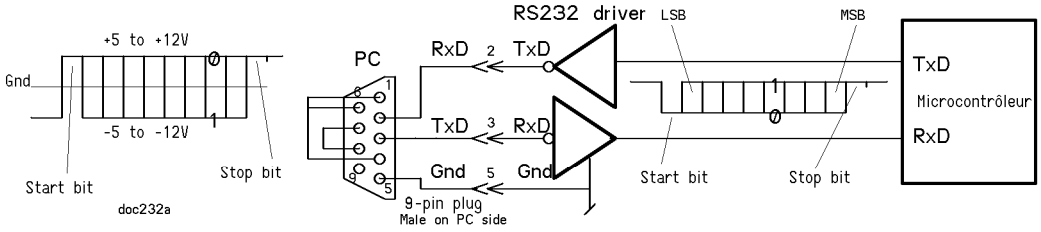
J.D. Nicoud, Mouette 5
CH-1092 Belmont, Switzerland
Tel +41 21 728-6156, Fax 728-6157
Email info@didel.com, www.didel.com

Fichier: www.didel.com/doc/DopicSer.pdf

RS232 routines for a PIC

1. RS232 interface

RS232 is based on precise timings, with a start bit to synchronize the transmission of a byte. Signals are specified for -12V/+12V levels, but most PCs accept 0/+5V levels. The Didel document www.didel.com/doc/Doc232.pdf provides several electrical interfaces between a microcontroller and a RS232 plug.



Doc232a

Fig. 1 RS232 signals

2. Send/receive routines

Several cases have to be considered for the basic send and receive routines that transfer an 8-bit word, usually an Ascii code. On the Pic 16F84, a software routine has to be written, and if processor oscillator is a quartz or resonator, precise timings are easy. On a F84 with RC oscillator, and on the low cost 12C508 controllers, precise timings ask for additional solutions. The 16F87x and other controllers have a wired serial interface (the UART) and a bit rate generator. It makes the life easier and the program more efficient, specially when data in is expected. Interrupt is also possible, but we will not detail the interrupt handling, which is easy in simple cases, and becomes rather complex when real time kernel and fifo have to be implemented.

2.1. Send routine for 16F84

If the F84 (or any processor of the PIC family) has a 4 MHz quartz or resonator, a delay routine will take care of the timings. One could use the timer, but there are better usages for it.

For the SndSer routine we need a delay loop of 4 μs repeated RateAdjust times. The bitrate period is R9600 = 10000000/9600 microseconds at 9600 bit/s (104,5 μs). Hence, RateAdjust = (R9600-9)/4, 9 being the estimated number of instructions (of 1 μs that will be executed between two waiting loops. By leaving the assembler doing the calcuation, it will be easy to change the bit rate (or Baud rate, the incorrect widely used word). At 9600 bits/s, the routine takes 1,15 ms to be executed.

The SndSer routine takes the data to be sent in W and load DataSnd variable, who will play the role of shift register. The Carry is cleard to send the start bit, and then Carry is set to fill the end of the data to be transmitted with stop bits. Hence 11 cycles are required, counted by the CSnd variable.

Routine **SndSer.asi** **8-bit serial transfer**

```
  in:   W
  mod: DataSnd, CSnd
SndSer:
        Move        W,DataSnd
        SerOff      ; Start bit
        Nop         ; More precise start bit duration
        Nop
        Move        #11,W
        Move        W,CSnd
S$:
        Move        #RateAdjust,W ; Calibrated wait
L$:     Add         #-1,W
        Skip,EQ
        Jump        L$
        SetC                      ; prepare Stop bits
        RRC         DataSnd
        Skip,CS
        SerOff
        Skip,CC
        SerOn
        DecSkip,EQ  CSnd
        Jump        S$
        Ret
```

The SerOn SerOff instructions are macros. One should not mention in a general routine which bits should be activated. If TxD is wired on RA0 and RxD on RA1, one will declare

in the port definition part of the listing

```
PortA definitions
bTxD      = 0
bRxD      = 1
DirA      = 11110 ; Last two bits are input and output
PortSer   = 5     ; PortA address, 6 for PortB, 7 for PortC
```

Macros can now be defined:

```
.Macro    SerOn
          Set       PortSer:#bTxD
.Endmacro
.Macro    SerOff
          Clr       PortSer:#bTxD
.Endmacro
.Macro    StopOn
          Set       PortSer:#bTxD
.Endmacro
```

The macro StopOn will be used at initialization, otherwise the first characters to be sent may be wrong if the TxD bit is zero after reset. One more thing to define are the variables: DataSnd and CSnd.

