MICROCHIP

AN237

Implementing a LIN Slave Node on a PIC16F73

Author: Ross M. Fosler
Microchip Technology Inc.

INTRODUCTION

This application note presents a LIN slave driver for the PIC16F73 using the standard hardware USART. There are many details to this firmware design; however, this application note focuses mainly on how to setup and use the driver. Therefore, the LIN system designer should be able to get an application running on LIN quickly, without spending a significant amount of time on the details of LIN.

Fortunately, the details are not completely absent. Some information about the firmware design is provided at the end of this document for the curious designer who wants to learn a little more about LIN and this driver implementation.

The information in this application note is presented with the assumption that the reader is familiar with LIN specification v1.2, the most current specification available at the time this document was written. Therefore, not all details about LIN are discussed. Refer to the References section of this document for additional information.

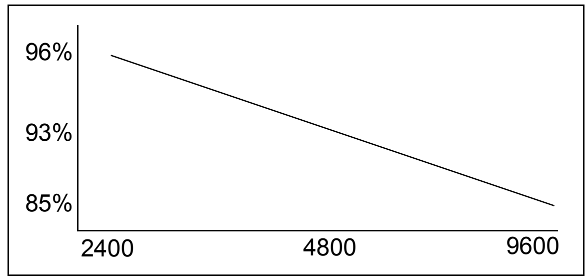
APPLICATIONS

The first question that must be asked is: "Will this driver work for my application?" The next few sections can help those who would like to know the answer to this question and quickly decide whether this is the appropriate driver implementation or device for their application. The important elements that have significant weight on this decision include available process time, resource usage, and bit rate performance.

Process Time

Available process time is dictated predominately by bit rate, clock frequency, and code execution. Fortunately, the driver implementation for the PIC16F73 uses the USART module. This hardware resource puts more processing in hardware and less demand for firmware. Thus, the average available process time is relatively high. Figure 1 shows the approximate average available process time for FOSC equal to 4 MHz.

FIGURE 1: AVAILABLE PROCESS TIME



When the LIN bus is IDLE, the driver uses even less process time, approximately 98% at 4 MHz.

Resource Usage

The resource usage is minimal on the PIC16F73. Only two hardware modules are used. The USART module is used for communications, and the Timer0 module is used for bus and frame timing.

Similarly, the driver consumes only a small portion of the memory resources. The bare driver consumes 5% of program memory of the PIC16F73 and 10% of the available data memory.

Bit Rate

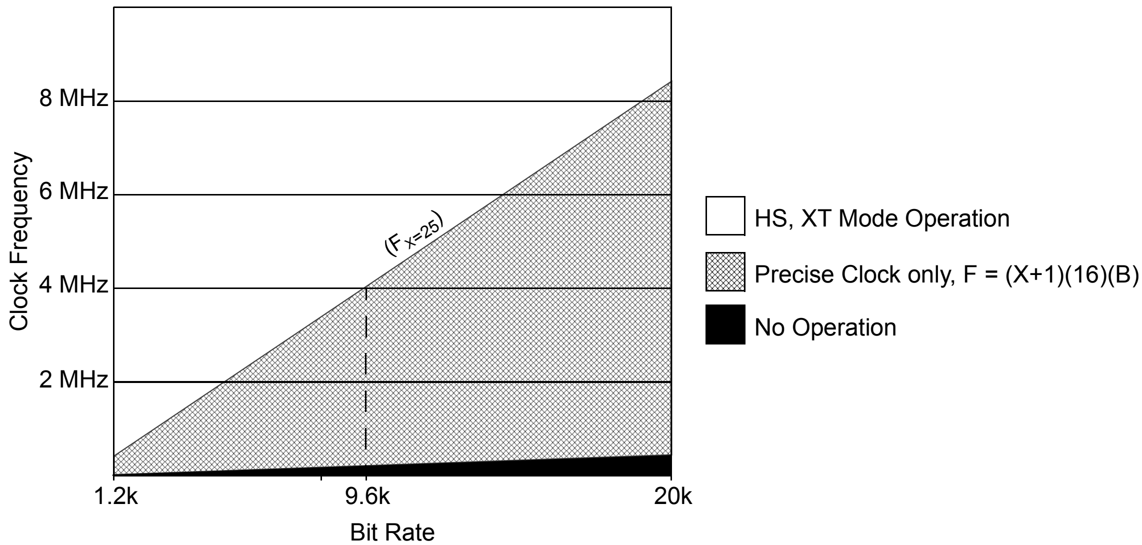
The driver is designed to achieve the maximum bit rate defined by the LIN specification: 20000 bps. However, the oscillator selection must be selected to achieve the application's designed bit rate with a 0.5% tolerance. Figure 2 shows the recommended operating region.

Summary

The LIN Slave driver takes advantage of the USART module to handle most of the otherwise demanding processing, so process time is of little concern. Timer0 is the only other resource, and it interrupts at the bit rate. Therefore, the driver can run virtually transparent in the background without significant interference to the application. This means there is plenty of time for firmware dominant applications. In addition, the PIC16F73 has additional hardware features such as PWM, CCP, A/D, and multiple timers.

Since most of the resources, including process time, are available, this driver is well suited for high demand, high process time applications. Some examples include complex motor controls, instrumentation, multiple feedback applications, and possibly, low to moderate speed engine controls.

FIGURE 2: RECOMMENDED OPERATING REGIONS



SETTING UP THE DRIVER

Now that the decision has been made to use this driver, it is time to set up the firmware and start building an application. For an example, a complete application provided in the appendixes, is built together with the LIN driver. The code provided is actually a simple, yet functional application, demonstrating controlling a motor driven mirror.

Here are the basic steps required to set up your project:

1. Set up a project in MPLAB® IDE. Make sure you have the important driver files included in your project:
   `lin.asm`, `timer.asm`, and `linevent.asm`.
2. Include a main entry point in your project, `main.asm`. Edit this file as required for the application. Make sure that the interrupt is setup correctly, and initialize the driver. Also, ensure any external symbols are included.
3. Edit `linevent.asm` to respond to the appropriate IDs. This could be a table or some simple compare logic. Be certain to include any externally defined symbols.
4. Add any additional application related modules. The example uses `idxx.asm` for application related functions related to specific IDs.
5. Edit the `lin.def` file to setup the compile time definitions of the driver. The definitions determine how the driver functions.

