Freescale Semiconductor
Application Note

Document Number: AN2498
Rev. 1, 01/2007

Initial Trimming of the MC68HC908 ICG

by: Peter Topping
East Kilbride

1 Introduction

The incorporation of an ICG (Internal Clock Generator) in the MC68HC908EY16 and the MC68HC908KX8 variants of the HC08 MCU family avoids the need for any external clock components. There is no need for a crystal, resonator, or any passive components. This reduces cost and avoids the need to dedicate any pins to clock circuitry. The ICG also provides the versatility of a PLL in that the clock frequency is programmable over a wide range (307.2kHz to 32MHz in multiples of 307.2kHz) and can change as required during normal code execution. As with all HC08s, the bus frequency is a quarter of this clock frequency. The ICG module optionally allows the use of a crystal or an external clock as an alternative to the internal clock described here.

The downside of utilising the internal clock is that the frequency is not as accurate as that of a crystal or resonator and this limitation has to be taken into account in the system design. Parametric spreads during manufacture result in an overall untrimmed accuracy of

Contents

freescale™ semiconductor

±25% from the specified nominal frequency. In many applications, this may be good enough and no trimming will be required. Sometimes, however, greater accuracy will be required and this application note describes a method of performing initial trimming to improve the accuracy to within ±0.3% [1] at a given V_{DD} and temperature.

The method described uses an external reference signal as shown in the block diagram in Figure 1. The application also includes an LCD module to display the bus frequency, its accuracy, and a simple LIN message (Local Interconnect Network [1]). LIN functionality is included as an example of a typical application that requires a more accurate clock frequency than that of the untrimmed ICG. While the trimming operation could occur at any time to achieve improved accuracy, it is envisaged that the external signal would not always be available. The operation would normally happen only once with the resultant trimming information stored in FLASH for use each time the application powers up. The frequency is then guaranteed to be within ±7% for 4.5 to 5.5 volts V_{DD} and –40°C to +85°C (±10% for the full automotive temperature range). Using a voltage regulator and a limited temperature range can assure overall accuracies of ±1% or better are achievable in many applications.

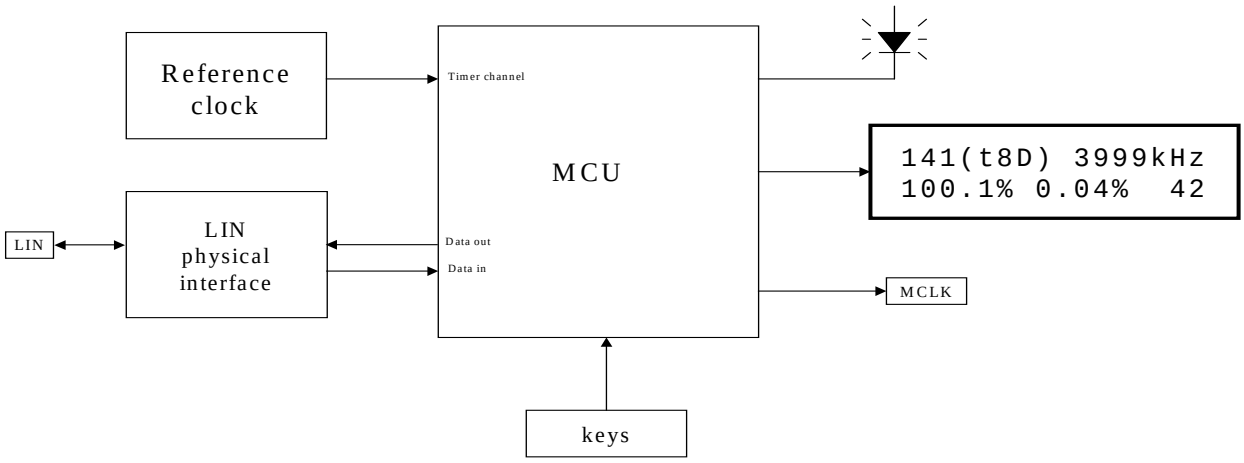

Figure 1. Block Diagram of ICG Trimming Application

Internal oscillator trimming is currently performed during factory testing of MC68HC908QT/QY MCUs. The resultant trim byte is stored in FLASH at address $FFC0. Once volume production starts, this will also be the case with the MC68HC908EY16 (the address has still to be decided but it may also be $FFC0). If this has been done, it is up to the user to leave this byte in FLASH during application code programming or, if an erase is employed, to retrieve it and reprogram the byte into FLASH along with the main code. It is planned that programmers will perform this process automatically. If this has been done, and the data has not been lost during programming, the user would not need to perform the initial trim described here. However, if this operation has not been performed (as is the case with the 908KX8 and current 908EY16 product), if the value has been lost or if the user requires to re-trim at a non-standard voltage or temperature, then trimming will be necessary if +/-25% accuracy is not adequate.

