AN2290

Bit-Banged Enhanced UART for 8-Bit PIC® Microcontrollers

Author: *Mary Tamar Tan*
Microchip Technology Inc.

INTRODUCTION

The majority of 8-bit PIC® microcontrollers have one or more on-chip Universal Asynchronous Receiver Transmitters (UARTs). But in cases where no UART hardware is available, or if an additional serial communication interface is required, bit-banging will be the best option. Bit-banging is a technique used to create a serial I/O communications interface in software instead of using a dedicated hardware peripheral. Data transmission and reception are controlled almost entirely in software. That includes sampling, level detection, timing, synchronization, buffer control, driver states switching, and error detection.

This application note focuses on the implementation of a bit-banged UART driver on an 8-bit PIC microcontroller. Calculations, performance and accuracy factors, limitations, and firmware details are also discussed. It should be noted that a couple of hardware peripherals, Timer0 and Interrupt-on-Change pin, are used in the driver for more accurate timing and process time reduction. This application note demonstrates how to configure these peripherals using the MPLAB® Code Configurator (MCC). The user should be able to setup the bit-banged UART driver in just a few minutes by following the step-by-step procedures presented in this document.

The driver also features enhanced capabilities for full compatibility with the existing MCC LIN Stack library. For this reason, the driver will be referred to as bit-banged Enhanced UART (EUART) throughout this document. Users interested in the implementation of the driver with the LIN library can refer to *AN2059 "LIN Basics and Implementation of the MCC LIN Stack Library on 8-Bit PIC® Microcontrollers"* (DS00002059) for a more detailed discussion.

FEATURES

The bit-banged EUART includes the following capabilities:

- Half-duplex asynchronous transmit and receive
- Baud rate of up to 38.4K @ 8 MHz clock frequency
- Interrupt-driven reception using an IOC pin
- Bit period dependent on Timer0 interrupt
- False start detection
- Input buffer overrun error detection
- Received character framing error detection
- Automatic error handling mechanisms

The bit-banged EUART implements the following additional features to support the MCC LIN Drivers:

- Automatic detection and calibration of baud rate
- Break detection for LIN slave nodes
- Break transmission for LIN master nodes

RESOURCE USAGE

A few on-chip peripherals are used to maintain robust communications and precise timing. Timer0 is used to maintain the timing of each bit transmission and reception. The timer prescaler is adjusted to ensure that the EUART is operating at the desired baud rate. An Interrupt-on-Change (IOC) pin is used for Start bit (negative edge) detection for each received byte. IOC is also used to detect the five rising (positive) edges of the Sync character in LIN bus systems.

PROCESS TIME

The process time used by the driver for the execution of EUART tasks is affected by the bit rate, clock frequency, and code design. The use of interrupts allows the EUART to consume much less process time compared with polling methods. For a single character transmitted or received, only a portion of the CPU process time during the whole character duration is actually used by the driver. For example, when shifting out a bit, the driver only executes tasks (i.e., set a pin to high or low) on every timer interrupt and eventually frees up the CPU for other applications. The available process time for non-UART tasks ($T_{NON\text{-}UART}$) is expressed in percentage and is calculated as shown in Equation 1.

EQUATION 1: PERCENT AVAILABLE PROCESS TIME

$$\%Available\ Process\ Time = \left(1 - \frac{T_{UART}}{T_{Char}}\right) \times 100\%$$

where,

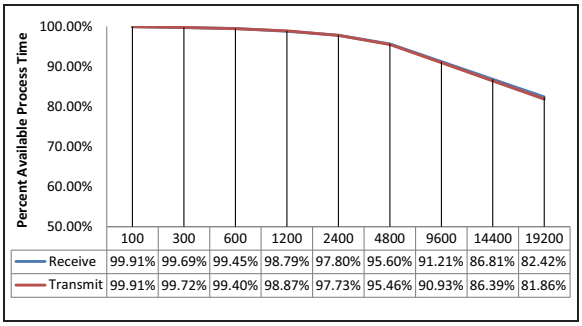
TUART = Total time required to emulate the EUART bit signal for each character

TCHAR = TUART + TNON-UART

Take note that the measured process time may vary per character transmitted or received due to code execution. The time it takes for setting a pin to 'high' is different from setting a pin to 'low', and the time it takes for sampling a 'high' is different from sampling a 'low'. In other words, the user should expect a very small difference between the process time for a character 'A' and character 'Z'. Since the EUART is interrupt-driven, only TUART and TNON-UART may vary but TCHAR will always remain constant given a specific baud rate.

Figure 1 shows the approximate available process time for a PIC16F1718 device running at 4 MHz using the 16 MHz INTOSC. The measurements were taken from sending and receiving the ASCII character 'M' (4Dh).

FIGURE 1: AVAILABLE PROCESS TIME AT 4 MHZ



	100	300	600	1200	2400	4800	9600	14400	19200
Receive	99.91%	99.69%	99.45%	98.79%	97.80%	95.60%	91.21%	86.81%	82.42%
Transmit	99.91%	99.72%	99.40%	98.87%	97.73%	95.46%	90.93%	86.39%	81.86%

It is evident from Figure 1 that the percent available process time decreases as the baud rate increases. This is because for a specific clock frequency, TUART is constant for all baud rates while TCHAR decreases as the baud rate increases.

ASYNCHRONOUS TRANSMIT AND RECEIVE

The bit-banged EUART transmits and receives data using the standard non-return-to zero (NRZ) format. Consecutively transmitted data bits of the same value stay at the output level of that bit without returning to a neutral level between each bit transmission.

Each character transmission consists of one Start bit followed by eight data bits and is always terminated by a Stop bit. The Start bit is always a "space" (low) and the Stop bit is always a "mark" (high). The transmission port idles in the Mark state. Each transmitted bit persists for a period of 1/(Baud Rate).

The EUART transmits and receives the LSb first. In Half-Duplex mode, transmit and receive cannot be performed simultaneously. They only share the same data format and baud rate.

BAUD RATE

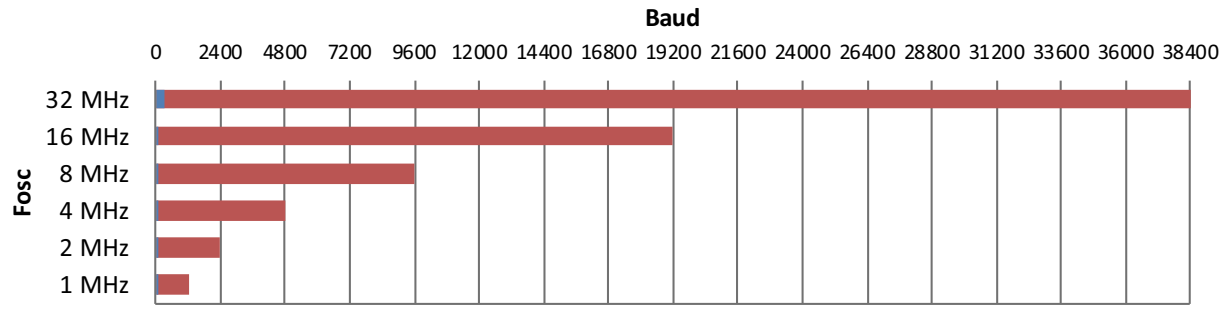
Baud rate defines how fast data is transferred through the serial port. In EUART, baud rate is equivalent to the bit rate which means that it is also the number of data bits that can be shifted in or out of the serial port in one second.

Note: The terms "baud rate" and "bit rate" will be used interchangeably throughout this document. Their definitions are literally the same in this context.

Baud Rate Limits

One of the major parameters in determining the baud rate limits of the bit-banged EUART is the clock frequency (FCLK). For 8-bit PIC MCUs, it is equal to FOSC/4, where FOSC is the device oscillator frequency. Figure 2 depicts the recommended baud rate limits using different FOSC.

FIGURE 2: RECOMMENDED BAUD RATE LIMITS



	1 MHz	2 MHz	4 MHz	8 MHz	16 MHz	32 MHz
Max Baud	1200	2400	4800	9600	19200	38400
Min Baud	100	100	100	100	100	300

Generally, the minimum recommended baud rate is 100, except when using a 32 MHz oscillator in which instruction execution is too fast for the driver to handle. Figure 2 points to a direct correlation between the oscillator frequency and maximum achievable baud rate. It can be observed that as the device frequency is doubled, the maximum baud rate also approximately doubles. Operation behind the recommended limits may be possible but does not guarantee the driver's behavior. This is because of several hardware and software factors that will be discussed in Section , Performance and Accuracy.

