

*Application Note*

*AN2343/D  
Rev. 0, 10/2002*

*HC908EY16 LIN Monitor*



by **Carl Culshaw**  
**Applications Engineering**  
**Motorola, East Kilbride**

---

## **Introduction**

With the continued increase in communications within the Automotive environment, car manufacturers are resorting more and more to serial multiplex bus systems. Controller Area Networks (CAN) continue to dominate, but many applications are now being designed around the Local Interconnect Network (LIN, reference 1).

To support this requirement, Motorola have designed a range of devices, development boards and software applications to facilitate the migration to LIN.

This application note will focus on the following:

1. Hardware: HC908EY16 Sample Evaluation board
  - A pcb designed to ease new application development when using the HC908EY16.
  - LIN Physical Interface
  - On board Programming
2. Software: Demonstration of LIN functionality
  - Use of the Metrowerks LIN Software Drivers
  - Monitoring LIN activity
  - Interfacing a LCD display with the Sample Evaluation Board

---

## Hardware: HC908EY16 Sample Evaluation board

### General Overview

The HC908EY16 Sample board has been designed to allow the user an easy development path with the EY family of parts.

Central to the board is the 44 pin HC908EY16. Each of the user pins (Ports A through E) is brought out to Single In Line Headers (P3 & P5). This is to allow the user to easily interface with much of the functionality of the EY16.

On board LEDs (on Ports A & D) are provided for general application use.

### Clock, Reset & IRQ Options

The EY16 is capable of running from a variety of clock sources: internally from the Internal Clock Generator (ICG), externally from a crystal source (from 32KHz up to 8MHz), or from an external oscillator module. This board allows the user to develop applications using any of the three options.

When using the on board crystal, it is necessary to make the link from pin 11 to pin 13 on P4 and also from J2 pin 1 to pin 2. If using a clock source such as the on board 9.8304MHz canned oscillator, these links should be removed. This allows easy clock selection between the different sources.

Power on reset is provided on the board, as well as a reset switch (S2).

The IRQ pin is used on all Motorola HC08s to select between Monitor mode and User mode. Link J1 allows the user to select the required mode. Its default position is from pin 1 to 2. If using the Cyclone (see later text), this link should be removed.

### Power and LIN functionality

Power can be provided in two ways: either 12V through the Battery Input Connector (B1) or 12V through the LIN Interface Connector (P2).

This LIN Connector is a three pin header, which is the requirement for connection with a LIN Communication network (two power lines & single LIN communication wire).

Whilst the HC908EY16 is designed as a general multi-purpose MCU, it does have several features to simplify LIN interaction. These include an Enhanced SCI module (with a special timer functionality for break detection & generation) and the ICG (no external crystal required).

With these LIN features in mind, a LIN Physical Interface (MC33399) has been included on the pcb. It is enabled by a port pin (PTB5 high) and can be configured as either a Master or Slave node. If D1 & R4 are fitted, then the node

will act as a LIN master; if D1/R4 are omitted, the node defaults to slave status. This behaviour is more fully described in the LIN specification (reference 1).

**NOTE:** *This option configures the hardware as a Master or Slave. For complete master or Slave status it is still necessary to configure the software appropriately.*

## On board programming

The HC908EY16 Flash device can be quickly programmed on this Sample board from two sources: the RS232 connector & the Cyclone Header.

If using the RS232 connector (P1), the following steps are recommended:

1. Power down the board
2. Ensure that the Cyclone is disconnected
3. Remove the P4 pin 11 – P4 pin 13 connection
4. The canned oscillator should be fitted
5. Connect a serial cable from P1 to the serial port on the PC
6. Connect J1 pin 2 to pin 3 (connects high voltage to IRQ)
7. Power the board (via switch S1)
8. Launch CodeWarrior and follow the on screen guide (reference 2)

Once the Flash is programmed, code can be executed and debugged using the CodeWarrior environment or the user may choose to disconnect the debugger and run from Flash. Please be sure to select the correct crystal option and IRQ configuration (typically J1 pin 1 – pin 2 connected) in either case.

If programming using the Cyclone (reference 3):

1. Power down the board
2. Remove the P4 pin 11 – P4 pin 13 connection
3. The canned oscillator should be removed
4. Connect J1 pin 1 to pin 2 (connects Vdd to IRQ – Cyclone will override as appropriate)
5. Connect the Cyclone to the 16 pin header (P4)
6. Connect the Cyclone to the PC as per the User guide (reference 3)
7. Power the board (via switch S1)
8. Launch CodeWarrior and follow the on screen guide (reference 2)

Again, code can be executed and debugged using the CodeWarrior environment or the user may choose to disconnect the debugger and run from Flash. If the Cyclone is used, it should be configured to provide the on board clock.

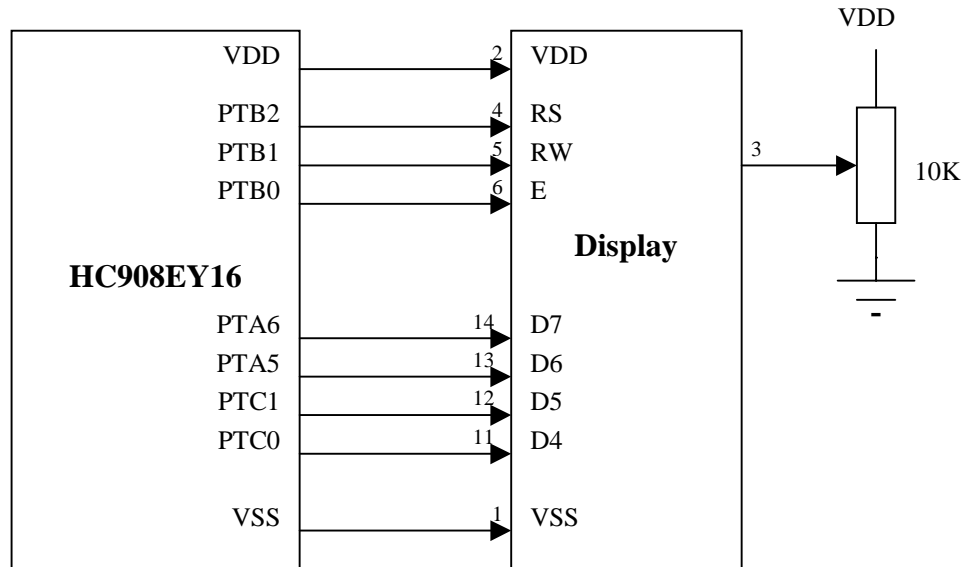


Once the id has been determined, LIN\_GetMsg is used to retrieve the last message data received on the bus.

This data is written onto the display, along with the appropriate message id.

A simplified block diagram of the MCU interfacing with the display would be:

**Interfacing a LCD display with the Sample Evaluation Board**



As can be seen from the diagram, only four lines are being used as the data bus between the EY16 & the display. This complicates the software slightly, since any data writes / reads take place over two cycles rather than one. However, this makes the hardware interface much more straightforward, minimising the lines required to drive the display.

During initial power on, the display defaults to 8 bit mode. Hence one of the essential events in the display initialisation routine changes this default into 4 bit mode.

All display actions should be preceded by examining the busy flag, which is the MSB of a two byte register read.

The majority of the software routines are very simple and self-explanatory: please see the attached software listings with comments.

