INTRODUCTION

Like most network protocols, the Local Interconnect Network (LIN) as described in the official specification is a multi-layered system. The levels vary from the physical interface up to the high level application, with logical data connections between nodes at various layers. This application note focuses on the implementation of an interface between the physical interface and higher level application firmware, essentially a hardware driver (the shaded blocks in Figure 1). Specifically, this document presents a Master node driver that is designed for PIC18 microcontrollers with a standard USART module.

OVERVIEW OF THE DRIVER

There are five functions found in the associated example firmware that control the operation of the LIN interface:
- The LIN Transmit/Receive Daemon
- LIN Timekeeper
- LIN Transmit
- LIN Receive
- Hardware Initialization

The Transmit/Receive Daemon

The USART module is the key element used for LIN communications. Using the USART module as the serial engine for LIN has certain advantages. One particular advantage is that it puts serial control in the hardware rather than in software; thus, miscellaneous processing can be performed while data is being transmitted or received. With this in mind, the Master Node LIN Protocol Driver is designed to run in the background, basically as a “daemon”. The user needs only to initiate the daemon through the transmit or receive functions.

The daemon is interrupt driven via the USART receive interrupt. Because of the physical feedback nature of the LIN bus (Figure 2), a USART receive interrupt will occur regardless of transmit or receive operations. Bit flags are used to retain information about various states within the daemon between interrupts. In addition, status flags are maintained to indicate errors during transmit or receive operations.

This application note provides a high level view of how the LIN driver is implemented, as well as examples of the actual code. Those who are interested in getting started right away may refer to “Setting Up and Using the Firmware” (page 8) on how to create their own software project.

It is assumed that the reader is familiar with the LIN specification. Therefore, not all of the details about LIN are discussed. Refer to the references listed at the end of this document for additional information.

Users interested in the implementation of LIN Slave nodes (not discussed in this document) are encouraged to visit the Microchip web site (www.microchip.com) for additional application notes and other information.
STATES AND STATE FLAGS

The LIN daemon uses state flags to remember where it is between interrupts. When an interrupt occurs, the daemon uses these flags to decide what is the next unexecuted state, then jumps to that state. Figure 3 and Figure 4 outline the program flow through the different states, which are listed and defined below.

STATUS AND ERROR FLAGS

Within various states, status flags may be set depending on certain conditions. For example, if the transmitted break is not received as a break within the read back state, then a bit error is indicated through a status flag. Unlike state flags, status flags are not reset automatically. Status flags are left for the LIN system designer to act upon within the higher levels of the firmware.

FIGURE 3: LIN HEADER FLOW CHART
COUNT, ID, AND MESSAGE

The daemon requires a data count, an identifier byte, and a pointer to a message area to function properly. The checksum and parity are automatically calculated; however, the data count is not. Although the specification defines the message size for most of the IDs, the Extended Frame ID is not defined. The data count of this ID is left for the user to define.

The LIN Timekeeper Function

The LIN specification dictates maximum frame times and bus IDLE times. For this reason, a timekeeping function is implemented. This function works together with the daemon and the transmit and receive functions. Essentially, the daemon and the transmit and receive functions update the appropriate time, bus and frame time when called. Figure 3 and Figure 4 show where the timers are updated.

Although the timekeeping function is implemented, the timing base is not, since there are numerous ways of generating a time-base on a PIC18 microcontroller. This is left for the LIN system designer. The example firmware for this application note uses Timer0 to generate a time-base.
Transmit and Receive Functions

Although the transmit and receive functions are called separately, they are very nearly the same function. They differ only by one state flag. These functions basically initiate the first state for either a LIN frame transmit or receive operation. Once initiated, the daemon takes control via a receive interrupt. The program flow is outlined in Figure 5.

Hardware Initialization Function

An initialization function is provided to configure USART operation. The state and status flags are also cleared. Flags related to hardware interrupts and timers are not modified.

FIGURE 5: TRANSMIT AND RECEIVE FUNCTION FLOW
IMPLEMENTING THE DRIVER

The core of the firmware is written in an assembly module to provide good execution performance and use less program memory. However, the examples provided in this section use the C file definitions, with the core being linked into a C programming environment. Both the assembly and C include files that are provided with the example firmware.

