INTRODUCTION

The PIC17C42 has an on-chip high speed Universal Synchronous Asynchronous Receiver Transmitter (USART). The serial port can be configured to operate either in full-duplex asynchronous mode or half duplex synchronous mode. The serial port has a dedicated 8-bit baud rate generator. Either 8- or 9-bits can be transmitted/received.

This application note provides information on using the serial port, parity generation, serial port expansion, RS-232 interface, I/O port expansion using the serial port in synchronous mode.

SERIAL PORT USAGE

A brief code to setup the serial port, and receive and transmit data is given in Example 1. Small sections of code for both asynchronous and synchronous mode are given.

EXAMPLE 1: INITIALIZATION EXAMPLE

| SPBRG  | 25 | 9600 baud @ 16 MHz input clock |
| TXSTA  | 00100000 (20h) | 8-bit transmission, async mode |
| RCSTA  | 10010000 (90h) | 8-bit reception, enable serial port, enable reception |

;************************************************************************
; Sample Code For Asynchronous Mode Serial Port Setup
;************************************************************************
#define ClkFreq 16000000 ; input clock frequency = 16 MHz
#define baud(X) ((10*ClkFreq/(64*X))+5)/10 - 1
#define TXSTA_INIT 0x90
#define RCSTA_INIT 0x90

#include "17C42.h" ; file containing the Register Definitions

Setup_Async_Mode
movl 0 ; SPBRG, TXSTA & RCSTA are in bank 0
movlw baud(9600) ; equals 25 for 9600 baud
movwf SPBRG ; baud rate generator is reset & initialized
movlw TXSTA_INIT
movwf TXSTA ; 8-bit transmission, async mode
movlw RCSTA_INIT
movwf RCSTA ; 8-bit reception, enable serial port, enable reception

return ;*************************************************************************

Asynchronous Mode Setup

Asynchronous mode setup requires selection of 8/9-bits of data transfer, baud rate, setting the baud rate generator, and configuring the TXSTA and RCSTA control registers. The baud rate generator is configured by writing the appropriate value to SPBRG register (bank0, file 17h). The value to be written to SPBRG is given by:

$$SPBRG = \frac{\text{Input}_\text{Clk}_\text{Freq}}{64 \cdot \text{Baud}_\text{Rate}} - 1$$

For example, to select a baud rate of 9600 bits/sec with input clock frequency of 16 MHz, SPBRG is computed from the above equation to be 25. Once the Baud Rate Generator is set up, it is necessary to configure the TXSTA and RCSTA control registers as shown in Example 1 (refer to the data sheet):
Synchronous Mode Setup

Synchronous mode setup requires selection of 8/9-bits of data transfer, bit rate, setting the baud rate generator, and configuring the TXSTA and RCSTA control registers. The baud rate generator is configured by writing the appropriate value to SPBRG register (bank0, file 17h). The value to be written to SPBRG is given by:

\[ SPBRG = \frac{Input\_Clk\_Freq}{4 \times Baud\_Rate} - 1 \]

For example, to select a bit rate of 1 Mbits/sec with input clock frequency of 16 MHz, SPBRG is computed from the above equation to be 3. Once the Baud Rate Generator is set up, it is necessary to configure the TXSTA and RCSTA control registers as follows (please refer to the data sheet):

\[ SPBRG = \frac{Input\_Clk\_Freq}{4 \times Baud\_Rate} - 1 \]

EXAMPLE 2: CONFIGURING TXSTA AND RCSTA CONTROL REGISTERS

<table>
<thead>
<tr>
<th>SPBRG</th>
<th>TXSTA</th>
<th>RCSTA</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>10110000 (B0h)</td>
<td>10010000 (90h)</td>
</tr>
</tbody>
</table>

SPBRG : 3 : 1 Mbits/sec @ 16 Mhz input clock
TXSTA : 10110000 (B0h) : 8-bit transmission, Sync mode (MASTER)
RCSTA : 10010000 (90h) : 8-bit reception, enable serial port, continuous reception

;*************************************************************************
; Sample Code For Synchronous Mode (MASTER) Serial Port Setup
;*************************************************************************