---

## References

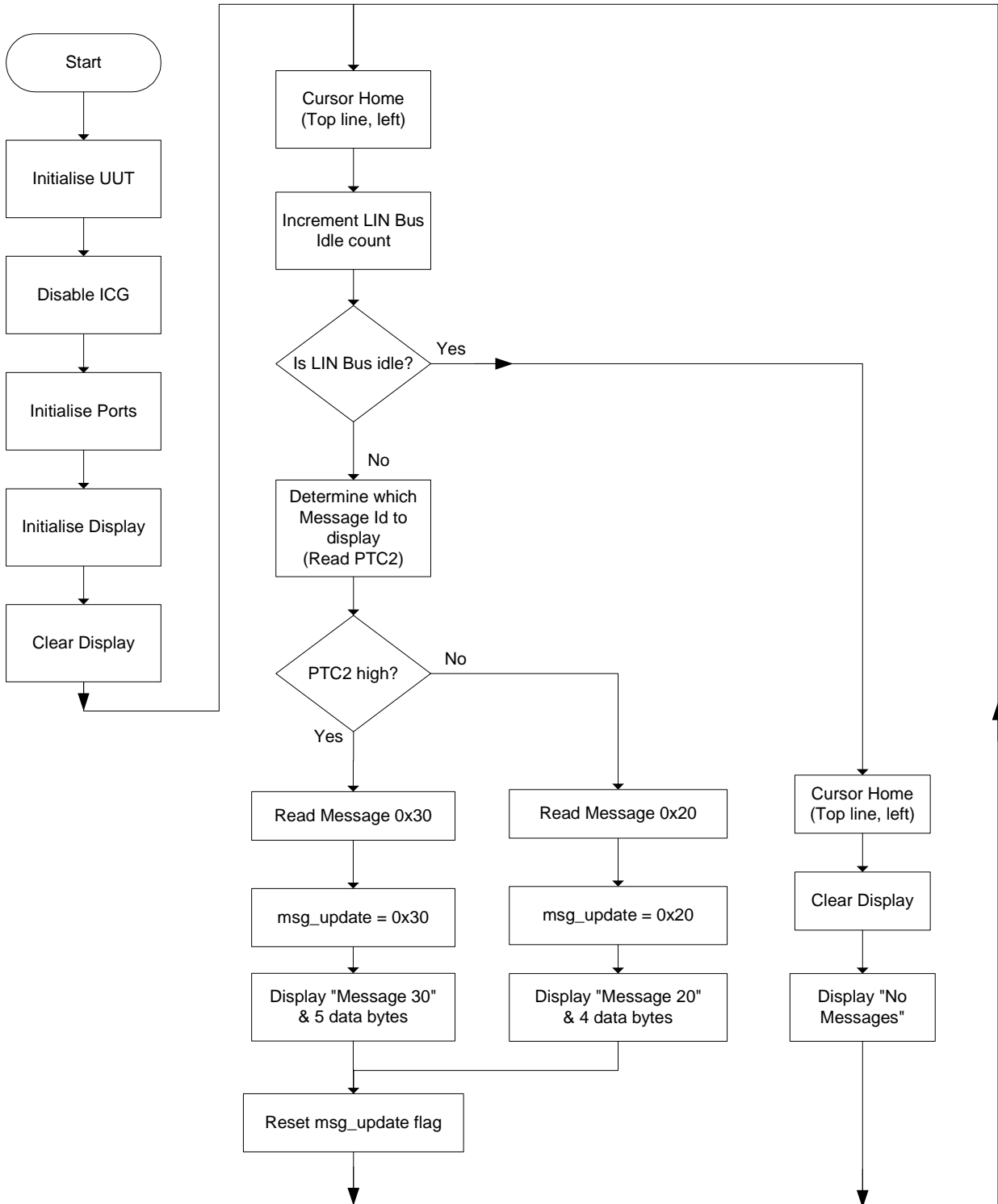
1. LIN Protocol Specification, Version 1.2, 17<sup>th</sup> November 2000
2. Metrowerks CodeWarrior, [www.metrowerks.com](http://www.metrowerks.com)
3. MON08 Cyclone, [www.pemicro.com](http://www.pemicro.com)
4. Metrowerks LIN Drivers, [www.metrowerks.com](http://www.metrowerks.com)

---

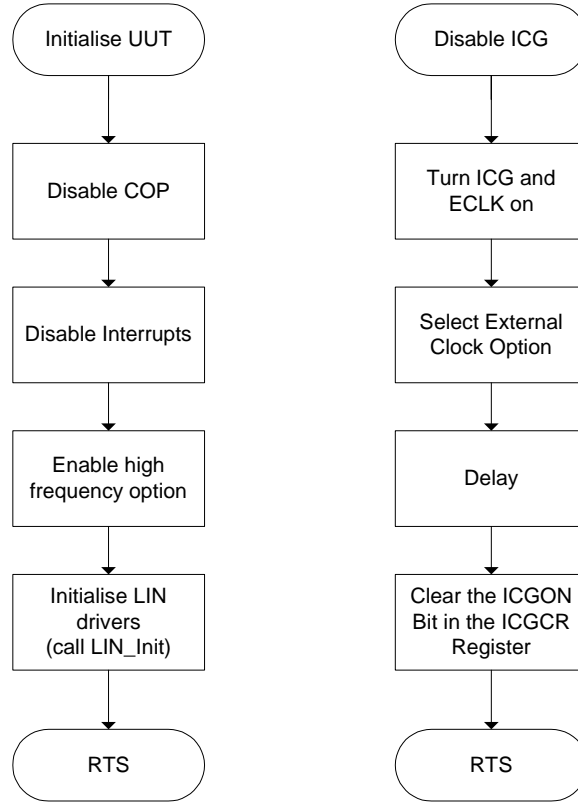
## Acronyms

UUT	Unit Under Test
LIN	Local Interconnect Network
CAN	Controller Area Network

Appendix A: Software Flowchart – Main Control loop

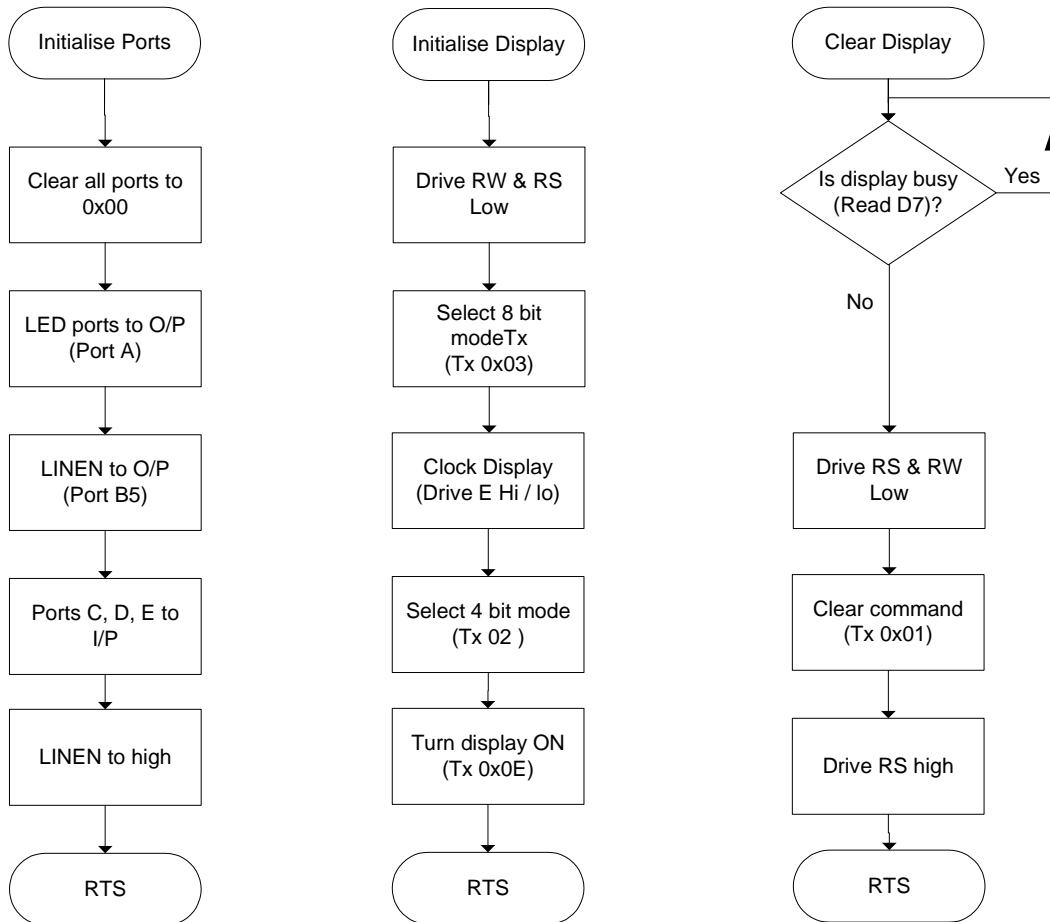


Appendix A: Software Flowchart – Initialisation Functions (1)





Appendix A: Software Flowchart – Initialisation Functions (2)



Freescale Semiconductor, Inc.

