

LIN Slave Node on a PIC16C433

*Author: Ross Fosler
Microchip Technology, Inc.*

INTRODUCTION

The PIC16C433 is a standard PIC16CXXX microcontroller with a LIN (Local Interconnect Network) transceiver integrated into the device. Therefore, the microcontroller already has the necessary hardware to easily integrate the device into a LIN system. This application note provides a firmware base (driver) for the system designer to use on the PIC16C433. The driver utilizes the resources available, including the Timer0 module, Timer0 prescaler, GPIO interrupt-on-change or the external interrupt, and the LIN transceiver. In this document, significant effort is spent demonstrating how to setup and use the driver. Some general information and tips are also discussed to help the designer build their application seamlessly in the LIN environment. In addition, for the curious designer, some additional details about the driver are provided toward the end of the document.

The reader should note 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 initial release of this document. Therefore, not all details about LIN are discussed. Refer to the references listed on page 14 for additional information.

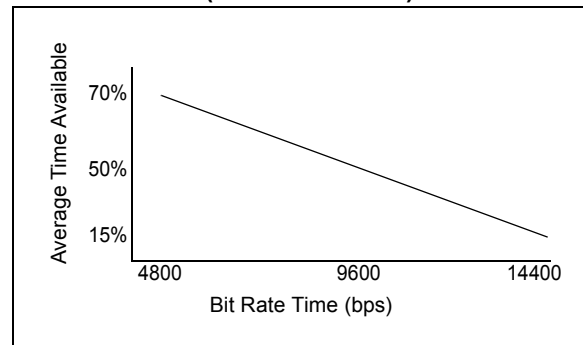
APPLICATIONS

The first question that must be asked is: "Will this driver work for my application?" The next few sections are written to 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 the application. The important elements that have significant weight on the 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. Since code execution varies depending on the state within the LIN driver and there being many states, generating a single equation for process time is unrealistic. A much simpler solution is to test the process time. Figure 1 shows the approximate average available process time for Fosc equal to the nominal internal oscillator frequency, 4 MHz.

FIGURE 1: AVAILABLE PROCESS TIME (Fosc AT 4 MHZ)



When the LIN bus is IDLE, the driver uses significantly less process time. Although dependent on the same conditions stated above, the used process time is extremely low. At 4 MHz, the average available process time is greater than 98%.

Resource Usage

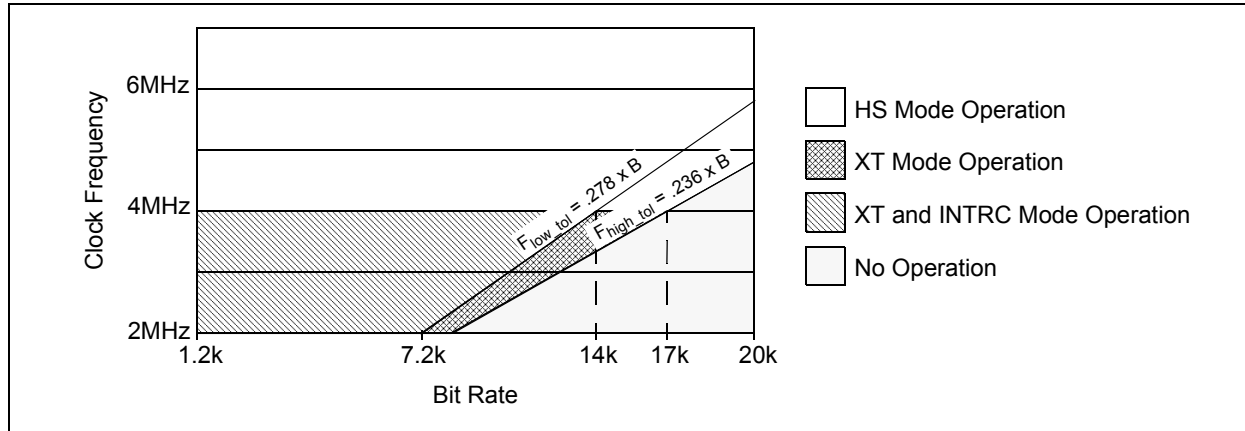
A few of the hardware resources are used to maintain robust communications and precise timing. Timer0 is used for maintaining communications timing and bus activity time. Along with this, the timer prescaler is adjusted under various conditions to simplify the code and improve performance in many states of the driver. An external interrupt is used for START edge detection for each received byte, either the GPIO interrupt-on-change or the INT pin can be configured as the interrupt source.

Regarding memory usage, the driver consumes a small portion of the memory resources. The bare driver only consumes 15% of program memory of the PIC16C433 and 28% of the available data memory.

Bit Rate

The driver can achieve 14.4 Kbps using the internal oscillator calibrated at its nominal operating frequency of 4 MHz. As shown in Figure 2, higher bit rates are also achievable by selecting HS mode and a higher clock frequency. If the clock source meets the high accuracy requirements defined in the LIN specification, then a 15% adjustment can be made. Figure 2 shows the lines dividing the operating regions for low tolerance and high tolerance clock sources.

FIGURE 2: RECOMMENDED OPERATING REGIONS



Summary

The driver is designed almost entirely in firmware. Only the hardware peripherals standard to the PIC16CXXX microcontroller are used. Thus, all communications and timing required by the LIN specification are controlled in firmware. This implies a certain percentage of software resources are consumed, most importantly, time. To put this into perspective, at 9600 bps using the 4 MHz internal oscillator, an average of 50% of the process time is used by the driver. Also, given the intolerance to uncertainty, interrupts should be restricted to the LIN communications driver.

In summary, this means this driver is well suited for applications that require only basic tasks. Thus, this driver may not necessarily be the best choice for complex timing critical applications, especially if the internal oscillator is used. The PIC16C433 has an A/D converter and up to 5 digital I/O pins with the LIN driver active. Thus, some very useful applications include simple motion control, on/off control, and sensor feedback. For more complex applications, refer to other Microchip LIN driver implementations that use Microchip Microcontrollers with a hardware USART, such as AN237, Implementing a LIN Slave Node on PIC16F73 (DS00237).

FIRMWARE SETUP

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

Here are the basic steps required to setup your project:

1. Setup a project in MPLAB® IDE. Make sure you have the important driver files included in your project: `serial.asm`, `lin.asm`, and `linevent.asm`.
2. Include a main entry point in your project, `main.asm`. Edit this file as required for the application. Be certain that the interrupt is setup correctly. In addition, initialize the driver and ensure any external symbols are included.
3. Edit `linevent.asm` to respond to the appropriate IDs. This could be a table or 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 connected 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.

