Freescale Semiconductor

Order this document
by AN2103/D

AN2103

Local Interconnect Network (LIN) Demonstration

by Alan Devine
Systems Engineering
East Kilbride, Scotland.

1 Introduction

This application note describes a LIN demo that was designed for the SAE show in March 2000. The project was intended to demonstrate the LIN protocol, tools and Freescale products that were available. Although the demo is purely visual and does not represent any particular application, it does introduce many features that would be implemented in actual applications, such as CAN-LIN gateway, sleep mode, messaging scheme, LIN drivers and LIN tools. The hardware was designed to be flexible and can easily be configured to drive many real applications.

An introduction to the LIN protocol and general description of the demo is presented first, followed by a detailed description of the hardware and software, including schematics and flow diagrams. All code listings are included in the Appendix.

2 Local Interconnect Network Bus (LIN)

The LIN bus is an inexpensive serial communications protocol, which effectively supports remote application within a car's network. It is particularly intended for mechatronic nodes in distributed automotive applications, but is equally suited to industrial applications. It is intended to complement the existing CAN network leading to hierarchical networks within cars. The protocol's main features are listed below:

- Single master, multiple slave (i.e. no bus arbitration)

- Single wire communications up to 20Kbit/s
- Guaranteed latency times
- Variable length of data frame (2, 4 and 8 byte)
- Configuration flexibility
- Multi-cast reception with time synchronization, without crystals or ceramic resonators.
- Data checksum and error detection
- Detection of defect nodes
- Low cost silicon implementation based on standard UART/SCI hardware
- Enabler for hierarchical networks

Data is transferred across the bus in fixed form messages of selectable lengths. The master task transmits a header that consists of a break signal followed by synchronization and identifier fields. The slaves respond with a data frame that consists of between 2, 4 and 8 data bytes plus 3 bytes of control information. Figure 1 shows the communication concept of message transfer and the message format.

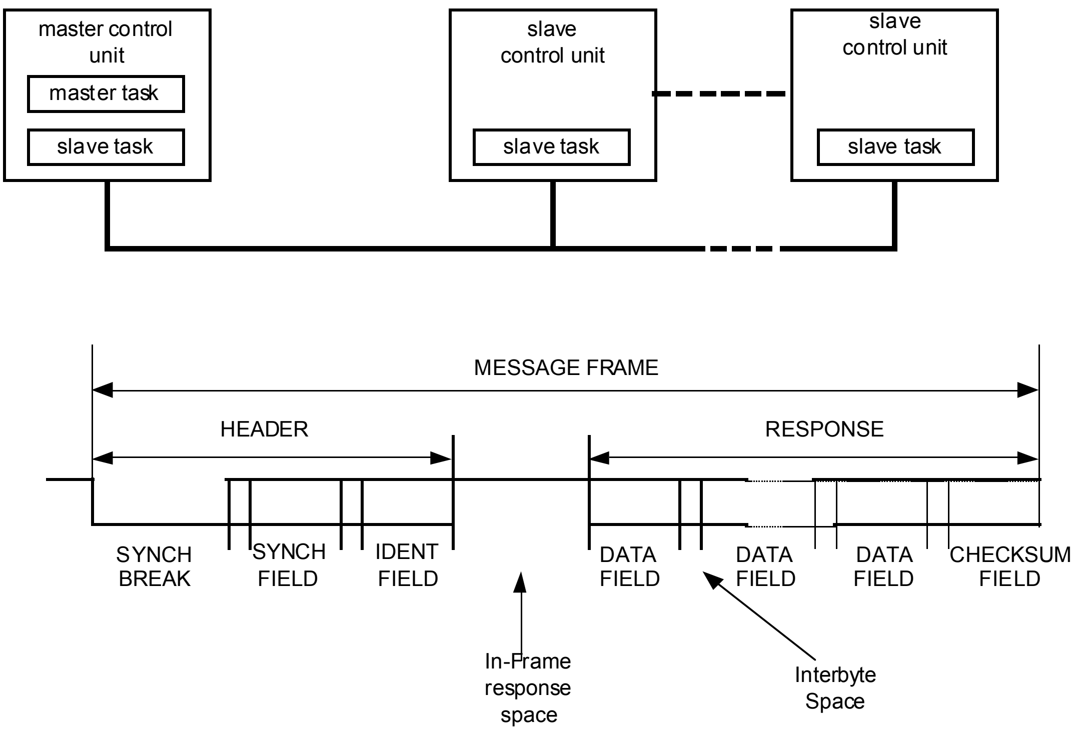


Figure 1 LIN Message frame

The master controls all bus traffic on the network. The master initiates communication by transmitting a header frame with synchronization and identifier information. Any slave, including the slave task in the master control unit, can respond with a data frame. Only one slave can respond to each identifier. However, any number of slaves can be configured to recognize a particular identifier driven on the bus. The master control unit can transfer data to any number of slaves through its slave task. i.e. the master's slave task responds to a header (sent by the master) and transmits data on to the bus. All other slaves can simultaneously receive the data frame.

2.1 Header Frame

The header frame consists of 3 main parts: a SYNCH BREAK signal, a SYNCH FIELD and an IDENTIFIER. The SYNCH BREAK is used to identify the beginning of a message frame and allow the slaves to synchronise to the master's bus clock. It is a unique signal that has 2 parts: a Dominant SYNCH BREAK that is longer than any regular dominant bit stream, and a synchronisation delimiter that is required to enable the detection of the start bit of the following SYNCH FIELD. Figure 2 shows the SYNCH BREAK field.

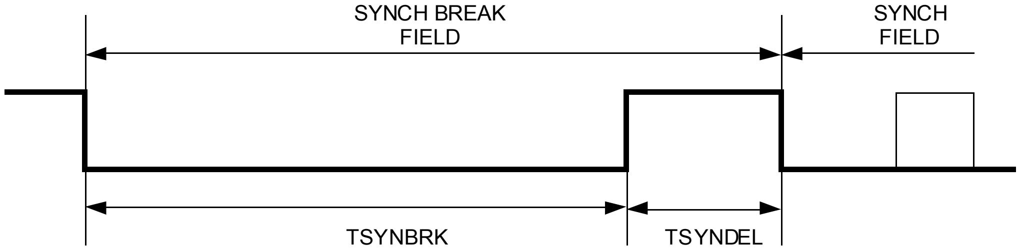


Figure 2 SYNCH BREAK field

The timing specification for the SYNCH_BREAK field is dependent on the tolerance of the slave node's clock source. The master is always required to transmit a dominant T_{SYNBRK} signal, that is a minimum 13 bits, measured in the master's time base. The slave detects a break signal if it is dominant for longer than any regular bit stream. If the slave's clock source has a tolerance lower than +- 15% ($F_{TOL\_UNSYNCH}$) the SYNCH BREAK THRESHOLD is 11 bit times (number of dominant bits required to be recognised as a SYNCH BREAK FIELD) measured in the slave's time base. If the clock source's tolerance is less than +- 2% (F_{TOL_SYNCH}) the threshold is reduced to 9 dominant bits.

The second part of the header is the SYNCH FIELD that contains the pattern 0x55 to allow the slave to synchronize with the master. This allows low cost microcontrollers with internal RC oscillators to be used in slave nodes.

NOTE: *Internal oscillators have a low tolerance that requires to be trimmed (actively changed) if reliable communication is to be maintained.*

The final part of the header is the IDENTIFIER FIELD that denotes the content and length of a message. The content is represented by 6 identifier bits and 2 parity bits. Identifier bits ID4 and ID5 specify the number of data fields in a message. Figure 3 shows the identifier field.

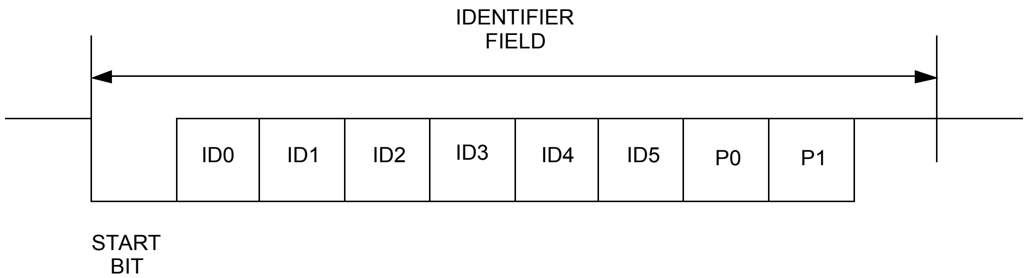


Figure 3 Identifier Field

The parity information is calculated using a mixed parity algorithm that prevents all bits being recessive or dominant.

2.2 Response Frame

The response frame is always transmitted by the slave task (this can be the slave in the master). It consists between 2, 4 or 8 data fields and a checksum field. The data fields consist of 8 bits of data transmitted LSB first. The checksum contains an inverted modulo 256 sum over all data bytes.

2.3 Sleep Mode Frame

A reserved Sleep Mode Frame with a fixed 0x80 identifier was specified in the original version of the LIN specification. In version 1.2 or later the Sleep Mode Frame has been removed and replaced by reserved identifiers, which are described in section 2.4. However, as the demo software was developed to the original specification it uses the fixed identifier 0x80 as the Sleep Mode Frame.

NOTE: *Any slave node can bring the bus out of sleep mode (by transmitting the WAKE_UP signal). However, it is only the master that is allowed to put the network to sleep.*

Refer to LIN specifications for further details.

2.4 Reserved Identifiers

Version 2.1 of the LIN specification contains reserved identifiers; Command frame identifiers and Extended frame identifiers.

2.4.1 Command Frame Identifiers

Two COMMAND FRAME IDENTIFIERS have been reserved to broadcast general command requests from master to all bus participants. The frame structure is identical to regular 8-byte message frames with the following reserved IDENTIFIER FIELDS:

0x3C – Download frame

0x7D – Upload frame

The download frame is used to send commands and data from master to the slave nodes. The upload frame is used to trigger one of the slave nodes (being addressed by a prior download frame) to send data to the master.

Additionally, command frames with their first byte containing 0x00 to 0x7F are reserved for specific use by the LIN consortium. The remaining frames are free to be assigned by the user.

2.4.2 Sleep Mode Command

The SLEEP MODE COMMAND is used to broadcast the sleep mode to all bus nodes. The SLEEP MODE COMMAND is a download COMMAND FRAME with the first data byte set to 0x00

2.4.3 Extended Frame Identifier

Two EXTENDED FRAME IDENTIFIERS have been reserved to allow the embedding of user defined message formats and for future expansion without violating the specification. The frame structure is identical to regular 8-byte message frames with the following reserved IDENTIFIER FIELDS:

0xFE – User defined extended frame

0xBF – Future LIN extension

The identifier can be followed by an arbitrary number of LIN BYTE FIELDS. The frame length, communication concept and data content are not specified. Also, the length coding within the ID field does not apply to the EXTENDED frame identifiers.

3 SAE Demo Description

The LIN Demo consists of a single master node and twelve slave nodes mounted on a ‘clock face’ (see Figure 4 for details). The master controls all slave nodes. It schedules messages that flash the slave’s LEDs in a predetermined sequence. In addition, on a request from the master, each slave node responds with a status messages. The status returned is the value of 2 HEX switches mounted on the slave hardware. The value can be changed, in real time, and monitored on the LIN and CAN buses.

The demo has several modes of operation that are described below. Each mode is selected by a CAN message or by a switch on the master node, when operated in standalone mode. Finally, the master node can be removed and the demo can be driven using a VCT LINspector configured in emulation mode.

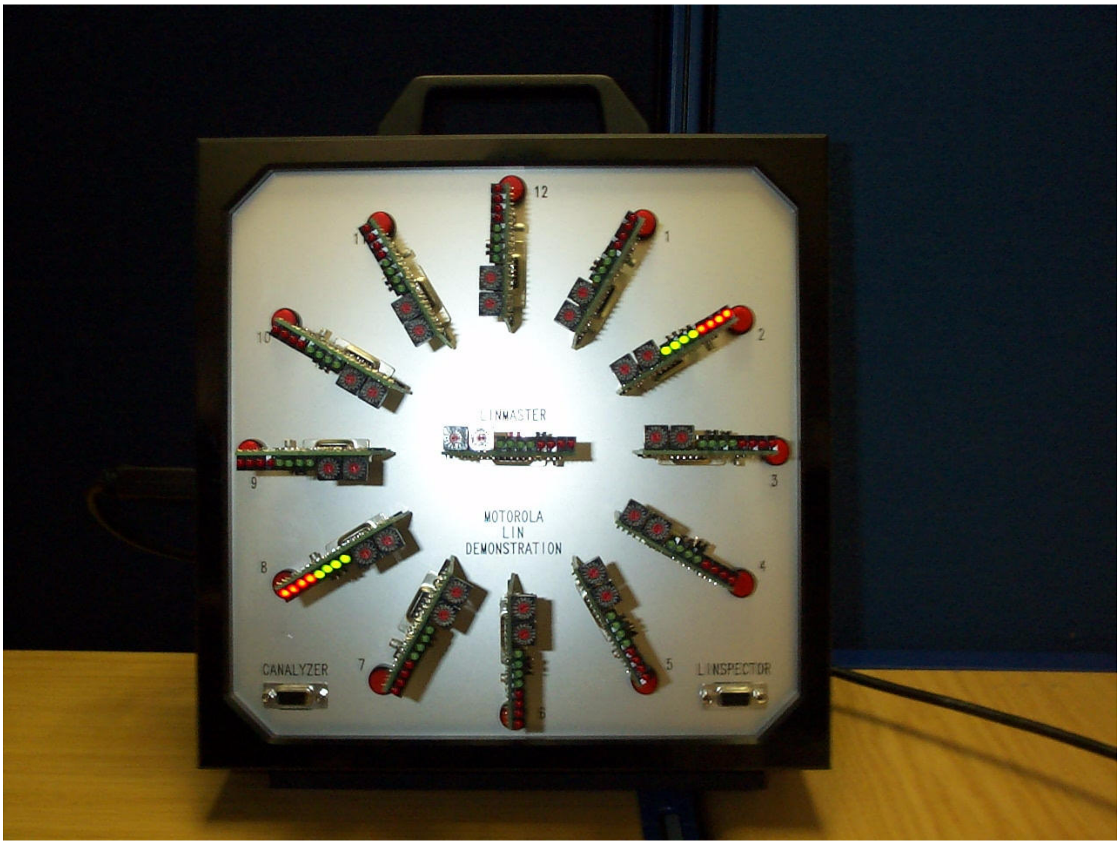


Figure 4 Clock Face Hardware

NOTE: *The LINspector is a cost effective LIN tool that can be used in a variety of situations, including development, testing and verification. It is driven from a LIN configuration description file, which contains all details of the network. It can be used to monitor all traffic, provide detailed timing information and advanced triggering functions. Additionally, it enables basic and advanced emulation features that allow the user to 'replace' any number of nodes on the network. Refer to VCT's web page for more details. The URL is http://www.vct.se*

3.1 Demo Configuration:

3.1.1 Standard:

Software on master node used to control the demo.

VCT LINspector used to monitor all bus activity.

CAN node used to activate different demo modes and display status messages.

3.1.2 Standalone:

Software on master node used to control the demo.

VCT LINspector used to monitor bus activity.

HEX switches on master used to select modes

3.1.3 Emulator:

VCT LINspector used to control the demo and display all bus activity.

3.2 Modes of operation:

3.2.1 Default Mode:

In this mode, the master sequentially transmits messages to the slave nodes that control their LEDs. The slaves respond with the settings of their HEX switches. The switch settings are translated to a CAN message and transmitted onto the CAN bus. If a slave node is removed a NO_NODE code (0x00) is transmitted on to the CAN bus

3.2.2 Broadcast Mode:

In this mode, the master periodically transmits a messages that each slave node simultaneously receives. The transmitted messages switch on the slave's LEDs. The master node controls the LED pattern.

3.2.3 Ident Mode:

This mode is primarily used to set-up the demo. The master transmits a broadcast message that each slave node receives. On reception of this message, the slaves output their IDs to the LEDs.

3.2.4 Sleep Mode:

This is the demo's low power mode. The master transmits a sleep command that signals to the slaves to enter sleep mode by disabling their voltage regulators. The master also enters sleep mode once the sleep command has been successfully transmitted. Any slave can wake-up the demo by pressing the red buttons around the perimeter of the 'clock face'. The slave node that is woken up wakes up the entire network by transmitting a wake-up sequence on the LIN bus.

The LIN physical interface (MC33399) supports wake-up from the bus and from an external source. On the detection of a valid wake-up signal, the physical interface drives its inhibit output signal low, enabling an external voltage regulator (if this feature is used). Alternatively, the inhibit output can be used to drive the IRQ of the microcontroller. Refer to MC33399 data sheet for specific application details.

3.3 LIN Messaging Scheme

A simple data driven messaging scheme was used to control the demo. Each slave node is statically configured to recognize 3 LIN message identifiers. These are a NodeX_Write, a NodeX_Read and a Broadcast message, where X denotes the node number. This allows the master to transmit commands and data to individual nodes and for each node to transmit status responses back to the master. The broadcast message identifier is common to each slave node. It allows the master to transmit data to all the nodes simultaneously. Table 1 lists the messages that were used for the demo.