Section Appendix A:, Baud Rate Versus Actual Rate shows a sample comparison between the desired baud rate and the actual baud rate for a PIC16F1718 microcontroller operating on different device frequencies.

Maximum and Minimum Baud Rate

The maximum and minimum achievable baud rate mainly depends upon the frequency of the selected clock source. Using Timer0, the maximum and minimum baud rates as a function of FOSC are expressed in Equation 2 and Equation 3, respectively.

EQUATION 2: MAXIMUM BAUD RATE FOR A GIVEN FOSC

$$Baud_{Max} = \frac{F_{OSC}}{4 \times (256 - 255) \times Prescalaer}$$

$$Baud_{Max} = \frac{F_{OSC}}{4 \times Prescaler}$$

EQUATION 3: MINIMUM BAUD RATE FOR A GIVEN FOSC

$$Baud_{Min} = \frac{F_{OSC}}{4 \times 256 \times Prescaler}$$

Timer0 Reload Values

Baud rate is basically the reciprocal of the timer period. Timer0 is implemented as an 8-bit timer that triggers an interrupt event on every overflow of the TMR0 Register. TMR0 is reloaded with a certain value such that it creates an interrupt on every bit period. Equation 4 shows how to calculate the TMR0 reload value for one bit period.

EQUATION 4: TMR0 RELOAD VALUE FOR ONE BIT PERIOD

$$TMR0_{OneBit} = 256 - \frac{F_{OSC}}{4 \times Baud \times Prescaler}$$

The bit-banged EUART receiver is designed to start the sampling routine with the Start bit. Hence, it is necessary to determine the period between the Start bit edge and the middle of the Start bit. The reload value for half bit period can be obtained using Equation 5.

EQUATION 5: CALCULATION OF TMR0 RELOAD VALUE FOR HALF BIT PERIOD

$$TMR0_{HalfBit} = TMR0_{OneBit} + \frac{256 - TMR0_{OneBit}}{2}$$

$$TMR0_{HalfBit} = 128 + \frac{TMR0_{OneBit}}{2}$$

$$\therefore 128 < TMR0_{HalfBit} < 256$$

The results of Equation 4 and Equation 5 are ideal, and therefore need to be adjusted by considering other parameters. These include hardware-induced delays such as interrupt latency, the two-instruction cycle delay following the write, and clock accuracy. Delays introduced by the firmware design should also be taken into account. Refer to Section , Performance and Accuracy for other factors that need to be considered for the calculations.

Timer0 Reload Adjustment

A few instruction cycles are executed before the TMR0 actually increments. Equation 6 shows how to calibrate the TMR0 reload value to compensate for the hardware- and software-introduced delays.

EQUATION 6: NEW TIMER0 RELOAD VALUE

$$TMR0_{New} = TMR0_{Ideal} + \frac{N}{Prescale\ Factor}$$

Where:

N = Number of Instruction Cycles

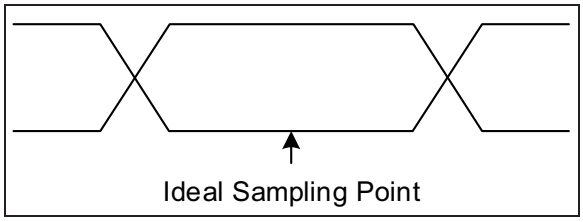
Prescale Factor = PS <2:0> + 1

During adjustment, it is suggested to modify only the "N" parameter. A logic analyzer with high timing accuracy can be connected to the TX pin while transmitting dummy data to determine if the transmitter has already met the desired baud rate. For the receiver, a test pin can be toggled upon entry and before exiting a Timer0 Interrupt Service Routine (ISR), with a two-channel logic analyzer connected to both the test pin and the RX pin to ensure that sampling rate is within the acceptable tolerance.

SAMPLING

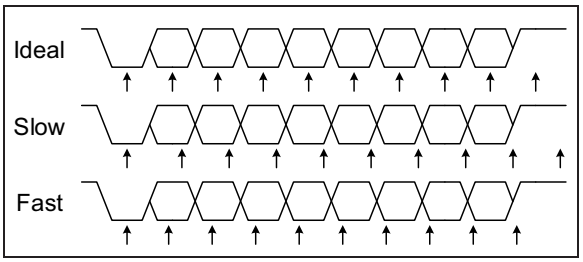
The ideal sampling point is assumed to be the center of the incoming bit, as shown in Figure 3.

FIGURE 3: SAMPLING



The bit-banged EUART implements an 8-bit character format which means that a total of ten bits, including the Start and Stop bits, need to be sampled. Each bit occupies 10% of the total time. So, if the difference between the incoming data rate and the receiver sampling rate is more than 10%, a bit will be missed out during sampling. This will result to inaccurate received data. The driver will never detect this unless a framing error has occurred. Figure 4 graphically depicts how data may be misinterpreted because of misaligned data and sampling rates.

FIGURE 4: DATA VS. SAMPLING



The accuracy of reception depends on the accuracy of both the sender and the receiver. If the sender is 5% faster and the receiver is 5% slower than the ideal baud rate, a 10% difference will occur and a bit will definitely be missed out. It is therefore recommended for each device to deviate not more than 2.5% from the ideal baud rate.

Note:	When this bit-banged EUART driver is used with the LIN driver, the bit tolerance is stricter. Refer to "Bit rate Tolerance" of LIN Spec 2.2A for more details.

PERFORMANCE AND ACCURACY

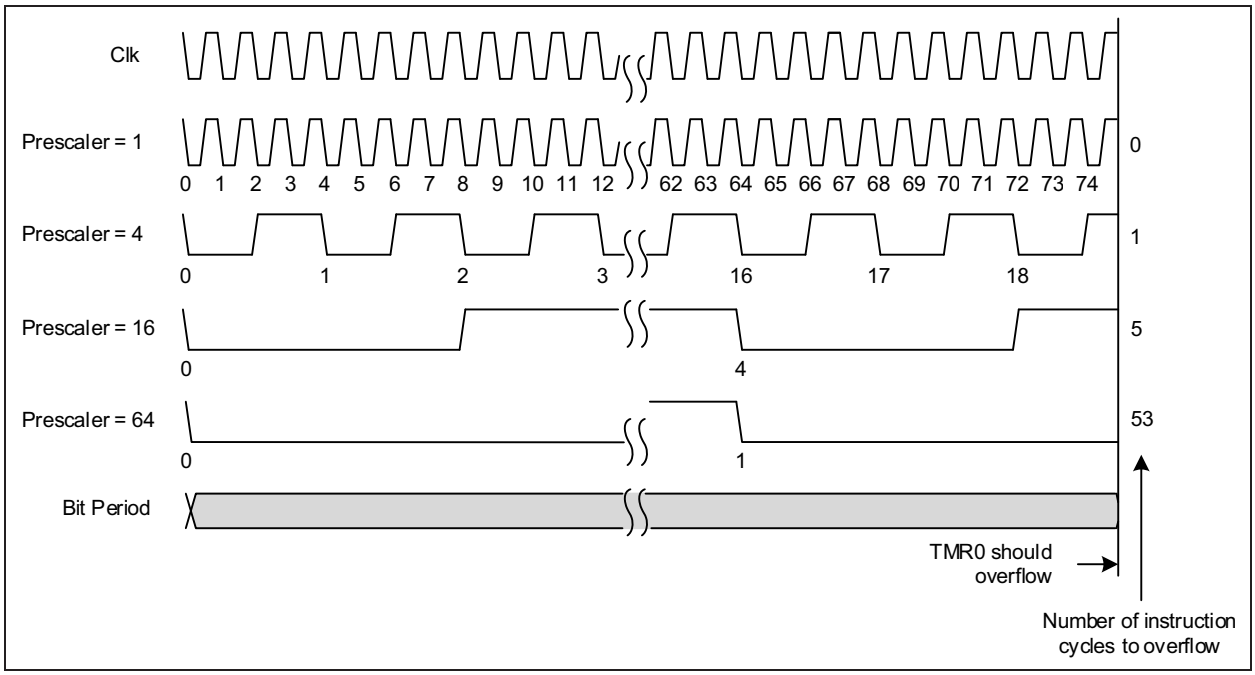
To get the best results, the user should be able to consider software and hardware factors that may affect the driver's performance and accuracy. A few of these factors are presented in this section.

