Software driven serial communication routines for the 83C751 and 83C752 microcontrollers

AN423

DESCRIPTION

The need often arises to make use of a serial port in connection with a microcontroller that does not have a hardware UART on-chip. Aside from the obvious cases where the microcontroller application intrinsically requires RS-232 communications to achieve its purpose, a serial output may often be a simple and convenient method of providing detailed diagnostic information to the outside world while using only a single I/O port pin. In many cases, the solution may be to implement the UART function in software. The routines included here demonstrate a method to add such a function to a microcontroller without the benefit of a hardware UART.

Examples of microcontrollers that do not have on-chip UARTs are the 83C751 and 83C752. While it is possible to connect an external UART chip to these microcontrollers, it tends to use up many I/O port pins and begins to become less economical than simply using a standard 80C51. The are several factors to be considered in deciding if the software UART method will be usable in a particular application. The first is whether the serial communication channel is to be simplex (transmit only or receive only), half-duplex (transmit and receive, but not simultaneously), or full-duplex (simultaneous transmit and receive). Both simplex and half-duplex operation are fairly easy to implement in software on an 80C51-type microcontroller, and will be covered by this application note. Full-duplex operation is more difficult to implement in software and can use up a large portion of the microcontroller's time and resources.

A second consideration to be taken into account is the amount of system resources that will be "'used up" by the serial communication software. First of all, such software routines will almost always require the use of at least one counter/timer to generate the time slices for the serial bit cells. Next, the physical connection to the outside world will require one I/O port pin each for the serial input and the serial output. Moreover, the port pin used for serial input should be an external interrupt input pin. This allows the software to be interrupted automatically at the beginning of an incoming start bit and synchronizes the timer accurately to the serial data stream. Additional port pins may be used to implement signals such as Request to Send (RTS), Clear to Send (CTS), etc.

Finally, serial communication software will take up a certain amount of CPU time, more than would be required to operate a hardware UART. The overhead of software implemented serial communication may or may not be an issue, depending on the application, the throughput of the serial channel(s), the baud rate, other tasks the CPU is handling and how time-critical they are, etc.

The program listing that is included here is a demonstration of half-duplex serial routines on the 83C751 or 83C752 microcontrollers. The operation of the software would be the same on other 80C51 derivatives, except that the counter/timer operation is slightly different. The program, as listed, will send a canned message to the serial output (port pin P1.0 in this case), then wait for data on the serial input (port pin P1.5/INT0). When a character has been received on the serial input, it will be echoed through the serial output. Since the software is inherently half-duplex, the rate at which characters are received must be less than half the rate that would be possible on a full-duplex channel. This example has been set up to receive and transmit at 9600 baud when run with a 16 MHz crystal.

The operation of the routines is fairly straightforward. Beginning with a start bit occurring on the serial input line, an interrupt (external interrupt 0) will occur. At the interrupt service routine Int0, the counter/timer is loaded with a value that will result in a time delay that is approximately equivalent to half a bit cell time for the baud rate being used, less some constant to account for the elapsed time between a timer interrupt and the point where the serial input is actually sampled. The timer reload register is loaded with a value that will result in a time delay that is as close as can be calculated to one full bit cell time. The program then starts the timer and simply returns to the main program, waiting for the timer to time out, generating another interrupt.

At that point, the serial start bit should be about halfway through its nominal duration. When the first timer interrupt occurs, the timer interrupt routine Timr0 calls the receive bit routine RxBit which checks to make sure that the start bit is still valid and flags an error if it is not. The RxBit routine will then return control to the main program routine, waiting for the next timer interrupt.

On the second timer interrupt, the RxBit routine reads the serial input line and shifts the value into the serial holding register RxDat. This process is repeated until 8 bits have been read in on consecutive timer interrupts. Finally, on the tenth timer interrupt, the receive routine looks for a valid stop bit and flags an error if one is not detected. At this point, the RcvRdy flag is set to inform the main program that a character is waiting in the holding register.

