

# LIN Node Temperature Display

by: Peter Topping  
East Kilbride

## 1 Introduction

Many of the problems of excessive wiring within a car can be resolved by using a serial multiplex bus like CAN or LIN. The LIN (Local Interconnect Network, reference 1) bus is lower in cost and ideally suited to use within a door or in other areas in the car where its limited data rate of 20,000 baud is adequate. The display of external temperature is an application which falls into this category. As a common location for the outside temperature sensor is in a wing mirror, the data often originates from the door. The actual display is, however, within the car some distance away and the LIN bus is eminently suitable to facilitate the connection. A LIN node requires only 3 wires; there is one LIN data line, the other two connections being the positive and negative supplies.

This application note presents the design of a temperature display LIN slave node. It was developed as part of a complete door project which also included the keypad module described in application note AN2205.

## Contents

1	Introduction .....	1
2	Hardware .....	2
3	Software .....	4
4	References .....	8
5	Software listing .....	8
	Appendix ALIN Driver Project Set-up .....	12
	Appendix BInclude File (register definitions for the 908AZ60A) .....	13
	Appendix CSlave.cfg (LIN configuration file) .....	14
	Appendix DSlave.id (LIN message ID file) .....	15

The messaging scheme for the door incorporated a byte for temperature in the mirror response field as shown in table 1. The LIN master is, in this case, the body controller. On a regular basis, say every 100ms, the master sends a frame header with the mirror's ID and it responds with the two-byte message shown. It includes a byte with the temperature encoded in half Centigrade degree increments from  $-30^{\circ}\text{C}$  to  $97.5^{\circ}\text{C}$ . In this case, the data is read directly by the temperature display node (slave to slave communication), but the master could alternatively read it and retransmit to the relevant slave node.

ID \$CA	Temperature Sensor (byte 0)	Error status. (byte 1)
bit 0	Temperature sensor data encoded in half Centigrade degree increments from $-30^{\circ}\text{C}$ (\$00) to $97.5^{\circ}\text{C}$ (\$FF).	LIN – bit error
bit 1		LIN – checksum error
bit 2		LIN – identifier parity error
bit 3		LIN – slave not responding error
bit 4		LIN – inconsistent sync. error
bit 5		LIN – no bus activity error
bit 6		not used
bit 7		not used

**Table 1. Format of Mirror Response Data**

## 2 Hardware

The target MCU for the temperature display module is the MC68HC908EY16. As this MCU was not available at the time of writing, this application note employs an MC68HC908AZ60A. Implementation on an MC68HC908EY16 would significantly reduce the cost. Not only is the MC68HC908EY16 a lower pin-count lower cost device but it will include an on-chip Internal Clock Generator (ICG) obviating the need for a crystal or ceramic resonator.

The circuit diagram used in the temperature display application 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 (or MC33661) and a 5 volt regulator. A single chip, the MC33689 (LIN SBC), can replace these chips. The regulator used is the 8-pin LT1121 which has the capability of shutting down into a low power sleep mode under the control of the MCU. In the arrangement shown, it wakes up via the MC33399 (or MC33661) by LIN bus activity.

The MC33399 (or MC33661) includes the 30kohm LIN pull-up so this does not need to be included on the PCB. The only discrete components required are pull-up resistors for the IRQ and Reset pins, decoupling capacitors and a crystal and its associated components. Two PortC pull-ups and a 9 volts zener circuit were also included to facilitate entry into monitor mode using an external serial interface. This facilitated in-circuit programming of the on-chip flash memory. The software was developed on the prototype PCB fitted with a target header for the MMDS development system rather than with an actual MC68HC908AZ60A.

As the LEDs are driven directly by the MCU, care had to be taken to avoid drawing excessive currents from port pins or exceeding the specified current or dissipation capability of the 5 volt regulator. The digit

current for an acceptable brightness is too high for a port pin so FET buffers are incorporated. The segment resistors of 220R were chosen to give a segment current of 10mA to allow a digit current up to 80mA. As the digits are driven with a 25% duty cycle this corresponds to an average current of 2.5mA per segment and 20mA for a digit with all 8 segments lit. The highest total current is drawn with a display of 88.0. This involves 21 active segments which, with a duty cycle of 25%, requires 52.5mA (2.5mA x 21).

	1 segment	1 digit (all segments)	Display: 88.0
Peak	10mA	80mA	80mA
Average	2.5mA	20mA	52.5mA

With an MCU  $I_{dd}$  of 15mA and 9 volts across the regulator (assuming a  $V_{Bat}$  of 14volts), the maximum dissipation of the LT1121 can be calculated:

$$\text{Dissipation} = 9\text{volts} \times (52.5\text{mA} + 15\text{mA}) \sim 610\text{mW}.$$

With an assumed ambient temperature of 26°C and a junction to ambient thermal resistance of 100C°/W, the maximum junction temperature of the LT1121 is:

$$T_J = T_{Amb} + 610\text{mW} \times 100\text{C}^\circ/\text{W} = 26 + 61 = 87^\circ\text{C}$$

This is well within the specified maximum of 125°C. If a significantly higher segment current were required, then segment drivers and a higher dissipation regulator should be employed.