Setup and Initialization

Before attempting to execute the LIN firmware, the related registers and hardware must be initialized. The l_init_hw function is provided for this reason. Its three key tasks are:

- Initialize the daemon (starts the LIN driver)
- Initialize registers (sets known values)
- Set up a timer (sets and starts a time-base)

This function has one static parameter: l_bit_rate. The bit rate value for PIC18 devices is calculated using the baud rate equation for standard USARTs:

$$B = \frac{Fosc}{16(X + 1)}$$

where B is the bit rate in bits per second, X is the value of the SPBRG register, and Fosc is the clock frequency (in Hz).

The initialization function also acts as a RESET. Thus, executing this function will clear all errors, including errors related to the USART.

EXAMPLE 1: SETUP EXAMPLE

```c
void main() {
    l_bit_rate = 25; // Start lin_d at
    l_init_hw(); // 9600 @ 4MHz
    l_data_count = 1; // Init some
    l_data = DUMMY; // registers
    l_id = 0;
    T0CON = 0xC0; // Enable timer0
    INTCONbits.TMR0IF = 0;
    INTCONbits.TMR0IE = 1;
    //PIE1bits.RCIE = 1; // Optional for
    //INTCONbits.PEIE = 1; // interrupt
    // driven driver
    INTCONbits.GIEH = 1; // Enable
    // interrupts
    while(1) { // Main program
    }
}
```

Setting Up Timing

The LIN specification sets limits on the frame time and the maximum bus IDLE time. For this reason, a time function, l_time_update, is provided. This function must be called once per bit time. Any time source can be used to perform this operation; the firmware provided with this application note uses Timer0 as the time-base (see Example 3, Example 4 and Example 5).

EXAMPLE 2: BASIC POLLING EXAMPLE

```c
while (1) { // Main loop
    l_txrx_daemon(); // Check for data.
    // Put code
    // to test
    // for finish and
    // errors.
}
```

The most convenient and transparent way to do this, however, is through the USART receive interrupt. Example 3 shows how the driver could be polled by calling the daemon every bit time. Since the daemon checks the RCIF bit before doing anything, calling the l_txrx_daemon function will not cause a problem.

EXAMPLE 3: USART INTERRUPT POLLING EXAMPLE

```c
void InterruptHandlerHigh() {
    if (INTCONbits.TMR0IF
        && INTCONbits.TMR0IE) {
        l_time_update();
        TMR0L = TMR0L + 0x99;
        INTCONbits.TMR0IF = 0;
        l_txrx_daemon(); // Polled driver
    }
}
```
In Example 4, the USART receive interrupt is used to update the LIN daemon. This method is extremely simple, but it does not allow any interbyte space. Some slave nodes may not be able to function well without interbyte space, especially if the bus is saturated with data. Example 5 shows a combined interrupt method to allow for interbyte space. The code in this example inserts one extra bit time between each byte.

**EXAMPLE 4: UPDATE VIA USART INTERRUPT EXAMPLE**

```c
void InterruptHandlerHigh() {
    if (INTCONbits.TMR0IF && INTCONbits.TMR0IE) {
        l_time_update();
        TMR0L = TMR0L + 0x99;
        INTCONbits.TMR0IF = 0;
    }
    if (PIE1bits.RCIE) { // Update
        l_txrx_daemon();
    }
}
```

**EXAMPLE 5: INTERBYTE SPACE EXAMPLE**

```c
void InterruptHandlerHigh() {
    if (INTCONbits.TMR0IF && INTCONbits.TMR0IE) {
        l_time_update();
        TMR0L = TMR0L + 0x6F;
        INTCONbits.TMR0IF = 0;
        if (!PIE1bits.RCIE) {
            l_txrx_daemon(); // Update
            PIE1bits.RCIE = 1; // Enable int
        }
    }
    if (PIE1bits.RCIE && PIR1bits.RCIF) {
        TMR0L = 0x6F; // Sync
        INTCONbits.TMR0IF = 0;
        PIE1bits.RCIE = 0; // Stop int
    }
}
```

### Sending and Receiving Frames

Frames are sent or received by calling `l_tx_frame` or `l_rx_frame`. There are three static parameters that must be passed to either function: `l_id`, `l_data_count`, and `l_data`. Example 6 demonstrates the operation.

The data count and pointer are modified during the operation, so it is important to load these registers before any operation is started. Modifying these during an operation may lead to unexpected results. When the daemon is finished, `l_data` points to the RAM location after the last received or transmitted byte. And the data in register `l_data_count` equals `00h`.

