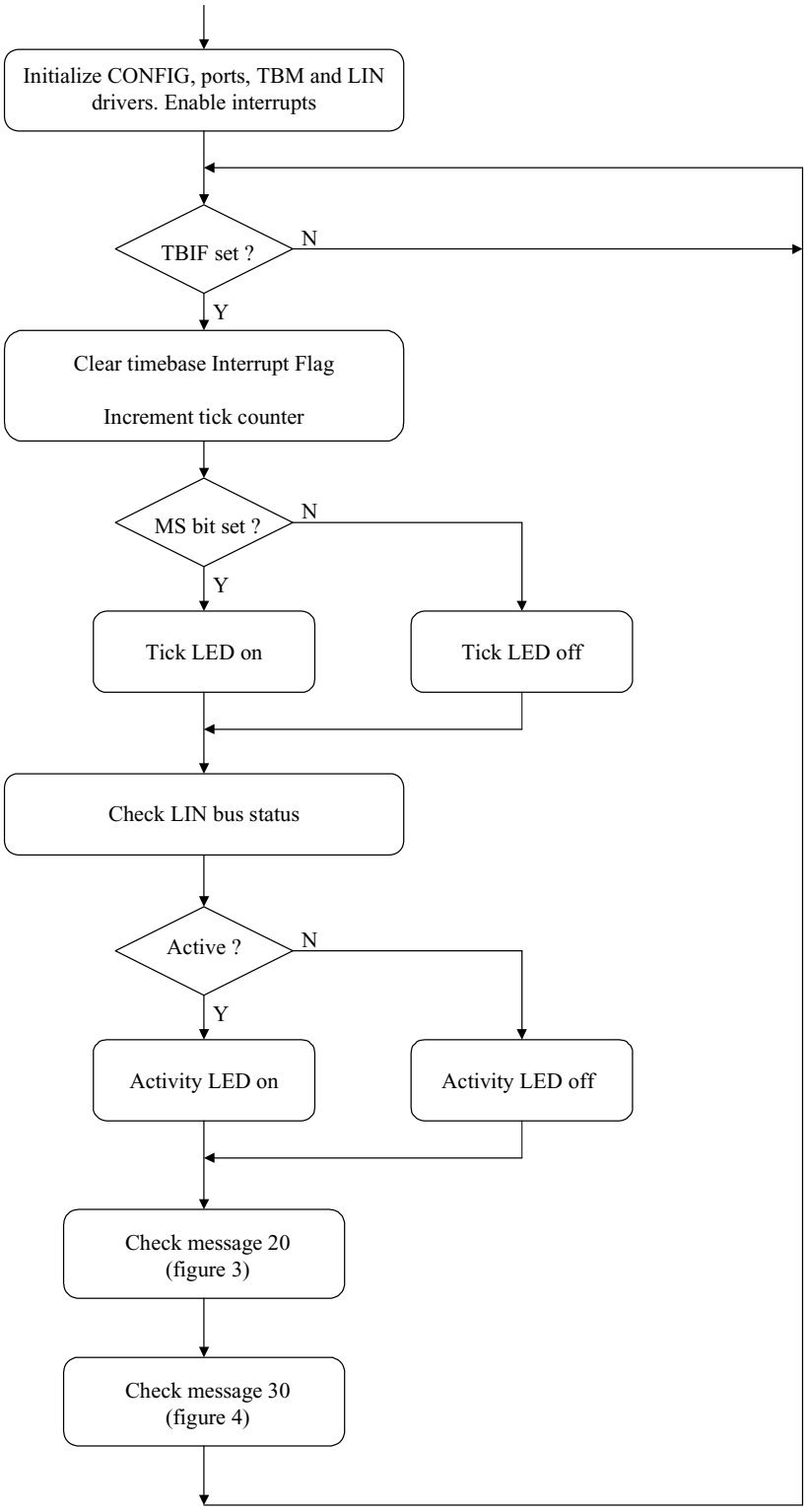


**Figure 2. Main Software Flow Chart**

The LIN driver functions “LIN_IdleClock()” and “LIN_DriverStatus()” are then used to determine whether or not there is any activity on the LIN bus. “LIN_IdleClock()” checks whether or not there is any bus activity. If not, it increments a counter whose value is compared with LIN_IDLETIMEOUT. LIN_IDLETIMEOUT is defined as 500 in slave.cfh. If this number is exceeded, the function “LIN_DriverStatus()” ceases to return a 1 (LIN_STATUS_RUN) indicating that the bus has been idle for, in this case, about 2 seconds. If a master node is connected to provide LIN activity then the LED, D5, will indicate its presence and that the MC33399 LIN physical interface is functioning correctly[1].

“LIN_MsgStatus(0x20)” is then used to check for the presence of a message with an ID of 20. The flow chart is shown in figure 3. In this case the timeout is determined by the application software and not by the LIN drivers. The counter value chosen (250) means that the LED will go out if there is no message with this ID for a period of about a second. An ID of 20 was used for the keypad in AN2205 (Car door keypad using LIN, reference [6]). If this message is present then the D7 LED is illuminated. If present, the message is read and D8 indicates the state of the most significant bit in the 4$^{th}$ byte. This is the bit used for the child-lock switch in the keypad application. D6 is similarly used to indicate the presence of a second message with an ID of 30 (figure 4). Clearly this is an arbitrary choice of message IDs and only a single bit of data can be displayed but it does indicate the full functionality of the LIN bus. For example, if the wrong baud rate is used, the bus activity LED will still come on but LEDs D6, D7 and D8 will not.

1. In an application in which the regulator can be controlled (e.g. by using an LT1121 instead of an MC78L05), the enable pin of the MC33399 could be taken low to put the node into sleep mode by switching off the regulator and powering down the MCU. The code to do this is included in the listing as a comment.

