Freescale Semiconductor
Application Note

AN2960
Rev. 0, 7/2005

Window Lift/Sunroof LIN 2.0 Slave

Based on the MC68HC908EY16 Microcontroller

by: Zdenek Kaspar
8/16-bit Systems Engineering
Roznov p. R., Czech Republic

Introduction

This application note describes the implementation of a LIN 2.0 enabled window lift/sunroof slave module based on the MC68HC908EY16, a Freescale 8-bit microcontroller unit (MCU).

The application demonstrates the capability and performance of the MC68HC908EY16 MCU by implementing LIN 2.0 connectivity and brush DC motor control routines. It does not include the implementation of sophisticated window lift control, which requires a combination of Hall sensor and current measurements; only simplified obstruction and stall detection based on current measurement is presented.

The implementation described is intended to be as reusable as possible, to help reduce the development time when introducing this kind of connectivity to a product.

The implementation is based on the latest available revision of the specification for Local Interconnect Network (LIN) LIN2.0, which was introduced in September 2003 (see Reference [4] for more information).

The LIN 2.0 connectivity of this project is based on the Volcano Automotive (www.volcanoautomotive.com) LIN Target Package (VCT LTP 2.0, Reference [5]). This application note also helps to introduce this tool. The main benefit of using this software package is its high level approach to creating and rebuilding a LIN cluster. This greatly simplifies the LIN cluster generation process, allowing the designer to focus fully on the application. For a general overview of the implementation and use of this software package, see AN2767 (Reference [1]).

This application note complements the following Freescale LIN 2.0 related application notes:

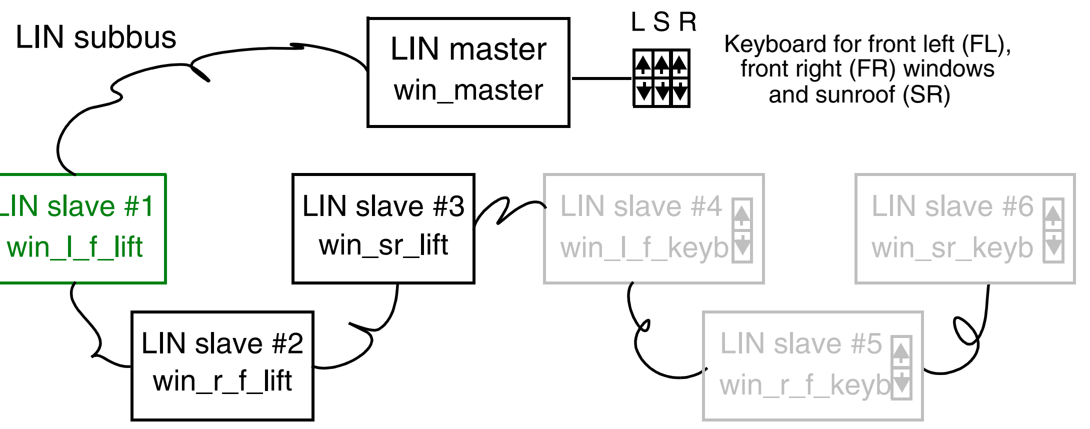
- AN2767 — LIN 2.0 Connectivity on Freescale 8/16-bit MCUs Using Volcano LTP (Reference [1]).
- AN2884 — LIN 2.0 Door Lock Slave Based on the MC68HC908GR16 MCU and the LIN 2.0 Communication Protocol (Reference [2]).
- AN2885 — LIN 2.0 Mirror Unit Slave Based on the MC68HC908EY16 MCU and the LIN 2.0 Communication Protocol, (Reference [3]).

The complete application software for the window lift/sunroof LIN 2.0 slave unit (AN2960SW) is downloadable from the Freescale website at www.freescale.com.

General Description

LIN Cluster Introduction

The window lift LIN cluster project outline and node naming convention are shown in Figure 1. The LIN slave application presented is dedicated to the left front window lift control module, and is called *win_l_f_lift.* However, the application can be modified easily to implement other units, such as the right front slave unit (*win_r_f_lift*).



node in green — node developed in this application
nodes in black — nodes included in message structure of this application
nodes in grey — other nodes of possible window lift cluster configuration

Figure 1. Designed LIN Cluster Outline

The message structure of the window lift LIN cluster is shown in Appendix A — Messaging Strategy.

According to the LIN 2.0 spec package (specifically the LIN Configuration Language Specification part), every LIN cluster can be described by a single file called the LIN Description File (*window_lift_net.ldf* in the case of the presented window lift application). Additional information, specific to the node of interest, is stored in the *window_l_f_lift_slave.prv* and *uart.cfg* files. All three files are provided in Appendix B — LIN Description File and Node Specific Files.

Slave Node Concept and Features

The node is based on the LINkit board for the MC68HC908EY16. (An evaluation board is available for most LIN 8/16-bit MCUs — see Reference [6] for more information.)

The MC33486, a dual high side switch for H-bridge automotive applications, is used for DC motor power control.

The node concept is depicted in Figure 2.

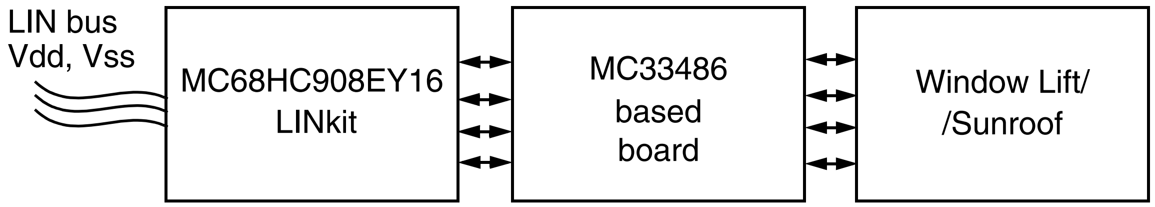


Figure 2. Slave Node Concept

The functionality implemented in the window lift/sunroof LIN 2.0 slave node is as follows:

- Normal running mode (including LIN 2.0 connectivity, window lift obstruction and stall detection, typical window lift command handling)
- Low-power modes (including LIN sleep and wake-up from LIN sleep modes)

In **normal running mode**, the device controls a DC motor using typical window lift commands provided by the master:

- Open window (active as long as the respective button is pressed)
- Open window completely
- Close window (active as long as the respective button is pressed)
- Close window completely
- Monitor an error that occurred during the last action

The slave also informs the master about its status:

- Report an error that occurred during the last action (signal name *win_l_f_err*)
- Report the LIN response error state (signal name *win_l_f_resp_err*)[1].

For the DC motor control related routines, the following features are included:

- Obstruction and stall detection, based on current measurement
- "Soft start" of the window lift motor (using PWM ramping-up).

In **low-power mode**, the use of the LIN go-to-sleep and wake-up commands is demonstrated. The slave node is wakened from low-power mode upon any LIN bus activity. On the other hand, MCU low-power mode is entered when the "go-to-sleep" LIN master request frame is received. Moreover, the application supports automatic "go-to-sleep" transition after four seconds of LIN bus inactivity, which is mandatory for all LIN 2.0 slaves.

To minimize current consumption, the node low-power mode in the described application implies an MCU power-off.

1. This mechanism allows the master to monitor LIN response error states of all slaves connected to the network.

Freescale Components Used

MC68HC908EY16 Microcontroller Unit

The MC68HC908EY16 is a member of the low-cost, high-performance M68HC08 family of 8-bit Freescale MCUs.

