

# SirPic

## *Serial & Infra-Red Data Logger*

### *PIC Programmers Guide* v1.1



Author : Ravi Deo  
Doc Version: 1.1  
Updated: 02 April 2006

© Copyright 2005 Ravi Deo  
All Rights Reserved  
All Trademarks mentioned herein are property of their respective companies.

## Contents

1	Introduction .....	3
1.1	Purpose Of Document.....	3
1.2	Related Documents.....	3
2	Interface Comparison.....	4
3	Clocking Methods .....	5
4	PIC ASSEMBLER .....	6
4.1	Infra-Red Bit-Bang Method (TxChlr) .....	6
4.1.1	Reserve Variable Space .....	6
4.1.2	Paste "TxChlr" subroutine.....	6
4.1.3	Configure for PIC device.....	7
4.1.4	Calculating the Timing Constants .....	8
4.1.5	Display a character .....	8
4.2	Serial Bit Bang Method (TxChSer & RxChSer).....	9
4.2.1	Calculating the Timing Constants .....	9
4.3	Serial USART & ISR Method .....	10
4.4	Framed Method .....	10
5	PIC BASIC .....	12
5.1	SERIN & SEROUT Method.....	12
5.2	Framed Method .....	13
6	APPENDIX A: Sample Infra-red Bit-Bang Listing .....	14
7	APPENDIX B: Sample Serial Bit-Bang listing.....	17
8	APPENDIX C: Sample Serial USART listing .....	20

### DISCLAIMER

THE SIRPIC FAMILY OF SOFTWARE, FIRMWARE AND HARDWARE IS PROVIDED AS IS, WITH NO WARRANTY OF MERCHANTABILITY AND/OR FITNESS FOR ANY PARTICULAR TASK. THE USER ASSUMES ALL RESPONSIBILITY FOR ITS USE. ALL SOFTWARE IS INSTALLED AT USERS OWN RISK. THE AUTHOR, OR ANY OTHER SIRPIC CONTRIBUTORS ARE NOT RESPONSIBLE FOR ANY DAMAGE AND/OR LOSS OF DATA OR PROFITS CAUSED BY THIS SOFTWARE OR HARDWARE. NO REPRESENTATION OR WARRANTY IS GIVEN AND NO LIABILITY IS ASSUMED WITH RESPECT TO THE INFORMATION GIVEN IN THIS DOCUMENT.

# 1 Introduction

SirPic is a Serial and Infra-red data terminal application for the Palm PDA. It is a versatile tool for sending and receiving data traffic via a wired (RS232) or wireless (infra-red) connection.

SirPic is ideal for PIC developers who require a portable wired or wireless terminal unit for their custom PIC project.

## 1.1 Purpose Of Document

To use SirPic with a custom PIC microcontroller, requires the use of a hardware SirPic interface. This document provides details for programming the PIC microcontroller for use with the various types of SirPic Interface hardware.

## 1.2 Related Documents

SirPic programmers should also read the following documents (available from <http://www.sirpic.com>):

- SirPic User Guide
- SirPic Interface Guide

## 2 Interface Comparison

SirPic supports the following interface types:

- ❑ **IrTx** :            Wireless raw Infra-red transmitter
- ❑ **SIrTx**:            Wireless Serial to Infra red transmitter
- ❑ **SerialPIC**:        Wired serial (RS232) bi-directional interface
- ❑ **InfraPIC**:         Wireless Infra-red bi-directional interface

The following table compares the supported programming methods:

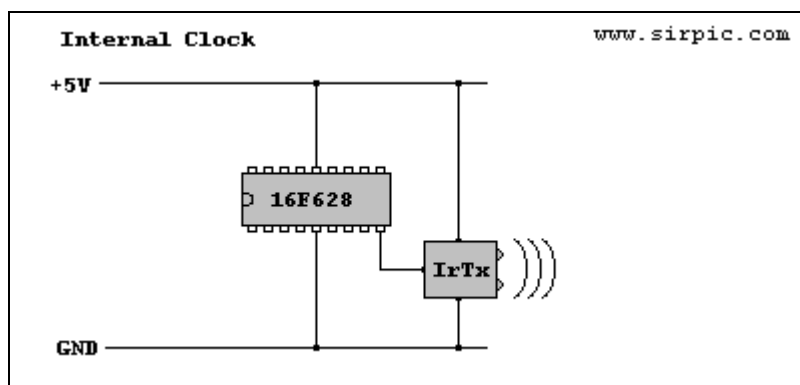
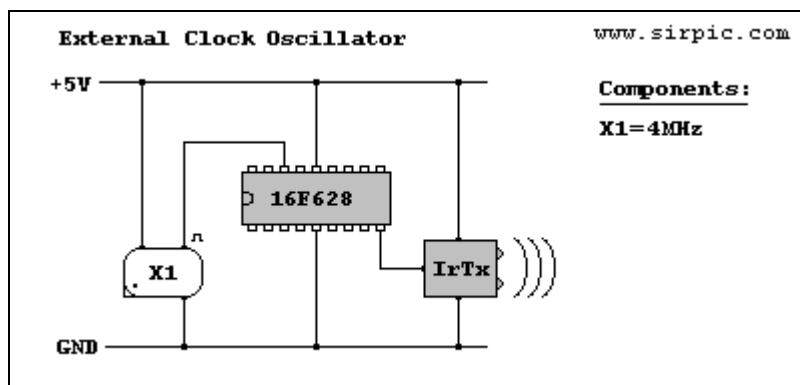
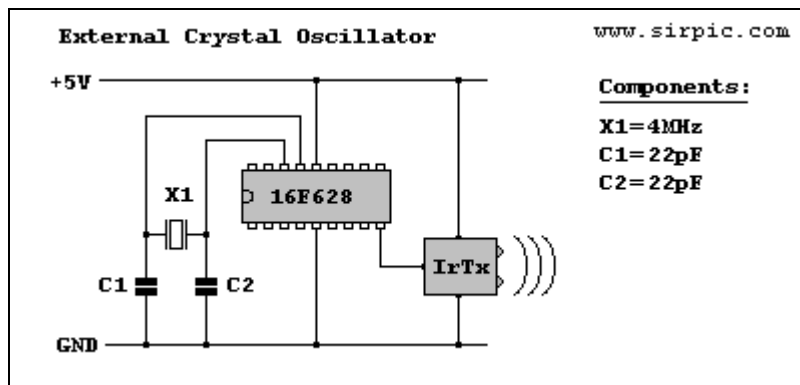
Interface Name	Connectivity	Transmit / Receive	API Support		Comments
			PIC Assembler	PIC BASIC	
<b>IrTx</b>	Wireless Infra-red (SIR)	Transmit only	Bit Bang (TxChlr)	Not Supported	Raw Infra-red bit-bang routine.
<b>SIrTx</b>	Wireless Infra-red (SIR)	Transmit only	Bit-Bang (TxChSer) & USART/ISR	<b>SEROUT, HSEROUT</b>	Supports 9600 & 19200 baud rates in both True and Inverted mode output.
<b>SerialPIC</b>	Wired Serial (RS232)	Transmit & Receive	Bit-Bang (TxChSer, RxChSer) & USART/ISR	<b>SERIN, SEROUT, HSERIN, HSEROUT</b>	Supports all RS232 baud rates in True mode output.
<b>InfraPIC</b>	Wireless Infra-red (SIR)	Transmit & Receive	BitBang (TxChSer, RxChSer) & USART/ISR	<b>SERIN, SEROUT, HSERIN, HSEROUT</b>	Supports 9600 baud rate in True mode output.

### 3 Clocking Methods

There are three possible clocking methods for use with the interfaces:

- ❑ External Crystal Oscillator circuit (2 capacitors and a crystal, or use integrated 3 pin resonator)
- ❑ External Clock in (using a clock oscillator unit)
- ❑ Internal 4 Mhz clock (crude timing will require tweaking)