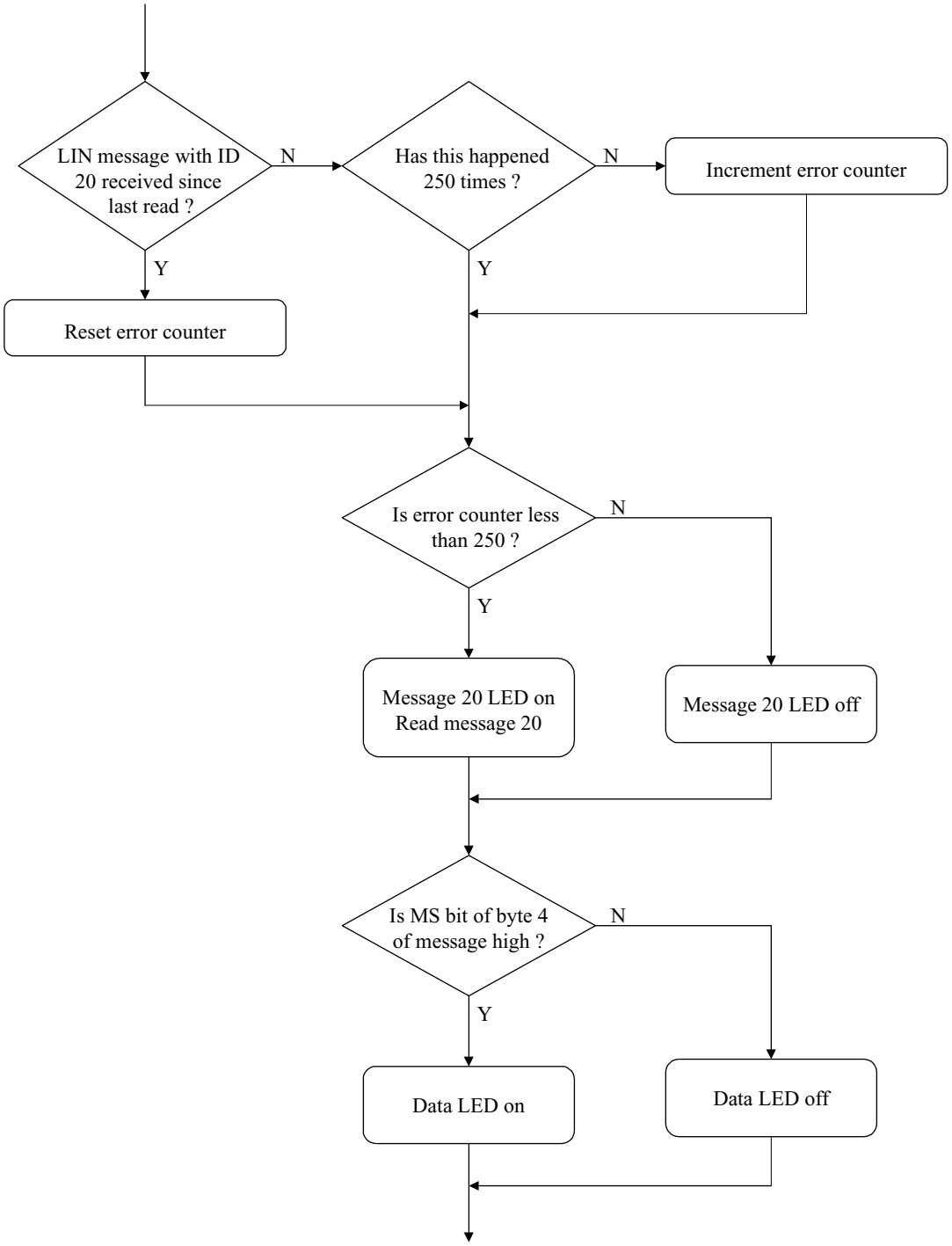


**Figure 3. Message 20 Flow Chart**

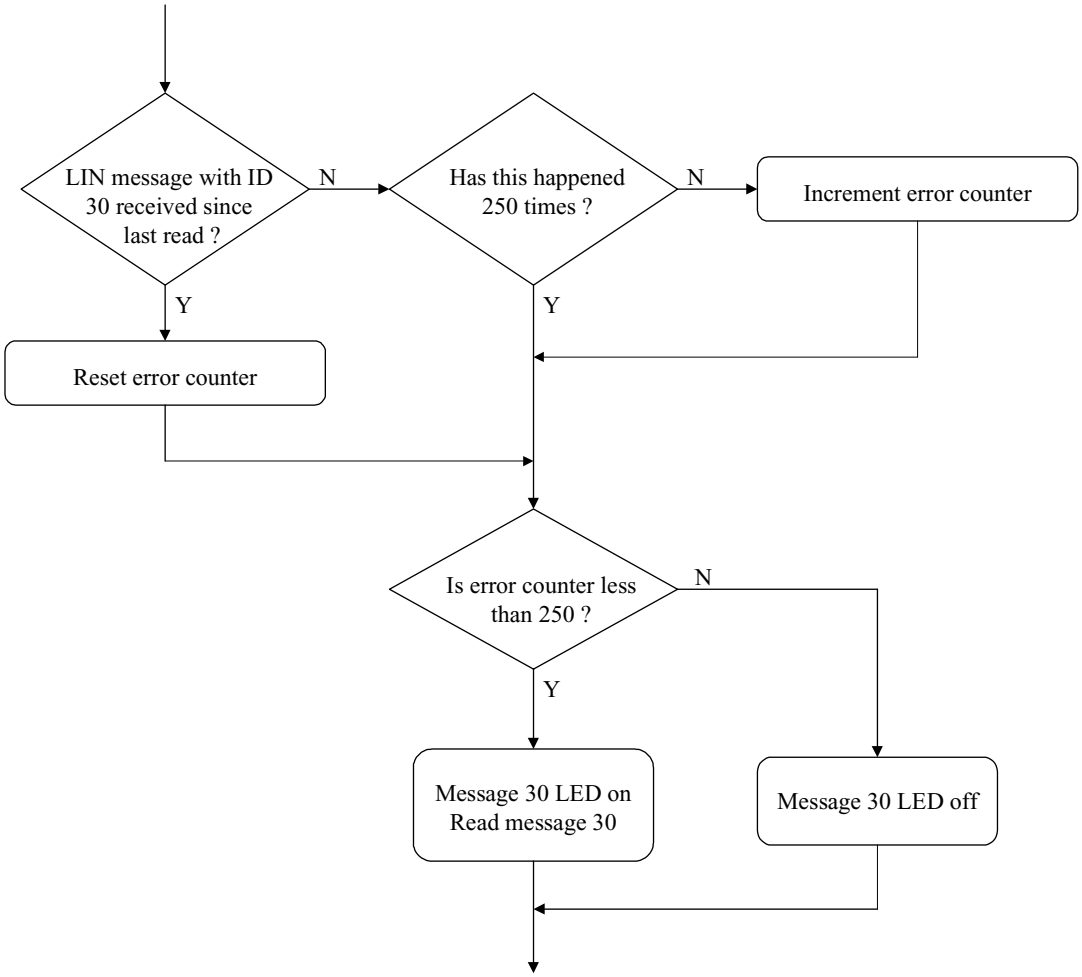


**Figure 4. Message 30 Flow Chart**

## References

1. LIN Protocol Specification, Version 1.2, 17 November 2000.
2. LIN08 Driver User's Manual rev 1.1, 13 March 2001.
3. MC68HC908EY16A Advance Information.
4. MC33399 data sheet.
5. AN2203, LIN demo, 2000.
6. AN2205, Car door keypad using LIN, November 2001.
7. AN2264, LIN node temperature display, June 2002.
8. AN2343, HC908EY16 LIN Monitor, September, 2002.