Table 1 LIN Messages

Message Name	Message ID (LIN ID)	Slave Response Source	Slave Response Destination	Description
Node1_Write	LINMsg01 (0xC1)	Master	Slave_ID 1	Master transmits node1 control command
Node2_Write	LINMsg02 (0x42)	Master	Slave_ID 2	Master transmits node2 control command
Node3_Write	LINMsg03 (0x03)	Master	Slave_ID 3	Master transmits node3 control command
Node4_Write	LINMsg04 (0xC4)	Master	Slave_ID 4	Master transmits node4 control command
Node5_Write	LINMsg05 (0x85)	Master	Slave_ID 5	Master transmits node5 control command
Node6_Write	LINMsg06 (0x06)	Master	Slave_ID 6	Master transmits node6 control command

Table 1 LIN Messages

Message Name	Message ID (LIN ID)	Slave Response Source	Slave Response Destination	Description
Node7_Write	LINMsg07 (0x47)	Master	Slave_ID 7	Master transmits node7 control command
Node8_Write	LINMsg08 (0x08)	Master	Slave_ID 8	Master transmits node8 control command
Node9_Write	LINMsg09 (0x49)	Master	Slave_ID 9	Master transmits node9 control command
Node10_Write	LINMsg0A (0xCA)	Master	Slave_ID 10	Master transmits node10 control command
Node11_Write	LINMsg0B (0x8B)	Master	Slave_ID 11	Master transmits node11 control command
Node12_Write	LINMsg0C (0x4C)	Master	Slave_ID 12	Master transmits node12 control command
Node1_Read	LINMsg11 (0x11)	Node1	Master	Slave transmits status data back to master.
Node2_Read	LINMsg12 (0x92)	Node2	Master	Slave transmits status data back to master.
Node3_Read	LINMsg13 (0xD3)	Node3	Master	Slave transmits status data back to master.
Node4_Read	LINMsg14 (0x14)	Node4	Master	Slave transmits status data back to master.
Node5_Read	LINMsg15 (0x55)	Node5	Master	Slave transmits status data back to master.
Node6_Read	LINMsg16 (0xD6)	Node6	Master	Slave transmits status data back to master.
Node7_Read	LINMsg17 (0x97)	Node7	Master	Slave transmits status data back to master.
Node8_Read	LINMsg18 (0xD8)	Node8	Master	Slave transmits status data back to master.
Node9_Read	LINMsg19 (0x99)	Node9	Master	Slave transmits status data back to master.
Node10_Read	LINMsg1A (0x1A)	Node10	Master	Slave transmits status data back to master.
Node11_Read	LINMsg1B (0x5B)	Node11	Master	Slave transmits status data back to master.
Node12_Read	LINMsg1C (0x9C)	Node12	Master	Slave transmits status data back to master.
BroadCast	LINMsg0F (0x80)	Master	All slave nodes	Master transmits command to all slave nodes

The messages selected for the demo are all 2 bytes long. The first byte is a command byte and the second is data. Table 2 details the NodeX_Write Message Format.

NodeX_Write Message Format

Table 2 NodeX_Write Messages

Identifier = See table	Byte1 = Command Byte	Byte2 = LED Pattern

Write Message Command Byte	
Command	**Code**
SLAVE_LEDS_COMMAND	0x01
CLOCK_LEDS_COMMAND	0x03

Broadcast_ Message Format

Table 3 Broadcast Messages

Identifier = See table	Byte1 = Command Byte	Byte2 = LED Pattern

Broadcast Message Command Byte	
Command	**Code**
BROADCAST_COMMAND	0x02
IDENT_COMMAND	0x04
SLEEP_COMMAND	0x80

NOTE: *LED pattern sent with command byte. Pattern written to slave node LEDs. IDENT and SLEEP commands the LED pattern is ignored.*

NodeX_Read Message Format

Table 4 NodeX_Read Messages

Identifier = See table	Byte1 = NodeID	Byte2 = Hex Switch

The message format was adopted to allow flexibility within the demo. Additional commands can be added without too much effort. The slaves can also easily decode the various commands and act accordingly. Another benefit is that the master node software has total control over the LED pattern that the slaves output. In order to change the LED's sequence, only the master software has to be changed.

In actual LIN applications a signal-based messaging scheme should be adopted. Refer to section Section 4.1 Freescale LIN Drivers and API for details of Motorola's signal-based LIN API.

3.4 Hardware Description

The demo consists of 13 LIN nodes; a single master and 12 slaves. The hardware for each node is identical as shown in the schematic (see appendix for schematic details). Identical hardware was used to enable a universal master/slave board to be designed. This makes the slaves more flexible and reduces demo cost. The main drawback of adopting this common solution is that the microcontroller required for the master node, MC68HC908AZ60, is not suitable for slave nodes because of its additional functionality (particularly CAN). This makes it too expensive for a typical slave node. Figure 5 shows a block diagram of the hardware. Table 5 details more suitable slave microcontrollers. Contact Freescale for further details (www.mcu.motsps.com).

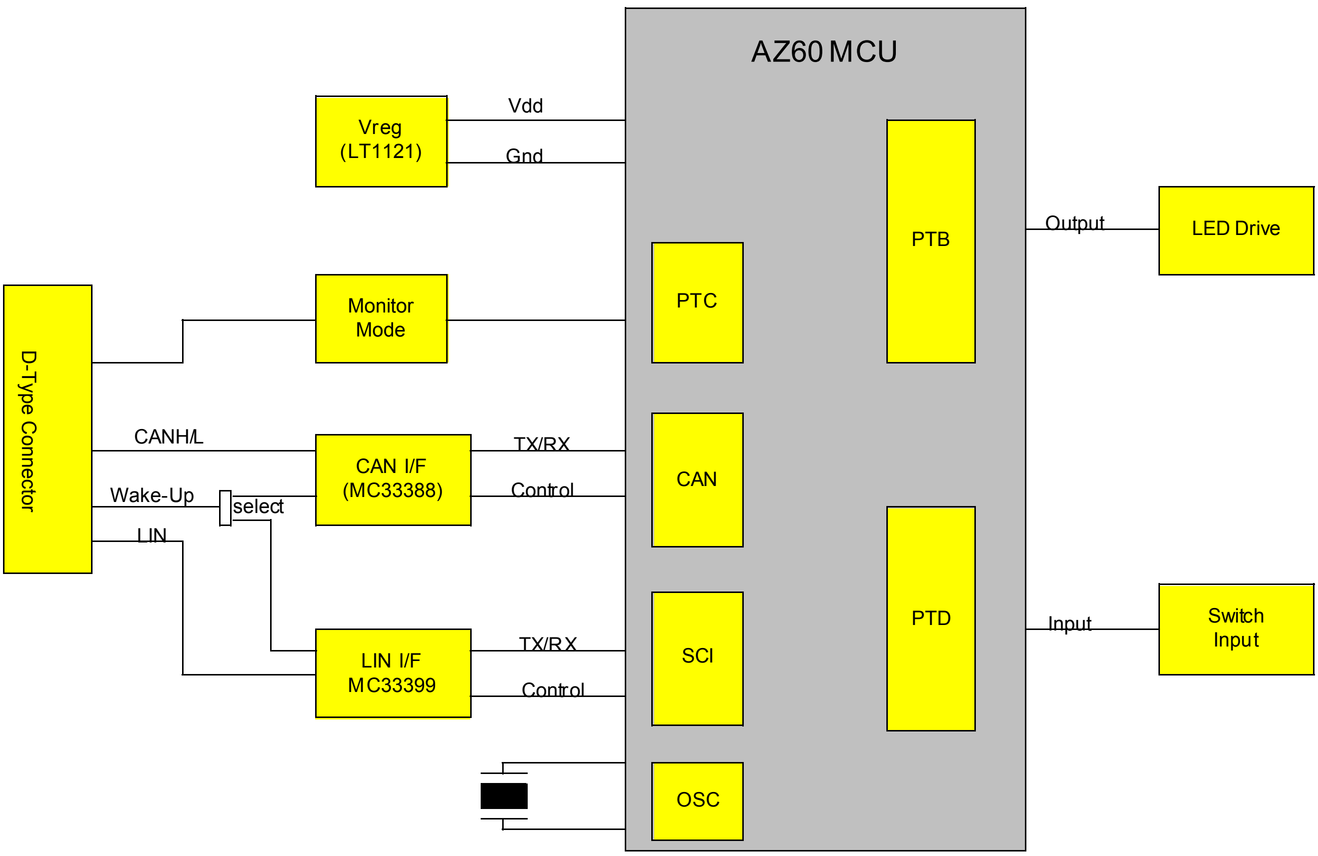


Figure 5 Master/Slave Hardware

The hardware has several functions that make the design flexible and suitable for other general CAN and LIN applications.

Table 5 LIN Slave Devices

LIN Slave MCUs				
Device	**ROM**	**FLASH**	**RAM**	**Features**
68HC908JK3	–	4K	128	Timer, PWM, ATD
68HC908JL3	–	4K	128	Timer, PWM, ATD
68HC908JK1	–	1.5K	128	Timer, PWM, ATD
68HC08AB16	16K	–	512	Timer, PWM, ATD, SCI, SPI
68HC908EY8	–	8K	256	Timers, ATD, SPI, Enhanced SCI

LIN Slave HyperIntegration/Mechatronics					
Device	**ROM**	**FLASH**	**RAM**	**EEPROM**	**Features**
68HC05PV8	8K	–	192	128	Timer, PWM, A/D, OSC, HV I/O, OP-Amp, Phy I/F
68HC805PV8	–	8K	192	128	Timer, PWM, A/D, OSC, HV I/O, OP-Amp, Phy I/F
33393TM			64	1k	Timer, Osc, 2x175mA H-Bridge, Mechatronic package

3.4.1 Monitor Mode

The hardware has the additional circuitry included to allow the microcontroller to communicate with a PC via its monitor mode. This enables in-circuit Flash programming and simple debugging to be performed. Hiware's MON08 target was used in the application development.

3.4.2 LIN Physical Interface (MC33399)

The MC33399 was used in the demo. This interface is a serial link bus interface designed to provide bi-directional, half-duplex communication interfacing in automotive applications. It is similar to the ISO9141 interface, but has additional features specific to the LIN protocol. These features are wake-up from the LIN bus, wakeup from an external source and slew rate control to reduce EMI emissions. Refer to the MC33399 data sheet for a detailed description and application diagrams.

3.4.3 CAN Physical Interface

In addition to the LIN interface, each node has a CAN physical interface (MC33388). This is required for the master node, as a simple CAN to LIN gateway is implemented.

3.4.4 LEDs and HEX Switch interface

Each node has 8 LEDs (4 red and 4 green) and 2 HEX switches. The LEDs are driven directly from Port B, configured as output, and used to demonstrate the LIN protocol and sequencing of messages from the master. The switches are input to Port D and allow each node to have a status and identifier value that can be changed in real time and transmitted on the LIN bus.

The LEDs and switches are positioned at the edge of the boards and can easily be removed and the board used to drive an actual application. For example, the port lines that interface to the LEDs and switches could be connected to a power drive board and could be used to control motors of a mirror module. This makes the board very flexible and allows quick prototyping of LIN applications.

3.4.5 Components used

MC68HC908AZ60 – General purpose flash MCU with CAN.

MC33399 – LIN Physical interface.

MC33388 – Low speed, fault tolerant, CAN physical interface.

4 Software Description

The demo comprises a single master and 12 slave nodes. The master software is responsible for scheduling LIN messages, providing a CAN to LIN gateway and general communications. The slave software interrogates all header frames transmitted on the bus and either receives a response frame from another slave or transmits a response frame on to the bus. Each slave waits for a pre-configured message, decodes the command and either outputs the data to its LED port, if it is a NodeX_Write or broadcast message, or transmits a status message to the LIN network, if a NodeX_Read message was detected. The code for each slave is practically identical, the only difference being the messages configured are specific to individual nodes. See Table 1 for details.

The master and slave code implementations both use the Freescale HC08 LIN low level drivers to manage all the LIN communications. The drivers and the application code for the demo are described in this section. All data flow diagrams and flow charts are included. Refer to the appendix for code listing.

4.1 Freescale LIN Drivers and API

The driver provides the full LIN protocol eliminating the application code from implementing the LIN low level kernel. The user interfaces with the drivers, statically at compile time and dynamically at run time through an API. Two versions of the drivers exist: one with a custom Freescale API and the second with the LIN API. The project used the Freescale API drivers.

The Freescale API is entirely message based. The message identifiers for a specific node are configured at compile time through header files. The application accesses the data transmitted using the LIN_GetMsg() and LIN_PutMsg() services. The application retrieves or transmits the data associated with the identifier. (The address of a buffer that contains the data to be transmitted or received and the message identifier are passed to the driver, by the application). The drivers are easily used with no additional tools and are linked with the application code.

The main difference of the LIN API is that it is signal based. The application code does not access the entire message data, but only specific signals. A signal consists of one or more bits of data. The drivers provide services to access particular signals of varying lengths. (1 bit, 2-8bits and 9-16bits). In order to use the drivers an additional description file is required that describes all the signals that are specific for a particular node. This description file is then converted to header files (an additional tool is required for this) and included with the application. Every node on the network requires a separate description file that contains its specific signals. This file is also used with the LINspector tool for development and evaluation. The LIN API has provision to connect to several hardware interfaces (more than one SCI).

The LIN Drivers are currently available for the HC05, HC08 and HC12 families of microcontrollers. Details of the HC08 implementation are given below.

Node	LINBaud Rate(bps)	MCU bus frequency	MCU load	RAM (bytes)	ROM (bytes)	Stack (bytes)
Master	20000	4	<9%	23	1391	<34
Slave	20000	4	<5%/6%	20/21	1071/689	<34/19

Note1: Figures exclude per message overhead
Note2: Freescale API/LIN API

Contact Freescale Software Systems for further information.
software.systems@www.freescale.com

4.1.1 Static Configuration

The drivers are statically configured through 2 header files, lincfg.h and linmsgid.h. The lincfg.h file is used to provide general LIN configuration information, such as baud rate and timer pre-scalers. The information in this file is the same for each node on the network, assuming they are all the same target hardware. The linmsgid.h file is used to define the node's messages and whether they are to be received or transmitted. This file is usually unique to every node on the network. For further details refer to the LIN drivers manual and the demo configuration files.

4.1.2 Driver API

The API is the interface with the drivers. The application code calls the run time services provided by the driver, during execution. The services used in the demo software are described below. Refer to driver manual for full descriptions of the Freescale and LIN API.

4.1.2.1 LIN_Init:

The LIN_Init service performs initialization of the driver. The function must be called before any other API service call is made. The service initializes the following functions:

- Sets baud rate (Information entered in lincfg.h file)
- Assigns physical interface pins
- Sets Tx to idle state
- Clears all error flags and counters
- Clears all data buffers
- Change state of drivers to run
- Initializes all variables

Syntax:	unsigned char LIN_Init (void);
Applicable:	Master, Slave
Parameters:	None
Return:	LIN_OK

4.1.2.2 LIN_GetMsg:

The LIN_GetMsg service retrieves the current content of the specified message buffer to an application defined buffer

Syntax:	unsigned char LIN_GetMsg (unsigned char MsgId, unsigned char * Data);
Applicable:	Master, Slave
Parameters:	MsgId – Message identifier Data – Pointer to memory buffer where received data is to be stored.
Return:	LIN_OK, LIN_NO_ID, LIN_INVALID_ID and LIN_MSG_NODATA

4.1.2.3 LIN_PutMsg:

The LIN_PutMsg service transmits current contents of specified message buffer to an application specified message buffer.

Syntax:	unsigned char LIN_PutMsg (unsigned char MsgId, unsigned char * Data);
Applicable:	Master, Slave
Parameters:	MsgId – Message identifier Data – Pointer to memory buffer where data is to be transmitted.
Return:	LIN_OK, LIN_NO_ID and LIN_INVALID_ID

4.1.2.4 LIN_RequestMsg:

The LIN_RequestMsg service transmits the message identifiers header frame

Syntax:	unsigned char LIN_RequestMsg (unsigned char MsgId);
Applicable:	Master
Parameters:	MsgId – Message identifier
Return:	LIN_OK, LIN_REQ_PENDING and LIN_MSG_SLEEP

4.1.2.5 LIN_MsgStatus: The LIN_MsgStatus service returns the current status of the specified message buffer.

Syntax:	unsigned char LIN_MsgStatus (unsigned char MsgId);
Applicable:	Master, Slave
Parameters:	MsgId – Message identifier
Return:	LIN_NO_ID, LIN_OK, LIN_MSG_NOCHANGE and LIN_MSG_NODATA

4.2 Master code implementation

The master software has 2 main tasks that schedule LIN messages and provide a CAN to LIN gateway. The scheduler operates from a periodic tick, driven from Timer B overflow, and transmits a header frame every150ms. The gateway function is driven from the CANRx interrupt and either changes the demo mode or transmits a LIN message to a specific slave node. Figure 6 shows the data flow diagram of the master software.