MC68HC908EY16 features (Reference [7]):

- High-performance M68HC08 architecture optimized for C-compilers
- 8 MHz internal bus frequency at 5 V
- Internal clock generator module (ICG) with no external components
  - Software selectable bus frequencies
  - 25% accuracy with a trimming capability of better than 1%
  - Clock monitor
  - Option to allow use of external clock source or external crystal / ceramic resonator
- 15,872 bytes of on-chip FLASH memory with in-circuit programming
- 512 bytes of on-chip RAM
- Low voltage inhibit module (LVI)
- Internal clock generator module (ICG)
- Two 16-bit, 2-channel timer (TIMA and TIMB) interface modules with selectable input capture, output compare, and pulse-width modulation (PWM) capability on each channel
- 8-channel, 10-bit successive approximation analog-to-digital converter (ADC)
- Enhanced serial communications interface (ESCI)
- Serial peripheral interface (SPI)
- Timebase module (TBM)
- 5-bit keyboard interrupt (KBI) with wake-up feature
- 24 general I/O pins
- External asynchronous interrupt pin (/IRQ)
- System protection features:
  - Optional computer operating properly (COP) reset
  - Illegal opcode detection with reset
  - Illegal address detection with reset
- Standard low-power modes of operation:
  - Wait mode
  - Stop mode

The MCU structure is shown in Figure 3.

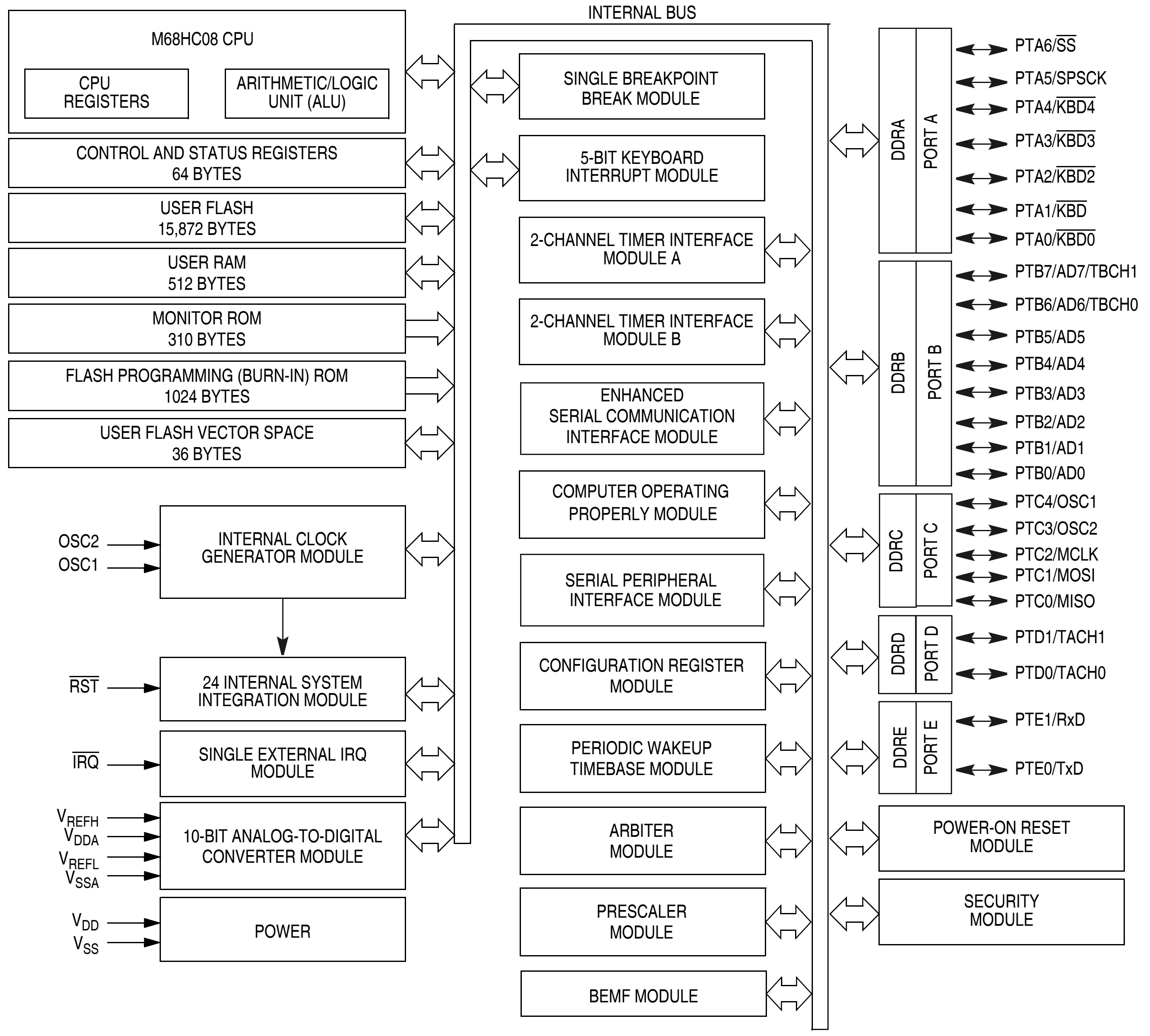

INTERNAL BUS
M68HC08 CPU
CPU REGISTERS
ARITHMETIC/LOGIC UNIT (ALU)
CONTROL AND STATUS REGISTERS 64 BYTES
USER FLASH 15,872 BYTES
USER RAM 512 BYTES
MONITOR ROM 310 BYTES
FLASH PROGRAMMING (BURN-IN) ROM 1024 BYTES
USER FLASH VECTOR SPACE 36 BYTES
OSC2
OSC1
INTERNAL CLOCK GENERATOR MODULE
RST
24 INTERNAL SYSTEM INTEGRATION MODULE
IRQ
SINGLE EXTERNAL IRQ MODULE
VREFH
VDDA
VREFL
VSSA
10-BIT ANALOG-TO-DIGITAL CONVERTER MODULE
VDD
VSS
POWER
SINGLE BREAKPOINT BREAK MODULE
5-BIT KEYBOARD INTERRUPT MODULE
2-CHANNEL TIMER INTERFACE MODULE A
2-CHANNEL TIMER INTERFACE MODULE B
ENHANCED SERIAL COMMUNICATION INTERFACE MODULE
COMPUTER OPERATING PROPERLY MODULE
SERIAL PERIPHERAL INTERFACE MODULE
CONFIGURATION REGISTER MODULE
PERIODIC WAKEUP TIMEBASE MODULE
ARBITER MODULE
PRESCALER MODULE
BEMF MODULE
DDRA
PORT A
DDRB
PORT B
DDRC
PORT C
DDRD
PORT D
DDRE
PORT E
PTA6/SS
PTA5/SPSCK
PTA4/KBD4
PTA3/KBD3
PTA2/KBD2
PTA1/KBD
PTA0/KBD0
PTB7/AD7/TBCH1
PTB6/AD6/TBCH0
PTB5/AD5
PTB4/AD4
PTB3/AD3
PTB2/AD2
PTB1/AD1
PTB0/AD0
PTC4/OSC1
PTC3/OSC2
PTC2/MCLK
PTC1/MOSI
PTC0/MISO
PTD1/TACH1
PTD0/TACH0
PTE1/RxD
PTE0/TxD
POWER-ON RESET MODULE
SECURITY MODULE

Figure 3. MC68HC908EY16 MCU Block Diagram

MC33399 LIN Physical Interface

The MC33399 (Reference [8]) is a LIN sub bus physical layer interface with an implemented bus wake-up capability, whose block diagram can be found in Figure 4.