Current versions of the MC68HC908AZ60A data sheet specify a maximum total source current of only 10mA. This was inserted as an interim figure while characterisation was done to check that the  $V_{oh}$  and  $V_{ol}$  level specifications were met under all possible conditions. In this application, these levels are not critical so the only concern is that the total current is not high enough to cause long term damage to the device. There is, however, no risk of this as long as the absolute maximum figure of 100mA per pin is observed. The maximum  $I_{dd}$  is significantly less than 100mA. Future versions of the data sheet will remove this unnecessary limitation.

A simple sleep mode was incorporated in the module when no bus activity occurs for two seconds. When this condition is detected, the MCU lowers the enable line to the MC33399 which in turn lowers the inhibit signal to the LT1121 regulator. The regulator goes into its standby mode and powers down the MCU. A resumption of bus activity wakes the MC33399 and the regulator switches the MCU on. In order to facilitate low sleep mode current consumption, an additional link was included to isolate the zener diode used to supply the high voltage required to enter monitor mode. The measured current consumption in sleep mode was 40µA. This is the combined standby current of the MC33399 and the LT1121 (both about 20µA).

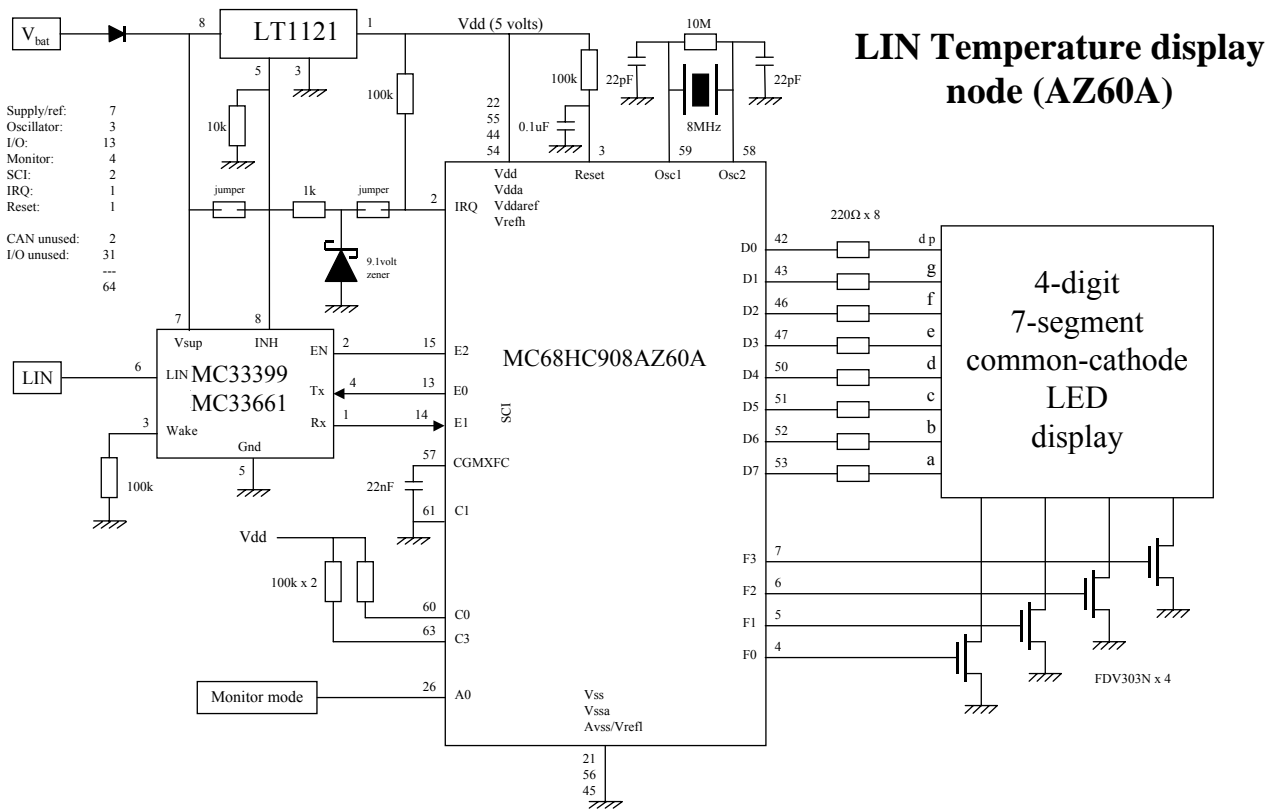


Figure 1. Temperature Display Module Circuit Diagram

### 3 Software

The temperature display module uses the Metrowerks LIN drivers so all I/O activity is handled without the application code which simply uses a “LIN\_GetMsg()” to receive the data provided by the sensor in the door mirror. The use of the LIN drivers results in fairly simple application software for the temperature node. In order to respond to a master request command frame (ID 0x3C), the user code has to include the function void LIN\_Command(). This is, for example, 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).

