**INTRODUCTION**

The PIC16CXX microcontrollers from Microchip Technology, Inc., are 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 straightforward 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. 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 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".

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ClkIn</td>
<td>Input clock frequency of the processor.</td>
</tr>
<tr>
<td>_BaudRate</td>
<td>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 &amp; _ClkIn are specified, the program automatically selects all the appropriate timings.</td>
</tr>
<tr>
<td>_DataBits</td>
<td>Can specify 1 to 8 data bits.</td>
</tr>
<tr>
<td>_StopBits</td>
<td>Limited to 1 Stop Bit. Must be set to 1.</td>
</tr>
<tr>
<td>_PARITY_ENABLE</td>
<td>Parity Enable Flag. Set it to TRUE or FALSE. If PARITY is used, then set it to TRUE, else FALSE. See &quot;_ODD_PARITY&quot; flag description below.</td>
</tr>
<tr>
<td>_ODD_PARITY</td>
<td>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.</td>
</tr>
<tr>
<td>_USE_RTSCTS</td>
<td>RTS &amp; CTS Hardware handshaking signals. If set to FALSE, no hardware handshaking is used. If set to TRUE, RTS &amp; CTS use up 2 I/O Pins of PortB.</td>
</tr>
</tbody>
</table>

FIGURE 1 - AN EFFICIENT PARITY GENERATION SCHEME IN SOFTWARE

TABLE 1 - LIST OF ASSEMBLY TIME OPTIONS
## TABLE 2 - BIT ASSIGNMENTS OF SERIAL STATUS/CONTROL REGISTER ("SERIALSTATUS" REG)

<table>
<thead>
<tr>
<th>Bit #</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>_txmtProgress</td>
<td>1 = Transmission in progress. 0 = Transmission line free.</td>
</tr>
<tr>
<td>1</td>
<td>_txmtEnable</td>
<td>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.</td>
</tr>
<tr>
<td>2</td>
<td>_rcvProgress</td>
<td>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.</td>
</tr>
<tr>
<td>3</td>
<td>_rcvOver</td>
<td>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.</td>
</tr>
<tr>
<td>4</td>
<td>_ParityErr</td>
<td>1 = Parity error on reception (irrespective of Even Or Odd parity chosen). Not applicable if No Parity is used.</td>
</tr>
<tr>
<td>5</td>
<td>_FrameErr</td>
<td>1 = Framing error on reception.</td>
</tr>
<tr>
<td>6</td>
<td>Unused</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>_parityBit</td>
<td>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.</td>
</tr>
</tbody>
</table>
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)

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 uS) 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).
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*8bit A/D/256). The default value is 5.0 volts. -S0 will display the data in raw format without any scaling.
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
if (_tempCompute >= 16)
    RtccPrescale set 4
    RtccPreLoad set (_CyclesPerBit >> 5)
endif

if (_tempCompute >= 32)
    RtccPrescale set 5
    RtccPreLoad set (_CyclesPerBit >> 6)
endif

if (_tempCompute >= 64)
    RtccPrescale set 6
    RtccPreLoad set (_CyclesPerBit >> 7)
endif

if (_tempCompute >= 128)
    RtccPrescale set 7
    RtccPreLoad set (_CyclesPerBit >> 8)
endif

if( (RtccPrescale == 0) && (RtccPreLoad < 60))
    messg "Warning : Baud Rate May Be Too High For This Input Clock"
endif

_SBitCycles set (_ClkOut/_BaudRate) + ((_ClkOut/4)/_BaudRate)
_tempCompute set (_SBitCycles >> 8)

_BIT1_INIT set 08
SBitPrescale set 0
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)
movlw -K + _Cycle_Offset1
else
movlw -K + (_Cycle_Offset1 >> (Prescale+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 Assignements
;**************************************************************************************************
#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
;***************************************************************************************************
Software Implementation of Asynchronous Serial I/O

