Application Note

AN2575/D
9/2003

MC68HC908EY16 ESCI LIN Drivers

By: Wolfgang Bihlmayr
Motorola, Munich
and Peter Topping
Motorola, East Kilbride

Introduction

Reliable communication via the asynchronous LIN[1] protocol requires an MCU with a bus clock accurate enough to avoid errors. MCUs that use clocks based on crystal or ceramic resonators easily provide very accurate bus clocks. The LIN protocol was designed to also allow more cost-effective solutions. MCUs with on-chip oscillators can be successfully used, even though the on-chip oscillators have accuracy poorer than a crystal's by several orders of magnitude.

The most significant change from normal UART timing is the increase of the usual 10-bit break to 13 bits. This allows an MCU with an inaccurate clock (up to ±14%) to reliably distinguish a break from a data byte containing eight zeros (nine including the start bit). Following this break, the protocol specifies the inclusion of a synchronization byte whose data is always \$55. This field includes five falling (recessive to dominant) edges that can be used as a reference for clock and/or baud rate adjustments.

The MC68HC908EY16 is a high-performance MCU suitable for use in a low-cost LIN slave node. It can be used with or without an external crystal. To enable use without a crystal or any other external clock component, it incorporates an internal clock generator (ICG). The ICG can be programmed to any frequency from 307.2 kHz to 32 MHz in increments of 307.2 kHz. As with all HC08 MCUs, the bus frequency is one-quarter of this clock frequency. The use of the ICG reduces cost and eliminates the need for pins dedicated to clock circuitry. Not putting the clock on external pins also greatly reduces electromagnetic emissions — see application note AN2344/D: *HC908EY16 EMI Radiated Emissions Results*. (This document and other helpful documents are listed in the **References** section.)

1. Local Interconnect Network, see the **References** section.

NOTE: *With the exception of mask set errata documents, if any other Motorola document contains information that conflicts with the information in the data sheet, the data sheet should be considered to have the most current and correct data.*

The downside of using the ICG is that the frequency is not as accurate as that of a crystal or resonator. Parametric spreads during manufacture result in an overall untrimmed accuracy of ±25% from the specified nominal reference frequency of 307.2 kHz. In many applications, this is acceptable and no trimming is necessary. For LIN, however, greater accuracy is required. This application note describes a method of achieving the required performance.

MC68HC908EY16 Enhancements for LIN

The MC68HC908EY16 includes an ESCI (enhanced serial communications interface) module that incorporates three enhancements specifically for LIN (compared with the standard SCI used on many other HC08 MCUs).

- The recognition of a 13-bit break
- A fractional-divide baud rate prescaler that allows fine adjustment of the baud rate
- An arbiter counter that has 10 bits (9 bits plus an overflow bit) and can be used as a mini-timer to measure break and bit times

The ICG incorporates a digitally controlled oscillator (DCO) whose operation is fully described in application note AN2498/D: *Initial trimming of the MC68HC908 ICG.* The DCO uses registers DDIV and DSTG that are managed by the ICG hardware to make automatic corrections to the frequency of the clock. The actual output frequency is determined by the user via the ICGTR and ICGMR registers. The output frequency is set with the multiplier register, ICGMR. This register is set to 21 ($15) at reset, giving a default clock frequency of 6.4512 MHz ±25% (21 times 307.2 kHz). The user can change this value as required. In this application, a value of 64 is used to give a nominal bus frequency of 4.9152 MHz.

Baud Rate Requirements for LIN

The clock accuracy required for a LIN node using a crystal or ceramic resonator is ±1.5%. Although slave-to-slave communications tighten this requirement, it is so easily met by a crystal that this rarely causes a problem. The requirements for a node using an internal oscillator are more complex. The LIN baud rate requirements are summarized in **Table 1**. Slave-to-slave communication using the LIN protocol requires that no two nodes may differ by more than ±2%. Halving the individual requirement to ±1% is the simplest way of achieving this level of accuracy. Doing this is necessary if there are two or more similar nodes using internal clocks. If there is only one of this type of node, the better accuracy of the other nodes may render ±1% unnecessarily tight.

Table 1. LIN Baud Rate Requirements

Sync. Status	Requirement	Accuracy	Condition
Before synchronization	Break recognition	±14%(1)	Master clock accuracy assumed to be within ±0.5% of nominal
After synchronization (master-slave)	LIN messaging with the master	±2%	Relative to the master's clock
After synchronization (inc. slave-slave)	LIN messaging with the master and other slaves	±1%	

1. The pre-synchronization accuracy in rev. 1.3 is ±15%, but this will be tightened to 14% in future revisions.

Two separate trimming operations are required to meet these requirements. To meet the ±14% accuracy required to reliably recognize the break signal, pre-trimming the ICG frequency is necessary. To achieve the more precise ±1% or ±2% requirement for LIN communication, a further adjustment is required. These adjustments are covered in the next two sections.

ICG Trimming

The untrimmed internal oscillator on the MC68HC908EY16 is only accurate to about ±25%. Therefore, initial trimming is required to adjust the oscillator frequency near enough to the target frequency to allow recognition of a LIN break symbol. Normally, this initial trimming would be done during production of the application PCB. Then the trim value for the specific device would be stored in the nonvolatile memory of the MCU. Each time the MC68HC908EY16 is powered up, the stored trim value would be copied from nonvolatile memory into the working trim register to adjust the internal oscillator frequency. Most production programming tools support trimming, but the exact implementation varies from one tool to another. Because of that, refer to the documentation for your programmer for more information.

One method of initially trimming the ICG is fully described in AN2498/D (see the **References** section). This trimming would usually be done only on power-up with a value that is stored in FLASH and loaded into the ICGTR register. This value can be either embedded into the code (as shown in the software listing) or stored in a pre-defined FLASH location and transferred to ICGTR on power-up. With this pre-trimming done on an MC68HC908EY16, the frequency is guaranteed to be within ±7% for 4.5 to 5.5 V V_{DD} and –40°C to +85°C (±10% for the full automotive temperature range). This is well within the ±14% requirement to enable the MCU to recognize the 13-bit break.

The trimming adjustment involves measuring the actual bus frequency with the use of an external pulse of known length. Then, the trim register is adjusted by the number that gives the required correction. (Note that changing its value by 1 causes a frequency change of ~0.195%.) In the case of AN2498/D, the external pulse is 1024 μs between successive rising edges. The number of 4915.2-kHz bus cycles that would occur in this period is given by:

$$\text{cnt1024} = (64 \times 307.2\ \text{kHz} \times 1024\ \mu\text{s}) \div 4 \div 1000 = 5033$$

The adjustment to the ICGTR is made using the following equation where delta0 is the actual number of counts (measured by timer A channel 0) between successive rising edges of the external 1024 μs signal.

$$\text{ICGTR} = \text{ICGTR} + (512 \times (\text{delta0} - 5033)) \div 5033$$

Independent of how the initial trimming is done, the improved accuracy achieved by this initial ICG pre-trimming allows the break to be recognized. It is, however, not sufficiently accurate to meet the ±2% specification required to guarantee reliable LIN communication. Adjustment to within ±1% can be achieved using the LIN message itself because the synchronization byte of $55 immediately following the break has been incorporated for this purpose. Because the baud rate is always known, the time between edges within this byte can be used to measure the precise bit-time relative to the actual clock frequency. In particular, the time between two successive negative edges yields a known reference of two bit-times. The use of the same edge for the start and finish of the measurement eliminates inaccuracies due to asymmetry between rising and falling edges. The negative edge is preferred because it is generated by a dominant, active-low logic level. This is more accurate than the use of positive edges which are generated by pullup resistors.