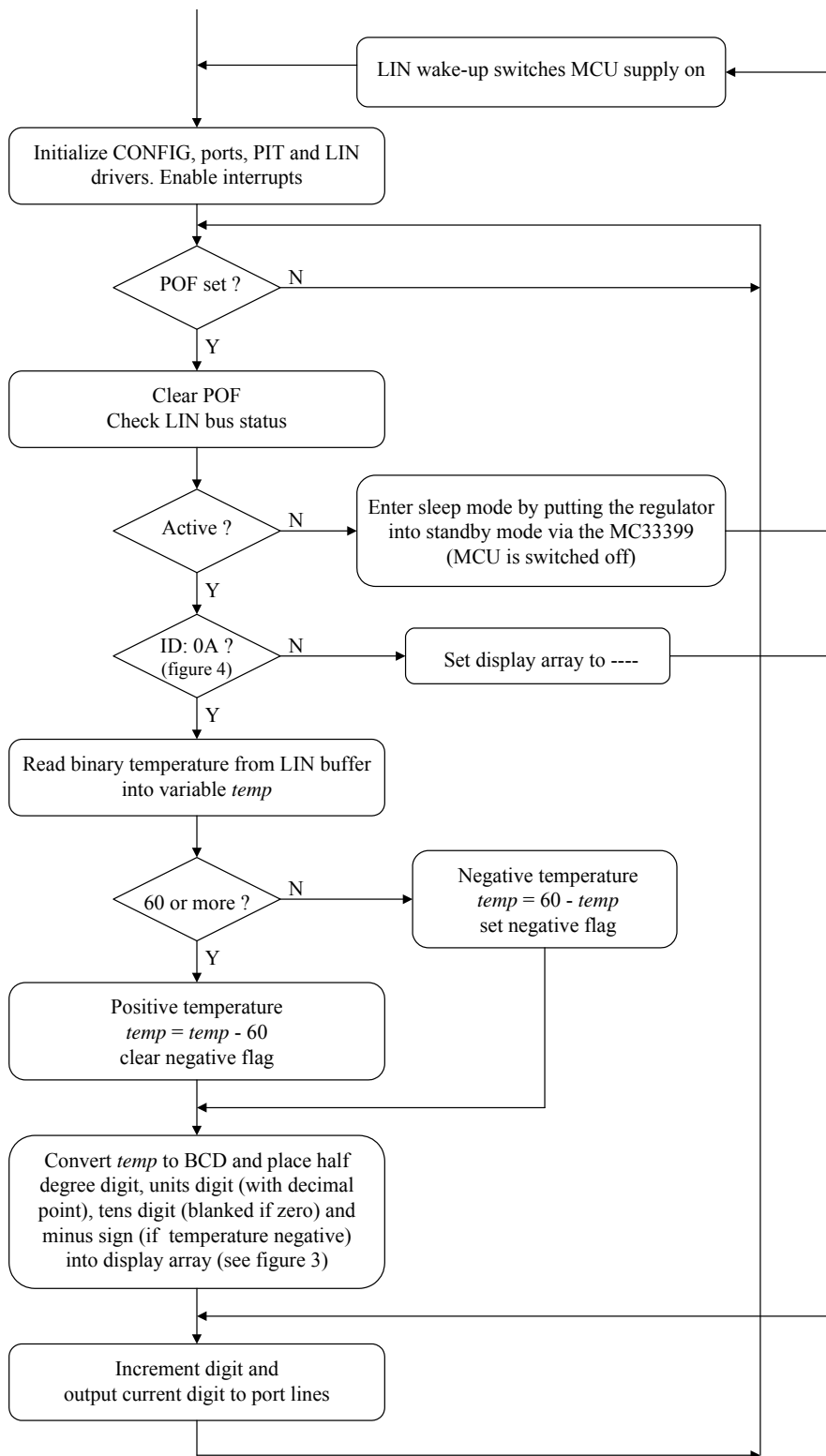
The main software flow diagram is shown in figure 2 and the complete code listed in section 5. Once the variables have been declared, the CONFIG and I/O registers are initialised. The CONFIG1 value of 0x71 disables the COP while the CONFIG2 value of 0x19 configures the MCU as an MC68HC908AZ60A (as opposed to an AS60A). Interrupts enable so that the LIN drivers, once initialised by LIN\_init(), can function. The main while loop uses the programmable interrupt timer (PIT) to facilitate a 200Hz repetition rate based on an 8MHz crystal. Once every 5ms, the PIT overflow flag sets and the main loop executes to convert the binary input data into 7-segment format. The 200Hz rate is also used to cycle round the 4 LED digits facilitating a flicker free 50Hz multiplexed display. Although the LIN buffer is read and the temperature converted into 4-digit 7-segment format every time, only one digit is actually driven every 5ms.