## Software listing

```
/*****************************************************************************
*                        Copyright (c)  2002
*
*
*                            908EY16 LED LIN demo.
*                            =====================
*
*    Originator:    P. Topping
*    Date:          5th December 2002
*    Revision:      1.0
*    Function:      Demonstration of LIN functionality using only the five
*                   LEDs on the EY16 LIN evaluation board.
*
*    LED use:       D4 (port D, bit 0): flashes @ ~1Hz to show code is running
*                   D5 (port D, bit 1): indicates any LIN bus activity
*                   D6 (port A, bit 4): shows presence of ID 30 message
*                   D7 (port A, bit 5): shows presence of ID 20 message
*                   D8 (port A, bit 6): indicates bit 7 of byte 3 of message 20
*
*****************************************************************************/

/*****************************************************************************

 Freescale reserves the right to make changes without further notice to any
product herein to improve reliability, function or design.  Freescale does not
assume any liability arising out of the application or use of any product,
circuit, or software described herein; neither does it convey any license
under its patent rights nor the rights of others.  Freescale products are not
designed, intended, or authorized for use as components in systems intended
for surgical implant into the body, or other applications intended to support
life, or for any other application in which the failure of the  Freescale product
could create a situation where personal injury or death may occur. Should
Buyer purchase or use  Freescale products for any such unintended or
unauthorized application, Buyer shall indemnify and hold  Freescale and its
officers, employees, subsidiaries, affiliates, and distributors harmless
against all claims costs, damages, and expenses, and reasonable attorney fees
arising out of, directly or indirectly, any claim of personal injury or death
associated with such unintended or unauthorized use, even if such claim
alleges that  Freescale was negligent regarding the design or manufacture of the
part.  Freescale and the  Freescale logo are registered trademarks of  Freescale Ltd.

*****************************************************************************/


/*****************************************************************************
*
*    Header file includes and globals
*
*****************************************************************************/

#include "HC08EY16.h"
#include <linapi.h>

unsigned char Data_buf[8];
unsigned char count = 0;
unsigned char no_20 = 0;
```

```
unsigned char no_30 = 0;



/*****************************************************************************
*
*    Function name: Main
*
*****************************************************************************/


void main (void)
{

CONFIG1 = 0x01;                                 /* disable COP             */
CONFIG2 = 0x29;                                 /* ext.clk, fast timebase  */

PTB     = 0x20;                                 /* MC33399 enable high     */

DDRA    = 0x7C;                                 /* A4, 5 & 6 out for LEDs  */
DDRB    = 0xE7;                                 /* B5 out for 33399 enable */
DDRC    = 0x83;                                 /* enable MCLK (C2)        */
DDRD    = 0x03;                                 /* D0 & D1 out for LEDs    */

while (ICGCR != 0x13)                           /* switch to ext. clock    */
{ ICGCR = 0x12; }                               /* and wait for switch     */

TBCR = 0x00;                                    /* divide by 32768 (244Hz  */
TBCR = 0x02;                                    /* @ 8MHz) & switch TBM on */

asm CLI;                                        /* enable interrupts       */

LIN_Init();                                     /* initialise LIN drivers  */

while (1)
{