ESCI Prescaler Baud Rate Adjustment

One possible method of adjusting the baud rate would be to use the timing information derived from the LIN message to re-trim the ICG. However, the DCO can take several hundred microseconds to settle after an adjustment. Thus, it is not always fast enough to facilitate the reception of the message whose synchronization byte is being used. With this method, that message could be missed, although the following message would be recognized. To adjust the receiving baud rate fast enough to respond to the message whose synchronization byte is being used, the MC68HC908EY16 incorporates a fractional divide prescaler. This prescaler is additional to the standard SCI prescaler. It allows a division ratio of 1 (bypassed by default) or 2 to 8 $^{31}/_{32}$ programmable in increments of $^{1}/_{32}$.

Adjustment of this prescaler could be carried out by using data derived from a timer channel connected to the ESCI Rx pin. With the MC68HC908EY16, the use of this additional resource is not necessary because the ESCI includes a small timer in the form of its arbiter counter. This is a 9-bit counter with a tenth bit to indicate that an overflow has occurred. The arbiter counter/timer has two modes of operation:

- Measurement of the break signal — This requires counting from a negative edge to the following positive edge. This mode is not used here because break recognition is guaranteed by the initial ICG trimming.
- Measurement of time between two successive negative edges — This is intended for bit-time measurement. It is the mode used here.

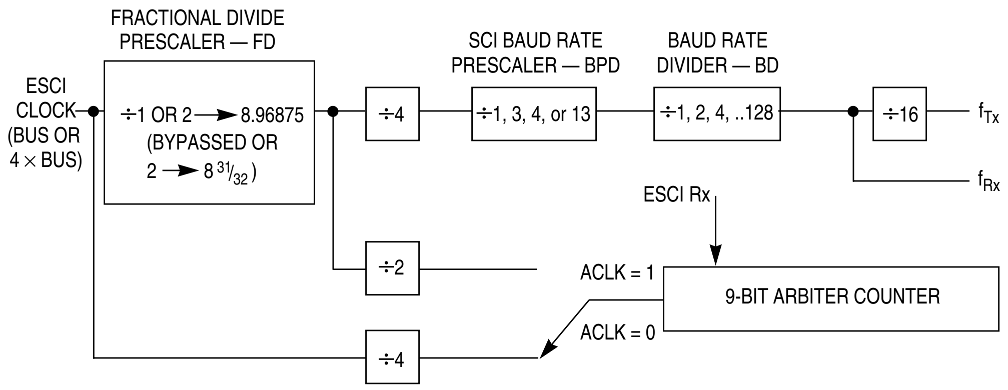

FRACTIONAL DIVIDE PRESCALER — FD
ESCI CLOCK (BUS OR 4 × BUS)
÷1 OR 2 → 8.96875 (BYPASSED OR 2 → 8 31/32)
÷4
SCI BAUD RATE PRESCALER — BPD
÷1, 3, 4, or 13
BAUD RATE DIVIDER — BD
÷1, 2, 4, ..128
÷16
fTx
fRx
ESCI Rx
÷2
ACLK = 1
ACLK = 0
9-BIT ARBITER COUNTER
÷4

Figure 1. ESCI Baud-Rate Prescalers

Figure 1 shows all the division ratios within the ESCI and its arbiter counter. The input clock to the arbiter counter is selected according to the mode being used. In the bit-time measuring mode being used here, the ACLK bit in the arbiter control register (SCIACTL) is 0 and the arbiter counter is clocked at one-quarter of the ESCI input clock. This input clock is selectable in CONFIG2 to be either the bus clock or the CGMXCLK clock, which is bus $\times$ 4. In the MC68HC908EY16 LIN drivers, the bus clock — in this case 4.9152 MHz — is used. BPD is fixed at 2 ($\div$4) and BD is fixed at 1 ($\div$2) for LIN communications at 9600 baud (see the technical data sheet MC68HC908EY16/D). In the equations below, FD is the fractional divide prescaler ratio.

In two bit-times, the arbiter counter counts $2 \times t_{Bit} \times f_{Bus} \div 4$ where t_{Bit} is in microseconds and f_{Bus} is in MHz.

Therefore $t_{Bit} = 2 \times \text{count} \div f_{Bus}$

The required bit-time $(1 \div \text{baud rate}) = 64 \times BPD \times BD \times FD \div f_{Bus} = 128 \times FD \div f_{Bus}$ (see the technical data sheet MC68HC908EY16/D).

The bit-times are equal when $2 \times \text{count} = 128 \times FD$;
i.e., when $FD = \text{count} \div 64$.

The ESCI fractional divide prescaler register (SCPSC) is programmed in two parts as shown in Table 2. The top three bits (PDS) constitute an integral divide ratio of 1 to 8. (1 is a special case where the prescaler is bypassed and the bottom five bits are ignored.) The divide ratio is one larger than the value in the top three bits of the register, and this must be taken into account when writing to SCPSC.

Table 2. Fractional Divide Prescaler Register (SCPSC)

7	6	5	4	3	2	1	0
PDS			PSSB (not used if PDS = 0)				

PDS	PD = PDS + 1	PSSB	PDFA = PSSB ÷ 32
0	1 (bypass)	0	0
1	2	1	$^{1}/_{32}$
2	3	2	$^{2}/_{32}$
3	4	3	$^{3}/_{32}$
.	.	.	.
.	.	.	.
7	8	31	$^{31}/_{32}$

Example: If SCPSC = 0×73 (01110011), PDS = 3 and PSSB = 19 so FD = 4 $^{19}/_{32}$
(in decimal, SCPSC is 115 so FD = 1 + $^{115}/_{32}$ = 4 $^{19}/_{32}$)

The bottom five bits (PSSB) contain the fractional part in increments of $^1/_{32}$. Because the integral part is five places to the left, the whole byte can be considered as 32 times the divide ratio. The only complication is the offset of 1 in the top three bits. Because of this, the overall fractional divide prescaler divide ratio, FD, is 1 + SCPSC ÷ 32. The integral part of the divide ratio (PDS + 1) is referred to as PD in the MC68HC908EY16 data sheet and the fractional part (PSSB ÷ 32) as PDFA, so the relationships are:

$$\text{FD} = \text{PD} + \text{PDFA} = 1 + \text{PDS} + \text{PSSB} \div 32 = 1 + \text{SCPSC} \div 32$$