OPTION_SBIT set 0x30 ; 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 & Rev 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
Half Duplex RS-232 Mode is implemented in Software.
Both Reception & Transmission are Interrupt driven
Only 1 peripheral (RTCC) used for both transmission & reception
RTCC is used for both timing generation (for bit transmission & bit polling)
and Start Bit Detection in reception mode.
This is explained in more detail in the Interrupt Subroutine.
Programmable Baud Rate (speed depending on Input Clock Freq.), programmable
# of bits, Parity enable/disable, odd/even parity is implemented.
Parity & Framing errors are detected on Reception
RS-232 Parameters

The RS-232 Parameters are defined as shown below:

_INPUT_ : Input Clock Frequency of the processor
(ByVal: RC Clock Mode Is Not Suggested due to wide variations)
_BaudRate_: Desired Baud Rate. Any valid value can be used.
The highest Baud Rate achievable depends on Input Clock Freq.
300 to 4800 Baud was tested using 4 Mhz Input Clock
300 to 19200 Baud was tested using 10 Mhz Input Clock
Higher rates can be obtained using higher Input Clock Frequencies.
Once the _BaudRate_ & _ClkIn_ are specified the program
automatically selects all the appropriate timings
_DataBits_: Can specify 1 to 8 Bits.
_StopBits_: Friendly to 1 Stop Bit. Must set it 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
Even Parity Scheme is used.
This Flag is ignored if _PARITY_ENABLE_ is set to FALSE.
Software Implementation of Asynchronous Serial I/O

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 9th 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 inadvertent WDT reset.

Processor 16C71
Radix DEC
EXPAND

include "16Cxx.h"

;******************************************************************************
;                               Setup RS-232 Parameters;******************************************************************************

_Clkin equ 4000000 ; 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

GetADCon0:
andlw 0x03 ; mask off all bits except 2 LSBs (for Channel # 0, 1, 2, 3)
addwf _pclretlw (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 : 5
;
; *******************************************************

InitADtoD:
bsf _rp0
clrf _adcon1
bcf _rp0
movlw 0xC1
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_RTSCTS
bcf _rp0
bcf _RTS
; ready to accept data from host
endif
call GetChar
btfsc _rcvOver
; _rcvOver Gets Cleared when a Byte Is Received (in RxReg)
goto $-1
; USER can perform other jobs here, can poll _rcvOver bit

; A Byte is received, Select The Desired Channel & TMXT the desired A/D Channel Data
bcf _rp0
; make sure to select Page 0
movf RxReg, w               ; WREG = Commanded Channel # (0 thru 3)
call GetADCon0               ; Get ADCON0 Reg Constant from Table Lookup
movwf _adcon0                ; Load ADCON0 reg, selecting the desired channel
nop
;
bsf _go                      ; start conversion
btfsc _done
    goto $-1                   ; Loop Until A/D Conversion Done
movf _adres, w
movf _adcon1, w
if _USE_RTSCTS
    bsf _RTS                  ; Half duplex mode, transmission mode, ask host not to send data
    btfsc _CTS                ; Check CTS signal if host ready to accept data
    goto $-1
endif
    call PutChar
    btfsc _txmtProgress
    goto $-1                   ; Loop Until Transmission Over, User Can Perform Other Jobs

;
    goto WaitForNextSel        ; wait for next selection (command from Serial Port)
;
*************************************************************************
*************************************************************************
                         RS-232 Routines
*************************************************************************
*************************************************************************
    Interrupt Service Routine
;
    Only RTCC Inturrupt Is used. RTCC Inturrupt 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:
    btfss _rtif
    retfie                      ; other interrupt, simply return & enable GIE
; Save Status On INT : WREG & STATUS Regs
;    movf   SaveWReg
    swapf   _status,w               ; affects no STATUS bits : Only way OUT to save STATUS Reg ?????
    movwf   SaveStatus
;    btfs   _txmtProgress
    goto   _TxmtNextBit            ; Txmt Next Bit
    btfs   _rcvProgress
    goto   _RcvNextBit             ; Receive Next Bit
    goto   _SBitDetected           ; Must be start Bit
;
RestoreIntStatus:
    swapf   SaveStatus,wmovwf   _status                 ; restore STATUS Regswapf   SaveWReg                ; save WREG
    bcf     _rtifretfie
;*********************************************************************************************************