MC33399 device features:

- Communication speed from 1to 20 kbps
- Nominal operation from Vsup 8 to 18 V DC

- Interfaces to the MCU with CMOS compatible I/O pins
- Two operational modes: Normal and Sleep
- Very low standby current of 20 µA during Sleep mode
- An unpowered node does not disturb the LIN network
- Wake up capability from the LIN bus, MCU, or by high voltage on the wake-up pin
- Controls an external voltage regulator (via INH pin)
- High EMC immunity

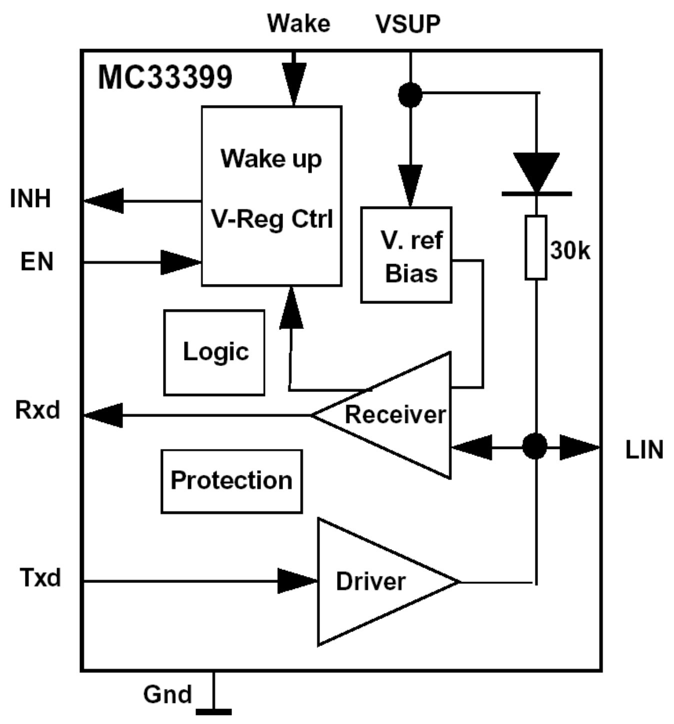


Figure 4. MC33399 (LIN Physical Interface) Block Diagram

NOTE

The new enhanced LIN physical interface MC33661 (Reference [9]) fully replaces the MC33399 described above. With a signal slew rate selection option, active bus wave shaping, and a special mode for operating above 100 kbps for testing and programming, it provides excellent radiated emission performance.

MC33486 Dual High Side Switch for H-bridge Automotive Applications

This device is a dual high side switch for automotive applications that incorporates a dual low side switch control feature. This device is designed to monitor two low side switches for typical DC motor control in an H-bridge configuration. It can be interfaced directly with a microcontroller for control and diagnostic functions, is PWM capable, and has a self-adjusted switching speed to minimize electromagnetic emission.

MC33486 (Reference [10]) features:

- 10 A nominal DC current
- 35 A maximum peak current
- Operating voltage from 8 V to 28 V
- Up to 30 kHz PWM capability
- Current recopy to monitor the high side current
- Overvoltage detection: switches off device when Vbat exceeds 28 V
- High side and low side overcurrent protection

An MC33486 simplified block diagram and a typical application are shown in Figure 5.

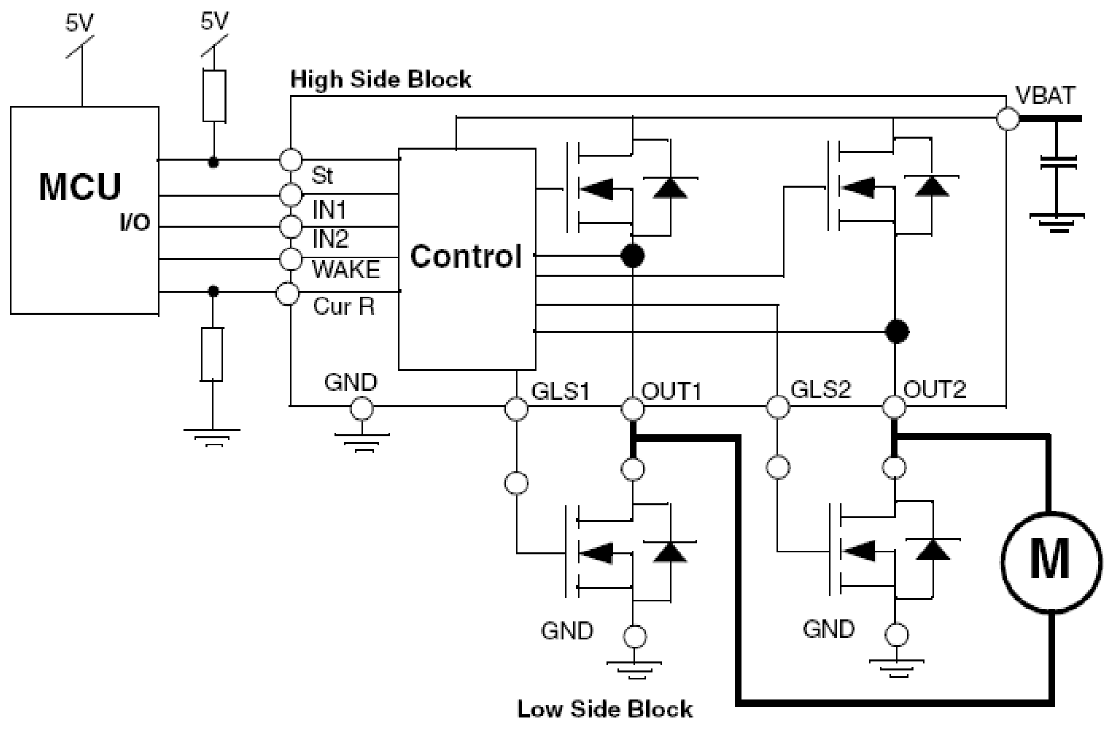


Figure 5. MC33486 Simplified Block Diagram and Typical Application

Hardware Description

This section describes the major steps in the development of the hardware.

A hardware schematic of the window lift LIN slave module is shown in Figure 6. It shows the connections between the MC68HC908EY16 LINkit board and the MC33486 dual high side switch. (Note that the schematic diagram shows only those MCU pins relevant to the application. Other pins, such as those for LINkit debugging support, and unused MCU pins are not depicted.)

Jumpers JP1 and JP2 are used only during programming and debugging of the MCU. For more information about LINkit board programming, or about the MC68HC908EY16 LINkit board, see Reference [6].