The LIN driver function `LIN_IdleClock` can for any bus activity. If not it increments a counter whose value compares with `LIN_IDLETIMEOUT` (defined in `slave.cfg`). If this number exceeds the function, `LIN_DriverStatus()` ceases to return a 1 (`LIN_STATUS_RUN`) indicating that the bus has been idle for, in this application, two seconds. If this is detected, the enable pin of the MC33399 is taken low and the MCU powers down by the disabling of the LT1121. The code also sets the display to "...". This is irrelevant in the powered down application, but gave an indication of a detected idle condition in the development environment where the emulated MCU's power was not controlled by the LT1121. In this application, it also made sense to show a special display if temperature data was not provided to the module. Without this addition the display would continue to show the last received temperature (or  $-30^{\circ}\text{C}$  if none had been received since the last power up). This feature was incorporated using `LIN_MsgStatus(0x0A)` in conjunction with the variable `error_count` and forces the display to "----" after a second of not receiving a LIN message with an ID of 0A (see figure 4).

To perform the conversion, the temperature byte transfers to the variable `bits` and, if the temperature is positive, the offset of 60 ( $30^{\circ}\text{C}$ ) subtracted. This would result in an incorrect underflowed value in `bits` if the temperature is negative ( $<60$ ) so if this is the case the subtraction is done the other way round ( $60 - bits$ ) to give a sensible (but positive) value. The fact that it was actually negative is recorded by setting the `negative` flag.

The value in `bits` is then converted as shown in figure 3 into the required 7-segment format using the array `seg`. Firstly the digit after the decimal point is configured as a zero or a five according to the least significant bit in `bits`. `Bits` divides by two to yield the binary temperature in  $^{\circ}\text{C}$ . This divides again by ten to give the tens digit while the units digit remains from this division. The units, tens and sign (hundreds) digits are then configured with the appropriate segments, the only complications being the addition of the decimal point to the units digit and the various possibilities for the tens digit. If the tens digit constitutes a leading zero then it should be blank unless the temperature is negative. In this case the display is neater if the negative sign is displayed in the tens position rather than in the hundreds position.

To output information to the LED display, `dcount` is incremented and the digit to be driven this time around the loop is selected by using the two least significant bits of the variable `dcount`. They are used as an index in the array `display` for the segment data going to portD and also in `scan` for the digit drive to portF.