Project Setup

The first step is to setup the project. 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
- `serial.asm` - the serial communications engine
- `lin.asm` - the LIN driver
- `linevent.asm` - LIN event handling table

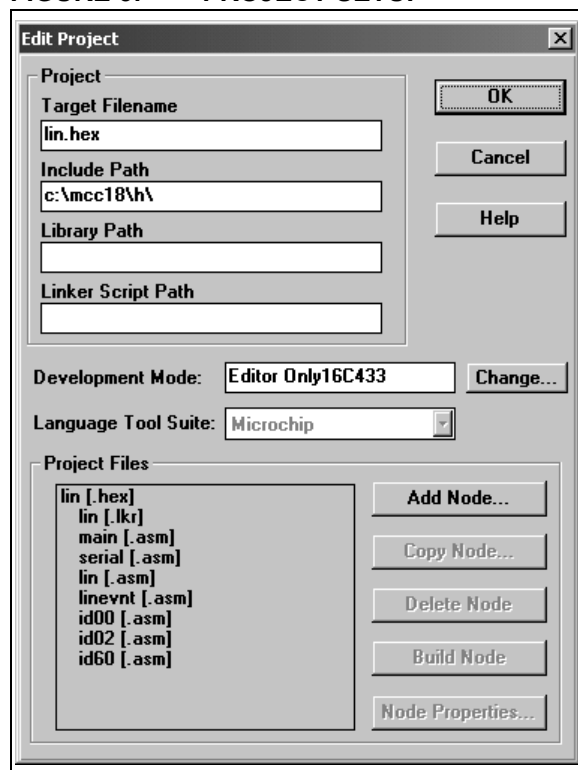
Any additional files are defined by the system designer for the specific application. For example, Figure 3 shows these files labeled as `idxx.asm`, where `xx` represents the LIN ID number. This is simply a programming style that separates ID handling into individual objects, thus, making the project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

Main Object

The `main.asm` module contains the entry point into the program. This is where the driver, hardware, and variables should be initialized. To initialize the driver, call the `l_init` function. The firmware in Appendix C demonstrates this.

Also included in this module is the interrupt vector. The serial function is interrupt based and must be included in the interrupt vector routine. This is demonstrated in the code in Example 1.

FIGURE 3: PROJECT SETUP



EXAMPLE 1: INTERRUPT CODE

```

_INTERRUPT_V   CODE 0x0004
InterruptHandler
    movwf  W_TEMP      ;Save
    swapf STATUS, W
    clrf   STATUS
    movwf  STATUS_TEMP

    call  SerialEngine

    swapf  STATUS_TEMP, W ;Restore
    movwf  STATUS
    swapf  W_TEMP, F
    swapf  W_TEMP, W

    retfie

```

Definitions

There are numerous compile time definitions, all of them located in `lin.def`, that are used to setup the system. Table 1 lists and describes these definitions. Likewise, the definitions are also listed in Appendix B. Only five of these definitions are critical for getting a system running. They are:

- `MAX_BIT_TIME`
- `NOM_BIT_TIME`
- `MIN_BIT_TIME`
- `USE_GP_CHANGE`
- `RX_ADVANCE` (or `RX_DELAY`)

The first three definitions, the bit time definitions, setup the baud rate and its boundary for communication. The next item depends on your application hardware design. Setup an external START edge detection source; the two options for the PIC16C433 are the INT pin or the GPIO interrupt-on-change. The last, yet very important definition, is the receive advance or delay. The receive advance or delay is used to advance or delay the time-base to align to the center of the next bit after the START bit.

AN240

TABLE 1: COMPILE TIME DEFINITIONS

Definition Name	Value	Description
LIN_IDLE_TIME_PS	b'10010110'	This is the value loaded into the option register when the LIN bus is IDLE. A prescaler setting of 128x is the desired choice for the prescaler.
LIN_ACTIVE_TIME_PS	b'10001000'	This is the value loaded into the option register when the slave is actively receiving or transmitting on the LIN bus. A prescaler value of 1x is ideal for bit rates between 4800 and 14400 bps at 4 MHz.
LIN_SYNC_TIME_PS	b'10010010'	This is the value loaded into the option register when the slave is capturing the sync byte. A prescaler setting of 8x is the desired choice for the prescaler.
MAX_BIT_TIME	d'118'	This is the upper bound bit time for synchronization. This should equal $((F_{osc} \times 1.15) / 4) / (\text{bit rate}) - 2$.
NOM_BIT_TIME	d'102'	This is the nominal bit time for synchronization. This should equal $(F_{osc} / 4) / (\text{bit rate}) - 2$.
MIN_BIT_TIME	d'87'	This is the lower bound bit time for synchronization. This should equal $((F_{osc} \times 0.85) / 4) / (\text{bit rate}) - 2$.
MAX_IDLE_TIME	d'195'	This defines the maximum bus IDLE time. The specification defines this to be 25000 bit times. The value equals $25000 / 128$. The 128 comes from the LIN_IDLE_TIME_PS definition.
MAX_HEADER_TIME	d'39'	This is the maximum allowable time for the header. This value equals $((34 + 1) \times 1.4) - 10$. This should not be changed unless debugging.
MAX_TIME_OUT	d'128'	This specifies the maximum time-out between wake-up requests. The specification defines this to be 128 bit times.
RC_OSC	NA	This definition enables synchronization. Do not use this definition if using a crystal or resonator.
USE_GP_CHANGE	NA	Use this definition to configure the external interrupt to be a GPIO interrupt-on-change. The alternative is to use the INT pin.
BRK_THRESHOLD	d'11'	This value sets the receive break threshold. For low tolerance oscillator sources, this value should be '11'. For high tolerance sources, this value should be '9', as defined in the LIN specification.
RX_DELAY	0xF0	This is the receive delay. Use this to adjust center sampling for low bit rates, less than 7400 bps at 4 MHz. For lower bit rates, the delay should be longer. Note, this value is complimented (i.e., 0xF0 is a delay of 16 cycles). This must not be used in conjunction with RX_ADVANCE.
RX_ADVANCE	0x10	This is the receive advance. Use this to adjust center sampling for high bit rates, 7400 bps or greater at 4 MHz. This must not be used in conjunction with RX_DELAY.

