



Software Implementation of Asynchronous Serial I/O

INTRODUCTION

The PIC16CXX microcontrollers from Microchip Technology, Inc., are mid-range, high performance EPROM based 8-bit microcontrollers. Some of the members of this series (like PIC16C71 and PIC16C84) do not have an on-chip hardware asynchronous serial port. This application note describes the Interrupt driven Software implementation of Asynchronous Serial I/O (Half Duplex RS-232 Communications) using PIC16CXX microcontrollers. These microcontrollers can operate at very high speeds with a minimum of 250 ns cycle time (with input clock frequency of 16 MHz). To test the RS-232 routines, a simple Digital Volt Meter (DVM)/Analog Data Acquisition Systems has been implemented using PIC16C71 in which upon reception of a command from host (IBM® PC), an 8-bit value of the selected A/D channel is transmitted back to host.

IMPLEMENTATION

A half duplex Interrupt driven software implementation of RS-232 communications using PIC16C71 is described in detail below. The transmit pin used in the example code is RB7 and receive pin is connected to RTCC/RA4 pin (see Figure 2). Of course these pins are connected with appropriate voltage translation to/from RS-232/CMOS levels. The voltage translation is given described with schematics in the hardware section of this application note.

Transmit Mode

The transmit mode in software is quite straight forward to implement using interrupts. Once the input clock frequency and baud rate is known, the number of clock cycles per bit can be computed. The on-chip Real Time Clock Counter (RTCC) along with the prescaler can be used to generate interrupt on RTCC overflow. This RTCC overflow interrupt can be used as timing to send each bit. The Input clock frequency (“_CkIn”) and the Baud Rate (“_BaudRate”) are programmable by the user and the RTCC time-out value (the period for each bit) is computed at assembly time. Whether the prescaler must be assigned to RTCC or not is also determined at assembly time. This computation is done in the header file “rs232.h”. Note that very high speed transmissions can be obtained if transmission is done with software delays instead of every interrupt driven, however, the processor will be totally dedicated to this job.

Transmission of a byte is performed by calling “PutChar” function and the data byte in the “TxReg” is transmitted out. Before calling this function (“PutChar”), the data must be loaded into TxReg and also made sure that serial port is free. The serial port is free when both _txmtProgress and _rcvOver bits are cleared (see description of these bits in the Serial Status/Control Reg table given later).

Summary of “PutChar” function :

- 1) Make sure _txmtProgress & _rcvOver bits are cleared
- 2) Load TxReg with data to be transmitted
- 3) CALL PutChar function

Receive Mode

The reception mode implementation is slightly different from the transmit mode. Unlike the transmit Pin (TX pin in the example code is RB7, but could be any I/O pin), the receive pin (RX Pin) must be connected to RTCC/RA4 Pin. This is because in reception, the Start Bit which is asynchronous in nature, must be detected. To detect the start bit, when put in Reception mode, the RTCC module is configured to counter mode. The OPTION register is configured so that RTCC module is put in counter mode (increment on external clock on RTCC/RA4 Pin) and set to increment on falling edge on RTCC/RA4 pin with no prescaler assigned. After this configuration setup, RTCC (File Reg 1) is loaded with 0xFF. A falling edge on RTCC Pin will make RTCC roll over from 0xFF to 0x00, thus generating an interrupt indicating a Start Bit. The RTCC/RA4 pin is sampled again to make sure the transition on RTCC is not a glitch. Once the start bit has been detected, the RTCC module is reconfigured to increment on internal clock and the prescaler is assigned to it depending on input master clock frequency and the baud rate (configured same way as the transmission mode).

The software serial port is put in reception mode when a call is made to function “GetChar”. Before calling this function make sure serial port is free (i.e. _txmtProgress and _rcvOver status bits must be 0). On completion of reception of a byte, the data is stored in RxReg and _rcvOver bit is set to 0.

Summary of “GetChar” function:

- 1) Make sure _txmtProgress & _rcvOver bits are cleared
- 2) CALL GetChar function
- 3) The received Byte is in TxReg after _rcvOver bit is cleared

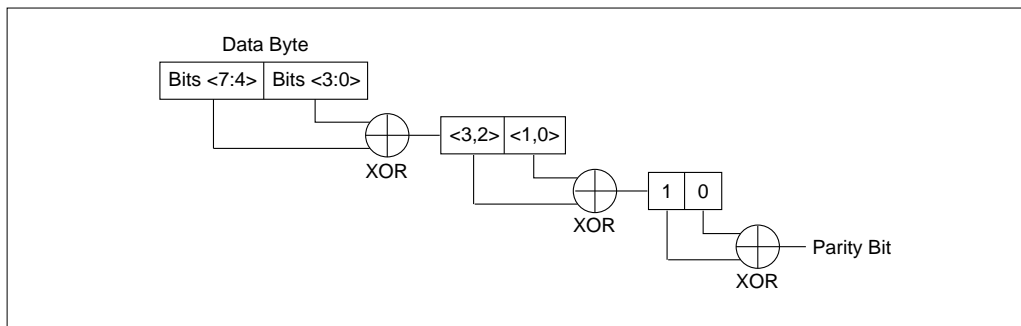
Software Implementation of Asynchronous Serial I/O

Parity Generation

Parity can be enabled at assembly time by setting “_PARITY_ENABLE” flag to TRUE. If enabled, the parity can be set to either EVEN or ODD parity. In transmission mode, if parity is enabled, the parity bit is computed and transmitted as the ninth bit. On reception, the parity is computed on the received byte and compared to the ninth bit received. If a match does not occur the parity error bit is set in the RS-232 Status/Control

Register (_ParityErr bit of SerialStatus reg). The parity bit is computed using the algorithm shown in Figure 1. This algorithm is highly efficient using PIC16CXX’s SWAPF and XORWF instructions (with ability to have the destination as either file register itself or W register) and the sub-routine (called “GenParity”) is in file “txmtr.asm”.

FIGURE 1 - AN EFFICIENT PARITY GENERATION SCHEME IN SOFTWARE



Assembly Time Options

The firmware is written as a general purpose routines and the user must specify the following parameters before assembling the program. The Status/Control register is also described below:

TABLE 1 - LIST OF ASSEMBLY TIME OPTIONS

_CkIn	Input clock frequency of the processor.
_BaudRate	Desired Baud Rate. Any valid value can be used. The highest Baud Rate achievable depends on Input Clock Freq. 600 to 4800 Baud was tested using 4 MHz Input Clock. 600 to 19200 Baud was tested using 10 MHz Input Clock. Higher rates can be obtained using higher Input Clock Frequencies. Once the _BaudRate & _CkIn are specified, the program automatically selects all the appropriate timings.
_DataBits	Can specify 1 to 8 data bits.
_StopBits	Limited to 1 Stop Bit. Must be set to 1.
_PARITY_ENABLE	Parity Enable Flag. Set it to TRUE or FALSE. If PARITY is used, then set it to TRUE, else FALSE. See “_ODD_PARITY” flag description below.
_ODD_PARITY	Set it to TRUE or FALSE. If TRUE, then ODD PARITY is used, else mEVEN Parity Scheme is used. This Flag is ignored if _PARITY_ENABLE is set to FALSE.
_USE_RTSCS	RTS & CTS Hardware handshaking signals. If set to FALSE, no hardware handshaking is used. If set to TRUE, RTS & CTS use up 2 I/O Pins of PortB.

Software Implementation of Asynchronous Serial I/O

TABLE 2 - BIT ASSIGNMENTS OF SERIAL STATUS/CONTROL REGISTER ("SERIALSTATUS" REG)

Bit #	Name	Description
0	_txmtProgress	1 = Transmission in progress. 0 = Transmission line free.
1	_txmtEnable	Set this bit to 1 on initialization to enable transmission. This bit may be used to abort a transmission. The transmission is aborted if in the middle of a transmission (i.e. when _txmtProgress bit is 1) _txmtEnable bit is set to 0. This bit gets automatically set when PutChar function is called.
2	_rcvProgress	1 = Middle of a byte reception. 0 = Reception of a byte (in RxReg) is complete and is set to 1 when a valid start bit is detected in reception mode.
3	_rcvOver	0 = Completion of reception of a byte. The user's code can poll this bit after calling "GetChar" function and check to see if it is set. When set, the received byte is in RxReg. Other status bits should also be checked for any reception errors.
4	_ParityErr	1 = Parity error on reception (irrespective of Even Or Odd parity chosen). Not applicable if No Parity is used.
5	_FrameErr	1 = Framing error on reception.
6		Unused
7	_parityBit	The 9th bit of transmission or reception. In transmission mode, the parity bit of the byte to be transmitted is set in this bit. In receive mode, the 9th bit (or parity bit) received is stored in this bit. Not Applicable if no parity is used.