**Figure 2. Main Software Flow Chart**

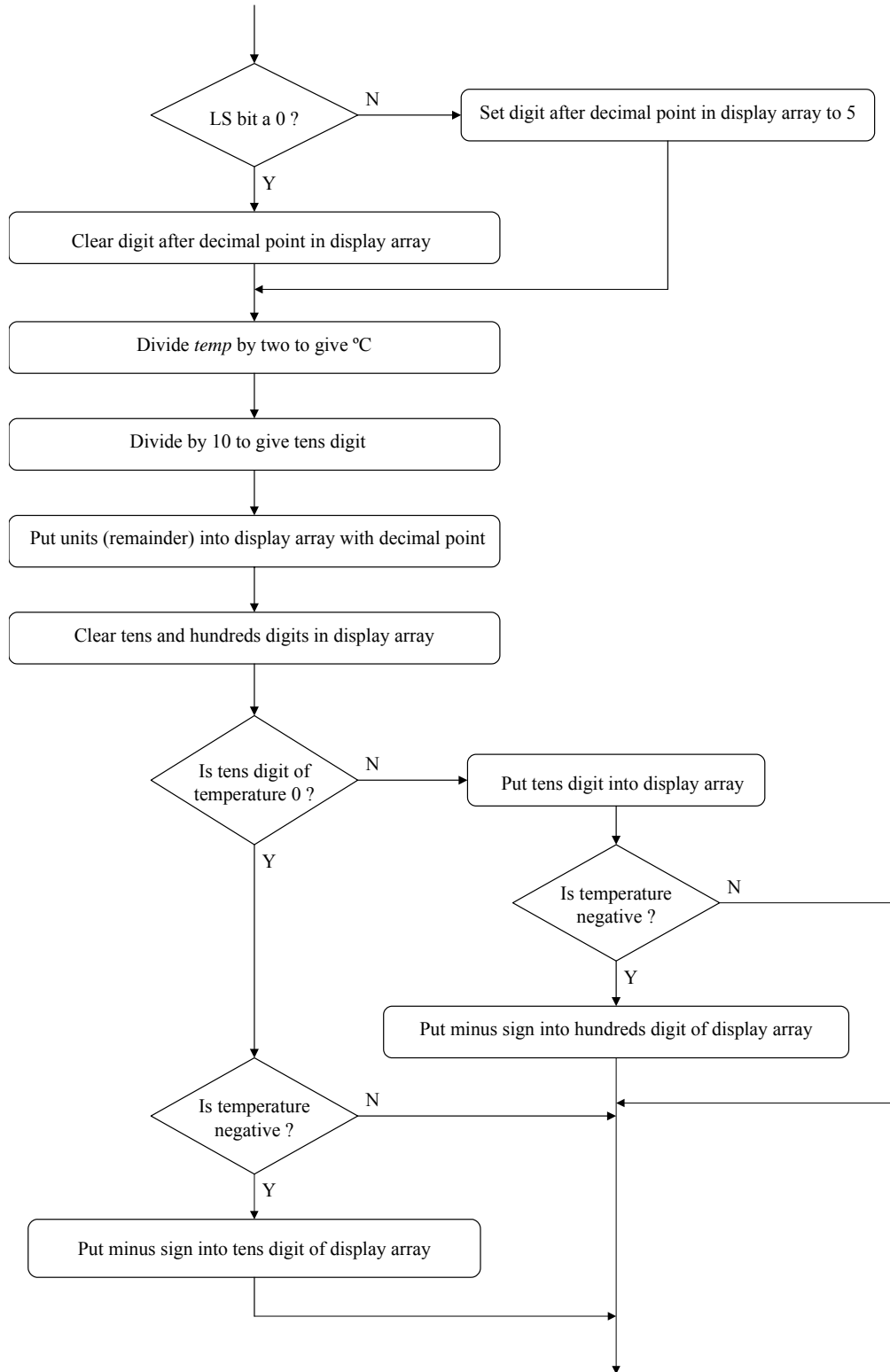


Figure 3. Flow Chart to Fill Display Array with Data

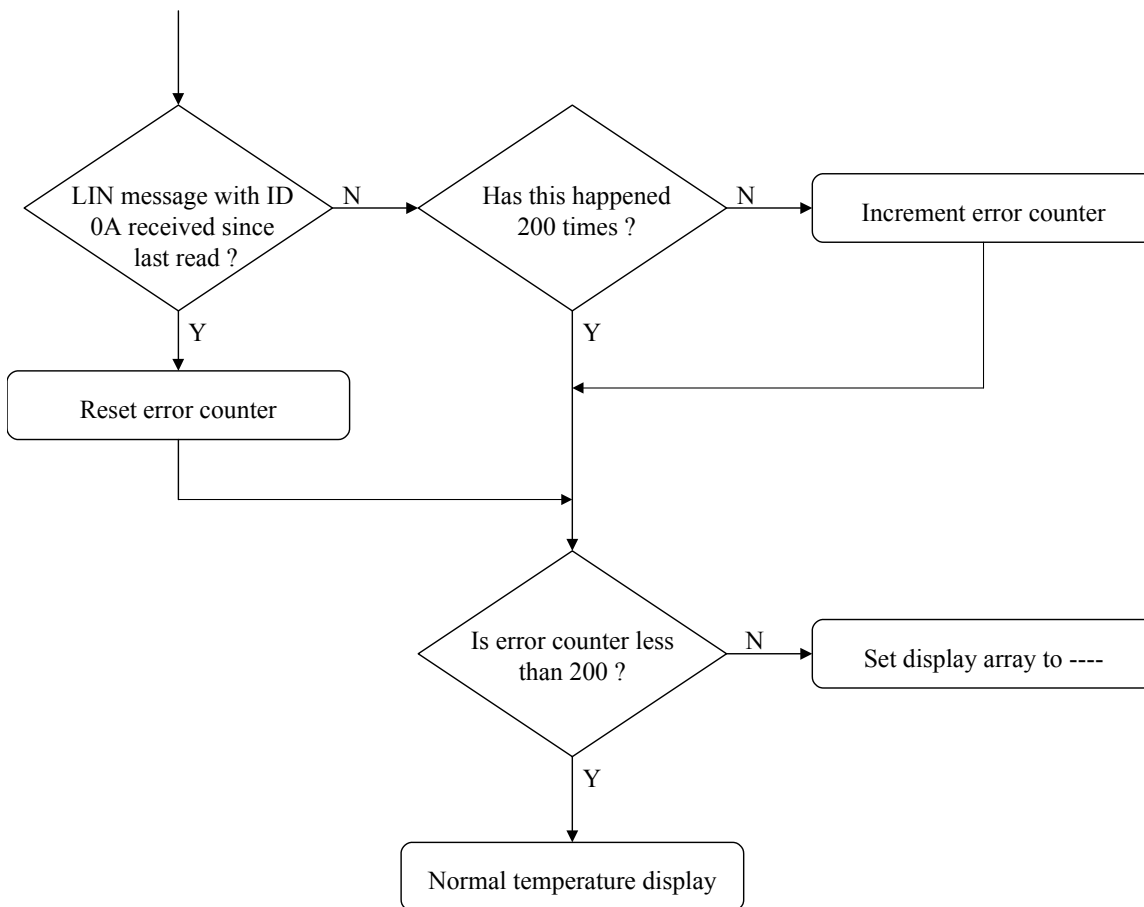


Figure 4. Flow Chart of Temperature Message Presence Detection

## 4 References

1. LIN Protocol Specification, Version 1.2, 17 November 2000.
2. LIN08 Driver User’s Manual rev 1.1, 13 March 2001.
3. MC68HC908AZ60A Technical Data.
4. MC68HC908EY16A Advance Information.
5. AN2205, Car door keypad using LIN, November 2001.

## 5 Software listing

/\*\*\*\*\*\*

Motorola reserves the right to make changes without further notice to any Product herein to improve reliability, function, or design. Motorola 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 licensed under its patent rights nor the right of others. Motorola products are not designed,

intended or authorised for use as components intended for surgical implant into the body, or other applications intended to support life, or any other application in which failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such intended or unauthorised application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, expenses and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorised use, even if such a claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola and the Motorola logo are registered trademarks of Motorola Ltd.

\*\*\*\*\*/

/\*\*\*\*\*

```
*
*
*          LIN Temperature Display Node - MC68HC908AZ60A
*          =====
*
*   Originator:   P. Topping
*   Date:         28th November 2001
*   Modified:     5th January 2002
*   Comment:      Sleep facility added
*
```

\*\*\*\*\*/

/\*\*\*\*\*

```
*
*   Globals and header files
*
```

\*\*\*\*\*/

```