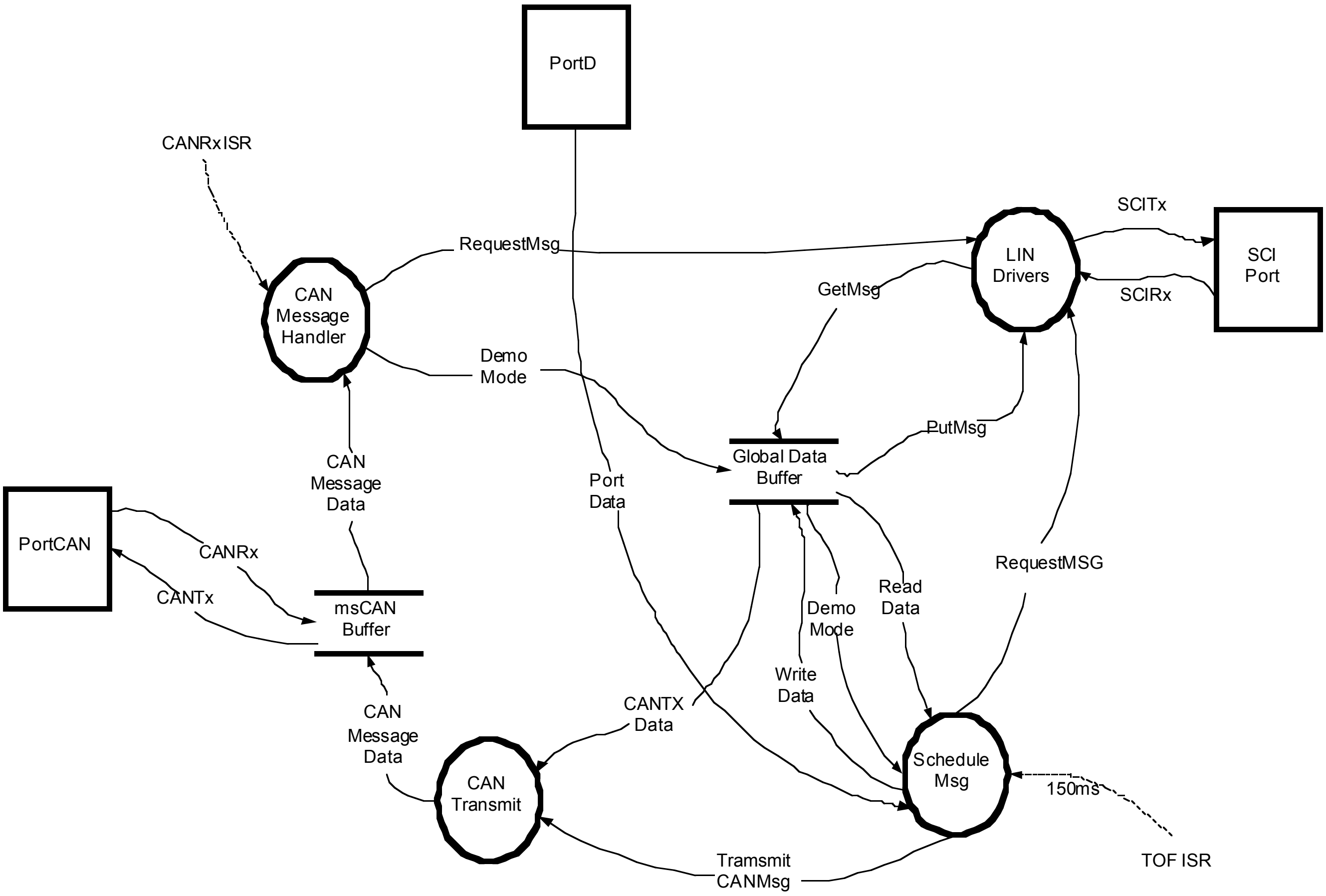


Figure 6 Master Data flow Diagram

4.2.1 Demo Modes

As discussed the demo has several modes of operation that are user selectable. The mode is controlled by a CAN message or by the HEX switch on the master PEC, when operated in standalone mode. The mode can be changed at any time. Table 6 shows the CAN messages and switch positions that select different modes.

Table 6 Mode Selection Table

Mode Selection Table		
Mode	**CAN Message**	**Switch Position**
Default	ID 0x00, Byte0=0x0E, 0x00	0
Broadcast	ID 0x00, Byte0=0x0E, 0x01	1
Ident	ID 0x00, Byte0=0x0E, 0x02	2
Sleep	ID 0x00, Byte0=0x0E, 0x03	3

The mode select is controlled in the ScheduleMessage function. The software reads a demomode control variable and depending on its value calls a default message handler, broadcast message handler, ident message handler or a sleep message handler. The control variable is initialized to DEFAULT mode out of reset, but can be updated directly from the hex switch (on Port D) or from a CAN message.

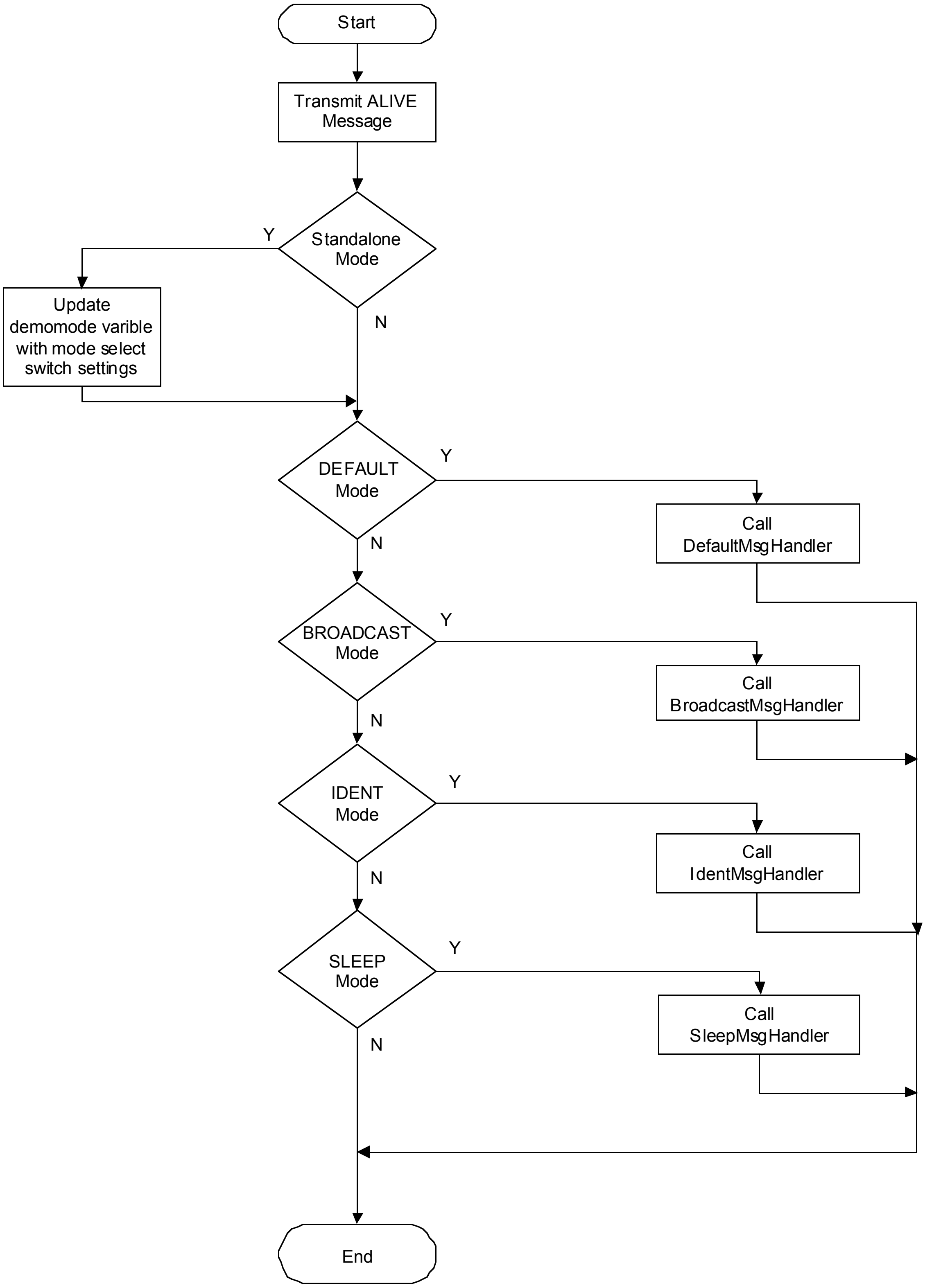


Figure 7 ScheduleMessage Flow Diagram

Each mode and its software are described below:

4.2.1.1 Default mode

In default mode the software executes in a loop that sequentially transmits 2 messages to each slave node. See Table 7 for sequence. The first message is a NodeX_Read message that requests a status response from the slave to indicate that it is present. If the master does not receive the slave response within a 10mS timeout it is assumed that the node is not present and a NO_NODE code (0x00) is transmitted on the CAN bus. The second message transmitted (assuming a node is present) is a NodeX_Write message that transmits a command byte and a data byte to a specific slave. The slave decodes the command and outputs the data to its LED port.

The status response message is checked to see if it has changed since the last interrogation. If it has changed it is translated to a CAN message and transmitted onto the CAN bus. The status information is updated before the function is exited.

NOTE: *The CAN transmission is not performed if the demo is in standalone mode.*

The scheduler software exits this function then waits in the main loop before transmitting to the next node in the sequence. The node number to be transmitted is controlled in the TIMBOVF_ISR. See main loop for further details.

Table 7 Schedule Sequence

Default Message Sequence	
Node Number	**Data**
12	ALL_LEDS_ON
1 11	RED_LEDS_ON GREEN_LEDS_ON
2 10	RED_LEDS_ON GREEN_LEDS_ON
3 9	RED_LEDS_ON GREEN_LEDS_ON
4 8	RED_LEDS_ON GREEN_LEDS_ON
5 7	RED_LEDS_ON GREEN_LEDS_ON
6	ALL_LEDS_ON
7 5	RED_LEDS_ON GREEN_LEDS_ON
8 4	RED_LEDS_ON GREEN_LEDS_ON
9 3	RED_LEDS_ON GREEN_LEDS_ON
10 2	RED_LEDS_ON GREEN_LEDS_ON
11 1	RED_LEDS_ON GREEN_LEDS_ON
REPEAT	REPEAT

The default message table shows the order that the nodes are written to and the data that is transmitted.

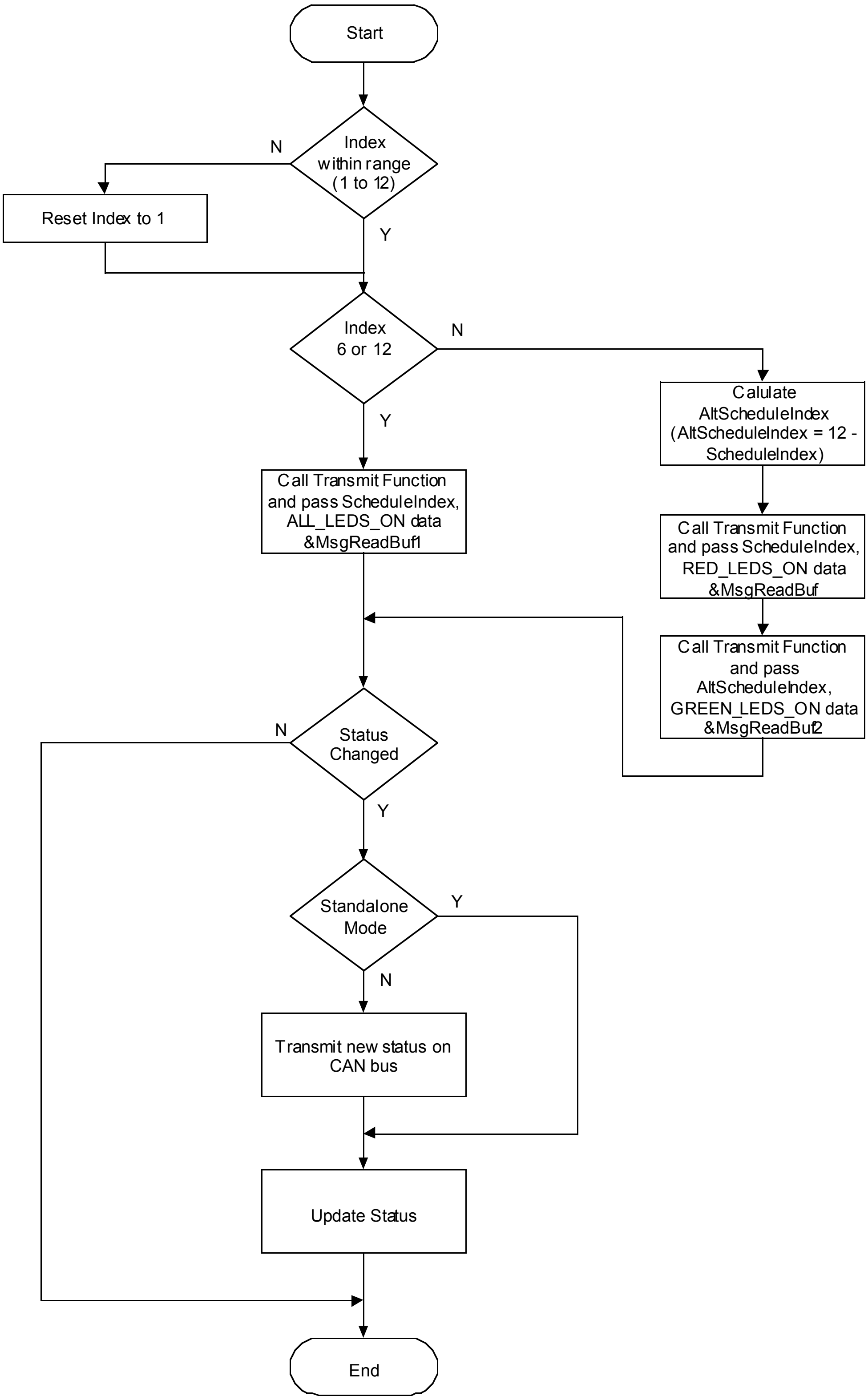


Figure 8 Default Mode Flow Diagram

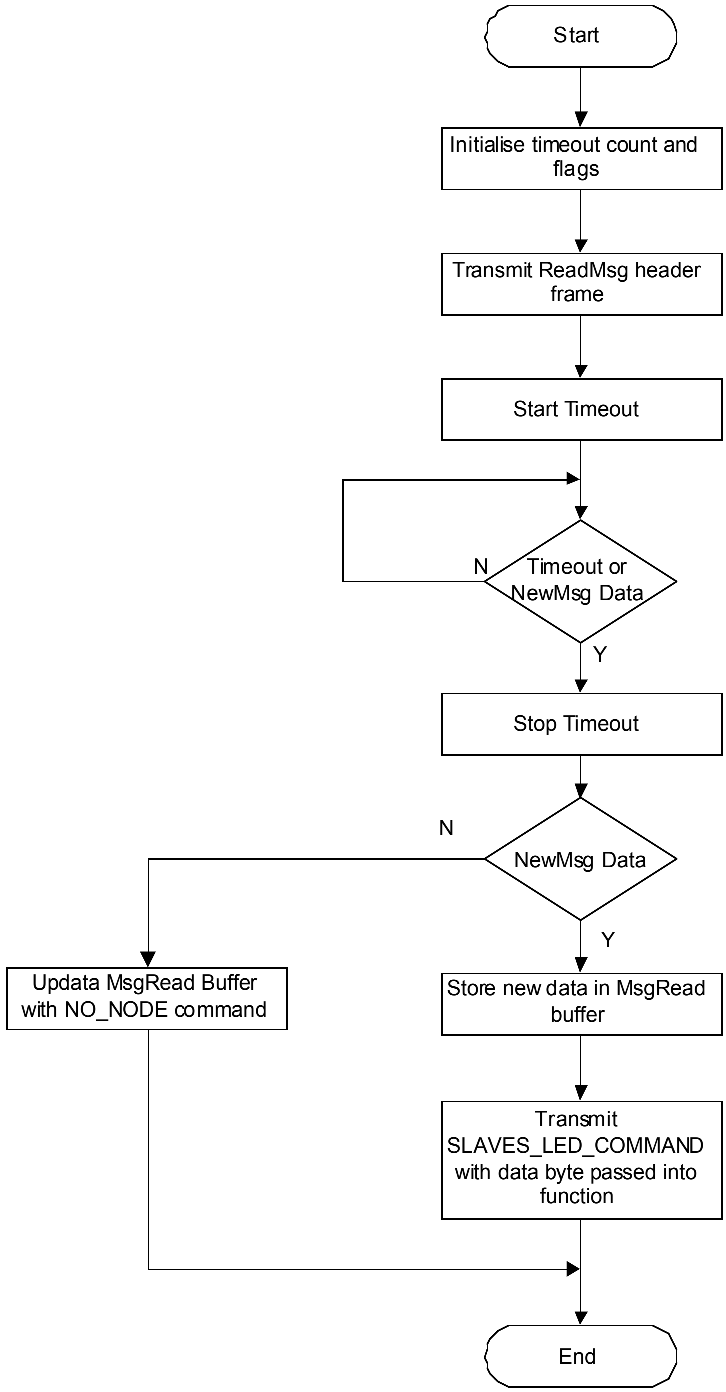


Figure 9 Default Transmit Function

4.2.1.2 Broadcast Mode

In this mode, the master software periodically transmits a broadcast message which is received by every slave node. The first byte of the message contains the broadcast command, and the second byte contains the data byte that each slave outputs to its LED port. A single 1 is shifted through the data byte from bit0 to bit7, the sequence is then reversed and repeated. The data bytes to be broadcast are stored in a lookup table. See Table 8 below for details.