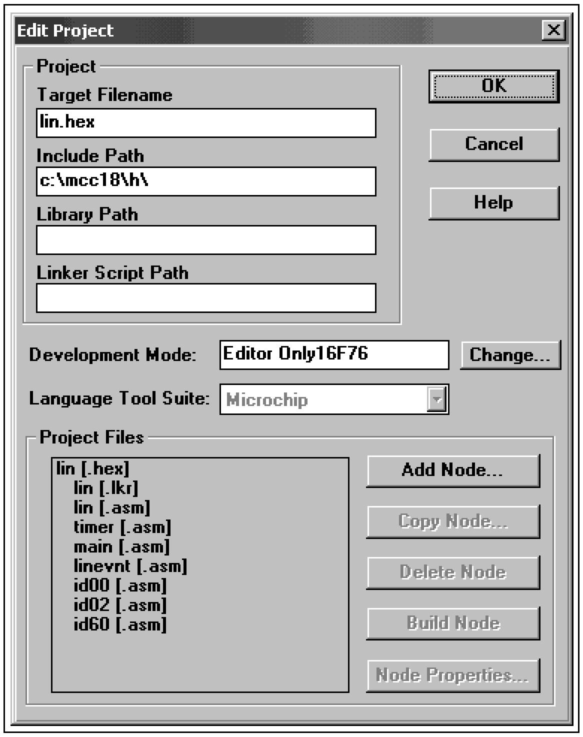
The Project

The first step is to setup the project in MPLAB IDE. Figure 3 shows an example of what the project setup should look like. The following files are required for the LIN driver to operate:

- `lin.lkr` - linker script file
- `main.asm` - the main entry point into the program
- `timer.asm` - Timer0 control
- `lin.asm` - the LIN driver
- `linevent.asm` - LIN event handling table

Any additional files are defined by the system designer for the specific application. For example, Figure 3 lists these project files as `idxx.asm`, where `xx` represents the LIN ID number. This is simply a programming style that separates ID handling into individual objects, thus, making the project format easier to understand. Other objects could be added and executed through the main module, `main.asm` and the event handler.

FIGURE 3: PROJECT SETUP



The Main Object

The `main.asm` module contains the entry point into the program, which is where the driver, hardware, and variables should be initialized. To initialize the driver, call the `l_init_hw` function (refer to Appendix B for an example).

Within the main object is the interrupt vector. This is where the driver function, `l_txrx_driver`, must be called as shown in Example 1. Within the function, the interrupt flag for the USART module is automatically checked.

The timer function is also placed in the interrupt. The example firmware uses Timer0 for bit timing; however, the LIN designer can choose any timer and write the appropriate code. Again, Example 1 shows the placement within the interrupt. Refer to Appendix B for details about the `UpdateTimer` function.

EXAMPLE 1: INTERRUPT VECTOR CODE EXAMPLE

```
_INTERRUPT_V   CODE 0x0004
     movwf     W_TEMP              ; Save important registers
     swapf     STATUS, W
     clrf      STATUS
     movwf     STATUS_TEMP
     movf      FSR, W
     movwf     FSR_TEMP

     call      UpdateTimer         ; Update time
     call      l_txrx_driver       ; Check for any incoming data

     movf      FSR_TEMP, W         ; Restore important registers
     movwf     FSR
     swapf     STATUS_TEMP, W
     movwf     STATUS
     swapf     W_TEMP, F
     swapf     W_TEMP, W
     retfie
```


Definitions

There are a few compile time definitions, all of them located in `lin.def`, that are used to setup the system. Table 1 lists and describes these definitions. The definitions are also listed in Appendix A.

TABLE 1: COMPILE TIME DEFINITIONS

Definition Name	Value	Description
FOSC	d'4000000'	This value is the frequency of the oscillator source.
BIT_RATE	d'9600'	This value is the bit rate for the slave node.
MAX_IDLE_TIME	d'25000'	This value is the maximum IDLE bus time. The LIN specification defines this to be 25000.
MAX_HEADER_TIME	d'39'	This value is the maximum allowable header time. The specification defines this to be 49; however, timing doesn't start until after the first byte (break), so it is actually 39 (10 less than the definition).
MAX_TIME_OUT	d'128'	This is the maximum time allowed to wait after the wake-up request has been made.

LIN Events

LIN event functions are where the ID is decoded to determine what to do next, transmit, receive, and how much. The designer should edit or modify the event function to handle specific LIN IDs (refer to Appendix B, for an example). One possibility is to set up a jump table, which is useful for applications that require responding to multiple IDs. Another option is to setup some simple compare logic.

ID Modules

The application firmware must be developed somewhere in the project. The firmware can be in main or in separate modules; however, from a functional perspective, it does not matter. The example firmware uses separate ID modules for individual handling of IDs and their associated functions. The most important part to remember is to include all of the external symbols that are used. The symbols used by the driver are in `lin.inc`, which should be included in every application module.

The modules that are setup in the example have two parts. One part is the handler for the ID Event. This small function is used to setup the driver to handle the data. Any other functions are part of the application.

USING THE DRIVER

After setting up a project with the LIN driver's necessary files, it is time to start using the driver. This section presents pertinent information about using the driver. The important information addressed is:

- Handling finish flags
- Handling error flags
- State flags within the driver
- LIN ID events
- Bus wake-up

The source code provided is a simple yet nice example on using the LIN driver in an application.

Finish Flags

There are two flags that indicate when the driver has successfully transmitted or received data. The receive flag is set when data has been received without error. This flag must be cleared by the user after it is handled. Likewise, the transmit flag indicates when data has been successfully transmitted without error. The transmit flag must also be cleared by the user after it is handled. Refer to Appendix A for the list of flags and their definitions.

Error Flags

Certain error flags are set when expected conditions are not met. For example, if the slave failed to generate bit timing within the defined range, a sync error flag will get set in the driver.