#include <hc08az60.h>
#include <linapi.h>
```

```
unsigned char Mirror_data[2];
unsigned char display[4];
unsigned char scan[4]= {0x08,0x02,0x04,0x01};
unsigned char seg7[10]= {0xFC,0x60,0xDA,0xF2,0x66,0xB6,0xBE,0xE0,0xFE,0xF6};
unsigned char dcount;
```

## Software listing

```

/*****
*
* Function:          LIN_Command
* Description:      User call-back.
*                  Called by the driver after successful transmission or
*                  receiving of the Master Request Command Frame (ID: 0x3C).
* Returns:         never returns
*
*****/

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

/*****
*
*   Function name: Main
*   Originator:    P. Topping
*   Date:         5th January 2002
*
*****/

void main (void)
{

unsigned char bits;
unsigned char units;
unsigned char tens;
unsigned char negative;
unsigned char error_count;

CONFIG1 = 0x31;          /* disable LVI and COP    */
CONFIG2 = 0x11;          /* AZ60A, MSCAN disabled */

DDRA   = 0xFF;          /* configure unused port  */
DDRB   = 0xFF;          /* pins as outputs        */
DDRC   = 0x3F;
DDRD   = 0xFF;          /* segment drive          */
DDRE   = 0xFD;          /* LIN interface (MC33399) */
DDRF   = 0x7F;          /* digit drive via FETs   */
DDRG   = 0x07;
DDRH   = 0x03;

PTE     = 0x04;          /* MC33399 enable high    */

asm CLI;                /* clear interrupt mask    */

LIN_Init();             /* initialise LIN drivers  */

PITSC = 0x10;           /* start PIT at /1        */
PMDH = 0x27;            /* /10000 for a repetition */
PMDL = 0x10;           /* rate of 200Hz @ 8MHz.  */

while (1)
{
    if (PITSC & 0x80)    /* is PIT overflow set?   */
    {
        PITSC &= ~(0x80); /* yes, clear it          */
    }
}