LIN Events

The LIN event function decodes the ID to determine the next step, what to transmit, receive, and how much. The designer should edit or modify the event function to handle specific LIN IDs. Refer to Appendix C for an example. One possibility is to setup a jump table. Another option is to setup some simple compare logic. The example firmware uses simple compare logic.

ID Modules

The application firmware must be developed somewhere in the project. It can be in main or in separate modules; 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 is to remember to include all the external symbols that are used. The symbols used by the driver are in `lin.inc`, which should be included in every application module.

DRIVER USAGE

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:

- Using the `l_rxtx_driver` function
- Handling error flags
- Handling finish 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.

LIN Slave Driver

The LIN slave driver is a state machine written for foreground processing. Being in the foreground implies the function must be called often enough to retrieve data from the receive buffer within the serial engine. Typically, the best place to call the driver is in the main program loop, and it should be called as often as possible. Thus, if some application level tasks are sufficiently long, then the driver function will most likely need to be called more than once in the main loop.

Example 2 demonstrates what the main program loop might look like with the call to the `l_txrx_driver` function.

EXAMPLE 2: MAIN APPLICATION LOOP

```

Main                                ; Main application loop

    call    l_txrx_driver

    call    l_id_02_function          ; Check for ID02 (tx)

    btfsc  LF_RX
    call    l_id_00_function          ; Check for ID00 (rx)

    movf   LIN_STATUS_FLAGS, W       ; Handle errors
    btfsc  STATUS, Z
    goto   Main

    btfsc  LE_BTO                     ; Was the bus time exceeded?
    goto   PutLINToSleep

    clrf   LIN_STATUS_FLAGS          ; Reset any errors
    goto   Main

```

Finish Flags

Two flags indicate when the driver has successfully transmitted or received data, the receive and the transmit flags. The receive flag is set when data has been received without error. The flag must be cleared by the user after it is handled. Likewise, the transmit flag indicates when data has been successfully transmitted without error. It must also be cleared by the user when it is handled. Example 2 gives an example of this.

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 (`LE_SYNC`) will get set in the driver. Refer to Appendix B for a list of all errors.

Errors are considered fatal until they are handled and cleared. Thus, if the error is never cleared, the driver will ignore incoming data. Example 2 demonstrates this and tests the bus time-out error.

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, thus, 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 B for descriptions of the state flags. For most situations, these flags will not need to be used within the application.

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?" In addition, "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. Example 3 shows an example.

EXAMPLE 3: VARIABLE INITIALIZATION

```
l_id_00
  GLOBAL      l_id_00

  movlw      ID00_BUFF      ; Set the pointer
  movwf      LIN_POINTER

  movlw      0x20           ; Adjust the frame time
  addwf      FRAME_TIME, F
  movlw      0x02           ; Setup the data count
  movwf      LIN_COUNT
  retlw      0x00           ; Read
```

The pointer to memory, `LIN_POINTER`, tells the driver where to store data for receiving, or where to retrieve data for sending. The frame time, `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 and the checksum bits. The counter, `LIN_COUNT`, simply tells the driver how much data is needed to operate.

Note: The count must always be initialized to something greater than zero for the driver to function properly.

Bus Wake-up

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

GENERAL INFORMATION

Additional Interrupts

It is possible to add extra interrupts, however, it is definitely not recommended. The driver uses an external interrupt to synchronize timing to the START edge. If additional interrupts are added, then uncertainty is added to the received START edge. Uncertainty in receiving the START edge can severely degrade the maximum bit rate.

There is a finite amount of uncertainty that is acceptable for a given bit rate and clock frequency, defined as follows:

$$\frac{1}{B_I} - (T_{E1(low)} + T_{E2(high)} + 2T_{ES}) = T_w = \frac{4N_{INS}}{F_{OSC}}$$

Without going into too much detail, this equation derives the maximum number of instructions related to uncertainty in terms of the ideal bit rate and frequency, which is discussed later in this document. The real question is: what instructions can be counted in this uncertainty, and the answer is: it depends on the way the code is written for the application. It is also important to note that some uncertainty is already assumed.

To be safe (i.e., not sacrifice reliability), avoid adding any extra code to the interrupt unless your instruction rate to bit rate ratio is greater than 200, or the number of instructions added is extremely small.

Read-Modify-Write on GPIO

When writing code that uses the GPIO port, be aware of problems that arise when using read-modify-write instructions, `bsf` and `bcf`. The LINRX bit in GPIO is physically an open drain output pin connected to the LIN transceiver. If data is being received via the interrupt driven serial engine and a `bsf` or `bcf` instruction is executed, it is possible that a low bit could be written back to the RXPIN and actively pull the received data low. Although the serial engine is designed to always return the RXPIN to a recessive high state, this type of condition should be avoided whenever possible.

Prescaler Definitions

The prescaler definitions are set to achieve bit rates between 4800 and 14400 bps, using a 4 MHz clock source. It is possible to adjust these to achieve lower bit rates. For example, multiplying all the prescale values by two would yield much lower bit rates, if desired.

LIN Event Handler

Event handling should be as short as possible. If the event handler is long, unacceptable interbyte space may be seen between receiving the ID byte and transmitting data from the slave to the master. Thus, choose the best method for decoding in your application. If you are only responding to a few IDs, then simple XOR logical compares will suffice. If any more IDs are responded to, then use a jump table. A complete jump table uses a significant amount of program memory; however, it is very quick to decode IDs.

Driver Call

The driver is not a true background task, only the serial communications are. Thus, the driver function must be called frequently. There are two potential problems if the driver function is not called frequently enough. The receive buffer could be overrun, which means the entire packet would be corrupted. Another problem is unacceptable interbyte space during slave to master transmissions. To be safe, insure the driver function is called at least four times for every byte. The driver function will execute very quickly if there is no action required.

IMPLEMENTATION

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

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

Serial Engine

The serial engine is interrupt driven firmware. It handles all bit level communications and synchronization. The function requires an external interrupt source configurable to either GPIO interrupt-on-change, or the INT pin. In addition, Timer0 is used to control asynchronous communications.