Errors are considered fatal until they are handled and cleared. Thus, if the error is never cleared, then the driver will ignore incoming data.

The following code, shown in Example 2, demonstrates how to handle errors within the main program loop. This example only shows a response to a bus time-out error. This same concept can be applied to other types of errors.

EXAMPLE 2: ERROR HANDLING

```
.
.
movf  LIN_STATUS_FLAGS, W; Any errors?
btfsc STATUS, Z
goto  Main

btfsc LE_BTO         ; Was the
goto  PutToSleep     ; bus time exceeded?

clrf  LIN_STATUS_FLAGS   ; Reset any
goto  Main               ; errors
```

Notice that the errors are all contained within a single register. So the `LIN_STATUS_FLAGS` register can be checked for zero to determine if any errors did occur.

Driver State Flags

The LIN driver uses state flags to remember where it is between received bytes. After a byte is received, the driver uses these flags to decide what is the next unexecuted state, then jumps to that state. One very useful flag is the `LS_BUSY` flag. This bit indicates when the driver is active on the bus, so this flag could be used in applications that synchronize to the communications on the bus. The other flags indicate what has been received and what state the bus is in. Refer to Appendix A for descriptions of the state flags.

ID Events and Functions

For each ID there is an event function. The event function is required to tell the driver how to respond to the data following the ID. For example, does the driver need to prepare to receive or transmit data. Also, how much data is expected to be received or transmitted.

For successful operation, three variables must be initialized: a pointer to data memory, frame time, and the count, as shown in Example 3.

EXAMPLE 3: VARIABLE INITIALIZATION

```
movlw   ID00_BUFF    ; Set the pointer
movwf   LIN_POINTER

movlw   d'43'  ; Adjust the frame time
addwf   FRAME_TIME, F
movlw   0x02   ; Setup the data count
movwf   LIN_COUNT

retlw   0x00   ; Read
```

The pointer to memory tells the driver where to store data or where to retrieve data. The frame time is the adjusted time based on the amount of bytes to expect. Typically, the frame time register will already have time left over from the header, so time should be added to the register. For two bytes this would be an additional (30 + 1) * 1.4 bit times, or 43; the value 30 is the total bits of data, START bits, and STOP bits plus the checksum bits. The counter simply tells the driver how much data to operate on. Note that the count must always be initialized to something greater than zero for the driver to function properly.

Waking the Bus

A LIN bus wake-up function, `l_tx_wakeup`, is provided for applications that need the ability to wake the bus up. Calling this function will broadcast the wake-up request character.

IMPLEMENTATION

There are four functions found in the associated example firmware that control the operation of the LIN interface:

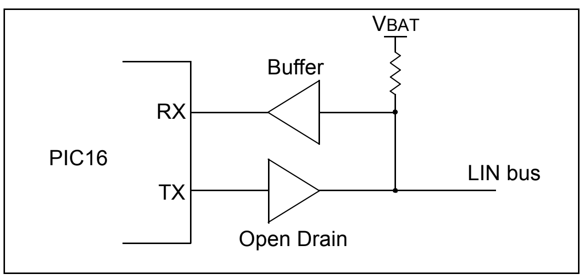
- LIN Transmit/Receive Driver
- LIN Timekeeper
- LIN Hardware Initialization
- LIN Wake-up

The Driver

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 it puts serial control in the hardware, rather than in the software. Thus, miscellaneous processing can be performed while data is being transmitted or received. With this in mind, the Slave Node LIN Protocol Driver is designed to run in the background, basically as a daemon.

The driver is interrupt driven via the USART receive interrupt. Because of the physical feedback nature of the LIN bus (Figure 4), a USART receive interrupt will occur regardless of transmit or receive operations. Bit flags are used to retain information about various states within the driver between interrupts. In addition, status flags are maintained to indicate errors during transmit or receive operations.

FIGURE 4: SIMPLIFIED LIN TRANSCEIVER



STATES AND STATE FLAGS

The LIN driver uses state flags to remember where it is between interrupts. When an interrupt occurs, the driver uses these flags to decide what is the next unexecuted state, then jumps to that state. Figure 5 and Figure 6 outline the program flow through the different states. The states are listed and defined later in this document.

SYNCHRONIZATION

Synchronization is the second normal state and is handled two different ways. Synchronization can be enabled for poor tolerance clock sources or it can be disabled for clock sources with good precision. If enabled, the break and sync byte are received together, as shown in Figure 5.

TX/RX TABLE

A transmit/receive table is provided to determine how to handle data after the node has successfully received the ID byte. The table returns information to the driver about data size and direction.

STATUS FLAGS

Within various states, status flags may be set depending on certain conditions. For example, if the slave receives a corrupted checksum, then a checksum 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.

LIN Timers

The LIN specification identifies maximum frame times and bus IDLE times. For this reason, a timekeeping function is implemented. The timekeeping function works together with the driver and the transmit and receive functions. Essentially, the driver and the transmit and receive functions update the appropriate time, bus and frame time, when called. Figure 5 and Figure 7 show where the timers are updated.

The timekeeping function is implemented independent of a timing source. All that is required is that the timekeeping function be called at least once per bit time. The example firmware provided (see Appendix B) uses the Timer0 module; however, it is possible to use any other time source. Some examples include using Timer1, Timer2, or even an external time source into an interrupt pin.

Hardware Initialization

An initialization function is provided to set up the necessary hardware settings, basically the USART. Also, the state and status flags are all cleared. Flags related to hardware interrupts and timers are not modified.

Wake-up

The only time the slave can transmit to the bus without a request is when the bus is sleeping. Basically, any slave can transmit a wake-up signal. For this reason, a wake-up function is defined, and it sends a wake-up signal when called.