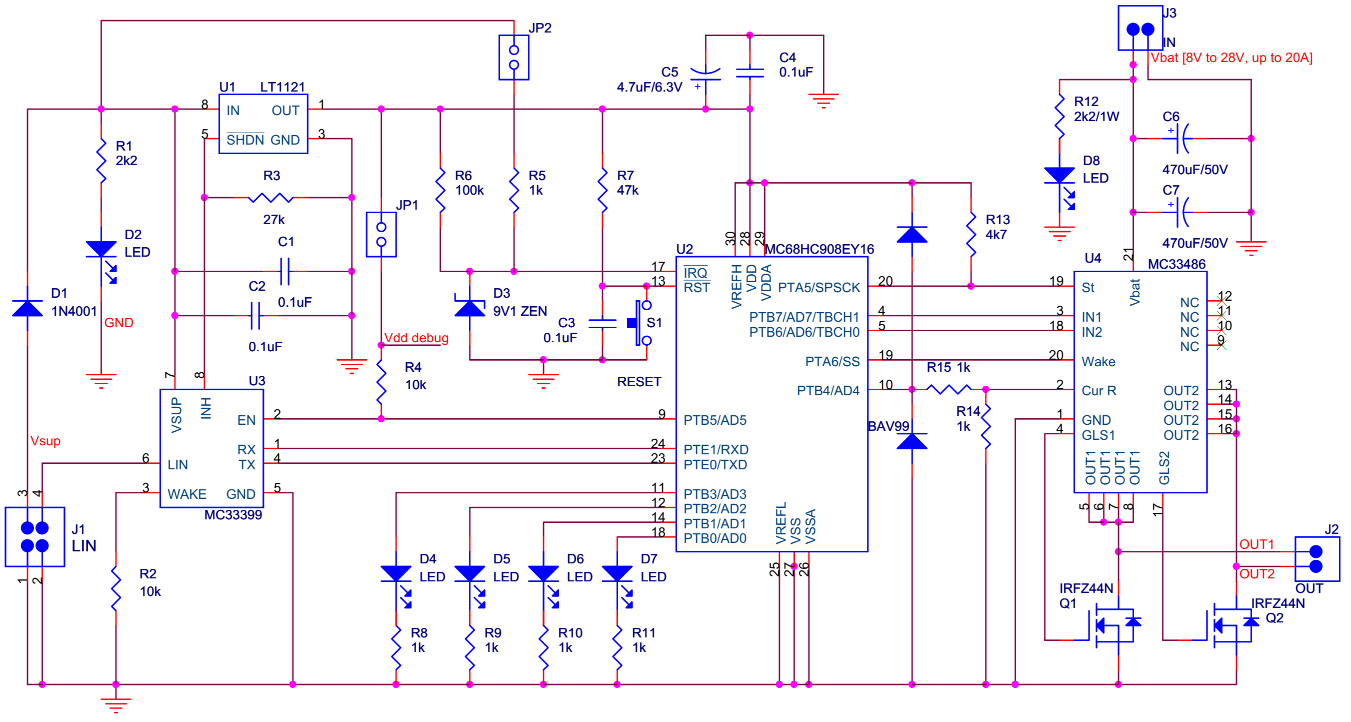


Figure 6. Window Lift Module Schematic Diagram

NOTE

LINkit boards are fitted with the MC33399 LIN physical interface. As the new, pin compatible, MC33661 LIN physical interface with many enhancements (especially slew rate control and low EMC emission) is now available, the MC33661 is recommended instead of the MC33399.

CAUTION (when using MC33661)

LINkit slave boards use the MC33399 to control the LT1121 regulator, with a direct connection between the MC33399 INH pin and the LT1121 SHDN *pin. Although this is a 12 V output from the MC33399, which is clamped to about 7 V on the LT1121, the current is only about 200 μA, owing to the low drive capability of the INH pin. On the MC33661, however, the drive capability is much higher (35 Ω typical output impedance).*

Therefore, to prevent excessive current from flowing into the LT1121 from the MC33661, a 10 kΩ series resistor must be added between the MC33661 INH pin and the LT1121 $\overline{SHDN}$ pin.

To connect the MC33486 to the MCU, it is necessary to add pullup resistor R13 to enable reading of the status pin St. It is also necessary to connect resistor R14 for current sensing measurement. As the nominal current recopy ratio of the MC33486 is 3700[1], the following equation can be written:

$$U_{CRnom} = \frac{I_{OUT}}{3700} \times R_{CR}$$

where:

- U_{CRnom} is a nominal voltage value of the current recopy measure pin (PTB4/AD4)
- I_{OUT} is current of the connected motor
- R_{CR} is a current recopy resistor (R14)

The value of R_{CR} is selected to be 1 kΩ. Thus the voltage on the PTB4/AD4 pin, on condition that I_{OUT} = 10A, can be calculated as follows.

$$U_{CRnom} = \frac{I_{OUT}}{3700} \times R_{CR} = \frac{10\,A}{3700} \times 1000\Omega \cong 2,7V$$

This voltage value is then converted by the ADC of the MC68HC908EY16 (ADC 8-bit accuracy selected), so resulting in an unsigned value of 138.

1. The MC33486 has a guaranteed current recopy ratio accuracy of ±15%, or ±10% based on I_{OUT} value range (I_{OUT} in the range 4 A to 8 A or 8 A to 20 A).

Software Introduction

This section describes the major steps taken during the software part of the development. All software is written in C language, using Metrowerks CodeWarrior for HC08, version 3.0.

The presented software implements the left front window lift/sunroof LIN 2.0 slave unit; however, it is a straightforward operation to modify it for another configuration.

MCU Peripherals Used

This section describes briefly, in the form of tables, all MCU resources used in the project, such as peripheral components and interrupts.

The GPIO (general purpose inputs and outputs) and the analog input pins used are listed in Table 1. The table is divided into LINkit[1] and application related areas.

Table 1. GPIO and Analog inputs

Pin	Direction	Symbolic Name	Purpose
Application related			
PTA5	input	St	MC33486 St pin — Status
PTA6	output	Wake	MC33486 Wake pin — Enable/Disable
PTB4/AD4	analog input	Cur R	MC33486 Cur R pin — Current Recopy
PTB6/AD6	output	IN1	MC33486 IN1 pin — Input #1 (for Open direction)
PTB7/AD7	output	IN2	MC33486 IN1 pin — Input #2 (for Close direction)
LINkit related			
PTB0–3/AD0–3	output	LEDs	LINkit LED #1, #2, #3, #4
PTB5/AD5	output	EN	MC33399 EN pin - Enable
PTE0/TXD	output	TX	MC33399 TX pin - Transmit Data
PTE1/RXD	input	RX	MC33399 RX pin - Receive Data

Brief descriptions of the timer (TIM) and timebase (TBM) modules used in the project are given in Table 2.

1. Pins used for MCU debugging and code flashing are not included.

Table 2. Timers and Their Interrupt Service Routines (ISR)

Timer	ISR Function	Configuration and Description
TIMB	*Mc_TimBOvfISR()*	TIMB set to overflow, used for PWM signal ramping-up of IN1/IN2 inputs of MC33486
TBCH0 (TIMB Channel 0)	no ISR	Timer channel set to unbuffered PWM signal generation for input IN1 of MC33486 (Open direction)
TBCH1 (TIMB Channel 1)	no ISR	Timer channel set to unbuffered PWM signal generation for input IN2 of MC33486 (Close direction)
TIMA	no ISR	TIMA initialized by the LTP (target.c), but not used, as it is required by LIN master only
TBM	*TbmISR()*	TBM used for the 4s period measurement, which is necessary for the LIN “go-to-sleep” transition

All interrupts used in the window lift/sunroof LIN 2.0 slave project are detailed briefly in Table 3.

Table 3: Interrupts

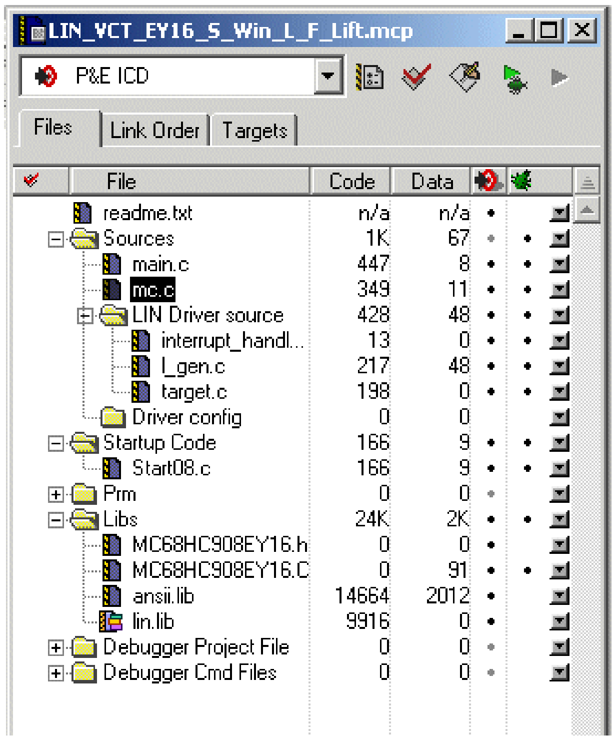
Peripheral Module	ISR Function	Type of Interrupt	Purpose
TIMB	*Mc_TimBOvfISR()*	timer overflow	PWM signal ramping-up of IN1/IN2 inputs of MC33486
TBM (timebase module)	*TbmISR()*	TBM	low speed timer for LIN “go-to-sleep” transition
ESCI	*uart_0_rx_handler()*	SCI Rx	LIN connectivity (VCT LTP function)

Software Development

Software development can be divided into two parts, LIN related and application related:

1. LIN related — the implementation of LIN connectivity. LIN related files are:
   a. *l_gen.c*, *l_gen.h* — these two files are LTP auto generated files,
   b. *target.c*, *target.h* — contain target specific functions for the LTP,
   c. *lin.lib* — a library containing all the LIN LTP related stuff,
   d. several other LTP related *.h files.
2. Application related — brush DC motor control routines, with PWM ramping-up, current recopy feedback measurement. Application related files are:
   a. *main.c*, *main.h* — the main files of the window lift/sunroof LIN 2.0 slave project,
   b. *mc.c*, *mc.h* — files contain all the DC motor control related routines,
   c. *interrupt_handlers.c* — contains the interrupt handler for the SCI, as an interface between the application and the LIN driver.