A test program (www.didel.com/doc/Xser.asm) can easily be written around this routine. It sends characters * and U, the most appropriate to check if the timing is correct.

Program **XSnd** 30.12.00 Serial out

```
.Proc 16F84
; PIC at 4MHz
R2400= 1000000/2400
R9600= 1000000/9800
R38400          = 1000000/38400   ; more critical ti
RateAdjust      = R9600/4-9
```

Variables **Ports**

```
PortA
bTxD  = 0
bRxD  = 1         ; not used here
DirA  = 2'10010
PortSer           = 5           ; PortA
.Ins  XSndM.asi   ; Serial macros as above
```

Variables

```
.Loc  16'C
DataSnd:          .Blk.16       1
CSnd: .Blk.16     1
.Loc  0
```

Program **Test program**

```
Start:
      Move        #DirA,W
      Move        W,TrisA
      StopOn
Loop:             Continuous send of * and U
      Move        #"*",W
      Call        SndSer
      Move        #"U",W
      Call        SndSer
      Jump        Loop
.Ins  XSndR.asi   ; Serial routine as above
.End
```

Correct operations and timings can be verified with a scope, or with a terminal. To be sure the timings are not critical, it is recommended to change the correction in RateAdjust constant by adding or subtacting 1 (4 μs); it should work within this range.

If the 16F84 has a cheap RC oscillator, one can adjust the R9600 constant according to the measured speed of the processor. If the frequency is higher than 4MHz by 5%, one should increase the 9600 quotient by 5%.

On the 12C508, the OscCal register allows to adjust more precisely the frequency. A solution is to load a test program (10 instructions, leaving the first instructions with value 16'FF) and measure the frequency. The R9600 parameter can be adjusted and the final program loaded. A tricky solution, also suitable for the 16F84, is to measure the start bit duration of adequate initial characters transmitted by the PC.

One more point. If the electrical interface does not invert (1 diode and two resistors between the PIC and the PC), only the macros have to be changed, and it will work.

2.2. Send routine for 16F87x

Several registers of the 16C76 and 16F87x allow to program the asynchronous receiver and transmitter (USART). The Snd routine has fewer instructions, but a correct initialization of the control registers is required. Microchip documents all this, so we just give below the initialization routine and the new RecSer routine. TxD is on bit 6 and RxD on bit 7 of PortC.

Routine **IniSer** **Init serial port at 9600b/s**

```
        Move    #IniRcSta,W
        Move    W,RcSta
    Bank1
        Move    #IniTxSta,W
        Move    W,TxSta
        Move    #IniSpBrg,W
        Move    W,SpBrg
    Bank0
        Clr     PIR1
        Ret
```

Routine **SndSer.asi** **Serial transfer for 16F87x**

```
  in:   W
  mod: -
SndSer:
        Move    W,TxReg
        Nop     ; nop required
  A$:
        TestSkip,BS PIR1:#TxIF
        Jump    A$
        Ret
```

The test program is quite similar to the Xser.asm program above. The SerOn macro is replaces by a Call IniSer, there is no need of XRecM.asi, and XrecR.asi is replaced by XSnd7R.asi (see later). The definitions and macros (bank switching) of the 16F870 are found in www.didel.com/doc/16F870.asi and www.didel.com/doc/16F870M.asi. One can extract from these files the required definitions and macros, but .Ins macros are quite convenient for this.