#include "17C42.h" ; file containing the Register Definitions
Setup_Sync_Master_Mode
  movlb 0 ; SPBRG, TXSTA & RCSTA are in bank 0
  movlw baud(1000000) ; equals 3 for 1 Mbits/sec
  movwf SPBRG ; baud rate generator is reset & initialized
  movlw TXSTA_INIT
  movwf TXSTA ; 8-bit transmission, async mode
  movlw RCSTA_INIT
  movwf RCSTA ; 8-bit reception, enable serial port, enable reception
return
;***************************************************************************
Receiving Data (Software Polling)
The sample code in Example 3 provides a way to read the received serial data by software polling (with no serial port interrupts). This applies to both asynchronous and synchronous mode. Software polling is done by checking the RCIF bit (PIR<0>). If this bit is set it means that a word has been received (8 bits are in RCREG and the 9th bit in RCSTA<0>).

**EXAMPLE 3: POLLING FOR RECEIVE DATA**

```assembly
;*******************************************************************
; Return The 8-bit received Data By Software Polling
; The received data is returned in location SerInData
;*******************************************************************
Get_Serial_Data_Poll
    movlb 1 ; PIR is in bank 1
PollRcv
    btfss PIR,0 ; check the RBIF bit
    goto PollRcv ; loop until char received, assume WDT is off
    movlb 0 ; RCREG is in bank 0
    movpf RCREG,SerInData
    return ; Received 8-bits are in SerInData
;*******************************************************************
```

Transmitting Data (Software Polling)
The sample code in Example 4 provides a way to transmit serial data by software polling (no serial port interrupts). Software polling is done by checking the bit (PIR<1> in bank 1) to be one, indicating the transfer of TXIF to the serial shift register.

**EXAMPLE 4: POLLING FOR TRANSMIT DATA**

```assembly
;*******************************************************************
; Transmit 8-bit Data By Software Polling
; The data to be transmitted is in location SerOutData
;*******************************************************************
Send_Serial_Data_Poll
    movlb 1 ; PIR is in bank 1
PollTXIF
    btfss PIR,1 ; check the TXIF bit of PIR register in bank1
    goto PollTXIF ; loop until char received, assume WDT is off
    movlb 0 ; RCREG is in bank 0
    movpf SerOutData,TXREG
    return ; Received 8-bits are in SerInData
;*******************************************************************
```
Transmitting & Receiving A Block Of Data
(Interrupt Driven)

A general purpose routine which is interrupt driven, that transmits and receives a block of data is provided in Example 5. The reception or transmission of the block is ended when an end of block character is detected. As an example, the end of block is identified by a 0. The block of data to be transmitted is stored in the program memory and the TABLRD instruction is used to transfer this example data to the file registers and serial port. The user may modify this code to a more general purpose routine that suits the application.