The window lift/sunroof LIN 2.0 slave Metrowerks CodeWarrior project is depicted in Figure 7.



File	Code	Data		
readme.txt	n/a	n/a	●	
Sources	1K	67	●	●
main.c	447	8	●	●
mc.c	349	11	●	●
LIN Driver source	428	48	●	●
interrupt_handl...	13	0	●	●
l_gen.c	217	48	●	●
target.c	198	0	●	●
Driver config	0	0		
Startup Code	166	9	●	●
Start08.c	166	9	●	●
Prm	0	0	●	
Libs	24K	2K	●	●
MC68HC908EY16.h	0	0	●	
MC68HC908EY16.C	0	91	●	●
ansii.lib	14664	2012	●	
lin.lib	9916	0	●	
Debugger Project File	0	0	●	
Debugger Cmd Files	0	0	●	

Figure 7. Metrowerks CodeWarrior Project Structure

Pulse Width Modulation Related Calculation

To implement the “soft-start” of the window lift motor, pulse width modulation (PWM) is used to drive the MC33486. This device allows a PWM frequency of up to 30 kHz. It is suggested to use a PWM frequency higher than 20 kHz, to limit the acoustic noise.The MC68HC908EY16 has an internal clock generator (ICG) with a programmable frequency output providing integer multiples of the nominal frequency (307.2 kHz ± 25 percent).

To fulfil these conditions, the integer multiple N was selected to be 79, according to the following formulae:

$$f_{BUSnom} = \frac{F_{nom}}{4} = \frac{N \times 307,2kHz}{4} = \frac{79 \times 307,2kHz}{4} = 6067,2kHz$$

As 255 levels of PWM are used,

$$f_{PWMnom} = f_{BUSnom}/255 \cong 23,79kHz$$
$$f_{PWMmin} = 0,75 \times f_{BUSnom}/255 \cong 17,84kHz$$
$$f_{PWMmax} = 1,25 \times f_{BUSnom}/255 \cong 29,74kHz$$

For more information, refer to the ICG module and TIM module chapters of Reference [7].

Motor Control Routines Introduction

As both motor directions are required for the window lift/sunroof LIN 2.0 slave application, both of the high side switches of the MC33486 are used. MC33486 inputs IN1 and IN2 are controlled via TBCH0 and TBCH1 (TIMB channels 0 and 1) of the MC68HC908EY16.

Status of the actual motor activity is described by the following variable:

```
unsigned char mcControlState = MC_NO_OPERATION; /* motor control status byte */
```

where

```
/* states of motor, as stored in mcControlState variable */
#define MC_NO_OPERATION     0       /* no motor control activities */
#define MC_OPENING          0x10    /* OPEN command via IN1 input of MC33486  */
#define MC_CLOSING          0x11    /* CLOSE command via IN2 input of MC33486 */
```

Ramping up of the PWM signal, which implements the motor “soft-start”, is done via the TIMB overflow. The PWM ramp has an initial value, a ramp increment, and a counter describing how many times each PWM level will be used. Values employed in the application are shown below. These are, however, specific to the motor and the window lift used in the system.

```
/****************************************************************************/
/*     TIMER CONTROL RELATED SYMBOLICS FOR PWM GENERATION                   */
/****************************************************************************/
#define RAMP_INCREMENT  1    /* new value of ramp = old value + RAMP_INCREMENT*/
#define RAMP_INIT       70   /* initial value of PWM ramp */
#define COUNTER_LEVEL   50   /* PWM timing counter symbol */
/* for COUNTER_LEVEL = 50, RAMP_INIT = 70 and RAMP_INCREMENT = 1, the duration
   of the ramp up was measured as approx. 375ms long */
```

The current recopy feedback measurement routine is called periodically. The analog value is digitized via pin PTB4/AD4 of the MC68HC908EY16, then averaged (over the CURRENT_HISTORY long current measurement buffer) to filter the value. This averaged value is then used for simple stall/obstruction detection, where the value is compared to the CURRENT_RECOPY_MAX symbolics, representing the current limit at which the motor should be switched off. It is also necessary to report the error signal (*l_bool_wr_win_l_f_err*), and to properly update the window status variable (*windowStatus*).

All these current recopy measurement activities are implemented in the *Mc_control()* routine of the mc.c file, called from the *main()* never ending loop (see Figure 8).

Finally, it is also necessary to check the MC33486 status pin “St”. The normal state of this pin is “1”. When logical “0” is detected, either overtemperature and/or overcurrent occurred on the device. In this case, it is necessary to stop the motor, report the error signal (*l_bool_wr_win_l_f_err*), and update properly the window status variable (*windowStatus*).

Window Command Flow

The window is controlled via the window keyboards. User’s commands are transmitted to the window lift/sunroof LIN 2.0 slave device via the LIN network. This LIN information (called signals and defined in the *Signals* section of the Window_Lift_Net.ldf file) is received by the slave node. Reception is implemented in the *TxRxLIN()* routine of the main.c file, called from the *main()* never ending loop (see

Figure 8). It updates the window status variable according to the requested command, and properly controls the motor.

The variable describing the status of the window is called *windowStatus.*

```
sWindowStatus windowStatus;              /* window/roof status */
```

where *sWindowStatus* type is defined as:

```
typedef struct
{
    unsigned char opening      : 1; /* window/roof is in opening state */
    unsigned char openingCompl : 1; /* window/roof is in completely opening state */
    unsigned char closing      : 1; /* window/roof is in closing state */
    unsigned char closingCompl : 1; /* window/roof is in completely closing state */
    unsigned char openCStopped : 1; /* in completely opening stopped state */
    unsigned char closeCStopped: 1; /* in completely closing stopped state */
/* last two are special states describing that the window was stopped from
   completely closing/opening sequence */

} sWindowStatus;
```

Main Program Structure

The flowchart of the *main()* function is depicted in Figure 8. Most of its functions have already been introduced, apart from the function style macro *PET_WATCHDOG()*, which feeds the computer operating properly (COP) module, and the routine *adjust_clock(),* which trims the ICG module based on the LIN frame bit time measurement[1].

1. The second part of the LIN header consists of a one byte long Synch pattern, which is equal to 0x55. All LIN drivers are supposed to measure this period. Where the LIN slave does not use a precise clock source, but ICG, this information should be used to trim the LIN slave.

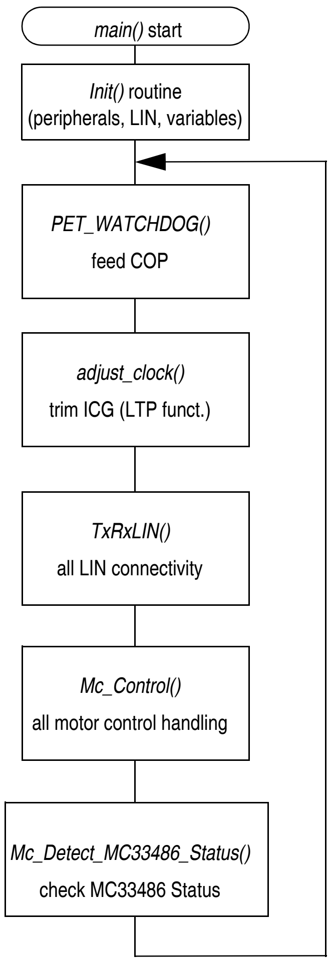


Figure 8. Main Program Flow Chart

LIN Connectivity Development

The first step in adding LIN 2.0 connectivity is to create the cluster message strategy (see Appendix A — Messaging Strategy). It outlines completely the communication between all units in the cluster. It consists of a list of all frames, with defined frame IDs, frame publisher and subscribers, data field content (signal structure), and so on. It is also necessary to create a schedule table for the master. These inputs should be embraced by a LIN description file (*.ldf), whose syntax is given by the LIN specification. The *Window_Lift_Net.ldf* file, as well as the node specific *Window_l_f_lift_slave.prv* and *uart.cfg* (node private files), can be found in Appendix B — LIN Description File and Node Specific Files.[1]

LIN 2.0 connectivity was added to the project by using the Volcano LIN Target Package (LTP) tool[2]. This tool generates the LIN specific C code files out of the cluster LDF file. These files are then directly added to the Metrowerks CodeWarrior compiler/linker, to include the LIN connectivity routines in the project.