Prescaling

The prescaler divides the device clock frequency with a certain fixed value before feeding it to the timer module. Timer0 has an exclusive 8-bit prescaler that can be configured with clock ratios from 2 up to 256. Prescaling allows the bit-banged EUART to operate at much lower baud rates given a specific device frequency. However, as the prescaler increases, the possible amount of error may also increase for a certain baud rate. Figure 5 shows how, without prescaler, TMR0 will overflow on the expected time. But, with prescalers of 4, 16 and 64, more instruction cycles are executed before the timer overflows. Prescalers must be carefully selected such that the desired baud rate is achieved with the best possible timer resolution.

In other words, it is preferable to have a faster input clock source than the desired EUART baud, and then divide down the speed using prescaler to the slower desired EUART speed. This will allow more instructions to be executed in between EUART bit generation.

FIGURE 5: PRESCALING



Integer Arithmetic Truncation

Results from manual calculations do not often exactly agree with the results of integer arithmetic performed by the microcontroller. For example, a baud rate of 9600 is desired using an 8 MHz oscillator. Plugging these values into Equation 4 will result to a TMR0 reload value of approximately equal to 47.67. The only integer values that can be used for TMR0 are 47 and 48. This will result to errors of 1.4% and 0.7%, respectively.

Interrupt Latency

The bit-banged EUART is mostly built out of state machines and interrupts. Interrupt latency is defined as the time from when the interrupt event occurs to the time code execution at the interrupt vector begins. Typical latency for synchronous (i.e., timer) interrupts is three to four instruction cycles. For asynchronous interrupts (i.e., interrupt-on-change), the latency is three to five instruction cycles, depending upon when the interrupt occurs. Refer to the specific device data sheet for more details on interrupt latency.

Clock Accuracy

Timer0 increments on every instruction cycle making it highly dependent on the accuracy of the selected oscillator. The frequency of oscillators may drift due to change in V_{DD}, environmental parameters such as temperature, humidity, pressure, and aging. The internal oscillator block of PIC microntrollers are factory-calibrated, but are also prone to frequency changes due to these factors. It is therefore necessary for the user to calibrate the device clock against a reference clock to achieve extremely minimal timing errors. Or, alternatively, use an external clock source that has less deviation over temperature and V_{DD} operating range.

SETTING UP ON MPLAB® X

System Requirements

- MPLAB® X IDE v3.40 or newer
- MPLAB® Code Configurator (MCC) v3.26 or newer
- Foundation Services Library v0.1.15 or newer

MCC Configuration

It is assumed that the user has basic knowledge on creating a new project on MPLAB X. Make sure that the project is set as main. The bit-banged EUART requires all hardware and software initialization to be done using MCC. The following steps show an example on how to configure a PIC16F1718 for 9600 baud with 16 MHz internal oscillator.

1. *Navigate to Tools > Embedded > MPLAB Code Configurator v3* to launch MCC version 3.
2. Under Project Resources, click **System** and select **System Module**.
3. Select the desired system clock settings. For the sample application, select Internal Oscillator (INTOSC) mode: I/O function on the CLK pin on the Oscillator Select menu. Set FOSC as the "System Clock" and 16 MHz_HF as the "Internal Clock". Uncheck the PLL Enabled, Software PLL Enabled, and Low-voltage Programming Enable checkboxes.
4. Disable Watchdog Timer.
5. Under Device Resources, select *Libraries > Foundation Services > SWUART*.
6. Select TMR0 as timer. Set baud rate to 9600, Transmit buffer size to 8 and Receive buffer Size to 8. The user should notice that the Actual Baud Rate is equal to 0.0 b/s. To set the actual baud rate, TMR0 should be configured with the Required Timer Period of 104.17 us.
7. When the SWUART is selected, the TMR0 module is automatically added to the **Project Resources** under **Peripherals**.
8. Set Clock Source to FOSC/4. To achieve the desired timer period, check **Enable Prescaler** and set to 1:2. Set the Requested Period to 104.17 us. The user will notice that the Actual Period is 104 us. This is the nearest possible period due to the arithmetic limitations as discussed in Section , Integer Arithmetic Truncation.
9. Check **Enable Timer Interrupt**.
10. Go back to *Project Resources > Libraries > Foundation Services > SWUART*. The Actual Baud Rate has changed into 9615.38 b/s.

Note:	The displayed "Actual Baud Rate" is calculated based on the "Actual Period" of Timer0 alone and does not consider other software and hardware limitations.

11. For this example, RC6 and RC7 are used as the TX and RX pins, respectively. To configure these pins, go to **Project Resources** and select **Pin Module**.
12. Under Pin Manager: Grid, set **Package** to PDIP28. Set RC7 as input and RC6 as output of the SWUART module. Go back to the Pin Module GUI.
13. To configure the TX pin, make sure that RC6 is set as output. Disable Analog and WPU and set IOC to "None".
14. For the RX pin, uncheck the **Start High**, **Analog**, **Output** and **WPU** checkboxes, and set RC7 IOC to "Negative". This will enable IOC on negative edge for this pin.
15. Navigate to *Project Resources > System > Interrupt Module*. In the GUI, make sure that the "Preemptive interrupt routine" is checked. TMR0 and Pin Module should be placed on the first and second interrupt priority, respectively.
16. Near the Project Resources menu, click **Generate**. When generating the MCC code for the first time, a pop-up window will appear asking for the configuration settings to be saved in a `.mc3` file format. Enter the desired file name then click **Save**.
17. The MCC configuration is now complete. A summary of the settings are shown in Figure 6.

