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

DIDEL

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

Utility routines for serial transfers

1. Serial primitives

We suppose you have connected your Pic module to your PC, in which an Hyperterminal or similar program is running. Serial transfers use the asynchronous transfer universally, although improperly, named RS232. Hardware interface is documented on www.didel.com/doc/RS232.pdf. Software primitives for 16F84 and 16F87x are documented on www.didel.com/doc/DocSer.pdf. Three routines handle the serial transfer, and it is easy to use them. Understanding their code in detail is not an absolute requirement for the beginner: who knows how a ReadLine / Writeline is programmed inside the C library? But it is essential to define correctly which port bits are used. The absract view of these three routines are:

Routine **IniSer** **Initialize the serial port**

```
; no parameter is passed to or given by this routine
  mod: W  – register W and Status word will always be modified, and we may omit to mention this every time
```

The IniSer routine is called once after initializing the port directions.

Routine **SndSer.asi** **Serial out**

```
  in:  W
  mod: DataSnd, plus two temporary registers in case of 16F84
```

Hence to send an Ascii character, its code must be prepared in W before the Call. If letter A must be sent, the instructions to be written are:

```
        Move    #"A",W
        Call    SndSer
```

The assembler knows the Ascii code values. The syntax "A" is indeed a number, the value of the code of A, that is 16'41. One does not need to know Ascii codes, but it is important to understand the general order: digits, upper cases, lower cases, special characters inbetween. The complet code can be found in www.didel.com/doc/DocSer.pdf.

If one wishes to print the alphabet, a variable is initialized with the code of letter A, and is incremented until a count of 26 or the code of letter Z is reached. Program Xalpha.ASM (www.didel.com/doc/DopicSources.html) is a possible implementation.

The third routine wait for a character, to be received by the serial port. The result is given in W with a copy in DataRec.

Routine **RecSer** **Réception série**

```
  out:  DataRec = W
  mod:  Two local variables for 16F84
```

As example, the following program module waits for characters, but accepts only letter S (upper or lower case) to continue.

```
L$:     Call     RecSer
        Clr      DataRec:#6          ; convert lower case to upper.
        Xor      #"S",W
        Skip,EQ
        Jump     L$
.. Do whatever is required when lettre S is depressed
```

Additional routines

Frequently, one puts a space or a carriage return between variables or texts. Three routines are provided to save a little bit of program space, and improve the legibility. Note that the return to next line "Call SndCr" sends a Carriage return (code 16'D) and a Line feed (code 16'A), to be compatible with PCs.

Routine **SndCr** **Send a CRLF**

Routine **SndSpace** **Send a space**

Routine **SndBell** **Send the Bell code**

Inserted files

In order to be compatible with large programs, we use inserted files even in simple applications. The main program will always be kept short. General routines are kept within an inserted module. Their variables are defined in another module, but it may be preferable to put together all the variables in a single module, the same way as all the I/O and constant definitions are in a module that depends on the hardware.

The general structure of a program will be:

```
        .Proc   16F84
        .Ref    16F84
        .Ins    XXDef.Asi
        .Loc    DebVar
        .Ins    XXVar.asi
        .Loc    0
Begin:
        ; initalizations
Loop:
        ; program loop
        Jump    Loop
        ; Routines
        .Ins    XXroutines.asi
        .End
```

Displaying a text

Texts on the 16F84

If one needs to send a text to the terminal, it is not elegant to send it one letter at a time, as shown before with letter A. A table containing the text is more clear and more efficient. On the 16F84, every text has to be embeded inside a specific routine, and the letters are fetched in the table with the RetMove instruction. The text must not overlap a page, as for all "retmove" tables.

The routine for displaying "Hello" needs an incrementing pointer and some way to stop the transfer at the end of the text. Instead of using a counter, we prefer to use a special character as the terminator for the text. Ascii code 0 is reserved for that. Hence, the loop will read the consecutive characters, and when code zero is found, it means the end of the text.

```
Hello:  Clr       Index     ; Point the beginning of text
 T$:    Move      Index,W
        Inc       Index
        Call      TextHello
        Or        #0,W      ; Test if W is zero
        Skip,NE
        Ret                 ; All text has been scanned
        Call      SndSer
        Jump      T$
TextHello:
        Add       W,PCL
        RetMove   #"H",W
        RetMove   "e",W
        RetMove   "l",W
        RetMove   "l",W
        RetMove   "o",W
        RetMove   #0,W
```

The problem, if a second text has to be displayed, is the "Call TextHello" instruction cannot be computed. A new "Hello" routine must be written for a new text, unless all texts are consecutive within the same page.

The send text routine is anyway good only for text longer than 8 letters. For short text, the naive way is more efficient, but one should simplify the writing with a macro:

```
          .Macro    Txt
                    Move      #"%1",W
                    Call      SndSer
          .Endmacro
Hello:
          Txt       H
          Txt       e
          Txt       l
          Txt       l
          Txt       o
          Ret
```

If one needs to read a text, the problem is where to store it: it is easy to point with the FSR register and store data into registers, but there are very few registers. Usually, the Ascii dialogues are simples (one or two lettres), and interpreted on the fly. It is a microcontroller, isnt-it?

Texts on the 16F87x

Handling texts on the 16F87x is easier and faster, since a register that points to program memory has been added. It is a 16-bit register (13 bits are used on the 16F87x), that is two registers, EEAdrH et EEAdr, must be prepared. The result is a 14-bit word in EEDataH and EEData. Hence to read program memory, one prepares the two address registers, ask for a read as specified in Microchip documentation, and one can transfer the data from the data registers. If only the 8 low bits are significant, as in a text table, one does not need to read the high part.

