Application Note 75
Using the High-Speed Micro's Serial Ports

www.dalsemi.com

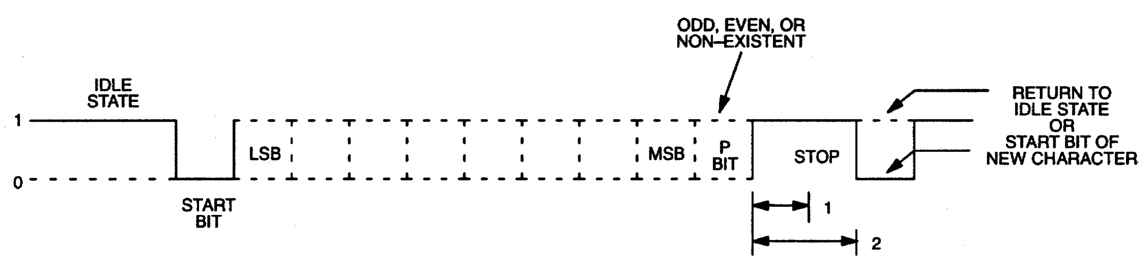
INTRODUCTION

The High-Speed Microcontroller's serial interfaces are functionally identical to those found on other lower performance 8051 processors. They are based on a Universal Synchronous/Asynchronous Receiver/Transmitter (USART) implementation. As the name implies, a USART converts parallel data to and from a synchronous or asynchronous serial bit stream. The parallel data side of the device interfaces with the processor's internal data bus, and the serial side interfaces with the outside world. This application note describes the setup and operation of the most commonly encountered operating modes of this interface.

Because of its universal applicability, the most frequently used configuration of the High-Speed Micro's serial channel is its 10-bit asynchronous mode. In this application note, this configuration will be described in detail. A general overview of the port's operation will be provided and a detailed software example will be presented. Examples illustrating the use of dual serial ports and 11-bit address recognition capability will also be presented. Through these three examples, different aspects of serial port operation will be discussed.

The most frequently encountered serial communication modes are based on asynchronous data transmission. In this mode of transmission, there is no separate clock signal. The classical serial asynchronous data communications format illustrated in Figure 1 provides the necessary synchronization without a clock signal. In this format, 8 or 9 data bits are accompanied by one start bit and one or two stop bits. The 9th data bit is frequently used as a parity bit. The start and stop bits provide the necessary synchronization information. The serial bit stream's content in this mode is compatible with the popular RS-232 protocol. However, the signal levels are not. For signal level compatibility, a level translator such as the DS232A must be used to convert TTL/ CMOS levels to/from RS-232 levels.

ASYNCHRONOUS DATA FORMAT Figure 1



Please note that many members of the High-Speed Microcontroller family have two functionally identical serial ports. As new members are added to the processor family, some may not have both serial ports. Refer to the individual device's data sheet for specific features. This application note will concentrate on the use of a single serial port (port 0), but in the examples, the ideas presented are equally applicable to both ports. One example program demonstrates the use of both serial ports.

021401

BAUD CLOCK SOURCES

While it is not the purpose of this document to discuss details of the High-Speed Micro's internal timers, there must be some discussion of them since they are frequently used as the source of the baud rate clock. The High-Speed Micro contains three internal timers, two of which may be used as baud rate generators. The following sections briefly describe these timers and their modes most commonly used for baud rate generation.

Detailed information on the timers' setup will be illustrated by the software examples that follow.

On both timers, auto-reload mode is frequently used for baud rate generation because it requires no intervention by the processor. Once initialized, the timers run automatically, producing a baud rate clock based on a value loaded into the timers' reload registers. When in this mode, timer 1 uses SFR registers TL1 to count and TH1 to store a reload value. Software must initialize TH1 with the desired reload value, and if the first time interval is to be correct, TL1 must also be loaded with the same value. Timer 2 uses registers TL2 and TH2 for counting, and registers RCAP2L and RCAP2H for holding the reload values. Again, these registers must be initialized by software. When enabled, the timers will begin counting based on the selected clock source. The clocks possible for timer 1 are oscillator/12 or oscillator/4. For timer 2, oscillator/2 is the only possibility when in baud clock generation mode. When the count registers roll over from their maximum count to 0, they will be automatically reloaded with the value in the reload registers. The reload value remains unchanged unless modified by software. Every time the rollover occurs, a clock pulse is generated. This clock pulse is used by the serial port as a baud rate clock. By changing the reload value, a wide variety of baud rates may be achieved.