2.3. Receive routine for 16F84

The principle of the routine is to wait for the start bit, and the processor will not be able to do anything else, even interrupts, while a character is expected from the PC. This is acceptable for most applications. Macros are defined to avoid the mention of port bits in the RecSer routine. They make also the routine much more easy to understand.

```
.Macro    SkipIfRxOne
          TestSkip,BS  PortSer:#bRxD
.Endmacro
.Macro    SkipIfStartbit
          TestSkip,BC  PortSer:#bRxD
.Endmacro
.Macro    NoSkipIfStartbit
          TestSkip,BS  PortSer:#bRxD
.Endmacro
.Macro    SkipIfStopbit
          TestSkip,BS  PortSer:#bRxD
.Endmacro
```

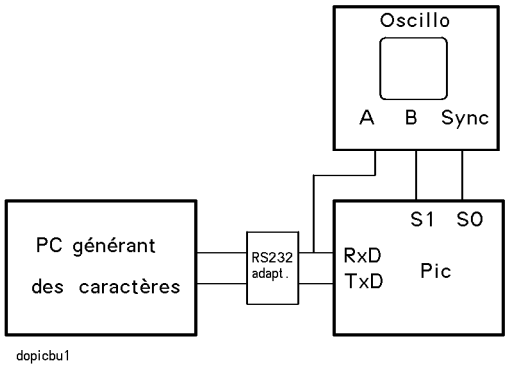
Once the start bit edge is detected, the first waiting loop waits for one and a half period before sampling the first bit. After the last bit sampling, a delay of one more one period reach the stop bit part. This is not an absolute requirement, since when testing the start bit one test to be sure to be on a stopp bit before waiting for the start bit. If the received charactes are coming at full speed, on have a maximum time equivalent to the second stop bit sent by the PC before calling again the RecSer routine. This leave about 100 instructions to handle the data, and it is an important constraint if you do not want to miss characters.

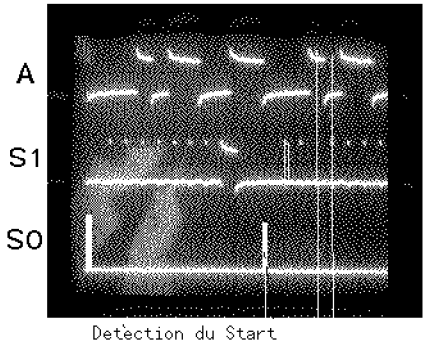
Routine **RecSer** **Serial receive for 16F84 at 9600 .. 2400 b/s**

```
; One stop bit (>100 us) to handle data before checking for new data
  out:   W = DataRec
  mod:  CRec
RecSer:             ; Wait for Start bit
  A$:   SkipIfStopBit
        Jump    A$
  B$:   SkipIfStartBit
        Jump    B$
;s0on   ; Scope syncro
;s0off
Lecture des data
        Move    #8,W
        Move    W,CRec
        Move    #(3*RateAdjust)/2,W         ; wait 1-1/2 period
 L$:
 W$:    Add     #-1,W
        Skip,EQ
        Jump    W$
;s1on   ; Sample time
        SetC
        SkipIfRxOne
        ClrC
;s1off
        RRC     DataRec             ; LSB first
        Move    #RateAdjust,W
        DecSkip,EQ  CRec
        Jump    L$
```

```
;s1on     ; stop bit check
 VV$:     Add       #-1,W    ; wait end of last bit
          Skip,EQ
          Jump      VV$
;s1off
          Move      DataRec,W
          Ret
```

For testing the routine one can display the incoming character code on PortB, and as before change the RateAdjust value to verify the security margins. An Echo program will send back to the terminal the characters received. If it does not work, a scope is required. It is synchronized by S0, and a maintained key on a PC provides an adequate repetition rate.





Both SndSer and RecSer routines for 16F84 can be found in www.didel.com/doc/XSerR.asi.

2.4. Receive routine for 16F870

On PICs with an USART, the initialisation is as before, and the receive routine is given below . One have now the time of the arrival of a complete character (1.1 ms) for executing the instructions

Routine **RecSer** **Serial receive at 9600 bit/s**

```
          out:   DataRec = W
RecSer:
  R$:     TestSkip,BS PIR1:#RcIF
          Jump      R$
          Move      RcReg,W
          Move      W,DataRec
          Ret
```

Both SndSer and RecSer routines for 16F870 can be found in www.didel.com/doc/XSer7R.asi.