1. More about LDF files, their structure and function can be found in Reference [1].
2. LTP release for MC68HC908EY16, version LTP20_1_4_0.

NOTE

To start the LIN slave communication properly, LIN slave clock source deviation, from the nominal master clock before LIN synchronization, must be better than ±14%. (see Reference [4]).

LIN Low-power Mode

The application supports the LIN low-power mode, which, in the case of the LINkit board, consists of a system power-down.

There are two ways whereby low-power mode could be entered: either by the reception of the dedicated LIN “go-to-sleep” command, or by four seconds of bus inactivity[1].

When one of these conditions is detected, the MCU disables the LIN physical layer (MC33399) via its EN pin. This causes the MC33399 to turn off (via the INH pin) the switchable voltage regulator LT1121 (connected via the $\overline{\text{SHDN}}$ pin), to complete the power-down of the system. The only way to wake up the system is via LIN bus wake-up or via the WAKE pin of the MC33399.

Entering the low-power mode is implemented as follows:

```
/* LIN "go-to-sleep" transition handling */
    /* check if "go-to-sleep" LIN command was received or
       period of 4 seconds long LIN bus inactivity expired */
    if (((l_ifc_read_status_i1() & 0xFF) & L_SLEEP_REQUEST) || (!counterGoToSleep))
    {
        SwitchOffMC33486();     /* switch off the MC33486 */
        MC33399_EN = 0;         /* disable MC33399 LIN interface */
    /* This will cause the MC33399 to turn off (via the INH pin)
       the switchable voltage regulator LT1121 to achieve the lower power consumption */
    }
```

1. One of the LIN 2.0 mandatory features for LIN slaves is the automatic “go-to-sleep” transition after 4 s of bus inactivity.

Conclusion

The total VCT LTP 2.0 memory consumption depends on the number of LIN communication frames used. Moreover, memory consumption and, in this second case, also the code efficiency, will depend on the LTP configuration, like the *supports_user_defined_diagnostic*[1] and the *concurrency_safety* settings of the prv file.

MCU memory utilization for this application is shown in Table 4.

Table 4. Particular Code Sizes

MCU Memory Type	MCU Memory Size	LTP 2.0 Software Occupies	Application Software Occupies	Total Occupied Area	Free Space (	Free Space
FLASH	15,872 bytes	2870 bytes	1940 bytes	4810 bytes	11,062 bytes	70%
RAM	512 bytes	52 bytes	15 bytes	67 bytes	445 bytes	87%

In the actual project settings, user diagnostics are not supported, but the pre-emptive protection from another call is enabled (set to LTP)[2].

The window lift/sunroof LIN 2.0 slave application uses the following MCU peripherals:

- One of eight ADC channels for current recopy measurement
- Twelve of 24 I/O MCU pins
- ICG module (no external clock source is needed)
- ESCI module for LIN
- TIMB module (both two channels)
- TBM module.

The remaining MCU resources are free, and can be used for other application purposes (see the MCU block diagram in Figure 3).

The main aim of this application note is to demonstrate:

1. An implementation of LIN 2.0 connectivity
2. Brush DC motor control routines for a window lift/sunroof LIN 2.0 slave based on the MC68HC908EY16.

This should help car manufacturers and all interested parties to adapt their designs to in-vehicle networking systems like LIN 2.0.

1. If the LIN 2.0 User Defined Diagnostics features are required, the *supports_user_defined_diagnostic* key word should be declared. If the LIN Configuration Tool finds the *supports_user_defined_diagnostic* key word, it generates the necessary signal read and write calls to allow direct access to the diagnostic frames using the signal API.
2. By adding the line "*concurrency_safety* = LTP" in the private file, LCFG will automatically add calls to disable and re-enable interrupts around each *l_type_wr_sss* and *l_type_rd_sss* call, where it is needed. For more, see Reference [1] and Reference [5].

Using the Volcano Automotive LIN Target Package 2.0 (VCT LTP 2.0) to implement LIN 2.0 connectivity brings both advantages and disadvantages. Developing with the VCT LTP may seem to be initially more complicated, and it does also require an initial investment in the tool. However, it can lead to a significant reduction in total cycle time, as the whole LIN connectivity part can be created and added to an application within a few hours. It also yields more elegant and readable code, as LTP uses LIN API[1].

The downloadable part of this application note is the complete left front window lift/sunroof LIN 2.0 slave software AN2960SW.

1. Specification of the LIN API is an integral part of the LIN specification package. Moreover, starting from LIN 2.0, it is mandatory to use LIN API in all code written in C.

References

1. LIN 2.0 Connectivity on Freescale 8/16-bit MCUs Using Volcano LTP, Revision 1.0, 11/2004, Freescale Semiconductor document number:AN2767
2. LIN 2.0 Door Lock Slave Based on the MC68HC908GR16 MCU and the LIN 2.0 Communication Protocol, Revision 1.0, 11/2004, Freescale Semiconductor document number: AN2884
3. LIN 2.0 Mirror Unit Slave Based on the MC68HC908EY16 MCU and the LIN 2.0 Communication Protocol, Revision 1.0, 11/2004, Freescale Semiconductor document number: AN2885
4. LIN Specification Package, Revision 2.0, 23 September 2003,
   LIN consortium, http://www.lin-subbus.org
5. User's Guide for LTP 2.0, Rev. D, 31 October 2003
   Volcano Communications Technologies
6. LINkits Evaluation Boards, Revision 1.0, 11/2003,
   Freescale Semiconductor document number (AN2573)
7. MC68HC908EY16 Advance Information, Revision 7.0, 05/2004,
   Freescale Semiconductor data sheet
8. MC33399 Automotive LIN Physical Interface, Revision 4.0, 08/2001,
   Freescale Semiconductor data sheet
9. MC33661 LIN Enhanced Physical Interface, Revision 3.0, 10/2004,
   Freescale Semiconductor data sheet
10. MC33486 Dual High Side Switch for H-Bridge Automotive Applications, Revision 3.3, 06/2001
    Freescale Semiconductor data sheet

Acronyms and Abbreviations

ADC	analog to digital converter (module)
COP	computer operating properly (module)
ESCI	enhanced serial communication interface module
ICG	internal clock generator (module)
I/O	input/output ports
LED	light emitting diode
LIN	local interconnect network
LINkit	evaluation board for LIN development
LTP	LIN Target Package
MCU	microcontroller unit
TBM	timebase module
VCT	Volcano Automotive Technology