; Configure TX Pin as output, make sure TX Pin Comes up in high state on Reset
; Configure, RX_Pin (RTCC pin) as Input, which is used to poll data on reception;
;  Program Memory :     9 locations
;  Cycles         :     10;*********************************************************************************************************
InitSerialPort:
    clrf    SerialStatus
    ;
    bcf     _rp0                            ; select Page 0 for Port Access
    bsf     TX                              ; make sure TX Pin is high on powerup, use RB Port Pullup
    bsf     _rp0                            ; Select Page 1 for TrisB access
    bcf     TX                              ; set TX Pin As Output Pin, by modifying TRIS
    if !_USE_RTSCTS
    bcf     _RTS                            ; RTS is output signal, controlled by PIC16Cxx
    bcf     _CTS                            ; CTS is Input signal, controlled by the host
    endif
    bsf     RX_Pin                          ; set RX Pin As Input for reception
    return
;
;*********************************************************************************************************
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 ; Enable Reception, this bit gets reset on Byte Rcv Complete
LOAD_BITCOUNT
clrf RxReg
bcf _FrameErr
bcf _ParityErr ; Init Parity & Framing Errors
clrf _rtcc
clrwdt
bsf _rp0
movlw 07h
movwf _option
bcf _rp0
clrf _rtcc
bsf _rp0
movlw 0FH
movwf _option
clrwdt
movlw _OPTION_SBIT ; Inc On Ext Clk Falling Edge
movw _option
bcf _rp0 ; make sure to select Page 0
movlw 0xFF
movw _rtcc
bcf _rtif
bsf _rtie ; Enable RTCC Interrupt
retfie ; Enable Global Interrupt

;*****************************************************************************************
; Internal Subroutine
; entered from Interrupt Service Routine when Start Bit Is detected.
; 
; Program Memory : 14 locations
; Cycles : 12 (worst case)
; 
;*****************************************************************************************
_SBitDetected:
bcf _rp0 ; Make sure Start Bit Interrupt is not a Glitch
btfs RX_Pin ; False Start Bit
goto _FalseStartBit ; False Start Bit
bsf _rcvProgress
clrwdt
bsf _rp0
movlw 07h
movwf _option
bcf _rp0
clr _rtcc
bsf _rp0
movlw 0Fh
movwf _option
clrwdt
movw (_BIT3_INIT | SBitPrescale) ; Switch Back to INT Clock
movf _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:
    clrwdt
    bsf _rp0
    movlw 07h
    movwf _option
    bcf _rp0
    clf _rtcc
    clrwdt
    bsf _rp0
    movlw 07h
    movwf _option
    bcf _rp0
    clf _rtcc
    bsf _rp0
movlw 0Fh
movwf _option
clrwdt
movw (_OPTION_INIT | RtccPrescale) ; Switch Back to INT Clock
movwf _option ; Set Option Reg Located In Page 1
,
bcf _rp0
movf _porta,w ; read RX pin immediately into WREG
movwf RxTemp
LOAD_RTCC 0,RtccPreLoad, RtccPrescale
movf _porta,w
xorwf RxTemp,w
andlw RX_MASK
btfsc _z
goto _PinSampled ; mask for only RX PIN (RA4)
_SampleAgain:
movf _porta,w
movwf RxTemp
_PinSampled:
movf BitCount,1
btfsc _z
goto _RcvP_Or_S
; _RcvP_Or_S:
if _PARITY_ENABLE
decfsz ExtraBitCount
goto _RcvParity
endif
; _RcvStopBit:
bcf RX
btf _FrameErr ; may be framing Error or Glitch
bcf _rTe ; disable further interrupts
bcf _rcvProgress
bcf _rcvOver ; Byte Received, Can RCV/DMT an other Byte
if _PARITY_ENABLE
movf RxReg,w
call GenParity
movlw 0
btfsc _parityBit
movlw 0x10 ; to mask off Received Parity Bit in _ParityErr
xorwf SerialStatus ; _ParityErr bit is set accordingly
endif
if _DataBits == 7
rrf RxReg,1
endif
bcf RxReg,7
eendif
goto RestoreIntStatus