Table 8 Broadcast Table

0x01
0x03
0x07
0x0F
0x1F
0x3F
0x7F
0xFF

NOTE: *A lookup table was used to allow the pattern sequence to be changed with ease*

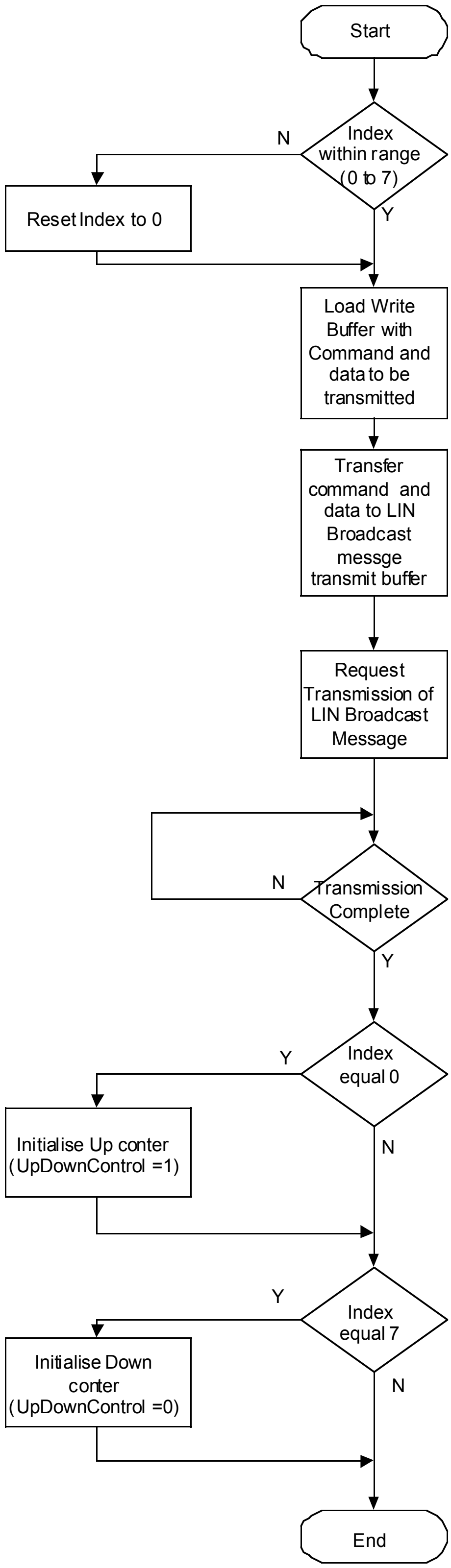


Figure 10 Broadcast Mode Flow Diagram

4.2.1.3 IDENT Mode

In this mode, the master software periodically transmits a broadcast message with an IDENT command. Each slave node receives this command and outputs its individual identifier to its LED port.

NOTE: *The slave is pre-programmed with its identifier.*

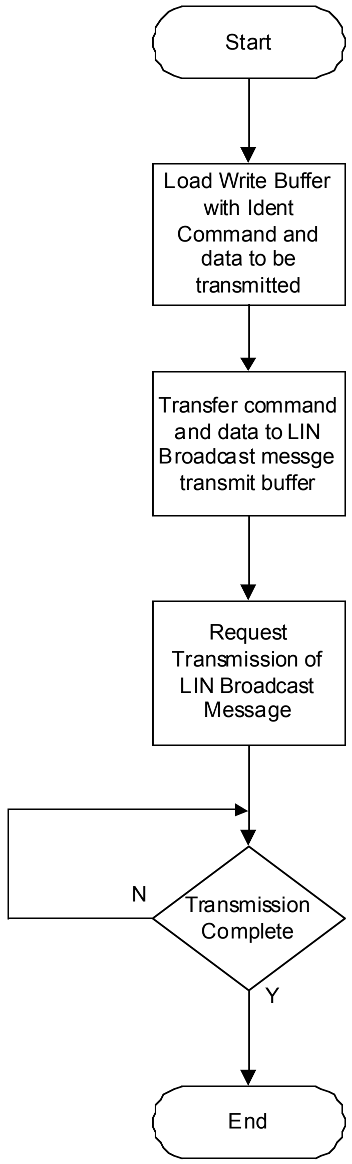


Figure 11 Ident Mode Flow Diagram

4.2.1.4 Sleep Mode

In this mode, the demo enters its low power state by switching off its voltage regulator. The master node transmits a SLEEP command that is received by all slave nodes. Once the SLEEP command has been successfully transmitted, the master disables its voltage regulator by driving the LIN interface into its Sleep mode. The software then waits in an infinite loop until the regulator is disabled. The master is the only node that can issue a SLEEP command.

The master is woken by a wake-up request initiated by one of the slave nodes. The LIN I/F device recognizes a specific wake-up message driven onto the bus. The LIN I/F brings the node out of sleep mode by turning on the voltage regulator.

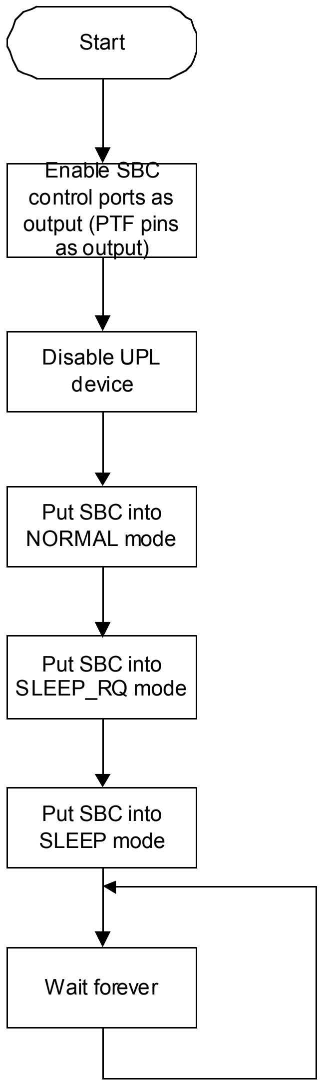


Figure 12 Sleep Mode Flow Diagram

4.2.2 CAN Communication

The master software also provides CAN communication and implements a simple CAN to LIN gateway.

The CAN receive and the transmit functions are described below:

NOTE: *CAN communication do not occur when the demo is configured in standalone mode.*

4.2.2.1 Receiving CAN Messages

The receiver function is interrupt driven. If a message passes the CAN filters it is written into the CAN receive buffer and a receive interrupt request is issued. The CANRx ISR sets a msgrxd flag that indicates that a new message has been received and clears the interrupt flag to enable further interrupts. The main routine, discussed below, polls the msgrxd flag, waiting until it is set. When a new CAN message is received, the code calls a MsgHandlerTable (array of pointers to functions) and jumps to the appropriate MsgHandlerFunction. The function that is executed is dependent on the first byte of the received CAN message. This byte, masked off with 0x0F, determines the index of the MsgHandlerTable and subsequently the handler that the code executes.

4.2.2.2 CAN Message Handlers

The basis of the CAN communication and gateway function is performed using a series of message handlers. These are described below:

4.2.2.2.1 Common Message Handler

This handler is used to transmit a LIN message to a particular slave node. The node that the message is transmitted to is determined by an index (CanMsgIndex). The index is calculated from the CAN message that was received. The slaves receive the LIN messages and control the external button LED around the circumference of the 'clock face'.

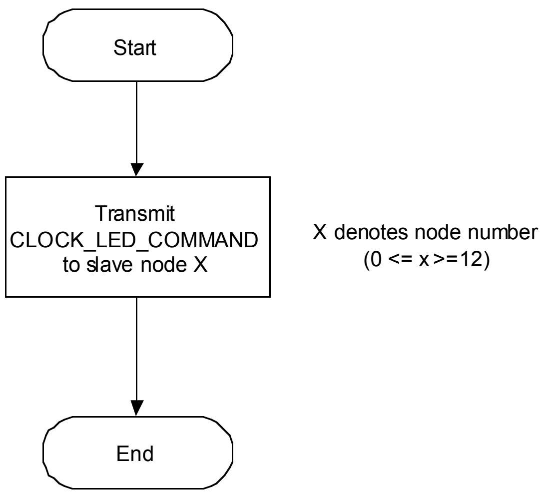


Figure 13 Message Handler (1-12) flow diagrams

4.2.2.2.2 SleepHandler

See sleep mode description

4.2.2.2.3 ModeSelect Handler

This function decodes the CAN mode select message and writes the appropriate value to the demomode variable.

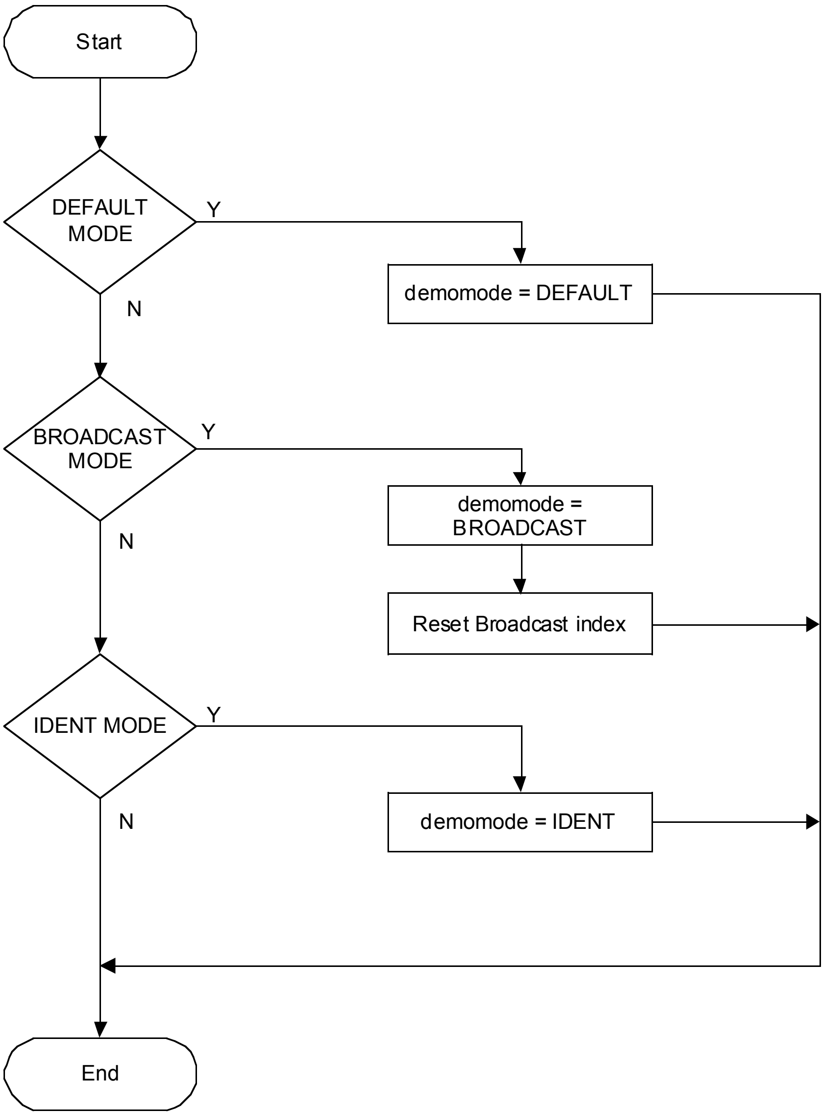


Figure 14 ModeSelectHandler flow diagrams

4.2.2.2.4 DefaultHandler

This handler consists of the function prototype. This is included for expandability and to ensure that the code does not ‘run away’ even if an invalid entry in the message handler vector table is accessed, i.e. all unused entries in the table jump to DefaultMessage handler.

4.2.2.3 Transmitting CAN Messages

The CAN transmitter function is called from the initialisation and default mode functions. The identifier and data to be transmitted are written to TxBuffer0 and transmitted when the bus is idle. The function that transmits the message, TxCANBuffer(), receives a pointer to a structure that contains the id and data to be transmitted and the TxBuffer number as arguments.

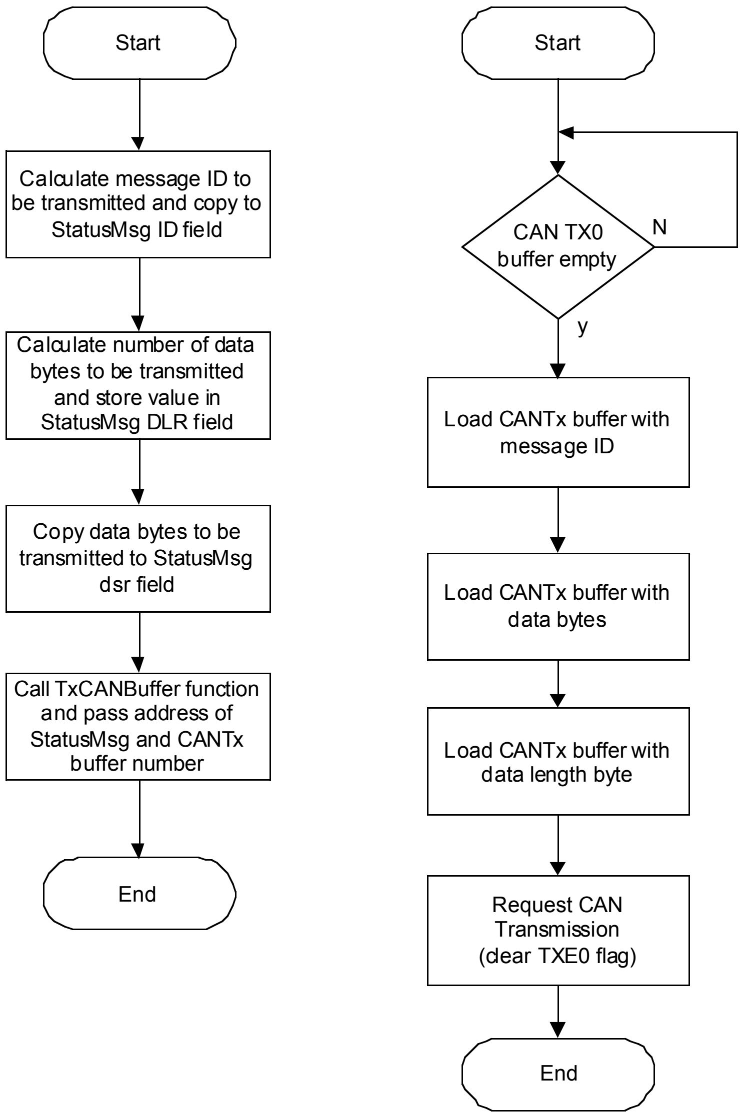


Figure 15 CAN Transmit flow diagrams

4.2.2.4 Master Node Main Loop

The main routine is essentially a simple infinite loop that waits for either a msgrxd system flag to be set or a loopcontrol system flag to be cleared. The msgrxd flag is set in the CANRx_ISR when a CAN message is received. If the flag is set, the MsgHandlerTable is called and the appropriate handler routine executed as described in the Receiving CAN Messages section. The loopcontrol flag is cleared periodically in the TIMBOVF_ISR, which overflows every 150ms. When the flag is cleared, the ScheduleMsg function is called and the selected mode executed. Once the CAN message is received or the mode executed, the main function resets the appropriate flags and returns to the infinite loop waiting for a flag to change again.

The main loop also performs initialization, by calling the appropriate initialization function, before the infinite loop is entered. Initialisation of the MCU registers, LIN drivers and application is also performed.