It is recommended to use an external crystal oscillator or external clock to provide a stable 4MHz clock. Although it is possible to use the PICs internal 4MHz clock, this will require some tweaking. This is because the internal 4MHz clock is unstable and the actual frequency can drift widely with temperature and voltage. Some PIC devices and programmers are capable of calibrating the internal clock by modifying the OSCAL calibration register. If this is not possible with the PIC device/programmer being used, the timing constant in the PIC code to be "tweaked" to match the actual clock frequency.





```

nop
nop

BitLoop          ; This is the bit loop for each byte
  rrf    TxChar, f ; rotate right lsb data bit into f reg
  btfss STATUS, C ; if bit set skip next instruction
  call   IrPulse0  ; transmit a zero bit
  btfsc STATUS, C ; if bit clear skip next instruction
  call   IrPulse1  ; transmit a one bit
  decfsz BitCount, f ; decrement bit count, if zero skip next
  goto   BitLoop  ; Repeat for next bit
  nop      ; uniform delay for last iteration

  call   IrPulse1  ; transmit the stop bit
  call   IrPulse1  ; transmit the parity bit (not used)
  movlw  3         ; post-delay for stop & parity
  call   WaitDelay ;
  return        ; end of TxChIr subroutine

IrPulse1         ; Pulse 1 waveform: _____
  bcf    IrTxPin  ; tx 1 as a low
  movlw  FullBit  ; Wait for full bit duration
  call   WaitDelay ; for full bit duration
  return

IrPulse0         ; Pulse 0 waveform: ---_____
  bsf    IrTxPin  ; set output high
  movlw  PulseBit ; Wait for pulse bit duration
  call   WaitDelay ; keep high for pulse duration (3/16th bit)
  bcf    IrTxPin  ; set output low
  movlw  TailBit  ; Wait for tail bit duration
  call   WaitDelay ; keep low for tail duration (13/16th bit)
  return

WaitDelay        ; pause loop for timing
  movwf  Delay

LoopDelay
  decfsz Delay, f
  goto   LoopDelay
  return

; ^^^^^^^^^^^ End of SirPic subroutines ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

```

### 4.1.3 Configure for PIC device

The following initialisation code defines the physical output pin used for infra-red transmission, along with timing constants.

```

#define IrTxPin PORTB, 2 ; Infra red output pin as RB2 (pin 8)
FullBit EQU .29 ; (F) Full bit timing constant
PulseBit EQU .5 ; (P) Pulse bit timing constant
TailBit EQU .22 ; (T) Tail bit timing constant

```

The timing constants used will depend upon the selected baud rate and clock speed in use.

NOTE: The selected pin must also be initialised for output via the TRISIO (or equivalent) register, within the projects main initialisation code.

#### 4.1.4 Calculating the Timing Constants

The timing constants (F,P and T) are calculated using the following equations (derived from the instruction cycles for the above code). Using these equations, the SirPic timing constants F,P and T can be calculated from the actual clock speed and desired baud rate.

Tclk = Time for single instruction cycle

Tbit = Time for single bit duration

$$T_{\text{bit}} = \frac{1}{\text{Baud}}$$

$$T_{\text{clk}} = \frac{4}{F_{\text{osc}}}$$

$$F = \frac{T_{\text{bit}} - 16T_{\text{clk}}}{3T_{\text{clk}}}$$

$$P = \frac{T_{\text{bit}}}{16T_{\text{clk}}} - 2$$

$$T = \frac{\frac{13T_{\text{bit}}}{16} - 17T_{\text{clk}}}{3T_{\text{clk}}}$$

Due to rounding errors, pick integers F,P,T values (>0) such that following equations approximate closest to (Tbit/Tclk):

$$3P+3T+23 \geq (T_{\text{bit}}/T_{\text{clk}})$$

$$3F+17 \geq (T_{\text{bit}}/T_{\text{clk}})$$

For Example, Where Fosc=4MHz, BaudRate=9600bps

Tclk=1us, Tbit=104us

The timing constants are F=29, P=5, T=22

#### 4.1.5 Display a character

To output a character to SirPic, simply load the character code into the W register, and call the TxChIr subroutine

```
movlw 'a'           ; load W reg with character code
call TxChIr        ; transmit char via IR
```

The complete sample PIC Assembler code is listed in Appendix A, and can also be found at the PIC Developers Lab on the SirPic website (<http://www.sirpic.com>).