As shown above, FD = count ÷ 64 therefore:

$$\text{SCPSC} = 32 \times ((\text{count} \div 64) - 1) = \text{count} \div 2 - \$20$$

From this equation, it can be seen that the arbiter count value should be divided by two and $20 subtracted from it to get the value which should be loaded into the SCPSC register. The code shown below has been extracted from the LIN driver software to illustrate how this can be done.

```
if(LIN_SCIACTL & LIN_SCIA_AFIN)                              // arbiter count finished ?
{
    if(LIN_CfgConst.LIN_BaudRate == 0)                                    // 19200 baud
    {
        LIN_SCPSC = LIN_SCIADAT - 0x20;                                   // count - 32
    }
    else if(LIN_CfgConst.LIN_BaudRate==1)                                 // 9600 baud
    {
        LIN_SCPSC = (((LIN_SCIACTL&0x03)<<7)|LIN_SCIADAT>>1) - 0x20;      // count/2 - 32
    }
}
```

The code includes the option of using 19,200 baud. This is simpler because there is no need to shift the value before adding $20. Because there is no room for the ninth bit (it will be zero anyway), it is not used. For 9600 baud, however, the ninth bit is relevant so the count value first has to be divided by two. This is done by shifting the ninth bit (from SCIACTL) up to bit 7 and ORing this with the shifted down (divided by two) value of the other eight bits (from SCIADAT). The least significant bit is discarded. After $20 is added, this gives the data required for the SCPSC register.

Hardware

The LINkits LIN evaluation board described in AN2573/D[1] is used for this application. A 2 × 16 character LCD has been added using the same interface as that described in AN2498/D. The complete circuit diagram used here is shown in **Figure 2**. Apart from the MCU and the MC33399, only a 5-volt regulator (in this case an LT1121) is required to implement the simple LIN node. This chip count could be further reduced by using the LIN system base chip (SBC), MC33689, because this chip incorporates a LIN physical interface and a voltage regulator. The other components required are the LCD module and a few resistors and decoupling capacitors (not shown).

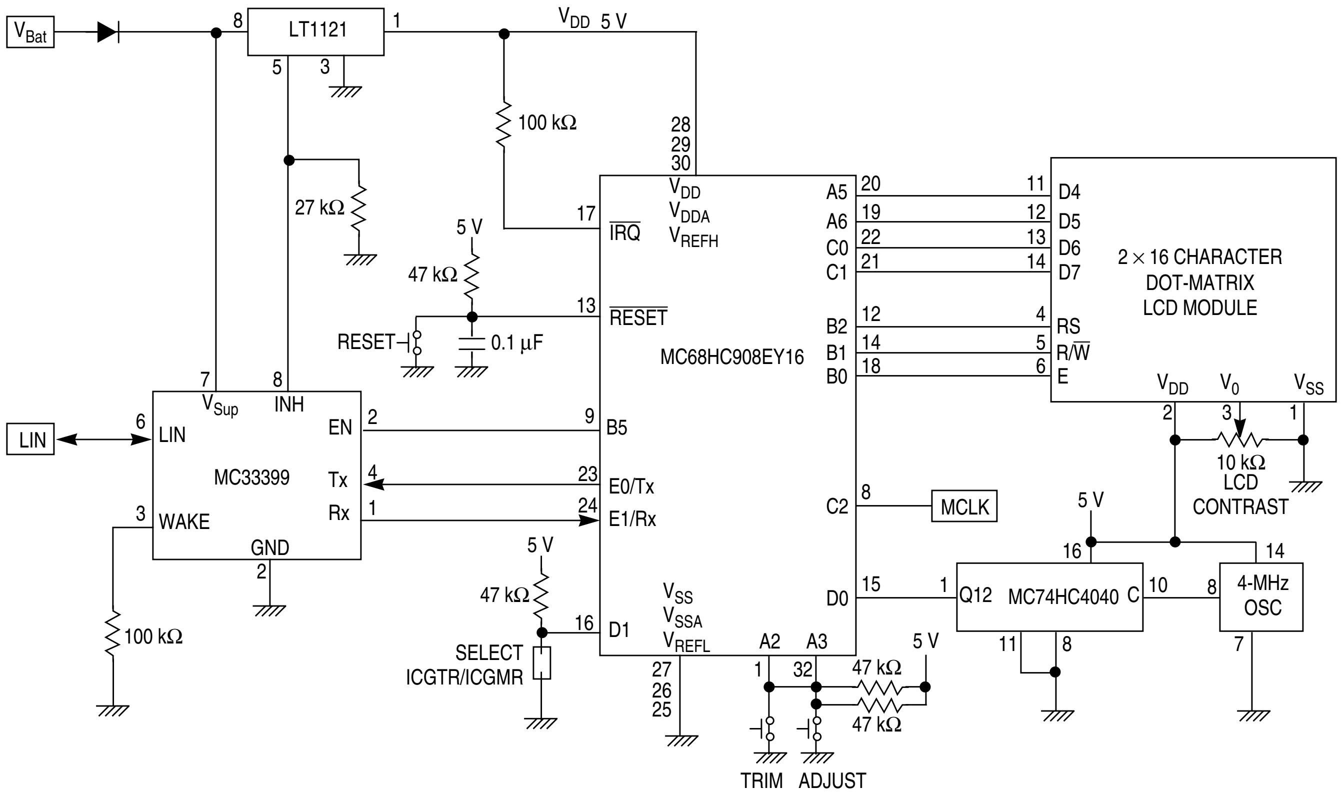
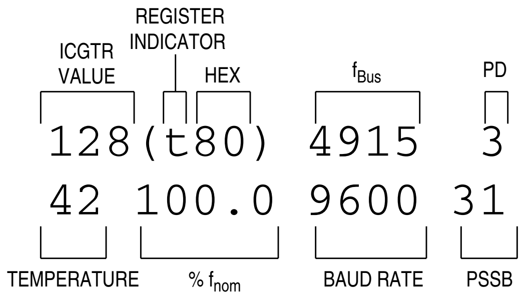


Figure 2. Circuit Diagram of ESCI Baud-Rate Adjustment Application

1. See the **References** section for a list of helpful documents, including the application notes cited here.

To supply the required calibration pulse to the MC68HC908EY16's PTD0 timer pin, some extra circuitry built onto a plug-in top board is used. The LINkits evaluation boards incorporate two 20-pin connectors using roughly the same pinout on all the boards. This allows the design of a top board that can be used on any LINkits board. In this case, a 4-MHz oscillator has been fitted to the top board. Its output is divided by an MC74HC4040 12-stage ripple counter to give the required 976.5625-kHz signal (1024 µs period).