FIGURE 6: MCC CONFIGURATION SETTINGS

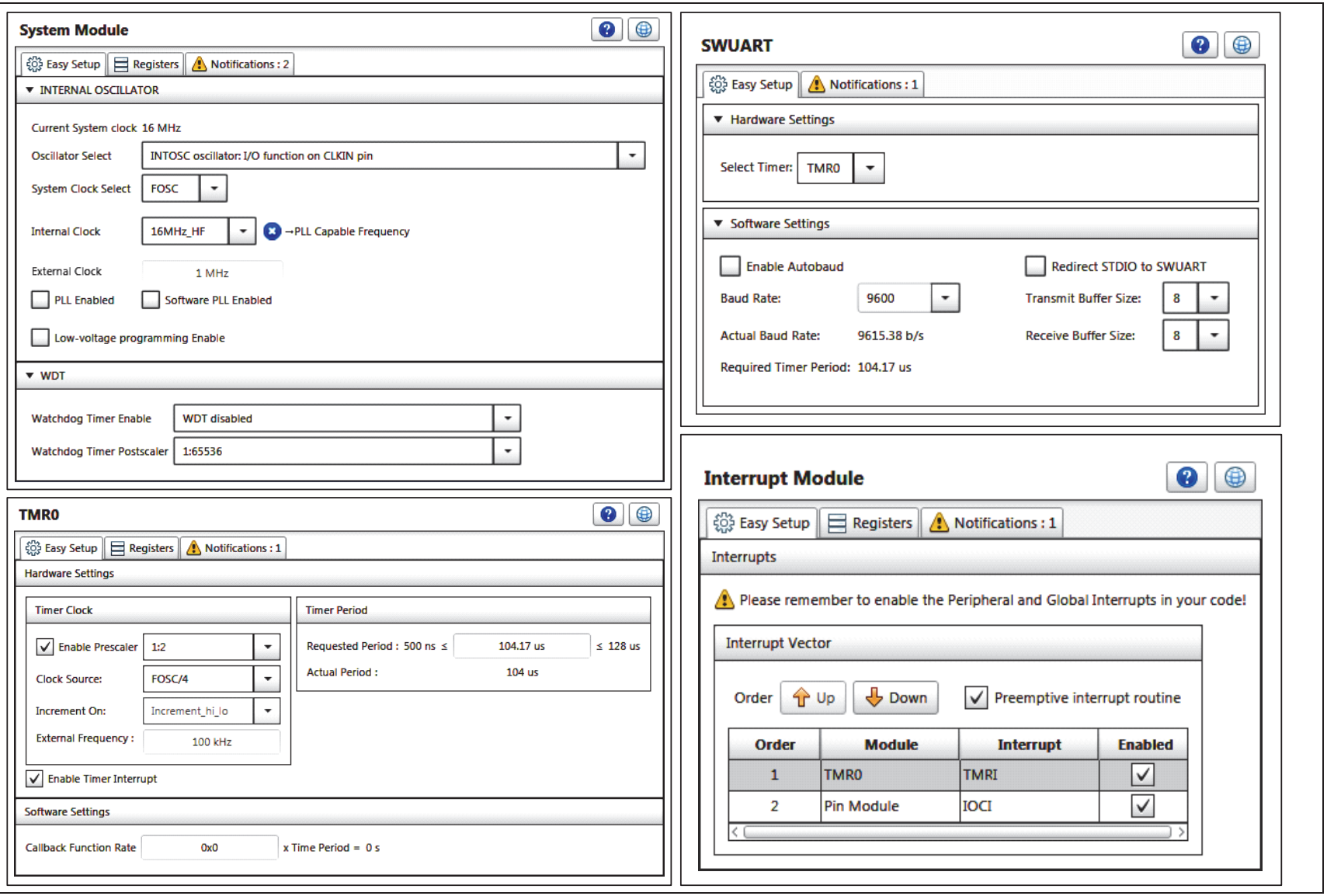
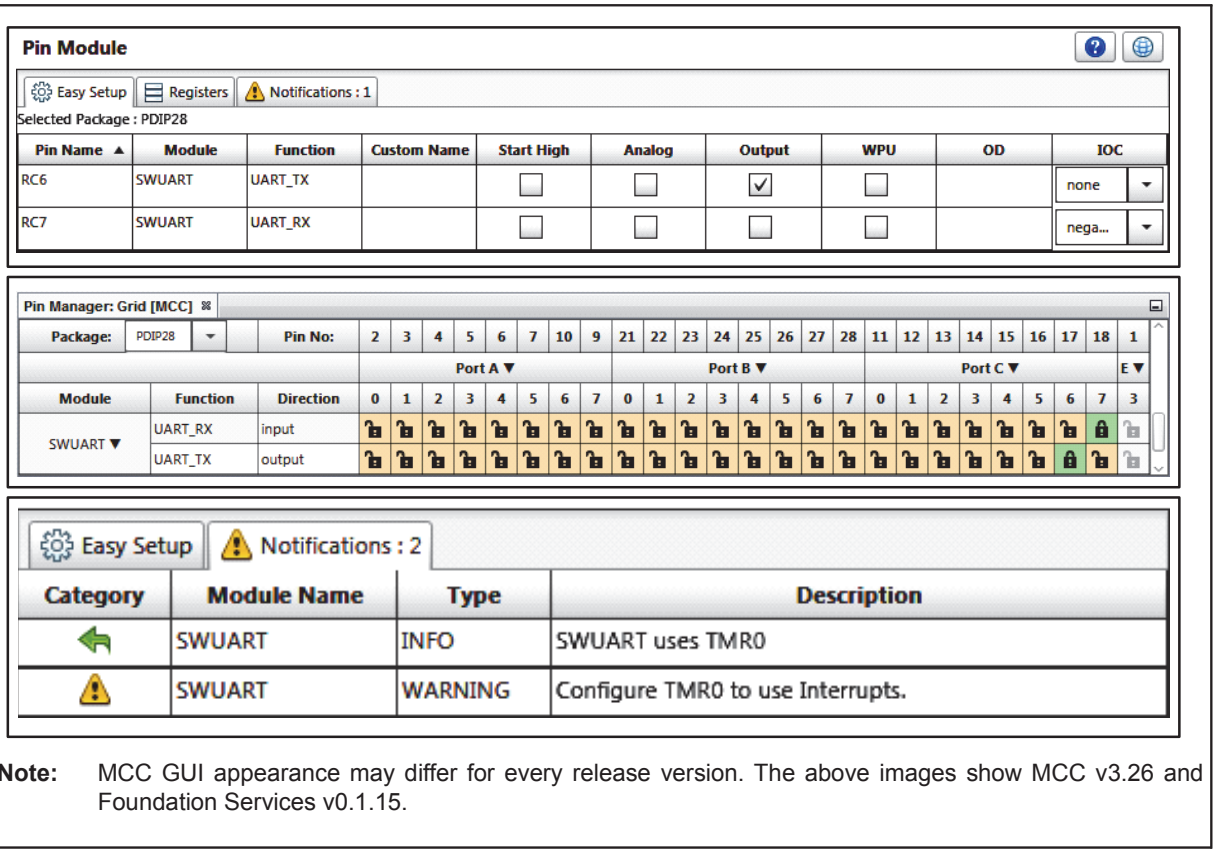


FIGURE 7: MCC PIN SETTINGS AND SWUART NOTIFICATIONS

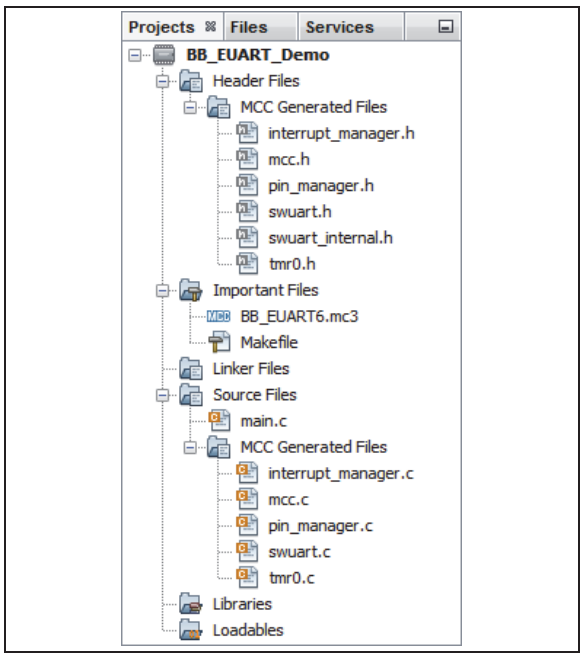


Note: MCC GUI appearance may differ for every release version. The above images show MCC v3.26 and Foundation Services v0.1.15.

MCC-Generated Files

Figure 8 shows all the files that will automatically be added to the main project upon generation in MCC.

FIGURE 8: MCC-GENERATED FILES



Two header files and one C file will be generated for the bit-banged EUART.

- `swuart.h` – This file contains all the variable declarations and functions prototypes for the basic functionalities of a UART such as initialization, read, and write.
- `swuart_internal.h` – Contains all macros necessary to mimic the behavior of the hardware EUSART in half-duplex Asynchronous mode.
- `swuart.c` – Contains variable declarations and functions to support the different basic and enhanced capabilities of the bit-banged EUART.

Two important values in `swuart.c` may need to be adjusted by the user to compensate for the hardware and software-introduced delays.

- `ONE_BIT_DELAY_COMP` – To compensate for delays during one-bit transmission and one-bit sampling intervals. This might need to be modified especially when using oscillators rather than INTOSC.
- `HALF_BIT_DELAY_COMP` – To compensate for delays during Start bit sampling. This might need some adjustments especially when the read functionality is to be utilized. For write only, this value will not matter.

During initialization, the new Timer0 reload values are automatically calculated based on the set compensator values (see Section , Timer0 Reload Adjustment). Automatic calibration will be supported by the MCC SWUART driver in future releases.

The Main File

For new projects, you will notice that the MCC automatically generates the `main.c` file. The following final steps must be performed for the driver to be properly initialized:

1. Open the MCC-generated main.c file.
2. Uncomment the `INTERRUPT_PeripheralInterruptEnable();` and `INTERRUPT_GlobalInterruptEnable();` lines of code.
3. The driver is now ready to use. You may now add your application code inside the `while(1)` loop.

Now that the user already knows how to setup the driver using MPLAB X and MCC, he/she may find the subsequent discussions useful to gain a better understanding on how the driver really works. The next sections will cover the firmware design and all the basic details about the driver.

FIRMWARE

The bit-banged EUART is built almost entirely in software and only a few hardware resources are used. This section will give an overview on how these resources are utilized to create a complete stand-alone driver.

Transmit

Figure 9 shows a simplified flowchart on how the bit-banged EUART transmits data asynchronously through the TX pin after a valid write.