    if (TBCR & 0x80)                            /* is TBM flag set?        */
    {
        TBCR |= 0x08;                           /* yes, clear it           */

        count ++;                               /* increment tick counter  */

        if (count & 0x80)                       /* check MS bit of count   */
        {
            PTD |= 0x01;                        /* tick LED off            */
        }
        else
        {
            PTD &= ~(0x01);                     /* tick LED on             */
        }


        LIN_IdleClock ();                       /* check for bus activity  */
        if (LIN_DriverStatus() & LIN_STATUS_IDLE) /* bus idle for 500 tries? */
        {
            PTD |= 0x02;                        /* yes, activity LED off   */
         /* PTB |= 0x20; */                     /* MC33399 enable low      */
        }
        else
        {
            PTD &= ~(0x02);                     /* no, bus activity LED on */
```

```
/*****************************************************************************
*
*    ID 20
*
*****************************************************************************/

            if (LIN_MsgStatus (0x20) != LIN_OK)   /* new ID20 message ?      */
            {
                if (no_20 < 251)                  /* no, already over 250 ?  */
                {
                    no_20 ++;                     /* no, increment           */
                }
            }
            else
            {
                no_20 = 0;                        /* yes, new 20 received    */
            }

            if (no_20 > 250)                      /* any 20s in last second? */
            {
                PTA |= 0x60;                      /* no, ID20 LEDs off       */
            }
            else
            {
                PTA &= ~(0x20);                   /* yes, ID20 LED on        */

                LIN_GetMsg (0x20, Data_buf);      /* and get data            */

                if (Data_buf[3] & 0x80)           /* check MS bit, last byte */
                {
                    PTA &= ~(0x40);               /* bit high, LED on        */
                }
                else
                {
                    PTA |= 0x40;                  /* bit low, LED off        */
                }
            }


/*****************************************************************************
*
*    ID 30
*
*****************************************************************************/

            if (LIN_MsgStatus (0x30) != LIN_OK)   /* new ID30 message ?      */
            {
                if (no_30 < 251)                  /* no, already over 250 ?  */
                {
                    no_30 ++;                     /* no, increment           */
                }
            }
            else
            {
                no_30 = 0;                        /* yes, new 30 received    */
            }
```

```
            if (no_30 > 250)                    /* data in last second?    */
            {
                PTA |= 0x10;                    /* no, ID30 LEDs off       */
            }
            else
            {
                PTA &= ~(0x10);                 /* yes, ID30 LED on        */

                LIN_GetMsg (0x30, Data_buf);    /* read, data not used     */
            }
        }
    }
}
}


/*******************************************************************************
* Function:        LIN_Command
*
* Description:     User call-back. Called by the driver after transmission or
*                  reception of the Master Request Command Frame (ID: 0x3C).
*
*******************************************************************************/

void LIN_Command()
{
    while(1)
    {
    }
}
```

## Appendix I – Codewarrior LIN project cloning

The easiest way to generate an MC68HC908EY16 LIN Codewarrior project is to “clone” the example application included on the CD. This automatically ensures that the compiling, linking and building process is configured correctly. An appropriate procedure is shown below.