Software Implementation of Asynchronous Serial I/O

Hardware

The hardware is primarily concerned with voltage translation from RS-232 to CMOS levels and vice versa. Three circuits are given below and the user may choose which ever best suits his application. The primary difference between each solution is cost versus number of components. Circuits in Figure 3 and 4 are very low cost but have more components than the circuit in Figure 2. The circuit in Figure 2 interfaces to RS-232 line using a single chip (MAX-232) and single +5V supply. The circuit in Figure 3 is a low cost RS-232 Interface but requires two chips and a single +5V supply source.

Figure 4 shows a very low cost RS-232 Interface to an IBM PC® with no external power requirements. The circuit draws power from RS-232 line (DTR) and meets the spec of drawing power less than 5mA. This requires that the host to communicate must assert DTR high and RTS low. The power is drawn from DTR line and this requires that DTR to be asserted high and must be at least 7V. The negative -5 to -10 V required by LM339 is drawn from RTS line and thus the host must assert RTS low. This circuit is possible because of the low current consumption of PIC16C71 (typical 2 mA).

FIGURE 2 - SINGLE CHIP FOR RS-232 INTERFACE (SINGLE +5V SUPPLY)

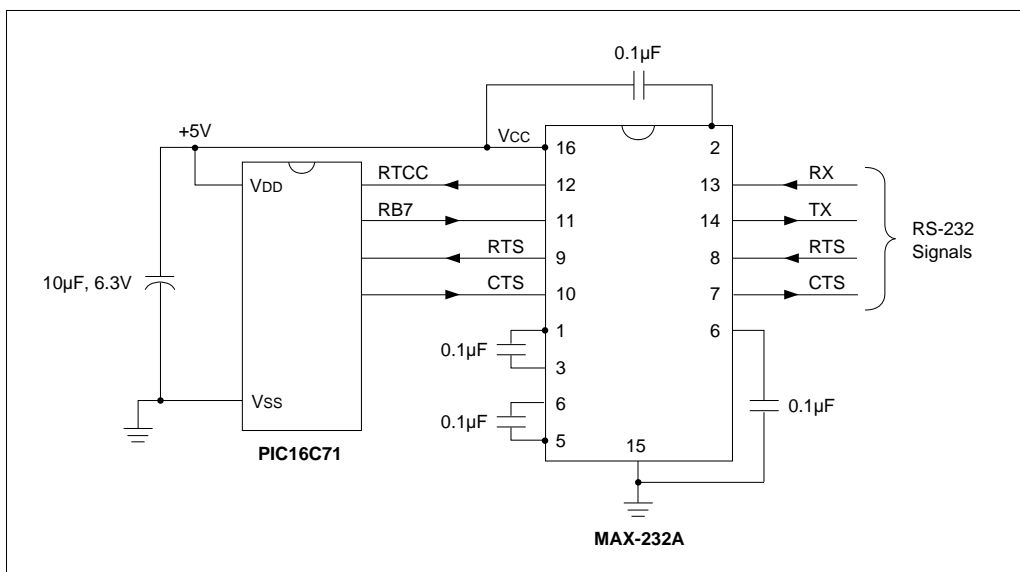
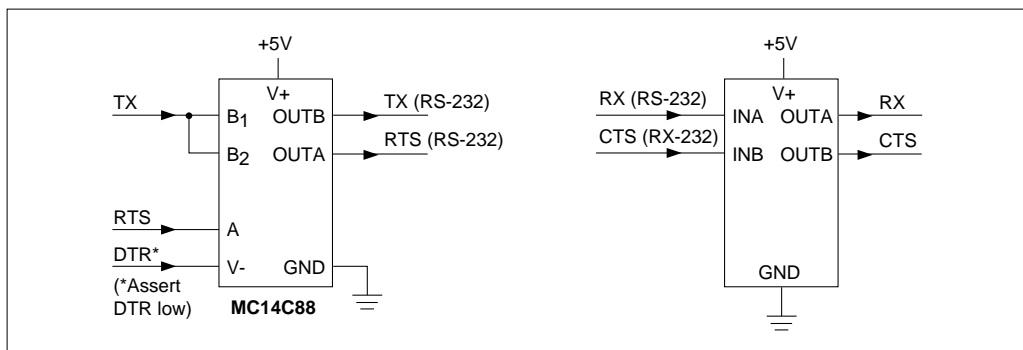
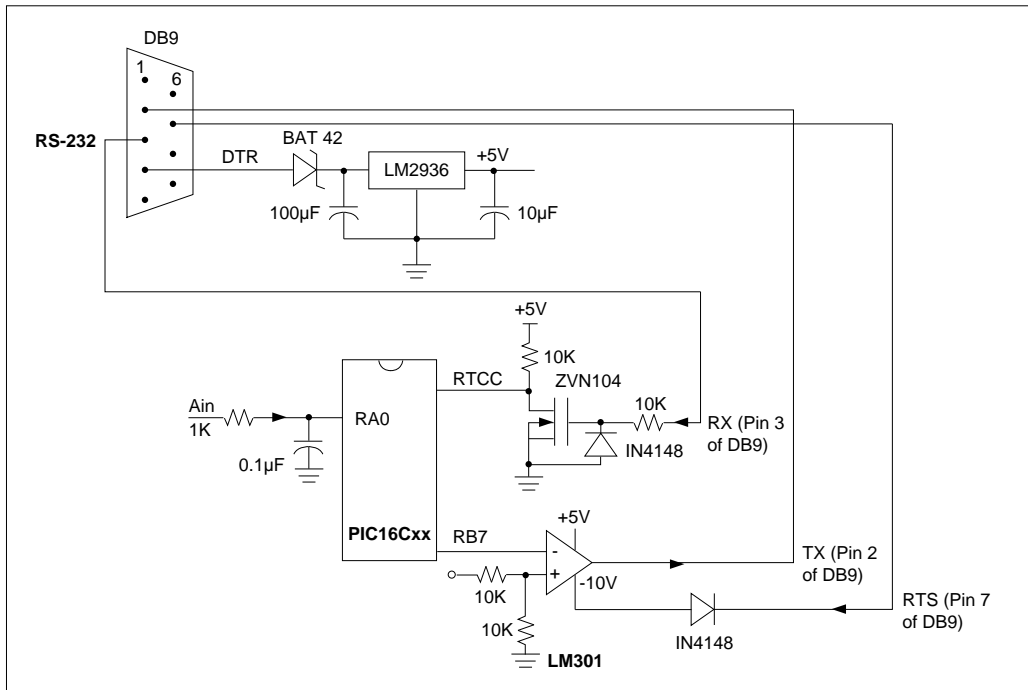


FIGURE 3 - LOW COST RS-232 INTERFACE (TWO CHIPS, SINGLE +5V SUPPLY)



Software Implementation of Asynchronous Serial I/O

FIGURE 4 - LOW COST, LOW POWER RS-232 INTERFACE (POWER SUPPLIED BY RS-232 LINES)

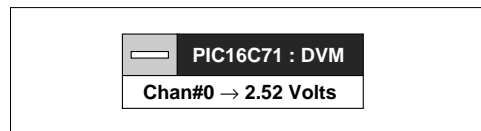


3

Test Program

To test the transmission and reception modules, a main program is written in which the PIC16C71 waits to receive a command from a host through the RS-232. On reception of a byte (valid commands are 0x00, 0x01, 0x02 & 0x03), the received byte is treated as the PIC16C71's A/D channel number and the requested channel is selected, an A/D conversion is started and when the conversion is complete (in about 20 µs) the digital data (8 bits) are transmitted back to the host. A Microsoft® Windows® program running on an IBM PC/AT® was written to act as a host and collect the A/D data from PIC16C71 via an RS-232 port. The Windows program (DVM.EXE) runs as a background job and displays the A/D data in a small window (similar to the CLOCK program that comes with MS Windows). The windows program and the PIC16C71 together act like a data acquisition system or a digital volt meter (DVM). The block diagram of the system is shown in Figure 2. The input clock frequency is fixed at 4 MHz and RS-232 parameters are set to 1200 Baud, 8-bits, 1 Stop Bit and No Parity. The program during development stage was also tested at 1200, 2400, 4800 Baud Rates @ 4 MHz Input Clock and up to 19200 Baud @ 10 MHz input clock frequency (all tests were performed with No Parity, Even Parity and Odd Parity at 8 and 7 Data Bits).

FIGURE 5 - MS WINDOWS PROGRAM FETCHING A/D DATA FROM PIC16C71 VIA RS-232



Software Implementation of Asynchronous Serial I/O

Source Code

The PIC16CXX source code along with the Microsoft® Windows™ DVM Program (executable running on an IBM PC/AT under MS Windows 3.1 or higher) is available on Microchip's BBS. The assembly code for PIC16CXX must be assembled using Microchip's Universal Assembler, MPASM. The code cannot be assembled using the older assemblers without significant modifications. It is suggested that user's who do not have the new assembler MPASM, must change to the new version.