## 4.2 Serial Bit Bang Method (TxChSer & RxChSer)

If the PIC device you are using does not provide an onboard USART, the following bit-bang routines can be used for transmitting and receiving data (on a bit by bit basis).

The integration procedure steps are similar to the IrTx integration procedure.

APPENDIX B lists a sample “Ping Pong” 16F628 PIC application, which uses the bit-bang routines (TxChSer & RxChSer) for receiving and transmitting data. For a practical demonstration of this project, please visit the SirPic Developers Lab at [www.sirpic.com](http://www.sirpic.com).

The main loop in this sample application simply waits for a character to be received (referred to as the “ping”), adds one to the received character byte, and then transmits it as a byte of data (referred to as the “pong”).

### 4.2.1 Calculating the Timing Constants

The SerialPIC timing constants (F and H) are calculated using the following equations (derived from the instruction cycles for the bit-bang code). Using these equations, the SirPic timing constants F and H can be calculated from the actual clock speed and desired baud rate.

Tclk = Time for single instruction cycle  
Tbit = Time for single bit duration

$$T_{\text{bit}} = \frac{1}{\text{Baud}}$$

$$T_{\text{clk}} = \frac{4}{F_{\text{osc}}}$$

$$F = \frac{T_{\text{bit}} - 17 T_{\text{clk}}}{3 T_{\text{clk}}}$$

$$H = \frac{T_{\text{bit}} - 8 T_{\text{clk}}}{6 T_{\text{clk}}}$$

Due to rounding errors, pick integers F and H values (>0) such that following equations approximate closest to (Tbit/Tclk):

$$6H+8 \geq (T_{\text{bit}}/T_{\text{clk}})$$
$$3F+17 \geq (T_{\text{bit}}/T_{\text{clk}})$$

For Example, Where Fosc=4MHz, BaudRate=9600bps  
Tclk=1us, Tbit=104us  
The timing constants are F=29, H=16

## 4.3 Serial USART & ISR Method

If the PIC device in use provides an onboard USART, this can be used for asynchronous bi-directional communication. The main advantage of using the onboard USART, is that it allows the PIC application to concurrently receive/transmit serial data whilst doing other application specific processing.

APPENDIX C lists a sample “Ping Pong” 16F628 PIC application, which uses the onboard USART for receiving and transmitting data. For a practical demonstration of this project, please visit the SirPic Developers Lab at [www.sirpic.com](http://www.sirpic.com).

The integration procedure steps are similar to the IrTx integration procedure, with the additional code for the Interrupt Service Routine. Please refer to the PIC microchip reference manual for initialising the UART and setting the appropriate SPBRG value.

The main loop in this sample application does nothing (a simple forever loop), but your application could be doing useful processing here. On receiving a byte of data (referred to as the “ping”), the USART generates an interrupt which invokes the Interrupt Service Routine (ISR). The ISR checks for any receive errors and adds one to the received character byte. This is then transmitted as a byte of data, back to the sender (referred to as the “pong”).

## 4.4 Framed Method

This technique is useful for users with PDAs which do not support “raw SIR” mode, but do however support “Framed mode”. Please see the SirPic website's Compatibility page for an up to date list of PDA models and supported infra-red modes.

Framing a data packet requires insertion of a Start Of Frame (SOF = 0xC0 hex, 192 decimal) and End Of Frame (EOF = 0xC1 hex, 193 decimal) as a frame header and trailer byte, encapsulating the data packet.

To allow the SOF, and EOF characters to be contained within the data packets (as well as the frame), SirPic recognises 0x7D as the escape character. Any byte following the escape byte, is XOR'ed with 0x20, and is translated and displayed as the resultant single byte.

eg 0xC0 in the data packet = 0x7D, 0xE0 = displayed as 0xC0

eg 0x7D in the data packet = 0x7D, 0x5D = displayed as 0x7D

eg 0xC1 in the data packet = 0x7D, 0xE1 = displayed as 0xC1

For example to send “Hello” as a framed packet, the following snippet of code could be used:

```
movlw    .192                ; (Start of Frame)
call     TxChIr              ; transmit char via IR
movlw    'H'                 ; set char to output
call     TxChIr              ; transmit char via IR
movlw    'e'                 ; set char to output
call     TxChIr              ; transmit char via IR
movlw    'l'                 ; set char to output
call     TxChIr              ; transmit char via IR
movlw    'l'                 ; set char to output
call     TxChIr              ; transmit char via IR
movlw    'o'                 ; set char to output
call     TxChIr              ; transmit char via IR
movlw    .193                ; (End of Frame)
call     TxChIr              ; transmit char via IR
```

The user should also set the communication port on the receiving Palm to “(SIR)Framed” mode, which filters out the SOF and EOF characters, and performs any escape sequence translation.