---

## Appendix B – Source Files

### ey16 giveaway.c

---

```

/*****
                                     Copyright (c) Motorola 2002
File Name       :       ey16 giveaway.c
Originator      :       C. Culshaw
Location        :       EKB
Date Created    :       8/3/02
Current Revision :       1.0
Function        :       Demonstrate LIN functionality using the EY16 Sample board
Notes           :       Utilises the HC908EY16 as a LIN monitor.
                   Software requires companion hardware,
                   'EY16 SAMPLE BOARD, BEAPP038' & a hitachi LCD display

*****

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

#pragma DATA_SEG SHORT _DATA_ZEROPAGE

// Prototypes for functions
#include "ey16port.h"
#include "ey16config.h"
#include "ey16icg.h"
#include "linapi.h"          // Include the Motorola EY16 LIN drivers as part of the project

// Global register definition

#define EY16PORT (* ((tEY16PORT *)0x00))          //Register block definition
#define EY16SI (* ((tEY16SI *)0x0D))             //Serial ports
#define EY16CONFIG (* ((tEY16CONFIG *)0x1E))     //Configuration registers
#define EY16TIMER (* ((tEY16TIMER *)0x20))      //ECT registers
#define EY16ICG (* ((tEY16ICG *)0x36))         //ICG registers
#define EY16ATD (* ((tEY16ATD *)0x3C))         //ATD registers

```

```

void Initialise_UUT(void);
void Disable_ICG(void);
void Initialise_Ports(void);
void Initialise_Display(void);

void Delay(unsigned long);
void Clock_Display(void);
void Display_Port(unsigned char);
void Display_Data(unsigned char, unsigned char);
void Clear_Display(void);
unsigned char Busy_Status(void);
void Cursor_Home(void);
void Write_Line2(unsigned char);
void DisplayChar(char, unsigned char, unsigned char);

// Global variable definition
unsigned char No_messages[] = "No messages";
unsigned char Message[] = "Message ";

unsigned char Message_20[4]; // Used to store LIN received message data
unsigned char Message_30[8]; //

/*****
* Function      :      LIN_Command
*
* Description   :      User call-back.
*                    Called by the driver after successful transmission or receiving
*                    of the Master Request Command Frame (ID Field value '0x3C').
*
* Returns      :      never return
*
* Notes:
*****/
void LIN_Command()
{
    while(1)
    {
    }
}

/*****
Function Name   :      Initialise_UUT
Engineer       :      C. Culshaw
Date           :      06/09/02
Parameters     :      None
Returns        :      None
Notes          :      Initialise micro to default conditions
*****/
void Initialise_UUT(void)
{
    EY16CONFIG.ey16config1.byte = 0x01; // Disable COP watchdog
    asm sei; // Disable interrupts
    EY16CONFIG.ey16config2.byte = 0x29; // Configure part for high frequency
                                           // external crystal, ssb pull up resistor disabled
    LIN_Init(); // Initialise SCI / LIN registers
}

/*****

```

```

Function Name      :      Disable_ICG
Engineer          :      C. Culshaw
Date              :      06/09/02
Parameters        :      None
Returns           :      None
Notes             :      The EY16 defaults to its internal clock generator (ICG).
                   :      For this application, disable the ICG & select the
                   :      external crystal
*****/
void Disable_ICG(void)
{
    unsigned short delay1 = 0;
    EY16ICG.icgcr.byte = 0x0A;                // ICG and ECLK ON
    EY16ICG.icgcr.byte = 0x1A;                // set CS
    while (delay1++ != 100);
    EY16ICG.icgcr.byte = 0x12;                // now put ICG OFF

    while (!(EY16ICG.icgcr.byte & 0x13))
    {
        EY16ICG.icgcr.byte = 0x12;           // CS = ECLK, Ext clk ON and ICG off
    }
}

```

```

/*****
Function Name      :      Initialise_Ports
Engineer          :      C. Culshaw
Date              :      06/09/02
Parameters        :      None
Returns           :      None
Notes             :      For this LIN application, Port B bit 5 is used to enable
                   :      the LIN Physical Interface
                   :      General purpose LEDs are connected to port A bits 4,5,6
                   :      & port D bits 0 & 1
*****/
void Initialise_Ports(void)
{
    EY16PORT.pta.byte = 0x00;                // Ensure all port registers
    EY16PORT.ptb.byte = 0x00;                // 'cleared' to default
    EY16PORT.ptc.byte = 0x00;
    EY16PORT.ptd.byte = 0x00;
    EY16PORT.pte.byte = 0x00;

    EY16PORT.ddra.byte = 0x70;                // Set port A bits 4,5,6 (LEDs) to O/P, rest input
    EY16PORT.ddrb.byte = 0x20;                // Set port B bit 5 (LINEN) to O/P, rest input
    EY16PORT.ddrc.byte = 0x00;                // Set port C to input & do not output MCLK
    EY16PORT.ddrd.byte = 0x00;                // Set port D to input
    EY16PORT.ddre.byte = 0x00;                // Set port E to input

    EY16PORT.ptb.byte = 0x20;                // LIN enabled
}

```

/\*\*\*\*\*
 Following display routines are to be used in conjunction with a Hitachi LCD display.
 This application only uses a 7 line interface, connected as follows:
 Display D7 D6 D5 D4 RS RW E
 EY16 Port C1 C0 A6 A5 B2 B1 B0
 \*\*\*\*\*/

The interface will be initialised into four bit mode, where each byte transfer is completed

```

using a two nibble write
*****/

/*****
Function Name      :      Clock_Display
Engineer          :      C. Culshaw
Date              :      06/09/02
Parameters        :      None
Returns           :      None
Notes             :      Drive the Display E line high / low to clock data
*****/
void Clock_Display(void)
{
    EY16PORT.ddrb.byte |= 0x01;          // Set port B bit 0 (Display E) to O/P
    EY16PORT.ptb.byte |= 0x01;          // Display E high
    EY16PORT.ptb.byte &= ~(0x01);       // Display E low
}

/*****
Function Name      :      Busy_Status
Engineer          :      C. Culshaw
Date              :      06/09/02
Parameters        :      None
Returns           :      busy_state - if set indicates display busy
Notes             :      Read the busy state of the display and return status to
                        :      calling routine.
                        :      Bits manipulated one at a time, otherwise display
                        :      timing requirements will be violated
*****/
unsigned char Busy_Status(void)
{
    unsigned char busy_state;
    EY16PORT.ddra.byte &= ~(0x60);       // Set port A bits 5,6 (Display D4 & D5) to I/P
    EY16PORT.ddrc.byte &= ~(0x03);       // Set port C bits 0,1 (Display D6 & D7) to I/P

    EY16PORT.ptb.byte &= ~(0x04);       // Display RS low
    EY16PORT.ptb.byte |= 0x02;          // Display RW high
    EY16PORT.ptb.byte |= 0x01;          // Display E high
    busy_state = (EY16PORT.ptc.bit.ptc1 & 0x01); // Check display status
    EY16PORT.ptb.byte &= ~(0x01);       // Display E low
    EY16PORT.ptb.byte &= ~(0x02);       // Display RW low
    Delay(1);
    return(busy_state);
}

/*****
Function Name      :      Display_Data
Engineer          :      C. Culshaw
Date              :      06/09/02
Parameters        :      data - byte to be written to display
                        :      regsel - used to drive Register Select Line
                        :      (0 = Command mode, 1 = data mode)
Returns           :      None
Notes             :      'data' will be written as two nibbles,
*****/
void Display_Data(unsigned char data, unsigned char regsel)
{
    EY16PORT.ddrb.byte |= 0x07;          // Set port B bits 0,1,2 (Display RS, R/W, E) to O/P

```

```

while (Busy_Status() == 0x01);           // Read busy status (Display D7)

if (regsel == 1)
    {
        EY16PORT.ptb.byte |= 0x04;      // Display RS high
    }
else
    {
        EY16PORT.ptb.byte &= ~(0x04);   // Display RS low
    }

// shift data to allow high nibble to be written to display
Display_Port(data/16);
Clock_Display();                        // Display E Hi / Lo

// mask & transmit low nibble
Display_Port(data & 0x0F);
Clock_Display();                        // Display E Hi / Lo
asm NOP;                                // Convenient Break point setting!
}

/*****
Function Name      :      Write_Line2
Engineer          :      C. Culshaw
Date              :      06/09/02
Parameters        :      msg_id - id of message to be displayed
Returns           :      None
Notes             :      Update second line of display with the received message
                   :      data bytes
                   :      Currently set up to act on ids 0x20 & 0x30 only
*****/
void Write_Line2(unsigned char msg_id)
{
    char nibble_pos = 0x00, byte;
    unsigned char hi_nibble, lo_nibble;

    switch (msg_id)
    {
        case 0x20:
            for(byte = 0; byte < 4; byte++)
            {
                // Extract high nibble, then add ascii offset
                hi_nibble = (Message_20[byte]/16) + 0x30;
                // Extract low nibble, then add ascii offset
                lo_nibble = (Message_20[byte]&0x0F) + 0x30;
                // if either nibble between A & F, additional ascii offset required
                if (hi_nibble > 0x39)
                    hi_nibble+=0x07;
                if (lo_nibble >= 0x39)
                    lo_nibble+=0x07;
                DisplayChar(2, nibble_pos++, hi_nibble);
                DisplayChar(2, nibble_pos++, lo_nibble);
                nibble_pos++;
            }
            break;
            // Message 0x30 is a 8 byte message, but because of display size
            // & since only the first 5 bytes are being used in this
            // application, then only display these 5 bytes.
        case 0x30:
            for(byte = 0; byte < 5; byte++)