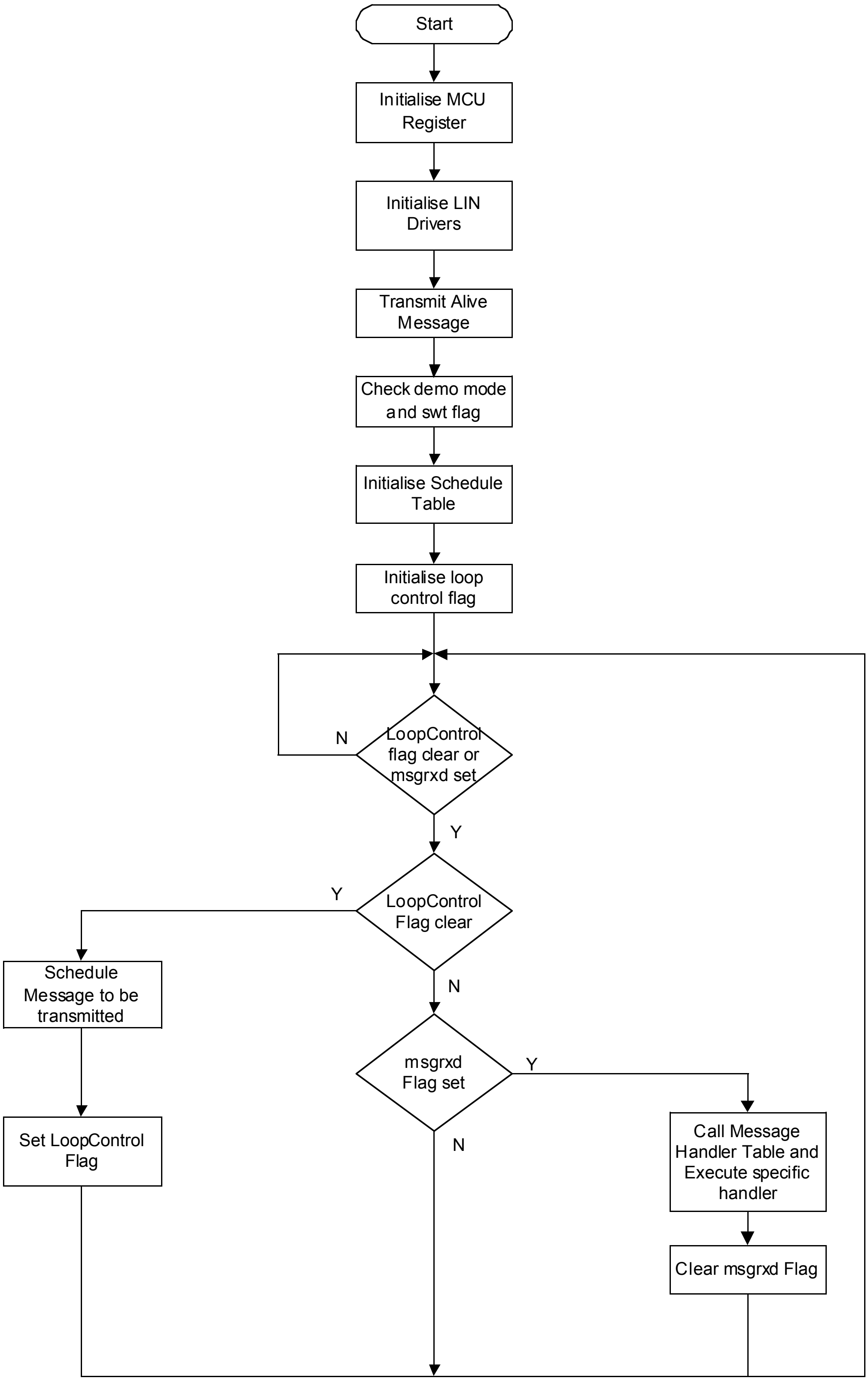


Figure 16 Main Loop flow diagram

4.3 Slave Code description

The slave code is entirely message driven. The software for each node is practically identical, the only difference being the messages the node is configured to recognize and the nodeID.

The individual slaves are configured to react to 3 preprogrammed message identifiers: a NodeX_Write, a NodeX_Read and a common broadcast message (NodeX_Write message that every slave is programmed to receive). See Table 1 for details. Each slave monitors every header that the master drives on the bus, but only reacts to its configured identifiers. If a NodeX_Write message is detected, it receives the message data, decodes the command, and either writes to its LED output port or the external output LED or enters SLEEP mode. If a NodeX_Read message is detected, the slave automatically transmits its status information bytes (i.e. Its ID and the HEX switch settings) on to the LIN bus.

Received messages are handled in exactly the same way as the master code handles CAN messages. When a new LIN message is received (NodeX_Write) the code calls a MsgHandlerTable (array of pointers to functions) and jumps to the appropriate MsgHandlerFunction.

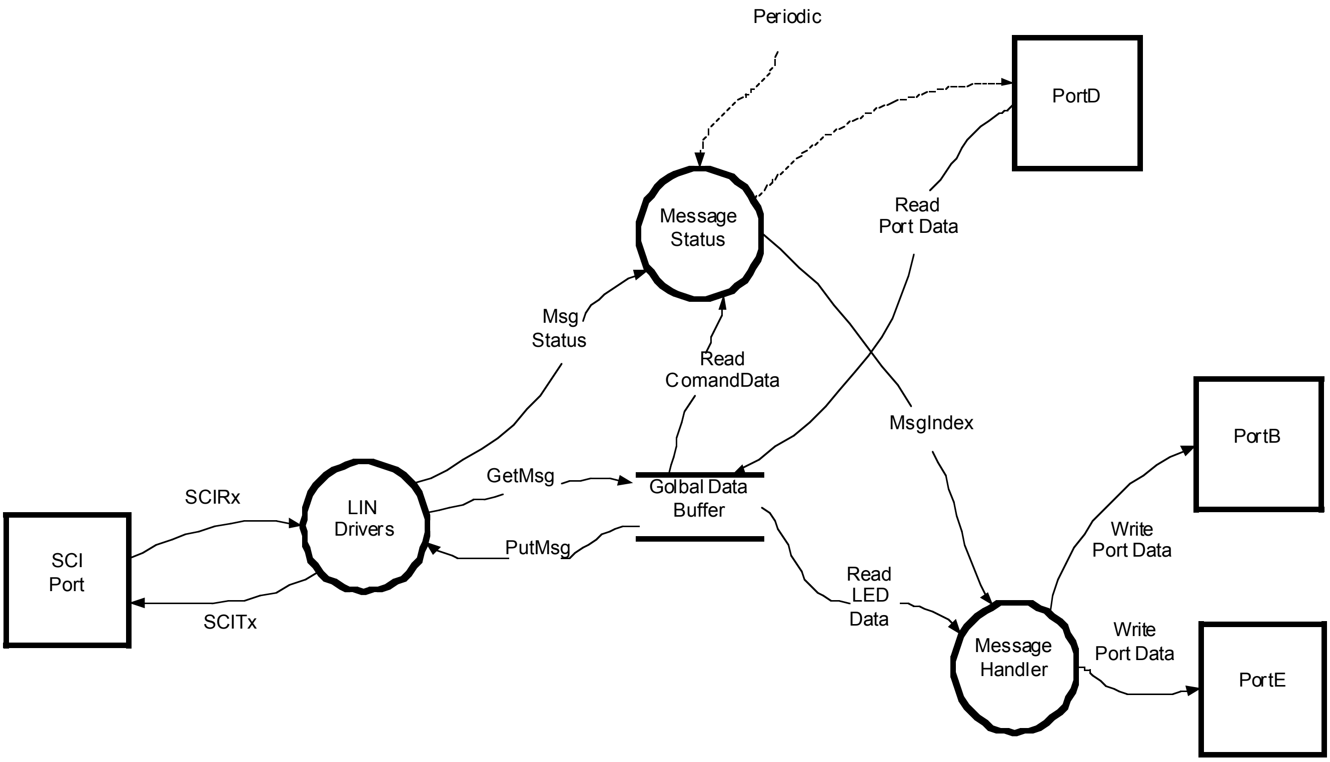


Figure 17 Slave code data flow diagram

4.3.1 Slave Code Main Function

The slave code's main function is very similar to the master's in that it performs some initialization before entering an infinite loop. During an iteration of the loop, the code updates the status message buffer by writing the HEX switch settings to the NodeX_Read message buffer using the LIN_PUTMsg() service and checks to see if a new received message has been detected. If a new message has been detected, the MsgHandlerTable is called and the appropriate handler function is executed. Several housekeeping tasks are also performed in the main loop, such as control of timeouts etc. Once the iteration is complete, the code jumps back to the start of the loop and performs the tasks again.

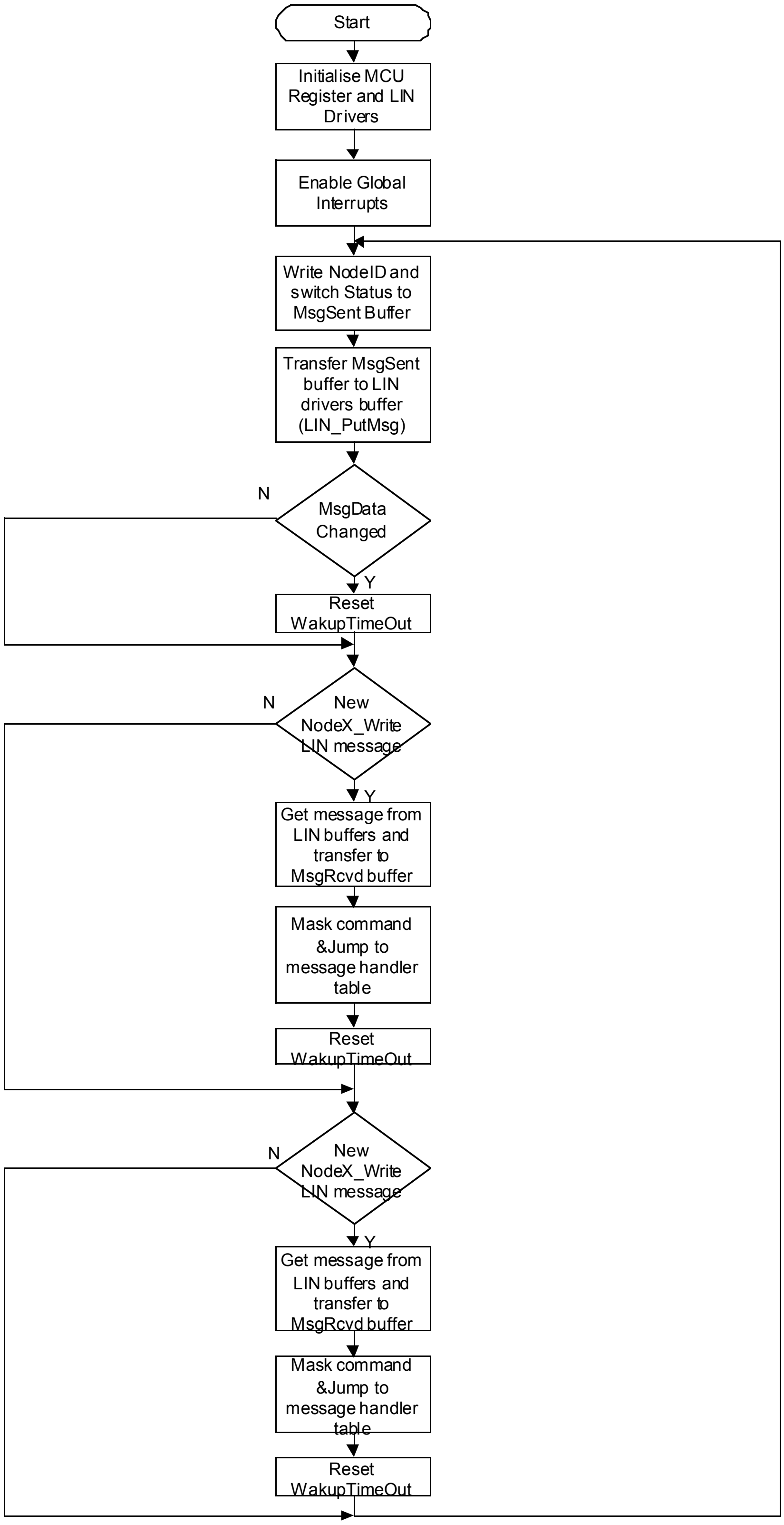


Figure 18 Main Loop flow diagram

4.3.2 Receiving LIN messages

Each slave node is configured to receive 2 LIN messages: a broadcast message and a NodeX_Write message. The first byte of the message is a command. The main function polls the message status flags of the configured messages and transfers the received data to the application receive buffer, MsgRcvd, when a valid message is received. The command byte is decoded and the appropriate handler function is executed. Each message handler is described below:

4.3.2.1 Rotating Handler

This handler is entered when the demo is configured for default mode (SLAVE_LEDS command transmitted by the master). The handler decodes the LED data received, to see if it is to illuminate the RED or GREEN LEDs, resets the appropriate time out counter and then outputs the data to its LED output port.

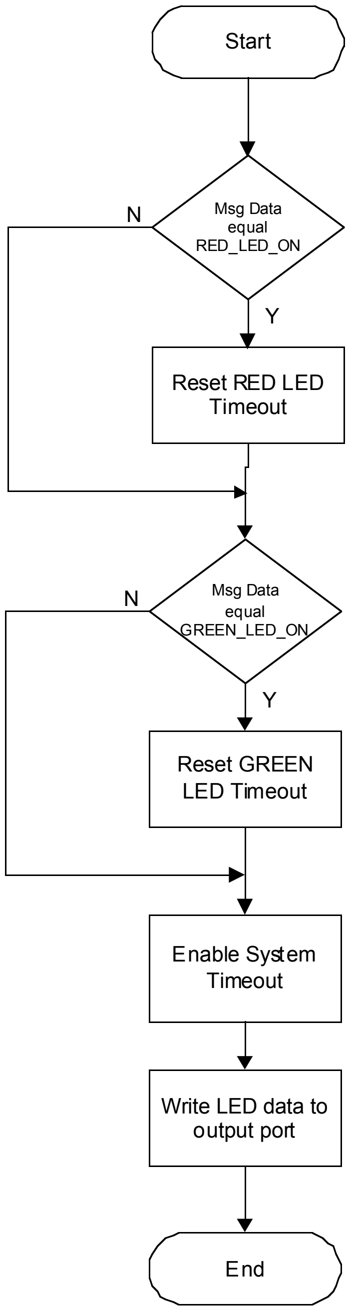


Figure 19 Rotating Message Handler flow diagram

4.3.2.2 Broadcast Handler

This handler is entered when the demo is configured for broadcast mode (BROADCAST command transmitted by the master). The handler disables the time out and then outputs the data to its LED output port.

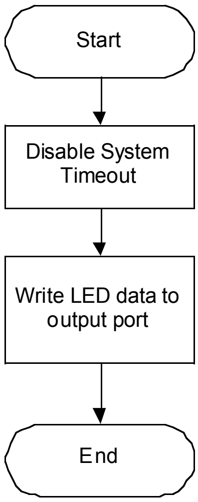


Figure 20 Broadcast Message Handler flow diagram

4.3.2.3 External Handler

This handler is entered when the demo is configured for external mode (CLOCK_LEDS_COMMAND command transmitted by the master). The handler enables the time out and then outputs the data to its external output port (PortE, bit4)

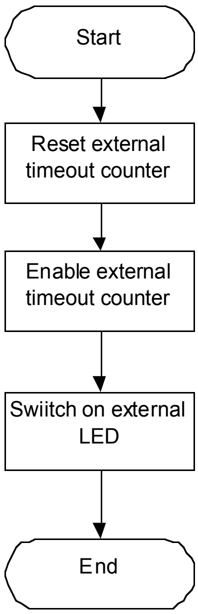


Figure 21 External Message Handler flow diagram

4.3.2.4 Identify Handler

This handler is entered when the demo is configured for IDENT mode (IDENT command transmitted by the master). The handler disables the time out and then outputs the data to its LED output port

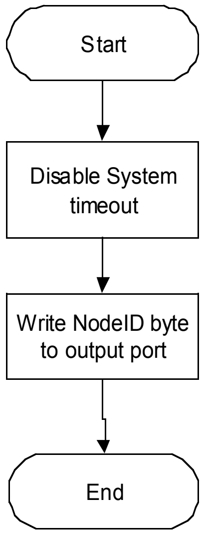


Figure 22 Identify Message Handler flow diagram

4.3.2.5 Default Handler

This handler consists of the function prototype. This is included for expandability and to ensure that the code does not ‘run away’ even if an invalid entry in the message handler vector table is accessed. I.e. All unused entries in the table jump to DefaultMessage handler.

4.3.2.6 SleepHandler

See sleep mode description

4.3.3 Sleep Mode

Sleep mode is the demo's low power mode. It is entered when the master transmits the SLEEP identifier (0x80). Each slave receives the command and jumps to the sleep message handler. The handler disables the voltage regulator by driving the LIN I/F into sleep mode.

Any node can be brought out of sleep mode by an external switch (switches around the circumference of the clock face) or a wake-up message (see protocol specification for details) detected on the LIN bus. The slave that is woken up transmits the wake-up message onto the bus, which wakes up the master and the other nodes.