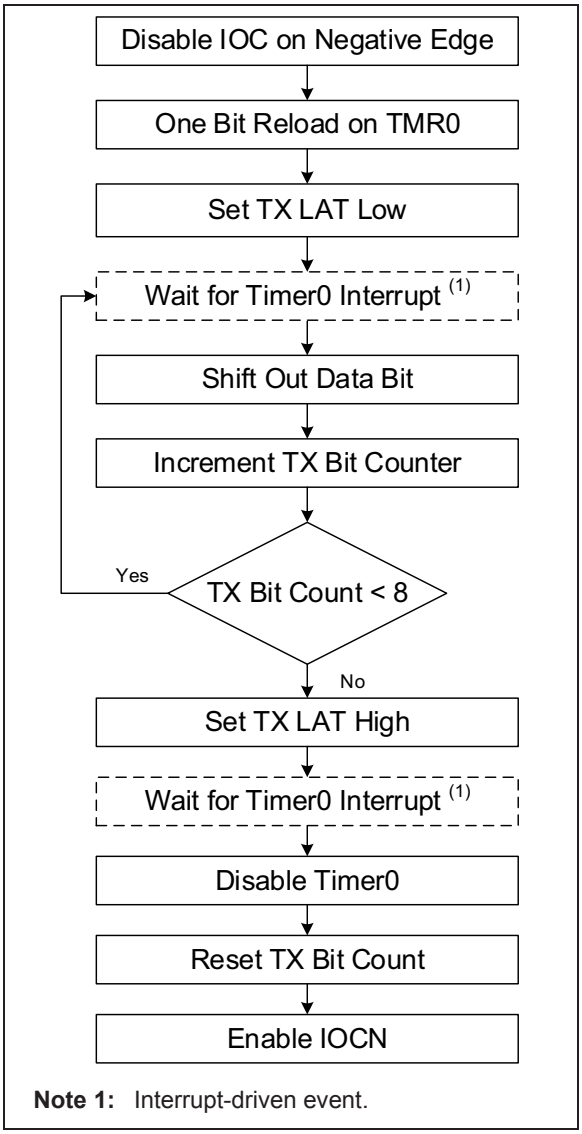
In a half-duplex concept, communication is two-way but no two devices are allowed to transmit simultaneously. Thus, the first step before transmission in the bit-banged EUART is to disable IOC on the RX pin. Asynchronous transmission commences when TMR0 is reloaded to interrupt after exactly one bit period and the TX pin is pulled low to transmit a Start bit. The driver is interrupt-driven to allow the processor to perform other important tasks while waiting for an interrupt. When Timer0 overflows, an interrupt occurs. The character bits are shifted out of the TX pin one at a time on every Timer0 interrupt, from LSb to MSb. The pin is eventually set high after transmission of the eighth bit. It is left high for another one bit period for Stop bit transmission. After a complete data transmission, Timer0 is disabled and IOCN is enabled to allow reception.

Receive

Figure 10 depicts how data is being received asynchronously through the RX pin.

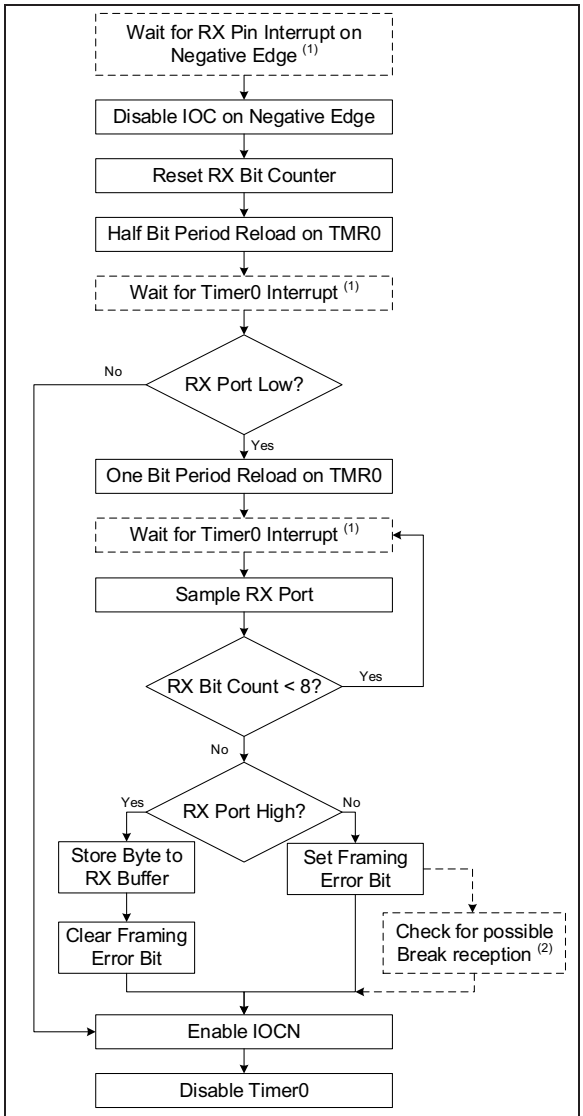
Asynchronous reception starts when an IOC on the negative edge of the RX pin is detected. This means that a high-to-low transition is detected on the RX pin signaling a Start bit edge. The IOCN is then disabled and TMR0 is reloaded to interrupt after a half bit period. After the first Timer0 interrupt, the RX pin is sampled and tested if a low is detected. If yes, TMR0 is reloaded to interrupt on every one bit period to sample the incoming eight data bits and the Stop bit. If a Stop bit is received, the received data is stored in the RX FIFO buffer and the Framing Error bit is cleared. Otherwise, the Framing Error bit will be set. The IOCN is then enabled and Timer0 is disabled to allow another reception.

FIGURE 9: TRANSMIT FLOWCHART



Note 1: Interrupt-driven event.

FIGURE 10: RECEIVE FLOWCHART



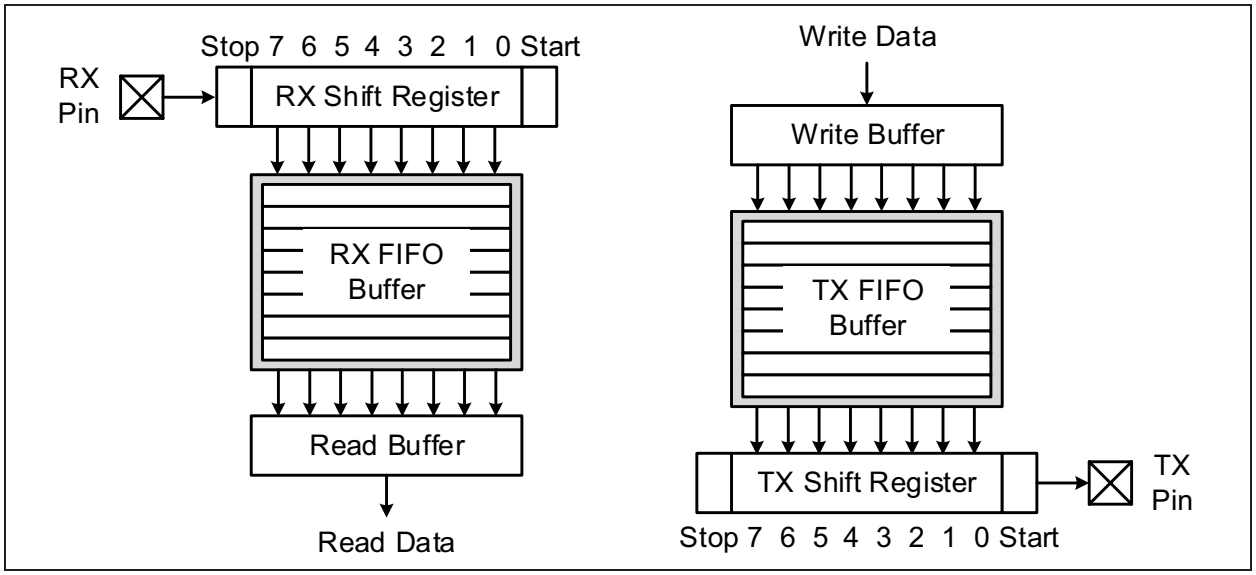
Note 1: Interrupt-driven event.

2: For LIN slave nodes only.

Data Read and Write

Figure 11 shows how data is being processed by the firmware before a read (left) and after a write (right).

FIGURE 11: READ AND WRITE



A valid read occurs only when the Read buffer is not empty. Data bits are shifted into the RX Shift Register through the RX pin. The data is automatically transferred to the RX FIFO buffer after a Stop bit is detected. The RX FIFO stores all unread data. During a read, the top unread character in the FIFO is immediately transferred to the Read buffer and the data is now available for use to other applications. The bit-banged EUART firmware disregards all subsequent incoming data when the RX FIFO buffer is loaded with unread characters.

Data can only be written directly to the Write buffer. When both the TX FIFO buffer and the TX Shift Registers are empty, data is automatically flushed from the Write buffer to the TX Shift Register. If not, all written data are queued in the TX FIFO buffer. The top unsent data is immediately transferred to the TX Shift Register after a Transmit is complete. When a write occurs while the TX FIFO buffer is loaded with unsent characters, the data stays at the Write buffer until the Stop bit is shifted out of the TX pin.