The MS Windows Program (DVM.EXE) runs under MS Windows 3.1 or higher. The program does not have any menus and shows up as a small window displaying A/D Data and runs as a background job. There are a few command line options and are described below :

- Px : x is the comm port number (e.g. - P2 selects COM2). Default is COM1
- Cy : y is the number of A/D channels to display. Default is one channel (channel #1)
- Sz : z is a floating point number that represents the scaling factor (For example - S5.5 would display the data as $5.5 * \langle 8\text{bit A/D} \rangle / 256$). The default value is 5.0 volts. -S0 will display the data in raw format without any scaling.

<i>Author:</i>	<i>Amar Palacherla</i> <i>Logic Products Division</i>
<i>Code</i>	
<i>Update:</i>	<i>Scott Fink - Sr. Applications Engineer</i> <i>Logic Products Division</i>

Appendix A - RS232.H

```

NOLIST
;*****
; RS-232 Header File
; PIC16C6X/7X/8X
;*****
_ClkOut      equ    (_ClkIn >> 2)      ; Instruction Cycle Freq = CLKIN/4
;
_CyclesPerBit set  (_ClkOut/_BaudRate)
_tempCompute set  (_CyclesPerBit >> 8)
;
;*****
; Auto Generation Of Prescaler & Rtcc Values
; Computed during Assembly Time
;*****
; At first set Default values for RtccPrescale & RtccPreLoad
;
RtccPrescale set  0
RtccPreLoad  set  _CyclesPerBit
UsePrescale  set  FALSE
if (_tempCompute >= 1)
  RtccPrescale set  0
  RtccPreLoad  set  (_CyclesPerBit >> 1)
  UsePrescale  set  TRUE
endif
if (_tempCompute >= 2)
  RtccPrescale set  1
  RtccPreLoad  set  (_CyclesPerBit >> 2)
endif
if (_tempCompute >= 4)
  RtccPrescale set  2
  RtccPreLoad  set  (_CyclesPerBit >> 3)
endif
if (_tempCompute >= 8)
  RtccPrescale set  3
  RtccPreLoad  set  (_CyclesPerBit >> 4)
endif

```

Software Implementation of Asynchronous Serial I/O

```
if (_tempCompute >= 16)
  RtcPrescale set 4
  RtcPreLoad set (_CyclesPerBit >> 5)
endif

if (_tempCompute >= 32)
  RtcPrescale set 5
  RtcPreLoad set (_CyclesPerBit >> 6)
endif

if (_tempCompute >= 64)
  RtcPrescale set 6
  RtcPreLoad set (_CyclesPerBit >> 7)
endif

if (_tempCompute >= 128)
  RtcPrescale set 7
  RtcPreLoad set (_CyclesPerBit >> 8)
endif

;
; if ( (RtcPrescale == 0) && (RtcPreLoad < 60) )
;   messg "Warning : Baud Rate May Be Too High For This Input Clock"
;   endif
;
; Compute RTCC & Prescaler Values For 1.5 Times the Baud Rate for Start Bit Detection
;
_SBitCycles set (_ClkOut/_BaudRate) + ((_ClkOut/4)/_BaudRate)
_tempCompute set (_SBitCycles >> 8)

_BIT1_INIT set 08
SBitPrescale set 0
SBitRtccLoad set _SBitCycles
SBitRtccLoad set _SBitCycles

if (_tempCompute >= 1)
  SBitPrescale set 0
  SBitRtccLoad set (_SBitCycles >> 1)
  _BIT1_INIT set 0
endif

if (_tempCompute >= 2)
```



```
SBitPrescale set 1
SBitRtccLoad set (_SBitCycles >> 2)
endif

if (_tempCompute >= 4)
  SBitPrescale set 2
  SBitRtccLoad set (_SBitCycles >> 3)
endif

if (_tempCompute >= 8)
  SBitPrescale set 3
  SBitRtccLoad set (_SBitCycles >> 4)
endif

if (_tempCompute >= 16)
  SBitPrescale set 4
  SBitRtccLoad set (_SBitCycles >> 5)
endif

if (_tempCompute >= 32)
  SBitPrescale set 5
  SBitRtccLoad set (_SBitCycles >> 6)
endif

if (_tempCompute >= 64)
  SBitPrescale set 6
  SBitRtccLoad set (_SBitCycles >> 7)
endif

if (_tempCompute >= 128)
  SBitPrescale set 7
  SBitRtccLoad set (_SBitCycles >> 8)
endif

;
;*****
;
#define _Cycle_Offset1 24 ;account for interrupt latency, call time
LOAD_RTCC MACRO Mode, K, Prescale
  if (UsePrescale == 0 && Mode == 0)
```

Software Implementation of Asynchronous Serial I/O

```
movlw -k + _Cycle_Offset1
else
movlw -k + (_Cycle_Offset1 >> (Prescaler+1)) ; Re Load RTCC init value + INT Latency Offset
endif
movwf _rtcc ; Note that Prescaler is cleared when RTCC is written

ENDM
;*****
LOAD_BITCOUNT MACRO
    movlw _DataBits+1
    movwf BitCount
    movlw 1
    movwf ExtraBitCount
    if _PARITY_ENABLE
        movlw 2
        movwf ExtraBitCount
    endif
    ENDM
;*****
;*****
;***** Pin Assignments *****
;*****
#define RX_MASK 0x10 ; RX pin is connected to RA4, ie. bit 4
#define RX_Pin _porta,4 ; RX Pin : RA4
#define RX RxTemp,4
#define TX _portb,7 ; TX Pin , RB7
#define RTS _portb,5 ; RTS Pin, RB5, Output signal
#define CTS _portb,6 ; CTS Pin, RB6, Input signal

#define_txmtProgress SerialStatus,0
#define_txmtEnable SerialStatus,1
#define_rcvProgress SerialStatus,2
#define_rcvOver SerialStatus,3
#define_ParityErr SerialStatus,4
#define_FrameErr SerialStatus,5
#define_parityBit SerialStatus,7
;*****
```

```

_OPTION_SBIT set 0x38 ; Increment on Ext Clock (falling edge), for START Bit Detect
if UsePrescale
_OPTION_INIT set 0x00 ; Prescaler is used depending on Input Clock & Baud Rate
else
_OPTION_INIT set 0x0F
endif

CBLOCK 0x0C
TxReg ; Transmit Data Holding/Shift Reg
RxReg ; Rcv Data Holding Reg
RxTemp
SerialStatus ; Txmt & Rcv Status/Control Reg
BitCount
ExtraBitCount ; Parity & Stop Bit Count
SaveWReg ; temp hold reg of WREG on INT
SaveStatus ; temp hold reg of STATUS Reg on INT

temp1, temp2
ENDC
;*****
LIST
;*****
```



```

; Usage :
;
; An example is given in the main program on how to Receive & Transmit Data
; In the example, the processor waits until a command is received. The command is interpreted
; as the A/D Channel Number of PIC16C71. Upon reception of a command, the desired A/D channel
; is selected and after A/D conversion, the 8 Bit A/D data is transmitted back to the Host.
;
; The RS-232 Control/Status Reg's bits are explained below :
;
; "SerialStatus" : RS-232 Status/Control Register
;
; Bit 0 : _txmtProgress (1 if transmission in progress, 0 if transmission is complete)
; After a byte is transmitted by calling "PutChar" function, the
; user's code can poll this bit to check if transmission is complete.
; This bit is reset after the STOP bit has been transmitted
;
; Bit 1 : _txmtEnable
; Set this bit to 1 on initialization to enable transmission.
; This bit can be used to Abort a transmission while the transmitter
; is in progress (i.e when _txmtProgress = 1)
;
; Bit 2 : _rcvProgress
; Indicates that the receiver is in middle of reception.It is reset when
; a byte is received.
;
; Bit 3 : _rcvOver
; This bit indicates the completion of Reception of a Byte. The user's
; code can poll this bit after calling "GetChar" function. Once "GetChar"
; function is called, this bit is 1 and is set to 0 after reception of
; a complete byte (parity bit if enabled & stop bit)
;
; Bit 4 : _ParityErr
; A 1 indicates Parity Error on Reception (for both even & odd parity)
;
; Bit 5 : _FrameErr
; A 1 indicates Framing Error On Reception
;
; Bit 6 : _unused_
; Unimplemented Bit
;
; Bit 7 : _parityBit
; The 9 th bit of transmission or reception (status of PARITY bit
; if parity is enabled)
;
; To Transmit A Byte Of Data :
; 1) Make sure _txmtProgress & _rcvOver bits are cleared
; 2) Load TxReg with data to be transmitted
; 3) CALL PutChar function
;
; To Receive A Byte Of Data :
; 1) Make sure _txmtProgress & _rcvOver bits are cleared
; 2) CALL GetChar function
; 3) The received Byte is in TxReg after _rcvOver bit is cleared
;
; Rev 2, May 17,1994 Scott Fink
; Corrected 7 bit and parity operation, corrected stop bit generation, corrected
; receive prescaler settings. Protected against inadvertant WDT reset.
;*****
Processor 16C71