**EXAMPLE 6: TRANSMIT EXAMPLE**

```c
void InterruptHandlerHigh() {
    if (INTCONbits.TMR0IF && INTCONbits.TMR0IE) {
        l_time_update();
        TMR0L = TMR0L + 0x6F;
        INTCONbits.TMR0IF = 0;
        if (!PIE1bits.RCIE) {
            l_txrx_daemon(); // Update
            PIE1bits.RCIE = 1; // Enable int
        }
    }
    if (PIE1bits.RCIE && PIR1bits.RCIF) {
        TMR0L = 0x6F; // Sync
        INTCONbits.TMR0IF = 0;
        PIE1bits.RCIE = 0; // Stop int
    }
}
```

### Handling Error Flags

Error flags are set by the daemon at the time of occurrence. These flags do not affect the operation of the daemon if they are received. It is left up to the LIN system designer to determine how to handle the flags. To catch errors immediately, they must be tested after the daemon has finished each cycle. The code in Example 7 shows an example of how errors can be captured.

**EXAMPLE 7: HANDLING ERRORS**

```c
void InterruptHandlerHigh() {
    if (LIN_ERROR_FLAGS) {
        if (l_error_flags.LE_BIT) {
            // Handle bit error
            INTCONbits.TMR0IF = 0;
        }
        // Handle other errors
        LIN_ERROR_FLAGS = 0; // Clear
    }
}
```

### Using State Flags

State flags dictate where the daemon is in the process of transmitting or receiving data. Thus, it is possible to prematurely terminate transmit and receive operations by simply clearing the state flags. Likewise, it is possible to artificially enter a state by setting certain state flags. This is useful for handling errors and debugging the system.
Globals and Their Definitions

The key core globals and their meanings are described in Table 1 through Table 3, below.

### TABLE 1: LIN FIRMWARE FUNCTIONS

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_txrx_daemon</td>
<td>The background LIN transmit/receive handler. This function can be called from a receive interrupt or polled periodically.</td>
</tr>
<tr>
<td>l_rx_frame</td>
<td>Initiates a receive from the LIN bus.</td>
</tr>
<tr>
<td>l_tx_frame</td>
<td>Initiates a transmit to the LIN bus.</td>
</tr>
<tr>
<td>l_time_update</td>
<td>Updates the frame and bus timers. It should be called once per bit time.</td>
</tr>
<tr>
<td>l_init_hw</td>
<td>Initializes all flags and resets the hardware used by LIN.</td>
</tr>
</tbody>
</table>

### TABLE 2: LIN FIRMWARE REGISTERS

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_id</td>
<td>LIN identifier byte to be transmitted. The parity bits (two Most Significant bits) are pre-calculated before being transmitted.</td>
</tr>
<tr>
<td>l_data_count</td>
<td>Holds the number of bytes to be transmitted. The count will automatically decrease as data is transmitted or received.</td>
</tr>
<tr>
<td>l_data</td>
<td>16-bit pointer to the LIN data in memory. The pointer will automatically increase as data is transmitted or received.</td>
</tr>
<tr>
<td>l_bit_rate</td>
<td>Holds the bit rate of the LIN bus.</td>
</tr>
<tr>
<td>l_state_flags</td>
<td>Flags used to control the state of the LIN daemon.</td>
</tr>
<tr>
<td>l_status_flags</td>
<td>Contains status information.</td>
</tr>
<tr>
<td>l_error_flags</td>
<td>Contains error information.</td>
</tr>
</tbody>
</table>