```

```

        {
            hi_nibble = (Message_30[byte]/16) + 0x30;
            lo_nibble = (Message_30[byte]&0x0F) + 0x30;
            if (hi_nibble > 0x39)
                hi_nibble+=0x07;
            if (lo_nibble >= 0x39)
                lo_nibble+=0x07;
            DisplayChar(2, nibble_pos++, hi_nibble);
            DisplayChar(2, nibble_pos++, lo_nibble);
            nibble_pos++;
        }
        break;
    default:
        break;
}

// Clear last two bytes of display
DisplayChar(2, nibble_pos++, ' ');
DisplayChar(2, nibble_pos, ' ');
}

/*****
Function Name      :      DisplayChar
Engineer          :      S. McAslan
Date              :      16/08/01

Parameters        :      line - line of display; pos - position on line;
                        character - char to display
Returns           :      None
Notes             :      Display character at pos on line.
*****/
void DisplayChar(char line, unsigned char pos, unsigned char character)
{
    unsigned char value;

    while (Busy_Status() == 0x01); // Read busy status (Display D7)

    value = pos & 0x1F;
    if (line == 1)
    {
        value += 0x80;
    }
    else
    {
        value += 0xC0;
    }

    EY16PORT.ddrb.byte |= 0x06; // Set port B bits 1&2 (Display RS & R/W) to O/P
    EY16PORT.ptb.byte &= ~(0x04); // Display RS low
    EY16PORT.ptb.byte &= ~(0x02); // Display RW low
    Display_Data(value, 0x00); // Select DD RAM / cursor location

    while (Busy_Status() == 0x01); // Read busy status (Display D7)

    EY16PORT.ptb.byte |= 0x04; // Display RS high

    Display_Data(character, 0x01);
}

```

```

/*****
Function Name      :      Initialise_Display
Engineer          :      C. Culshaw
Date              :      06/09/02
Parameters        :      None
Returns           :      None
Notes             :      Display defaults to 8 bit data bus mode
                   :      First part of routine will change display to be 4 bit
                   :      Once 4 bit is selected, data is transmitted to the
                   :      display in two 4 bit nibbles
*****/
void Initialise_Display(void)
{
    EY16PORT.ddrb.byte |= 0x06;           // Set port B bits 1&2 (Display RS & R/W) to O/P
    EY16PORT.ptb.byte &= ~(0x06);        // Display RW & RS low

    Display_Port(0x03);                  // Function set (initially DL =1 = 8 bit mode)
    Clock_Display();                     // Display E Hi / Lo
    Delay(100);

    Clock_Display();                     // Display E Hi / Lo
    Delay(100);

    Clock_Display();                     // Display E Hi / Lo
    Delay(100);

    Display_Port(0x02);                  // Function set, with DL = 0 to 4 bit mode
    Clock_Display();                     // Display E Hi / Lo
    Delay(100);

    Display_Data(0x28, 0x00);            // 4 bit mode, N=1 (2 line display)
    Display_Data(0x08, 0x00);            // Display OFF
    Display_Data(0x0E, 0x00);            // Display ON

    Display_Data(0x80, 0x00);            // DD RAM selected

    EY16PORT.ptb.byte |= 0x04;           // Display RS high
}

/*****
Function Name      :      Clear_Display
Engineer          :      C. Culshaw
Date              :      06/09/02
Parameters        :      None
Returns           :      None
Notes             :      Clear display contents
*****/
void Clear_Display(void)
{
    while (Busy_Status() == 0x01);      // Read busy status (Display D7)
    EY16PORT.ddrb.byte |= 0x06;         // Set port B bits 1&2 (Display RS & R/W) to O/P
    EY16PORT.ptb.byte &= ~(0x04);       // Display RS low
    EY16PORT.ptb.byte &= ~(0x02);       // Display RW low
    Display_Data(0x01, 0x00);           // Clear display command
    EY16PORT.ptb.byte |= 0x04;         // Display RS high
}

/*****
Function Name      :      Cursor_Home
Engineer          :      C. Culshaw

```



```
Date          :          06/09/02
Parameters    :          None
Returns       :          None
Notes        :          Return cursor to home position (top line, left)
*****
void Cursor_Home(void)
{
    while (Busy_Status() == 0x01);           // Read busy status (Display D7)
    EY16PORT.ddrb.byte |= 0x06;             // Set port B bits 1&2 (Display RS & R/W) to O/P
    EY16PORT.ptb.byte &= ~(0x04);          // Display RS low
    EY16PORT.ptb.byte &= ~(0x02);          // Display RW low
    Display_Data(0x02, 0x00);               // Cursor home command
    EY16PORT.ptb.byte |= 0x04;             // Display RS high
}

```

```
*****
Function Name :          Display_Port
Engineer     :          C. Culshaw
Date        :          06/09/02
Parameters   :          data - output onto display lines
Returns      :          None
Notes       :          Place data onto the four display lines, D4,5,6,7
*****
void Display_Port(unsigned char data)
{
    EY16PORT.ddra.byte |= 0x60;           // Set port A bits 5,6 (Display D4 & D5) to O/P
    EY16PORT.ddrc.byte |= 0x03;           // Set port C bits 0,1 (Display D6 & D7) to O/P

    switch (data)
    {
        case 0x00:
            EY16PORT.ptc.bit.ptc1 = 0x00; // Display D7 low
            EY16PORT.ptc.bit.ptc0 = 0x00; // Display D6 low
            EY16PORT.pta.bit.pta6 = 0x00; // Display D5 low
            EY16PORT.pta.bit.pta5 = 0x00; // Display D4 low
            break;

        case 0x01:
            EY16PORT.ptc.bit.ptc1 = 0x00; // Display D7
            EY16PORT.ptc.bit.ptc0 = 0x00; // Display D6
            EY16PORT.pta.bit.pta6 = 0x00; // Display D5
            EY16PORT.pta.bit.pta5 = 0x01; // Display D4
            break;

        case 0x02:
            EY16PORT.ptc.bit.ptc1 = 0x00; // Display D7
            EY16PORT.ptc.bit.ptc0 = 0x00; // Display D6
            EY16PORT.pta.bit.pta6 = 0x01; // Display D5
            EY16PORT.pta.bit.pta5 = 0x00; // Display D4
            break;

        case 0x03:
            EY16PORT.ptc.bit.ptc1 = 0x00; // Display D7
            EY16PORT.ptc.bit.ptc0 = 0x00; // Display D6
            EY16PORT.pta.bit.pta6 = 0x01; // Display D5
            EY16PORT.pta.bit.pta5 = 0x01; // Display D4
            break;

        case 0x04:
            EY16PORT.ptc.bit.ptc1 = 0x00; // Display D7
            EY16PORT.ptc.bit.ptc0 = 0x01; // Display D6
            EY16PORT.pta.bit.pta6 = 0x00; // Display D5
            EY16PORT.pta.bit.pta5 = 0x00; // Display D4
            break;
    }
}

```

```

case 0x05:
    EY16PORT.ptc.bit.ptc1 = 0x00; // Display D7
    EY16PORT.ptc.bit.ptc0 = 0x01; // Display D6
    EY16PORT.pta.bit.pta6 = 0x00; // Display D5
    EY16PORT.pta.bit.pta5 = 0x01; // Display D4
    break;

case 0x06:
    EY16PORT.ptc.bit.ptc1 = 0x00; // Display D7
    EY16PORT.ptc.bit.ptc0 = 0x01; // Display D6
    EY16PORT.pta.bit.pta6 = 0x01; // Display D5
    EY16PORT.pta.bit.pta5 = 0x00; // Display D4
    break;

case 0x07:
    EY16PORT.ptc.bit.ptc1 = 0x00; // Display D7
    EY16PORT.ptc.bit.ptc0 = 0x01; // Display D6
    EY16PORT.pta.bit.pta6 = 0x01; // Display D5
    EY16PORT.pta.bit.pta5 = 0x01; // Display D4
    break;

case 0x08:
    EY16PORT.ptc.bit.ptc1 = 0x01; // Display D7
    EY16PORT.ptc.bit.ptc0 = 0x00; // Display D6
    EY16PORT.pta.bit.pta6 = 0x00; // Display D5
    EY16PORT.pta.bit.pta5 = 0x00; // Display D4
    break;

case 0x09:
    EY16PORT.ptc.bit.ptc1 = 0x01; // Display D7
    EY16PORT.ptc.bit.ptc0 = 0x00; // Display D6
    EY16PORT.pta.bit.pta6 = 0x00; // Display D5
    EY16PORT.pta.bit.pta5 = 0x01; // Display D4
    break;

case 0x0A:
    EY16PORT.ptc.bit.ptc1 = 0x01; // Display D7
    EY16PORT.ptc.bit.ptc0 = 0x00; // Display D6
    EY16PORT.pta.bit.pta6 = 0x01; // Display D5
    EY16PORT.pta.bit.pta5 = 0x00; // Display D4
    break;

case 0x0B:
    EY16PORT.ptc.bit.ptc1 = 0x01; // Display D7
    EY16PORT.ptc.bit.ptc0 = 0x00; // Display D6
    EY16PORT.pta.bit.pta6 = 0x01; // Display D5
    EY16PORT.pta.bit.pta5 = 0x01; // Display D4
    break;

case 0x0C:
    EY16PORT.ptc.bit.ptc1 = 0x01; // Display D7
    EY16PORT.ptc.bit.ptc0 = 0x01; // Display D6
    EY16PORT.pta.bit.pta6 = 0x00; // Display D5
    EY16PORT.pta.bit.pta5 = 0x00; // Display D4
    break;

case 0x0D:
    EY16PORT.ptc.bit.ptc1 = 0x01; // Display D7
    EY16PORT.ptc.bit.ptc0 = 0x01; // Display D6
    EY16PORT.pta.bit.pta6 = 0x00; // Display D5
    EY16PORT.pta.bit.pta5 = 0x01; // Display D4
    break;

case 0x0E:
    EY16PORT.ptc.bit.ptc1 = 0x01; // Display D7
    EY16PORT.ptc.bit.ptc0 = 0x01; // Display D6
    EY16PORT.pta.bit.pta6 = 0x01; // Display D5
    EY16PORT.pta.bit.pta5 = 0x00; // Display D4
    break;

case 0x0F:

```

```

        EY16PORT.ptc.bit.ptc1 = 0x01; // Display D7
        EY16PORT.ptc.bit.ptc0 = 0x01; // Display D6
        EY16PORT.pta.bit.pta6 = 0x01; // Display D5
        EY16PORT.pta.bit.pta5 = 0x01; // Display D4
        break;
    }
}

/*****
Function Name      :      Delay
Engineer          :      C. Culshaw
Date              :      06/09/02
Parameters        :      delay_var - varies length of delay loop
Returns           :      None
Notes             :      Requires rewriting to make use of timer module
*****/
void Delay(unsigned long delay_var)
{
    char countloop = 0;
    unsigned long mainloop = 0;

    while (mainloop++ < delay_var)
        while (countloop++ < 200);
}

/*****
Function Name      :      main
Engineer          :      C. Culshaw
Date              :      06/09/02
Parameters        :      None
Returns           :      None
Notes             :      Continuously update the display with the latest data read
                    :      from the LIN bus.
                    :      Displays the id selected by PortC bit 2 (high = id 0x30,
                    :      low = id 0x20)
*****/
void main(void)
{
    LINDriverStatusType driver = LIN_STATUS_IDLE;
    unsigned char count = 0x00, msg_update = 0x00, msg_sel;

    Initialise_UUT(); // Put pcb & micro into default conditions//
    Disable_ICG();
    Initialise_Ports();
    Initialise_Display();

    asm cli; // Enable interrupts

    Clear_Display();

    while(1)
    {

        // Prepare display
        Cursor_Home();
        // Display "Message "
        for (count = 0; count < 8; count++)
        {
            Display_Data(Message[count], 0x01);
        }
    }
}

```

```

    }

    LIN_IdleClock();           // Increment bus activity counter

    // Determine if bus is active.
if ((driver = LIN_DriverStatus()) != LIN_STATUS_IDLE)
{
    // Determine which message Id will be displayed
    // Tie Port C2 high or low to select message 20 or 30
    msg_sel = (EY16PORT.ptc.bit.ptc2);

    // Read LIN message buffer
if (msg_sel)
{
    LIN_GetMsg (0x30, Message_30);
        msg_update = 0x30;
    }
    else
    {
        LIN_GetMsg (0x20, Message_20);
        msg_update = 0x20;
    }

switch (msg_update)
{
    case 0x20:
        DisplayChar(1, 9, '2');// Write '20' to line 1
        DisplayChar(1, 10, '0');
        Write_Line2(0x20);           // Write Rx data to line 2
        break;
    case 0x30:
        DisplayChar(1, 9, '3');// Write '30' to line 1
        DisplayChar(1, 10, '0');
        Write_Line2(0x30);           // Write Rx data to line 2
        break;
}
msg_update = 0;

    asm NOP;
}
else
{
// No message has been seen on bus for > LIN_IDLETIMEOUT
// Display No messages
    Clear_Display();
    Cursor_Home();
    for (count = 0; count < 11; count++)
    {
        Display_Data(No_messages[count], 0x01);
    }
    for (count = 0; count <16; count++)
    {
        DisplayChar(2, count, '.');
        DisplayChar(2, count+1, '.');
        DisplayChar(2, count, ' ');
    }
}
}
}
}

```

## vector.c

```

#define VECTOR_C
/*****
*
*   Copyright (C) 2001 Motorola, Inc.
*   All Rights Reserved
*
*   The code is the property of Motorola GSG St.Petersburg
*   Software Development
*   and is Motorola Confidential Proprietary Information.
*
*   The copyright notice above does not evidence any
*   actual or intended publication of such source code.
*
*   Filename:      $Source: /net/sdt/vault-rte/cvsroot/lin/release/hc08/hc08/vector.c,v $
*   Author:       $Author: kam $
*   Locker:      $Locker: $
*   State:       $State: Exp $
*   Revision:    $Revision: 1.17 $
*
*   Functions:    Vectors table for LIN08 Drivers with Motorola API
*
*   History:     Use the CVS command log to display revision history
*               information.
*
*   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(HC08AZ32)
extern void LIN_ISR_SCI_Receive(); /* SCI receive ISR */
extern void LIN_ISR_SCI_Error(); /* SCI error ISR */
// extern void BREAK_Command(); /* SWI ISR */

#if defined MASTER                /* (used for Master node only) */
extern void LIN_ISR_SCI_Transmit(); /* SCI transmit ISR */
extern void LIN_ISR_Timer0(); /* Timer Interface Module Channel 0 ISR */
#endif /* defined MASTER */
#endif /* defined(HC08AZ32) */

#if defined(HC08EY16)
extern void LIN_ISR_SCI_Receive(); /* ESCI receive ISR */
extern void LIN_ISR_SCI_Error(); /* ESCI error ISR */
// extern void BREAK_Command(); /* SWI ISR */

#if defined MASTER                /* (used for Master node only) */
extern void LIN_ISR_SCI_Transmit(); /* ESCI transmit ISR */
extern void LIN_ISR_Timer0(); /* Timer Interface Module Channel 0 ISR */
#endif /* defined MASTER */
#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) */

#if defined(COSMIC08)
#define Node_Startup  _stext
extern void _stext();           /* Cosmic compiler startup routine declaration */
#endif /* defined(COSMIC08) */

/*****
  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(COSMIC08)
#define LIN_VECTF (void *const)
void *const _vectab[] =
#endif /* defined(COSMIC08) */

#if defined(HC08AZ32)

/*****
/*      HC08AZ32
*****/

{
  LIN_VECTF NULL,          e          /* 0xFFD0  ADC          */
  LIN_VECTF NULL,          /* 0xFFD2  IRQ2/Keypad  */
#if defined(MASTER)
  LIN_VECTF LIN_ISR_SCI_Transmit, /* (used for Master node only)*/
  /* 0xFFD4  SCI transmit */
#endif /* defined(MASTER) */
#if defined(SLAVE)
  LIN_VECTF NULL,          /* 0xFFD4  SCI transmit */
#endif /* defined(SLAVE) */
  LIN_VECTF LIN_ISR_SCI_Receive, /* 0xFFD6  SCI receive  */
  LIN_VECTF LIN_ISR_SCI_Error,  /* 0xFFD8  SCI error    */
  LIN_VECTF NULL,          /* 0xFFDA  MSCAN Wakeup */
  LIN_VECTF NULL,          /* 0xFFDC  MSCAN Error  */
  LIN_VECTF NULL,          /* 0xFFDE  MSCAN Receive */
  LIN_VECTF NULL,          /* 0xFFE0  MSCAN Transmit */
  LIN_VECTF NULL,          /* 0xFFE2  SPI transmit */