The transmit routine works in a somewhat similar fashion, beginning with a call to the byte transmit routine XmtByte, which first checks to make sure that a byte receive operation is not already in progress. The RSXmt routine will then set up the timer and timer reload registers to correspond to one bit cell time, start the timer, and assert a start bit.

At each subsequent timer interrupt, the routine TxBit shifts out the next bit from the transmit holding register XmtDat, until all 8 bits have been transmitted. Once all of the data has been sent, the stop bit is asserted on the next timer interrupt. A final timer interrupt is required to insure that the stop bit lasts at least one full bit cell time. At this point, transmit flag TxFlag is cleared in order to inform the main program that the transmission is completed.

A few other useful routines are embedded in the sample program: PrByte, which converts a byte of data to hexadecimal form and transmits it; HexAsc, which converts one nibble of raw data to hexadecimal form; and Mess, which transmits an absolute string of data (usually a text message) which is terminated by a 0 byte.

This demonstration of software driven serial port routines uses 5 bytes of microcontroller RAM, two port bits (including one external interrupt input), one counter/timer, and about 256 bytes of code space, excluding the message string at the end of the listing.

```
RS751          Half-Duplex Serial Communication Routines                              11/14/89

                ;**************************************************************************
                ;         Software Driven Half-Duplex Serial Communication Routines
                ;               for 83C751 and 83C752 series Microcontrollers
                ;
                ;**************************************************************************
                $Title(Half-Duplex Serial Communication Routines)
                $Date(11/14/89)
                $MOD751
                ;**************************************************************************
  FF75          BaudVal  EQU      -139                ;Timer value for 9600 baud @ 16 MHz.
                                                      ;(one bit cell time)
  FFD9          StrtVal  EQU      -39                 ;Timer value to start receive.
                                                      ;(half of one bit cell time, minus the
                                                      ;time it takes the code to sample RxD)
  0010          XmtDat   DATA     10h                 ;Data for RS-232 transmit routine.
  0011          RcvDat   DATA     11h                 ;Data from RS-232 receive routine.
  0012          BitCnt   DATA     12h                 ;RS-232 transmit & receive bit count.
  0013          LoopCnt  DATA     13h                 ;Loop counter for test routine.
  0020          Flags    DATA     20h
  0000          TxFlag   BIT      Flags.0             ;Receive-in-progress flag.
  0001          RxFlag   BIT      Flags.1             ;Transmit-in-progress flag.
  0002          RxErr    BIT      Flags.2             ;Receiver framing error.
  0003          RcvRdy   BIT      Flags.3             ;Receiver ready flag.
  0090          TxD      BIT      P1.0                ;Port bit for RS-232 transmit.
  0095          RxD      BIT      P1.5                ;Port bit for RS-232 receive (INT0).
                ;**************************************************************************
                ; Interrupt Vectors
0000                     ORG      0                   ;Reset vector.
0000 0124                AJMP     Reset
0003                     ORG      03H                 ;External interrupt 0.