### TABLE 3: FLAGS DEFINED IN THE FIRMWARE REGISTERS

<table>
<thead>
<tr>
<th>Flag Name</th>
<th>Register</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>L_TXRX</td>
<td>l_state_flags</td>
<td>Indicates transmit or receive operation (state flag).</td>
</tr>
<tr>
<td>L_RBACK</td>
<td>l_state_flags</td>
<td>Indicates a read back is pending (state flag).</td>
</tr>
<tr>
<td>L_BREAK</td>
<td>l_state_flags</td>
<td>Indicates a break has been sent (state flag).</td>
</tr>
<tr>
<td>L_SYNC</td>
<td>l_state_flags</td>
<td>Indicates a sync byte has been sent (state flag).</td>
</tr>
<tr>
<td>L_ID</td>
<td>l_state_flags</td>
<td>Indicates the identifier has been sent (state flag).</td>
</tr>
<tr>
<td>L_DATA</td>
<td>l_state_flags</td>
<td>Indicates all data has been sent or received (state flag).</td>
</tr>
<tr>
<td>L_CHKSM</td>
<td>l_state_flags</td>
<td>Indicates the checksum has been sent or received (state flag).</td>
</tr>
<tr>
<td>L_BUSY</td>
<td>l_status_flags</td>
<td>Indicates a LIN transmit or receive is in progress (state flag). This bit can be polled to determine when a LIN operation has completed.</td>
</tr>
<tr>
<td>L_SLEEP</td>
<td>l_status_flags</td>
<td>Indicates the LIN bus is inactive (state flag). It is up to the LIN system designer to set this flag at the appropriate time.</td>
</tr>
<tr>
<td>L_RWAKE</td>
<td>l_status_flags</td>
<td>Indicates a wake-up has been requested by a slave (status flag).</td>
</tr>
<tr>
<td>LE_BIT</td>
<td>l_error_flags</td>
<td>Indicates a bit error (status flag).</td>
</tr>
<tr>
<td>LE_CHKSM</td>
<td>l_error_flags</td>
<td>Indicates a checksum error during a receive (status flag).</td>
</tr>
<tr>
<td>LE_FTO</td>
<td>l_error_flags</td>
<td>Indicates the frame has exceeded its maximum time (status flag).</td>
</tr>
<tr>
<td>LE_BTO</td>
<td>l_error_flags</td>
<td>Indicates the bus IDLE time limit has been exceeded (status flag). This could be used as an error or a warning to set L_SLEEP.</td>
</tr>
</tbody>
</table>
SETTING UP AND USING THE FIRMWARE

As noted, the code accompanying this application note includes both assembly and C files. The examples in C are targeted for the Microchip PICC 18™ C compiler. Adjustments for other compilers may be necessary.

Setting Up the Project

For the project to build correctly, it is necessary to include all of the required files in the development environment, including header and definition files. A typical project for Microchip’s MPLAB® 32, showing the hierarchical relationship of the necessary files, is shown in Figure 6. All of the required files are included in the Zip archive accompanying this application note.

The key files to include are the lin_d.asm and main.c (or some other entry file) as source files, as well as a linker script appropriate for the microcontroller. The listings for the source files are presented in Appendix B and Appendix A, respectively.

USING THE HEADER FILES

Header files for both PICC 18 and MPASM™ are provided. The header files lin.inc and lin.h contain all the necessary symbols used in the core lin_d.asm module. Either of these should be included in each application module that uses the daemon, lin.inc for MPASM modules and lin.h for PICC 18 modules.

SETTING THE DEFINITIONS

The file lin.def contains all the important definitions for the lin_d.asm file and any other objects that use the state, status, or error flags. For most situations, this file will not need to be edited. Like the include file, this must be included in all assembly modules that use any part of the daemon (i.e., uses LIN flags or functions).

MEMORY USAGE

The core module is 188 words long. It is written entirely in a relative coding scheme and thus, can be placed anywhere in the program memory map, regardless of its assembled location. The code is also written as a module, so it can be easily linked with C source code.

The core module consumes 12 bytes of data memory when active.

REFERENCES


FIGURE 6: PROJECT SETUP (MPLAB 32)

[Diagram showing project setup for MPLAB 32]
Software License Agreement

The software supplied herewith by Microchip Technology Incorporated (the “Company”) for its PICmicro® Microcontroller is intended and supplied to you, the Company’s customer, for use solely and exclusively on Microchip PICmicro Microcontroller products. The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved. Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN ‘AS IS’ CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

APPENDIX A: LIN TEST PROGRAM (main.c)

//****************************************************************************
// LIN Test Program By Ross Fosler
// 04/24/02
//****************************************************************************

#include <p18cxxx.h>
#include "lin_d.h" // Include LIN functions

//****************************************************************************
// Main routine
void main()
{  
    l_bit_rate = 71; // 9600 @ 11.059MHz
    l_init_hw();

    T0CON = 0xC0; // Enable timer0
    INTCONbits.TMR0IF = 0;
    INTCONbits.TMR0IE = 1;
    PIE1bits.RCIE = 1;
    INTCONbits.PEIE = 1;
    INTCONbits.GIEH = 1; // Enable interrupts