## 5 PIC BASIC

SirPic supports the following commands for PIC BASIC programmers:

- SERIN / SERIN2 / HSERIN
- SEROUT / SEROUT2 / HSEROUT

The above commands (where provided by the compiler) will be fully documented in the user manual accompanying the PIC BASIC Compiler. Please refer to you PIC BASIC reference manual for syntax and use of these commands.

### 5.1 SERIN & SEROUT Method

The example snippet of code below continuously transmits "Hello Palm" with a pause between loops.

```
include "bs2defs.bas"
DEFINE CHAR_PACING 10

loop:
    SEROUT PORTB.3,N9600,["Hello Palm",10] ' send string with
carriage return (10)                    ' to pin RB3 @ 9600 baud

inverted (N)
    PAUSE 1000                            ' pause 1 second
    GOTO loop                              ' Forever
```

The next example snippet of code comprises of a simple loop which waits indefinitely for a character. The received character is incremented and transmitted back to the sender, and then loops around again.

```
include "bs2defs.bas"
DEFINE CHAR_PACING 10
char VAR byte

loop:
    SERIN  PORTB.1,T9600,char  ' wait for a character
    char = char + 1           ' increment it
    SEROUT PORTB.2,T9600,[char] ' transmit it back
    Goto loop
```

## 5.2 Framed Method

This technique is useful for users with PDAs which do not support “raw SIR” mode, but do however support “Framed mode”. Please see the SirPic websites Compatability page for an up to date list of PDA models and supported infra-red modes.

Framing a data packet requires insertion of a Start Of Frame (SOF = 0xC0 hex, 192 decimal) and End Of Frame (EOF = 0xC1 hex, 193 decimal) as a frame header and trailer byte, encapsulating the data packet.

Note: The encapsulated data packet should not contain either of these characters.

For example to send “Hello Palm” as a framed packet, the following snippet of code could be used:

```
include "bs2defs.bas"
DEFINE CHAR_PACING 10
loop:
    SEROUT PORTB.2,T9600,[192,"Hello Palm",193]
    Goto loop
```

The user should also set the communication port on the receiving Palm to “(SIR)Framed” mode, which filters out the SOF and EOF characters.





```

    call    WaitDelay          ; keep high for pulse duration (3/16th bit)
    bcf     IrTxPin           ; set output low
    movlw   TailBit           ; Wait for tail bit duration
    call    WaitDelay          ; keep low for tail duration (13/16th bit)
    return

WaitDelay
    movwf   Delay
LoopDelay
    decfsz  Delay, f
    goto   LoopDelay
    return

; ^^^^^^^^^^^ End of SirPic subroutines ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

    end                        ; end of PIC code