1. Although the specified resolution of 0.195% implies the possibility of a 0.1% trimming accuracy, non-linearities within the ICG render 0.3% a more realistic figure.

In applications where improved accuracy is required over the full temperature range (e.g. 2% for LIN or other applications involving UART asynchronous serial communication), regular trimming may be necessary during normal operation. If there is no explicit external reference available, then a reference for the ongoing trimming may be derived from the serial communication itself. This second level of clock or baud rate trimming is not covered here.

2 The ICG

The ICG incorporates a digitally controlled oscillator in a control loop as shown in Figure 2. An understanding of the operation of the digitally controlled oscillator itself is not necessary in order to use the ICG effectively but for completeness it is described in a later section. It uses registers DDIV and DSTG that are handled by the ICG hardware to make automatic corrections to the frequency of the clock. This activity shouldn't be confused with the user's manipulation of the ICGTR and ICGMR registers to determine what that frequency should be. To use the ICG it is only necessary to remember that the digitally controlled oscillator outputs a signal at the frequency used as the MCU's clock. Like the VCO in a PLL, it is divided down to a lower frequency before a comparison with a reference is made.

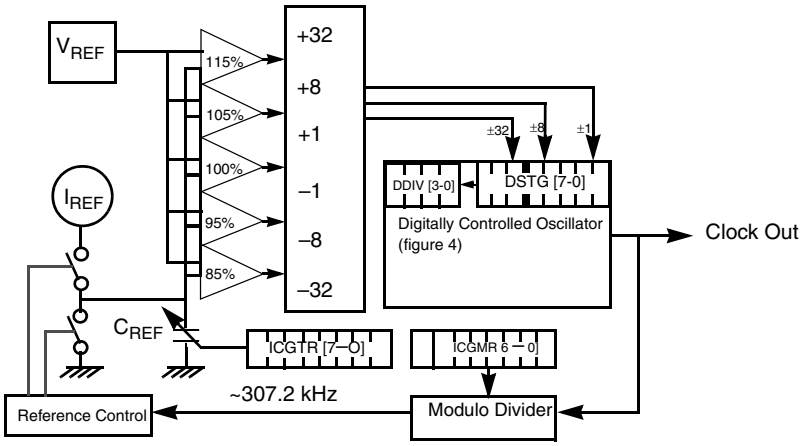

Figure 2. The ICG Control Loop

In the ICG, this division ratio is determined by the multiplier register, ICGMR. This register is set to 21 ($15) at reset giving a default clock frequency of 6.4512MHz ± 25% (21 times 307.2kHz). The user can change this value as required. In this application, a value of 52 is used to give a nominal clock frequency of ~16MHz. The hardware employed allows modification of ICGMR so other values can be used. The only limitation caused by the use of lower frequencies is that the measurement and adjustment resolution will be poorer unless the length of the external reference pulse is correspondingly increased.

When using other values of ICGMR, whose theoretical range is 1 – 127, it should be remembered that the initial frequency can be up to 25% high so a nominal bus frequency of more than 6.4MHz is not appropriate. The +25% tolerance could take higher nominal frequencies over 8MHz and functionality would not be guaranteed. The maximum safe value before any trimming is thus 83 (4 * 6.4 / 0.3072).

After trimming the maximum safe number depends on knowledge of the tolerance of the clock. At ± 10% up to 94 (7.27MHz nominal bus frequency) can be used and at ± 1% up to 103 (7.91MHz nominal bus frequency) would be appropriate. Even after trimming, 104 should really not be used as the 8MHz

maximum bus frequency requirement could only be observed if the accuracy was known to be better than 0.16%. This is not achievable over any realistic temperature range.

Unlike in a PLL, where the comparison is with an accurate reference frequency, the ICG's reference takes the form of voltage and current sources as shown in Figure 2. A frequency to voltage conversion is made using a current reference and a capacitor and the resultant voltage is compared with voltage references. The ICG's trim byte (ICGTR) controls the size of the capacitor by switching in or out between 384 and 639 small capacitors. As the precision of these internal references is limited by the manufacturing process, improving the accuracy requires the user to supply an external timing reference to decide on the best value to put into the trim register. Once this is done, any inaccuracies due to manufacturing variations are reduced to an insignificant level compared to those that arise as a result of temperature and supply voltage variations.

The trim register is, however, volatile so it has to be loaded at each reset or power up. The inclusion of FLASH on MC68HC908EY16 and MC68HC908KX8 MCUs facilitates an appropriate place to store this information. The trim byte is centred at $80 (512 capacitors) at reset and should be loaded with the information stored in FLASH at the start of the application code. In order to avoid an inappropriate value being used in the case where the FLASH location has been erased or programmed to a default value, $80 should be retained if the information in FLASH is $00 or $FF.