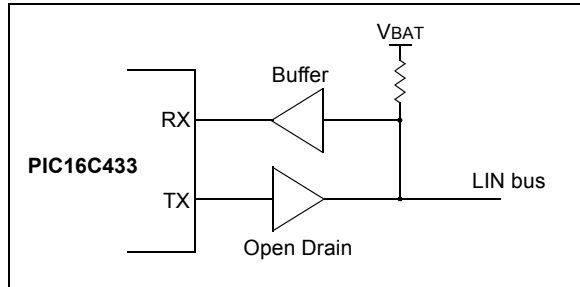
SYNCHRONIZATION

Synchronization is performed by stretching the bit rate clock and using the external interrupt to count the edges of the sync byte. After the last falling edge of the sync byte, the time is captured and compared to the maximum and minimum bit time tolerances specified. If within the tolerance, the value is used as the new time-base.

READ BACK TRANSMISSION

The software UART handles asynchronous communications much like a hardware UART; it receives data and generates errors under various conditions. Because the LIN physical layer has a feedback path for data (see Figure 4), the UART also reads back transmitted data.

FIGURE 4: SIMPLIFIED LIN TRANSCEIVER



The UART is designed to pre-sample before transmitting to capture feedback information. Transmit operations take 11 bit times to accurately capture the last bit in the transmission.

SERIAL STATUS FLAGS

There are a few flags within the software UART to control its operation, and to feed status information to functions outside the UART. Appendix B lists and defines these flags.

Transmit/Receive Driver

The `l_rxtx_driver` is a state machine. Bit flags are used to retain information about various states within the driver. In addition, status flags are maintained to indicate errors during transmit or receive operations.

STATES AND 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, and then jumps to that state. Figure 5 and Figure 6 outline the program flow through the different states. The states are listed and defined in Appendix B.

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 transmit/receive 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 transmit/receive 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. If time-out conditions do occur, the status flags are set to indicate the condition.

Hardware Initialization

An initialization function, `l_init`, is provided to setup the necessary hardware settings. Also, the state and status flags are all cleared. The function can also be used to reset the LIN driver.

Wake-up

The only time the slave can transmit to the bus without a request is when the bus is sleeping. 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