EXAMPLE 5: INTERRUPT DRIVEN TRANSMIT/RECEIVE

MPASM 01.40 Released       SERINT.ASM  1-22-1997  10:20:49       PAGE  1

LOC    OBJECT CODE     LINE SOURCE TEXT
       VALUE

00001 ;        TITLE   'Serial Interface Routines
00002 ;        PROCESSOR  42
00003
00004 ;This is a short program to demonstrate how to transmit and receive
00005 ;serial data using the PIC17C42.
00006 ;
00007 ;A message will be transmitted and routed right back to the processor
00008 ;and read. The read information will be saved in an internal buffer.
00009 ;
00010 ; Program:          SERINT.ASM
00011 ; Revision Date:
00012 ;                      1-22-97      Compatibility with MPASMWIN 1.40
00013 ;
00014 ;
00015 ; LIST P = 17C42
00016
00017 #include    <p17c42.inc>
00018
00019 TX_BUFFER       equ     0x80
00020 RX_BUFFER       equ     0xB0
00021 RXPTR           equ     0x20
00022 TXPTR           equ     0x21
00023 SERFLAG         equ     0x22
00024 RTINUM          equ     0x23
00025 RXDONE          equ     1
00026 TXDONE          equ     0
00027 HILOB           equ     2
00028 ;
00029 ; initialize the serial port baud rate interrupts etc.
0030  00043  init_serial
0030  00044  clrf    SERFLAG, F ;clear all flags
0031  00045  movlb  0 ;select bank 0
0032  00046  movlw  0x07 ;select 9600 baud
0033  00047  movfp  W,SPBRG ; /
0034  00048  movlw  0x90 ;set up serial pins
0035  00049  movfp  W,RCSTA ; /
0036  00050  clrf    TXSTA, F ;setup transmit status
0037  00051  movlb  1 ;select bank 1
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0038  00052  clrf    PIR, F ;clear all interrupts
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0039  00053  clrf    PIE, F ;clear all enables
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0040  00054  movlw  RX_BUFFER       ;set pointer to rx buffer
0041  00055  movpf   W,RXPTR         ; /
0042  00056  bsf     PIE,RCIE        ;enable receive interrupt
0043  00057  movlb   1               ;select bank 1
0044  00058  bsf     INTSTA, PEIE     ;enable peripheral ints
0045  00059  retfie
0046  00060  ;start transmission of first two bytes
0047  00061  start_xmit
0048  00062  movlb  0 ;select bank 0
0049  00063  tablrd  1,1,W           ;load latch
0050  00064  bsf     TXSTA,TXEN      ;enable transmit
0051  00065  tablrd  0,1,TXREG       ;load high byte
0052  00066  movlb  1 ;select bank 1
0053  00067  bsf     PIE,TXIE        ;enable transmit interrupts
0054  00068  empty_chk
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0055  00069  btsc  PIR, TXIF        ;TXBUF empty?
0056  00070  goto    empty_chk      ;no then keep checking
0057  00071  movlb  0 ;select bank 0
0058  00072  tablrd  0,1,TXREG       ;load lo byte
0059  00073  movlb  1 ;select bank 1
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0060  00074  bsf     TXSTA,TXEN      ;enable transmit interrupts
0061  00075  bsf     SERFLAG,HILOB ;set up next for high byte
0062  00076  return
0063  00077  ;PAGE
0064  00079  ;check for transmit or receive interrupts only
0065  00080  service_perf
0066  00081  service_perf
0067  00082  ;start transmission of first two bytes
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0068  00083  btsc  PIR, TXIF        ;RX buffer full?
0069  00084  service_recv      ;yes then service
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0070  00085  btsc  PIR, TXIF        ;TX buffer empty?
0071  00086  goto    exit_perf     ;no, ignore other int.
0072  00087  service_xmt
0073  00088  btsc  SERFLAG, TXDONE ;all done?
0074  00089  goto    exit_perf     ;yes then quit
0075  00090  btsc  SERFLAG, HILOB ;if clr, do low byte
0076  00091  goto    rd_hi         ;else read high byte
0077  00092  tablrd  0,1,W           ;read lo
0078  00093  goto    sx_cont      ;continue
0079  00094  rd_hi
0080  00095  tablrd  1,W           ;read high byte
0081  00096  sx_cont
0082  00097  btg     SERFLAG, HILOB ;toggle flag
0083  00098  movlb  0 ;bsr=0
0084  00099  movfp  W,TXREG        ;load tx reg
0085  00100  tstfsz  W               ;last byte?
0086  00101  goto    exit_perf     ;no then cont
005D 00102 end_xmt ;else end transmit
005D B801 00103 movlb 1 ;select bank 1

Message[302]: Register in operand not in bank 0. Ensure that bank bits are correct.
005E 8917 00104 bcf PIE,TXIE ;disable tx interrupt
005F B022 00105 bsf SERFLAG,TXDONE ;set done flag