```

```

    LIN_VECTF NULL,                /* 0xFFE4   SPI receive   */
    LIN_VECTF NULL,                /* 0xFFE6   TIMER B overflow */
    LIN_VECTF NULL,                /* 0xFFE8   TIMER B channel 1 */
    LIN_VECTF NULL,                /* 0xFFEA   TIMER B channel 0 */
    LIN_VECTF NULL,                /* 0xFFEC   TIMER A overflow */
    LIN_VECTF NULL,                /* 0xFFEE   TIMER A channel 3 */
    LIN_VECTF NULL,                /* 0xFFF0   TIMER A channel 2 */
    LIN_VECTF NULL,                /* 0xFFF2   TIMER A channel 1 */
#if defined(MASTER)                /* (used for Master node only)*/
    LIN_VECTF LIN_ISR_Timer0,      /* 0xFFF4   TIMER A channel 0 */
#endif /* defined(MASTER) */
#if defined(SLAVE)
    LIN_VECTF NULL,                /* 0xFFF4   TIMER A channel 0 */
#endif /* defined(SLAVE) */
    LIN_VECTF NULL,                /* 0xFFF6   PIT           */
    LIN_VECTF NULL,                /* 0xFFF8   PLL           */
    LIN_VECTF NULL,                /* 0xFFFA   IRQ1         */
// LIN_VECTF BREAK_Command,      /* 0xFFFC   SWI          */
    LIN_VECTF NULL,                /* 0xFFFC   SWI          */
    LIN_VECTF Node_Startup        /* 0xFFFE   RESET        */
};

#endif /* defined(HC08AZ32) */

#if defined(HC08EY16)

/*****
/*   HC08EY16
*****/

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

```

```
};

#endif /* defined(HC08EY16) */

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

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

---

### slave.cfg

---

```
#ifndef LINCFG_H
#define LINCFG_H

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

#if defined (HC08)

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

/* External MCU frequency = 8MHz       */
/* SCI Baud rate          = 19K2       */

/*
   This definition configures the LIN bus baud rate.
```



```
This value shall be set according to target MCU
SCI register usage.
HC908EY16: the 8-bit value will be masked by 0x37
and put into SCBR register.
*/
/* Selects 9600 baud rate if using a 9.8304MHz crystal */
/*#define LIN_BAUDRATE          0x04u*/

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

/*
This definition set 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      400u

#endif /* defined (HC08) */

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

---

## slave.id

---

```

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

/* In this application the LIN 'monitor' software is only monitoring network
activity, hence all messages are LIN_RECEIVE only.
Change to LIN_SEND if the LIN node is required to transmit
*/

#define LIN_MSG_09  LIN_RECEIVE
#define LIN_MSG_0A  LIN_RECEIVE
#define LIN_MSG_20  LIN_RECEIVE
#define LIN_MSG_21  LIN_RECEIVE
#define LIN_MSG_30  LIN_RECEIVE

/* this string is not necessary - just as an example */
#define LIN_MSG_20_LEN4  /* standard length */
#define LIN_MSG_09_LEN2  /* standard length */
#define LIN_MSG_0A_LEN2  /* standard length */
#define LIN_MSG_21_LEN  4  /* standard length */
#define LIN_MSG_30_LEN  8  /* standard length */

#endif /* defined(LINMSGID_H)*/

```

---

## hc08ey16.prm

---

```
LINK slave.abs

NAMES
ansi.lib
    ..\..\..\lib\hicross08\lin08EYs.lib
/* other object files to link are passed from the IDF with the linker -Add option */
END

SECTIONS
    LIN_ZRAM    = READ_WRITE 0x0040 TO 0x00FF;        /* zero page*/
    LIN_RAM     = READ_WRITE 0x0100 TO 0x01FF;        /* program data */
    LIN_STACK   = READ_WRITE 0x0200 TO 0x023F;        /* stack*/
    LIN_ROM     = READ_ONLY  0xC000 TO 0xFDF;         /* program code & constants */
    LIN_VECTORS = READ_ONLY  0xFFDC TO 0xFFFF;        /* interrupt vectors (use your vector.obj) */
END

PLACEMENT
    ZeroSeg, _DATA_ZEROPAGE INTO LIN_ZRAM;
    DEFAULT_ROM, ROM_VAR    INTO LIN_ROM;
    DEFAULT_RAM             INTO LIN_RAM;
    SSTACK                  INTO LIN_STACK;
    VECTORS_DATA            INTO LIN_VECTORS;
END

STACKSIZE 0x0040

ENTRIES
    _vectab
END
INIT      _Startup /* contains line replacing default _PRESTART*/
MAPFILE ON
```

---

---

## Appendix C – Header Files

### ey16port.h

---

```

/****

Filename : EY16port.h

Defines the port block as a datastructure

Datastructure base address (out of reset) -

EY16      : 0x0000

Written by Ross McLuckie (R38917)

Revision history -

7/2/00   - Initial coding
8/1/02   - Modified for the HC908EY16 (TMCh)

****/

#ifndef EY16PORT_H           //prevent duplicate includes
#define EY16PORT_H

#ifndef COMMON_H //prevent duplicate includes
#include <common.h>
#endif

typedef union uPTA
{
    tU08  byte;
    struct
    {
        tU08  pta0    :1;           //i/o port pins
        tU08  pta1    :1;
        tU08  pta2    :1;
        tU08  pta3    :1;
        tU08  pta4    :1;
        tU08  pta5    :1;
        tU08  pta6    :1;
        tU08           :1;           //not used
    }bit;
}tPTA;

#define PTA0    0x01           //bit masks
#define PTA1    0x02
#define PTA2    0x04
#define PTA3    0x08
#define PTA4    0x10
#define PTA5    0x20
#define PTA6    0x40

typedef union uPTB
{

```

```

tU08  byte;
struct
{
    tU08 ptb0    :1;        //i/o port pins
    tU08 ptb1    :1;
    tU08 ptb2    :1;
    tU08 ptb3    :1;
    tU08 ptb4    :1;
    tU08 ptb5    :1;
    tU08 ptb6    :1;
    tU08 ptb7    :1;
}bit;
}tPTB;

#define PTB00x01 //bit masks
#define PTB1     0x02
#define PTB2     0x04
#define PTB3     0x08
#define PTB4     0x10
#define PTB5     0x20
#define PTB6     0x40
#define PTB7     0x80

typedef union uPTC
{
    tU08  byte;
    struct
    {
        tU08 ptc0    :1;        //i/o port pins
        tU08 ptc1    :1;
        tU08 ptc2    :1;
        tU08 ptc3    :1;
        tU08 ptc4    :1;
        tU08        :3;        //not used
    }bit;
}tPTC;

#define PTC00x01 //bit masks
#define PTC1     0x02
#define PTC2     0x04
#define PTC3     0x08
#define PTC4     0x10

typedef union uPTD
{
    tU08  byte;
    struct
    {
        tU08 ptd0    :1;        //i/o port pins
        tU08 ptd1    :1;
        tU08        :6;        //not used
    }bit;
}tPTD;

#define PTD0     0x01        //bit masks
#define PTD1     0x02

typedef union uDDRA
{
    tU08  byte;

```

```

struct
{
    tU08 ddra0    :1;    //data direction bits (0:input;1:output)
    tU08 ddra1    :1;
    tU08 ddra2    :1;
    tU08 ddra3    :1;
    tU08 ddra4    :1;
    tU08 ddra5    :1;
    tU08 ddra6    :1;
    tU08          :1;    //not used
}bit;
}tDDRA;

#define DDRA0    0x01    //bit masks
#define DDRA1    0x02
#define DDRA2    0x04
#define DDRA3    0x08
#define DDRA4    0x10
#define DDRA5    0x20
#define DDRA6    0x40

typedef union uDDRB
{
    tU08 byte;
    struct
    {
        tU08 ddrb0    :1;    //data direction bits (0:input;1:output)
        tU08 ddrb1    :1;
        tU08 ddrb2    :1;
        tU08 ddrb3    :1;
        tU08 ddrb4    :1;
        tU08 ddrb5    :1;
        tU08 ddrb6    :1;
        tU08 ddrb7    :1;
    }bit;
}tDDRB;

#define DDRB0    0x01    //bit masks
#define DDRB1    0x02
#define DDRB2    0x04
#define DDRB3    0x08
#define DDRB4    0x10
#define DDRB5    0x20
#define DDRB6    0x40
#define DDRB7    0x80

typedef union uDDRC
{
    tU08 byte;
    struct
    {
        tU08 ddrc0    :1;    //data direction bits (0:input;1:output)
        tU08 ddrc1    :1;
        tU08 ddrc2    :1;
        tU08 ddrc3    :1;
        tU08 ddrc4    :1;
        tU08          :2;    //not used
        tU08 molken:1;
    }bit;
}tDDRC;