;_NextRcvBit:
  bcf _carry
  btfsc RX
  bsf _carry
  rrf RxReg
  goto RestoreIntStatus

;iF _PARITY_ENABLE
  _RcvParity:
    bcf _ParityErr
    btfsc 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
    LOAD_BITCOUNT ; Macro to load bit count
    decf BitCount,1
    if _DataBits == 7
        bsf TxReg,7
    endif
    if _PARITY_ENABLE
        movf TxReg,W
        call GenParity ; If Parity is used, then Generate Parity Bit
    endif
    call _TxmtStartBit
    bsf _rtie ; Enable RTCC Overflow INT
    retfie ; return with _GIE Bit Set

; Internal Subroutine
; entered from Interrupt Service Routine when Start Bit Is detected.

_TxmtNextBit:
    bcf _rp0
    LOAD_RTCC 0,RtccPrescale, RtccPreLoad ; Macro to reload RTCC
    movf BitCount ;done with data xmission?
    btsc _
goto _ParityOrStop ;yes, do parity or stop bit
; decf BitCount
  goto _NextTxmtBit ;no, send another

_PARITYOrStop:
  if _PARITY_ENABLE
    btfsc ExtraBitCount,1 ;ready for parity bit?
    goto _SendParity 
  endif
  movf ExtraBitCount,1 ;check if sending stop bit
  btfsc _z
  goto DoneTxmt
  decf ExtraBitCount,1

_StopBit:
  bsf TX
  goto RestoreIntStatus
  goto DoneTxmt

_NextTxmtBit:
  bsf _carry
  rrf TxReg
  btfss _carry
  bcf TX
  btfss _carry
  bsf TX
  btfss _txmtEnable
  bsf _rtie ; disable further interrupts, Transmission Aborted
  goto RestoreIntStatus
  if _PARITY_ENABLE
    _SendParity:
    decf ExtraBitCount,1 ;subtract parity from count
    btfss _parityBit
    bcf TX
    btfsc _parityBit
    bsf TX
    goto RestoreIntStatus
  endif

DoneTxmt
  bsf TX ;STOP Bit is High
  bcf _rtie ;disable further interrupts
  bcf _txmtProgress ;indicates end of transmission
  goto RestoreIntStatus
;*********************************************************************************************************
; Internal Subroutine
; entered from Interrupt Service Routine when Start Bit Is detected.
; Program Memory : 9 locations
; Cycles : 10
;
;*****************************************************************************
_TxmtStartBit:
bcf _rp0
clr _rtcc
clrwdt
movlw 07h
movwf _option
bcf _rp0
clr _rtcc
bsf _rp0
movlw 0Fh
movwf _option
clrwdt
mov (_OPTION_INIT | RtccPrescale)
movf _option ; Set Option Reg Located In Page 1
bcf _rp0 ; make sure to select Page 0
bcf TX ; Send Start Bit
mov (_rtcPrescale) ; Prepare for Timing Interrupt
movfd _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:
movw temp2 ;save data
movf BitCount,w ;save bitcount
movf temp1
Parityloop
rrf temp2
Software Implementation of Asynchronous Serial I/O

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

goto NotOnexorlw 00h ;parity calculated by XORing all data bits
goto OneDone
NotOne
xorlw 01h
OneDone
decfsz temp1
goto Parityloop
; Parity bit is in Bit 0 of temp1
movwf temp1
; if _ODD_PARITY
bsf _parityBit
btfsc temp1,0
bcf _parityBit
else
bcf _parityBit
endif
return
endif

**********************************************************************************************************
Appendix E - RS232.LST