Each slave distinguishes between a wake-up (switch on voltage regulator) and a normal power up sequence, by detecting if the master is present on the bus (master will be communicating on bus for standard power up sequence). The master transmits an ALIVE (0x0F) message every 150ms to signal its presence. If a slave detects an ALIVE message it assumes that it is a power up sequence. If the ALIVE is not detected within 200ms the slave assumes that it has woken up and subsequently transmits a wake-up message to the network.

5 Code Listings

5.1 Master Code – Master08.C

```
/*****************************************************************************************
                                                   Copyright (c)
File Name           :      MASTER08.C

Engineer            :      R29414

Location            :      EKB

Date Created        :      07/02/2000

Current Revision    :      $Revision: 1.0$

Notes               :      Master software for the LIN Demo

*****************************************************************************************
Freescale reserves the right to make changes without further notice to any
Product herein to improve reliability, function or design. Freescale 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. Freescale 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 Freescale product
could create a situation where personal injury or death may occur. Should
Buyer purchase or use Freescale products for any such unintended or
Unauthorized application, Buyer shall idemnify and hold Freescale and its
officers, employees, subsidiaries, affiliates, and distributors harmless
against all claims costs, damages, and expenses, and reasonable attorney fees
arising out of, directly or indirectly, any claim of personal injury or death
associated with such unintended or unauthorized use, even if such claim alleges
that Freescale was negligent regarding the design or manufacture of the part.
Freescale and the Freescale logo* are registered trademarks of Freescale Semiconductor, Inc.
*************************************************************************************/


/************************* System Include Files *****************************************/
#include <linapi.h>                                // LIN Drivers Header file
/************************* Project Include Files ****************************************/

#include <master08CodeReview.h>                    // Master08 header file
#include <common.h>                                // Common data structure
#include <port.h>                                  // Port register definitions
#include <timer.h>                                 // Timer register definitions
#include <sim.h>                                   // SIM Register definitions
#include <si.h>                                    // Serial Interface Register definitions
#include <kbd.h>                                   // Keyboard wakeup Register definitions
#include <mscan08.h>                               // msCAN Register definitions


/*************************** Constants ****************************************/
/* Slave Node Message array */
const SlaveNodeMsgType SlaveMsg [] =
{
    {0x0F,0x0F},                                   // Broadcast message */
    {0x01,0x11},                                   // Node1 Write messageID, Node1 Read messageID
    {0x02,0x12},                                   // Node2 Write messageID, Node2 Read messageID
    {0x03,0x13},                                   // Node3 Write messageID, Node3 Read messageID
```

```
    {0x04,0x14},                                // Node4 Write messageID, Node4 Read messageID
    {0x05,0x15},                                // Node5 Write messageID, Node5 Read messageID
    {0x06,0x16},                                // Node6 Write messageID, Node6 Read messageID
    {0x07,0x17},                                // Node7 Write messageID, Node7 Read messageID
    {0x08,0x18},                                // Node8 Write messageID, Node8 Read messageID
    {0x09,0x19},                                // Node9 Write messageID, Node9 Read messageID
    {0x0A,0x1A},                                // Node10 Write messageID, Node10 Read messageID
    {0x0B,0x1B},                                // Node11 Write messageID, Node11 Read messageID
    {0x0C,0x1C}                                 // Node12 Write messageID, Node12 Read messageID
};

/* Broadcast table settings */
const tU08 BroadcastTable [] =
{
    0x01,
    0x03,                                       // Data bytes transmitted when in broadcast mode
    0x07,
    0x0F,
    0x1F,
    0x3F,
    0x7F,
    0xFF
};

/******************************************************************************
Message handler vector table
******************************************************************************/

void (* const MsgHandlerTable[])() =
    {
    CommonMsgHandler,
    CommonMsgHandler,                           // If required different handlers can be used
    CommonMsgHandler,
    CommonMsgHandler,
    CommonMsgHandler,
    CommonMsgHandler,
    CommonMsgHandler,
    CommonMsgHandler,
    CommonMsgHandler,
    CommonMsgHandler,
    CommonMsgHandler,
    CommonMsgHandler,
    SleepHandler,
    ModeSelect,
    DefaultHandler
    };

/************************* Global Variables *********************************/

SlaveMsgBufferType CurrentSlaveStatus [12];     // Current slave node status
DemoModeType_t DemoMode = DEFAULT;              // Current demo mode (DEFAULT,BROADCAST,IDENT,SLEEP)
tFLAG   SystemFlags;                            // System flags
tTXBUF  StatusMsg;

tU08 MsgWriteBuf [2];                           // Temp write buffer used with LIN Drivers
tU08 ScheduleIndex = 0x01;                      // Slave node index
tU08 AltScheduleIndex = 0x00;
tU08 BroadcastIndex = 0x00;                     // Broadcast control index
tU08 MsgReadBuf1[2] = {0x00,0x00};              // Temp read buffer used with LIN Drivers
tU08 MsgReadBuf2[2] = {0x00,0x00};
tU08 CanMsgIndex = 0x00;
tU08 TimeoutCount = 0x00;                       // Timeout counter for slave nodes. Set at 5ms
tU08 LoopControlTime = 0x00;
```

```
/************************* #Defines ******************************************/
/* Points to register block in memory */
#define AZ60_PORT (*(tPORT*) (0x0000))
#define AZ60_TIMER (*(tTIMER*) (0x0020))
#define AZ60_SI (*(tSI*) (0x0010))
#define AZ60_SIM (*(tSIM*) (0xFE00))
#define AZ60_KBD (*(tKBD*) (0x001A))
#define AZ60_MSCAN08 (*(tMSCAN*) (0x0500))

/******************************************************************************
Function Name         :      Main
Engineer              :      R29414
Date                  :      09/02/2000

Parameters            :      none
Returns               :      none
Notes                 :      Main loop
******************************************************************************/

void main(void)
{
    /* Local variables */
    tU08 i;                                        // loop control

    MCUInitialisation();                           // Initialisation of MCU register

    LIN_Init();                                    // LIN Drivers Initialisation service

    /****************************************
    LIN drivers TOC attached to output ports
    Disabled on production software
    ****************************************/

    /* Disable timer channels from ports */
    AZ60_TIMER.tasc0.byte = AZ60_TIMER.tasc0.byte & 0xF3;
    AZ60_TIMER.tasc1.byte = AZ60_TIMER.tasc1.byte & 0xF3;

    /* Enable MC33399 device */                    //LIN I/F
    AZ60_PORT.pte.bit.pte3 = 1;
    AZ60_PORT.ddre.bit.ddre3 = 1;


    for (i=0; i<0xff; ++i)                         // Wait for LIN to switch ON
         ;

    /* Enable Global Interrupts */
    asm
{
    cli
}

    /* Transmit Alive Message */
    MsgWriteBuf[0] =   ALIVE_COMMAND;
    MsgWriteBuf[1] =   ALIVE_BYTE;
    LIN_PutMsg(SlaveMsg[0].WriteMsg, MsgWriteBuf); // Copy transmit data to LIN buffers
    LIN_RequestMsg(SlaveMsg[0].WriteMsg);          // Transmit LIN message

    if ((AZ60_PORT.ptd.byte & 0xF0) != 0x00)       // Standalone mode or CAN mode
    {
        SystemFlags.bit.canmode=1;                 // CAN mode
    }

    ScheduleTableInit();                           // Initialise schedule table
```

```
    SystemFlags.bit.loopcontrol=1;                  // Initialise loop control

    /* Main Loop */
while(1)
{
    /* Wait for timer interrupt or CAN receive message */
    while ((SystemFlags.bit.loopcontrol==1) && (SystemFlags.bit.msgrxd==0))
        ;

        if ((SystemFlags.bit.loopcontrol==0) && (SystemFlags.bit.msgrxd==0))
        {
                                                    // Timer interrupt
            ScheduleMsg();                          // Schedule another message

        SystemFlags.bit.loopcontrol = 1;            // Set up for next loop iteration
}
    else                                            // CAN message received
        {
        CanMsgIndex = (AZ60_MSCAN08.rxbuf.dsr[0] & 0x0F);
        MsgHandlerTable[CanMsgIndex]();             // Jump to message handler table

            SystemFlags.bit.msgrxd = 0;             // Set up for next CAN message

    }

    }   // end of while (1)
}           // end of main


/*******************************************************************************
Function Name           :       ScheduleTableInit
Engineer                :       R29414
Date                    :       09/02/2000

Parameters              :       none
Returns                 :       none
Notes                   :       Initialise the schedule table. It identifies the slave
                                on the LIN bus
*******************************************************************************/
void ScheduleTableInit (void)
{
    /* Local variables */
    tU08 i;                                         // Index counter
    tU08 msgTempBuffer[] = {0x00, 0x00};            // Initialise temp buffer

    /* LIN Schedule table Initialisation */
    for (i=0 ; i<12 ; i++)
    {
        /* Clear all elements in Slave table array*/
        CurrentSlaveStatus [i].Byte0 = NO_NODE;     // NO_NODE = 0x00
        CurrentSlaveStatus [i].Byte1 = NO_NODE;

    }

    /* Check nodes on Bus */                        // Nodes 1 to 12
    for (i=1 ; i<13 ; i++)
    {
        while (LIN_RequestMsg(SlaveMsg[i].ReadMsg) != LIN_OK)   // Transmit LIN header
            ;

        TimeoutCount = 0x00;                        // Initialise Timeout count

        SystemFlags.bit.linmsgtimeout=0;            // Initialise timeout flag
```

```
        SystemFlags.bit.starttimeout=1;            // Start time out

        while (LIN_MsgStatus(SlaveMsg[i].ReadMsg) !=LIN_OK && SystemFlags.bit.linmsgtimeout==0)
             ;                                     // Wait for new data or timeout

        SystemFlags.bit.starttimeout=0;            // Stop timeout. Disables timeout in TIMBOVF_ISR


        if (SystemFlags.bit.linmsgtimeout==0)      // No Timeout
        {
                                                   // Valid message received
             LIN_GetMsg (SlaveMsg[i].ReadMsg, msgTempBuffer);   // Transfer Node status to Temp buffer  */

             /* Update Current slave status */
             CurrentSlaveStatus[i-1].Byte0 = msgTempBuffer[0];  // 1=< i <13
             CurrentSlaveStatus[i-1].Byte1 = msgTempBuffer[1];

        }

        /* Transmit initial status via CAN if not in standalone mode */
        if (SystemFlags.bit.canmode==1)            // Standalone mode canmode= 0
        {
             (tU16) i;                             // i is the CANId
             TxCANMsg(msgTempBuffer,i);            // Transmit CAN message */
        }

    } // end of for

} // End of scheduleTableInit function

/******************************************************************************
Task Name            :      ScheduleMsg
Engineer             :      R29414
Date                 :

Parameters           :      none
Returns              :      none
Notes                :      Determine mode of demo and calls handler
******************************************************************************/

void ScheduleMsg(void)
{

    /* Transmit Alive Message */
    MsgWriteBuf[0] =   ALIVE_COMMAND;
    MsgWriteBuf[1] =   ALIVE_BYTE;
    LIN_PutMsg(SlaveMsg[0].WriteMsg, MsgWriteBuf);
    LIN_RequestMsg(SlaveMsg[0].WriteMsg);          // Transmit message
    while (LIN_MsgStatus(SlaveMsg[0].WriteMsg) !=LIN_OK)  // Wait for message request complete
         ;

    if (SystemFlags.bit.canmode==0)                // Check mode
    {
         DemoMode = AZ60_PORT.ptd.byte & 0x03;     // Stand alone mode
    }
    switch (DemoMode)
    {
         case DEFAULT:
              DefaultMsgHandler();                 // Call NodeMsgHandler
              break;

         case BROADCAST:
```

```
            BroadcastMsgHandler();                  // Call BroadcastMsgHandler
            break;

        case IDENT:
            IdentMsgHandler();                      // Call IdentMsgHandler
            break;

        case SLEEPMODE:                             // Call SleepHandler
        SleepHandler();
            break;

        default:
            break;
    }   // End of switch

} // End scheduleMsg

/******************************************************************************
Task Name           :       DefaultMsgHandler
Engineer            :       R29414
Date                :       26/07/2000

Parameters          :       none
Returns             :       none
Notes               :
******************************************************************************/
void DefaultMsgHandler(void)
{

    if ((ScheduleIndex <1) || (ScheduleIndex >= 13))           // Check index is within range 1 to 12
    {
                                                                // Out of range
        ScheduleIndex = 1;                                      // make index = 1
    }

    /* Within range */
    if ((ScheduleIndex == 12) || (ScheduleIndex == 6))         // Valid ScheduleIndex
    {
        /* Node 12 or 6 */
        LINTransmit(ScheduleIndex, ALL_LEDS_ON, MsgReadBuf1);  // Transmit specified message header frame
    }
    else
    {
        AltScheduleIndex = 12 - ScheduleIndex;                  // Calculate alternate index

        LINTransmit(ScheduleIndex, RED_LEDS_ON, MsgReadBuf1);  // Transmit specified message header frame

        LINTransmit(AltScheduleIndex, GREEN_LEDS_ON, MsgReadBuf2);    // Transmit specified message header
frame */
    }

    /* Compare status data with previous data */
    if ((CurrentSlaveStatus[ScheduleIndex - 1].Byte0 != MsgReadBuf1[0]) ||
(CurrentSlaveStatus[ScheduleIndex - 1].Byte1 != MsgReadBuf1[1]))
    {
        if (SystemFlags.bit.canmode==1)                         // Do not transmit if standalone mode
        {                                                       // Standalone =0

            (tU16) ScheduleIndex;                               // ScheduleIndex is the CANId

            TxCANMsg(MsgReadBuf1,ScheduleIndex);                // Transmit CAN message
        }

        CurrentSlaveStatus[ScheduleIndex - 1].Byte0 = MsgReadBuf1[0]; // Update status data
```

```
        CurrentSlaveStatus[ScheduleIndex - 1].Byte1 = MsgReadBuf1[1];
    }

} // End DefaultMsgHandler

/*******************************************************************************
Task Name           :       LINTransmit
Engineer            :       R29414
Date                :       26/07/2000

Parameters          :       index, ledcommand, &msgbuffer
Returns             :       none
Notes               :
*******************************************************************************/
void LINTransmit(tU08 index, tU08 ledcommand, tU08 *msgbuffer)
{
    TimeoutCount = 0x00;                            // Initialise Timeout count
    SystemFlags.bit.linmsgtimeout=0;                // Initialise Timeout

    LIN_RequestMsg(SlaveMsg[index].ReadMsg);        // Transmit specified message header frame

    SystemFlags.bit.starttimeout=1;                 // Start Timeout

    /* Wait for new data or timeout */
    while (LIN_MsgStatus(SlaveMsg[index].ReadMsg) !=LIN_OK && SystemFlags.bit.linmsgtimeout==0)
        ;
    SystemFlags.bit.starttimeout=0;                 // Stop timeout

    if (SystemFlags.bit.linmsgtimeout==0)           // Check if valid LIN message received
    {
        LIN_GetMsg(SlaveMsg[index].ReadMsg, msgbuffer);   // Store status information

        MsgWriteBuf[0] =   SLAVE_LEDS_COMMAND;
        MsgWriteBuf[1] =   ledcommand;              //Set up write buffer to transmit command to node

        /* Transmit default message to node x */
        LIN_PutMsg(SlaveMsg[index].WriteMsg, MsgWriteBuf);
        LIN_RequestMsg(SlaveMsg[index].WriteMsg);
        while (LIN_MsgStatus(SlaveMsg[index].WriteMsg) !=LIN_OK)      // Wait for transmission to complete
*/
            ;
    }
    else
    {                                               // Timeout
        msgbuffer[0] = NO_NODE;                     // Update temporary buffer indicating that no node */
        msgbuffer[1] = NO_NODE;

    }

} /* End LINTransmit */

/*******************************************************************************
Task Name           :       BroadcastMsgHandler
Engineer            :       R29414
Date                :

Parameters          :       none
Returns             :       none
Notes               :
*******************************************************************************/

void
BroadcastMsgHandler(void)
```

```
{
    if (BroadcastIndex < 0 || BroadcastIndex >7)          // Dont let Index outwith range
    {
        BroadcastIndex = 0;
    }

    MsgWriteBuf[0] =   BROADCAST_COMMAND;
    MsgWriteBuf[1] =   BroadcastTable[BroadcastIndex];
    LIN_PutMsg(SlaveMsg[0].WriteMsg, MsgWriteBuf);
    LIN_RequestMsg(SlaveMsg[0].WriteMsg);
    while (LIN_MsgStatus(SlaveMsg[0].WriteMsg) !=LIN_OK)  // Wait for message complete
        ;

    if (BroadcastIndex == 0)
    {
        SystemFlags.bit.updowncontrol = 0;                // Up Counter
    }
    if (BroadcastIndex == 7)
    {
        SystemFlags.bit.updowncontrol = 1;                // Down Counter
    }

} // End of BroadcastMsgHandler

/*******************************************************************************
Task Name           :      IdentMsgHandler
Engineer            :      R29414
Date                :

Parameters          :      none
Returns             :      none
Notes               :
*******************************************************************************/

void
IdentMsgHandler(void)
{
    MsgWriteBuf[0] =   IDENT_COMMAND;
    MsgWriteBuf[1] =   AZ60_PORT.ptd.byte;                // Hex switch input
    LIN_PutMsg(SlaveMsg[0].WriteMsg, MsgWriteBuf);
    LIN_RequestMsg(SlaveMsg[0].WriteMsg);
    while (LIN_MsgStatus(SlaveMsg[0].WriteMsg) !=LIN_OK); // Wait for message complete

} // End of BroadcastMsgHandler


/*******************************************************************************
Task Name           :      MCUInitialisation
Engineer            :      R29414
Date                :

Parameters          :      none
Returns             :      none
Notes               :      Initialise the mcu hardware
*******************************************************************************/

void MCUInitialisation(void)
{
    /* Device configuration */
    AZ60_KBD.config1.bit.copd = 1;                        // Disable Watchdog
    AZ60_SIM.config2.byte = 1;                            // AZ Mode and CAN enabled

    /* Ports Initialisation */
    AZ60_PORT.ddrd.byte = 0x00;                           // PortD i/p for HEX switches
```

```
    AZ60_PORT.ptb.byte = 0xFF;
    AZ60_PORT.ddrb.byte = 0xFF;                         // Port B o/p for LEDs

    AZ60_PORT.ptc.bit.ptc0 = 0;
    AZ60_PORT.ddrc.bit.ddrc0 = 1;

    /* CAN I/F Configuration */
    SetMC33388Mode(NORMAL);                             // put MC33388 into normal mode
    AZ60_PORT.ddrf.byte = PTF4|PTF3;                    // PTF3=EN and PTF4=STB pins on MC33388

    SystemFlags.byte = 0;                               // reset all system flags

    InitialiseMSCAN08();                                // MSCAN Initialisation

    StatusMsg.id.w[0] = 0x0000;                         // standard 11-bit ID = 1
    StatusMsg.dlr = 0;                                  // 0 data bytes
    StatusMsg.tbpr = 0;                                 // set CAN status message ID, DLR and priority

    /* Initialise TIMB Overflow */
    AZ60_TIMER.tbsc.byte = 0x30;                        // Reset TIMB
    AZ60_TIMER.tbmod.word = 0x3E8;                      // 1ms overflow when 1Meg Bus and pre-scale=0
    AZ60_TIMER.tbsc.byte = 0x40;                        // Enable TIMB OVR Interrupt & Start timer

} //End of MCU Initialisation

/*******************************************************************************
Function Name       :       TxCANMsg
Engineer            :       R29414
Date                :       22/02/00

Parameters          :       *MsgBuffer, CANId
Returns             :       None
Notes               :
*******************************************************************************/
void TxCANMsg (tU08 *MsgBuffer, tU16 CANId)
{
    tU08 i;

    StatusMsg.id.w[0] = ((CANId <<5) & 0xFFE0);         // Calculate id

    StatusMsg.dlr = sizeof(MsgBuffer);                  // Calculate number of data bytes in buffer

    for (i=0 ; i < StatusMsg.dlr ; i++ )                // Transfer data bytes to CAN message buffer
    {
        StatusMsg.dsr[i] = MsgBuffer[i];
    }
    TxCANBuffer(&StatusMsg,MSCAN_TX0);

}
/*******************************************************************************
Function Name       :       TxCANBuffer
Engineer            :       R38917
Date                :       11/02/00

Parameters          :       *Buffer, TxBufferID
Returns             :       None
Notes               :
*******************************************************************************/

void TxCANBuffer(tTXBUF *Buffer, tU08 TxBufferID)
    {
    tU08  i;
```

```
    while(!(AZ60_MSCAN08.ctflg.byte & TxBufferID))
         ;
/***
Timeout Period And Abort Stuff ?
***/

    AZ60_MSCAN08.txbuf[(TxBufferID >> 1)].id.l = Buffer -> id.l;

    for(i=0 ; i < Buffer -> dlr ; i++)
         AZ60_MSCAN08.txbuf[(TxBufferID >> 1)].dsr[i] = Buffer -> dsr[i];

    AZ60_MSCAN08.txbuf[(TxBufferID >> 1)].dlr  = Buffer -> dlr;
    AZ60_MSCAN08.txbuf[(TxBufferID >> 1)].tbpr = Buffer -> tbpr;

    AZ60_MSCAN08.ctflg.byte = TxBufferID;
    }

/*******************************************************************************
Function Name          :      SetMC33388Mode
Engineer               :      R38917
Date                   :      11/02/00

Parameters             :      Mode
Returns                :      None
Notes                  :
*******************************************************************************/

void SetMC33388Mode(enum tMC33388 Mode)
    {
    switch(Mode)
         {
         case SLEEP:
              AZ60_PORT.ptf.byte = 0;              //STB=EN=0
              break;

         case SLEEP_RQ:
              AZ60_PORT.ptf.byte = PTF3;           //STB=0,EN=1
              break;

         case RX_ONLY:
              AZ60_PORT.ptf.byte = PTF4;           //STB=1,EN=0
              break;

         case NORMAL:
              AZ60_PORT.ptf.byte = PTF4|PTF3;      //STB=EN=1
              break;

         default:
              ;
         }
    }

/*******************************************************************************
Function Name          :      CommonMsgHandler
Engineer               :      R38917
Date                   :      20/07/00

Parameters             :      None
Returns                :      None
Notes                  :      This handler is common to the 12 messages as only the index is different.
*******************************************************************************/

void CommonMsgHandler(void)                   // Common handler replaces Node1 - Node12 handlers
```

```
    {

         tU08 msgSent[] = {CLOCK_LEDS_COMMAND, ALL_LEDS_OFF};    // Local Declaration

         LIN_PutMsg (SlaveMsg[CanMsgIndex].WriteMsg, msgSent);   // Transfer data to LIN Msg buffer
         LIN_RequestMsg(SlaveMsg[CanMsgIndex].WriteMsg);         // Request a message

        while (LIN_MsgStatus(SlaveMsg[CanMsgIndex].WriteMsg) != LIN_OK)// Wait a while
         ;
    }

/******************************************************************************
Function Name         :      SleepHandler
Engineer              :      R38917
Date                  :      11/02/00

Parameters            :      None
Returns               :      None
Notes                 :
******************************************************************************/

void SleepHandler(void)
    {
    /* Send out LIN Sleep command */
    MsgWriteBuf[0] =   SLEEP_COMMAND;
    MsgWriteBuf[1] =   0x00;
    LIN_PutMsg(SlaveMsg[0].WriteMsg, MsgWriteBuf);
    LIN_RequestMsg(SlaveMsg[0].WriteMsg);


    while (LIN_MsgStatus(SlaveMsg[0].WriteMsg) !=LIN_OK)  // Wait for message complete
         ;

    AZ60_PORT.pte.bit.pte3 = 0;                        // Disable UPL

    SetMC33388Mode(SLEEP_RQ);

    SetMC33388Mode(SLEEP);

    while(1)
         ;                                             //placing MC33388 into SLEEP mode switches off the Vreg
    }


/******************************************************************************
Function Name         :      ModeSelect
Engineer              :      R38917
Date                  :      10/02/00

Parameters            :      None
Returns               :      None
Notes                 :
******************************************************************************/

void ModeSelect(void)
    {
    switch(AZ60_MSCAN08.rxbuf.dsr[1])                  // Demo mode in byte1
         {
         case DEFAULT:
              DemoMode = DEFAULT;                      // Put demo in default mode
              break;

         case BROADCAST:
```

```
            DemoMode = BROADCAST;                   // Put demo in broadcast mode
            BroadcastIndex = 0;
            break;

        case IDENT:
            DemoMode = IDENT;                       // Put demo in IDENT mode
            break;

        default:
            ;
        }
    }

/*****************************************************************************
Function Name       :      DefaultHandler
Engineer            :      R38917
Date                :      11/02/00

Parameters          :      None
Returns             :      None
Notes               :
*****************************************************************************/

void DefaultHandler(void)
    {
    }


/*****************************************************************************
Function Name       :      InitialiseMSCAN08
Engineer            :      R38917
Date                :      10/02/00

Parameters          :      None
Returns             :      None
Notes               :
*****************************************************************************/

void InitialiseMSCAN08(void)
    {
    AZ60_MSCAN08.cmcr0.bit.sftres = 1;              //put CAN module in soft reset

    AZ60_MSCAN08.cbtr0.byte = CBT0_125K;
    AZ60_MSCAN08.cbtr1.byte = CBT1_125K;

    AZ60_MSCAN08.cid.mr.l = 0xFFFFFFFF;             //accept all messages

    AZ60_MSCAN08.cmcr0.bit.sftres = 0;              //release CAN module from soft reset

    AZ60_MSCAN08.crier.bit.rxfie = 1;               //enable CAN receive interrupts

    while(!AZ60_MSCAN08.cmcr0.bit.synch)
        ;                                           //wait for CAN bus to synchronize
    }

/*****************************************************************************
Task Name           :      TimerB Overflow
Engineer            :      R29414
Date                :

Parameters          :      none
Returns             :      none
Notes               :      Overflow period set at 1ms.
*****************************************************************************/
```