```

## 7 APPENDIX B: Sample Serial Bit-Bang listing

```
; *****
; File: BitBangPp_16f628.asm
; www.sirpic.com
; Author: Ravi Deo
; Version:1.0
; Description: "Bit-Bang Ping-Pong" Demo
; This sample program demonstrate use of the use of the "bit bang" routines
; to communicate. It waits for an incoming character (ping), and then transmits
; the character+1 (pong), eg receives 'b', transmits 'c'.
; Note this implementation assumes I/O is driven True (T-MODE: same as USART).
; *****

title "BitBangPp"
subtitle "Bit-Bang Ping-Pong demo"
list p=16F628, f=inhx8m ; device name and type
include <P16F628.INC> ; pic specific constants

radix dec

#define SerRxPin PORTB, 1 ; Serial Rx input pin as RB1 (pin 7)
#define SerTxPin PORTB, 2 ; Serial Tx output pin as RB2 (pin 8)

; ***** CONFIGURATION BITS *****
__config _CP_OFF & _DATA_CP_OFF & _BODEN_ON & _MCLRE_OFF & _WDT_OFF & _PWRTE_ON &
_XT_OSC & _LVP_OFF

; SirPic timing constants
; These constants are dependent upon desired baud rate and actual clock speed.
; Where:
; Tclk=4/Fosc ; Time for single instruction cycle
; Tbit=1/BaudRate ; Time for single bit duration
; F = (Tbit-(17*Tclk))/(3*Tclk)
; H = (Tbit-(8*Tclk))/(6*Tclk)
; Due to rounding errors, pick integers F,P,T values (>0) such that following
; equations approximate closest to (Tbit/Tclk):
; (3*F)+17 >= (Tbit/Tclk)
; (6*H)+8 >= (Tbit/Tclk)
; eg For Fosc=4MHz, BaudRate=9600bps, Tclk=1us, Tbit=104us
; Use values F=29, H=16

; SirPic config params for 4MHz clock at 9600 baud
SpFullBit EQU .29 ; (F) Full bit iteration count
SpHalfBit EQU .16 ; (H) Half bit iteration count

; ***** VARIABLES *****
; variable storage space starts at 0x20 in Bank 0
CBLOCK 0x20
BitCount ; Holds bit count
SpChar ; Holds transmit/receive character
Delay ; Holds Delay count
ENDC

;***** CODE SECTION *****
org 0
goto InitPic

org 0x4 ; start address 4 (same as ISR vector)
InitPic ; Initialise the PIC
bcf STATUS,RP0 ; switch to bank 0 for PORTB register
bcf STATUS,RP1
clrf PORTB ; init all RB output pins to low

bsf STATUS, RP0 ; switch to bank 1 for TRISB register
movlw B'00000010' ; Set RB1 for input
movwf TRISB ; apply to TRISB register

bcf STATUS,RP0 ; switch back to bank 0
bsf SerTxPin ; set tx pin to inactive high (T-Mode)

MainLoop ; Main receive/process/transmit loop
call RxChSer ; Wait for 'ping' character to be received
addlw 1 ; increment char by 1
```

```

        call    TxChSer                ; Transmit the 'pong' character
        goto    MainLoop              ; loop forever

; *****
; * TxChSer:  Transmits character via Serial to SirPic.          *
; * Character to be transmitted is passed in via W register.    *
; *****
TxChSer:
    bcf        STATUS,RP0             ; switch to bank 0
    bcf        STATUS,RP1

    movwf     SpChar                  ; move transmit byte in W to variable
    movlw     8
    movwf     BitCount                ; set BitCount var to 8

    bcf        SerTxPin               ; start bit = low
    movlw     SpFullBit               ; Wait for full bit duration
    call      WaitDelay
    goto      $+1                     ; extra balancing delay
    goto      $+1
    goto      $+1
    nop

TxBitLoop                ; This is the bit loop for each byte
    rrf        SpChar, f              ; rotate right lsb data bit into f register
    btfsc     STATUS, C              ; if bit set skip next instruction
    goto      TxBitHigh
    nop                                ; balancing delay

TxBitLow
    bcf        SerTxPin               ; tx 0 as a low
    goto      TxBitDelay

TxBitHigh
    bsf        SerTxPin               ; tx 1 as a high
    goto      $+1                     ; balancing delay

TxBitDelay
    movlw     SpFullBit               ; Wait for full bit duration
    call      WaitDelay
    goto      $+1                     ; balancing delay

    decfsz    BitCount, f            ; decrement bit count, if zero skip next
    goto      TxBitLoop              ; Repeat for next bit

    goto      $+1                     ; balancing delay for stop bit
    goto      $+1
    nop

    bsf        SerTxPin               ; return to high for stop birt
    movlw     SpFullBit               ; Wait for stop bit
    call      WaitDelay               ; no parity bit

    return                                ; end of TxChSer subroutine

; *****
; * RxChSer:  Recieves character via Serial to SirPic.          *
; * Character received is returned in W register.              *
; *****
RxChSer:
    bcf        STATUS,RP0             ; switch to bank 0
    bcf        STATUS,RP1
    clrf      SpChar

RxWait
    btfsc     SerRxPin                ; Wait for the start bit
    goto      RxWait                  ; loop until start bit is encountered

    movlw     SpHalfBit
    call      WaitDelay                ; Wait through half of the start bit
    movlw     8
    movwf     BitCount                ; set bit count to eight
    nop

RxNextBit
    movlw     SpFullBit+1
    call      WaitDelay                ; Wait one whole bit width
    goto      $+1                     ; delay tweak

    bcf        STATUS, C              ; clear carry flag
    btfsc     SerRxPin                ; skip next if low