1. In the directory *…\lin08EY\sample*, make an additional copy of the folder *EYLEDemo* and give it an appropriate name (e.g. LINnode).
2. Delete the *LEDemo.c* source file from the new folder *…\lin08EY\sample\LINnode* and add the application source file(s).
3. In the directory *…\lin08EY\sample\ide*, make an additional copy of the folder *EYLEDemo* and give it the same name as that used in step 1.
4. Rename the file *…\lin08EY\sample\ide\LINnode\LEDemo.mcp to LINnode.mcp.*
5. Launch Codewarrior, close any open projects and drag in the file *…\lin08EY\sample\ideLINnode LINnode.mcp.*
6. Remove LEDemo.c from the project by selecting the file (in files folder) and deleting it using the “Edit” pull-down menu.
7. Add the required source file(s) to the project by selecting “Add Files” from the “project” pull-down menu. Browse for the file(s), select and click on “add”. Typically the single file *…\lin08EY\sample\LINnode\LINnode.c* is required. If the added file appears in an inappropriate position in the list of project files it can be dragged into the source folder.
8. Use the LIN08 slave settings (leftmost) icon to select “target – access paths”. Remove path *(Project)..\..\EYLEDdemo* to ensure that it isn’t inadvertently accessed. The path *(Project)..\..LINnode* is required but should already have been added automatically.
9. If any additional include files are required they should be added to folder *…\lin08EY\inc*. Include files can optionally be added to the project as described above for source files.
10. The file *hc08ey16.prm* will have been copied to *…\lin08EY\sample\ide\LINnode\hc08ey16.prm*. and this file should be inspected and edited or replaced as necessary. Sometimes no modification will be necessary.
11. In cloned projects, compiler macros are set up to use *slave.cfg* in place of *lincfg.h* and *slave.id* in place of *linmsgid.h* (see reference [2]). *slave.cfg* and *slave.id* will have been copied into *…\lin08EY\sample\LINnode* and should be inspected and edited if changes are required. In particular *slave.id* should be modified to specify the appropriate IDs and *slave.cfg* should be modified to enter the appropriate values for the baud rate and the bus timeout. The *slave.cfg*

and *slave.id* files used in LEDemo are shown in appendix II.

12. Many files within the project, for instance the output files *slave.abs* and *slave.sx* will have retained their original *"slave"* names. The simplest option is to retain these names as they are but they can be changed if desired. *Slave.sx* is the S19 record file required to program the flash of an MC68HC908EY16.
13. Close Codewarrior. When relaunched, the newly created project will be available for simulation and/or dubugging under *"open recent"* in the *file* pull-down menu.

## Appendix IIa – Include file (register definitions for the MC68HC908EY16)

```
/***************************************************
  HC08EY16.H
  Register definitions for the 908EY16

  P. Topping                                24-01-02
***************************************************/


#define PTA *((volatile unsigned char *)0x0000)
#define PTB *((volatile unsigned char *)0x0001)
#define PTC *((volatile unsigned char *)0x0002)
#define PTD *((volatile unsigned char *)0x0003)
#define PTE *((volatile unsigned char *)0x0008)

#define DDRA *((volatile unsigned char *)0x0004)
#define DDRB *((volatile unsigned char *)0x0005)
#define DDRC *((volatile unsigned char *)0x0006)
#define DDRD *((volatile unsigned char *)0x0007)
#define DDRE *((volatile unsigned char *)0x000A)

#define CONFIG1 *((volatile unsigned char *)0x001F)
#define CONFIG2 *((volatile unsigned char *)0x001E)

#define TBCR   *((volatile unsigned char *)0x001C)

#define TBSC   *((volatile unsigned char *)0x002B)
#define TBCNTH *((volatile unsigned char *)0x002C)
#define TBCNTL *((volatile unsigned char *)0x002D)
#define TBMODH *((volatile unsigned char *)0x002E)
#define TBMODL *((volatile unsigned char *)0x002F)
#define TBSC0  *((volatile unsigned char *)0x0030)
#define TBCH0H *((volatile unsigned char *)0x0031)
#define TBCH0L *((volatile unsigned char *)0x0032)
#define TBSC1  *((volatile unsigned char *)0x0033)
#define TBCH1H *((volatile unsigned char *)0x0034)
#define TBCH1L *((volatile unsigned char *)0x0035)

#define ICGCR  *((volatile unsigned char *)0x0036)
#define ICGMR  *((volatile unsigned char *)0x0037)
#define ICGTR  *((volatile unsigned char *)0x0038)

#define VECTF (void(*const)()) /* Vector table function specifier */
```