The aim of the trimming operation is to configure the voltage/current/capacitor reference system so that the divided down frequency entering it is as close as possible to 307.2kHz. This may seem a somewhat obscure number but it is just a sixteenth of the common crystal frequency 4.9152MHz. The value 4.9152MHz has, itself, been chosen to facilitate standard serial baud rates so 307.2kHz is also appropriate for this purpose (9600 times 32 = 307200).

3 Trimming Calculation

The trimming adjustment involves measuring the actual bus frequency with the use of an external pulse of known length and then adjusting the trim register by the number which gives the required correction given that changing its value by 1 causes a frequency change of ~0.195%. In this case the external pulse is 1024µs between successive rising edges.

As explained above, the value chosen for the multiplier register is not critical and 52 is used. This corresponds to a nominal clock frequency of 15.9744MHz and a nominal bus frequency of 3.9936MHz. The software works for any value of multiplier as it calculates a "constant" that takes into account the multiplier as well as the length of the external pulse:

cnt1024 = (ICGMR * 307.2kHz * 1024µs) / 4 / 1000

The "/4" converts from clock to bus frequency. The ".2" ensures that floating point arithmetic is used and that there is thus no unnecessary rounding of the answer. The calculation gives the answer 4089 for an ICGMR of 52. This is just the number of 3994kHz bus cycles that would occur in the 1024µs period. The "/1000" is required to convert the units to counts from "millicounts" (kHz x µs). For effective trimming a single fixed value of ICGMR could be used. In this case cnt1024 would simply be a constant.

The adjustment to the ICGTR is made using the following equation where delta0 is the number of counts (measured by timer A channel 0) between successive rising edges of the external 1024µs signal. The initial number of small capacitors (512 with ICGTR = $80), gives rise to the resolution of 0.195% (1/512 = 0.001953).

ICGTR = ICGTR + (512 * (delta0 – cnt1024)) / cnt1024

This procedure takes the ICG to within 3 or 4 of the optimal trim value. Due to imperfections in the linearity of the ICGTR value vs. frequency plot it may not always be exact. Repeating the procedure once more results in the correct value to within ± 1. The method works equally well with devices whose initial untrimmed frequency is above or below nominal.

Table 1 shows the results obtained with two devices whose untrimmed frequencies were on either side of nominal. The frequency shown is expressed as a percentage of 3994kHz. This is displayed as well as the actual frequency on the LCD in this application as it is much easier to follow the progress of the trimming process using this number than it is using the frequency itself. The ICGTR value is shown in decimal in brackets. Any subsequent attempts to trim after those shown caused no change as the errors were already less than 0.195%.

Table 1. The Actual Triming Behavior of Two Devices

Device	out of reset	after 1st trim	after 2nd trim	after 3rd trim
1	101.8%(128)	100.7%(137)	100.3%(140)	100.1%(141)
2	88.1%(128)	100.6%(68)	100.0%(71)	

The software includes the reception of the LIN message described in AN2264 (LIN Node Temperature Display [5]). The two digit temperature is displayed on the LCD if it is correctly received but a default display of ** is shown if the LIN function "LIN_MsgStatus()" does not recognise the presence of the message. This is the case if the baud rate of the MCU is not accurate enough for the LIN protocol to function correctly. The MCU's baud rate is directly related to its clock frequency and in this application it is set up so that it will be correct (i.e. 9600 baud) with a 4MHz bus clock (9600 is achieved exactly at 3.9936MHz). Device 1 with less than a 2% error out of reset always displayed the message correctly. Device 2 displayed ** out of reset but once the first trim had been executed it too displayed the message correctly.

4 Hardware

The LIN evaluation board described in AN2343 (HC908EY16 LIN Monitor [3]) and AN2432 (LIN sample application for the MC68HC908EY16 Evaluation Board [4]) was used for this application. A 2x16 character LCD was added using the same interface as that described in AN2343. The MCU used on the LIN evaluation board is the MC68HC908EY16 [2] so this is the MCU used here but the results are equally valid for the MC68HC908KX8.

The board incorporates an MC33399 (or MC33661) LIN physical interface and thus can constitute a LIN slave node to demonstrate LIN functionality using the ICG. The full layout of the LIN evaluation board is shown in application notes AN2343 and AN2432 but the complete circuit diagram used here is shown in Figure 3. Apart from the MCU and the MC33399 (or MC33661), only an MC7805 5 volt regulator is required to implement the simple LIN node. This chip count could be further reduced by using the

MC33689 (LIN SBC) as this chip incorporates a voltage regulator. The only other components required for the ICG trimming application are the LCD module and a few resistors and decoupling capacitors (not shown).