Appendix A — Messaging Strategy

Node Name for Signal Provider	ID [0..5]	LIN ID Field (w/ parity)	Frame Name	Frame Description	Frame Size (Bytes)	Sig #	Signal Name	Signal Description	Signal Length (bits)	Signal Start Bit	Signal Initial Value	Raw Value Range	Normalized Value Range	Resolution (Degrees/bit)	Which nodes are reading this response data?	Comments
WIN_MASTER	0x20		WIN_L_F_CMD msg_id = 0x2000	Left front window lift command	1	0	WIN_L_F_OPEN	Opening of left front window	1	0	0	0 = NO ACTION 1 = OPEN			X	Initial value is equal to 0, thus not action to a window / sunroof lift
						1	WIN_L_F_OPEN_COMPL	Open of left front window completely	1	1	0	0 = NO ACTION 1 = OPEN COMPLETELY				Initial value is equal to 0, thus not action to a window / sunroof lift
						2	WIN_L_F_CLOSE	Closing of left front window	1	2	0	0 = NO ACTION 1 = CLOSE				Initial value is equal to 0, thus not action to a window / sunroof lift
						3	WIN_L_F_CLOSE_COMPL	Close of left front window completely	1	3	0	0 = NO ACTION 1 = CLOSE COMPLETELY				Initial value is equal to 0, thus not action to a window / sunroof lift
WIN_MASTER	0x21		WIN_R_F_CMD msg_id = 0x2001	Right front window lift command	1	0	WIN_R_F_OPEN	Opening of right front window	1	0	0	0 = NO ACTION 1 = OPEN			X	Initial value is equal to 0, thus not action to a window / sunroof lift
						1	WIN_R_F_OPEN_COMPL	Open of right front window completely	1	1	0	0 = NO ACTION 1 = OPEN COMPLETELY				Initial value is equal to 0, thus not action to a window / sunroof lift
						2	WIN_R_F_CLOSE	Closing of right front window	1	2	0	0 = NO ACTION 1 = CLOSE				Initial value is equal to 0, thus not action to a window / sunroof lift
						3	WIN_R_F_CLOSE_COMPL	Close of right front window completely	1	3	0	0 = NO ACTION 1 = CLOSE COMPLETELY				Initial value is equal to 0, thus not action to a window / sunroof lift
WIN_MASTER	0x24		WIN_SR_CMD msg_id = 0x2002	Sunroof lift command	1	0	WIN_SR_OPEN	Opening of sunroof	1	0	0	0 = NO ACTION 1 = OPEN			X	Initial value is equal to 0, thus not action to a window / sunroof lift
						1	WIN_SR_OPEN_COMPL	Open of sunroof completely	1	1	0	0 = NO ACTION 1 = OPEN COMPLETELY				Initial value is equal to 0, thus not action to a window / sunroof lift
						2	WIN_SR_CLOSE	Closing of sunroof	1	2	0	0 = NO ACTION 1 = CLOSE				Initial value is equal to 0, thus not action to a window / sunroof lift
						3	WIN_SR_CLOSE_COMPL	Close sunroof completely	1	3	0	0 = NO ACTION 1 = CLOSE COMPLETELY				Initial value is equal to 0, thus not action to a window / sunroof lift
WIN_L_F_LIFT	0x25		WIN_L_F_STATUS msg_id = 0x2003	Status of the left front window lift	2	0	WIN_L_F_ERR	Error occurred on left front window lift	1	0	0	0 = NO ERROR 1 = ERROR OCCURRED			X	Set to 1 if error occurred from last window lift move.
						1	WIN_L_F_RESP_ERR	Communication error occurred with the left front window lift	1	1	0	0 = NO ERROR 1 = ERROR OCCURRED			X	Set to 1 if communication error occurred from last frame reception.
						2	WIN_L_F_CURRENT_RECP	Current recopy info of the left front window lift	8	8	0	analog value between 0 to 255			X	When this value equals to 0 - no current throught the motor; the higher value, the higher current
WIN_R_F_LIFT	0x26		WIN_R_F_STATUS msg_id = 0x2004	Status of the right front window lift	1	0	WIN_R_F_ERR	Error occurred on right front window lift	1	0	0	0 = NO ERROR 1 = ERROR OCCURRED			X	Set to 1 if error occurred from last window lift move.
						1	WIN_R_F_RESP_ERR	Communication error occurred with the right front window lift	1	1	0	0 = NO ERROR 1 = ERROR OCCURRED			X	Set to 1 if communication error occurred from last frame reception.
WIN_SR_LIFT	0x29		WIN_SR_STATUS msg_id = 0x2005	Status of the sunroof lift	1	0	WIN_SR_ERR	Error occurred on sunroof lift	1	0	0	0 = NO ERROR 1 = ERROR OCCURRED			X	Set to 1 if error occurred from last window lift move.
						1	WIN_SR_RESP_ERR	Communication error occurred with the sun roof lift	1	1	0	0 = NO ERROR 1 = ERROR OCCURRED			X	Set to 1 if communication error occurred from last frame reception.
WIN_L_F_KEYB	0x2A		n/a													
WIN_R_F_KEYB	0x2B		n/a													
WIN_SR_KEYB	0x2E		n/a													
WIN_MASTER	0x3C	0x3C	MasterReq	LIN Master Request Command	1	0	SYS_SLEEP	System Sleep Mode	8	0	0x00	0x00-0x7F	-	-	X X X X X X X X X X X	Data 0x00-0x7F reserved by LIN spec. 0x00= Sleep
-	0x3D	0x7D	SlaveResp	LIN Slave Response Command	0	-	-	-	-	-	-	-	-	-	X	
-	0x3E	0xFE	<reserved>	Reserved for user-defined extended frame	0	-	-	-	-	-	-	-	-	-		
-	0x3F	0xBF	<LIN reserved>	Reserved for future extended LIN version	0	-	-	-	-	-	-	-	-	-		