The format of the LCD displayed data is shown below. The top line shows the current value of ICGTR in decimal and hexadecimal. (The hexadecimal value is in brackets.) The "t" indicates that ICGTR is being displayed. (This display can be changed to ICGMR if PTD1 is held low. In that case the "t" would be replaced with an "m.") The actual bus frequency is displayed on the top line. MCLK is enabled on PTC2 as an external check of this frequency. At the right-hand end of the top line, the integral part (PD) of the current value of the fractional divide prescaler is shown.



The bottom line of the display shows the data from a sample LIN message (see AN2264/D listed in the **References** section). The two-digit temperature is displayed on the LCD if it is correctly received. However, a default display of "**" is shown if the LIN function *LIN_MsgStatus()* does not recognize the presence of the message. (This happens if the baud rate of the MCU is not accurate enough for the LIN protocol to function correctly.) This data is followed by the bus frequency. It is expressed as a percentage of the nominal frequency that would be expected using the current value of ICGMR. The 16 most recent measurements of the external pulse are stored in an array. The displayed percentage is the average of these values (shown with a resolution of 0.1%). The next number on the bottom line is the internally calculated baud rate of the ESCI. The baud rate will change when the bus frequency is changed (ICGTR is modified). It soon readjusts to 9600 baud (within +150/–20 baud) as the ESCI fractional divide prescaler register values are automatically adjusted by the LIN drivers. At the right-hand end of the bottom line, the fractional part (PSSB) of the current value of the fractional divide prescaler is shown.

In addition to the reset button, two buttons (trim and adjust) are also included, as shown in **Figure 2**. They perform the same functions as those described in AN2498/D (see **References**). Pressing the trim button causes a trim calculation and adjustment to take place. This function is debounced so that only a single trim takes place even if the button is held down. The adjust button can be used to manually change the value of ICGTR. It decrements this register and does so repeatedly if it is held down. If, when the adjust button is pressed, the trim button is also pressed, ICGTR is incremented. The software incorporates an interlock so that the buttons can be released in either order without causing an inadvertent trim operation that would corrupt the deliberately modified value in ICGTR. If PTD1 is held low using the jumper shown, ICGMR is displayed instead of ICGTR and the incrementing and decrementing functions of the buttons are applied to ICGMR instead of ICGTR. This arrangement allows any value of either register to be achieved and, if desired, a trim operation performed from that configuration. However, for the purposes of demonstrating the baud rate adjustment, only a single trim is required and the value of ICGMR should not be changed from its initially programmed value of 64.

Software

The software for the ESCI baud rate adjustment demonstration application was written and debugged using the LINkits evaluation PCB and the Metrowerks' CodeWarrior® development environment with a P&E Multilink interface. The application incorporates the Motorola LIN drivers (1.3) whose baud rate adjustment method is described in this application note.

The main software flow chart is **Figure 3**. After the CONFIG and port registers have been initialized, ICGMR is given the value 64. This gives the nominal bus speed of 4.9152 MHz. The LIN drivers are configured (in file slave.cfg) for 9600 baud. The time base module is programmed with its maximum divide ratio of 4,194,304 to give a slow (4 Hz) repetition rate through the loop. This was done because the LCD is written to at this rate and there is no reason to update it at a rate faster than the changing digits can be read. Timer A channel 0 is then set up to measure the time between successive rising edges of the external reference signal coming from the MC74HC4040. The display module and the LIN drivers are then initialized and interrupts enabled.

The main loop is timed by polling the time base module's interrupt/overflow flag. The buttons on bits 2 and 3 of port A are then read to determine whether a trim operation or register adjustment is being requested. This is handled by the function *Read_buttons* and incorporates a simple debounce and interlock using the flag *bounce*. This flag prevents multiple trims from occurring if the trim button is held down. It also prevents a trim if both buttons are pressed to increment ICGTR or ICGMR and are then released in such a way that the trim button is held down after the adjust button has been released.

The LIN message is then read using function *Read_LINtemp()*. This is similar to the code used in AN2264/D (see **References**) and the format of the message is described in detail there. In this application, the message is only being used to indicate whether the baud rate is accurate enough for the LIN protocol to function correctly. Because of the low repetition rate through the main loop, the buffer may be read less often than the arrival rate of the message on the LIN bus. This is why a *LIN_MsgStatus(0x0A)* return of *LIN_MSG_OVERRUN* is regarded as normal, along with *LIN_OK.*

The main loop then performs the calculation of the constant cnt1024. This is a constant for a given value of ICGMR (5033 for an ICGMR of 64 in this application) but is calculated each time around the loop in case ICGMR has been changed using the adjust button. This is followed by the two functions *Format_line1()* and *Format_line2()*, which convert the various numbers that must be displayed into the ASCII format required for the LCD module. Adding 0x30 is all that is required for the decimal numbers. The array *ASCIIconv[16]* is used to convert hexadecimal digits. The averaging calculations for the bottom line display are performed within *Format_line2()*.

The last function performed within the loop is actually writing the data for display to the LCD module using the function *Display_Data(data, mode)*. This function in turn uses functions *LCD_busy()*, *Write_Nibble(data)*, and *Clock_LCD()*. *Write_Nibble(data)* is required because of the somewhat inconvenient choice of port lines used to drive the LCD. (They were chosen to be consistent with AN2498/D).

The interrupt service routine *TimerA()* uses channel 0 as an input capture to read the number of timer counts between successive rising edges of the external timing reference. The difference between the current value and the previous value is calculated. Then that delta is put into the array of 16 values that are used for the average frequency display.