0003 019F                AJMP     ExInt0              ;Indicates RS-232 start bit received.
000B                     ORG      0BH                 ;Timer 0 interrupt.
000B 0175                AJMP     Timr0               ;Baud rate generator.
0013                     ORG      13H                 ;External interrupt 1 (not used).
0013 32                  RETI
001B                     ORG      1BH                 ;Timer I interrupt (not used).
001B 32                  RETI
0023                     ORG      23H                 ;I2C interrupt (not used).
0023 32                  RETI
                ;**************************************************************************
                ;Simple test of RS-232 transmit and receive.
0024 758130     Reset:   MOV      SP,#30h
0027 752000              MOV      Flags,#0            ;Clear RS-232 flags.
002A C201                CLR      RxFlag
002C 758800              MOV      TCON,#00h           ;Set up timer controls.
002F 75A882              MOV      IE,#82h             ;Enable timer 0 interrupts.
0032 751310              MOV      LoopCnt,#16         ;Test transmit first.
0035 7900                MOV      R1,#0               ;Zero line count.
0037 90010C              MOV      DPTR,#Msg1          ;Point to message string.
```

```
003A 11FB      Loop1:   ACALL   Mess               ;Send an RS–232 message repeatedly.
003C 743A               MOV     A,#':'
003E 1154               ACALL   XmtByte
0040 E9                 MOV     A,R1
0041 11DD               ACALL   PrByte             ;Print R1 contents.
0043 09                 INC     R1                 ;Advance R1 value.
0044 D513F3             DJNZ    LoopCnt,Loop1

0047 D2A8      Loop2:   SETB    EX0                ;Enable interrupt 0 (RS–232 receive).
0049 3003FD             JNB     RcvRdy,$           ;Wait for data available.
004C C203               CLR     RcvRdy
004E E511               MOV     A,RcvDat           ;Echo same byte.
0050 1154               ACALL   XmtByte
0052 80F3               SJMP    Loop2

               ; Send a byte out RS–232 and wait for completion before returning.
               ; (use if there is nothing else to do while RS–232 is busy)

0054 2001FD    XmtByte: JB      RxFlag,$           ;Wait for receive complete.
0057 115D               ACALL   RSXmt              ;Send ACC to RS–232 output.
0059 2000FD             JB      TxFlag,$           ;Wait for transmit complete.
005C 22                 RET

               ; Begin RS–232 transmit.

005D F510      RSXmt:   MOV     XmtDat,A           ;Save data to be transmitted.
005F 75120A             MOV     BitCnt,#10         ;Set bit count.
0062 758CFF             MOV     TH,#High BaudVal   ;Set timer for baud rate.
0065 758A75             MOV     TL,#Low BaudVal
0068 758DFF             MOV     RTH,#High BaudVal  ;Also set timer reload value.
006B 758B75             MOV     RTL,#Low BaudVal
006E D28C               SETB    TR                 ;Start timer.
0070 C290               CLR     TxD                ;Begin start bit.
0072 D200               SETB    TxFlag             ;Set transmit–in–progress flag.
0074 22                 RET

               ; Timer 0 timeout: RS–232 receive bit or transmit bit.

0075 C0E0      Timr0:   PUSH    ACC
0077 C0D0               PUSH    PSW
0079 20013E             JB      RxFlag,RxBit       ;Is this a receive timer interrupt?
007C 200007             JB      TxFlag,TxBit       ;Is this a transmit timer interrupt?
007F C28C      T0Ex1:   CLR     TR                 ;Stop timer.
0081 D0D0      T0Ex2:   POP     PSW
0083 D0E0               POP     ACC
0085 32                 RETI

               ; RS–232 transmit bit routine.