```

Software Implementation of Asynchronous Serial I/O

```
Radix DEC
EXPAND
include "l6Cxx.h"
;*****
;***** Setup RS-232 Parameters *****
;*****
_ClkIn equ 400000 ; Input Clock Frequency is 4 Mhz
_BaudRate set 1200 ; Baud Rate (bits per second) is 1200
_DataBits set 8 ; 8 bit data, can be 1 to 8
_StopBits set 1 ; 1 Stop Bit, 2 Stop Bits is not implemented
#define _PARITY_ENABLE FALSE ; NO Parity
#define _ODD_PARITY FALSE ; EVEN Parity, if Parity enabled
#define _USE_RTSCTS FALSE ; NO Hardware Handshaking is Used
include "rs232.h"
;*****
;*****
ORG _ResetVector
goto Start
;
ORG _IntVector
goto Interrupt
;*****
;***** Table Of ADCON0 Reg *****
; Inputs : WREG (valid values are 0 thru 3)
; Returns In WREG, ADCON0 Value, selecting the desired Channel
; Program Memory : 6 locations
; Cycles : 5
;*****
GetADCon0:
andlw 0x03 ; mask off all bits except 2 LSBs (for Channel # 0, 1, 2, 3)
addwf _pcl
retlw (0xC1 | (0 << 3)) ; channel 0
retlw (0xC1 | (1 << 3)) ; channel 1
retlw (0xC1 | (2 << 3)) ; channel 2
GetADCon0_End:
retlw (0xC1 | (3 << 3)) ; channel 3
```

```

if( (GetADCon0 & 0xff) >= (GetADCon0_End & 0xff))
    MESSG "Warning : Crossing Page Boundary in Computed Jump, Make Sure PCLATH is Loaded Correctly"
endif
;*****
;***** Initialize A/D Converter *****
; <RA0:RA3> Configure as Analog Inputs, VDD as Vref
; A/D Clock Is Internal RC Clock
; Select Channel 0
;
; Program Memory : 6 locations
; Cycles : 7
;*****
InitAtod:
    bsf    _rp0
    clrf  _adcon1
    bcf   _rp0
    movlw 0xCl
    movwf _adcon0
    return
;*****
;***** Main Program Loop *****
;
; After appropriate initialization, The main program wait for a command from RS-232
; The command is 0, 1, 2 or 3. This command/data represents the A/D Channel Number.
; After a command is received, the appropriate A/D Channel is selected and when conversion is
; completed the A/D Data is transmitted back to the Host. The controller now waits for a new
; command.
;*****
Start:
    call  InitSerialPort
;
WaitForNextSel:
    if _USE_RTSCIS
        bcf  _rp0
        bcf  _RTS
    endif
    call  GetChar
    btfsc _rcvOver
    goto $-1
;
; A Byte is received, Select The Desired Channel & TMXT the desired A/D Channel Data
;
; bcf  _rp0 ; make sure to select Page 0

```

Software Implementation of Asynchronous Serial I/O

```
movf RxReg,w
call GetADCon0
movwf _adcon0
nop
;
; WREG = Commanded Channel # (0 thru 3)
; Get ADCON0 Reg Constant from Table Lookup
; Load ADCON0 reg, selecting the desired channel
;
; start conversion
; Loop Until A/D Conversion Done
;
; Half duplex mode, transmission mode, ask host not to send data
; Check CTS signal if host ready to accept data
; Loop Until Transmission Over, User Can Perform Other Jobs
;
movf _adres,w
movwf TxReg
if _USE_RTSCCTS
bsf _RTS
btfsf _CTS
goto $-1
endif
call PutChar
btfsf _txmtProgress
goto $-1
;
; goto WaitForNextSel
; wait for next selection (command from Serial Port)
;
;*****
; RS-232 Routines
;*****
; Interrupt Service Routine
;
; Only RTCC Interrupt Is used. RTCC Interrupt is used as timing for Serial Port Receive & Transmit
; Since RS-232 is implemented only as a Half Duplex System, The RTCC is shared by both Receive &
; Transmit Modules.
; Transmission :
; RTCC is setup for Internal Clock increments and interrupt is generated when
; RTCC overflows. Prescaler is assigned, depending on The INPUT CLOCK & the
; desired BAUD RATE.
; Reception :
; When put in receive mode, RTCC is setup for external clock mode (FALLING EDGE)
; and preloaded with 0xFF. When a Falling Edge is detected on RTCC Pin, RTCC
; rolls over and an Interrupt is generated (thus Start Bit Detect). Once the start
; bit is detected, RTCC is changed to INTERNAL CLOCK mode and RTCC is preloaded
; with a certain value for regular timing interrupts to Poll RTCC Pin (i.e RX pin).
;*****
Interrupt:
btfsf _rtif
retfie
; other interrupt, simply return & enable GIE
```


Software Implementation of Asynchronous Serial I/O

```
include "Txmtr.asm" ; The Transmit routines are in file "Txmtr.asm"  
include "Rcvr.asm" ; The Receiver Routines are in File "Rcvr.asm"  
;*****  
END
```

Appendix C - RCVR.ASM

```

;*****
; GetChar Function
; Receives a Byte Of Data
; When reception is complete, _rcvOver Bit is cleared
; The received data is in RxReg
;
; Program Memory : 15 locations (17 locations if PARITY is used)
; Cycles : 16 (18 if PARITY is USED)
;*****
GetChar:
    bcf _rp0
    bsf _rcvOver
    LOAD_BITCOUNT
    clrf RxReg
    bcf _FrameErr
    bcf _ParityErr
    clrf _rtcc
    clrwdt
    bsf _rp0
    movlw 07h
    movwf _option
    bcf _rp0
    clrf _rtcc
    bsf _rp0
    movlw 0Fh
    movwf _option
    clrwdt
    movlw _OPTION_SBIT
    movwf _option
    bcf _rp0
    movlw 0xFF
    movwf _rtcc
    bcf _rtif
    bsf _rtie
    retfie
;*****
; Internal Subroutine
; entered from Interrupt Service Routine when Start Bit Is detected.
;
; Program Memory : 14 locations
; Cycles : 12 (worst case)
;*****
;_SbitDetected:

```

Software Implementation of Asynchronous Serial I/O

```
bcf _rp0
btfsc RX_Pin
goto _FalseStartBit
bsf _rcvProgress
clr _rtcc
clrdt
bsf _rp0
movlw 07h
movwf _option
bcf _rp0
clr _rtcc
bsf _rp0
movlw 0Fh
movwf _option
clrdt
movlw (_BIT1_INIT | SBitPrescale) ; Switch Back to INT Clock
movwf _option ; Set Option Reg Located In Page 1
bcf _rp0 ; make sure to select Page 0
LOAD_RTCC 1,(SBitRtccLoad), SBitPrescale
goto RestoreIntStatus
;
;_FalseStartBit:
movlw 0xFF
movwf _rtcc ; reload RTCC with 0xFF for start bit detection
goto RestoreIntStatus
;
;*****
; Internal Subroutine
; entered from Interrupt Service Routine when Start Bit Is detected.
;
; Program Memory : 28 locations ( 43 locations with PARITY enabled)
; Cycles : 24 Worst Case
;
;*****
;_RcvNextBit:
clrdt
bsf _rp0
movlw 07h
movwf _option
bcf _rp0
clr _rtcc
clrdt
bsf _rp0
movlw 07h
movwf _option
bcf _rp0
clr _rtcc
bsf _rp0
```


Software Implementation of Asynchronous Serial I/O

```
    bcf  RxReg,7
    endif
    goto RestoreIntStatus
;
    _NextRcvBit:
    bcf  _carry
    btfs RX
    bsf  _carry
    rrf  RxReg
    goto RestoreIntStatus
;
    if _PARITY_ENABLE
    _RcvParity:
    bcf  _ParityErr
    btfs RX
    bsf  _ParityErr
    goto RestoreIntStatus
    endif
;*****
```

Appendix D - TXMTR.ASM

```

;*****
; PutChar Function
;
; Function to transmit A Byte Of Data
; Before calling this routine, load the Byte to be transmitted into TxReg
; Make sure _txmtProgress & _rcvOver bits (in Status Reg) are cleared before
; calling this routine
;
; Program Memory : 6 locations (10 locations if PARITY is Used)
; Cycles : 8 (13 if PARITY is Used)
;*****
PutChar:
    bsf _txmtEnable ; enable transmission
    bsf _txmtProgress ; Macro to load bit count
    LOAD_BITCOUNT
    decf BitCount,1
    if _DataBits == 7
        bsf TxReg,7
    endif
    if _PARITY_ENABLE
        movf TxReg,W
        call GenParity
    endif
    ; If Parity is used, then Generate Parity Bit

    call _TxmtStartBit
    bsf _rtie
    retfie
;*****
; Internal Subroutine
; entered from Interrupt Service Routine when Start Bit Is detected.
;
; Program Memory : 30 locations (38 locations if PARITY is used)
; Cycles : 15 Worst Case
;*****
_TxmtNextBit:
    bcf _rfp0
    LOAD_RTCC 0,RtccPreLoad,RtccPrescale ; Macro to reload RTCC
;
    movf BitCount
    btfsz _z
    goto _ParityOrStop
;done with data xmission?
;yes, do parity or stop bit