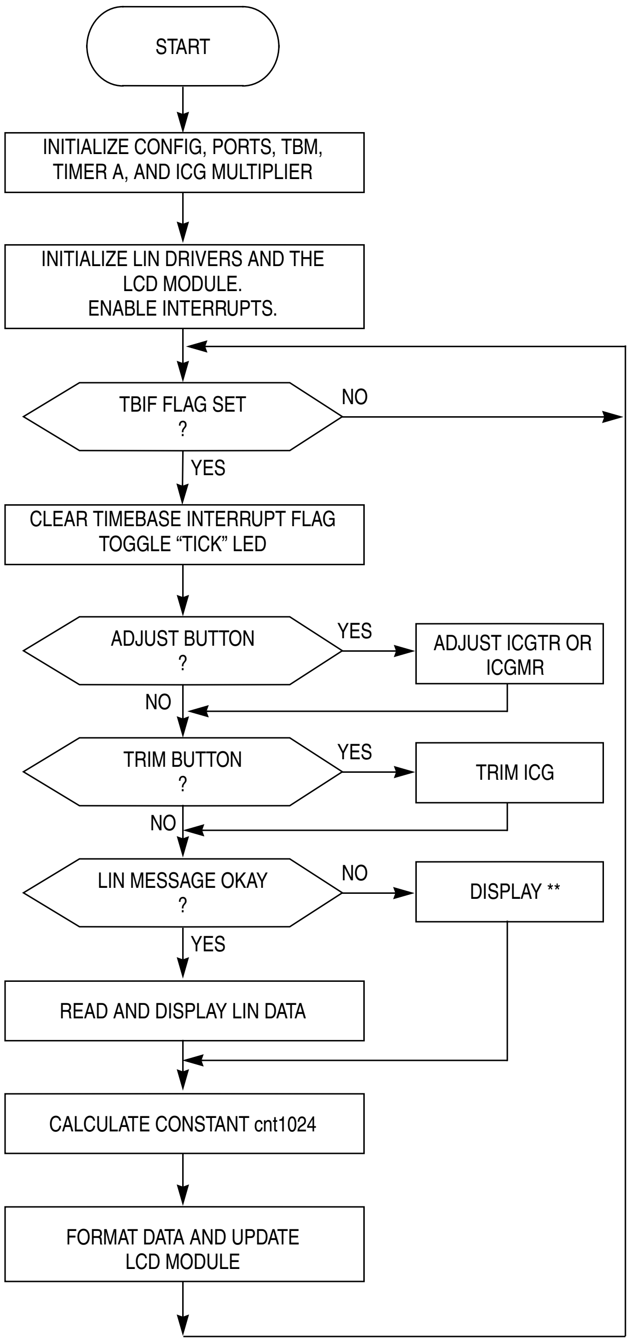

START
INITIALIZE CONFIG, PORTS, TBM, TIMER A, AND ICG MULTIPLIER
INITIALIZE LIN DRIVERS AND THE LCD MODULE. ENABLE INTERRUPTS.
TBIF FLAG SET ?
NO
YES
CLEAR TIMEBASE INTERRUPT FLAG TOGGLE “TICK” LED
ADJUST BUTTON ?
YES
ADJUST ICGTR OR ICGMR
NO
TRIM BUTTON ?
YES
TRIM ICG
NO
LIN MESSAGE OKAY ?
NO
DISPLAY **
YES
READ AND DISPLAY LIN DATA
CALCULATE CONSTANT cnt1024
FORMAT DATA AND UPDATE LCD MODULE

Figure 3. Flow Chart of Main Software Loop

Alternative Strategies

1. In the current implementation, FD reverts back to 4 before the reception of each message. As an alternative, it would be possible to retain the adjusted value of FD and change it only when required (e.g., due to temperature change causing a drift in the ICG frequency). To do this without risk of getting stuck on an incorrect value, the range of FD should be limited to ±14% of its nominal value (i.e., if the nominal is 4., the acceptable FD range is between $3\ ^{14}/_{32}$ and $4\ ^{18}/_{32}$).
2. To incorporate other bus frequencies and baud rates, leave BPD at 1 and select the most appropriate values of BD and nominal FD to suit the required combination.

   Let R = bus clock ÷ (64 × baud rate), divide R by 2^N so that $2 < R \div 2^N < 4$

   Then BD = N and $FD_{nom} = R \div 2^N$

   Calculate $x = (R \div 2^N - 2) \div 32$ to nearest integer to get $FD_{nom} = PS + 1 = 2\ ^x/_{32}$

   When the arbiter count is being transferred to SCPCS, it should be shifted to the right N times.
3. If the methods described in 1 and 2 above are both used, FD should be limited to the following values:

   $SCPSC_{min} = INT\ (0.86 \times (SCPSC_{nom} + 32) - 32)$

   $SCPSC_{max} = INT\ (1.14 \times (SCPSC_{nom} + 32) - 32)$
4. Avoid uncontrolled overflow of the arbiter counter. Although the overflow bit can be used as a tenth bit, it is safer to limit the maximum acceptable count to 511 (nine bits) and leave the overflow bit for use as an indication of an error. The maximum count = 1.1 × nominal $f_{Bus} \div 2 \times$ baud rate. At 9600 baud, this gives a count of 458 at a bus clock of 8 MHz. This is acceptable, but is near the limit. Therefore, the use of lower baud rates necessarily implies the use of lower bus speeds.

References

[1] LIN Protocol Specification, Version 1.3, 12 December 2002.

[2] AN2498/D, *Initial trimming of the MC68HC908 ICG*.

[3] MC68HC908EY16/D, *MC68HC908EY16 Technical Data Sheet*.

[4] AN2264/D, *LIN Node Temperature Display*.

[5] AN2344/D, *HC908EY16 EMI Radiated Emissions Results*.