0060 00106 exit_perf
0060 8F07 00107 bcf INTSTA,PEIF ;clear peripheral int
0061 0005 00108 retfie
00109 ;
0062 9922 00110 service_recv
0062 9922 00111 btfsc SERFLAG,RXDONE ;RX complete?
0063 C060 00112 goto exit_perf ;exit int
0064 6120 00113 movfp RXPTR,FSR0 ;get pointer
0065 B800 00114 movlb 0 ;select bank 0
0066 6014 00115 movfp RCREG,INDF0 ;load received value
0067 290A 00116 clrf WREG, F ;clr W
0068 3200 00117 opfsag INDF0 ;value = 0?
0069 C060 00118 goto exit_perf ;yes then end
006A 1501 00119 incf FSRO, F ;inc pointer
006B 4120 00120 movpf FSRO,RXPTR ;save pointer
006C C060 00121 goto exit_perf ;return from int
006D 00122 end_recv
006D 8122 00123 bsf SERFLAG,RXDONE ;set flag
006E 2907 00124 clrf INTSTA, F ;clear all int
006F B801 00125 movlb 1 ;select bank 1

Message[302]: Register in operand not in bank 0. Ensure that bank bits are correct.
0070 8817 00126 bcf PIE,RCIE ;disable rx interrupts
0071 C060 00127 goto exit_perf ;return
00128 PAGE
0072 2909 00130 ;
0072 2909 00131 start
0073 C076 00132 clrf FSR1, F ;assign FSR1 as S.P.
0074 B020 00133 decf FSR1, F ;
0075 6100 00134 movlw 0x20 ;clear ram space
0076 00135 movfp W,FSR0 ;do indirect addressing
0076 00136 start1
0076 2900 00137 clrf INDF0, F ;clear ram
0077 1F01 00138 incfsz FSRO, F ;inc and skip if done
0078 C076 00139 goto start1
0079 E030 00140 call init_serial ;initialize serial port
007A B000 00141 movlw LOW MESSAGE ;load table pointer
007B 400D 00142 movpf W,TBLPTRL ; /
007C B001 00143 movlw HIGH MESSAGE ; /
007D 400E 00144 movpf W,TBLPTRH ; /
007E E040 00145 call start_xmit ;start transmission
007F 00146 chk_end
007F 9122 00147 btfss SERFLAG,RXDONE ;receive all?
0080 C07F 00148 goto chk_end ;no then keep checking
0080 00149 ;
0081 C081 00150 loop goto loop ;spin wheel
00152 ORG 0x100
0100 00153 MESSAGE
0100 5468 6520 636F 00154 DATA "The code is: Tea for the Tillerman"
6465 2069 733A
2054 6561 2066
6F72 2074 6865
2054 696C 6C65
726D 616E
0111 0000 00155 DATA 0
00156 ;
00157 ;
00158 END
MEMORY USAGE MAP ('X' = Used, '-' = Unused)

0000 : X----------------- ---------------- X----------------- XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XX----------------- ---------------- ---------------- ----------------
0100 : XXXXXXXXXXXXXXXX XX----------------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used: 102

Errors : 0
Warnings : 0 reported, 0 suppressed
Messages : 9 reported, 0 suppressed
PARITY GENERATION

Since the serial port of the PIC17C42 does not have an on-chip parity generator, parity is generated using software. It takes only 10 program memory words and executes in 10 instruction cycles to generate parity. Since the serial port of the PIC17C42 can operate in a 9-bit mode, the parity bit can be generated in software and transmitted as the 9th bit or it can be compared with the received 9th bit.

In case of transmission, set TX9 to 1 (TXSTA < 6>) to enable 9-bit transmission and write the computed parity bit to TX9D (TXSTA<0>). The 9th bit (parity bit) must be written prior to writing the 8 data bits to TXREG.

In case of a reception, first of all enable 9-bit reception by setting RX9 to 1 (RCSTA<6>). Upon successful reception, the 9th bit is received in RX9D (RCSTA<0>). Parity of the 8 bits of received data is computed using the routine listed below and compared with the 9-bit received.