```

```

#define DDRC0      0x01      //bit masks
#define DDRC1      0x02
#define DDRC2      0x04
#define DDRC3      0x08
#define DDRC4      0x10
#define MCLKEN     0x80

typedef union uDDR0
{
    tU08  byte;
    struct
    {
        tU08 ddr0      :1;      //data direction bits (0:input;1:output)
        tU08 ddr1      :1;
        tU08      :6;      //not used
    }bit;
}tDDR0;

#define DDRD0      0x01      //bit masks
#define DDRD1      0x02
#define DDRD2      0x04
#define DDRD3      0x08
#define DDRD4      0x10
#define DDRD5      0x20
#define DDRD6      0x40
#define DDRD7      0x80

typedef union uPTE
{
    tU08  byte;
    struct
    {
        tU08 pte0      :1;      //i/o port pins
        tU08 pte1      :1;
        tU08      :6;      //not used
    }bit;
}tPTE;

#define PTE0       0x01      //bit masks
#define PTE1       0x02

typedef union uDDRE
{
    tU08  byte;
    struct
    {
        tU08 ddre0     :1;      //data direction bits (0:input;1:output)
        tU08 ddre1     :1;
        tU08      :6;      //not used
    }bit;
}tDDRE;

#define DDRE0      0x01      //bit masks
#define DDRE1      0x02

typedef struct      //port control
{
    volatile tPTA   pta;      //port A data register
}

```

```
volatile tPTB  ptb;    //port B data register
volatile tPTC  ptc;    //port C data register
volatile tPTD  ptd;    //port D data register
        tDDRA  ddra;   //port A data direction register
        tDDRB  ddrb;   //port B data direction register
        tDDRC  ddrc;   //port C data direction register
        tDDRD  ddrd;   //port D data direction register
volatile tPTE  pte;    //port E data register
        tU08   rsv;    //maintaining memory map
        tDDRE  ddre;   //port E data direction register
}tEY16PORT;

#endif //EY16PORT_H
```

---



## ey16icg.h

---

```

/****

Filename : EY16icg.h

Defines the serial internal clock generation register block as a datastructure

ICG datastructure base address (out of reset) -

EY16      : 0x0036

Written by Ross McLuckie (R38917)

Revision history -

7/5/02   - Initial coding

****/

#ifndef EY16ICG_H           //prevent duplicated includes
#define EY16ICG_H

#ifndef COMMON_H           //prevent duplicated includes
#include <common.h>
#endif

typedef union uICGCR
{
    tU08  byte;
    struct
    {
        tU08  ecgs      :1;      //transmit interrupt enable
        tU08  ecgon     :1;      //spi enable
        tU08  icgs      :1;      //wired-OR mode
        tU08  icgon     :1;      //clock phase
        tU08  cs        :1;      //clock polarity
        tU08  cmon      :1;      //master mode (1), slave mode (0)
        tU08  cmf       :1;      //not used
        tU08  cmie      :1;      //receiver interrupt enable
    }bit;
    }tICGCR;

#define ECGS      0x01      //bit masks
#define ECGON     0x02
#define ICGS      0x04
#define ICGON     0x08
#define CS        0x10
#define CMON      0x20
#define CMF       0x40
#define CMIE      0x80

typedef union uICGMR
{
    tU08  byte;
    struct
    {
        tU08  n0        :1;      //spi baud rate bits
        tU08  n1        :1;      //mode fault enable
    }
}

```

```

    tU08 n2      :1;      //transmitter empty flag
    tU08 n3      :1;      //mode fault flag
    tU08 n4      :1;      //receiver overflow flag
    tU08 n5      :1;      //error interrupt enable
    tU08 n6      :1;      //receiver full flag
    tU08         :1;      //not used
}bit;
}tICGMR;

```

```

#define N0      0x01      //bit masks
#define N1      0x02
#define N2      0x04
#define N3      0x08
#define N4      0x10
#define N5      0x20
#define N6      0x40

```

```

typedef union uICGTR
{
    tU08 byte;
    struct
    {
        tU08 trim0   :1;
        tU08 trim1   :1;
        tU08 trim2   :1;
        tU08 trim3   :1;
        tU08 trim4   :1;
        tU08 trim5   :1;
        tU08 trim6   :1;
        tU08 trim7   :1;
    }bit;
}tICGTR;

```

```

#define TRIM0    0x01      //bit masks
#define TRIM1    0x02
#define TRIM2    0x04
#define TRIM3    0x08
#define TRIM4    0x10
#define TRIM5    0x20
#define TRIM6    0x40
#define TRIM7    0x80

```

```

typedef union uICGDVR
{
    tU08 byte;
    struct
    {
        tU08 ddiv0   :1;      //parity type
        tU08 ddiv1   :1;      //parity enable
        tU08 ddiv2   :1;      //idle line type
        tU08 ddiv3   :1;      //wake up by address mark/idle
        tU08         :4;      //not used
    }bit;
}tICGDVR;

```

```

#define DDIV0    0x01      //bit masks
#define DDIV1    0x02
#define DDIV2    0x04
#define DDIV3    0x08

```

```

typedef union uICGDSR
{
    tU08  byte;
    struct
    {
        tU08 dstg0    :1;    //send break character
        tU08 dstg1    :1;    //receiver wake-up control
        tU08 dstg2    :1;    //receiver enable
        tU08 dstg3    :1;    //transmitter enable
        tU08 dstg4    :1;    //idle line interrupt enable
        tU08 dstg5    :1;    //receiver interrupt enable
        tU08 dstg6    :1;    //transmit complete interrupt enable
        tU08 dstg7    :1;    //transmit interrupt enable
    }bit;
    }tICGDSR;

#define DSTG0    0x01    //bit masks
#define DSTG1    0x02
#define DSTG2    0x04
#define DSTG3    0x08
#define DSTG4    0x10
#define DSTG5    0x20
#define DSTG6    0x40
#define DSTG7    0x80

typedef struct
{
    volatile tICGCR    icgcr; //icg control register
    volatile tICGMR    icgmr; //icg multiplier register
    volatile tICGTR    icgtr; //icg trim register
    volatile tICGDVR    icgdvr; //icg divider control register
    volatile tICGDSR    icgdsr; //icg dco stage control register
    }tEY16ICG;

#endif //EY16ICG_H

```

## ey16config.h

```

/****

Filename : ey16config.h

Defines the config registers as a datastructure

base address (out of reset) -

EY16      : 0x001E

Written by Ross McLuckie (R38917)

Revision history -

8/2/00   - Initial coding

****/

#ifndef EY16CONFIG_H          //prevent duplicated includes
#define EY16CONFIG_H

#ifndef COMMON_H             //prevent duplicated includes
#include <common.h>
#endif

typedef union uEY16CONFIG2
{
    tU08    byte;
    struct
    {
        tU08    ssbpuenb          :1;          //ssb pull up enable bit
        tU08    oscennstop        :1;          //oscillator enable in stop mode bit
        tU08    tmbclkssel        :1;          //cop long timeout bit
        tU08    extclken          :1;          //short stop recovery bit
        tU08    extslow           :1;          //lvi power enable bit
        tU08    extxtalen         :1;          //lvi reset bit
        tU08                                :2;          //not used
    }bit;
}tEY16CONFIG2;

#define      SSBPUENB            0x01
#define      OSCENNSTOP          0x02
#define      TMBCLKSEL           0x04
#define      EXTCLKEN            0x08
#define      EXTSLOW             0x10
#define      EXTXTALEN           0x20

typedef union uEY16CONFIG1
{
    tU08    byte;
    struct
    {
        tU08    copd              :1;          //cop disable bit
        tU08    stop              :1;          //stop enable bit
        tU08    ssrec             :1;          //short stop recovery bit
        tU08    lvi5or3           :1;          //lvi 5-v or 3-v operating mode bit
        tU08    lvipwr            :1;          //lvi power disable bit
    }
}

```

```

    tU08      lvirstd      :1;          //lvi reset disable bit
    tU08      lvistop     :1;          //lvi enable in stop mode bit
    tU08      coprs       :1;          //cop rate select bit
}bit;
}tEY16CONFIG1;

#define      COPD          0x01
#define      STOP          0x02
#define      SSREC         0x04
#define      LVI5OR3       0x08
#define      LVIPWRD       0x10
#define      LVIRSTD       0x20
#define      LVISTOP       0x40
#define      COPRS         0x80

typedef struct
{
    tEY16CONFIG2      ey16config2;      //timer count registers
    tEY16CONFIG1      ey16config1;      //timer modulo registers
}tEY16CONFIG;

#endif //EY16CONFIG_H