## Appendix IIb – vector.c

```
#define VECTOR_C
/*****************************************************************************
*
*       Copyright (C) 2001
** Functions:    Vectors table for LIN08 Drivers with  Freescale API
*
* Description:  Vector table and node's startup for HC08.
*               The users can add their own vectors into the table,
*               but they should not replace LIN Drivers vectors.
*
* Notes:
*
*****************************************************************************/

#if defined(HC08)                       /* for HC08 */

#if defined(HC08EY16)
extern void LIN_ISR_SCI_Receive();      /* ESCI receive ISR       */
extern void LIN_ISR_SCI_Error();        /* ESCI error ISR         */
extern void TimerB();                   /* Timer Module B Overflow ISR - */
// extern void TimerB0();               /* Timer Module B Channel 0 ISR */
// extern void BREAK_Command();         /* SWI ISR                */
#endif /* defined(HC08EY16) */


/*****************************************************************************
    NODE STARTUP
    By default compiler startup routine is called.
    User is able to replace this by any other routine.
*****************************************************************************/

#if defined(HICROSS08)
#define Node_Startup    _Startup
extern void _Startup();                 /* HiCross compiler startup routine declaration */
#endif  /* defined(HICROSS08) */

/*****************************************************************************
    INTERRUPT VECTORS TABLE
    User is able to add another ISR into this table instead NULL pointer.
*****************************************************************************/

#if !defined(NULL)
#define NULL    (0)
#endif /* !defined(NULL) */

#undef  LIN_VECTF

#if defined(HICROSS08)
#define LIN_VECTF ( void ( *const ) ( ) )
#pragma CONST_SEG VECTORS_DATA              /* vectors segment declaration */
void (* const _vectab[])( ) =
#endif  /* defined(HICROSS08) */

#if defined(HC08EY16)
```

```
/*************************************************************************/
/*      HC08EY16                                                         */
/*                                                                       */
/*      These vectors are appropriate for the following MC68HC908EY16    */
/*      mask sets:-   0L38H, 1L38H, 0L31N,and 1L31N                      */
/*      These mask sets had a fault in their interrupt vector table and  */
/*      hence in the interrupt priorities.                               */
/*                                                                       */
/*      For the vector address in the corrected mask set (2L31N) see     */
/*      the MC68HC908EY16 technical data sheet.                          */
/*                                                                       */
/*************************************************************************/

{
    LIN_VECTF NULL,                         /* 0xFFDC   Timebase          */
    LIN_VECTF NULL,                         /* 0xFFDE   SPI transmit      */
    LIN_VECTF NULL,                         /* 0xFFE0   SPI receive       */
    LIN_VECTF NULL,                         /* 0xFFE2   ADC               */
    LIN_VECTF NULL,                         /* 0xFFE4   Keyboard          */
    LIN_VECTF LIN_ISR_SCI_Error,            /* 0xFFE6   ESCI error        */
#if defined(MASTER)                         /* (used for Master node only)*/
    LIN_VECTF LIN_ISR_SCI_Transmit,         /* 0xFFE8   ESCI transmit     */
#endif /* defined(MASTER) */
#if defined(SLAVE)
    LIN_VECTF NULL,                         /* 0xFFE8   ESCI transmit     */
#endif /* defined(SLAVE) */
    LIN_VECTF LIN_ISR_SCI_Receive,          /* 0xFFEA   ESCI receive      */
    LIN_VECTF NULL,                         /* 0xFFEC   TIMER B overflow  */
    LIN_VECTF NULL,                         /* 0xFFEE   TIMER B channel 1 */
    LIN_VECTF NULL,                         /* 0xFFF0   TIMER B channel 0 */
    LIN_VECTF NULL,                         /* 0xFFF2   TIMER A overflow  */
    LIN_VECTF NULL,                         /* 0xFFF4   TIMER A channel 1 */
#if defined(MASTER)                         /* (used for Master node only)*/
    LIN_VECTF LIN_ISR_Timer0,               /* 0xFFF6   TIMER A channel 0 */
#endif /* defined(MASTER) */
#if defined(SLAVE)
    LIN_VECTF NULL,                         /* 0xFFF6   TIMER A channel 0 */
#endif /* defined(SLAVE) */
    LIN_VECTF NULL,                         /* 0xFFF8   CMIREQ            */
    LIN_VECTF NULL,                         /* 0xFFFA   IRQ               */
//  LIN_VECTF BREAK_Command,                /* 0xFFFC   SWI               */
    LIN_VECTF NULL,                         /* 0xFFFC   SWI               */
    LIN_VECTF Node_Startup                  /* 0xFFFE   RESET             */
};

#endif  /* defined(HC08EY16) */


#if defined(HICROSS08)
#pragma CONST_SEG DEFAULT
#endif  /* defined(HICROSS08) */

#endif  /* defined(HC08) */
```