    LINDATA[0] = 24;
    LINDATA[1] = 43;
    l_data = LINDATA;
    while(1) 
    {  
        
    
}
if (!PORTDbits.RD1) {
    l_data = LINDATA; // Receive data from slave
    l_data_count = 2;
    l_id = 3;
    l_rx_frame();
    while (!PORTDbits.RD1) {
    }
}
if (!PORTDbits.RD3) {
    l_data = LINDATA; // Transmit data to slave
    l_data_count = 2;
    l_id = 2;
    l_tx_frame();
    while (!PORTDbits.RD3) {
    }
}
}

//*****************************************************************************/
// High priority interrupt vector
#pragma code InterruptVectorHigh = 0x08
void InterruptVectorHigh(void) {
    _asm
        bra InterruptHandlerHigh // jump to interrupt routine
    _endasm
} //*****************************************************************************
// High priority interrupt routine
#pragma code
#pragma interrupt InterruptHandlerHigh
void InterruptHandlerHigh() {
    if (INTCONbits.TMR0IF & INTCONbits.TMR0IE) {
        if (PIR1bits.RCIF) { // Keep a count for interbyte space
            MYCOUNT.Byte++;
        }
        l_time_update();
        TMR0L = TMR0L + 0x71;
        INTCONbits.TMR0IF = 0;
        if (l_status_flags.LE_SLAVE) {
            LIN_STATUS_FLAGS = 0;
            LIN_STATE_FLAGS = 0;
        }
    }
    if (PIE1bits.RCIE) { // check for recv int
        if (MYCOUNT.Bit.EO2) { // Use counter to add interbyte space
            l_txrx_daemon();
            MYCOUNT.Byte = 0;
        }
    }
    if (l_status_flags.LE_TOUT) { // Put code to check flags
        if (!(l_status_flags.L_BUSY) { // Transmit a 'keep alive' packet
            l_data = LINDATA;
            l_data_count = 2;
            l_id = 0;
            l_tx_frame();
            l_status_flags.LE_TOUT = 0;
        }
    }
} //*****************************************************************************
APPENDIX B: LIN CORE FUNCTIONS (lin.d.asm)

;********************************************************************
; Core Functions for a LIN Master Node on a PIC18
; by Ross M. Fosler  04/18/02
;********************************************************************

#include DEVICES.INC
#include lindefs.inc

; *******************************************************************
#include       DEVICES.INC
#include       lindefs.inc
; *******************************************************************

*******************************************************************

.GetHashCode UDATA_ACS

l_readback  LIN_READ_BACK res 1 ; LIN readback compare register
GLOBAL l_readback
GLOBAL LIN_READ_BACK

l_id  LIN_IDENT res 1 ; LIN Identifier
l_data_count  LIN_DATA_COUNT res 1 ; LIN Data count
l_data  LIN_POINTER_L res 1 ; Pointer to the data
LIN_POINTER_H res 1
l_chksum  LIN_CHKSUM res 1 ; LIN checksum
GLOBAL l_id, l_data_count
GLOBAL l_data, l_chksum
GLOBAL LIN_IDENT
GLOBAL LIN_DATA_COUNT
GLOBAL LIN_POINTER_H
GLOBAL LIN_POINTER_L
GLOBAL LIN_CHKSUM

l_state_flags  LIN_STATE_FLAGS res 1 ; Some flags
l_status_flags  LIN_STATUS_FLAGS res 1
GLOBAL l_state_flags
GLOBAL l_status_flags
GLOBAL LIN_STATE_FLAGS
GLOBAL LIN_STATUS_FLAGS

l_bit_rate  LIN_SPBRG res 1 ; LIN bit rate
GLOBAL l_bit_rate
GLOBAL LIN_SPBRG

l_frame_time  LIN_FRAME_TIME res 1
l_bus_time  LIN_BUS_TIME res 1
LIN_BUS_TIME_L res 1
LIN_BUS_TIME_H res 1
GLOBAL l_frame_time
GLOBAL l_bus_time
GLOBAL LIN_FRAME_TIME
GLOBAL LIN_BUS_TIME_L
GLOBAL LIN_BUS_TIME_H

; *******************************************************************
; This is the transmit/receive daemon. This function should be called
; from an interrupt handler function after the USART receive
; interrupt. Alternatively, this function could be called
; periodically.