FIGURE 5: RECEIVE HEADER PROGRAM FLOW

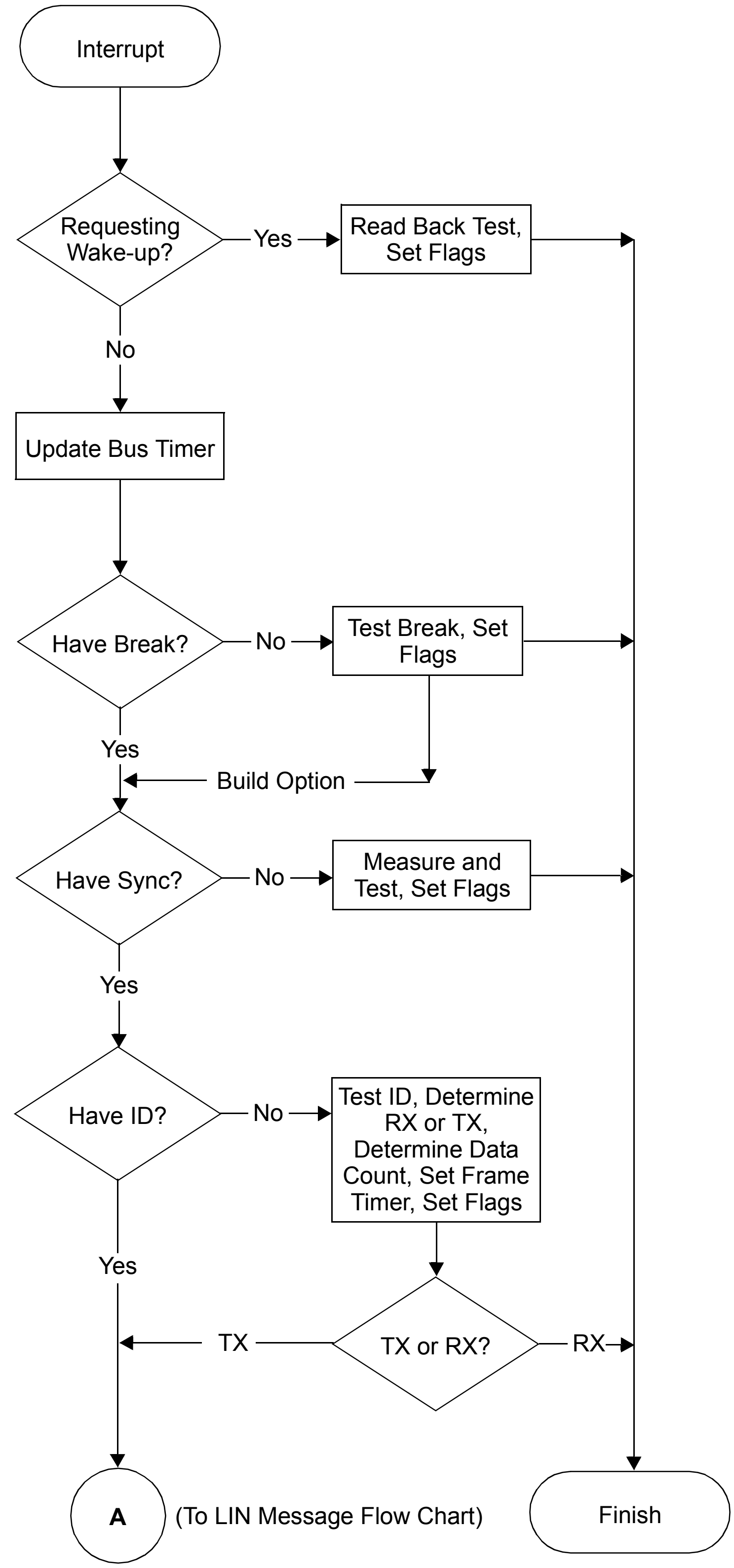


FIGURE 6: TRANSMIT/RECEIVE MESSAGE PROGRAM FLOW

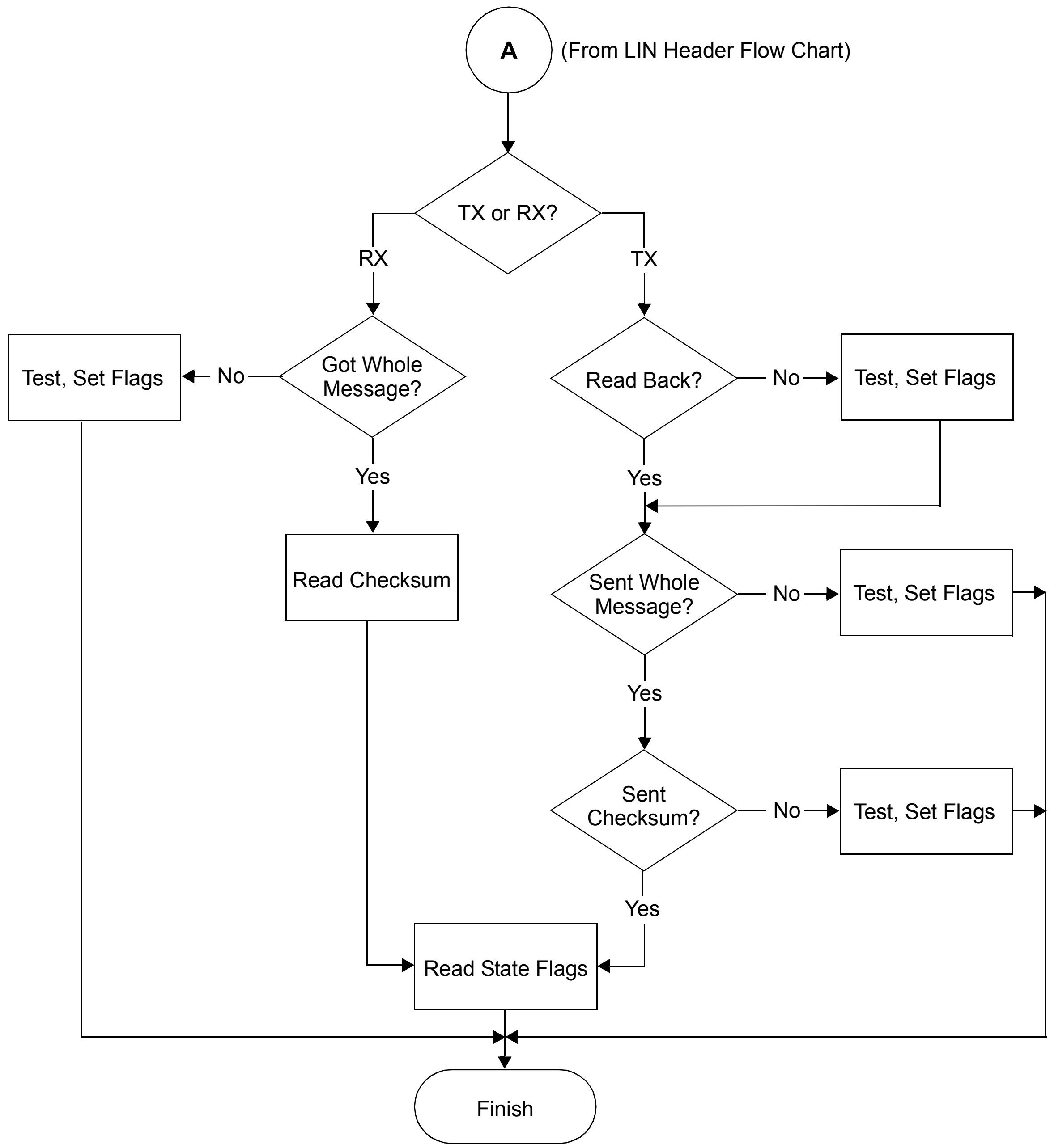
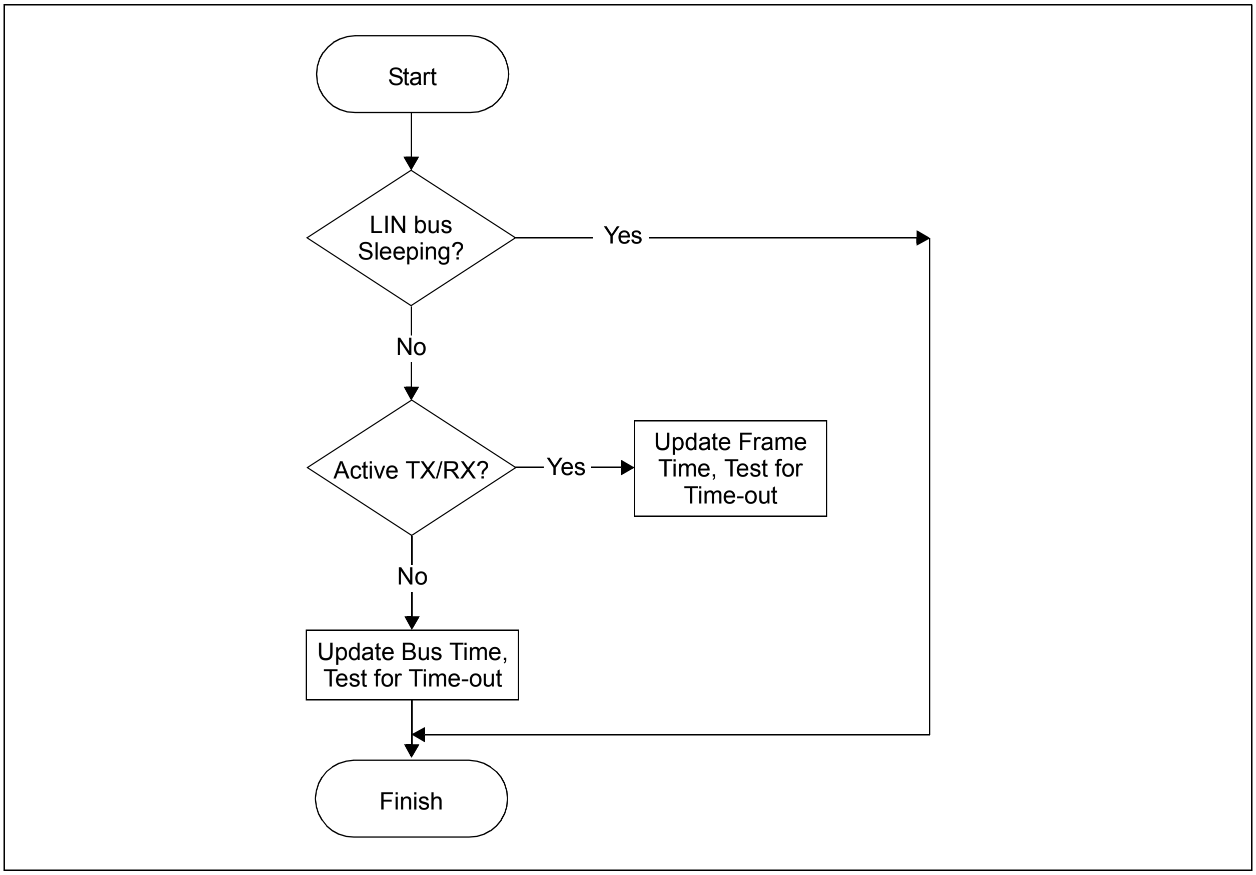


FIGURE 7: TIMEKEEPING PROGRAM FLOW



DETERMINING OPERATING REGION

It is important to understand the relationship between bit rate and clock frequency when designing a slave node in a LIN network. This section focuses on developing this understanding based on the LIN specification. It is assumed that the physical limits defined in the LIN specification are reasonable and accurate; therefore, this section merely uses the defined physical limits and does not present any analysis of the limits defined for the physical interface to the LIN bus. Essentially, the focus of this section is to analyze the firmware and its performance based on the defined conditions in LIN Protocol Specification v1.2.

General Information

Some general information used throughout the analysis is provided here.

DATA RATE VS. SAMPLING RATE

There are essentially two rates to compare, the incoming data rate and the sampling rate. The slave node only has control of the sampling rate. Therefore, for this discussion, the logical choice for a reference is the incoming data rate, B_I. The equations that follow assume B_I is the ideal data rate of the system.