## Appendix IIc – slave.cfg (LIN configuration file)

```
#ifndef LINCFG_H
#define LINCFG_H

/*****************************************************************************
*
*       Copyright (C) 2001
*
* Functions:    LIN Driver static configuration file for LIN08 Slave sample
*               with  Freescale API
*
* Notes:
*
*****************************************************************************/

#if defined (HC08)

/*
    This definition configures the LIN bus baud rate.
    This value shall be set according to target MCU
    SCI register usage.
    HC08EY16: the 8-bit value will be masked by 0x37
    and put into SCBR register.
*/

/* Selects 9600 baud rate if using a 9.8304MHz crystal */
//#define LIN_BAUDRATE            0x04u

/* Selects 9600 baud rate if using a 8MHz crystal */
#define LIN_BAUDRATE            0x30u

/*
    This definition sets the number of user-defined time clocks
    (LIN_IdleClock service calls), recognized as "no-bus-activity"
    condition. This number shall not be greater than 0xFFFF.
*/

#define LIN_IDLETIMEOUT         500u

#endif /* defined (HC08) */

#endif /* !define (LINCFG_H) */
```

## Appendix IId – slave.id (LIN message ID file)

```
#ifndef LINMSGID_H
#define LINMSGID_H
/*****************************************************************************
*
*                   Copyright (C) 2001
*
* Functions:    Message Identifier configuration for LIN08 Slave sample
*               with  Freescale API
*
*
* Notes:
*
*****************************************************************************/

#define LIN_MSG_20  LIN_RECEIVE
#define LIN_MSG_21  LIN_RECEIVE
#define LIN_MSG_30  LIN_RECEIVE

/* this string is not necessary - just as an example */

#define LIN_MSG_20_LEN  4       /* standard length */
#define LIN_MSG_21_LEN  4       /* standard length */
#define LIN_MSG_30_LEN  8       /* standard length */

#endif /* defined(LINMSGID_H)*/
```

# This page is intentionally left blank

### *How to Reach Us:*

**Home Page:**
www.freescale.com

**E-mail:**
support@freescale.com

**USA/Europe or Locations Not Listed:**
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

**Europe, Middle East, and Africa:**
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

**Japan:**
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

**Asia/Pacific:**
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

***For Literature Requests Only:***
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com