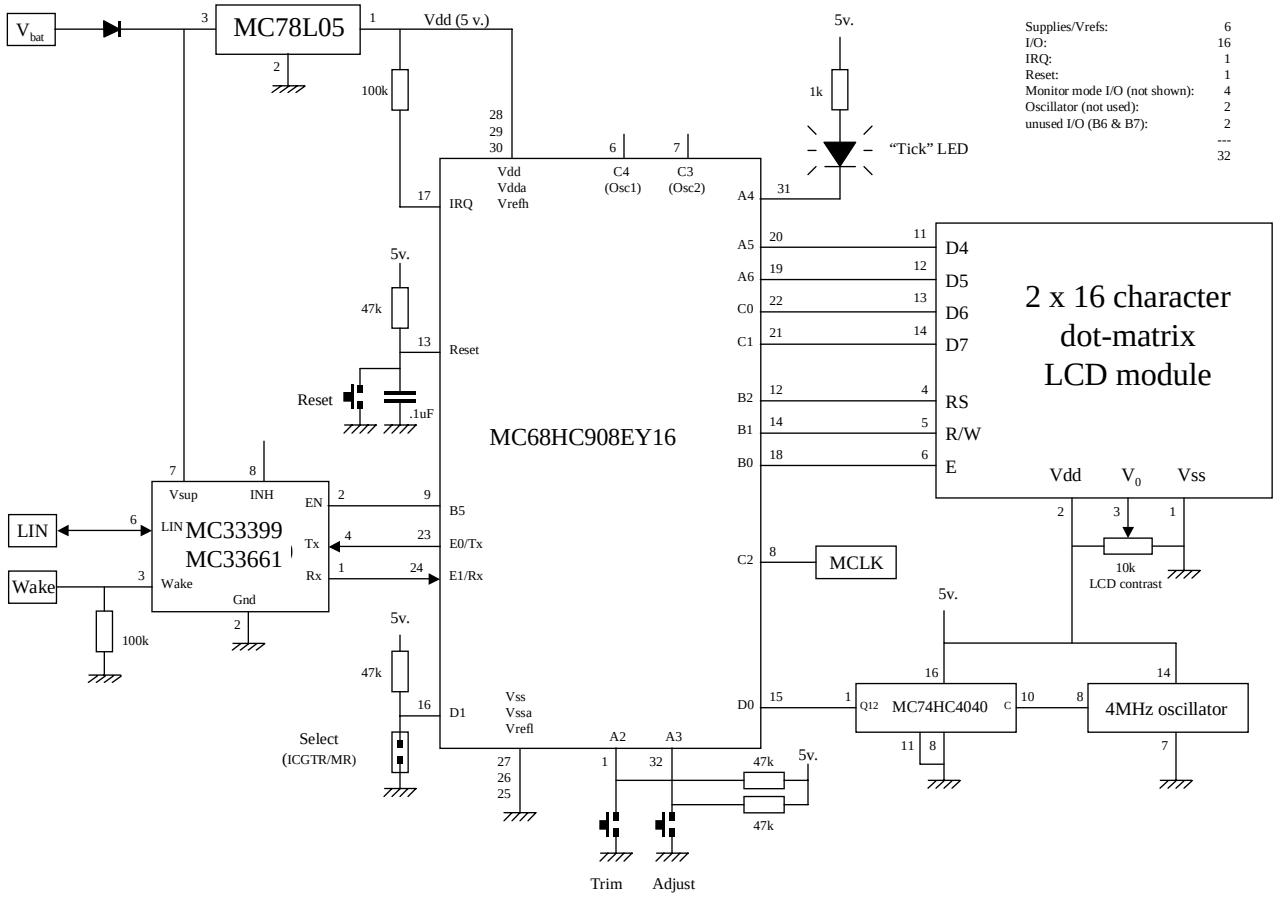

Figure 3. Circuit Diagram of ICG Trimming Application

The LIN evaluation board includes an RS232 interface and an 8MHz crystal and a socket for a 9.8304MHz oscillator module as alternative clock sources. These components and the pull-ups and pull-downs required to enter monitor mode are, for clarity, not shown in Figure 3 but they allow the board to be used to develop code without the requirement for a sophisticated development system like the Motorola/Metrowerks MMDS/MMEVS. The board also incorporates a 16-pin header to interface with P&E Cyclone or Multilink hardware (http://www.pemicro.com). This allows even easier debugging as the power and reset control required for FLASH programming can be handled by the PC running P&E WINIde or Metrowerks Codewarrior software (http://www.metrowerks.com).