BASE EQUATIONS

The frequency/bit rate relationship of the USART module is defined as:

$$X = \frac{Fosc}{16B} - 1$$

The value X represents the 8-bit value loaded in the SPBRG register. A more useful form of the equation is as follows:

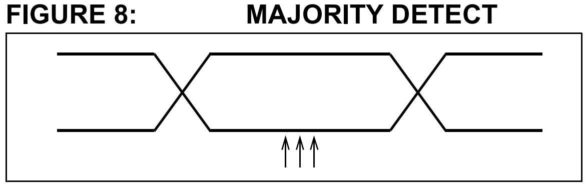
$$B = \frac{Fosc}{16(X+1)}$$

This shows bit rate as a function of frequency and X.

SAMPLING

The USART does a three sample majority detect of the incoming signal, shown in Figure 8. Analytically, this looks like a single sample at the center with some noise immunity and this is assumed in the analysis.

FIGURE 8: MAJORITY DETECT



RELATING CLOCK FREQUENCY ERROR TO BIT ERROR

The LIN Protocol Specification v1.2 refers to clock frequency error rather than bit error. Because of this, technically, the LIN system designer must design the system with like clock sources, which is rather impractical. It is more feasible to have clock sources designed for the individual needs of the node. For this reason, all of the equations in this section refer to bit error rather than frequency error. The following equation relates frequency error to bit rate error.

$$\frac{1}{1+E_F} - 1 = E_B$$

For very low clock frequency errors, the bit rate error can be approximated by:

$$-E_F \approx E_B$$

Thus, a ±2% frequency error is nearly the same bit rate error.

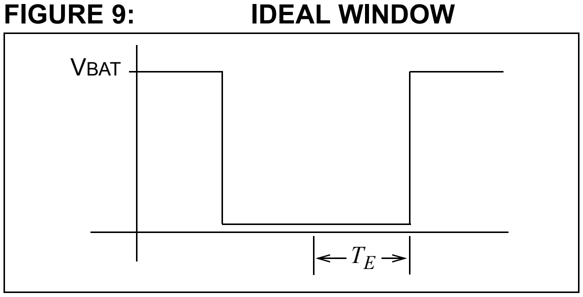
Acceptable Bit Rate Error

The LIN Protocol Specification v1.2 allows for a ±2% error for master - slave communications. This section evaluates this tolerance based on specified worst case conditions (slew rate, voltage, and threshold) and the USART module design.

IDEAL SAMPLING WINDOW

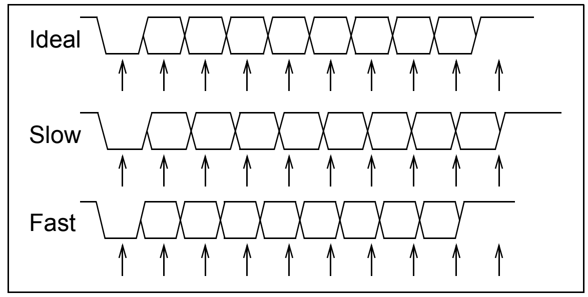
It is relatively easy to see the maximum allowed error in the ideal situation. Ideal is meant by infinite slew rate with a purely symmetrical signal, like the signal shown in Figure 9.

FIGURE 9: IDEAL WINDOW



If the data sampling is greater or less than half of one bit time, T_E, over nine bits, the last bit in one byte will be interpreted incorrectly. Figure 10 depicts how data may be misinterpreted because the incoming bit rate is misaligned with the sampling bit rate.

FIGURE 10: DATA VS. SAMPLING



The two equations that give the maximum and minimum bit rates based on time shifting $T_E = \pm 1/(2B)$ are:

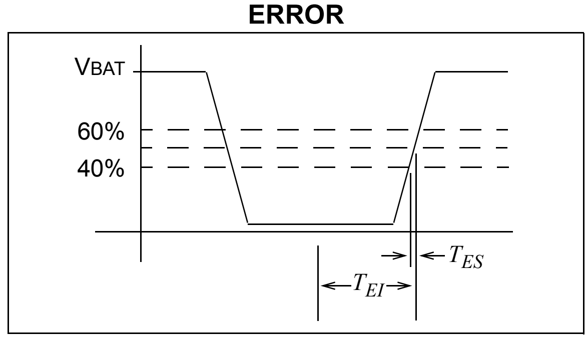
$$\frac{1}{B} - \frac{T_E}{9} = \frac{1}{B_{max}} \quad \text{and} \quad \frac{1}{B} + \frac{T_E}{9} = \frac{1}{B_{min}}$$

SHORTENED WINDOW DUE TO SLEW RATE

Although the ideal sampling window may be a useful approximation at very low bit rates, slew rate and threshold must be accounted for at higher rates. Thus, the ideal analysis serves as a base for more realistic analysis.

The LIN specification defines a tolerable slew rate range and threshold. The worst case is the minimum slew rate at the maximum voltage, 1V/µs and 18V, according to LIN Protocol Specification v1.2. The threshold is above 60% and below 40% for valid data. Figure 11 shows the basic measurements.

FIGURE 11: ADJUSTED BIT TIME ERROR



Taking the difference of the ideal maximum time and the slight adjustment due to specified operating conditions, yields the following equation:

$$T_{EI} - T_{ES} = \frac{1}{2B} - \frac{(0.5V - 0.4V)}{(dV)/(dt)_{min}} = T_E$$

Thus, T_E is slightly smaller than the ideal case. The minimum and maximum equations in the previous section yield slightly narrower range for bit rate.