MPASM 01.00.02 Alpha \PICMASTR\CU 5-20-1994 9:13:56 PAGE 1

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 ;  These routines are intended to be used with PIC16C6X/7X family. These routines can be
0008 ;  used with processors in the 16C6X/7X family which do not have on board Hardware Async
0009 ;  MX..
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 ; RS-232 Parameters
0025 ;
0026 ;The RS-232 Parameters are defined as shown below:
0027 ;
0028 ;_ClikIn : Input Clock Frequency of the processor
0029 ;_BaudRate : Desired Baud Rate. Any valid value can be used.
0030 ;The highest Baud Rate achievable depends on Input Clock Freq.
0031 ; 300 to 4800 Baud was tested using 4 Mhz Input Clock
0032 ; 300 to 19200 Baud was tested using 10 Mhz Input Clock
0033 ; Higher rates can be obtained using higher Input Clock Frequencies.
0034 ; Once the _BaudRate & _ClikIn are specified the program
0035 ; automatically selects all the appropriate timings
0036 ;
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
is used, then set it to TRUE, else FALSE. See "_ODD_PARITY" flag

_description below

_set it to TRUE or FALSE. IF TRUE, then ODD PARITY is used, else
_EVEN Parity Scheme is used.

This Flag is ignored if _PARITY_ENABLE is set to FALSE.

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)

Bit 1 : _txmtEnable Set this bit to 1 on initialization to enable transmission.

Bit 2 : _rcvProgress Indicates that the receiver is in middle of reception. It is reset when

Bit 3 : _rcvOver This bit indicates the completion of Reception of a Byte. The user's

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
Software Implementation of Asynchronous Serial I/O

Processor 16C71
Radix DEC
EXPAND

#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
GetADCon0:

0005 3903 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 (0xC1 | (0 << 3)) ; channel 0
0139 retlw (0xC1 | (1 << 3)) ; channel 1
0140 retlw (0xC1 | (2 << 3)) ; channel 2
0141 GetADCon0_End:
0142 retlw (0xC1 | (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:
000B 1683 0159 bsf _rp0
000C 0188 0160 clrf _adcon1
000D 1283 0161 bcf _rp0
000E 30C1 0162 movlw 0xC1
000F 0088 0163 movwf _adcon0
0010 0008 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:
0177 call InitSerialPort
SOFTWARE IMPLEMENTATION OF ASYNCHRONOUS SERIAL I/O

3-211

0178  ;
0179  WaitForNextSel:
0180  if _USE_RTSCTS
0181  bcf  _rp0
0182  bcf  _RTS
0183  endif
0184  call  GetChar
0185  btfs  _rcvOver
0186  goto  $-1
0187  ;
0188  ; A Byte is received, Select The Desired Channel & TMXT 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  btfs  _done
0198  goto  $-1
0199  ;
0200  movf  _adres,w
0201  movwf  TxReg
0202  if _USE_RTSCTS
0203  bsf  _RTS
0204  btfs  _CTS
0205  goto  $-1
0206  endif
0207  call  PutChar
0208  btfs  _txmtProgress
0209  goto  $-1
0210
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  ;
0224  ; 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.

When put in receive mode, RTCC is set up 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:**

```asm
btfss _rtif, 0009
retfie ; other interrupt, simply return & enable GIE

movwf _status
movwf SaveStatus
swapf _status,w
btfsc _txmtProgress
goto _TxmtNextBit
btfsc _rcvProgress
goto _RcvNextBit
goto _SBitDetected

swapf SaveStatus,w
movwf _status
movwf SaveWReg
swapf _rtif
retfie
```

**Configure TX Pin as output, make sure TX Pin Comes up in high state on Reset**

```asm
swapf SaveStatus, w
mwf _status
swtf _txmtProgress
swtf _rcvProgress
swtf _SBitDetected
```