```

```

    bsf     STATUS, C           ; set carry flag
    rrf     SpChar, F          ; rotate carry flag right into SpChar
    decfsz BitCount, F        ; decrement bit count, skip if zero
    goto   RxNextBit          ; loop for next bit

    movlw  SpFullBit          ; no parity bit
    call   WaitDelay          ; Wait through the stop bit
    movf   SpChar, W          ; return SpChar in W reg
    return

WaitDelay                               ; utility delay routine
    movwf  Delay

LoopDelay
    decfsz Delay, f
    goto   LoopDelay
    return

end                                     ; end of PIC code

```

## 8 APPENDIX C: Sample Serial USART listing

```
; *****
; File: UsartPp_16f628.asm
; www.sirpic.com
; Author: Ravi Deo (RD)
; (c) Copyright 2005 Ravi Deo
; Version:1.0
; Description: USART Ping Pong test
; This sample program demonstrate use of the onboard USART for
; communication. It waits for an incoming character, and then transmits
; the character+1, eg receives 'b', transmits 'c'.
; *****

title "UsartPp"
subtitle "USART ping pong demo"
list p=16F628, f=inhx8m ; device name and type
include <P16F628.INC> ; pic specific constants

radix dec ; default radix of decimal

; ***** CONFIGURATION BITS *****
__config _CP_OFF & _DATA_CP_OFF & _BODEN_ON & _MCLRE_OFF & _WDT_OFF & _PWRTE_ON &
_XT_OSC & _LVP_OFF

;***** CODE SECTION *****
org 0x0 ; start address 0
goto Init

org 0x4 ; ISR vector starts at address 4

; ***** INTERRUPT SERVICE ROUTINE *****
Isr: ; ISR handles incoming character
    btfsc RCSTA,FERR ; check for Rx framing error
    goto IsrRxError
    btfsc RCSTA,OERR ; check for Rx overrun error
    goto IsrRxError

    movf RCREG,W ; ping: read char in rcreg
    addlw 1 ; char = char+1

IsrTxChar:
    movwf TXREG ; pong: transmit char via txreg
    retfie ; return from interrupt

IsrRxError: ; ignore erroneous incoming data
    bcf RCSTA,CREN ; clear rx error 1.clear CREN
    bsf RCSTA,CREN ; clear rx error 2.set CREN
    retfie ; return from interrupt

; ***** INIT *****
Init:
    bcf STATUS,RP0 ; BANK 0
    bcf STATUS,RP1

    movlw 0x07
    movwf CMCON ; switch comparators off

    bsf STATUS,RP0 ; BANK 1

    movlw B'00000010' ; switch on RB1 as tri-state (input)
    movwf TRISB
    clrf OPTION_REG

    bsf TXSTA,BRGH ; brgh=1
    movlw .25 ; constant for 9600 baud
    movwf SPBRG

    bcf STATUS,RP0 ; BANK 0
    bsf RCSTA,SPEN ; set SPEN
    bsf RCSTA,CREN ; set CREN

    bsf STATUS,RP0 ; BANK 1
    bsf TXSTA,TXEN
```

```
bcf    TXSTA,SYNC          ; clear SYNC
bsf    PIE1, RCIE         ; enable receive interrupt

bcf    STATUS,RP0         ; BANK 0
bsf    INTCON,PEIE       ; enable peripheral interrupt
bsf    INTCON,GIE        ; enable general interrupt
```

Loop:

```
    ; you can do something more useful here!
    ; Loop gets interrupted on receiving a character
    goto Loop
```

END