```

Software Implementation of Asynchronous Serial I/O

```
;
  decf BitCount
  goto _NextTxmtBit
;
_ParityOrStop:
  if _PARITY_ENABLE
  btfs ExtraBitCount,1
  goto _SendParity
  endif
  movf ExtraBitCount,1
  btfsc _Z
  goto DoneTxmt
  decf ExtraBitCount,1
;
_StopBit:
  bsf TX
  goto RestoreIntStatus
  goto DoneTxmt
;
_NextTxmtBit:
  bsf _carry
  rrf TxReg
  btfs _carry
  bcf TX
  btfs _carry
  bsf TX
;
  btfs txmtEnable
  bsf _rtie
;
  goto RestoreIntStatus
;
  if _PARITY_ENABLE
  _SendParity:
  decf ExtraBitCount,1
  btfs _parityBit
  bcf TX
  btfs _parityBit
  bsf TX
  goto RestoreIntStatus
  endif
DoneTxmt
  bsf TX
  bcf _rtie
  bcf _txmtProgress
  goto RestoreIntStatus
;
```

;no, send another

;ready for parity bit?

;check if sending stop bit

; STOP Bit is High

; disable further interrupts, Transmission Aborted

;subtract parity from count

;STOP Bit is High
;disable further interrupts
;indicates end of xmission


```

;*****
; Internal Subroutine
; entered from Interrupt Service Routine when Start Bit Is detected.
;
; Program Memory : 9 locations
; Cycles : 10
;*****
;*****
;_TxmtStartBit:
    bcf _rp0
    clr_f _rtcc
    clr_wdt
    bsf _rp0
    movlw 07h
    movwf _option
    bcf _rp0
    clr_f _rtcc
    bsf _rp0
    movlw 0Fh
    movwf _option
    clr_wdt
    movlw (_OPTION_INIT | RtccPrescale)
    movwf _option
    bcf _rp0
    bcf TX
    movlw -RtccPreLoad
    movwf _rtcc
    bcf _rtif
    return
;*****
;***** Generate Parity for the Value in WREG *****
;
; The parity bit is set in _parityBit (SerialStatus,7)
; Common Routine For Both Transmission & Reception
;
; Program Memory : 16 locations
; Cycles : 72
;*****
;***** if _PARITY_ENABLE *****
GenParity:
    movwf temp2
    movf BitCount,w
    movwf temp1
    ParityLoop
        rrf temp2

```

Software Implementation of Asynchronous Serial I/O

```

;put data in carry bit
;parity calculated by XORing all data bits

btfss _carry
goto NotOne
xorlw 00h
goto OneDone
NotOne
xorlw 01h
OneDone
decfsz    temp1
goto Parityloop
movwf temp1
; Parity bit is in Bit 0 of temp1
;
    if _ODD_PARITY
        bsf _parityBit
        btfsc temp1,0
        bcf _parityBit
    else
        bcf _parityBit
        btfsc temp1,0
        bsf _parityBit
    endif
return
endif
;*****

```

Appendix E - RS232.LST

```

MPASM 01.00.02 Alpha \PICMASTR\CU 5-20-1994 9:13:56 PAGE 1
RS232 Communications : Half Duplex : PIC16C6x/7x/8x
Software Implementation : Interrupt Driven
LOC OBJECT CODE LINE SOURCE TEXT
0001 TITLE "RS232 Communications : Half Duplex : PIC16C6x/7x/8x"
0002 SUBTITLE "Software Implementation : Interrupt Driven"
0003 *****
0004 ;*****
0005 ; Software Implementation Of RS232 Communications Using PIC16CXX
0006 ; Half-Duplex
0007 ;
0008 ; These routines are intended to be used with PIC16C6X/7X family. These routines can be
0009 ; used with processors in the 16C6X/7X family which do not have on board Hardware Async
0010 ; Serial Port.
0011 ; MX..
0012 ;
0013 ; Description :
0014 ; Half Duplex RS-232 Mode Is implemented in Software.
0015 ; Both Reception & Transmission are Interrupt driven
0016 ; Only 1 peripheral (RTCC) used for both transmission & reception
0017 ; RTCC is used for both timing generation (for bit transmission & bit polling)
0018 ; and Start Bit Detection in reception mode.
0019 ; This is explained in more detail in the Interrupt Subroutine.
0020 ; Programmable Baud Rate (speed depending on Input Clock Freq.), programmable
0021 ; #of bits, Parity enable/disable, odd/even parity is implemented.
0022 ; Parity & Framing errors are detected on Reception
0023 ;
0024 ;
0025 ; RS-232 Parameters
0026 ; The RS-232 Parameters are defined as shown below:
0027 ;
0028 ; _ClkIn : Input Clock Frequency of the processor
0029 ; (NOTE : RC Clock Mode Is Not Suggested due to wide variations)
0030 ; _BaudRate : Desired Baud Rate. Any valid value can be used.
0031 ; The highest Baud Rate achievable depends on Input Clock Freq.
0032 ; 300 to 4800 Baud was tested using 4 Mhz Input Clock
0033 ; 300 to 19200 Baud was tested using 10 Mhz Input Clock
0034 ; Higher rates can be obtained using higher Input Clock Frequencies.
0035 ; Once the _BaudRate & _ClkIn are specified the program
0036 ; automatically selects all the appropriate timings
0037 ; _DataBits : Can specify 1 to 8 Bits.
0038 ; _StopBits : Limited to 1 Stop Bit. Must set it to 1.
0039 ; _PARITY_ENABLE : Parity Enable Flag. Set it to TRUE or FALSE. If PARITY

```

Software Implementation of Asynchronous Serial I/O

```

0040 ; is used, then set it to TRUE, else FALSE. See "_ODD_PARITY" flag
0041 ; description below
0042 ; Set it to TRUE or FALSE. If TRUE, then ODD PARITY is used, else
0043 ; EVEN Parity Scheme is used.
0044 ; This Flag is ignored if _PARITY_ENABLE is set to FALSE.
0045 ;
0046 ;
0047 ; Usage :
0048 ;
0049 ; An example is given in the main program on how to Receive & Transmit Data
0050 ; In the example, the processor waits until a command is received. The command is interpreted
0051 ; as the A/D Channel Number of PIC16C71. Upon reception of a command, the desired A/D channel
0052 ; is selected and after A/D conversion, the 8 Bit A/D data is transmitted back to the Host.
0053 ;
0054 ;
0055 ; "SerialStatus" : RS-232 Status/Control Register
0056 ;
0057 ; Bit 0 : _txmtProgress (1 if transmission in progress, 0 if transmission is complete)
0058 ; After a byte is transmitted by calling "PutChar" function, the
0059 ; user's code can poll this bit to check if transmission is complete.
0060 ; This bit is reset after the STOP bit has been transmitted
0061 ; Set this bit to 1 on initialization to enable transmission.
0062 ; This bit can be used to Abort a transmission while the transmitter
0063 ; is in progress (i.e when _txmtProgress = 1)
0064 ; Indicates that the receiver is in middle of reception.It is reset when
0065 ; a byte is received.
0066 ;
0067 ; Bit 3 : _rcvOver
0068 ; This bit indicates the completion of Reception of a Byte. The user's
0069 ; code can poll this bit after calling "GetChar" function. Once "GetChar"
0070 ; function is called, this bit is 1 and is set to 0 after reception of
0071 ; a complete byte (parity bit if enabled & stop bit)
0072 ; A 1 indicates Parity Error on Reception (for both even & odd parity)
0073 ; A 1 indicates Framing Error On Reception
0074 ;
0075 ; Bit 6 : _unused_ Unimplemented Bit
0076 ;
0077 ; Bit 7 : _parityBit The 9 th bit of transmission or reception (status of PARITY bit
0078 ; if parity is enabled)
0079 ;
0080 ; To Transmit A Byte Of Data :
0081 ; 1) Make sure _txmtProgress & _rcvOver bits are cleared
0082 ; 2) Load TxReg with data to be transmitted
0083 ; 3) CALL PutChar function
0084 ;
0085 ; To Receive A Byte Of Data :
0086 ; 1) Make sure _txmtProgress & _rcvOver bits are cleared
0087 ; 2) CALL GetChar function
0088 ; 3) The received Byte is in TxReg after _rcvOver bit is cleared