**Configure RX_Pin (RTCC pin) as Input, which is used to poll data on reception**

```asm
```

**Program Memory:** 9 locations

**Cycles:** 10

**Software Implementation of Asynchronous Serial I/O**
Software Implementation of Asynchronous Serial I/O

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
bsf _txmtProgress
LOAD_BITCOUNT
movlw _DataBits+1
movwf BitCount
movlw 10
movwf ExtraBitCount
if _PARITY_ENABLE
movlw 2
movwf ExtraBitCount
endif

if _DataBits == 7
bsf TxReg,7
endif

if _PARITY_ENABLE
movf TxReg, w
call GenParity
endif

; If parity is used, then generate parity bit

Software Implementation of Asynchronous Serial I/O

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

Internal Subroutine

Macro to reload RTCC

Note that Prescaler is cleared when RTCC is written
Software Implementation of Asynchronous Serial I/O

NextTxmtBit:

1403 bnf _carry
1C8C rrf TxReg
1386 btfss _carry
1803 btfsc _carry
1803 bsf TX
;
1C8F btfss _txmtEnable
168B bsf _rtie ; disable further interrupts, Transmission Aborted
282D goto RestoreIntStatus
;
1283 bcf _rp
0181 clrf _rtcc
1683 bsf _rp
100F movlw 07h
100F btfsc _rtcc
1011 bcf _rtcc
1011 clrf _rtcc
1012 bsf _rp
1013 movlw 0Fh

SendParity:

decf ExtraBitCount,1 ; subtract parity from count
btfss _parityBit
bcf TX
btfsc _parityBit
bsf TX
goto RestoreIntStatus

DoneTxmt

1786 bsf TX ; STOP Bit is High
100F bcf _txmtProgress ; indicates end of xmission
282D goto RestoreIntStatus

**************************************************************************************************

Internal Subroutine entered from Interrupt Service Routine when Start Bit Is detected.

Program Memory : 9 locations
Cycles : 10

**************************************************************************************************

_TxmtStartBit:

1283 bcf _rp
0181 clrf _rtcc
0064 clrwdt
1007 bnf _rp
1007 movlw 07h
1009 bsf _option
1010 bcf _rp
1011 clrf _rtcc
1012 bnf _rp
1013 movlw 0Fh
006A 0081 0114  movwf _option
006B 0064 0115  clrwdt
006C 3001 0116  movlw (_OPTION_INIT | RtccPrescale)
006D 0081 0117  movwf _option ; Set Option Reg Located In Page 1
006E 1283 0118  bcf _rp0 ; make sure to select Page 0
006F 1386 0119  bcf TX ; Send Start Bit
0070 3030 0120  movlw -RtccPreLoad ; Prepare for Timing Interrupt
0071 0081 0121  movwf _rtcc
0072 110B 0122  bcf _rtif
0073 0008 0123  return
0124 ;*********************************************************************************************************
0125 ; Generate Parity for the Value in WREG
0126 ;*********************************************************************************************************
0127 ; The parity bit is set in _parityBit (SerialStatus,7)
0128 ; Common Routine For Both Transmission & Reception
0129 ; Program Memory : 16 locations
0130 ; Cycles : 72
0131 ;*********************************************************************************************************
0132 ;*********************************************************************************************************
0133 ;*********************************************************************************************************
0134 ;*********************************************************************************************************
0135 if _PARITY_ENABLE
0136 0136 GenParity:
0137 movwf temp2 ;save data
0138 movf BitCount,w ;save bitcount
0139 movwf temp1
0140 Parityloop
0141 movwf temp2 ;put data in carry bit
0142 rrf temp2
0143 btfss _carry
0144 goto NotOne
0145 xorlw 00h ;parity calculated by XORing all data bits
0146 goto OneDone
0147 NotOne
0148 xorlw 01h
0149 OneDone
0150 decfsz templ,0 ;decrement count
0151 goto Parityloop ;Parity bit is in Bit 0 of templ
0152 movwf templ
0153 ;*********************************************************************************************************
0154 if _ODD_PARITY
0155 bcf _parityBit
0156 btfsb _parityBit,0
0157 bcf _parityBit
0158 else
0159 bcf _parityBit
0160 btfsb _parityBit,0
Software Implementation of Asynchronous Serial I/O