OFFSET DUE TO SLEW RATE

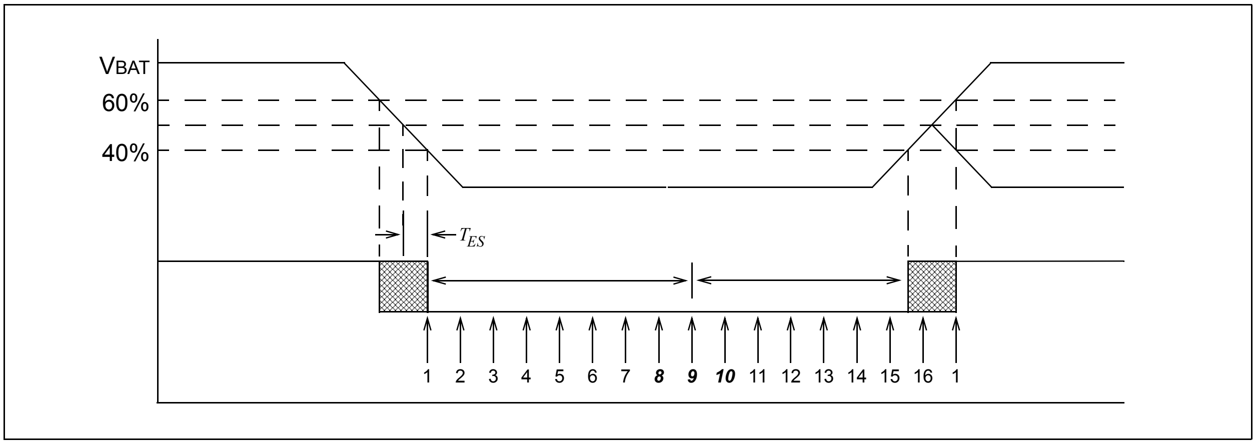
Not only does the slew rate and thresholds contribute to a slightly smaller window, they affect offset of all samples after the first synchronous edge, the START bit.

An offset affects the symmetry of the sampling window rather than the range. Figure 12 shows how this offset favors a negative bit rate error more than a positive bit rate error.

The offset is added to both minimum and maximum equations:

$$\frac{T_{ES}}{9} + \frac{1}{B} + \frac{T_E}{9} = \frac{1}{B_{min}} \quad \text{and} \quad \frac{T_{ES}}{9} + \frac{1}{B} - \frac{T_E}{9} = \frac{1}{B_{max}}$$

FIGURE 12: OFFSET FROM START EDGE DUE TO SLEW RATE & THRESHOLD



OFFSET DUE TO SAMPLING ERROR

Sampling error of the START edge is very similar to the slew rate offset described above. The design of the USART module dictates what the magnitude of this offset is. In this case, the error is simply one cycle of the clock. It is added to the minimum and maximum bit rate equations:

$$\frac{1}{9FOSC} + \frac{T_{ES}}{9} + \frac{1}{B} + \frac{T_E}{9} = \frac{1}{B_{min}}$$

and

$$\frac{1}{9FOSC} + \frac{T_{ES}}{9} + \frac{1}{B} - \frac{T_E}{9} = \frac{1}{B_{max}}$$

OFFSET DUE TO CIRCUIT DELAY

Offsets related to circuit conditions also affect the minimum and maximum error. Since this application note does not describe the physical interface, hardware delays are ignored in this analysis.

Minimum SPBRG Value

Given a finite bit rate error range and finite control of the bit rate, this leads to areas where the slave cannot operate. These are basically gaps where the error is outside the defined bit rate error range for a particular SPBRG value. This section provides the mathematical basis for these gaps. The equations developed in this section are provided to help the LIN designer build a robust network.

FREQUENCY RANGE

The following equation determines the clock frequency as a function of SPBRG, bit rate, and oscillator error.

$$F_{OSC} = (E_B + 1)(X + 1)(16)(B)$$

OVERLAPPING OPERATION

For most SPBRG values, operating range overlaps each other from one SPBRG to the next. Therefore, the slave will communicate with the master for most of the common conditions. Except for a particular error range and some clock frequencies, it is possible to never have a valid SPBRG value.

To approach this problem, the maximum frequency for a particular SPBRG value must be compared to the minimum frequency of the next SPBRG. Where they are equal is the border between continuous and discontinuous operation for any given input frequency.

$$(E_{BH}+1)(X)(16)(B) = (E_{BL}+1)(X+1)(16)(B)$$

Solving this equation yields:

$$X_{low} = \frac{(E_{BL}+1)}{(E_{BH}+1)-(E_{BL}+1)}$$

Therefore, for any given frequency and a defined error, a good SPBRG value will always be above X_{low}. Of course, the frequency and baud rate must be selected such that SPBRG is less than or equal to 255, the largest value supported by SPBRG. For example, for a 2% error, the lowest SPBRG value before certain clock frequencies become a problem is 25. If the theoretical minimum and maximum are used, about ±5% from the previous sections, then a SPBRG value below 10 is a problem. Therefore, for master - slave communications, a SPBRG value above 10 will work. However, to be within the specification, the SPBRG should be above 25.

Summary of Operating Regions

Figure 2 summarizes the various operating regions based on the typical device specifications and information provided. The LIN designer should consult the graph in Figure 2 to find the best operating region for the application.

MEMORY USAGE

The firmware code size depends on the build conditions. As it is currently built, the core module only requires 333 words of program memory and 12 bytes of data memory.

REFERENCES

LIN Protocol Specification v1.2, http://www.lin-subbus.org/

MPASM™ User's Guide with MPLINK™ and MPLIB™, Microchip Technology Incorporated, 1999

MPLAB®-CXX User's Guide, Microchip Technology Incorporated, 2000

APPENDIX A: SYMBOLS AND THEIR DEFINITIONS

TABLE A-1: COMPILE TIME DEFINITIONS

Definition Name	Value	Description
FOSC	d’4000000'	The frequency of the oscillator source.
BIT_RATE	d’9600'	The bit rate for the slave node.
MAX_IDLE_TIME	d’25000'	The maximum IDLE bus time. The LIN specification defines this to be 25000.
MAX_HEADER_TIME	d’39’	The maximum allowable header time. The specification defines this to be 49; however, timing doesn’t start until after the first byte (break), so it is actually 39 (10 less than the definition).
MAX_TIME_OUT	d’128’	The maximum time allowed to wait after the wake-up request has been made.

TABLE A-2: FUNCTIONS

Function Name	Purpose
l_init_hw	Initializes or resets the hardware associated to the LIN interface.
l_txrx_daemon	Core transmit and receive function. This function manages transmit and receive operations to the bus. State flags are set and cleared within this function. Status flags are also set based on certain conditions, i.e., errors.
l_txrx_table	Called by the driver after the identifier byte has been received. Message length and direction is returned to the driver. Within the table, pointers could be setup for different identifies.
l_tx_wakeup	Wake-up function. Call this to wake-up the bus if asleep.
l_update_timers	Used to update the bus and frame timers. This should be called once per bit time.