```

```

0088 ;
0089 ; Rev 2, May 17,1994 Scott Fink
0090 ; Corrected 7 bit and parity operation, corrected stop bit generation, corrected
0091 ; receive prescaler settings. Protected against inadvertant WDT reset.
0092 ; *****
0093
0094 Processor 16C71
0095 Radix DEC
0096 EXPAND
0097
0098 include "16Cxx.h"
0099
0100 ; *****
0101 ; Setup RS-232 Parameters *****
0102 ; *****
0103
0104 _ClkIn equ 400000 ; Input Clock Frequency is 4 Mhz
0105 _BaudRate set 1200 ; Baud Rate (bits per second) is 1200
0106 _DataBits set 8 ; 8 bit data, can be 1 to 8
0107 _StopBits set 1 ; 1 Stop Bit, 2 Stop Bits is not implemented
0108
0109 #define _PARITY_ENABLE FALSE ; NO Parity
0110 #define _ODD_PARITY FALSE ; EVEN parity, if Parity enabled
0111 #define _USE_RTSCTS FALSE ; NO Hardware Handshaking is Used
0112
0113 include "rs232.h"
0114
0115 ; *****
0116 ;
0117
0118 ORG _ResetVector
0119 goto Start
0120 ;
0121
0122 ORG _IntVector
0123 goto Interrupt
0124 ;
0125 ; *****
0126 ; Table Of ADCON0 Reg
0127 ; Inputs : WREG (valid values are 0 thru 3)
0128 ; Returns In WREG, ADCON0 Value, selecting the desired Channel
0129 ;

```

Software Implementation of Asynchronous Serial I/O

```
0130 ; Program Memory : 6 locations
0131 ; Cycles : 5
0132 ;
0133 ;*****
0134
0135 GetADCon0:
0136 andlw 0x03 ; mask off all bits except 2 LSBs (for Channel # 0, 1, 2, 3)
0137 addwf _pcl
0138 retlw 0xCl | (0 << 3)) ; channel 0
0139 retlw 0xCl | (1 << 3)) ; channel 1
0140 retlw 0xCl | (2 << 3)) ; channel 2
0141 GetADCon0_End:
0142 retlw 0xCl | (3 << 3)) ; channel 3
0143
0144 if( (GetADCon0 & 0xff) >= (GetADCon0_End & 0xff))
0145 MESSG "Warning : Crossing Page Boundary in Computed Jump, Make Sure PCLATH is Loaded Correctly"
0146 endif
0147 ;
0148 ;*****
0149 ; Initialize A/D Converter
0150 ; <RA0:RA3> Configure as Analog Inputs, VDD as Vref
0151 ; A/D Clock Is Internal RC Clock
0152 ; Select Channel 0
0153 ;
0154 ; Program Memory : 6 locations
0155 ; Cycles : 7
0156 ;
0157 ;*****
0158 InitAtoD:
0159 bsf _rp0
0160 clrfsf _adcon1
0161 bcf _rp0
0162 movlw 0xCl
0163 movwf _adcon0
0164 return
0165 ;
0166 ;*****
0167 ; Main Program Loop
0168 ;
0169 ; After appropriate initialization, The main program wait for a command from RS-232
0170 ; The command is 0, 1, 2 or 3. This command/data represents the A/D Channel Number.
0171 ; After a command is received, the appropriate A/D Channel is selected and when conversion is
0172 ; completed the A/D Data is transmitted back to the Host. The controller now waits for a new
0173 ; command.
0174 ;*****
0175
0176 Start: call InitSerialPort
0177
000B 1683
000C 0188
000D 1283
000E 30C1
000F 0088
0010 0008

0005 3903
0006 0782
0007 34C1
0008 34C9
0009 34D1
000A 34D9

0011 2033
```

```

0178 ;
0179 WaitForNextSel:
0180   if _USE_RTSCSTS
0181     bcf   _rp0
0182     bcf   _RTS
0183   endif
0184   call  GetChar
0185   btfsc _rcvOver
0186   goto $-1
0187 ;
0188 ; A Byte is received, Select The Desired Channel & TXMT the desired A/D Channel Data
0189 ;
0190   bcf   _rp0
0191   movf  RxReg,w
0192   call  GetADCon0
0193   movwf _adcon0
0194   nop
0195 ;
0196   bsf   _go
0197   btfsc _done
0198   goto $-1
0199
0200   movf  _adres,w
0201   movwf TxReg
0202
0203   if _USE_RTSCSTS
0204     bsf   _RTS
0205     btfsc _CTS
0206     goto $-1
0207   endif
0208   call  PutChar
0209   btfsc _txmtProgress
0210   goto $-1
0211
0212 ;
0213   goto WaitForNextSel
0214 ;
0215 ;*****
0216 ; RS-232 Routines
0217 ;*****
0218 ; Interrupt Service Routine
0219 ;
0220 ; Only RTCC Interrupt Is used. RTCC Interrupt is used as timing for Serial Port Receive & Transmit
0221 ; Since RS-232 is implemented only as a Half Duplex System, The RTCC is shared by both Receive &
0222 ; Transmit Modules.
0223 ; Transmission :
0224 ;
0225 ; RTCC is setup for Internal Clock increments and interrupt is generated when

```

Software Implementation of Asynchronous Serial I/O

```

0225 ; RTCC overflows. Prescaler is assigned, depending on The INPUT CLOCK & the
0226 ; desired BAUD RATE.
0227 ;
0228 ; Reception :
0229 ; When put in receive mode, RTCC is setup for external clock mode (FALLING EDGE)
0230 ; and preloaded with 0xFF. When a Falling Edge is detected on RTCC Pin, RTCC
0231 ; rolls over and an Interrupt is generated (thus Start Bit Detect). Once the start
0232 ; bit is detected, RTCC is changed to INTERNAL CLOCK mode and RTCC is preloaded
0233 ; with a certain value for regular timing interrupts to Poll RTCC Pin (i.e RX pin).
0234 ; *****
0235
0236 Interrupt:
0237     btfsf    _rtif
0238     retfie
0239 ;
0240 ; Save Status On INT : WREG & STATUS Regs
0241 ;
0242     movwf   SaveWReg
0243     swapf  _status,w
0244     movwf  SaveStatus
0245 ;
0246     btfsf  _TxmtProgress
0247     goto  _TxmtNextBit
0248     btfsf  _rcvProgress
0249     goto  _RcvNextBit
0250     goto  _SBitDetected
0251 ;
0252 RestoreIntStatus:
0253     swapf  SaveStatus,w
0254     movwf  _status
0255     swapf  SaveWReg
0256     swapf  SaveWReg,w
0257     bcf   _rtif
0258     retfie
0259 ;
0260 ; *****
0261 ;
0262 ;
0263 ;
0264 ; Configure TX Pin as output, make sure TX Pin Comes up in high state on Reset
0265 ; Configure, RX_Pin (RTCC pin) as Input, which is used to poll data on reception
0266 ;
0267 ; Program Memory : 9 locations
0268 ; Cycles : 10
0269 ; *****
0270
0271 InitSerialPort:

```



```

0033 018F      clr    SerialStatus
0272 ;
0273 ;
0274 ; select Page 0 for Port Access
0275 ; make sure TX Pin is high on powerup, use RB Port Pullup
0276 ; Select Page 1 for TrisB access
0277 ; set TX Pin As Output Pin, by modifying TRIS
0278 if _USE_RTSCTS
0279     bcf    _RTS
0280     bsf    _CTS
0281 endif
0282     bsf    RX_Pin
0283     return
0284 ;
0285 ;*****
0286 include "txmtr.asm" ; The Transmit routines are in file "txmtr.asm"
0001 ;*****
0002 ; PutChar Function
0003 ;
0004 ; Function to transmit A Byte Of Data
0005 ; Before calling this routine, load the Byte to be transmitted into TxReg
0006 ; Make sure _txmtProgress & _rcvOver bits (in Status Reg) are cleared before
0007 ; calling this routine
0008 ;
0009 ; Program Memory :      6 locations (10 locations if PARITY is Used)
0010 ; Cycles      :      8 (13 if PARITY is Used)
0011 ;
0012 ;*****
0013 PutChar:
0014     bsf    _txmtEnable
0015     bsf    _txmtProgress
0016     LOAD_BITCOUNT
0017     movlw  _DataBits+1
0018     movwf BitCount
0019     movlw  1
0020     movwf ExtrabitCount
0021     if _PARITY_ENABLE
0022         movlw  2
0023         movwf ExtrabitCount
0024     endif
0025     decf  BitCount,1
0026     if _DataBits == 7
0027         bsf  TxReg,7
0028     endif
0029     if _PARITY_ENABLE
0030         mov  TxReg,W
0031         call GenParity
0032     ; If Parity is used, then Generate Parity Bit