[6] AN2573/D, *LINkits LIN Evaluation Boards.*

Software Listing

```
/*****************************************************************************
*                (c) MOTOROLA Inc. 2003  all rights reserved.                *
*                                                                            *
*                                                                            *
*         MC68HC908EY16 ESCI baud rate trimming demonstration program.       *
*         ============================================================       *
*                                                                            *
*    Originator:    P. Topping                                               *
*    Date:          22nd June 2003                                           *
*    Revision:      1.0                                                      *
*    Function:      LCD display of the ESCI prescaller settings which are    *
*                   automatically adjusted by the LIN drivers (1.3), The     *
*                   code includes the ability to adjust or trim the ICG to   *
*                   force changes to the fractional divide prescaler value.  *
*                   The hardware is based in the EY16 "LINkits" eval.board.  *
*                                                                            *
*****************************************************************************/

#pragma DATA_SEG SHORT _DATA_ZEROPAGE

/*****************************************************************************
*
*    Header file includes
*
*****************************************************************************/

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


/*****************************************************************************
*
*    Function prototypes
*
*****************************************************************************/

void Initialise_Display (void);
void Display_Data (unsigned char, unsigned char);
void Write_Nibble (unsigned char);
void Clock_LCD (void);
unsigned char LCD_busy (void);
void Read_buttons (void);
void Format_line1 (void);
void Format_line2 (void);
void Read_LINtemp (void);
```

```
/*****************************************************************************
*
*    Global variables
*
*****************************************************************************/

unsigned char LIN_data[2];
unsigned char error_count = 3;
unsigned char ASCIIconv[] = {48,49,50,51,52,53,54,55,56,57,65,66,67,68,69,70};
unsigned char Line1[] = "---(---) ----  -";
unsigned char Line2[] = "-- ---.- ---- --";
unsigned char Power_of2 [8] = {1, 2, 4, 8, 16, 32, 64, 128};
unsigned char Prescaler [8] = {1, 3, 4, 13};
unsigned char count;
unsigned char tick;
unsigned char bounce;
unsigned char bpoint;
int delta_buffer[16];
int tcount0;
int delta0;
long thousand = 1000;
long cnt1024;
int bus_freq;
unsigned char PD;
unsigned char tmpSCPSC;
```

```
/*******************************************************************************
*
*    Function name: Main
*    Originator:    P. Topping
*    Date:          21th March 2003
*    Function:      Initialise CONFIG registers, ICGMR, Ports and Timers.
*                   Initialise LCD module & LIN drivers and enable interrupts.
*                   Pace a slow loop (~4Hz for a 4MHz bus) using the timebase
*                   module. Within this loop the keys are read, the LIN buffer
*                   is read and the LCD module is updated. If requested by
*                   pressing the PTA2 key, a trimming operation is performed.
*                   ICGTR or ICGMR are incremented or decremented (according to
*                   the level of PTA2) once per loop if the PTA3 key is pressed
*                   PTD1 selects ICGTR or ICGMR for display and inc./dec.
*
*******************************************************************************/

void main (void)
{
    CONFIG1 = 0x01;                             /* dissable COP             */
    CONFIG2 = 0x45;                             /* slow clock for TBM       */
    ICGMR = 64;                                 /* ICG nominal 15.9744 MHz */
    DDRA = 0x70;                                /* port A 5-6, port B 0-2   */
    DDRB = 0x27;                                /* and port C 0-1 for LCD   */
    DDRC = 0x83;                                /* port B bit 5 for LPI     */
    PTB  = 0x20;                                /* enable MC33399 LPI       */
    TBCR = 0x00;                                /* divide by 2**22: ~4Hz @ */
    TBCR = 0x02;                                /* 4MHz bus & enable TBM    */
    TASC  = 0x30;                               /* reset timer A            */
    TASC0 = 0x44;                               /* IC TAch0 - rising edges */
    TASC  = 0x00;                               /* and start timer A        */

    Initialise_Display();                       /* initialise LCD module    */
    asm cli;                                    /* enable interrupts        */
    LIN_Init();                                 /* initialise LIN drivers   */

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

        tick++;
        if (tick & 0x01)                        /* check bit 0 of tick      */
        {
            PTA |= 0x10;                        /* tick LED (PTA4) off      */
        }
        else
        {
            PTA &= ~(0x10);                     /* tick LED (PTA4) on       */
        }
        Read_buttons();                           /* read buttons on A2 & A3 */

        Read_LINtemp();                           /* check for LIN message   */

        cnt1024 = (ICGMR * 307.2 * 256) / 1000;   /* constant for this ICGMR */

        Format_line1();                           /* data for LCD line 1     */

        Format_line2();                           /* data for LCD line 2     */

        Display_Data(0x80, 0x00);                 /* DDRAM address to line 1 */
        for (count = 0; count < 16; count++)
        {
            Display_Data(Line1[count], 0x01);     /* and write LCD line 1    */
        }

        Display_Data(0xC0, 0x00);                 /* DDRAM address to line 2 */
        for (count = 0; count < 16; count++)
        {
            Display_Data(Line2[count], 0x01);     /* and write LCD line 2    */
        }
    }
}
}
```

```
/******************************************************************************
*
*    Function name: Read_buttons
*    Originator:    P. Topping
*    Date:          26th March 2003
*    Function:      PTA3 PTA2 PTD1     PTA2 & 3 are low if their key is pressed
*                     1    1    x      no key function (release debounce lock)
*                     1    0    x      calculate trim and adjust ICGTR
*                     0    1    0      decrement multiplier register ICGMR
*                     0    1    1      decrement trim register ICGTR
*                     0    0    0      increment multiplier register ICGMR
*                     0    0    1      increment trim register ICGTR
*
******************************************************************************/

void Read_buttons (void)
{
    if ((PTA & 0x08) == 0)                          /* PTA3 key pressed ?      */
    {
        if ((PTA & 0x04) == 0)                      /* yes, PTA2 key pressed ? */
        {
            if (PTD & 0x02)                         /* yes, check PTD1         */
            {
                ICGTR += 1;                         /* inc. trim if high       */
            }
            else
            {
                ICGMR += 1;                         /* or multiplier if low    */
            }
        }
        else                                        /* PTA3 but not PTA2       */
        {
            if (PTD & 0x02)                         /* check PTD1              */
            {
                ICGTR -= 1;                         /* decrement trim if high  */
            }
            else
            {
                ICGMR -= 1;                         /* or multiplier if low    */
            }
        }
        bounce = 1;                                 /* inhibit trimming        */
    }
    else                                            /* PTA3 key not pressed    */
    {
        if (((PTA & 0x04) == 0) && (bounce == 0))  /* PTA2 key pressed ?      */
        {
            ICGTR += (512*(delta0-cnt1024))/cnt1024;         /* yes, trim    */
            bounce = 1;                             /* and inhibit repeat      */
        }
        else if ((PTA & 0x04) != 0)                 /* neither pressed so      */
        {
            bounce = 0;                             /* re-enable trimming      */
        }
    }
}
```

```
/***************************************************************************
*
*    Function name: Format_line1
*    Originator:    P. Topping
*    Date:          26th March 2003
*    Function:      Format display of ICGTR (or ICGMR) in decimal and hex
*                   and bus frequency in kilohertz (blanking a leading zero).
*                   PTD1 selects between the display of ICGTR or ICGMR
*
***************************************************************************/

void Format_line1 (void)
{
int remain;

    if (PTD & 0x02)                              /* check PTD1              */
    {
        remain = ICGTR;                          /* high so display ICGTR   */
        Line1[4] = 0x74;                         /* "t"                     */
    }



    else
    {
        remain = ICGMR;                          /* low so display ICGMR    */
        Line1[4] = 0x6D;                         /* "m"                     */
    }
    Line1[5] = ASCIIconv [remain/16];            /* upper nibble in HEX     */
    Line1[6] = ASCIIconv [remain & 0x0F];        /* lower nibble in HEX     */
    for (count = 3; count != 0; count--)
    {
        Line1[count-1] = (remain%10) + 0x30;     /* display also in decimal */
        remain = remain/10;
    }
    if (Line1[0] == 0x30) { Line1[0] = 0x20; }   /* leading 0 becomes space */

    bus_freq = (delta0*thousand)/1024;           /* calc. freq. from delta  */
    remain = bus_freq;
    for (count = 12; count >= 8; count--)
    {
        Line1[count] = (remain%10) + 0x30;       /* display frequency       */
        remain = remain/10;
    }
    if (Line1[8] == 0x30) { Line1[8] = 0x20; }   /* leading 0 becomes space */

    if (SCPSC != 0x60) tmpSCPSC = SCPSC;
    PD = 1 + tmpSCPSC/32;                        /* ESCI prescaller reg.    */
    Line1[15] = PD + 0x30;                       /* upper three bits (+1)   */
}
```

```
/******************************************************************************
*
*    Function name: Format_line2
*    Originator:    P. Topping
*    Date:          26th March 2003
*    Function:      Format display of average frequency as a percentage of the