This application was developed using a P&E Multilink interface which, like the Cyclone, can supply a clock to the board, so neither of the oscillator circuits nor the RS232 interface were used. While most of the software could be debugged using this set-up, it was not possible to debug the trimming process itself. This is a consequence of the monitor mode used by the Multilink interface utilising its own (external to the MCU) clock. It was thus necessary to check this aspect of the application by trial and error by removing the Multilink connection and running in user mode.

In order to supply the required calibration pulse to the MC68HC908EY16's PTD0 timer pin, some extra circuitry external to the PCB is employed. A 4MHz oscillator is used. Its output is divided by an MC74HC4040 12 stage ripple counter to give the required 976.5625kHz signal (1024µs period).

The format of the LCD displayed data is shown in Figure 1. The top line shows the current value of ICGTR in decimal and, in brackets, in hexadecimal. The "t" indicates that it is ICGTR which is being displayed as this display can be changed to ICGMR if PTD1 is held low. In this case the "t" is replaced with an "m". The actual bus frequency is also displayed on the top line. MCLK is enabled on PTC2 as an external check of this frequency.

The bottom line of the display shows the frequency 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 and the displayed percentage is the average of these values (shown with a resolution of 0.1%). Multiple readings are saved so that the jitter can also be displayed. The next figure on the bottom line is half the difference between the largest and the smallest result in the array shown to a resolution of 0.01%. This resolution is achieved at a bus frequency of 4MHz but falls off linearly as the bus frequency drops. A longer external reference pulse would be required to achieve this resolution at lower bus speeds.

Two buttons are also included as shown in Figure 3. 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 which would corrupt the deliberately modified value in ICGTR. If PTD1 is held low using the jumper shown, then 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 standard trimming operation normally required (and illustrated in the examples in Table 1), no manual adjustments of either register is necessary.

5 Digitally Controlled Oscillator

The digitally controlled oscillator (DCO) consists of a ring oscillator (see Figure 4) with a variable odd number of stages between 17 and 31. This oscillator runs at a higher frequency than necessary (in the region of 40-100MHz) so that any required frequency can be derived by an appropriate division. The oscillator is followed by a binary weighted divider controlled by the 4-bit register DDIV (DCO divide register). This introduces a divide ratio of 2DDIV so a DDIV of 1 causes a divide by 2, a 2 causes a divide by 4, a 4 causes a divide by 16 and an 8 causes a divide by 256. The maximum value of DDIV is $09 so the maximum division ratio is 512.

An 8-bit register DSTG (DCO stage register) facilitates finer adjustment. The top 3 bits of DSTG determine the frequency of the ring oscillator by controlling the number of stages included in the ring (stages = 17 + 2 x DSTG[7–5]). This effectively provides fractional division ratios smaller than the factor-of-two resolution of the DDIV divider. The bottom 5 bits are used to adjust the output frequency in

even smaller increments by hopping between two adjacent numbers of invertors in the ring oscillator for different duty cycles within an overall cycle of 32 clocks.

To do this the ring oscillator output is fed into a 5-bit counter. The value in this counter is constantly compared with the lower 5-bits of DSTG. When the counter's output is less than the value in DSTG[4–0], the comparator's output is one, otherwise, it is zero. This output is added to the upper 3-bits of DSTG to determine the number of stages in the ring oscillator. The lower 5-bits of DSTG thus determine how many cycles will occur at the lower of the two frequencies as the higher adder output selects two more inverters in the ring oscillator. The 5-bit counter dictates that the output pattern will repeat every 32 clock cycles. The resultant frequency is thus not completely steady over short periods and only gives its true overall figure when averaged over 32 cycles.

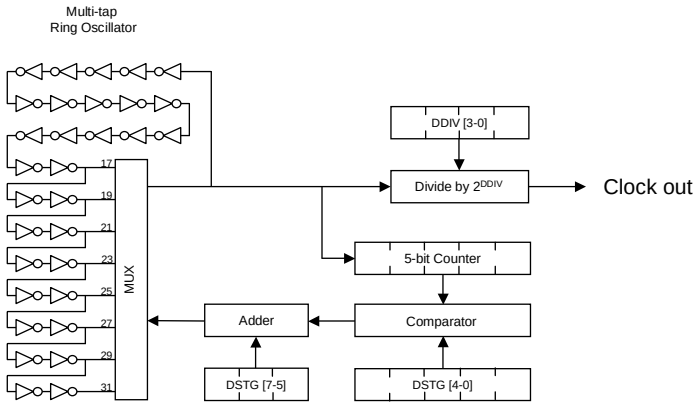

Figure 4. The Digitally Controlled Oscillator

This results in some jitter which is quantified in detail in the specification [2]. As the adjustment is always by 2 stages, the quantisation jitter varies according to how many stages are actually in use. The lowest jitter is at 29 & 31 (6.45%) and the highest at 17 & 19 (11.76%). This is, however, a cycle-to-cycle jitter and is averaged out by the divide by 32 counter which corresponds to bits 4-0 of DSTG. The result is that the actual jitter is only 1/32 of these figures (0.368% maximum) when measured over a period of 32 or more clock cycles. This is 32 ring oscillator clock cycles (not MCU clock cycles) so the averaging happens over a period of less than 1µs regardless of the actual MCU clock frequency.