```

Software Implementation of Asynchronous Serial I/O

```
0025     endif
0026 ;
0027     call    _TxmtStartBit
0028     bsf    _rtie
0029     retfie
0030 ;
0031 ;*****
0032 ; Internal Subroutine
0033 ; entered from Interrupt Service Routine when Start Bit Is detected.
0034 ;
0035 ; Program Memory : 30 locations (38 locations if PARITY is used)
0036 ; Cycles       : 15 Worst Case
0037 ;
0038 ;*****
0039
0040 _TxmtNextBit:
0041     bcf    _rp0
0042     LOAD_RTCC 0,RtccPreLoad, RtccPrescale ; Macro to reload RTCC
M     if(UsePrescale == 0 && 0 == 0)
M     movlw  -RtccPreLoad + _Cycle_Offset1
M     else
M     movlw  -RtccPreLoad + (_Cycle_Offset1) ; Re Load RTCC init value + INT La
M     endif
M     movwf  _rtcc
0043 ;
0044     movf  BitCount
0045     btfsc _z
0046     goto _ParityOrStop
0047 ;
0048     decf  BitCount
0049     goto _NextTxmtBit
0050 ;
0051 _ParityOrStop:
0052     if _PARITY_ENABLE
0053     ExtrabitCount,1
0054     goto _SendParity
0055     endif
0056     movf  ExtrabitCount,1
0057     btfsc _z
0058     goto DoneTxmt
0059     decf  ExtrabitCount,1
0060 ;
0061 _StopBit:
0062     bsf  TX
0063     goto RestoreIntStatus
0064     goto DoneTxmt
0065 ;
```

```

0053 1403      _carry
0067 bsf      _carry
0068 rrf      TxReg
0069 btfs    _carry
0070 bcf      TX
0071 btfs    _carry
0072 bsf      TX
0073 ;
0074 btfs    _txmtEnable
0075 bsf      _rtie
0076 ;
0077 goto    RestoreIntStatus
0078 ;
0079 if _PARITY_ENABLE
0080 _SendParity:
0081 decf    ExtraBitCount,1
0082 btfs    _parityBit
0083 bcf      TX
0084 btfs    _parityBit
0085 bsf      TX
0086 goto    RestoreIntStatus
0087 endif
0088
0089 DoneTxmt
0090 bsf      TX
0091 bcf      _rtie
0092 bcf      _txmtProgress
0093 goto    RestoreIntStatus
0094 ;
0095 ;*****
0096 ; Internal Subroutine
0097 ; entered from Interrupt Service Routine when Start Bit Is detected.
0098 ;
0099 ; Program Memory :      9 locations
0100 ; Cycles       :     10
0101 ;
0102 ;*****
0103 _TxmtStartBit:
0104 bcf      _rp0
0105 clrf     _rtcc
0106 clrwdt
0107 bsf      _rp0
0108 movlw   07h
0109 movwf   _option
0110 bcf      _rp0
0111 clrf     _rtcc
0112 bsf      _rp0
0113 movlw   0Fh

0053 1403      _carry
0067 bsf      _carry
0068 rrf      TxReg
0069 btfs    _carry
0070 bcf      TX
0071 btfs    _carry
0072 bsf      TX
0073 ;
0074 btfs    _txmtEnable
0075 bsf      _rtie
0076 ;
0077 goto    RestoreIntStatus
0078 ;
0079 if _PARITY_ENABLE
0080 _SendParity:
0081 decf    ExtraBitCount,1
0082 btfs    _parityBit
0083 bcf      TX
0084 btfs    _parityBit
0085 bsf      TX
0086 goto    RestoreIntStatus
0087 endif
0088
0089 DoneTxmt
0090 bsf      TX
0091 bcf      _rtie
0092 bcf      _txmtProgress
0093 goto    RestoreIntStatus
0094 ;
0095 ;*****
0096 ; Internal Subroutine
0097 ; entered from Interrupt Service Routine when Start Bit Is detected.
0098 ;
0099 ; Program Memory :      9 locations
0100 ; Cycles       :     10
0101 ;
0102 ;*****
0103 _TxmtStartBit:
0104 bcf      _rp0
0105 clrf     _rtcc
0106 clrwdt
0107 bsf      _rp0
0108 movlw   07h
0109 movwf   _option
0110 bcf      _rp0
0111 clrf     _rtcc
0112 bsf      _rp0
0113 movlw   0Fh

005C 1786      ;STOP Bit is High
005D 128B      ;disable further interrupts
005E 100F      ;indicates end of xmission
005F 282D

```

Software Implementation of Asynchronous Serial I/O

```
006A 0081      _option
006B 0064      clrwdt
006C 3001      movlw    (_OPTION_INIT | RtcPrescale)
006D 0081      movwf   _option
006E 1283      bcf     _rp0
006F 1386      bcf     TX
0070 3030      movlw   -RtcPreLoad
0071 0081      movwf  _rtcc
0072 110B      bcf     _rtif
0073 0008      return

0124
0125 ;*****
0126 ;          Generate Parity for the Value in WREG
0127 ;
0128 ; The parity bit is set in _parityBit (SerialStatus,7)
0129 ; Common Routine For Both Transmission & Reception
0130 ;
0131 ; Program Memory :      16 locations
0132 ; Cycles       :      72
0133 ;
0134 ;*****
0135 if _PARITY_ENABLE
0136
0137 GenParity:
0138     movwf temp2
0139     movf  BitCount,w
0140     movwf temp1
0141     Parityloop
0142     rrf  temp2
0143     btfs _carry
0144     goto NotOne
0145     xorlw 00h
0146     goto OneDone
0147     NotOne
0148     xorlw 01h
0149     OneDone
0150     decfsz temp1
0151     goto Parityloop
0152     movwf temp1
0153 ; Parity bit is in Bit 0 of temp1
0154 ;
0155     if _ODD_PARITY
0156     bsf  _parityBit
0157     btfsc temp1,0
0158     bcf  _parityBit
0159     else
0160     bcf  _parityBit
0161     temp1,0

; Set Option Reg Located In Page 1
; make sure to select Page 0
; Send Start Bit
; Prepare for Timing Interrupt

;save data
;save bitcount

;put data in carry bit
;parity calculated by XORing all data bits

;decrement count
```

```

0162      bsf      _parityBit
0163    endif
0164
0165    return
0166  endif
0167 ;*****
0287
0288  include "rcvr.asm" ; The Receiver Routines are in File "rcvr.asm"
0001 ;*****
0002 ;      GetChar Function
0003 ;      Receives a Byte Of Data
0004 ;      When reception is complete, _rcvOver Bit is cleared
0005 ;      The received data is in RxReg
0006 ;
0007 ;      Program Memory : 15 locations (17 locations if PARITY is used)
0008 ;      Cycles : 16 (18 if PARITY is USED)
0009 ;
0010 ;*****
0011 GetChar:
0012      bcf      _rp0
0013      bsf      _rcvOver
0014      LOAD_BITCOUNT
M      movlw    _DataBits+1
M      movwf   BitCount
M      movlw    1
M      movwf   ExtraBitCount
M      if     _PARITY_ENABLE
M      movlw    2
M      movwf   ExtraBitCount
M
M      endif
0015      clrf    RxReg
0016      bcf    _FrameErr
0017      bcf    _ParityErr
0018      clrf    _rtcc
0019      clrwdt
0020      bsf    _rp0
0021      movlw  07h
0022      movwf  _option
0023      bcf    _rp0
0024      clrf    _rtcc
0025      bsf    _rp0
0026      movlw  0Fh
0027      movwf  _option
0028      clrwdt
0029      movlw  _OPTION_SBIT
0030      movwf  _option
0031      bcf    _rp0
0032      movlw  0xFF

```

; Enable Reception, this bit gets reset on Byte Rcv Complete

; Init Parity & Framing Errors

; Inc On Ext Clk Falling Edge

; Set Option Reg Located In Page 1

; make sure to select Page 0

Software Implementation of Asynchronous Serial I/O

```

008C 0081      movwf  _rtcc      ; A Start Bit will roll over RTCC & Gen INT
008D 110B      bcf     _rtif
008E 168B      bsf     _rtie      ; Enable RTCC Interrupt
008F 0009      retfie      ; Enable Global Interrupt
0033 ;
0034 ;*****
0035 ;
0036 ;
0037 ;
0038 ;*****
0039 ; Internal Subroutine
0040 ; entered from Interrupt Service Routine when Start Bit Is detected.
0041 ;
0042 ; Program Memory : 14 locations
0043 ; Cycles : 12 (worst case)
0044 ;
0045 ;*****
0046 _SBitDetected:
0047 bcf     _rp0
0048 btfsc  RX_Pin      ; Make sure Start Bit Interrupt is not a Glitch
0049 goto  _FalseStartBit      ; False Start Bit
0050 bsf     _rcvProgress
0051 clrf   _rtcc
0052 clrwdt
0053 bsf     _rp0
0054 movlw  07h
0055 movwf  _option
0056 bcf     _rp0
0057 clrf   _rtcc
0058 bsf     _rp0
0059 movlw  0Fh
0060 movwf  _option
0061 clrwdt
0062 movlw  (_BIT1_INIT | SBitPrescale)      ; Switch Back to INT Clock
0063 movwf  _option      ; Set Option Reg Located In Page 1
0064 bcf     _rp0      ; make sure to select Page 0
0065 LOAD_RTCC 1,(SBitRtccLoad), SBitPrescale
M     if(UsePrescale == 0 && 1 == 0)
M     movlw  -(SBitRtccLoad) + _Cycle_Offset1
M     else
M     movlw  -(SBitRtccLoad) + (_Cycle_Offset1 >> (SBitPrescale+1)) ; Re Load RTCC init value + INT La
M     endif
M     movwf  _rtcc      ; Note that Prescaler is cleared when RTCC is written
0066 goto  RestoreIntStatus
0067 ;
0068 _FalseStartBit:
0069 movlw  0xFF
0070 movwf  _rtcc
0071 goto  RestoreIntStatus
0072 ;
0073 ;*****
0074 ; Internal Subroutine