Freescale Semiconductor, Inc.

```
/* Hiware compiler */

#pragma TRAP_PROC


void TIMBOVF_ISR (void)
{
                                                    // Loop control timer timeout
    if (LoopControlTime<LOOP_CONTROL_PERIOD)        // LoopControlPeriod set in header file
    {
         LoopControlTime++;
    }
    else
    {
         LoopControlTime=0x00;                      // Reset LoopControlTime
         SystemFlags.bit.loopcontrol =0;            // Clear system flag
         ScheduleIndex++;                           // Increment Schedule Table Index

         if (ScheduleIndex >=13)
         {

              ScheduleIndex = 0x01;                 // Reset Schedule Index
         }

         if (DemoMode == BROADCAST)
         {
              if (SystemFlags.bit.updowncontrol == 0)
              {
                      BroadcastIndex++;
              }
              else
              {
                      BroadcastIndex--;
              }
         }
    }

    if (SystemFlags.bit.starttimeout == 1)
    {
         if (TimeoutCount<TIMEOUT_PERIOD)           // Timeoutperiod set in header file
         {
              TimeoutCount++;
         }
         else
         {

              SystemFlags.bit.linmsgtimeout =1;     // Set Timeout flag
         }
    }

    /* Clear Interrupt flag */
    AZ60_TIMER.tbsc.byte &= ~TOF;                   // Read TBSC0 and write 0 to CH0F


}   // End of ISR

/******************************************************************************
Function Name        :      CANRxISR
Engineer             :      R38917
Date                 :      10/02/00

Parameters           :      None
```

```
Returns              :      None
Notes                :      CAN message received interrupt service routine
*****************************************************************************/

#pragma TRAP_PROC

void CANRx_ISR(void)
{
    SystemFlags.bit.msgrxd = 1;

    AZ60_MSCAN08.crflg.bit.rxf = 1;                 //clear interrupt flag
} // End ISR
```

5.2 Master Code – MASTER08.H

```
/*****************************************************************************
                                              Copyright (c)
File Name            :      MASTER08.C

Engineer             :      R29414

Location             :      EKB

Date Created         :      07/02/2000

Current Revision     :      $Revision:1.0 $

Notes                :      LIN driver header file

*****************************************************************************
Freescale reserves the right to make changes without further notice to any
Product herein to improve reliability, function or design. Freescale 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. Freescale 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  Freescale product
could create a situation where personal injury or death may occur. Should
Buyer purchase or use Freescale products for any such unintended or
unauthorized application, Buyer shall idemnify and hold Freescale and its
officers, employees, subsidiaries, affiliates, and distributors harmless
against all claims costs, damages, and expenses, and reasonable attorney fees
arising out of, directly or indirectly, any claim of personal injury or death
associated with such unintended or unauthorized use, even if such claim alleges
that Freescale was negligent regarding the design or manufacture of the part.
Freescale and the Freescale logo* are registered trademarks of Freescale, Inc.
*********************************************************************************/
#ifndef  MASTER08CODEREVIEW_H
#define  MASTER08CODEREVIEW_H


/************************* System Include Files ******************************/

/************************* Project Include Files *****************************/
#include "c:\header_files\hc08\common.h"           // common data structure
#include "c:\header_files\hc08\mscan08.h"          // common data structure

/************************* User #Defines *************************************/
/* Timmer period Control */
#define LOOP_CONTROL_PERIOD 150
#define TIMEOUT_PERIOD      10
```

```
/************************ typedefs *****************************************/

typedef struct SlaveNodeMsgStruct
    {
        tU08        WriteMsg;
        tU08        ReadMsg;

    } SlaveNodeMsgType;


typedef struct SlaveMsgBufferStruct
    {
        tU08        Byte0;
        tU08        Byte1;

    } SlaveMsgBufferType;


typedef union
    {
        tU08        byte;
    struct
        {
        tU08 msgrxd:1;                  //target connection established
        tU08 loopcontrol :1;            //Loop control flag
        tU08 updowncontrol :1;          //Broadcast message up down flag 0=Up, 1=Down
        tU08 canmode :1;                //Flag indicates mode.  Standalone = 1 CAN = 0
        tU08 linmsgtimeout :1;          //LIN timeout.  Timeout = 1, OK = 0
        tU08 starttimeout :1;           //LIN timeout.  Timeout = 1, OK = 0
        tU08 :2;                        // not used
        }bit;
    }tFLAG;



typedef enum
{
    DEFAULT,
    BROADCAST,
    IDENT,
    SLEEPMODE
} DemoModeType_t;


enumtMC33388
    {
    SLEEP,
    SLEEP_RQ,
    RX_ONLY,
    NORMAL
    };

/************************ #Defines *****************************************/
/* Timmer period Control */


#define NODE_CONNECTED      0x80
#define NO_NODE             0x00

#define CBT0_125K           0xC0
#define CBT1_125K           0xD8                //based on a 4MHz xtal, 3 sampling points, SJW = 4

#define MSCAN_TX0           0x01
```

```
#define MSCAN_TX1           0x02
#define MSCAN_TX2           0x04


#define LED0_ON             0x01
#define LED1_ON             0x02
#define LED2_ON             0x04
#define LED3_ON             0x08
#define LED4_ON             0x10
#define LED5_ON             0x20
#define LED6_ON             0x40
#define LED7_ON             0x80
#define ALL_LEDS_ON         0x00
#define ALL_LEDS_OFF        0xFF
#define  GREEN_LEDS_ON      0x0F
#define  RED_LEDS_ON        0xF0

/* command bytes */
#define SLAVE_LEDS_COMMAND  0x01
#define BROADCAST_COMMAND   0x02
#define CLOCK_LEDS_COMMAND  0x03
#define IDENT_COMMAND       0x04
#define SLEEP_COMMAND       0x08
#define ALIVE_COMMAND       0x0F
#define ALIVE_BYTE          0xAD


/************************* Macros ********************************************/


/************************* Prototypes ****************************************/
void MCUInitialisation (void);
void ScheduleTableInit (void);
void ScheduleMsg (void);
void DefaultMsgHandler (void);
void BroadcastMsgHandler (void);
void IdentMsgHandler(void);
void TxCANMsg (tU08 * , tU16);

void CommonMsgHandler(void);
void TxCANBuffer(tTXBUF * ,tU08);
void SetMC33388Mode(enum tMC33388);
void SleepHandler(void);
void StatusRequestHandler(void);
void ModeSelect(void);
void DefaultHandler(void);
void InitialiseMSCAN08(void);
void LINTransmit(tU08, tU08, tU08 *);


#endif/* End of Header file ifndef*/
```