EXAMPLE 6: PRIORITY GENERATION

```assembly
LIST OBJECT CODE LINE SOURCE TEXT
VALUE

00001         TITLE "Generate Parity Bit"
00002 ;
00003 ;***********************************************************************
00004 ;       Generate Parity Bit for the 8 bit register 'txmt'
00005 ;       The parity bit is stored in Bit 0 of 'parity'
00006 ;
00007 ;
00008 ;       Program:          PARITY.ASM
00009 ;       Revision Date:  1-22-97  Compatibility with MPASMWIN 1.40
00010 ;
00011 ;
00012 ;***********************************************************************
00013 ;
00014         LIST            P = 17C42
00015
00016         #include        <p17c42.inc>
00017
00018         #define TRUE    1
00019         #define FALSE   0
00020
00021 #define ODD_PARITY      FALSE
00022 ;
00023
00024         CBLOCK 0x20
00025                 txmt, parity
00026         ENDC
00027
00028         swapf txmt,W
00029         xorwf txmt,W
00030         movwf parity
00031         rrncf parity, F
00032         rrncf parity, F
00033         xorwf parity,W
00034         andlw 0x03
00035         addlw 0x01
00036         rrncf WREG, F
00037         movwf parity
```

Please check the Microchip BBS for the latest version of the source code. Microchip’s Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).
MEMORY USAGE MAP ('X' = Used, '−' = Unused)

0000 : XXXXXXXXXXXX------ ---------------- ---------------- ----------------
All other memory blocks unused.

Program Memory Words Used:  10

Errors :   0
Warnings : 0 reported,  0 suppressed
Messages : 0 reported,  0 suppressed
SERIAL PORT EXPANSION

The PIC17C42 has only one serial port. For applications that require the PIC17C42 to communicate with multiple serial ports, a scheme that multiplexes and demultiplexes the RX and TX pins is provided below. This method is suitable only if no more than one USART is needed at any one time. This is the case in many applications where the microcontroller drives several output devices serially. Figure 1, shown below suggests a way to expand the on-chip serial port to four serial ports. To use the scheme shown in Figure 1, the PIC17C42 must select the desired serial port by appropriately setting the two pins of PORTB. The same scheme may be used to further expand the serial ports by using more I/O Ports.

FIGURE 1: MULTIPLEXING THE ON-CHIP USART

RS-232 INTERFACE

Two circuits are provided to interface the CMOS levels of a PIC17C42 to the RS-232 levels. Figure 2 provides an interface to MAX232 (MAXIM's RS-232 Driver/Receiver) with a single +5V power supply. Figure 3 provides a low cost two-chip solution for the RS-232 level translation using a single +5V supply (Note that V- of MC14C88 is connected to DTR of RS-232 Interface. By asserting DTR to low, V- gets the negative voltage from the RS-232 line). An alternative single chip low cost solution is provided in Figure 4. However, 3V sources (+5, +12, -12) are necessary.

FIGURE 2: RS-232 INTERFACE TO MAX232

FIGURE 3: LOW-COST TWO-CHIP SOLUTION USING SINGLE POWER SOURCE

FIGURE 4: LOW-COST SINGLE CHIP SOLUTION USING THREE-POWER SOURCES
Table 1 provides the summary of RS-232 and V.28 Electrical Specifications.

**TABLE 1: SUMMARY OF RS-232C AND V.28 ELECTRICAL SPECIFICATIONS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Specification</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Driver Output Voltage</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 level</td>
<td>+5V to +15V</td>
<td>With 3-7 kΩ load</td>
</tr>
<tr>
<td>1 level</td>
<td>-5V to -15V</td>
<td>With 3-7 kΩ load</td>
</tr>
<tr>
<td>Max. output</td>
<td>±25V</td>
<td>No load</td>
</tr>
<tr>
<td>Receiver Input Thresholds (Data and clock signals)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 level</td>
<td>+3V to +125V</td>
<td></td>
</tr>
<tr>
<td>1 level</td>
<td>-3V to -25V</td>
<td></td>
</tr>
<tr>
<td>Receiver Thresholds RTS, DSR, DTR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>On level</td>
<td>+3V to +25V</td>
<td>Detects power Off condition at driver</td>
</tr>
<tr>
<td>Off level</td>
<td>Open circuit or -3V to -25V</td>
<td></td>
</tr>
<tr>
<td>Receiver Input resistance</td>
<td>3 kΩ to 7 kΩ</td>
<td></td>
</tr>
<tr>
<td>Driver Output resistance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power off condition</td>
<td>300Ω Min.</td>
<td>VOUT &lt; ±2V</td>
</tr>
<tr>
<td>Driver slew rate</td>
<td>30V/μs max.</td>
<td>3 kΩ &lt; RL &lt; 7k 0pF &lt; CL &lt; 2500 pF</td>
</tr>
<tr>
<td>Signalling rate</td>
<td>Up to 20K bits/sec.</td>
<td></td>
</tr>
<tr>
<td>Cable length</td>
<td>50'/15m. Rec.</td>
<td>Longer cables permissible, if CLOAD ≤ 2500 pF</td>
</tr>
</tbody>
</table>