An indication of jitter is displayed in this application as the min-max percentage display. The jitter averages out even more over longer periods and the low measured jitter of about 0.04% is the result of it being measured over the relatively long period of ~1ms. The time periods relevant to serial protocols like LIN are also sufficiently long that any jitter is averaged out to less than ± 0.1% and is thus not a significant issue.

The tables below show the actual percentage difference between all pairs of ring oscillator possibilities for both increasing and decreasing frequency. As described above, the practical quantisation jitter achieved in each case is 1/32 of these figures once the clock emerges from the 5-bit fine adjustment counter. This results in a maximum cycle by cycle jitter of 0.368% (11.76%/32). The last ratio in each table shows the loopback to the other extreme number of stages in the ring. When this happens there is a carry or borrow from DSTG to DDIV thus adding or removing a divide by 2.

Table 2. Decreasing Frequency

Tabs	17	19	21	23	25	27	29	31	17

Ratio	17/19	19/21	21/23	23/25	25/27	27/29	29/31	½ x 31/17
- %	10.53	9.52	8.69	8.00	7.41	6.90	6.45	8.82

Table 3. Increasing Frequency

Tabs	31	29	27	25	23	21	19	17	31

Ratio	31/29	29/27	27/25	25/23	23/21	21/19	19/17	2 x 17/31
+ %	6.90	7.41	8.00	8.69	9.52	10.53	11.76	9.68

Table 4 shows the overall functioning of the DCO. The actual range of division ratios achieved by each bit is shown in the "Ratio" row. As the division ratios get close to 1, the deviation from 1 is, for clarity, shown as a percentage. The shaded area of the table shows the points within DSTG where ICG corrections adjust the register. This is also shown in Figure 2. There are two pairs of comparators that are used to make large frequency changes (e.g. during power-up or a deliberate frequency change made by the application code). According to whether or not the error is over 15%, large adjustments are made at bit 5 or bit 3. If, however, the error is less than 5%, a fifth comparator is used to make adjustments with the finest possible step size by utilising bit 0.

Table 4. Overall Operation of the ICG

	DDIV				DSTG								
	3	2	1	0	7	6	5	4	3	2	1	0
Ratio	256	16	4	2	8/31 – 8/17	4/31 – 4/17	2/31 – 2/17					
%					26% – 47%	13% – 24%	6.45% – 11.76%	3.23% – 5.88%	1.61% – 2.94%	0.81% – 1.47%	0.40% – 0.74%	0.202% – 0.368%
	Binary weighted divider				Ring oscillator taps			Fine adjustment (over 32 cycles)				
Error					⇑ > 15%			⇑ 5-15%				⇑ < 5%

Table 5 gives some insight into the maths of the ICG. The integer binary weighted divider DDIV can be considered to be continued into fractional divide ratios using DSTG. For this purpose the two registers can be regarded as a single 12 bit register with the bit numbers shown. The 2x row shows the ratios that result from this extrapolation. Again, ratios close to unity are shown as percentages. As can be seen, the numbers derived by continuing the binary weighted maths into the fractional domain give an accurate indication of what actually happens. The theoretical numbers are within the range of ratios actually achieved by each bit (Table 4) and very close to the geometric mean of the maximum and minimum values.

Table 5. Mathematical Behaviour of the ICG

	DDIV				DSTG							
	3	2	1	0	7	6	5	4	3	2	1	0
Bit	11	10	9	8	7	6	5	4	3	2	1	0
X	8	4	2	1	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256
2^x	256	16	4	2	1.41	1.19	1.09					
%					41%	19%	9.0%	4.4%	2.2%	1.1%	0.54%	0.27%

6 Software

The software was written and debugged using the LIN demo PCB and the Metrowerks' Codewarrior development environment with a P&E Multilink interface. The application uses the Motorola/Metrowerks LIN drivers to handle the LIN protocol outwith the application code. It thus only employs "LIN_MsgStatus()" and "LIN_GetMsg()" calls (after an initial "LIN_Init()") to demonstrate the functionality of the LIN bus.