*                   nominal frequency (0.1% resolution) and baud rate.
*                   (the LIN temperature display is added by Read_LINtemp())
*
******************************************************************************/

void Format_line2 (void)
{
int remain;
long total;
volatile unsigned char BPD;
volatile unsigned char BD;
unsigned char PDFA;

    total = 0;
    for (count = 0; count < 16; count++)
    {
        total += delta_buffer[count];             /* sum 16 deltas           */
    }
    remain = (thousand * total)/16/cnt1024;       /* average as % of nominal */
    Line2[7] = (remain%10) + 0x30;                /* display tenths          */
    for (count = 3; count != 0; count--)
    {
        remain = remain/10;
        Line2[count+2] = (remain%10) + 0x30;      /* and percentage          */
    }
    if (Line2[3] == 0x30) { Line2[3] = 0x20; }    /* leading 0 becomes space */
    PDFA = tmpSCPSC & 0x1F;                       /* ESCI pre. lower 5 bits  */
    remain = PDFA;                                /* ESCI pre. lower 5 bits  */
    Line2[15] = (remain%10) + 0x30;               /* display PSSB in decimal */
    remain = remain/10;
    Line2[14] = remain + 0x30;
    if (Line2[14] == 0x30) Line2[14] = 0x20;      /* leading 0 becomes space */

    BPD = Prescaler [(SCBR/16) & 3];
    BD =  Power_of2 [(SCBR & 7)];
    remain = (bus_freq * thousand)/(2 * BPD * BD * (PD*32 + PDFA));
    for (count = 12; count >= 8; count--)
    {
        Line2[count] = (remain%10) + 0x30;        /* display baud rate       */
        remain = remain/10;
    }
    if (Line2[8] == 0x30) { Line2[8] = 0x20; }    /* leading 0 becomes space */
}
```

```
/******************************************************************************
*
*    Function name: Read_LINtemp
*    Originator:    P. Topping
*    Date:          26th March 2003
*    Function:      Check for LIN message, if there decode and put temperature
*                   into display array, if not enter **
*
******************************************************************************/

void Read_LINtemp (void)
{
unsigned char bits;
unsigned char units;
unsigned char tens;
unsigned char negative;

    bits = LIN_MsgStatus (0x0A);

    if ((bits != LIN_OK) && (bits != LIN_MSG_OVERRUN))   /* has there been   */
    {                                           /* an ID 0A message ?      */
        if (error_count < 5 )                   /* no, error counter       */
        {                                       /* already 5 ?             */
            error_count ++;                     /* no, inc. error counter  */
        }
    }
    else
    {
        error_count = 0;                        /* yes, new data available */
    }

    if (error_count > 4)                        /* data in last second ?   */
    {                                           /* (needs approx 4MHz bus) */
        Line2[2] = 0x20;                        /* no, display "** "       */
        Line2[0] = 0xA5;                        /* to signify no valid     */
        Line2[1] = 0xA5;                        /* LIN data available      */
    }
    else                                        /* yes, data available     */
    {
        LIN_GetMsg (0x0A, LIN_data);            /* read sensor message     */
        bits = LIN_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     */
        }
        bits = bits/2;                          /* lose LS bit            */
        tens = bits/10;                         /* find tens digit        */
        units = bits%10;                        /* and units digit        */

        Line2[1] = units + 0x30;                /* units digit            */
        Line2[0] = 0x20;                        /* clear tens digit       */
        Line2[2] = 0x20;                        /* and "-"                */

        if (tens != 0)                          /* tens digit zero ?      */
        {
            Line2[0] = tens + 0x30;             /* no, display it         */
            if (negative)                       /* negative ?             */
            {
                Line2[2] = 0xB0;                /* yes display "-"        */
            }
        }
        else if (negative)                      /* tens zero, negative ?  */
        {
            Line2[0] = 0xB0;                    /* yes, put "-" in tens   */
        }
    }
}
```

```
/*******************************************************************************
Function Name    :   Display_Data
Engineer         :   C. Culshaw
Date             :   06/09/02
Parameters       :   data - byte to be written to display
                 :   regsel - RS (0 = command mode, 1 = data mode)
*******************************************************************************/

void Display_Data (unsigned char data, unsigned char regsel)
{
    while (LCD_busy() == 0x01);                     /* read LCD busy status    */

    if (regsel == 1)
    {
        PTB |= 0x04;                                /* data, RS high           */
    }
    else
    {
        PTB &= ~(0x04);                             /* command, RS low         */
    }

    Write_Nibble (data/16);                         /* MS nibble               */
    Clock_LCD();                                    /* clock display           */
    Write_Nibble (data & 0x0F);                     /* LS nibble               */
    Clock_LCD();                                    /* clock display           */
}


/*******************************************************************************
*
*    Function name: Write_Nibble
*    Originator:    P. Topping
*    Date:          21st March 2003
*    Parameters:    nibble - 4-bit data to be written to display
*
*******************************************************************************/

void Write_Nibble (unsigned char nibble)
{
    PTA = (PTA & 0x9F) | nibble * 0x20;             /* format bits for LCD     */
    PTC = (PTC & 0xFC) | nibble / 4;                /* interface               */
}
```

```
/*****************************************************************************
*
*    Function name: LCD_busy
*    Originator:    P. Topping
*    Date:          10th February 2003
*
*****************************************************************************/

unsigned char LCD_busy (void)
{
    unsigned char busy;
    unsigned char wait;

    DDRA &= ~(0x60);                              /* make LCD data pins MCU  */
    DDRC &= ~(0x03);                              /* inputs                  */

    PTB &= ~(0x04);                               /* RS low                  */
    PTB |= 0x02;                                  /* RW high                 */
    PTB |= 0x01;                                  /* E high                  */
    busy = (PTC & 0x02);                          /* check busy output       */
    PTB &= ~(0x01);                               /* E low                   */
    Clock_LCD();                                  /* clock second nibble     */
    PTB &= ~(0x02);                               /* RW low                  */

    DDRC |= 0x03;                                 /* put LCD data pins back  */
    DDRA |= 0x60;                                 /* to MCU outputs          */

    wait = 0;  while (wait++ < 10);               /* wait ?                  */
    return (busy);
}
```

```
/******************************************************************************
*
*    Function name: Clock_LCD
*    Originator:    P. Topping
*    Date:          10th February 2003
*
******************************************************************************/

void Clock_LCD (void)
{
    PTB |= 0x01;                                /* E high                */
    asm NOP;                                    /* slow down for > 8MHz  */
    asm NOP;
    PTB &= ~(0x01);                             /* E low                 */
}


/******************************************************************************
*
*    Function name: Initialise_Display
*    Originator:    P. Topping
*    Date:          10th February 2003
*
******************************************************************************/

void Initialise_Display (void)
{

PTB &= ~(0x06);                                 /* RW and RS low         */
count = 0;

while (count < 8)
{
    if (TBCR & 0x80)                            /* is TBM flag set?      */
    {
        TBCR |= 0x08;                           /* yes, clear it         */

        switch (count)
        {
            case 1:
                Write_Nibble (0x03);            /* Function set (8 bits)  */
                Clock_LCD();                    /* and clock             */
                break;
            case 2:
                Clock_LCD();                    /* wait and clock again  */
                break;
            case 3:
                Clock_LCD();                    /* and again             */
                break;
            case 4:
                Write_Nibble (0x02);            /* Function set 4 bit mode */
                Clock_LCD();
                break;
            case 5:
                Display_Data (0x28, 0x00);      /* 2 line display        */
                break;
            case 6:
                Display_Data (0x08, 0x00);      /* display off           */
                break;
            case 7:
                Display_Data (0x0C, 0x00);      /* display on            */
                PTB |= 0x04;                    /* RS high               */
                break;
        }
        count++;
    }
}
}
```

```
/*****************************************************************************
*
*    Function name: TimerA0
*    Originator:    P. Topping
*    Date:          19rd March 2002
*    Function:      Timer A, channel 0 interrupt service routine
*                   Read timer, subtract from previous value and save delta
*                   in global "delta0" and in 16 result array "delta_buffer[]"
*
*****************************************************************************/

#pragma TRAP_PROC

void TimerA0 (void)
{
    unsigned char thigh;
    int tcount;

    TASC0 &= ~(0x80);                           /* clear interrupt flag    */
    thigh = TACH0H;                             /* read high byte first    */
    tcount = ((thigh*256) + TACH0L);            /* update counter          */
    delta0 = tcount - tcount0;                  /* calculate delta         */
    tcount0 = tcount;                           /* save timer value        */

    delta_buffer[bpoint & 0x0F] = delta0;       /* put delta into buffer   */
    bpoint++;                                   /* of 16 for averaging     */
}


/*****************************************************************************
* 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)
    {
    }
}
```