GetChar:  
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 ;  Program Memory : 15 locations (17 locations if PARITY is used)  
0007 ;  Cycles : 16 (18 if PARITY is USED)  
0008 ;*****************************************************************************************  
0009 ; 0010 ;*****************************************************************************************  
0011 GetChar:  
0074 1283 0012 bcf _rp0  
0075 158F 0013 bsf _rcvOver ; Enable Reception, this bit gets reset on Byte Rcv Complete  
0014 LOAD_BITCOUNT  
0076 3009 M 0015 movlw _DataBits+1  
0077 0090 M 0016 movwf BitCount  
0078 3001 M 0017 movlw 1  
0079 0091 M 0018 movwf ExtraBitCount  
007A 1283 M 0019 if _PARITY_ENABLE  
007B 30F3 M 0020 movlw 2  
007C 0091 M 0021 movwf ExtraBitCount  
007D 1283 M 0022 endif  
007E 0064 0023 clrwdt  
007F 1683 0024 bsf _rp0  
0080 3007 M 0025 movlw 07h  
0081 0081 M 0026 movwf _option  
0082 1283 M 0027 bcf _rp0  
0083 0081 M 0028 clf _rtcc  
0084 1683 M 0029 bsf _rp0  
0085 300F 002A movlw 0Fh  
0086 0081 002B movwf _option  
0087 0064 002C clrwdt  
0088 3038 002D movlw _OPTION_SBIT ; Inc On Ext Clk Falling Edge  
0089 0081 002E movwf _option ; Set Option Reg Located In Page 1  
008A 1283 M 002F bcf _rp0  
008B 30FF 0030 movlw 0xFF ; make sure to select Page 0
Software Implementation of Asynchronous Serial I/O

008C 0081 0033 movwf _rtcc ; A Start Bit will roll over RTCC & Gen INT
008D 110B 0034 bcf _rtif
008E 168B 0035 bsf _rtie ; Enable RTCC Interrupt
008F 0009 0036 retfie ; Enable Global Interrupt
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 ;_SBitDetected:
0050 bcf _rp0
0054 btfsc RX_Pin ; Make sure Start Bit Interrupt is not a Glitch
0058 goto _FalseStartBit ; False Start Bit
0062 clrf _rtcc
0066 clrwdt
0070 movlw 07h
0074 movwf _option
0078 clrf _rtcc
0082 bsf _rp0
0086 movlw (_BIT1_INIT | SBitPrescale) ; Switch Back to INT Clock
0090 LOAD_RTCC 1,(SBitRtccLoad), SBitPrescale
0094 M if(UsePrescale == 0 && 1 == 0)
0098 M movlw -(SBitRtccLoad) + _Cycle_Offset1
00A2 M endif
00A6 3081 M movlw -(SBitRtccLoad) + (_Cycle_Offset1 >> (SBitPrescale+1)) ; Re Load RTCC init value + INT La
00A0 0081 0063 movwf _option ; Set Option Reg Located In Page 1
00A4 M LOAD_RTCC 1,(SBitRtccLoad), SBitPrescale
00A8 M if(UsePrescale == 0 && 1 == 0)
00B2 M movlw -(SBitRtccLoad) + _Cycle_Offset1
00B6 M endif
00C0 M movlw -(SBitRtccLoad) + (_Cycle_Offset1 >> (SBitPrescale+1)) ; Re Load RTCC init value + INT La
00C4 M...
00C8 ...
00D2 ...
00D6 goto RestoreIntStatus
00D8 _FalseStartBit:
00DC 0081 0069 movwf _rtcc ; Note that Prescaler is cleared when RTCC is written
00E0 goto RestoreIntStatus
00E4 ;
00E8 ;*******************************************************************************************
00E9 ; Internal Subroutine
Software Implementation of Asynchronous Serial I/O