In addition to buffer control, error handling mechanisms are also implemented in the firmware.

Error Handling

Three possible errors may occur during receive: False Start, Framing Error, and Overrun Error.

FALSE START

A False Start occurs when an IOC on the negative edge of the RX pin is detected but a Start bit is not received. Whenever a False Start is detected, the serial driver state is set to Idle, and the firmware prepares for reception of another character.

FRAMING ERROR

Framing Error occurs when the Stop bit is not received at the expected time. Once a Framing Error is detected, `SW_FERR` is set. The firmware then checks for a possible Break character reception. `SW_FERR` is cleared on the next read.

OVERRUN ERROR

An overrun error occurs when the RX FIFO buffer is written while it is loaded with unread characters. `SW_OERR` is set once the buffer is full to prevent further receive. It is automatically cleared by firmware once the top unread character is read.

DRIVER DETAILS

This section describes the different software functions, flag bits, FIFO buffers, shift registers, and driver states implemented in the driver.

Functions

The bit-banged EUART is composed of firmware functions that can be broken up into the following:

- Initialization and Control – Includes configuration, read, write, control, and error handling functions.
- Status Checking – These functions are called to determine the status of the bit banged EUART transmitter and receiver.
- Interrupt-on-Change Handling – Handles interrupt events for positive and negative edge detection on the RX pin.
- Timing – Involves calculation, adjustment and setting of reload values, enabling/disabling of Timer0, and Timer0 interrupt handling.
- MCC LIN Support – Miscellaneous functions to support auto-baud detection and Break reception for LIN slave nodes and Break transmission for LIN master nodes. These functions enable the bit-banged EUART to be implemented with the MCC LIN library in place of the hardware EUSART module.

TABLE 1: FUNCTION NAMES

Function Name	Parameters	Returns	Description
Initialization and Control Functions			
SWUART_Initialize()	—	—	Performs initialization of the driver.
SWUART_Read()	—	readValue	Returns the top unread character of the RX FIFO buffer.
SWUART_Write()	txdata	-	Writes data to the TX FIFO buffer.
SWUART_EnableRx()	—	—	Disable Timer0 interrupt and enables interrupt-on-change on negative edge of the RX pin.
SWUART_CheckRx()	—	—	Check received data for framing error and/or Break reception for LIN slave nodes.
SWUART_RxOverrunControl()	—	—	Detects occurrence of an RX FIFO buffer overrun.
Status Checking			
SWUART_is_tx_ready	—	bool	Returns true if there is still space in the transmit FIFO. Otherwise, returns false.
SWUART_is_rx_ready	—	bool	Returns true if RX FIFO is not empty. Otherwise, returns false.
SWUART_is_tx_done	—	bool	Returns true if all characters have been shifted out of the TX shift register. Otherwise, returns false.
Interrupt-on-Change Handling Functions			
SWUART_SetIOCNHandler()	SWUART_IOCNHandler	—	Assigns SWUART_IOCNHandler() as part of the Interrupt Service Routine (ISR) for every interrupt on the negative edge of the selected RX pin.
SWUART_IOCNHandler()	—	—	Identifies whether the interrupt is for the start of reception of a new byte or a Sync character when auto-baud is enabled.
SWUART_SetIOCPHandler()	SWUART_IOCPHandler	—	Assigns SWUART_IOCPHandler() as part of the Interrupt Service Routine (ISR) for every interrupt on the positive edge of the selected RX pin.
SWUART_IOCPHandler()	—	—	Called within the IOC on positive edge ISR. For Sync character reception and auto-baud detection.
Timing Functions			
SWUART_SetTimerInterruptHandler()	—	—	Assigns SWUART_TimerInterruptHandler() as part of the Interrupt Service Routine (ISR) for Timer0 interrupt.
SWUART_TimerInterruptHandler()	—	—	Handles the code for every Timer0 interrupt event.
SWUART_SetTimerReload()	reload	—	Writes a new reload value to the TMR0 and enables Timer0 interrupt.
SWUART_CalcOneBitReload()	—	oneBit	Returns the adjusted one bit reload value based on the set one-bit delay compensator value ONE_BIT_DELAY_COMP.
SWUART_CalcHalfBitReload()	—	halfBit	Returns the adjusted half bit reload value based on the set half-bit delay compensator value HALF_BIT_DELAY_COMP.
SWUART_SetTimerPrescaler() (1)	prescaleEnable, prescaleVal	-	Enable and set, or disable the Timer0 prescaler.
MCC LIN Support Functions			
SWUART_SetAutoBaud() (1)	—	—	For LIN slave nodes. Captures the TMR0 value after the fifth rising edge on the RX pin during Sync reception. Controls auto-baud detection.
SWUART_CalcBaudRate() (1)	periodCapture	—	For LIN slave nodes. Identifies the new one bit and half bit reload values based on the captured Sync character period.
SWUART_SendBreak()	—	—	For LIN master nodes. Initiates transmission of a Break character after a dummy write.

Note 1: Generated only when auto-baud is enabled.

Flag Bits

Table 2 summarizes the serial flag bits used in the bit-banged EUART. These flag bits perform similar functionalities with their hardware EUSART counterparts.

TABLE 2: SERIAL FLAG BITS

Flag Bit	Description	Function	Hardware Equivalent
SW_TRMT	Transmit Shift Register Status bit	Indicates the status of the transmit shift register. Automatically set by firmware when the transmit shift register is empty and there are no pending characters in the TX FIFO. It is cleared when a character is flushed from the write buffer to the transmit shift register.	TRMT
SW_OERR	Overrun Error bit	Set when an RX FIFO buffer overrun occurs. Automatically cleared by firmware once the top character is read.	OERR
SW_FERR	Framing Error bit	Set when a framing error is detected during reception. Automatically cleared and set by firmware.	FERR
SW_SENDB	Send Break Character bit	Setting this bit means that a Break character will be sent on the next transmission. Automatically cleared by firmware upon completion.	SENDB
SW_ABDEN	Auto-Baud Detect Enable bit	Setting this bit enables auto-baud detection. Automatically cleared when auto-baud is complete.	ABDEN

FIFO Buffers

The bit-banged EUART uses two First-in-First-out (FIFO) buffers: Transmit and Receive. The default size of both buffers is eight but can be modified in software or through the MCC SWUART GUI.

TABLE 3: SERIAL FIFO BUFFERS

FIFO Buffer	Description	Function	Hardware Equivalent
swuartTxBuffer[]	Transmit FIFO	Where characters are written (queued) while transmission is in progress.	TXREG
swuartRxBuffer[]	Receive FIFO	Where characters are stored (queued) before being read.	RCREG

Shift Registers

Two Shift Registers are implemented on firmware, one each for transmit and receive.

TABLE 4: SERIAL SHIFT REGISTERS

Shift Register	Description	Function	Hardware Equivalent
swuartTxData	Transmit Shift Register	Shifts the data bits out of the TX pin from LSb to MSb.	TSR
swuartRxData	Receive Shift Register	Where the incoming bits from the RX pin are shifted in.	RSR

Driver States

Every Timer0 interrupt is handled by entering one of the different possible states during transmit, receive, or Idle conditions. The driver states and their definitions are summarized in Table 5. Figure 12 and Figure 13 depict the Transmit and Receive state diagrams, respectively.

TABLE 5: SERIAL DRIVER STATES

Serial State	Description
Transmit	
SERIAL_SEND_START_BIT	Entered after a valid write to the Write buffer. Pulls the line “low” for one bit period.
SERIAL_SEND_BYTE	Contains the Transmit Shift Register. Pulls the TX pin “high” after all eight bits are shifted out.
SERIAL_SEND_STOP_BIT	Prepares for another transmission if the TX FIFO buffer is not empty. Otherwise, enables receive by enabling the IOC on negative edge of the RX pin and sets the SW_TRMT flag bit to enable another transmit.
SERIAL_SEND_LIN_BRK	Entered when SW_SENDB is set and after writing a dummy byte to the TX buffer. Holds the TX pin “low” for thirteen bit periods.
Receive	
SERIAL_RCV_START_BIT	Entered half bit period after IOC on negative edge of the RX pin. Verifies if a Start bit is received.
SERIAL_RCV_BYTE	Contains the Receive Shift Register. Samples the incoming bits from the RX pin on every one bit timer interrupt. When all eight bits are shifted in, they are immediately transferred to the RX FIFO buffer.
SERIAL_RCV_STOP_BIT	Verifies if a Stop bit is received. Checks for framing error and buffer overrun error.
SERIAL_RCV_LIN_BRK	Entered when 00h is received and a framing error is detected (SW_FERR = 1).
SERIAL_RCV_LIN_SYNC	Counts the number of Timer0 overflows during auto-baud detection. Entered only when auto-baud is enabled (SW_ABDEN = 1).
Idle	
SERIAL_IDLE	Default state. Entered when neither transmit nor receive is in progress.