HC08EY16.h

```
/*****************************************************
  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 SCBR  *((volatile unsigned char *)0x0016)
#define SCPSC *((volatile unsigned char *)0x0017)

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

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

#define TASC   *((volatile unsigned char *)0x0020)
#define TACNTH *((volatile unsigned char *)0x0021)
#define TACNTL *((volatile unsigned char *)0x0022)
#define TAMODH *((volatile unsigned char *)0x0023)
#define TAMODL *((volatile unsigned char *)0x0024)
#define TASC0  *((volatile unsigned char *)0x0025)
#define TACH0H *((volatile unsigned char *)0x0026)
#define TACH0L *((volatile unsigned char *)0x0027)
#define TASC1  *((volatile unsigned char *)0x0028)
#define TACH1H *((volatile unsigned char *)0x0029)
#define TACH1L *((volatile unsigned char *)0x002A)

#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 DDIV   *((volatile unsigned char *)0x0039)
#define DSTG   *((volatile unsigned char *)0x003A)

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

vector.c

```
#define VECTOR_C
/******************************************************************************
*
*       Copyright (C) 2001 Motorola, Inc.
*
* Functions:    Vectors table for LIN08 Drivers with Motorola 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 TimerA0();                  /* Timer Module A Channel 0 ISR  */
extern void TimerA1();                  /* Timer Module A Channel 1 ISR  */
// extern void TimerB();                /* Timer Module B Overflow 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 2L31N mask set of the         */
/*      MC68HC908EY16 and all subsequent versions.                          */
/*                                                                          */
/*      Older mask sets, e.g. 0L38H, 1L38H, 0L31N and 1L31N had a fault     */
/*      in their interrupt vector table and hence in the priorities.        */
/*      For these older mask sets the order of the SCI vectors was:         */
/*                                                                          */
/*      SCI_Error_ISR,                      // 0xFFE6   ESCI error          */
/*      SCI_Transmit_ISR,                   // 0xFFE8   ESCI transmit       */
/*      SCI_Receive_ISR,                    // 0xFFEA   ESCI receive        */
/*                                                                          */
/*      All other vectors are unchanged.                                    */
/*                                                                          */
/****************************************************************************/

{
    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            */

#if defined(MASTER)                         /* (used for Master node only)*/
    LIN_VECTF LIN_ISR_SCI_Transmit,         /* 0xFFE6   ESCI transmit       */
#endif /* defined(MASTER) */
#if defined(SLAVE)
    LIN_VECTF NULL,                         /* 0xFFE6   ESCI transmit       */
#endif /* defined(SLAVE) */
    LIN_VECTF LIN_ISR_SCI_Receive,          /* 0xFFE8   ESCI receive        */
    LIN_VECTF LIN_ISR_SCI_Error,            /* 0xFFEA   ESCI error          */

    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   */
    TimerA0,                                /* 0xFFF6   TIMER A channel 0   */
    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) */
```

slave.cfg

```
#ifndef LINCFG_H
#define LINCFG_H

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

#if defined (HC08)


/*  This definition configures the ESCI Prescaler divide ratio    */

#define LIN_SCIPRESCALER            0x60u         /* divide by 4  */


/*  This definition configures the LIN bus baud rate. This value  */
/*  shall be set according to target MCU SCI register usage.      */
/*  HC08EY16: the 8-bit SCBR value will be masked by 0x37.        */
/*  The following numbers assume that the ESCI prescaller = 4     */

/* Selects 9600 baud for a nominal 2.4576 MHz clock (ICGMR=32)    */
//#define LIN_BAUDRATE            0x00u

/* Selects 9600 baud for a nominal 4.9152 MHz clock (ICGMR=64)    */
#define LIN_BAUDRATE            0x01u


/* Enable ESCI (fractional divide prescaler) baudrate synch.      */

#define LIN_SYNC_SLAVE

/* The following numbers assume that the ESCI prescaller = 1      */

/* Selects 9600 baud rate if using a 4.9152MHz crystal            */
//#define LIN_BAUDRATE            0x03u

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

/* Selects 9600 baud rate if using an 8.000MHz crystal            */
//#define LIN_BAUDRATE            0x30u

/* Selects 9600 baud rate if using a 16.000MHz crystal            */
//#define LIN_BAUDRATE            0x31u

/* Selects 9600 baud rate if using a 32.000MHz crystal            */
//#define LIN_BAUDRATE            0x32u


/*  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) */
```

slave.id

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

#define LIN_MSG_0A  LIN_RECEIVE

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

#define LIN_MSG_0A_LEN  2       /* standard length */

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

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:

Motorola Literature Distribution;
P.O. Box 5405, Denver, Colorado 80217
1-303-675-2140 or 1-800-441-2447

JAPAN:

Motorola Japan Ltd.; SPS, Technical Information Center,
3-20-1, Minami-Azabu Minato-ku, Tokyo 106-8573 Japan
81-3-3440-3569

ASIA/PACIFIC:

Motorola Semiconductors H.K. Ltd.;
Silicon Harbour Centre, 2 Dai King Street,
Tai Po Industrial Estate, Tai Po, N.T., Hong Kong
852-26668334

TECHNICAL INFORMATION CENTER:

1-800-521-6274

HOME PAGE:

http://www.motorola.com/semiconductors

MOTOROLA