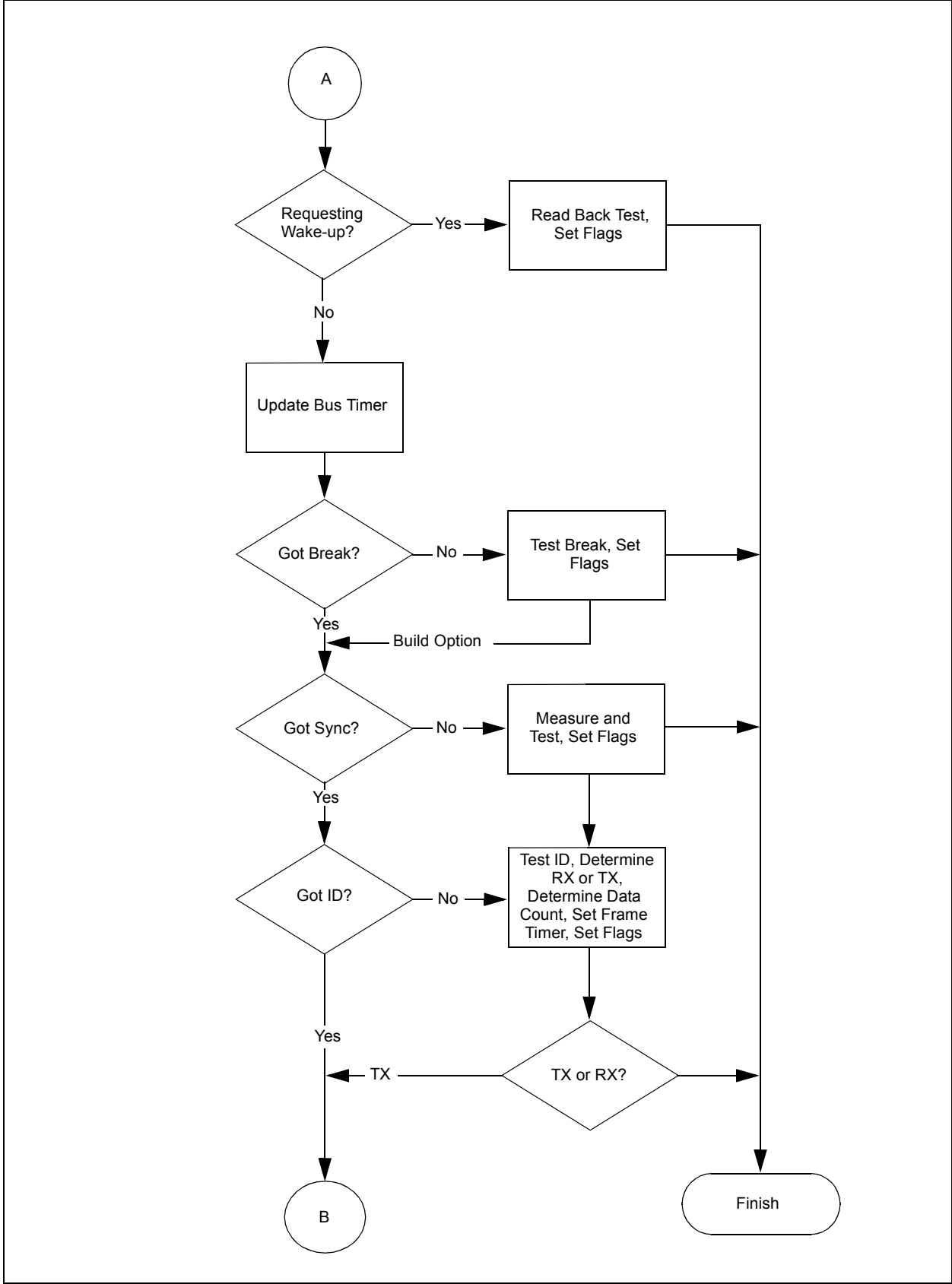


FIGURE 6: TRANSMIT/RECEIVE MESSAGE PROGRAM FLOW

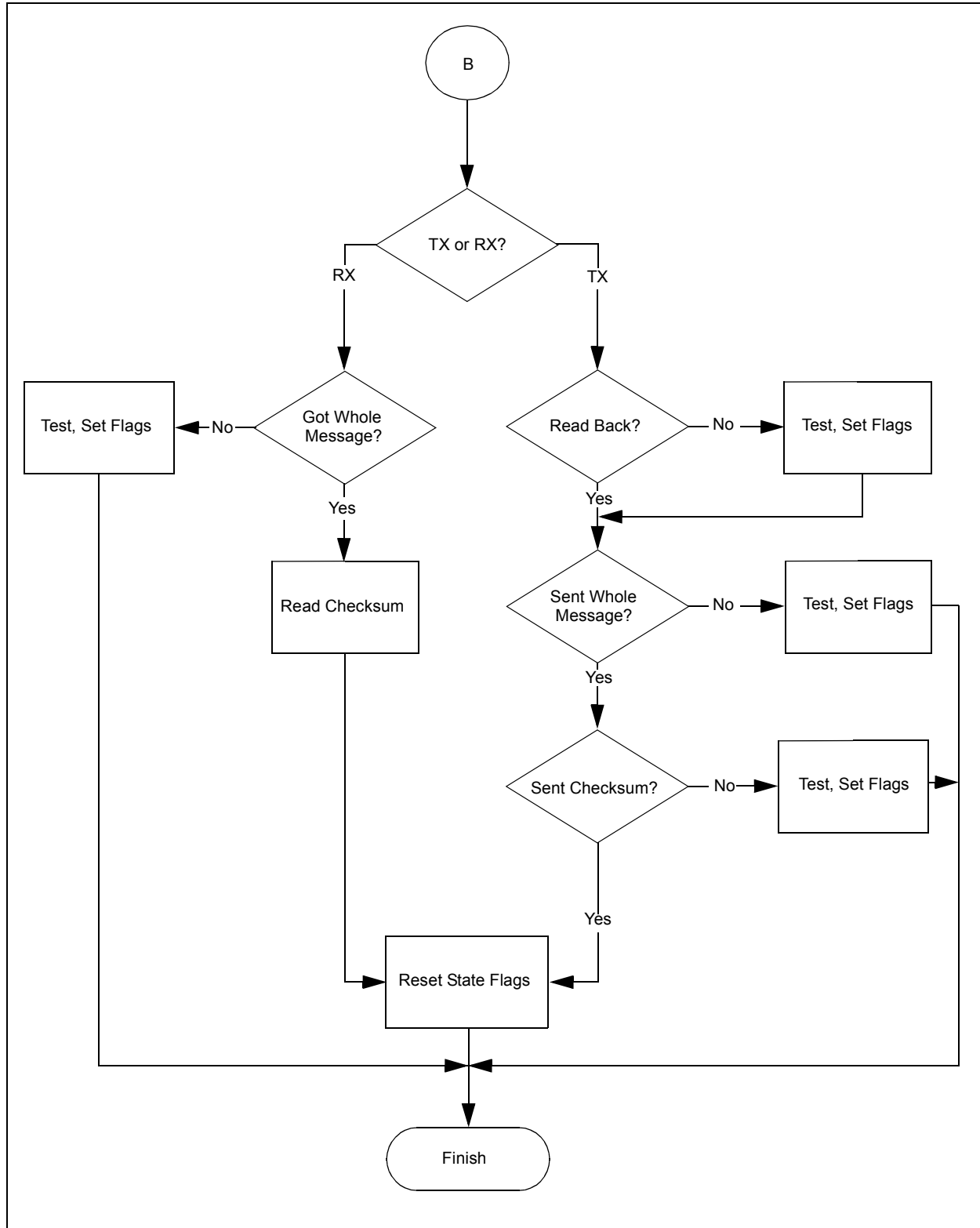
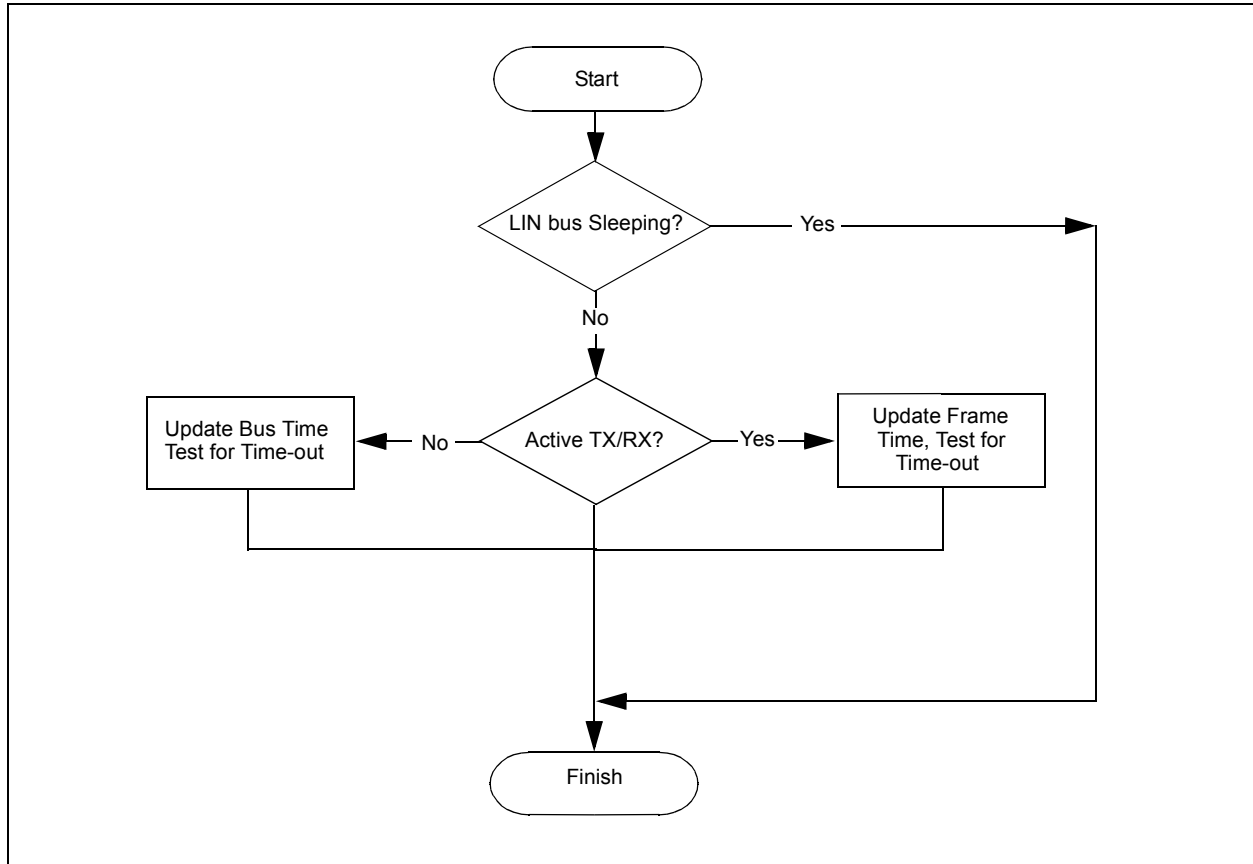


FIGURE 7: TIMEKEEPING PROGRAM FLOW



OPERATING REGION EVALUATION

It is important to understand the relationship between bit rate and clock frequency when designing a slave node in a LIN network. Therefore, 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. This section uses the defined physical limits and does not present any analysis of the limits defined for 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 the LIN specification.