The main software flow chart is shown in Figure 5. After the CONFIG and Port registers have been initialised, ICGMR is given the value 52. This gives the nominal bus speed (3.994MHz) that is closest to 4MHz. The LIN drivers are configured (in file slave.cfg) for 9600 baud at 4MHz. The time base module is programmed with its maximum divide ratio of 4,194,304 to give a slow (4Hz) repetition rate through the loop. This was done as the LCD is written to at this rate and there is no point updating it so fast that any changing digit cannot 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 initialised and interrupts enabled.

The main loop is timed by polling the time base module's interrupt/overflow flag and toggles the LED on bit 4 of port A to indicate that the code is running correctly. The buttons on bits 2 and 3 of port A are then read to decide if 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 and 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[5] and the format of the message is described in detail there. In this application the message is only being used to indicate whether or not 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 so 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 but is calculated each time round the loop in case that value 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 have to be displayed into the ASCII format required for the LCD module. Adding 0x30 is all that is required for the decimal numbers but the array "ASCIIconv[16]" is used to convert hexadecimal digits. The averaging and spread calculations for the bottom line display are performed within "Format_line2()".

The last function performed with 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 this way so that all the required pins were all on the same PCB header.

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 one is calculated and that delta put into the array of 16 values that are used for the averaged frequency and jitter displays.

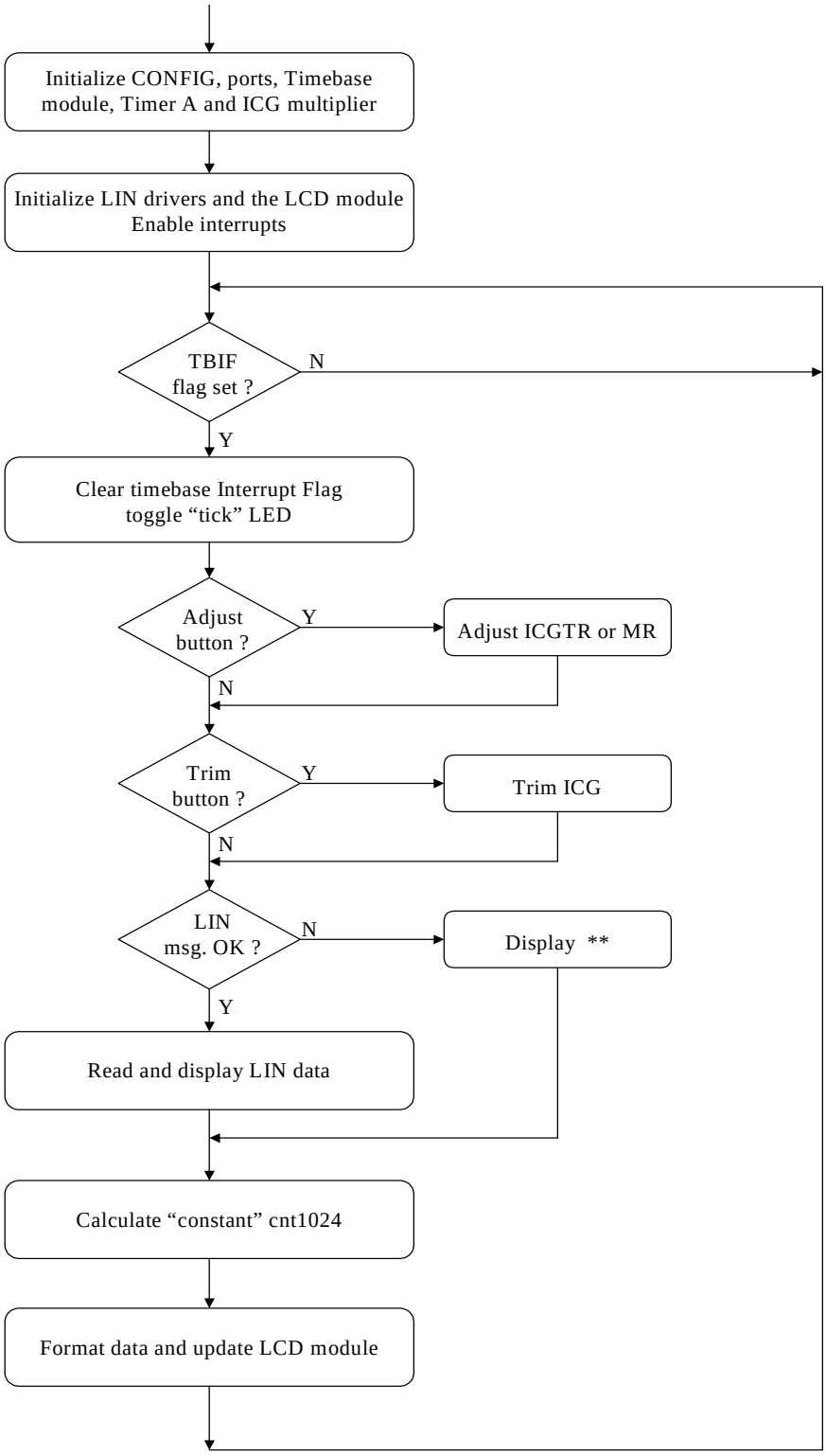

Figure 5. Flow Chart of Main Software Loop.