5.3 Slave Code – SLAVE08.C

```
/******************************************************************************
                                                 Copyright (c)
File Name           :       SLAVE08.C

Engineer            :       TTZ740
Project             :       SAE Demo  Project

Location            :       EKB
```

```
Date Created        :      27 January 2000

Current Revision    :      $Revision:1.0

Functions           :
Tasks               :

********************************************************************************
Freescale reserves the right to make changes without further notice to any
Product herein to improve reliability, function or design. Freescale 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. Freescale 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  Freescale product
could create a situation where personal injury or death may occur. Should
Buyer purchase or use Freescale products for any such unintended or
unauthorized application, Buyer shall idemnify and hold Freescale 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 ordeath
associated with such unintended or unauthorized use, even if such claim alleges
that Freescale was negligent regarding the design or manufacture of the part.
Freescale and the Freescale logo* are registered trademarks of Freescale Inc..
*************************************************************************************/


/************************* System Include Files ******************************/

#include "SAEdemo.h"                    //demo header file
#include <linapi.h>                     //lin driver api
#include <port.h>                       //port registers definitions
#include <sim.h>                        //system register definitions
#include <kbd.h>                        //register definitions
#include <si.h>                         //register definitions
#include <timer.h>                      //register definitions

/************************* Declarations **************************************/

/*************************** # defines ***************************************/

#define AZ60        (*(tPORT    *)(0x0000))
#define SIM         (*(tSIM     *)(0xFE00))
#define KBD         (*(tKBD     *)(0x001A))
#define SI          (*(tSI      *)(0x0010))
#define TIMER       (*(tTIMER   *)(0x0020))

/************************* typedefs ******************************************/

typedef union
    {
    tU08     byte;
    struct
        {
        tU08 enableTimeout       :1;   //enable defautl mode timeout
        tU08 enableExternal      :1;   //enable external timeout
        tU08 disableWaketime     :1;
        tU08                     :5;   //not used
        }bit;
    }tFLAG;
```

```
enum    tMC33388
    {
    SLEEP,
    SLEEP_RQ,
    RX_ONLY,
    NORMAL
    };

/************************* Global Variables *********************************/

tFLAG   SystemFlags;

tU08    REDTIMticks = 0;
tU08    GREENTIMticks = 0;
tU08    EXTERNTIMticks = 0;
tU08    TIMticks = 0;

tU08    MsgSent[2];
tU08    MsgRcvd[2];

tU16    WakeupTimeout = 0;

/************************* Prototypes ****************************************/


/******************************************************************************
Function Name       :     SetMC33388Mode
Engineer            :     r38917
Date                :     11/02/00

Parameters          :     Mode
Returns             :     None
Notes               :
******************************************************************************/

void SetMC33388Mode(enum tMC33388 Mode)
    {
    switch(Mode)
        {
        case SLEEP:
            AZ60.ptf.byte = 0;                  //STB=EN=0
            break;

        case SLEEP_RQ:
            AZ60.ptf.byte = PTF3;               //STB=0,EN=1
            break;

        case RX_ONLY:
            AZ60.ptf.byte = PTF4;               //STB=1,EN=0
            break;

        case NORMAL:
            AZ60.ptf.byte = PTF4|PTF3;          //STB=EN=1
            break;

        default:
            ;
        }
    }

/******************************************************************************
Function Name       :     Rotating LEDs message
Engineer            :     TTZ740
```

```
Date                 :      10/02/00

Parameters           :      None
Returns              :      None
Notes                :
*****************************************************************************/

void RotatingHandler(void)
    {
    if((MsgRcvd[1] & RED_LED_MASK) != RED_LED_MASK)
         REDTIMticks = 0;                         //reset red timeout period

    if((MsgRcvd[1] & GREEN_LED_MASK) != GREEN_LED_MASK)
         GREENTIMticks = 0;                       //reset green timeout period

    SystemFlags.bit.enableTimeout = 1;
    AZ60.ptb.byte = MsgRcvd[1];                   //send data to portb to switch on LEDs
    }

/*****************************************************************************
Function Name        :      Broadcast message
Engineer             :      TTZ740
Date                 :      10/02/00

Parameters           :      None
Returns              :      None
Notes                :
*****************************************************************************/

void BroadcastHandler(void)
    {
    SystemFlags.bit.enableTimeout = 0;
    AZ60.ptb.byte = ~MsgRcvd[1];                  //output data byte to port
    }

/*****************************************************************************
Function Name        :      External LED message
Engineer             :      TTZ740
Date                 :      10/02/00

Parameters           :      None
Returns              :      None
Notes                :
*****************************************************************************/

void ExternalHandler(void)
    {
    EXTERNTIMticks = 0;                           //reset external timeout period
    SystemFlags.bit.enableExternal = 1;
    AZ60.pte.bit.pte4 = 1;                        //switch on external LED
    }

/*****************************************************************************
Function Name        :      MsgHandler4
Engineer             :      TTZ740
Date                 :      10/02/00

Parameters           :      None
Returns              :      None
Notes                :
*****************************************************************************/

void IdentifyHandler(void)
```

```
    {
    SystemFlags.bit.enableTimeout = 0;
    AZ60.ptb.byte = ~nodeID;                        //display node ID on LEDs
    }

/*******************************************************************************
Function Name        :      SleepHandler
Engineer             :      r38917
Date                 :      11/02/00

Parameters           :      None
Returns              :      None
Notes                :
*******************************************************************************/

void SleepHandler(void)
    {
    AZ60.ddrf.byte = DDRF3|DDRF4;       //enable PTF pins as output
    AZ60.pte.bit.pte3 = 0;              //disable UPL interface

    SetMC33388Mode(NORMAL);             //MC33388 can't be put to sleep from Vbat
                                        //standby mode
    SetMC33388Mode(SLEEP_RQ);

    SetMC33388Mode(SLEEP);

    while(1)
        ;                               //placing MC33388 into SLEEP mode switches off the Vreg
    }

/*******************************************************************************
Function Name        :      DefaultHandler
Engineer             :      r38917
Date                 :      11/02/00

Parameters           :      None
Returns              :      None
Notes                :
*******************************************************************************/

void DefaultHandler(void)
    {
    }

/*******************************************************************************
Message handler vector table
*******************************************************************************/

void (* const MsgHandlerTable[])() =
    {
    DefaultHandler,
    RotatingHandler,
    BroadcastHandler,
    ExternalHandler,
    IdentifyHandler,
    DefaultHandler,
    DefaultHandler,
    DefaultHandler,
    SleepHandler,
    DefaultHandler,
    DefaultHandler,
    DefaultHandler,
    DefaultHandler,
    DefaultHandler,
```

```
    DefaultHandler,
    DefaultHandler
    };

/*******************************************************************************
Task Name           :      LINInitialise
Engineer            :      TTZ740
Date                :

Parameters          :      none
Returns             :      none
Notes               :      LIN driver timer setup causes output compare pins to toggle
                    :      which creates a conflict with the hardware design
*******************************************************************************/

void LINInitialise(void)
    {
    LIN_Init();                              //initialise LIN driver

    TIMER.tasc0.byte = TIMER.tasc0.byte & 0xF3;
    TIMER.tasc1.byte = TIMER.tasc1.byte & 0xF3;
    }

/*******************************************************************************
Task Name           :      LINWakeup
Engineer            :      TTZ740
Date                :

Parameters:  none
Returns             :      none
Notes               :      Initial hardware design used the UPL interface, which is
                    :      not compatible with the LIN protocol for bus wakeup, hence
                    :      a custom wakeup routine is required for the UPL interface
*******************************************************************************/

void LINWakeup(void)
    {
    SI.sci.scc1.bit.ensci = 0;               // disable SCI
    SI.sci.scbr.bit.scr = 1;                 // 10400 baud
    SI.sci.scbr.bit.scp = 1;                 // 10400 baud
    SI.sci.scc1.bit.ensci = 1;               // enable SCI
    SI.sci.scc2.bit.te = 1;                  // enable transmit


        while (SI.sci.scs1.bit.scte == 0)
             ;

        SI.sci.scdr = 0xAA;                  // send wake up

        while (SI.sci.scs1.bit.tc == 0)
             ;


    LINInitialise();                         //initialise LIN driver
    }

/*******************************************************************************
Task Name           :      initialise
Engineer            :      TTZ740
Date                :

Parameters          :      none
Returns             :      none
```

```
Notes                :
*****************************************************************************/

void initialise (void)
{
    tU08 i = 0;

    SIM.config2.byte = 0x11;                //disable CAN module, enable AZ mode
    KBD.config1.byte = 0x71;                //disable COP

    AZ60.ptb.byte = ALL_LEDS_OFF;           //port b LEDs switch off
    AZ60.ddrb.byte = 0xFF;                  //set port b to output

    SystemFlags.byte = 0;                   //reset the system flags

    LINInitialise();

    TIMER.tsc.byte = 0x54;                  //enable ovf interrupt, rst counter and /16 prescaler
    TIMER.tmod.word = 0x003E;               //1mS overflow based on 4MHz xtal

    AZ60.pte.byte = PTE3;                   //enable UPL device, switch off external LED
    AZ60.ddre.byte = DDRE3|DDRE4;

    for (i = 0; i < 0xFF; ++i)
        ;                                   //delay for UPL to switch on
}


/*****************************************************************************
Task Name            :      main
Engineer             :      TTZ740
Date                 :

Parameters           :      none
Returns              :      none
Notes                :
*****************************************************************************/

void main( void )
    {
    initialise();                                 //initialisation routine

    asm cli;                                      //enable global interrupts

    while( 1 )
    {
    MsgSent[0] = nodeID;                          //data byte1 to be sent is node ID
    MsgSent[1] = AZ60.ptd.byte;                   //Data byte1 to be sent is switch status
        LIN_PutMsg(MESSAGESEND, MsgSent);         //send data to data buffer

        if  (LIN_MsgStatus(MESSAGESEND) != LIN_MSG_NOCHANGE)
            {
            WakeupTimeout = 0;                    //reset wakeup timeout
            }

        if  (LIN_MsgStatus(MESSAGERECEIVE) == LIN_OK)//if new message
            {
            LIN_GetMsg(MESSAGERECEIVE, MsgRcvd);   //read the message
            MsgHandlerTable[(MsgRcvd[0] & 0x0F)]();//call subroutine for appropriate command
            WakeupTimeout = 0;                    //reset wakeup timeout
            }

        if  (LIN_MsgStatus(BROADCASTRECEIVE) == LIN_OK)//if new broadcast message
            {
```

```
            LIN_GetMsg(BROADCASTRECEIVE, MsgRcvd); //read the message
            MsgHandlerTable[(MsgRcvd[0] & 0x0F)](); //call subroutine for appropriate command
            WakeupTimeout = 0;                     //reset wakeup timeout
            }
        }
    }

/*******************************************************************************
Task Name           :      PIT_ISR
Engineer            :      TTZ740
Date                :

Parameters          :      none
Returns             :      none
Notes               :
*******************************************************************************/

#pragma TRAP_PROC

void PIT_ISR(void)
    {

    if(SystemFlags.bit.enableTimeout)
        {
        if(REDTIMticks < REDTIMPeriod)             //check if timeout period for red LEDs has expired
            {
            REDTIMticks++;                         //increment ticks
            }
        else
            {
            AZ60.ptb.byte |= RED_LED_MASK;         //switch off red LEDs
            }

        if(GREENTIMticks < GREENTIMPeriod)         //check if timeout period for green LEDs has expired
            {
            GREENTIMticks++;                       //increment ticks
            }
        else
            {
            AZ60.ptb.byte |= GREEN_LED_MASK;       //switch off green LEDs
            }

        if(AZ60.ptb.byte == ALL_LEDS_OFF)
            SystemFlags.bit.enableTimeout = 0;
        }

    if(SystemFlags.bit.enableExternal)
        {
        if(EXTERNTIMticks < EXTERNTIMPeriod)       //check if timeout period for external LEDs has expired
            {
            EXTERNTIMticks++;                      //increment ticks
            }
        else
            {
            AZ60.pte.bit.pte4 = 0;                 //switch off external LED
            }

        if(!AZ60.pte.bit.pte4)
            SystemFlags.bit.enableExternal = 0;
        }

        if (!SystemFlags.bit.disableWaketime)
            {
```

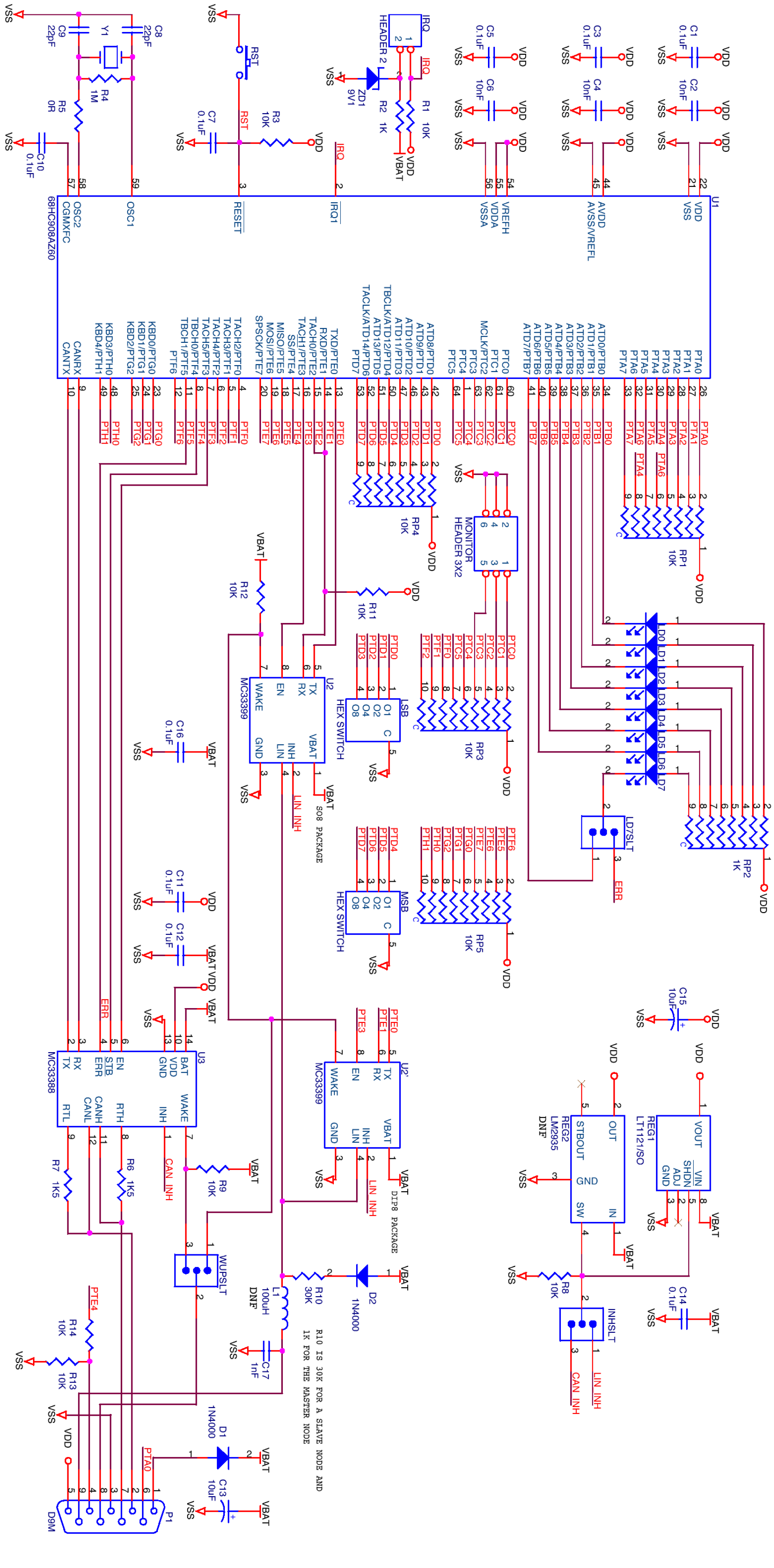
```
            if(WakeupTimeout < TXWAKEUPMSG)
                    WakeupTimeout++;
            else

                    {
                    static tU08retry = 0;
                    if (retry < 3)
                           {
                                 WakeupTimeout = 0;//reset wakeup timeout
                                 LINWakeup();      //attempt to wake up network
                                 retry++;
                           }
                    else
                    SystemFlags.bit.disableWaketime = 1;
                    }
            }

        TIMER.tsc.bit.tof = 0;               //clear the overflow flag
    }
```

5.4 Slave Code – SLAVE08.H

```
/*******************************************************************************
                                                    Copyright (c)
File Name           :      SLAVE08.C

Engineer            :      TTZ740

Location            :      EKB

Date Created        :      07/02/2000

Current Revision    :      $Revision:1.0 $

Notes               :      LIN driver header file

*******************************************************************************
Freescale reserves the right to make changes without further notice to any
Product herein to improve reliability, function or design. Freescale 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. Freescale 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  Freescale product
could create a situation where personal injury or death may occur. Should
Buyer purchase or use Freescale products for any such unintended or
unauthorized application, Buyer shall idemnify and hold Freescale 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 ordeath
associated with such unintended or unauthorized use, even if such claim alleges
that Freescale was negligent regarding the design or manufacture of the part.
Freescale and the Freescale logo* are registered trademarks of Freescale Inc.
*************************************************************************************/
#ifndef  SLAVE08_H
#define SLAVE08_H


 /************************ #Defines ******************************************/
/* standard defs that may be defined by compiler /
```

```
#if !defined(TRUE)
#define TRUE         0x01
#define FALSE        0x00
#endif

#define ON           TRUE
#define OFF          FALSE
#define YES          TRUE
#define NO           FALSE

#define RED_LED_MASK        0x0F
#define GREEN_LED_MASK      0xF0
#define RED_LEDS_ON         0xF0
#define GREEN_LEDS_ON       0x0F
#define ALL_LEDS_OFF        0xFF
#define ALL_LEDS_ON         0x00

#define REDTIMPeriod        150             //generates 150mS timeout (4MHz xtal)
#define GREENTIMPeriod      150             //generates 150mS timeout (4MHz xtal)
#define EXTERNTIMPeriod     150             //generates 150mS timeout (4MHz xtal)
#define TXWAKEUPMSG         200

/************************* Prototypes ****************************************/

#endif  /* End of Header file ifndef   */
```

6 Schematic



Freescale Semiconductor, Inc.

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

**For More Information On This Product,
Go to: www.freescale.com**