```

```

0075 ; entered from Interrupt Service Routine when Start Bit Is detected.
0076 ;
0077 ; Program Memory :      28 locations ( 43 locations with PARITY enabled)
0078 ; Cycles       :      24 Worst Case
0079 ;
0080 ;*****
0081 _RcvNextBit:
0082     clrwdt
0083     bsf   _rp0
0084     movlw 07h
0085     movwf _option
0086     bcf   _rp0
0087     clrf _rtcc
0088     clrwdt
0089     bsf   _rp0
0090     movlw 07h
0091     movwf _option
0092     bcf   _rp0
0093     clrf _rtcc
0094     bsf   _rp0
0095     movlw 0Fh
0096     movwf _option
0097     clrwdt
0098     movlw (_OPTION_INIT | RtccPrescale) ; Switch Back to INT Clock
0099     movwf _option                    ; Set Option Reg Located In Page 1
0100 ;
0101     bcf   _rp0
0102     movf  _porta,w
0103     movwf RxTemp
0104     LOAD_RTCC 0,RtccPreLoad, RtccPrescale ; Macro to reload RTCC
M     if(UsePrescale == 0 && 0 == 0)
M     movlw -RtccPreLoad + _Cycle_Offset1
M     else
M     movlw -RtccPreLoad + (_Cycle_Offset1 >> (RtccPrescale+1)) ; Re Load RTCC init value + INT La
M     endif
M     movwf _rtcc ; Note that Prescaler is cleared when RTCC is written
0105     movf  _porta,w
0106     xorwf RxTemp,w
0107     andlw RX_MASK ; mask for only RX PIN (RA4)
0108     btfsz _z
0109     goto _PinsSampled ; both samples are same state
0110 _SampleAgain:
0111     movf  _porta,w
0112     movwf RxTemp ; 2 out of 3 majority sampling done
0113     _PinsSampled:
0114     movf  BitCount,1
0115     btfsz _z
00A8 0064
00A9 1683
00AA 3007
00AB 0081
00AC 1283
00AD 0181
00AE 0064
00AF 1683
00B0 3007
00B1 0081
00B2 1283
00B3 0181
00B4 1683
00B5 300F
00B6 0081
00B7 0064
00B8 3001
00B9 0081
00BA 1283
00BB 0805
00BC 008E
00ED 3036
00BE 0081
00BF 0805
00C0 060E
00C1 3910
00C2 1903
00C3 28C6
00C4 0805
00C5 008E
00C6 0890
00C7 1903

```

Software Implementation of Asynchronous Serial I/O

```
00C8 28CB      0116      goto      _RcvP_Or_S
00C9 0B90      0117      ;
00CA 28D1      0118      decfsz   BitCount
00CB 1E0E      0119      goto      _NextRcvBit
00CC 168F      0120      ;
00CD 128B      0121      _RcvP_Or_S:
00CE 110F      0122      if _PARITY_ENABLE
00CF 118F      0123      decfsz   ExtraBitCount
0100          0124      goto      _RcvParity
0101          0125      endif
0102          0126      ;
0103          0127      _RcvStopBit:
0104          0128      btfss   RX
0105          0129      bsf     _FrameErr
0106          0130      bcf     _rtie
0107          0131      bcf     _rcvProgress
0108          0132      bcf     _rcvOver
0109          0133      if _PARITY_ENABLE
0110          0134      movf   RxReg,w
0111          0135      call  GenParity
0112          0136      movlw  0
0113          0137      btfsc  _parityBit
0114          0138      movlw  0x10
0115          0139      xorwf  SerialStatus
0116          0140      endif
0117          0141      if _DataBits == 7
0118          0142      rrf   RxReg,1
0119          0143      bcf   RxReg,7
0120          0144      endif
0121          0145      goto  RestoreIntStatus
0122          0146      ;
0123          0147      _NextRcvBit:
0124          0148      bcf   _carry
0125          0149      btfsc RX
0126          0150      bsf   _carry
0127          0151      rrf   RxReg
0128          0152      goto  RestoreIntStatus
0129          0153      ;
0130          0154      if _PARITY_ENABLE
0131          0155      _RcvParity:
0132          0156      bcf   _ParityErr
0133          0157      btfsc RX
0134          0158      bsf   _ParityErr
0135          0159      goto  RestoreIntStatus
0136          0160      endif
0137          0161      ;
0138          0162      ;*****
0288          0288      ;*****
```

; may be framing Error or Glitch
; disable further interrupts
; Byte Received, Can RCV/TXMT an other Byte
; Generate Parity, for Parity check
; to mask off Received Parity Bit in _ParityErr
; _ParityErr bit is set accordingly
; prepare bit for shift
; shift in received data
; Temporarily store PARITY Bit in _ParityErr
; Sample again to avoid any glitches


```
0289
0290 ;*****
0291
0292
0293
0294
0295
0296

MEMORY USAGE MAP ('X' = Used, '-' = Unused)
0000 : X-XXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXXXXXX XXXXX-_____XXXXXXXXXXXXXXXXXXXXXXXXXX

All other memory blocks unused.
```

```
Errors : 0
Warnings : 0
```

Software Implementation of Asynchronous Serial I/O

NOTES:

WORLDWIDE SALES & SERVICE

AMERICAS

Corporate Office

Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 602 786-7200 Fax: 602 786-7277
Technical Support: 602 786-7627
Web: <http://www.mchip.com/microhip>

Atlanta

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

Boston

Microchip Technology Inc.
5 Mount Royal Avenue
Marlborough, MA 01752
Tel: 508 480-9990 Fax: 508 480-8575

Chicago

Microchip Technology Inc.
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 708 285-0071 Fax: 708 285-0075

Dallas

Microchip Technology Inc.
14651 Dallas Parkway, Suite 816
Dallas, TX 75240-8809
Tel: 214 991-7177 Fax: 214 991-8588

Dayton

Microchip Technology Inc.
35 Rockridge Road
Englewood, OH 45322
Tel: 513 832-2543 Fax: 513 832-2841

Los Angeles

Microchip Technology Inc.
18201 Von Karman, Suite 455
Irvine, CA 92715
Tel: 714 263-1888 Fax: 714 263-1338

New York

Microchip Technology Inc.
150 Motor Parkway, Suite 416
Hauppauge, NY 11788
Tel: 516 273-5305 Fax: 516 273-5335

AMERICAS (continued)

San Jose

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

ASIA/PACIFIC

Hong Kong

Microchip Technology
Unit No. 3002-3004, Tower 1
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T. Hong Kong
Tel: 852 2 401 1200 Fax: 852 2 401 3431

Korea

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

Singapore

Microchip Technology
200 Middle Road
#10-03 Prime Centre
Singapore 188980
Tel: 65 334 8870 Fax: 65 334 8850

Taiwan

Microchip Technology
10F-1C 207
Tung Hua North Road
Taipei, Taiwan, ROC
Tel: 886 2 717 7175 Fax: 886 2 545 0139

EUROPE

United Kingdom

Arizona Microchip Technology Ltd.
Unit 6, The Courtyard
Meadow Bank, Furlong Road
Bourne End, Buckinghamshire SL8 5AJ
Tel: 44 0 1628 851077 Fax: 44 0 1628 850259

France

Arizona Microchip Technology SARL
2 Rue du Buisson aux Fraises
91300 Massy - France
Tel: 33 1 69 53 63 20 Fax: 33 1 69 30 90 79

Germany

Arizona Microchip Technology GmbH
Gustav-Heinemann-Ring 125
D-81739 Muenchen, Germany
Tel: 49 89 627 144 0 Fax: 49 89 627 144 44

Italy

Arizona Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Pegaso Ingresso No. 2
Via Paracelso 23, 20041
Agrate Brianza (MI) Italy
Tel: 39 039 689 9939 Fax: 39 039 689 9883

JAPAN

Microchip Technology Intl. Inc.
Benex S-1 6F
3-18-20, Shin Yokohama
Kohoku-Ku, Yokohama
Kanagawa 222 Japan
Tel: 81 45 471 6166 Fax: 81 45 471 6122

9/22/95

All rights reserved. © 1995, Microchip Technology Incorporated, USA.

Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. 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. The Microchip logo and name are registered trademarks of Microchip Technology Inc. All rights reserved. All other trademarks mentioned herein are the property of their respective companies.