Routine AfText **16F87x – Send the text at address W in page TextPage**

```
  in:  W text address
AfText:
          Move      W,EeAdr
          Move      #TextPage,W
          Move      W,EeAdrH
  Bank2to3
          Clr       EECON1:#EEPGD
          Set       EECON1:#RD        ; Read bit
          Nop
          Nop
  Bank3to2
          Move      EeData,W
  Bank2to0
          Or        #0,W       ; Test if W is zero
          Skip,NE
          Ret                  ; All text has been scanned
          Call      SndSer
          Inc       EeAdr
          Jump      T$
```

Texts are directly stored in memory with the .16 pseudo.

```
  TextHello:        .16       "H","e","l","l","o"
  TextOK:           .16       "O","K"
```

When preparing the parameter before calling for a text, one needs to mask the high address byte (it is defined inside the routine). Having all texts in the same page simplifies greatly the routine and its call.

```
          Move      #TextHello.AND.16'FF,W
          Call      AfText
```

Displaying a variable

An 8-bit word from an I/O port or a variable is at best represented on the terminal as two 4-bit nibbles, coded in Ascii. In few applications, a conversion to decimal is usefull. Decimal is displayed with the hexadecimal routine.

Converting binary to hex is a little tricky due to the fact that digits 0 to 9 (4-bit nibbles) are mapped on Ascii codes "0" to "9", and the next 6 values are not consecutive on the Ascii code list. The algorithm is explained in www.didel.com/doc/DopicHex.pdf.

The SndHex routine sends the content of W as two hexa characters. W is saved in variable SavBin for possible usage at when returning from the routine.

Routine **SndHex** **Send an 8 bit byte as two hexadecimal characters**

```
; Call routine SndSer
  in:  W
  mod: SavBin = copy of initial W value
```

Routine SendHex can be found in XTrSndR.asi module, which does not includes the SndSer and RecSer routines, which depends on the hardware (XSerR.asm for 16F84, XSer7R.asm for 16F87x).

As an example, let us display the Ascii code of the letter typed on the terminal (16F87x):

```
Program XTrSer7.asm   Test série RecSer SndHex
PicGénial 870
.Proc 16F870
.Ref  16F870
.List 0
.Ins  16F870M.asi
.Endlist

Constant PortA et B  DirA      = -1         ; entrées
DirB    = 0           ; sorties

Constant PortC  bTxD           = 6
bRxD    = 7
DirC    = 2'10011000
IniRcSta          = 2'10011000
IniTxSta          = 2'00100100
IniSpBrg          = 10'25
PortSer           = 7          ; id
Ascii
CR      = 16'0D
LF      = 16'0A
BEL     = 16'07
```

```
Variables  Globales
.Loc  DebVar
.Ins  XSer7V.asi
.Ins  XTrSerV.asi
.Loc  0

Program Programme
Start:
        Move        #DirA,W
        Move        W,TrisA
        Move        #DirB,W
        Move        W,TrisB
        Call        IniSer
Loop:                Echo
        Call        RecSer
        Move        W,PortB
        Not         PortB          ; Code on Leds
        Call        SndHex
        Call        SndSpace
        Jump        Loop
.Ins  XSer7R.asi
.Ins  XTrSerR.asi
.End
```

It is important to remember that the output parameter of RecSer is W, and the input parameter of SndHex is also W. W is not modified between the two calls. But now, if one needs to improve the legibility and put a space between the character and its code (the terminal is supposed to be in "echo" mode), there is a problem.

```
Call     RecSer
Call     SndSpace
Call     SndHex
```

A wrong hexadecimal value will be displayed, since SndSpace modify W. One needs to remember that RecSer saves also the result in DataRec variable, where we can get it back after the "Call SndSpace".

```
Call     RecSer
Call     SndSpace
Move     DataRec,W
Call     SndHex
```

Displaying in decimal

It is frequently usefull to display a binary value (0 to 16'FF) in decimal (0 to 256). Binary-decimal routines are explained in www.didel.com/doc/DopiBinD.pdf. The conversion routine in TrHexR.asi file sends the characters to the serial line as soon as converted.

Routine **SndBinDec** 080201

```
; Convert a binary number 0-255 and sends it as 2 or 3 Ascii digits
; 16'C0 (192) --> 1 9 2   16'F --> 1 5   16'5 --> 0 5
  in:  SavBin  8 bits value
  out: 2 or 3 caracters
```

Receiving an 8-bit variable value

The RecHex routine reads two Ascii characters from the keybord, and returns the concatenated 8-bit word both in W and in SavBin. Its listing is explained in www.didel.com/doc/DocPicHex.pdf. As routine users, we just need to know what are the parameters in and out.

Routine **RecHex** **Read hex digits until a code <"0" is typed**

out: W = SavBin value, DataRec last character (terminator)

More than two digits can be typed, but only the last two will be stored. This is convenient if there is a typing error: two zero clear the buffer,and the corrected value can be typed. The terminator can be one of the following signs: CR Escape Space, ! " # & % ´ ( ) * + , - . /. With 5 to 10 more instructions in the routine, any non hexa character could terminate. With our short routine, many characters should not be typed.

As example, if one needs to check how the terminal respond to special characters, on can write the following test loop. Typing 41\space; (a space or CR as terminator) will display A. Typing 7\space; should ring the bell on the PC.

```
L$:     Call    RecHex
        Call    SndSer
        Jump    L$
```

Again, if a space must be added, W will be destroyed, but the binary value can be found in SavBin.