0086 D51204    TxBit:   DJNZ    BitCnt,TxBusy      ;Decrement bit count, test for done.
0089 C200               CLR     TxFlag             ;End of stop bit, release timer.
008B 80F2               SJMP    T0Ex1              ;Stop timer and exit.

008D E512      TxBusy:  MOV     A,BitCnt           ;Get bit count.
008F B40104             CJNE    A,#1,TxNext        ;Is this a stop bit?
0092 D290               SETB    TxD                ;Set stop bit.
0094 80EB               SJMP    T0Ex2              ;Exit.

0096 E510      TxNext:  MOV     A,XmtDat           ;Get data.
0098 13                 RRC     A                  ;Advance to next bit.
0099 F510               MOV     XmtDat,A
009B 9290               MOV     TxD,C              ;Send data bit.
009D 80E2               SJMP    T0Ex2              ;Exit.

               ;Begin RS–232 receive (after external interrupt 0).

009F 75120A    ExInt0:  MOV     BitCnt,#10         ;Set receive bit count.
00A2 758CFF             MOV     TH,#High StrtVal   ;First timeout in HALF a bit time.
```

```
00A5 758AD9               MOV      TL,#Low StrtVal
00A8 758DFF               MOV      RTH,#High BaudVal   ;Set timer reload for baud rate.
00AB 758B75               MOV      RTL,#Low BaudVal
00AE 751100               MOV      RcvDat,#0           ;Initialize received data to 0.
00B1 C2A8                 CLR      EX0                 ;Disable external interrupt 0.
00B3 C202                 CLR      RxErr               ;Clear error flag.
00B5 D28C                 SETB     TR                  ;Start timer.
00B7 D201                 SETB     RxFlag              ;Set receive–in–progress flag.
00B9 32                   RETI


                 ; RS–232 receive bit routine.

00BA D5120D      RxBit:   DJNZ     BitCnt,RxBusy       ;Decrement bit count, test for stop.
00BD 209502               JB       RxD,RxBitEx         ;Valid stop bit?
00C0 D202        RxBtErr: SETB     RxErr               ;Bad stop bit, tell mainline.
00C2 C201        RxBitEx: CLR      RxFlag              ;Release timer for other purposes.
00C4 D2A8                 SETB     EX0                 ;Re–enable external interrupt 0.
00C6 D203                 SETB     RcvRdy              ;Tell mainline that a byte is ready.
00C8 80B5                 SJMP     T0Ex1               ;Stop timer and exit.

00CA E512        RxBusy:  MOV      A,BitCnt            ;Get bit count.
00CC B40905               CJNE     A,#9,RxNext         ;Is this a start bit?
00CF 2095EE               JB       RxD,RxBtErr         ;Valid start bit?
00D2 80AD                 SJMP     T0Ex2               ;Exit.

00D4 E511        RxNext:  MOV      A,RcvDat            ;Get partial receive byte.
00D6 A295                 MOV      C,RxD               ;Get receive pin value.
00D8 13                   RRC      A                   ;Shift in new bit.
00D9 F511                 MOV      RcvDat,A            ;Save updated receive byte.
00DB 80A4                 SJMP     T0Ex2               ;Exit.


                 ; Print byte routine: print ACC contents as ASCII hexadecimal.

00DD C0E0        PrByte:  PUSH  ACC
00DF C4                   SWAP  A
00E0 11EB                 ACALL HexAsc
00E2 1154                 ACALL XmtByte
00E4 D0E0                 POP   ACC
00E6 11EB                 ACALL HexAsc              ;Print nibble in ACC as ASCII hex.
00E8 1154                 ACALL XmtByte
00EA 22                   RET


                 ; Hexadecimal to ASCII conversion routine.

00EB 540F        HexAsc:  ANL   A,#0FH              ;Convert a nibble to ASCII hex.
00ED 30E308               JNB   ACC.3,NoAdj
00F0 20E203               JB    ACC.2,Adj
00F3 30E102               JNB   ACC.1,NoAdj
00F6 2407        Adj:     ADD   A,#07H
00F8 2430        NoAdj:   ADD   A,#30H