TABLE A-3: VARIABLES

Variable Name	Purpose
BUS_TIME_H	Most Significant Byte of the bus timer.
BUS_TIME_L	Least Significant Byte of the bus timer.
FRAME_TIME	8-bit frame timer register.
HEADER_TIME	Same as FRAME_TIME.
LIN_COUNT	Used by the driver to maintain a message data count.
LIN_CHKSUM	Used by the driver to calculate checksum for transmit and receive.
LIN_FINISH_FLAGS	Contains flags indicating completion of transmit and receive data.
LIN_ID	Holding register for the received identifier byte. It is used in the l_txrx_table function to determine how the node should react.
LIN_POINTER	Pointer to a storage area used by the driver. Data is either loaded into or read from memory depending on the identifier.
LIN_READBACK	Holding register for transmitted data to be compared with received data for bit error detection.
LIN_STATE_FLAGS	Flags to indicate what state the LIN bus is in.
LIN_STATE_FLAGS2	Additional flags to indicate what state the LIN bus is in.
LIN_STATUS_FLAGS	Contains status information about the LIN bus.

TABLE A-4: FLAGS

Flag Name	Register	Purpose
LE_BIT	LIN_STATUS_FLAGS	Status flag indicating a bit error.
LE_BTO	LIN_STATUS_FLAGS	Status flag indicating a bus activity time-out error.
LE_CHKSM	LIN_STATUS_FLAGS	Status flag indicating a checksum error during a receive.
LE_FTO	LIN_STATUS_FLAGS	Status flag indicating a frame time-out error.
LE_PAR	LIN_STATUS_FLAGS	Status flag indicating a parity error.
LE_SYNC	LIN_STATUS_FLAGS	Status flag indicating a synchronization tolerance error.
LF_RX	LIN_FINISH_FLAGS	Finish flag indicating data has been received.
LF_TX	LIN_FINISH_FLAGS	Finish flag indicating data has been sent.
LS_BRK	LIN_STATE_FLAGS	State flag indicating a break has been received.
LS_BUSY	LIN_STATE_FLAGS	State flag indicating the LIN bus is busy.
LS_CHKSM	LIN_STATE_FLAGS	State flag indicating a checksum error has been sent or received.
LS_DATA	LIN_STATE_FLAGS	State flag indicating all data has been sent or received.
LS_ID	LIN_STATE_FLAGS	State flag indicating the identifier has been received.
LS_RBACK	LIN_STATE_FLAGS	State flag indicating a read back is pending.
LS_SLPNG	LIN_STATE_FLAGS	State flag indicating the LIN bus is sleeping.
LS_SYNC	LIN_STATE_FLAGS	State flag indicating a sync byte has been received.
LS_TXRX	LIN_STATE_FLAGS	State flag indicating a transmit or receive operation.
LS_WAKE	LIN_STATE_FLAGS	State flag indicating a wake-up has been requested (this node only).

APPENDIX B: SOURCE CODE

Due to size considerations, the complete source code for this application note is not included in the text. A complete version of the source code, with all required support files, is available for download as a Zip archive from the Microchip web site, at:

www.microchip.com

NOTES:

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200 Fax: 480-792-7277
Technical Support: 480-792-7627
Web Address: http://www.microchip.com

Rocky Mountain
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7966 Fax: 480-792-4338

Atlanta
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Boston
2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848 Fax: 978-692-3821

Chicago
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423 Fax: 972-818-2924

Detroit
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Kokomo
2767 S. Albright Road
Kokomo, Indiana 46902
Tel: 765-864-8360 Fax: 765-864-8387

Los Angeles
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

Toronto
6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699 Fax: 905-673-6509

ASIA/PACIFIC

Australia
Microchip Technology Australia Pty Ltd
Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733 Fax: 61-2-9868-6755

China - Beijing
Microchip Technology Consulting (Shanghai) Co., Ltd., Beijing Liaison Office
Unit 915
Bei Hai Wan Tai Bldg.
No. 6 Chaoyangmen Beidajie
Beijing, 100027, No. China
Tel: 86-10-85282100 Fax: 86-10-85282104

China - Chengdu
Microchip Technology Consulting (Shanghai) Co., Ltd., Chengdu Liaison Office
Rm. 2401, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-86766200 Fax: 86-28-86766599

China - Fuzhou
Microchip Technology Consulting (Shanghai) Co., Ltd., Fuzhou Liaison Office
Unit 28F, World Trade Plaza
No. 71 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7503506 Fax: 86-591-7503521

China - Shanghai
Microchip Technology Consulting (Shanghai) Co., Ltd.
Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700 Fax: 86-21-6275-5060

China - Shenzhen
Microchip Technology Consulting (Shanghai) Co., Ltd., Shenzhen Liaison Office
Rm. 1315, 13/F, Shenzhen Kerry Centre,
Renminnan Lu
Shenzhen 518001, China
Tel: 86-755-82350361 Fax: 86-755-82366086

China - Hong Kong SAR
Microchip Technology Hongkong Ltd.
Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200 Fax: 852-2401-3431

India
Microchip Technology Inc.
India Liaison Office
Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaugnessey Road
Bangalore, 560 025, India
Tel: 91-80-2290061 Fax: 91-80-2290062

Japan
Microchip Technology Japan K.K.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-882
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Singapore
Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-6334-8870 Fax: 65-6334-8850

Taiwan
Microchip Technology (Barbados) Inc.,
Taiwan Branch
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Austria
Microchip Technology Austria GmbH
Durisolstrasse 2
A-4600 Wels
Austria
Tel: 43-7242-2244-399
Fax: 43-7242-2244-393

Denmark
Microchip Technology Nordic ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France
Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - ler Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany
Microchip Technology GmbH
Steinheilstrasse 10
D-85737 Ismaning, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy
Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

United Kingdom
Microchip Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5869 Fax: 44-118 921-5820

10/18/02