```

```

LIN_IdleClock ();                                /* check for bus activity */
if (LIN_DriverStatus () != 0x01)                /* bus idle for 400 trys ? */
{
    display[0] = 0x01;                            /* LIN_STATUS_IDLE is 2    */
    display[1] = 0x01;                            /* but 3 is returned      */
    display[2] = 0x01;                            /* display ... (MMDS only) */
    display[3] = 0x01;                            /* real AZ60A powers down */
    PTE          = 0x00;                            /* from MC33399 enable low */
}
else
{
    if (LIN_MsgStatus (0x0A) != 0)                /* bus active but have we */
    {                                              /* had an ID of 0x0A ?    */
        if (error_count < 201 )                 /* no, error counter      */
        {                                        /* already 201 ?          */
            error_count ++;                       /* no, inc error counter  */
        }
    }
    else
    {
        error_count = 0;                          /* yes, new data available */
    }

    if (error_count > 200)                        /* data in last second ?  */
    {
        display[0] = 0x02;                        /* no, display ----      */
        display[1] = 0x02;                        /* to signify no valid    */
        display[2] = 0x02;                        /* data from temperature  */
        display[3] = 0x02;                        /* sensor                 */
    }

    else
    {                                              /* yes, data available    */
        LIN_GetMsg (0x0A, Mirror_data);           /* read sensor message    */
        bits = Mirror_data[0];                   /* and extract temp. byte */

        if (bits < 60)                           /* negative ?             */
        {
            bits = 60 - bits;                     /* yes, convert to positive */
            negative = 1;                          /* but remember it wasn't  */
        }
        else
        {
            bits = bits - 60;                      /* no, remove offset and  */
            negative = 0;                          /* clear negative flag    */
        }

        display[3] = seg7[0];                     /* display zero after     */
        if (bits & 0x01 == 0x01)                  /* decimal point but if LS */
        {
            display[3] = seg7[5];                 /* bit is a 1 display 5   */
        }

        bits = bits/2;                             /* lose LS bit            */
        tens = bits/10;                            /* find tens digit        */
        units = bits%10;                          /* and units digit        */

        display[2] = 1 | seg7[units];             /* units w. decimal point */
        display[1] = 0x00;                        /* clear tens digit       */
    }
}
    
```

## LIN Node Temperature Display, Rev. 1

## Software listing

```

display[0] = 0x00;          /* and hundreds digit */
if (tens != 0)             /* tens digit zero ? */
{
    display[1] = seg7[tens]; /* no, display it */
    if (negative)           /* negative ? */
    {
        display[0] = 0x02;  /* yes put "-" in hundreds */
    }
}
else if (negative)        /* tens zero, negative ? */
{
    display[1] = 0x02;     /* yes, put "-" in tens */
}
}                          /* */

dcount ++;                /* increment digit counter */
PTF = (PTF & 0xF0) | scan[dcount & 0x03]; /* and use the two LS bits */
PTD = display[dcount & 0x03]; /* to get data from arrays */
}
}
}

```

## Appendix A LIN Driver Project Set-up

This application was developed on a Motorola MMDS development system using the Metrowerks Codewarrior development environment. The easiest way to generate an MC68HC908AZ60A LIN Codewarrior project is to “clone” the example application included with the LIN drivers. This automatically ensures that the compiling, linking and building process is configured correctly. An appropriate procedure is shown below.

1. In the LIN driver directory ...\*lin08\sample*, make an additional copy of the folder *slave* and give it an appropriate name. In this application *temp08* was used.
2. Delete the *slave.c* source file from the new folder ...\*lin08\sample\temp08* and add application source file(s).
3. In the LIN driver directory ...\*lin08\sample\ide*, make an additional copy of the folder *slave* and give it the same name as that used in step 1.
4. Delete all files from ...\*lin08\sample\ide\temp08\slave\_Data\LIN08\_slave\objectCode* and ...\*lin08\sample\ide\temp08\bin*. This is not essential but makes it easier to see if the compiler and linker output files are being correctly generated.
5. Launch Codewarrior, close any open projects and drag in the file ...\*lin08\sample\ide\temp08\slave.mcp*.
6. Remove *slave.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”. In this application the single file ...\*lin08\sample\temp08\tempaz* was required. If the added file appears in an inappropriate position in the list of project files it can be dragged to a more suitable place.

8. Use the LIN slave settings (leftmost) icon to select “target – access paths”. Remove path (*Project*)\..\slave to ensure that it isn’t inadvertently accessed. The path (*Project*)\..\temp08 is required but should already have been added automatically.
9. If any include files are required they should be added to folder ... \lin08\inc. In this application, the file *hc08az60.h* (see appendix II) was added. Include files can optionally be added to the project as described above for source files.
10. The LIN driver’s .prm file *hc08az32.prm* will have been copied to ... \lin08\sample\ide\temp08\hc08AZ32.prm. and this file should be inspected and edited or replaced as necessary. In this application, no modification was necessary.
11. In cloned projects, compiler macros 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 ... \lin08\sample\temp08 and should be inspected and edited if changes are required. In this application *slave.id* was modified to specify the appropriate IDs and *slave.cfg* was modified to enter the appropriate values for the baud rate (0x30) and the bus timeout (400). *slave.cfg* and *slave.id* 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 MC68HC908AZ60A.
13. Close Codewarrior. When relaunched, the newly created project will be available for simulation and/or debugging under “open recent” in the *file* pull-down menu.