FIGURE 12: TRANSMIT STATE DIAGRAM

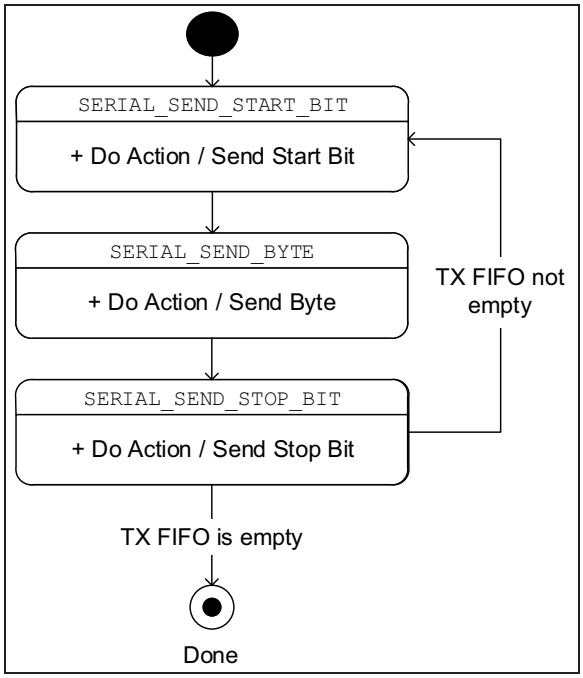
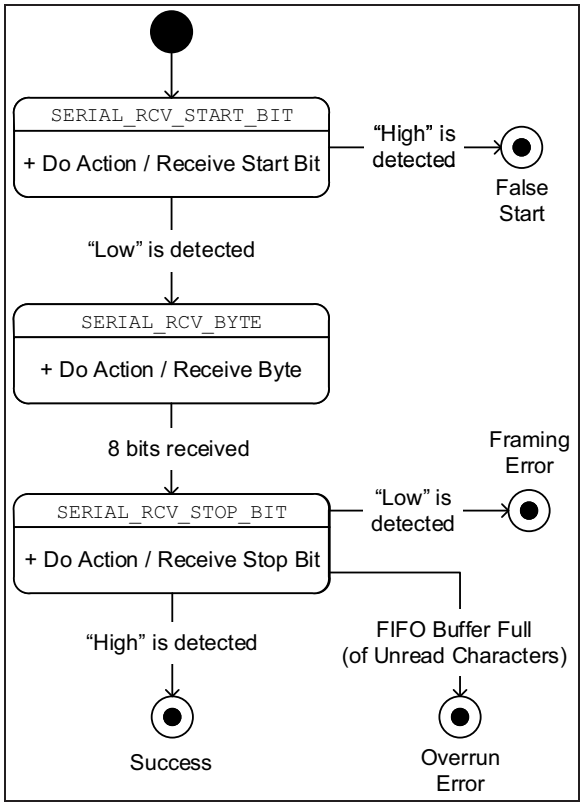


FIGURE 13: RECEIVE STATE DIAGRAM



ENHANCED FEATURES

This section describes the additional features of the bit-banged EUART to support the MCC LIN library.

Break Character Transmission

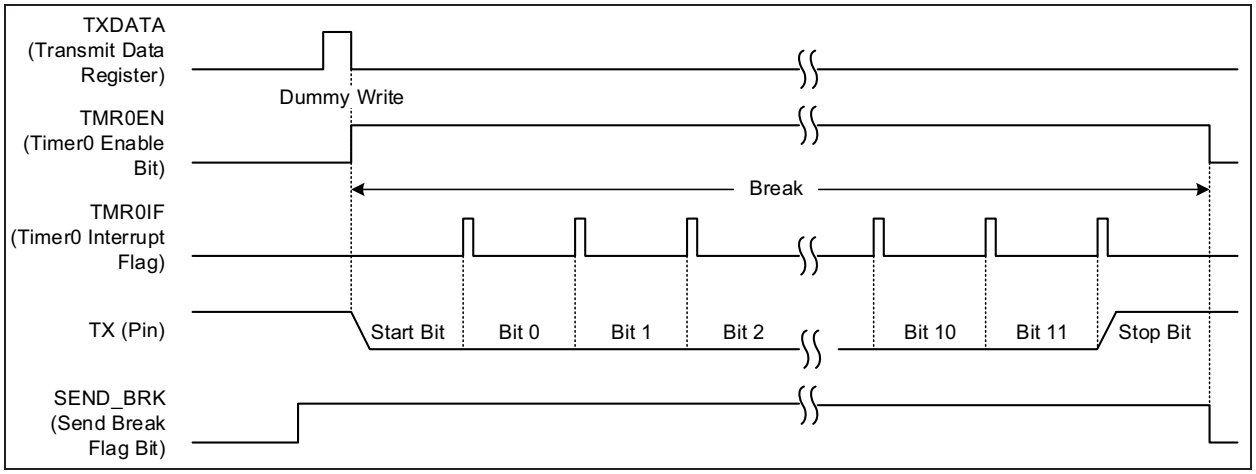
The bit-banged EUART has the capability of sending the special Break character sequence that are required by the LIN bus standard (see Figure 14). A Break character consists of a Start bit, followed by 12 '0' bits and a Stop bit. The EUART implements the Start bit as the first bit of the required 13-bit time long nominal signal and the Stop bit as the Break delimiter which is at least one bit time long of recessive signal.

To send a Break character, set the SW_SENDB serial flag bit. The value written to the Transmit Data Register will be ignored and all '0's will be transmitted. The SW_SENDB serial flag bit is automatically reset by firmware after the corresponding Stop bit is sent.

When implemented with the MCC LIN Master Driver, the bit-banged EUART performs the following sequence for Break transmission:

1. Serial Flag SW_SENDB is set to '1' to enable the Break sequence.
2. The Transmit Data Register is loaded with a dummy character to initiate transmission (the value is ignored).
3. TMR0 is loaded and TMR0IE is enabled to interrupt after one bit period.
4. TMRIF is automatically cleared on every Timer0 interrupt event.
5. TX pin is set low until the 13th interrupt in which it is eventually set high.
6. On the 14th interrupt, Timer0 interrupt is disabled and SW_SENDB is automatically cleared.

FIGURE 14: BREAK CHARACTER TRANSMISSION



Break Character Reception

Reception of the Break character follows the same flowchart in Figure 10, but with a few additional tasks inserted. The firmware checks if the received character is equal to 00h. If so, sampling of the incoming bits continue until the 13th space is received. TMR0 is assumed to have been initialized to interrupt on the expected bit period.

A Break character has been received when:

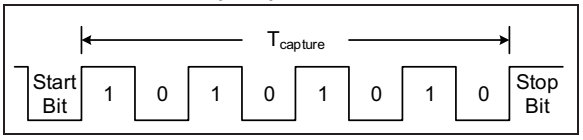
- Timer0 Interrupt is enabled (TMR0IE = 1)
- Framing Error serial flag bit is set (SW_FERR = 1)
- RX Shift Register is cleared (swuartRxData = 00h)

When using auto-baud, it is recommended for Timer0 period to be set with < ±14% deviation from the expected bit period.

Automatic Baud Rate Detection

The bit-banged EUART supports automatic detection and calibration of the baud rate. This feature is designed to support the auto-baud requirement of a LIN slave node. Timer0 module is used to capture the period of the Sync character with the value 55h (ASCII "U"). The unique feature of this character is that it has five rising edges including the Stop bit edge, as shown in Figure 15.

FIGURE 15: ASCII CHARACTER "U" (55h)



Rising edges are detected using an Interrupt-on-Change (IOC) pin. TMR0 is set to 00h soon after the first rising edge is detected. On the fifth rising edge, the timer value is captured. The timer period is based on the captured value, the prescaler and the number of timer overflows, as shown in Equation 7.

EQUATION 7: CAPTURED PERIOD USING TIMER0

$$T_{Capture} = \frac{4 \times (TMR0 + (Overflow\ Count \times 256)) \times Prescaler}{F_{OSC}}$$

To calculate the actual bit sampling time, the captured period must be divided by eight bits.