00FA 22                   RET


                 ; Message string transmit routine.

00FB C0E0        Mess:    PUSH     ACC
00FD 7800                 MOV      R0,#0               ;R0 is character pointer (string
00FF E8          Mesl:    MOV      A,R0                ; length is limited to 256 bytes).
0100 93                   MOVC     A,@A+DPTR           ;Get byte to send.
0101 B40003               CJNE     A,#0,Send           ;End of string is indicated by a 0.
0104 D0E0                 POP      ACC
0106 22                   RET

0107 1154        Send:    ACALL    XmtByte             ;Send a character.
0109 08                   INC      R0                  ;Next character.
010A 80F3                 SJMP     Mesl

010C 0D0A        Msg1:    DB       0Dh, 0Ah
```

```
010E 54686973        216            DB       'This is a test of the software serial routines.', 0
0112 20697320
0116 61207465
011A 7374206F
011E 66207468
0122 6520736F
0126 66747761
012A 72652073
012E 65726961
0132 6C20726F
0136 7574696E
013A 65732E00
                     217
                     218            END

ASSEMBLY COMPLETE, 0 ERRORS FOUND


ACC. . . . . . . . . . . . . . . . D ADDR  00E0H  PREDEFINED
ADJ. . . . . . . . . . . . . . . . C ADDR  00F6H
BAUDVAL. . . . . . . . . . . . . .   NUMB  FF75H
BITCNT . . . . . . . . . . . . . . D ADDR  0012H
EX0. . . . . . . . . . . . . . . . B ADDR  00A8H  PREDEFINED
EXINT0 . . . . . . . . . . . . . . C ADDR  009FH
FLAGS. . . . . . . . . . . . . . . D ADDR  0020H
HEXASC . . . . . . . . . . . . . . C ADDR  00EBH
IE . . . . . . . . . . . . . . . . D ADDR  00A8H  PREDEFINED
LOOP1. . . . . . . . . . . . . . . C ADDR  003AH
LOOP2. . . . . . . . . . . . . . . C ADDR  0047H
LOOPCNT. . . . . . . . . . . . . . D ADDR  0013H
MESL . . . . . . . . . . . . . . . C ADDR  00FFH
MESS . . . . . . . . . . . . . . . C ADDR  00FBH
MSG1 . . . . . . . . . . . . . . . C ADDR  010CH
NOADJ. . . . . . . . . . . . . . . C ADDR  00F8H
P1 . . . . . . . . . . . . . . . . D ADDR  0090H  PREDEFINED
PRBYTE . . . . . . . . . . . . . . C ADDR  00DDH
PSW. . . . . . . . . . . . . . . . D ADDR  00D0H  PREDEFINED
RCVDAT . . . . . . . . . . . . . . D ADDR  0011H
RCVRDY . . . . . . . . . . . . . . B ADDR  0003H
RESET. . . . . . . . . . . . . . . C ADDR  0024H
RSXMT. . . . . . . . . . . . . . . C ADDR  005DH
RTH. . . . . . . . . . . . . . . . D ADDR  008DH  PREDEFINED
RTL. . . . . . . . . . . . . . . . D ADDR  008BH  PREDEFINED
RXBIT. . . . . . . . . . . . . . . C ADDR  00BAH
RXBITEX. . . . . . . . . . . . . . C ADDR  00C2H
RXBTERR. . . . . . . . . . . . . . C ADDR  00C0H
RXBUSY . . . . . . . . . . . . . . C ADDR  00CAH
RXD. . . . . . . . . . . . . . . . B ADDR  0095H
RXERR. . . . . . . . . . . . . . . B ADDR  0002H
RXFLAG . . . . . . . . . . . . . . B ADDR  0001H
RXNEXT . . . . . . . . . . . . . . C ADDR  00D4H
SEND . . . . . . . . . . . . . . . C ADDR  0107H
SP . . . . . . . . . . . . . . . . D ADDR  0081H  PREDEFINED
STRTVAL. . . . . . . . . . . . . .   NUMB  FFD9H
T0EX1. . . . . . . . . . . . . . . C ADDR  007FH
T0EX2. . . . . . . . . . . . . . . C ADDR  0081H
TCON . . . . . . . . . . . . . . . D ADDR  0088H  PREDEFINED
TH . . . . . . . . . . . . . . . . D ADDR  008CH  PREDEFINED
TIMR0. . . . . . . . . . . . . . . C ADDR  0075H
TL . . . . . . . . . . . . . . . . D ADDR  008AH  PREDEFINED
TR . . . . . . . . . . . . . . . . B ADDR  008CH  PREDEFINED
TXBIT. . . . . . . . . . . . . . . C ADDR  0086H
TXBUSY . . . . . . . . . . . . . . C ADDR  008DH
TXD. . . . . . . . . . . . . . . . B ADDR  0090H
TXFLAG . . . . . . . . . . . . . . B ADDR  0000H
TXNEXT . . . . . . . . . . . . . . C ADDR  0096H
XMTBYTE. . . . . . . . . . . . . . C ADDR  0054H
XMTDAT . . . . . . . . . . . . . . D ADDR  0010H
```