_LINDAEMON     CODE
l_txrx_daemon
    GLOBAL l_txrx_daemon
    btfss LINIF           ; Do nothing unless data is ready
    return
    movlw high BUS_WARN_TIME ; Update the bus timer
    movwf LIN_BUS_TIME_H
    movlw low BUS_WARN_TIME
    movwf LIN_BUS_TIME_L
    btfsc L_BUSY           ; If not actively doing something
    bra l_test_readback    ; data might be a wakeup request.

; *******************************************************************
l_test_wake
    movf LINRX, W
    andlw b'00111111'
    btfsc STATUS, Z
    bsf L_RWAKE           ; Indicate wakeup has been requested
    return

; *******************************************************************
; *******************************************************************
l_test_readback
    btfss L_RBACK
    bra l_tx_break
    movf LINRX, W          ; Compare the data
    xorwf LIN_READ_BACK, W
    btfss STATUS, Z
    bsf LE_BIT             ; Indicate a bit error
    bcf L_RBACK

; *******************************************************************
; *******************************************************************
l_tx_break
    btfsc L_BREAK          ; Has a break been sent yet?
    bra l_tx_sync
    bcf STATUS, C
    rrcf LIN_SPBRG, W      ; Reset the TX rate to 1.5x
    addwf LINBRG, F
    movlw b'00000000'      ; Send sync break
    movwf LININTX
    movwf LIN_READ_BACK    ; Setup for readback test
    bsf L_RBACK            ; Set the readback flag
    bsf L_BREAK            ; Set the break flag
    return

; *******************************************************************
; *******************************************************************
l_tx_sync
    btfsc L_SYNC
    bra l_tx_id
    movff LIN_SPBRG, LINBRG ; Reset the bit rate
    movlw 0x55
    movwf LININTX          ; Send sync
    movwf LIN_READ_BACK    ; Setup for readback test
    bsf L_RBACK            ; Set the readback flag
    bsf L_SYNC             ; Set the sync flag
    return
; *******************************************************
l_tx_id
    btfs L_ID
    btfsc L_ID
    movlw b'00111111' ; Strip off 2 MSBits
    andwf LIN_IDENT
    swapf LIN_IDENT, W ; Get (ID0 xor ID4), (ID1 xor ID5)
xorf LIN_IDENT, W
    movwf LIN_CHKSUM ; Store in Checksum
    rrncf WREG, F
    rrcf WREG, F
    xorwf LIN_CHKSUM, F ; Get (ID0 xor ID2 xor ID4), (ID1 xor ID3 xor ID5)
    rrcf LIN_IDENT, W ; Get (ID0 ID1 ID2 ID4), (ID1 ID3 ID4 ID5)
xorf LIN_CHKSUM
    bsf LIN_IDENT, 7 ; Set P1
    btfsc LIN_CHKSUM, 3
    bcf LIN_IDENT, 7
    bsf LIN_IDENT, 6 ; Set P0
    btfss LIN_CHKSUM, 0
    bcf LIN_IDENT, 6
    movf LIN_IDENT, W
    movwf LINTX ; Transmit the ID
    movwf LIN_READ_BACK ; Setup for readback test
    clrf LIN_CHKSUM ; Init the checksum
    bsf L_RBACK ; Set the readback flag
    bsf L_ID ; Set the ID flag
    return
; *******************************************************

; *******************************************************
l_txrx_test
    btfs L_TXRX
    btfss L_TXRX
    ; *******************************************************

; *******************************************************
l_tx_msg
    btfs L_DATA
    btfsc L_DATA
    movff LIN_POINTER_H, FSR0H ; Setup pointer to memory
    movff LIN_POINTER_L, FSR0L
    movf INDF0, W ; Get the data
    movwf LINTX ; Send the data
    movwf LIN_READ_BACK ; Setup for readback test
    addwf LIN_CHKSUM, F ; Adjust the checksum
    btfs L_CHKSM ; Send the checksum
    btfsc L_CHKSM
    ; *******************************************************