```

---

## hc908ey16.h

---

```

/*****
HC08EY16.H
Register and bit definitions for the 908EY16

T. McHenry          08-01-02
*****/

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

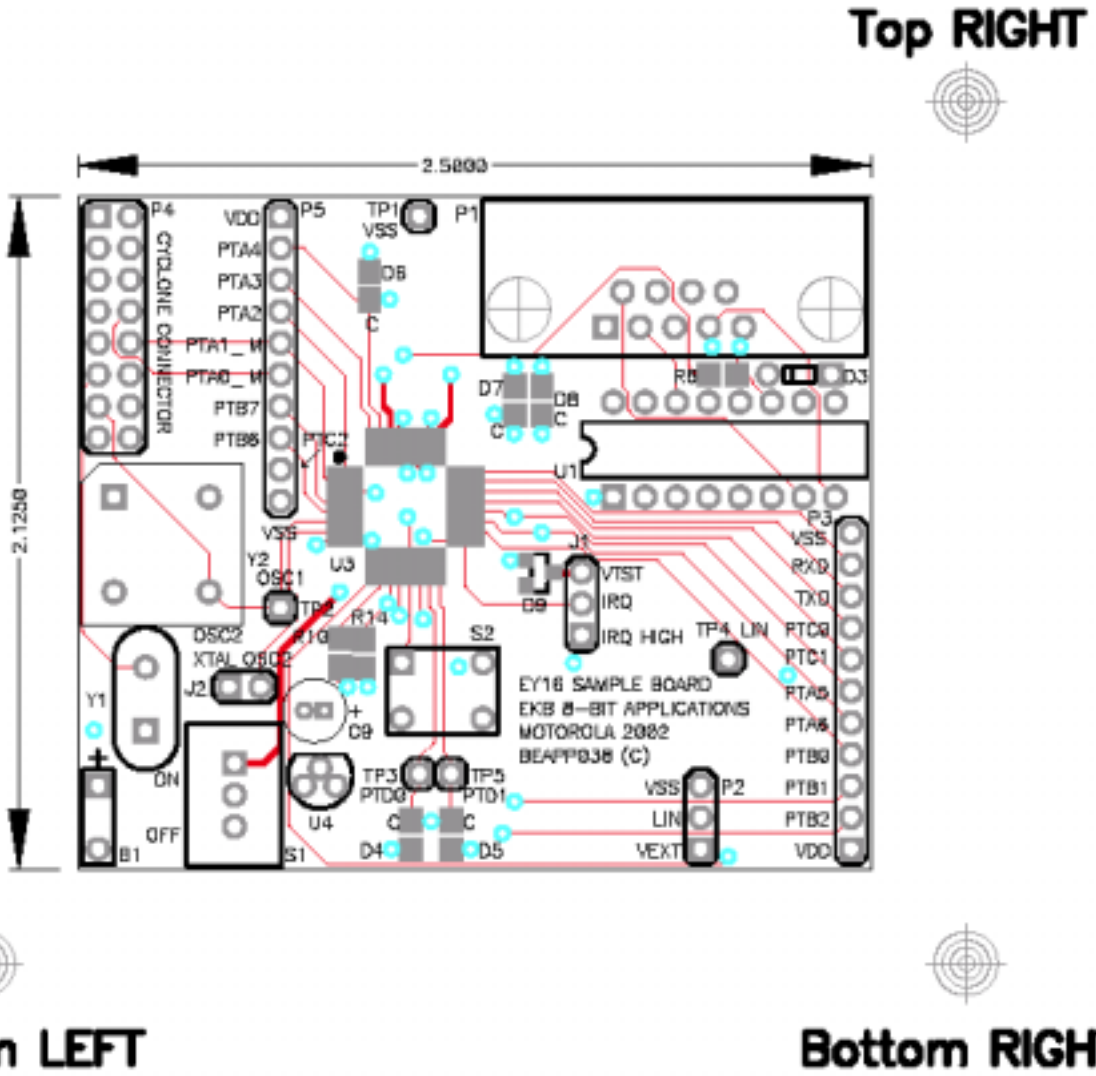
#define      ICGCR        *((volatile unsigned char *)0x0036)

```

---



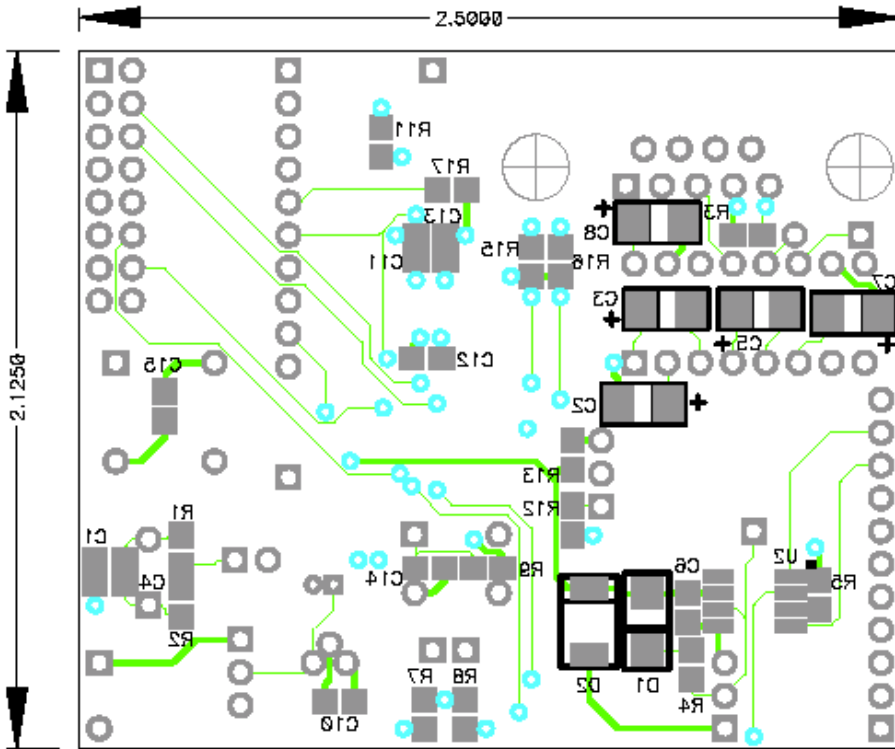
Appendix E – Layout Top View



Freescale Semiconductor, Inc.

Appendix E – Layout Bottom view

Top RIGHT



Bottom LEFT



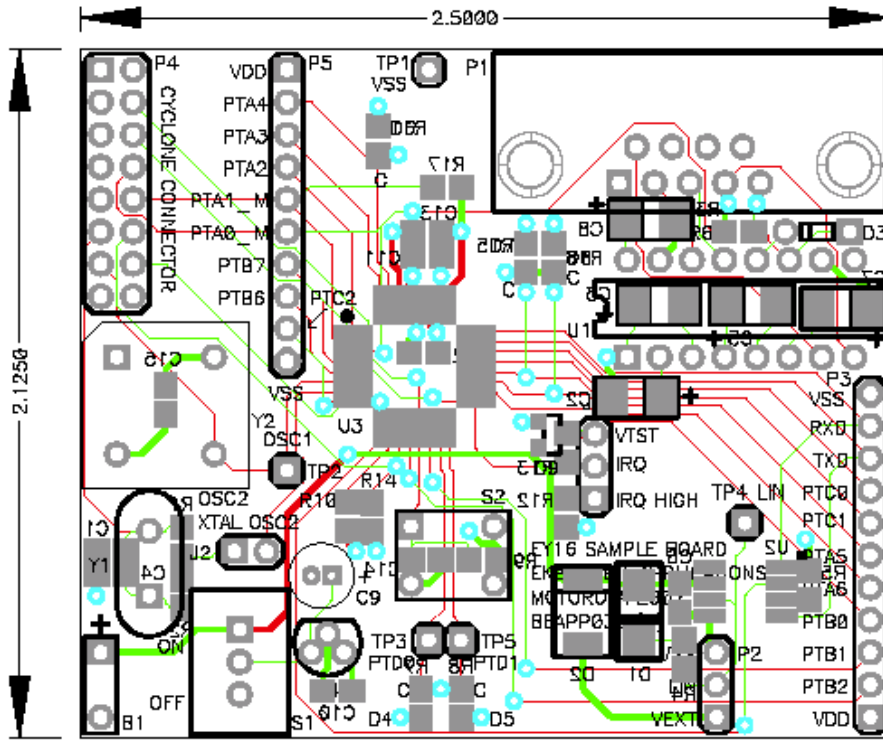
Bottom RIGHT

Freescale Semiconductor, Inc.



Appendix E – Layout Combined

Top RIGHT



Bottom LEFT

Bottom RIGHT

Freescale Semiconductor, Inc.





# Freescale Semiconductor, Inc.

## 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>

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

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Motorola data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part.



Motorola and the Stylized M Logo are registered in the U.S. Patent and Trademark Office. digital dna is a trademark of Motorola, Inc. All other product or service names are the property of their respective owners. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

© Motorola, Inc. 2002

AN2343/D

**For More Information On This Product,  
Go to: [www.freescale.com](http://www.freescale.com)**