Appendix B — LIN Description File and Node Specific Files

Window_Lift_Net.ldf

```
/*******************************************************************************
*
* Freescale Semiconductor.
* (c) Copyright 2004 Freescale Semiconductor.
* ALL RIGHTS RESERVED.
*
********************************************************************************
*
* Description: This is the LIN description file of the Window Lift / Sun Roof
*     project, which demonstrates the LIN 2.0 connectivity with Freeslace MCUs.
*
*     Master of this project is based on Freescale 16-bit MCU HCS12C32.
*     Slave is based on Freescale 8-bit MCU HC908EY16.
*
*     The LIN description file describes a complete LIN cluster and also
*     contains all information necessary to monitor the network.
*
* Written by Zdenek Kaspar (R55014)
*
*******************************************************************************/
LIN_description_file;
LIN_protocol_version = "2.0";
LIN_language_version = "2.0";
LIN_speed = 10.4 kbps;

Nodes
{
    Master: win_master, 20ms, 1ms;
    Slaves: win_l_f_lift, win_r_f_lift, win_sr_lift;
}

Signals
{
    win_l_f_open          : 1, 0x00, win_master, win_l_f_lift;
    win_l_f_open_compl    : 1, 0x00, win_master, win_l_f_lift;
    win_l_f_close         : 1, 0x00, win_master, win_l_f_lift;
    win_l_f_close_compl   : 1, 0x00, win_master, win_l_f_lift;
    win_r_f_open          : 1, 0x00, win_master, win_r_f_lift;
    win_r_f_open_compl    : 1, 0x00, win_master, win_r_f_lift;
    win_r_f_close         : 1, 0x00, win_master, win_r_f_lift;
    win_r_f_close_compl   : 1, 0x00, win_master, win_r_f_lift;
    win_sr_open           : 1, 0x00, win_master, win_sr_lift;
    win_sr_open_compl     : 1, 0x00, win_master, win_sr_lift;
    win_sr_close          : 1, 0x00, win_master, win_sr_lift;
    win_sr_close_compl    : 1, 0x00, win_master, win_sr_lift;
    win_l_f_err           : 1, 0x00, win_l_f_lift, win_master;
    win_r_f_err           : 1, 0x00, win_r_f_lift, win_master;
    win_sr_err            : 1, 0x00, win_sr_lift, win_master;
    win_l_f_resp_err      : 1, 0x00, win_l_f_lift, win_master;
    win_r_f_resp_err      : 1, 0x00, win_r_f_lift, win_master;
    win_sr_resp_err       : 1, 0x00, win_sr_lift, win_master;
    win_l_f_current_recp  : 8, 0x00, win_l_f_lift, win_master;
}

Diagnostic_signals
{
    MasterReqB0: 8, 0;
    MasterReqB1: 8, 0;
    MasterReqB2: 8, 0;
    MasterReqB3: 8, 0;
    MasterReqB4: 8, 0;
    MasterReqB5: 8, 0;
    MasterReqB6: 8, 0;
    MasterReqB7: 8, 0;
    SlaveRespB0: 8, 0;
    SlaveRespB1: 8, 0;
    SlaveRespB2: 8, 0;
    SlaveRespB3: 8, 0;
    SlaveRespB4: 8, 0;
    SlaveRespB5: 8, 0;
    SlaveRespB6: 8, 0;
    SlaveRespB7: 8, 0;
}

Frames {
    win_l_f_cmd : 0x20, win_master, 1
    {
      win_l_f_open,         0;
      win_l_f_open_compl,   1;
      win_l_f_close,        2;
      win_l_f_close_compl,  3;
    }
    win_r_f_cmd : 0x21, win_master, 1
    {
      win_r_f_open,         0;
      win_r_f_open_compl,   1;
      win_r_f_close,        2;
      win_r_f_close_compl,  3;
    }
    win_sr_cmd : 0x24, win_master, 1
    {
      win_sr_open,         0;
      win_sr_open_compl,   1;
      win_sr_close,        2;
      win_sr_close_compl,  3;
    }
    win_l_f_status : 0x25, win_l_f_lift, 2
    {
      win_l_f_err ,         0;
      win_l_f_resp_err,     1;
      win_l_f_current_recp, 8;
    }
    win_r_f_status : 0x26, win_r_f_lift, 1
    {
      win_r_f_err ,         0;
      win_r_f_resp_err,     1;
    }
    win_sr_status : 0x29, win_sr_lift, 1
    {
      win_sr_err ,          0;
      win_sr_resp_err,      1;
    }
}

Diagnostic_frames
{
    MasterReq: 60
    {
        MasterReqB0, 0;
        MasterReqB1, 8;
        MasterReqB2, 16;
        MasterReqB3, 24;
        MasterReqB4, 32;
        MasterReqB5, 40;
        MasterReqB6, 48;
        MasterReqB7, 56;
    }
    SlaveResp: 61
    {
        SlaveRespB0, 0;
        SlaveRespB1, 8;
        SlaveRespB2, 16;
        SlaveRespB3, 24;
        SlaveRespB4, 32;
        SlaveRespB5, 40;
        SlaveRespB6, 48;
        SlaveRespB7, 56;
    }
}

Node_attributes
{
    win_l_f_lift
    {
        LIN_protocol = “2.0”;
        configured_NAD = 0x30;             /* win_l_f_lift NAD = 0x30 */
        product_id = 0x000B, 0x0020, 1;    /* win_l_f_lift function id = 0x20 */
        response_error = win_l_f_resp_err;
        P2_min = 20 ms;
        ST_min = 20 ms;
        configurable_frames
        {
            win_l_f_cmd    = 0x2000;
            win_l_f_status = 0x2003;
        }
    }
    win_r_f_lift
    {
        LIN_protocol = “2.0”;
        configured_NAD = 0x31;             /* win_r_f_lift NAD = 0x31 */
        product_id = 0x000B, 0x0020, 1;    /* win_r_f_lift function id = 0x20 */
        response_error = win_r_f_resp_err;
        P2_min = 20 ms;
        ST_min = 20 ms;
        configurable_frames
        {
            win_r_f_cmd    = 0x2001;
            win_r_f_status = 0x2004;
        }
    }
    win_sr_lift
    {
        LIN_protocol = “2.0”;
        configured_NAD = 0x32;              /* win_sr_lift NAD = 0x32 */
        product_id = 0x000B, 0x0021, 1;     /* win_sr_lift function id = 0x21 */
        response_error = win_sr_resp_err;
        P2_min = 20 ms;
        ST_min = 20 ms;
        configurable_frames
        {
            win_sr_cmd     = 0x2002;
            win_sr_status  = 0x2005;
        }
    }
}

Schedule_tables {
    sch_conflict_resolving
    {
      SlaveResp                                     delay 20 ms;
      AssignFrameId {win_l_f_lift, win_l_f_cmd}     delay 20 ms;
      SlaveResp                                     delay 20 ms;
      AssignFrameId {win_l_f_lift, win_l_f_status}  delay 20 ms;
      SlaveResp                                     delay 20 ms;
      AssignFrameId {win_r_f_lift, win_r_f_cmd}     delay 20 ms;
      SlaveResp                                     delay 20 ms;
      AssignFrameId {win_r_f_lift, win_r_f_status}  delay 20 ms;
      SlaveResp                                     delay 20 ms;
      AssignFrameId {win_sr_lift, win_sr_cmd}       delay 20 ms;
      SlaveResp                                     delay 20 ms;
      AssignFrameId {win_sr_lift, win_sr_status}    delay 20 ms;
      SlaveResp                                     delay 20 ms;
    }
    normal_mode
    {
        win_l_f_cmd     delay 20 ms;
        win_r_f_cmd     delay 20 ms;
        win_sr_cmd      delay 20 ms;
        win_l_f_status  delay 20 ms;
        win_r_f_status  delay 20 ms;
        win_sr_status   delay 20 ms;
        MasterReq       delay 20 ms;
        SlaveResp       delay 20 ms;
    }
    low_power_mode
    {
    /* Note that the delay period is calculated as n * <time_base>, where
            n           - is an integer number from interval 1 to 255 (only!!!)
            <time_base> - as defined in section Nodes                         */

        MasterReq       delay 20 ms;
        SlaveResp       delay 5000 ms;
    }
}
```

Window_l_f_lift_slave.prv

```
/********************************************************************************
*
* Freescale Semiconductor.
* (c) Copyright 2004 Freescale Semiconductor.
* ALL RIGHTS RESERVED.
*
*********************************************************************************
*
* Description: This is the slave.prv file for the Window Lift LIN slave based on
*     Freescale 8 bit MCU HC908EY16.
*
*     This file is created for “win_l_f_lift” slave node (NAD = 0x30)
*
*     The private file describes information about a single node. It introduces
*     the name for a node, defines which layered modules that are supported and
*     the flags assiciated with signals on the network. Finally, a private file
*     defines the interface characteristics and the network to which each
*     interface connects.
*
* Written by Zdenek Kaspar (R55014)
*
********************************************************************************/
LIN_private_file;
LIN_protocol_version = “2.0”;
LIN_language_version = “2.0”;
concurrency_safety   = LTP;

network “window_lift_net”
{
        node win_l_f_lift;
        file “window_lift_net.Ldf”;
  original_NAD = 0x30;    /* win_l_f_lift NAD = 0x30 */
}

interface “i1”
{
        connects to window_lift_net;
        [include “uart.cfg”]
}
```

uart.cfg

```
/********************************************************************************
*
* Freescale Semiconductor.
* (c) Copyright 2004 Freescale Semiconductor.
* ALL RIGHTS RESERVED.
*
*********************************************************************************
*
```

```
* Description: This is the uart.cfg file for the Window Lift LIN slave based on
*     Freescale 8 bit MCU HC908EY16.
*
*     This file defines the interface characteristics.
*
* Written by Zdenek Kaspar (R55014)
*
********************************************************************************/
/* Configuration for HC908EY16 */

uart "hc08esci_II"
{
                                                   /* Input clock of the ESCI module. */
    clock_frequency_kHz  = 24269; /* By default ESCI runs from CGMXCLK. */
                    /* internal clock generator set up for CGMXCLK ~ 24,268800MHz nominal */

        esci_II_register_base  = 0x10;
        port_ddr_addr         = 0x0a;
        port_io_addr          = 0x08;
        rx_pin_offset         = 1;
        tx_pin_offset         = 0;
}
```

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