; *******************************************************
l_tx_chksum
    btfs L_CHKSM
    btfsc L_CHKSM
    comf LIN_CHKSUM, W ; Send the checksum
    movwf LINTX
    movwf LIN_READ_BACK ; Setup for readback test
    bsf L_RBACK ; Set the readback flag
    bsf L_CHKSM
    return
; *******************************************************
l_rx_msg
    btfsc L_DATA            ; Do nothing unless data is ready
    bra l_rx_chksum
    btfss LINIF
    return
    movff LIN_POINTER_H, FSR0H       ; Setup pointer to memory
    movff LIN_POINTER_L, FSR0L
    movf LINRX, W
    movwf INDF0
    addwf LIN_CHKSUM, F       ; Adjust the checksum
    btfsc STATUS, C
    incf LIN_CHKSUM, F
    infsnz LIN_POINTER_L, F   ; Adjust pointer to next byte
    incf LIN_POINTER_H, F
    dcfsnz LIN_DATA_COUNT, F ; Adjust the data count
    bsf L_DATA
    return

l_rx_chksum
    btfsc L_CHKSM
    return
    comf LINRX, W           ; Get the data
    xorwf LIN_CHKSUM, W     ; Test the checksum
    btfss STATUS, Z
    bsf LE_CHKSM
    bsf L_CHKSM

l_cleanup
    clrf LIN_STATE_FLAGS    ; Reset all states
    bcf L_BUSY             ; Clear the busy flag
    return

; These are the transmit and receive routines. Use these functions
; to initiate LIN activity.

l_rx_frame
    GLOBAL l_rx_frame
    btfsc L_BUSY
    return
    bcf L_TXRX
    bra l_txrx

l_tx_frame
    GLOBAL l_tx_frame
    btfsc L_BUSY
    return
    bsf L_TXRX

l_txrx
    ; Setup the frame timer
    rlncf LIN_DATA_COUNT, W ; REG = C x 8
    rlncf WREG, F
    rlncf WREG, F
    andlw b'11111100'
    movwf LIN_FRAME_TIME
    rlncf LIN_DATA_COUNT, W ; REG = (C x 2) + (C x 8)
    bcf WREG, 0
    addwf LIN_FRAME_TIME
    movlw d'44'
    addwf LIN_FRAME_TIME
    movlw d'44'
    rrrncf LIN_FRAME_TIME, W ; REG = REG + REG/4
    rrrncf WREG, F

© 2002 Microchip Technology Inc.
andlw b'11111100'
addwf LIN_FRAME_TIME
rrncf WREG, F ; REG = REG + REG/8
bcf WREG, 7
addwf LIN_FRAME_TIME
bsf L_BUSY ; Indicate LIN bus is now busy
btfss L_SLEEP ; Test for sleep
bra l_tx_break
bcf L_SLEEP
movlw 0x00 ; Send a wakeup
movwf LINTX
movwf LIN_READ_BACK ; Setup for readback test
bsf L_RBACK ; Set the readback flag
return
;
*******************************************************************
*******************************************************************
; The LIN timers are updated here. This function should be called
; once every bit time. The specification requires that bus time
; and frame time are measured.

l_time_update
    GLOBAL l_time_update
    btfss L_BUSY
    bra TestBusTime
    btfs LE_SLAVE ; Do not update if already timed out
    bra TestBusTime
    dcfsnz LIN_FRAME_TIME ; Test the frame timer
    bsf LE_SLAVE
    TestBusTime
    btfs LE_TOUT ; Do not update if already timed out
    return
    movlw 0x01 ; Test the bus timer
    subwf LIN_BUS_TIME_L, F
    btfs STATUS, C
    return
    subwf LIN_BUS_TIME_H, F
    btfs STATUS, C
    return
    bsf LE_TOUT ; Bus time out flag
    return
;
*******************************************************************
*******************************************************************
; Initialize the hardware for LIN.

l_init_hw
    GLOBAL l_init_hw
    clrf LIN_DATA_COUNT ; Reset the message data counter
    clrf LIN_STATUS_FLAGS ; Clear all flags
    clrf LIN_STATE_FLAGS
    bsf LATC, TX ; These are set to prevent glitches
    bsf LATC, RX ; when changing SPBRG on the fly
    bcf TRISC, TX ; Setup transmit pin
    bsf TRISC, RX ; Setup receive pin
    movwf LIN_SPBRG, SPBRG ; Set the bit rate
    clrf TXSTA
    movlw b'00100100' ; Setup transmit
    movwf TXSTA
    clrf RCSTA
    movlw b'10010000' ; Setup receive
    movwf RCSTA
    movf RCREG, W ; Empty the buffer
    movf RCREG, W
    return
;
*******************************************************************
*******************************************************************
APPENDIX C: C HEADER