General Information

Some general information used throughout the analysis is provided here.

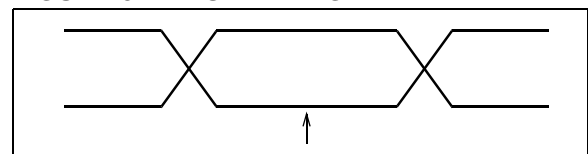
DATA RATE VS. SAMPLING RATE

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.

SAMPLING

The ideal sampling point is assumed as the center of the incoming bit, as shown in Figure 8. The equations presented in the following sections use this point as the reference.

FIGURE 8: SAMPLING



CLOCK FREQUENCY ERROR TO BIT ERROR RELATION

The LIN specification refers to clock frequency error, rather than bit error. Because of this, the LIN system designer must design the system with like clock sources; however, this is rather impractical. It is more feasible to have clock sources designed for the individual needs of the node. 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 low clock frequency errors, the bit rate error can be approximated by:

$$-E_F \approx E_B$$

Thus, a $\pm 2\%$ frequency error is nearly the same bit rate error.

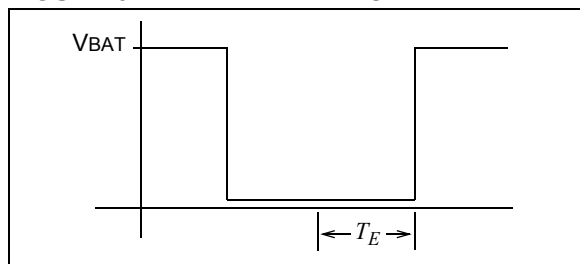
Acceptable Bit Rate Error

The LIN specification allows for a $\pm 2\%$ error for master/slave communications. This section evaluates this tolerance based on specified worst case conditions (slew rate, voltage, and threshold) and the implementation.

IDEAL SAMPLING WINDOW

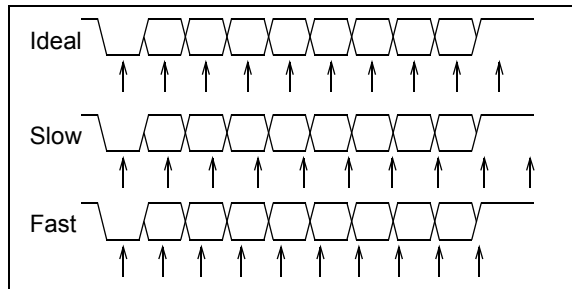
It is relatively easy to see the maximum allowed error in the ideal situation. Ideal is meant by infinite slew rate with a purely symmetrical signal, like the signal shown in Figure 9.

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 the transmission will be interpreted incorrectly. Figure 10 graphically depicts how data may be misinterpreted because of misaligned data and sampling rates.

FIGURE 10: DATA VS. SAMPLING



The following two equations give the maximum and minimum bit rates based on shifting time by one-half of one bit time, or $T_E = \pm 1/(2B_I)$.

$$\frac{1}{B_I} - \frac{T_E}{9} = \frac{1}{B_{max}}$$

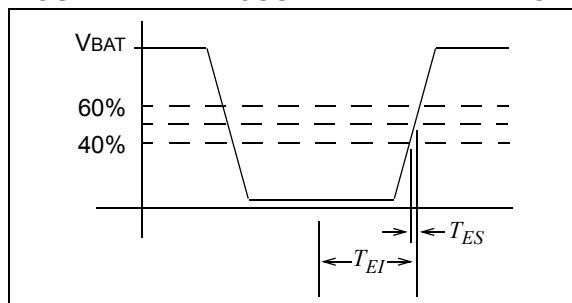
$$\frac{1}{B_I} + \frac{T_E}{9} = \frac{1}{B_{min}}$$

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. 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, $1V/\mu s$ and $18V$, according to the LIN specification. 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 error time adjustment:

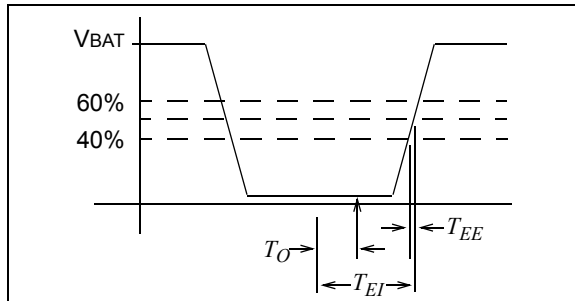
$$T_{EI} - T_{ES} = \frac{1}{2B_I} - \frac{(0.5V - 0.4V)}{(dV)/(dt)_{min}} = T_E$$

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

OFFSETS

An offset is a less than ideal sample point. For example, it is possible for a software UART to take a sample before or after the center point of an incoming bit, as shown in Figure 12. This is related to an offset from the START edge and ultimately shifts the bit rate error to favor one side over the other. For example, if the START edge detection is delayed for 10 μs from center of a 9.6 Kbit transmission, the absolute range for bit rate error is -4.1% and +6.9%.

FIGURE 12: OFFSET FROM CENTER



The example firmware leaves the Timer0 Interrupt enabled at all times to maintain some basic time about the LIN bus activity. A side effect of this is unpredictable offset. For example, if a START edge occurs while program execution is in an interrupt, the interrupt routine must finish before the START edge can be Acknowledged. Therefore, an undetermined offset from the START edge occurs.

Although the exact offset cannot be determined when interrupts are enabled, it is possible to determine a maximum offset. The maximum offset is related to the longest time through the interrupt when looking for a START edge. Having the maximum offset leads to the maximum bit rate.