## Appendix B Include File (register definitions for the 908AZ60A)

```

/*****
  HC08AZ60.H
  Register definitions for the MC68HC908AZ60(A)

  P. Topping                                1-06-01
  *****/

#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 PTF *((volatile unsigned char *)0x0009)
#define PTG *((volatile unsigned char *)0x000A)
#define PTH *((volatile unsigned char *)0x000B)

#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 *)0x000C)
#define DDRF *((volatile unsigned char *)0x000D)
#define DDRG *((volatile unsigned char *)0x000E)
#define DDRH *((volatile unsigned char *)0x000F)
    
```

## Software listing

```
#define CONFIG1 *((volatile unsigned char *)0x001F)
#define CONFIG2 *((volatile unsigned char *)0xFE09)

#define PITSC *((volatile unsigned char *)0x004B)
#define PCNTH *((volatile unsigned char *)0x004C)
#define PCNTL *((volatile unsigned char *)0x004D)
#define PMODH *((volatile unsigned char *)0x004E)
#define PMODL *((volatile unsigned char *)0x004F)

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

## Appendix C Slave.cfg (LIN configuration file)

```
#ifndef LINCFG_H
#define LINCFG_H

/*****
 *
 *      Copyright (C) 2001 Motorola, Inc.
 *      All Rights Reserved
 *
 *      The code is the property of Motorola GSG St.Petersburg
 *      Software Development
 *      and is Motorola Confidential Proprietary Information.
 *
 *      The copyright notice above does not evidence any
 *      actual or intended publication of such source code.
 *
 * Filename:      $Source: /net/sdt/vault-rte/cvsroot/lin/release/hc08/sample/slave/slave.cfg,v $
 * Author:       $Author: kam $
 * Locker:       $Locker: $
 * State:        $State: Exp $
 * Revision:     $Revision: 1.12 $
 *
 * Functions:    LIN Driver static configuration file for LIN08 Slave sample
 *              with Motorola API
 *
 * History:      Use the CVS command log to display revision history
 *              information.
 *
 * Description:  It is allowed to modify by the user.
 *
 * Notes:
 *
 *****/

#if defined (HC08)

/* External MCU frequency = 16MHz      */
/* SCI Baud rate           = 15.6K      */

/*
 * This definition configures the LIN bus baud rate.
 * This value shall be set according to target MCU
 * SCI register usage.
 * HC08AZ32: the 8-bit value will be masked by 0x37
 * and put into SCBR register.
 */
#define LIN_BAUDRATE           0x30u

/*
 * This definition set the number of user-defined time clocks
 * (LIN_IdleClock service calls), recognized as "no-bus-activity"
 * condition.
 */
```

```

    T|his number shall not be greater than 0xFFFF.
*/
#define LIN_IDLETIMEOUT          400u

#endif /* defined (HC08) */

#endif /* !define (LINCFG_H) */

```

## Appendix D Slave.id (LIN message ID file)

```

#ifndef LINMSGID_H
#define LINMSGID_H
/*****
*
*   Copyright (C) 2001 Motorola, Inc.
*   All Rights Reserved
*
*   The code is the property of Motorola GSG St.Petersburg
*   Software Development
*   and is Motorola Confidential Proprietary Information.
*
*   The copyright notice above does not evidence any
*   actual or intended publication of such source code.
*
* Filename:      $Source: /net/sdt/vault-rte/cvsroot/lin/release/hc08/sample/slave/slave.id,v $
* Author:       $Author: snl $
* Locker:       $Locker: $
* State:        $State: Exp $
* Revision:     $Revision: 1.8 $
*
* Functions:    Message Identifier configuration for LIN08 Slave sample
*               with Motorola API
*
* History:      Use the CVS command log to display revision history
*               information.
*
* Description:
*
* Notes:
*
*****/

#define LIN_MSG_09  LIN_RECEIVE
#define LIN_MSG_0A  LIN_RECEIVE
#define LIN_MSG_21  LIN_RECEIVE
#define LIN_MSG_20  LIN_SEND

/* this string is not necessary - just as an example */
#define LIN_MSG_20_LEN      4      /* standard length */
#define LIN_MSG_09_LEN     2      /* standard length */
#define LIN_MSG_0A_LEN     2      /* standard length */
#define LIN_MSG_21_LEN     4      /* standard length */

#endif /* defined(LINMSGID_H) */

```

## **How to Reach Us:**

### **Home Page:**

[www.freescale.com](http://www.freescale.com)

### **E-mail:**

[support@freescale.com](mailto: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](mailto: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](mailto: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](mailto: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](mailto: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](mailto:LDCForFreescaleSemiconductor@hibbertgroup.com)

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor 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 or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor 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 Semiconductor was negligent regarding the design or manufacture of the part.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

© Freescale Semiconductor, Inc. 2006. All rights reserved.

RoHS-compliant and/or Pb-free versions of Freescale products have the functionality and electrical characteristics as their non-RoHS-compliant and/or non-Pb-free counterparts. For further information, see <http://www.freescale.com> or contact your Freescale sales representative.

For information on Freescale's Environmental Products program, go to <http://www.freescale.com/epp>.