extern near unsigned char l_readback;  // Identifier byte
extern near unsigned char _id;         // Data count (bytes in the message)
extern unsigned char *l_data;          // Pointer to data
extern near unsigned char l_chksum;    // Checksum calculation area
extern near unsigned char l_data_count; // Data count (bytes in the message)
extern near unsigned char l_bit_rate;  // Desired bit-rate (used w/ USART SPBRG)
extern near unsigned char l_frame_time;
extern near unsigned int l_bus_time;
extern void l_txrx_daemon(void);       // Send a sync break signal
extern void l_rx_frame(void);          // Send a sync signal
extern void l_tx_frame(void);          // Send the ID byte
extern void l_time_update(void);       // Send the calculated checksum
extern void l_init_hw(void);           // Receive and compare to the calculated checksum
extern near unsigned char LIN_STATE_FLAGS;
extern near unsigned char LIN_STATUS_FLAGS;
extern near struct {
    unsigned L_TXRX:1;
    unsigned L_RBACK:1;
    unsigned L_BREAK:1;
    unsigned L_SYNC:1;
    unsigned L_ID:1;
    unsigned L_DATA:1;
    unsigned L_CHKSM:1;
} l_state_flags;
extern near unsigned char LIN_STATUS_FLAGS;
extern near struct {
    unsigned L_BUSY:1;
    unsigned L_SLEEP:1;
    unsigned L_RWAKE:1;
    unsigned LE_BIT:1;
    unsigned LE_CHKSM:1;
    unsigned LE_SLAVE:1;
    unsigned LE_TOUT:1;
} l_status_flags;

APPENDIX D: ASSEMBLY DEFINITIONS

; ********************************************************************************
#define BUS_WARN_TIME d'25000'
define LINRX RCREG
define LINTX TXREG
define LINBRG SPBRG
#define LINIF PIR1,RCIF
#define L_TXRX LIN_STATE_FLAGS,0
#define L_RBACK LIN_STATE_FLAGS,1
#define L_BREAK LIN_STATE_FLAGS,2
#define L_SYNC LIN_STATE_FLAGS,3
#define L_ID LIN_STATE_FLAGS,4
#define L_DATA LIN_STATE_FLAGS,5
#define L_CHKSM LIN_STATE_FLAGS,6
#define L_BUSY LIN_STATE_FLAGS,0
#define L_SLEEP LIN_STATE_FLAGS,1
#define L_RWAKE LIN_STATE_FLAGS,2
#define LE_BIT LIN_STATE_FLAGS,3
#define LE_CHKSM LIN_STATE_FLAGS,4
#define LE_SLAVE LIN_STATE_FLAGS,5
#define LE_TOUT LIN_STATE_FLAGS,6
; ********************************************************************************
APPENDIX E: ASSEMBLY HEADER

EXTERN LIN_READ_BACK ; Readback register for LIN tx functions
EXTERN LIN_IDENT ; LIN Identity
EXTERN LIN_DATA_COUNT ; Number of bytes to be sent or received
EXTERN LIN_POINTER_H ; Pointer to the data array
EXTERN LIN_POINTER_L
EXTERN LIN_CHKSUM ; LIN checksum
EXTERN LIN_STATE_FLAGS ; Flags to determine what state LIN is in
EXTERN LIN_STATUS_FLAGS ; LIN bus status flags
EXTERN LIN_SPBRG ; Bit rate
EXTERN LIN_FRAME_TIME ; Frame timer
EXTERN LIN_BUS_TIME_L ; Bus idle timer
EXTERN LIN_BUS_TIME_H
EXTERN l_txrx_daemon ; tx/rx daemon
EXTERN l_rx_frame ; Receive a frame
EXTERN l_tx_frame ; Transmit a frame
EXTERN l_time_update ; Update the timers
EXTERN l_init_hw ; Init the hardware
APPENDIX F: SOURCE CODE FOR THIS APPLICATION NOTE

The complete source code for the LIN Master Node Driver is available as a single WinZip archive file, which contains all necessary header and include files. It may be downloaded from the Microchip corporate web site at:

www.microchip.com
Note the following details of the code protection feature on PICmicro® MCUs.

- The PICmicro family meets the specifications contained in the Microchip Data Sheet.
- Microchip believes that its family of PICmicro microcontrollers is one of the most secure products of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowl-
  edge, require using the PICmicro microcontroller in a manner outside the operating specifications contained in the data sheet. The person doing so may be engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable”.
- Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our product.

If you have any further questions about this matter, please contact the local sales office nearest to you.

Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip’s products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.