The same equations apply as before; however, T_E is different for the maximum and minimum bit rate, because there is no time symmetry.

$$\frac{1}{B_I} - \frac{T_{E1}}{9} = \frac{1}{B_{max}}$$

$$\frac{1}{B_I} + \frac{T_{E2}}{9} = \frac{1}{B_{min}}$$

T_{E1} and T_{E2} are related:

$$\frac{1}{B_I} - 2T_{ES} = T_{E1} + T_{E2}$$

where:

$$T_{E2} = T_{EI} - T_{ES} - T_O$$

$$T_{E1} = T_{EI} - T_{ES} + T_O$$

Ultimately, the LIN specification requires that the slave accept as much as a $\pm 2\%$ error between the incoming bit rate (B_I) and the sampling bit rate. T_{E1} and T_{E2} have specific limits for offsets before and after the center sampling point. They are:

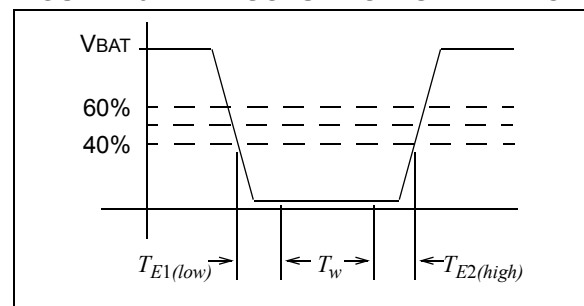
$$T_{E2(high)} = \frac{-9(-0.02)}{(0.98)(B_I)}$$

$$T_{E1(low)} = \frac{9(0.02)}{(1.02)(B_I)}$$

With these times, the total window time shown in Figure 13, can be calculated to determine the maximum allowable offset or the maximum interrupt duration:

$$\frac{1}{B_I} - (T_{E1(low)} + T_{E2(high)} + 2T_{ES}) = T_w = \frac{4N_{INS}}{F_{OSC}}$$

FIGURE 13: ABSOLUTE OFFSET WINDOW



The $4/F_{osc}$ term is the instruction time. Multiplying the instruction time by the number of executed instructions in the interrupt routine results in the total time through the interrupt.

Substituting the time equations $T_{E1(low)}$, $T_{E2(high)}$, and T_{ES} , and solving B_I yields the maximum bit rate:

$$B_I = \frac{0.6399}{\frac{4N}{F} - 1.8\mu s}$$

Adjusting this down by 15% to allow for synchronization tolerances leads to maximum allowable bit rate. For example, for a slave node operating at 4 MHz with a maximum instruction count of 40 through an interrupt, the maximum ideal bit rate would be about 14.2 kbps. Beyond 14.2 kbps, there is a significant probability that incoming data will be misinterpreted.

Minimum Samples Per Bit

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 gaps where the error is outside the defined bit rate error range for a particular number of instructions per bit. 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 the number of instructions executed per bit, bit rate, and bit rate error:

$$F_{OSC} = (E_B + 1)(N)(4)(B)$$

OPERATION OVERLAP

For a large number of instructions executed per bit, the slave will synchronize and communicate well with the master. However, for a particular error range, $\pm 2\%$, with higher bit rates and lower clock frequencies, the slave may never synchronize and communicate.

To approach this problem, the minimum frequency for a number of instructions, $(E_L + 1)(N)(4)(B_I)$, must be compared to the maximum frequency for one less number of instructions, $(E_H + 1)(N - 1)(4)(B_I)$. Where these are equal is the border between continuous and discontinuous operation for any given input frequency:

$$(E_L + 1)(N)(4)(B_I) = (E_H + 1)(N - 1)(4)(B_I)$$

Solving this equation yields:

$$N_{low} = \frac{(E_H + 1)}{(E_H + 1) - (E_L + 1)}$$

Therefore, the minimum number of instructions, N_{low} , must be executed per bit to accept the defined error. For example, for a $\pm 2\%$ error, the lowest number of instructions accepted before certain clock frequency/bit rate combinations become a problem is 26. Note that the value of 26 is much lower than the number of instructions through the interrupt.

MEMORY USAGE

The firmware code size depends on the build conditions. As it is currently built with the example application, the firmware occupies 412 words of program memory and 46 bytes of data memory.

REFERENCES

1. LIN Specification Package Revision 1.2, <http://www.lin-subbus.org>
2. MPASM™ User's Guide with MPLINK™ and MPLIB™, Microchip Technology Inc., 1999

APPENDIX B: SYMBOLS

TABLE B-1: FUNCTIONS

Function Name	Purpose
<code>l_txrx_table</code>	This function is called by the transmit/receive daemon after the identifier byte has been received. Message length and direction is returned to the driver. Within the table, pointers could be set up for different identifies.
<code>l_txrx_driver</code>	The core transmit and receive function, which 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).
<code>UpdateTimer</code>	Used to update the bus and frame timers. Called within the serial engine.
<code>SerialEngine</code>	This is the interrupt driven software UART.
<code>l_init</code>	Call this function to initialize or reset the hardware associated to the LIN interface.
<code>l_tx_wakeup</code>	Wake-up function. Call this function to wake-up the bus if asleep.

TABLE B-2: REGISTERS

Variable Name	Purpose
<code>BRK_CNT</code>	Break counter.
<code>BUS_TIME</code>	Bus timer.
<code>FRAME_TIME</code>	8-bit frame timer register.
<code>HEADER_TIME</code>	Same as <code>FRAME_TIME</code> .
<code>LIN_ID</code>	Holding register for the received identifier byte. This register is used in the <code>l_txrx_table</code> function to determine how the node should react.
<code>LIN_POINTER</code>	Pointer to a storage area used by the driver. Data is either loaded into or read from memory depending on the identifier.
<code>LIN_COUNT</code>	Used by the driver to maintain a message data count.
<code>LIN_CHKSUM</code>	Used by the driver to calculate checksum for transmit and receive.
<code>LIN_FINISH_FLAGS</code>	Flags to indicate a successful receive/transmit.
<code>LIN_STATE_FLAGS</code>	Flags to indicate what state the LIN bus is in.
<code>LIN_STATE_FLAGS2</code>	Flags to indicate what state the LIN bus is in.
<code>LIN_STATUS_FLAGS</code>	Contains status information about the LIN bus.
<code>RXDATA_BUF</code>	Buffer for received data.
<code>SYNC_CNT</code>	Sync counter.
<code>TIME_BASE</code>	This register holds the time per bit based on the number of instructions.
<code>TXSR</code>	The Most Significant Byte of the transmit shift register.
<code>TXSR_2</code>	The Least Significant Byte of the transmit shift register.
<code>RXDATA</code>	The Least Significant Byte of the receive shift register. The data is also pulled from this register after a complete receive.
<code>RXSR_2</code>	The Most Significant Byte of the receive shift register.
<code>RXTX_COUNT</code>	Bit counter register for the software UART.
<code>SERIAL_FLAGS</code>	This register holds the flags to control the software UART.