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: 00A8 0064
0082         clrwdt
0083         bsf   _rp0
0084         movlw  07h
0085         movwf   _option
0086         bcf     _rp0
0087         movlw  07h
0088         movwf   _option
0089         movlw   0Fh
0090         movwf   _option
0091         movlw   (_OPTION_INIT | RtccPrescale) ; Switch Back to INT Clock
0092         movwf   _option ; Set Option Reg Located In Page 1
0093         movf    _porta,w ; read RX pin immediately into WREG
0094         xorwf   RxTemp
0095         andlw   RX_MASK ; mask for only RX PIN (RA4)
0096         btfsc   _z
0097         goto    _PinSampled
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 ; read RX pin immediately into WREG
0103         movwf   RxTemp
0104 LOAD_RTCC 0, RtccPreLoad, RtccPrescale ; Macro to reload RTCC
0105 if(UsePrescale == 0 && 0 == 0)
0106 movlw   -RtccPreLoad + _Cycle_Offset1
0107 else
0108 movlw   -RtccPreLoad + (_Cycle_Offset1 >> (RtccPrescale+1)) ; Re Load RTCC init value + INT Latency
0109 endif
0110 movlw   -RtccPreLoad + _Cycle_Offset1
0111 movf    _porta,w
0112 movwf   _rtcc ; Note that Prescaler is cleared when RTCC is written
0113 endif
0114 movlw   -RtccPreLoad + (_Cycle_Offset1 >> (RtccPrescale+1)) ; Re Load RTCC init value + INT Latency
0115 movf    _porta,w 
0116 movwf   _rtcc 
0117 ; read RX pin immediately into WREG 
0118 xorwf   RxTemp 
0119 andlw   RX_MASK 
0120 goto    _PinSampled 
0121 ; both samples are same state 
0122 SampleAgain: 
0123 movf    _porta,w 
0124 movwf   _rtcc 
0125 goto    _PinSampled 
0126 ; 2 out of 3 majority sampling done 
0127 _PinSampled: 
0128 movf    _porta,w 
0129 movwf   RxTemp 
0130 btsc   _z 
0131 goto    _PinSampledAgain 
0132 } 
0133 } 
0134 }
00C0 28CB 0116    goto _RcvP_Or_S
0117 ;
00C9 0B90 0118    decfsz BitCount
00CA 28D1 0119    goto _NextRcvBit
0120 ;
0121 _RcvP_Or_S:
0122    if _PARITY_ENABLE
0123        decfsz ExtraBitCount
0124        goto _RcvParity
0125    endif
0126 ;
0127 _RcvStopBit:
00CB 1E0E 0128    btfss RX
00CC 128B 0129    bcf _rtie ; disable further interrupts
00CE 110F 0130    bcf _rcvProgress
00CF 118F 0131    bcf _rcvOver ; Byte Received, Can RCV/TXMT an other Byte
0132    if _PARITY_ENABLE
0133        movf RxReg,w
0134        call GenParity ; Generate Parity, for Parity check
0135        movlw 0
0136        btfsc _parityBit
0137        movlw 0x10 ; to mask off Received Parity Bit in _ParityErr
0138        xorwf SerialStatus ; _ParityErr bit is set accordingly
0139    endif
0140    if _DataBits == 7
0141        rrf RxReg,1
0142        bcf RxReg,7
0143    endif
00D0 282D 0144    goto RestoreIntStatus
0146 ;
0147 _NextRcvBit:
00D1 1003 0148    bcf _carry
00D2 1A0E 0149    btfsc RX ; prepare bit for shift
00D3 1403 0150    bcf _carry
00D4 0C8D 0151    rrf RxReg ; shift in received data
00D5 282D 0152    goto RestoreIntStatus
0153 ;
0154    if _PARITY_ENABLE
0155 _RcvParity:
0156        bcf _ParityErr ; Temporarily store PARITY Bit in _ParityErr
0157        btfsc RX ; Sample again to avoid any glitches
0158        bcf _ParityErr
0159        goto RestoreIntStatus
0160    endif
0161 ;
0162 ;*****************************************************************************************
MEMORY USAGE MAP ('X' = Used, ' '-' = Unused)

0000 : X-XXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXX—— ———————— ————————

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/microchip

Atlanta
Microchip Technology Inc.
500 Sugar Mill Road, Suite 200B
Atlanta, GA  30330
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 951077 Fax: 44 0 1628 950259

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.