**I/O PORT EXPANSION USING SYNCHRONOUS MODE**

Although the PIC17C42 has 33 I/O pins, most of these are multiplexed with other peripheral functions. In cases where more I/O ports are needed, the scheme below expands the I/O port using the synchronous mode of serial port by serially shifting the data. Figure 5 shows a scheme to expand the output ports to 16 bits using two standard logic chips (74HC595). The PIC17C42's serial port is configured in synchronous mode and set to be the MASTER. Thus serial data is available on DT (pin 22) and the clock is available on CK (pin 21). The following code will transmit 16 bits serially and clock all the 16 bits at the same time.

**FIGURE 5: OUTPUT PORT EXPANSION USING SYNCHRONOUS MODE**

**EXAMPLE 7: PORT IMPORT EXPANSION**

```assembly
InitSerialPortTxmt
movlb 0
clrf SPBRG ; set to highest baud rate = CLKOUT = CLKIN/4
movlw 0x80
movwf RCSTA ; enable serial port
movlw 0xB0
movwf TXSTA ; 8 bit synchronous master mode
bcf DDRB,0 ; set bit 0 of PortB as output, to be used as Latch Clk
return
SendSerialData ; shift out DataLo & DataHi serially
movlb 0
movfp DataLo,TXREG
nop ; wait for TXREG transfer : if slower baud rate is
    ; used check for TBMT reset
movfp DataHi,TXREG
wait btfs s TXSTA,TRMT
bcf PortB,0
bsf PortB,0 ; clock in the serial data to parallel output of HC595
return
```
A similar scheme as shown above may be implemented in a reverse manner to expand the input ports. This requires a Parallel In and Serial Out device. Using a standard logic chip (74HC165), the scheme is shown in Figure 6. In order to read the 16-bit input data, the serial port of the PIC17C42 is configured for Synchronous Mode Reception (MASTER mode). An I/O port (PORTB<0>) is used to parallel load the 16 inputs for reading serially. A sample code to read the 16 inputs is shown below.

**FIGURE 6: INPUT PORT EXPANSION USING SYNCHRONOUS MODE**

![Diagram showing input port expansion using synchronous mode](image)

**EXAMPLE 8:**

```assembly
InitSerialPortRcv
    movlb 0
    clrf SPBRG ; set to highest baud rate = CLKOUT = CLKIN/4
    movlw 0x80
    movwf RCSTA ; enable serial port
    movlw 0x90
    movwf TXSTA ; 8-bit synchronous master mode
    bcf DDRB,0 ; bit 0 of PortB is output, to be used as Parallel Load
    bsf PortB,0 ; disable parallel Load
    return

ReadSerialData
    ; shift out DataLo & DataHi serially
    movlb 0
    bcf PortB,0 ; Parallel Load The Inputs into 74HC165
    bsf PortB,0 ; disable parallel Load
    bsf RCSTA,SREN ; enable single byte reception and wait for data
    movlb 1
    wait1 btfs PIR,RBIF
    goto wait1 ; check until 8-bits are received
    movlb 0
    movpf RCREG,DataLo ; 1st byte is read
    movlb 0
    bsf RCSTA,SREN ; enable another byte of reception and wait for data
    movlb 1
    wait2 btfs PIR,RBIF
    goto wait2 ; check until 8-bits are received
    movlb 0
    movpf RCREG,DataHi ; 2nd byte is read
    return
```

© 1997 Microchip Technology Inc.
Note the following details of the code protection feature on PICmicro® MCUs.

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

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

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

Trademarks

The Microchip name and logo, the Microchip logo, FilterLab, KEELOQ, microID, MPLAB, PIC, PICmicro, PICMASTER, PICSTART, PRO MATE, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

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

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.