TABLE B-3: FLAGS

Variable Name		Purpose
LS_BUSY	LIN_STATE_FLAGS	Indicates the LIN bus is busy.
LS_TXRX	LIN_STATE_FLAGS	Indicates transmit or receive operation.
LS_RBACK	LIN_STATE_FLAGS	Indicates a read back is pending.
LS_BRK	LIN_STATE_FLAGS	Indicates a break has been received.
LS_SYNC	LIN_STATE_FLAGS	Indicates a sync byte has been received.
LS_ID	LIN_STATE_FLAGS	Indicates the identifier has been received.
LS_DATA	LIN_STATE_FLAGS	Indicates all data has been sent or received.
LS_CHKSM	LIN_STATE_FLAGS	Indicates the checksum has been sent or received.
LS_WAKE	LIN_STATE_FLAGS2	Indicates a wake-up has been requested (this node only).
LS_SLPNG	LIN_STATE_FLAGS2	Indicates the LIN bus is sleeping.
LE_BIT	LIN_STATUS_FLAGS	Indicates a bit error.
LE_PAR	LIN_STATUS_FLAGS	Indicates a parity error.
LE_CHKSM	LIN_STATUS_FLAGS	Indicates a checksum error during a receive.
LE_SYNC	LIN_STATUS_FLAGS	Indicates a synchronization tolerance error.
LE_GEN	LIN_STATUS_FLAGS	Indicates a general error, typically a framing error.
LE_FTO	LIN_STATUS_FLAGS	Indicates a frame time-out error.
LE_BTO	LIN_STATUS_FLAGS	Indicates a bus activity time-out error.
LF_RX	LIN_STATUS_FLAGS	Indicates data has been received.
LF_TX	LIN_STATUS_FLAGS	Indicates data has been sent.
S_BUSY	SERIAL_FLAGS	Indicates the software UART is busy receiving and/or transmitting.
S_TXRX	SERIAL_FLAGS	Indicates the UART is transmitting or receiving.
S_RXIF	SERIAL_FLAGS	Indicates data has been received.
S_FERR	SERIAL_FLAGS	Indicates an invalid STOP bit was received.
S_SYNC	SERIAL_FLAGS	Indicates serial communications is synching.
S_SSTRT	SERIAL_FLAGS	Indicates serial communications is waiting to start synching.
S_SYNCERR	SERIAL_FLAGS	Indicates a synchronization error has occurred.

AN240

TABLE B-4: COMPILE TIME DEFINITIONS

Definition Name	Value	Description
LIN_IDLE_TIME_PS	b'10010110'	This is the value loaded into the option register when the LIN bus is IDLE. A setting of 128x is the desired choice for the prescaler.
LIN_ACTIVE_TIME_PS	b'10001000'	This is the value loaded into the option register when the slave is actively receiving or transmitting on the LIN bus. A value of 1x is ideal for bit rates between 4800 and 14400 bps at 4 MHz.
LIN_SYNC_TIME_PS	b'10010010'	This is the value loaded into the option register when the slave is capturing the sync byte. A setting of 8x is the desired choice for the prescaler.
MAX_BIT_TIME	d'118'	This is the upper bound bit time for synchronization. This should equal $((Fosc \times 1.15) / 4) / (\text{bit rate}) - 2$.
NOM_BIT_TIME	d'102'	This is the nominal bit time for synchronization. This should equal $(Fosc / 4) / (\text{bit rate}) - 2$.
MIN_BIT_TIME	d'87'	This is the lower-bound bit time for synchronization. This should equal $((Fosc \times 0.85) / 4) / (\text{bit rate}) - 2$.
MAX_IDLE_TIME	d'195'	This defines the maximum bus IDLE time. The specification defines this to be 25000 bit times. The value equals $25000 / 128$. The 128 comes from the LIN_IDLE_TIME_PS definition.
MAX_HEADER_TIME	d'39'	This is the maximum allowable time for the header. This value equals $((34 + 1) \times 1.4) - 10$. This should not be changed unless debugging.
MAX_TIME_OUT	d'128'	This specifies the maximum time-out between wake-up requests. The specification defines this to be 128 bit times.
RC_OSC	NA	This definition enables synchronization. Do not use this definition if using a crystal or resonator.
USE_GP_CHANGE	NA	Use this definition to configure the external interrupt to be a GPIO interrupt-on-change. The alternative is to use the INT pin.
BRK_THRESHOLD	d'11'	This value sets the receive break threshold. For low tolerance oscillator sources, this value should be '11'. For high tolerance sources, this value should be '9', as defined in the LIN specification.
RX_DELAY	0xF0	This is the receive delay. Use this to adjust center sampling for low bit rates, less than 7400 bps at 4 MHz. For lower bit rates, the delay should be longer. Note, this value is complimented (i.e., 0xF0 is a delay of 16 cycles). This must not be used in conjunction with RX_ADVANCE.
RX_ADVANCE	0x10	This is the receive advance. Use this to adjust center sampling for high bit rates, 7400 bps or greater at 4 MHz. This must not be used in conjunction with RX_DELAY.

APPENDIX C: 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

AN240

NOTES:

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families 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 knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is 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 products.

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.

Trademarks

The Microchip name and logo, the Microchip logo, KEELOQ, MPLAB, PIC, PICmicro, PICSTART and PRO MATE are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.


FilterLab, microID, MXDEV, MXLAB, PICMASTER, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

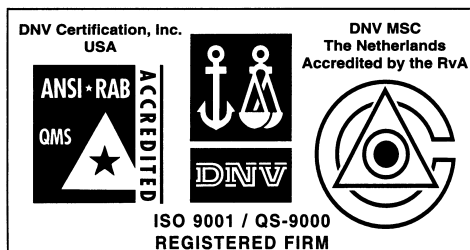
dsPIC, dsPICDEM.net, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, microPort, Migratable Memory, MPASM, MPLIB, MPLINK, MPSIM, PICC, PICDEM, PICDEM.net, rPIC, Select Mode and Total Endurance are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

Serialized Quick Turn Programming (SQTP) is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2002, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.



Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999 and Mountain View, California in March 2002. The Company's quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, non-volatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001 certified.



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'Shaughnessy 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 - 1er 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