EQUATION 8: ONE BIT PERIOD

$$T_{Bit} = \frac{T_{Capture}}{8}$$

Since baud rate is equal to 1/TBIT, combining Equation 7 and Equation 8 will lead to a single equation between the detected baud rate and other previously mentioned parameters (Equation 9).

EQUATION 9: DETECTED BAUD RATE

$$Baud_{Detected} = \frac{F_{OSC} \times 8}{4 \times (TMR0 + (Overflow\ Count \times 256)) \times Prescaler}$$

For convenience, the bit-banged EUART uses a default prescaler value of eight during Sync reception. The resulting captured value will automatically correspond to a one bit period with no prescaler. The detected baud rate equation can be simplified, as shown in Equation 10.

EQUATION 10: DETECTED BAUD RATE WITH PRESCALER = 8

$$Baud_{Detected} = \frac{F_{OSC}}{4 \times (TMR0 + (Overflow\ Count \times 256))}$$

Reload and prescaler values for both one bit and half bit sampling are then calculated automatically. The bit-banged EUART calibrates the LIN slave baud rate on the first Break-Sync reception only. The relative detection error can be calculated using Equation 11.

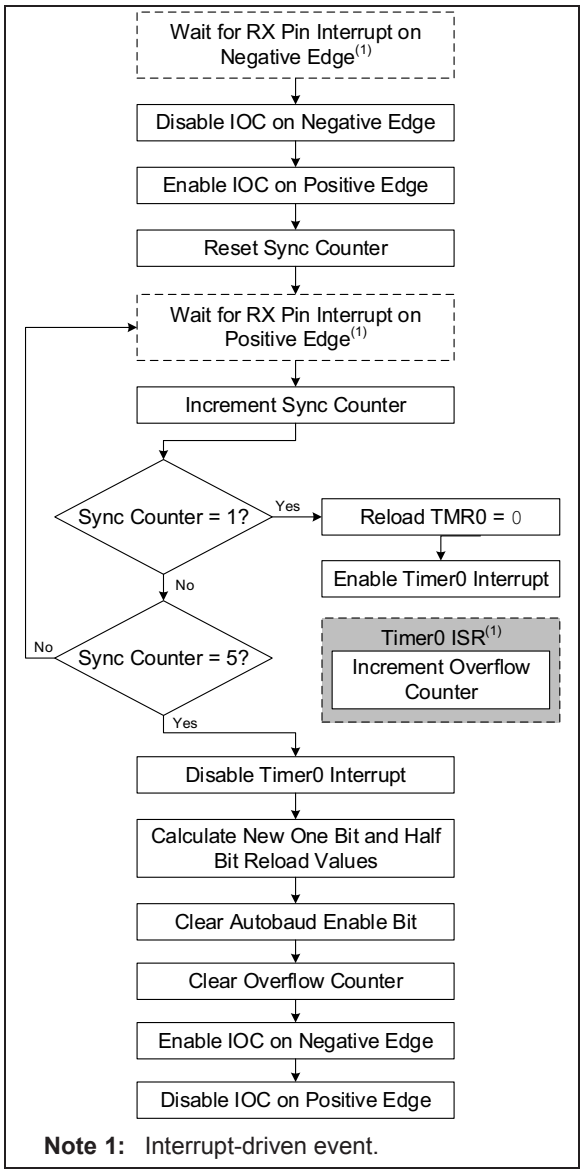
EQUATION 11: RELATIVE BAUD DETECTION ERROR

$$\%Error = \frac{Baud_{Detected} - Baud_{Input}}{Baud_{Input}} \times 100\%$$

$Baud_{INPUT}$ is the incoming baud rate (i.e., the master's baud rate for LIN bus systems). According to the LIN specification, the deviation of the slave node bit rate tolerance relative to master node bit rate after synchronization must be <±2%.

Equation 10 gives only the ideal detected baud rate. Refer to Section , Timer0 Reload Adjustment for calibration of baud rate through Timer0. A simplified flowchart of Sync character reception with auto-baud enabled is shown in Figure 16.

FIGURE 16: SYNC RECEPTION WITH AUTO-BAUD



Note 1: Interrupt-driven event.

CONCLUSION

This application note covers all the necessary information on how to implement a bit-banged EUART using the MPLAB X IDE, the MPLAB Code Configurator (MCC) and an 8-bit PIC microcontroller. It is important for the user to familiarize with the driver's features to maximize its capabilities and become aware of its limitations. Calculations and driver details are also presented for users who may want to modify the firmware for calibration and use on a variety of applications.

Only the basic features of the bit-banged EUART are demonstrated. For the enhanced features specifically designed for LIN-based applications, refer to *AN2059 "LIN Basics and Implementation of the MCC LIN Stack Library on 8-Bit PIC® Microcontrollers"* (DS00002059).

APPENDIX A: BAUD RATE VERSUS ACTUAL RATE

EXAMPLE A-1: BAUD RATE COMPARISONS USING A PIC16F1718'S INTERNAL OSCILLATOR (INTOSC)

Fosc = 32 MHz			
Baud	Actual Rate	Timer0 Prescaler	% Error
100	-	-	-
300	299.45	128	0.18%
600	598.67	64	0.22%
1.200	1195.81	32	0.35%
2.400	2398.08	16	0.08%
4.800	4793.29	8	0.14%
9.600	9638.55	4	0.40%
14.400	14466.55	4	0.46%
19.200	19370.46	2	0.89%
38.400	39215.69	-	2.12%

Fosc = 16 MHz			
Baud	Actual Rate	Timer0 Prescaler	% Error
100	99.81	256	0.19%
300	299.13	64	0.29%
600	597.64	32	0.39%
1.200	1198.32	16	0.14%
2.400	2397.36	8	0.11%
4.800	4813.48	4	0.28%
9.600	9673.52	2	0.77%
14.400	14545.45	2	1.01%
19.200	19559.90	-	1.87%
38.400	-	-	-

Fosc = 8 MHz			
Baud	Actual Rate	Timer0 Prescaler	% Error
100	99.80	128	0.20%
300	298.86	32	0.38%
600	599.21	16	0.13%
1.200	1199.04	8	0.08%
2.400	2408.19	4	0.34%
4.800	4839.69	2	0.83%
9.600	9779.95	-	1.87%
14.400	-	-	-
19.200	-	-	-
38.400	-	-	-

Fosc = 4 MHz			
Baud	Actual Rate	Timer0 Prescaler	% Error
100	99.74	64	0.26%
300	299.63	16	0.12%
600	599.30	8	0.12%
1.200	1204.28	4	0.36%
2.400	2420.57	2	0.86%
4.800	4889.98	-	1.87%
9.600	-	-	-
14.400	-	-	-
19.200	-	-	-
38.400	-	-	-

Fosc = 2 MHz			
Baud	Actual Rate	Timer0 Prescaler	% Error
100	99.59	32	0.41%
300	299.58	8	0.14%
600	602.14	4	0.36%
1.200	1209.92	2	0.83%
2.400	2443.49	-	1.81%
4.800	-	-	-
9.600	-	-	-
14.400	-	-	-
19.200	-	-	-
38.400	-	-	-

Fosc = 1 MHz			
Baud	Actual Rate	Timer0 Prescaler	% Error
100	99.93	16	0.07%
300	301.15	4	0.38%
600	605.28	2	0.88%
1.200	1222.68	-	1.89%
2.400	-	-	-
4.800	-	-	-
9.600	-	-	-
14.400	-	-	-
19.200	-	-	-
38.400	-	-	-

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. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
═ ISO/TS 16949 ═

Trademarks

ISBN: 978-1-5224-1573-2

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:
http://www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon

Hong Kong
Tel: 852-2943-5100
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8569-7000
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115
Fax: 86-571-8792-8116

China - Hong Kong SAR
Tel: 852-2943-5100
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-3326-8000
Fax: 86-21-3326-8021

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8864-2200
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

ASIA/PACIFIC

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-3019-1500

Japan - Osaka
Tel: 81-6-6152-7160
Fax: 81-6-6152-9310

Japan - Tokyo
Tel: 81-3-6880- 3770
Fax: 81-3-6880-3771

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-5778-366
Fax: 886-3-5770-955

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

Finland - Espoo
Tel: 358-9-4520-820

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

France - Saint Cloud
Tel: 33-1-30-60-70-00

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-67-3636

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Padova
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway - Trondheim
Tel: 47-7289-7561

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820