7 References

1. LIN Protocol Specification, Version 1.3, 12 December 2002.
2. MC68HC908EY16 Technical Data.
3. AN2343, HC908EY16 LIN Monitor.
4. AN2432, LIN sample application for the MC68HC908EY16 Evaluation Board.
5. AN2264, LIN Node Temperature Display

8 Software Listing

```
/*****************************************************************************
*               (c) MOTOROLA Inc. 2003  all rights reserved.                 *
*                                                                            *
*                                                                            *
*                    MC68HC908EY16 initial ICG trimming.                     *
*                    ===================================                     *
*                                                                            *
*    Originator:    P. Topping                                               *
*    Date:          22nd March 2003                                          *
*    Revision:      1.0                                                      *
*    Function:      Initial ICG trimming using the "LIN demo." PCB.          *
*                                                                            *
*****************************************************************************/


/*****************************************************************************
*
*    Header file includes, function prototypes and global variables
*
*****************************************************************************/

#pragma DATA_SEG SHORT _DATA_ZEROPAGE

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

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);

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[] = "---(---) ----kHz";
unsigned char Line2[] = "---.-% -.--%  --";
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;
```

```
/*****************************************************************************
*
*    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 = 0x05;                             /* slow clock for TBM      */
    ICGMR = 52;                                 /* 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 */

    remain = (delta0*thousand)/1024;            /* calc. freq. from delta  */
    for (count = 12; count >= 9; count--)
    {
        Line1[count] = (remain%10) + 0x30;      /* display frequency       */
        remain = remain/10;
    }
    if (Line1[9] == 0x30) { Line1[9] = 0x20; }  /* leading 0 becomes space */
}

/*****************************************************************************
*
*    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 jitter (0.01%)
*                   (the LIN temperature display is added by Read_LINtemp())
*
*****************************************************************************/

void Format_line2 (void)
{
int remain;
int min;
int max;
long total;

    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[4] = (remain%10) + 0x30;              /* display tenths          */
    for (count = 3; count != 0; count--)
    {
        remain = remain/10;
        Line2[count-1] = (remain%10) + 0x30;    /* and percentage          */
    }
    if (Line2[0] == 0x30) { Line2[0] = 0x20; }  /* leading 0 becomes space */

    min = delta_buffer[0];
    max = delta_buffer[0];
    for (count = 1; count < 16; count++)
    {
        if (delta_buffer[count] < min)
        {
            min = delta_buffer[count];          /* find smallest delta     */
        }
        if (delta_buffer[count] > max)
        {
            max = delta_buffer[count];          /* and largest delta       */
        }
    }
    remain = (5*thousand*(max-min))/cnt1024;    /* calc (max difference)/2 */
    Line2[10] = (remain%10) + 0x30;             /* as a % of nominal freq. */
    remain = remain/10;
    Line2[9] = (remain%10) + 0x30;
    remain = remain/10;
    Line2[7] = (remain%10) + 0x30;
}
```

```
/*****************************************************************************
*
*    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[13] = 0x20;                                /* no, display  **         */
        Line2[14] = 0xA5;                                /* to signify no valid     */
        Line2[15] = 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[15] = units + 0x30;                        /* units digit             */
        Line2[14] = 0x20;                                /* clear tens digit        */
        Line2[13] = 0x20;                                /* and hundreds digit      */

        if (tens != 0)                                   /* tens digit zero ?       */
        {
            Line2[14] = tens + 0x30;                     /* no, display it          */
            if (negative)                                /* negative ?              */
            {
                Line2[13] = 0xB0;                        /* yes put "-" in hundreds */
            }
        }
        else if (negative)                               /* tens zero, negative ?   */
        {
            Line2[14] = 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)
    {
    }
}
```

Appendix A HC08EY16.h (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 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 VECTF (void(*const)()) /* Vector table function specifier */
```

Appendix B 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 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 */
    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) */
```

Appendix C Slave.cfg (LIN configuration file)

```
#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 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 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 a 8MHz crystal */
//#define LIN_BAUDRATE            0x30u

/* Selects 9600 baud rate if using a 16MHz crystal */
#define LIN_BAUDRATE            0x31u

/*
    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 D Slave.id (LIN message ID file)

```
#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:

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

Document Number: AN2498
Rev. 1
01/2007

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.

freescale™
semiconductor