POLLED VS. INTERRUPT DRIVEN MODES

The High-Speed Micro's serial channels have flag bits in the SFR address space that indicate the status of the channel. For each serial channel, there is a flag indicating when a character has been received (RI) and a flag indicating when a character has been transmitted (TI). These flags are set whether or not the associated interrupts are enabled, and can therefore be used in a polled mode of operation. Alternatively if the interrupts are enabled, the setting of either of these flags will cause a jump to the serial channel's associated interrupt vector.

Since the serial interrupts are asynchronous, based on serial communications with an external device, most applications will benefit from using an interrupt driven communications scheme. In this way, the processor can accomplish other tasks while it is waiting to receive an interrupt. If a polling method were used, time would be consumed continually checking the flag bits to see if one has been set. The examples of this application note illustrate both approaches. The first example demonstrates a typical interrupt driven mode of operation. The second example executes a very structured set of events, so it is well suited to polled operation.

INSERTING BREAK CHARACTERS

A break character is a very long null in the communications stream. The exact length varies according to the communication format used. Software can easily create a null by writing a logic 0 to the port latch bit of the RX pin of the appropriate serial port. Note that writing 00h to SBUF0 or SBUF1 will not achieve the desired effect because the start and stop bits used in the data stream are logic 1. At the conclusion of the break character software needs only to write a logic 1 to the port latch bit of the RX pin of the appropriate serial port.

ASYNCHRONOUS 10-BIT MODE EXAMPLE

The asynchronous 10-bit mode of operation is arguably the most frequently used method of serial communication on the 8051 family. This is because this mode is compatible with the familiar RS-232 protocol. While the signal levels are different, the use of a simple level translator will allow communications with the standard serial ports of any personal computer. In fact, all of the software in this applications note was tested using a PC to interface to a DS80C320 test board.

This particular serial mode can use timer 1 to generate baud rates for serial port 1 or timers 1 or 2 for serial port 0. In this example, timer 2 is operated in auto-reload mode to generate the baud rate for serial port 0.

After establishing the timer's mode of operation, the reload value must be stored in the reload register. The contents of the reload registers may be calculated for a desired baud rate and oscillator frequency using the following equation:

$$\text{RCAP2H, RCAP2L} = 65536 - \frac{\text{Oscillator Frequency}}{32 * \text{Baud rate}}$$

Using this equation, the reload value for any desired baud rate and oscillator frequency may be calculated. For this software example, an oscillator frequency of 11.0592 MHz is assumed. The table below shows the reload values for several common baud rates based on this crystal frequency. It should be noted that not every crystal value will produce acceptable baud rates. Some evaluation of the above equation may be necessary before selecting the crystal for a system if a specific baud rate is required.

TIMER 2 RELOAD VALUES Table 1

BAUD RATE	RCAP2H	RCAP2L
57600	0FFh	0FAh
9600	0FFh	0DCh
2400	0FFh	090h
1200	0FEh	0E0h

Once a reload value is determined, it must then be loaded into the timer's reload registers to establish the timer's output clock frequency.

After initializing the timer, the serial port may be set up as desired. To establish the correct operating mode for serial port 0, the SM0 and SM1 bits of the SCON register (address 098h) must be set appropriately. The following table shows the possible settings of these bits and the resulting mode. As shown, SM0 and SM1 (SCON0.1 and SCON0.0) must be set to 0 and 1 respectively for 10-bit asynchronous operation.

SERIAL MODE BITS Table 2

SM0	SM1	MODE	FUNCTION	LENGTH	PERIOD
0	0	0	Sync	8 bits	4/12 t_{CLK}
0	1	1	Async	10 bits	Timer 1 or 2*
1	0	2	Async	11 bits	64/32 t_{CLK}
1	1	3	Async	11 bits	Timer 1 or 2*

*Timer 2 is only available for baud rate generation on serial port 0.

Since the serial port will be operated in interrupt driven mode, the interrupt enables must be set appropriately. By setting the ES0 (address 0ACh) and EA (address 0AFh) bits to 1, serial port 0 will generate an interrupt when a character has been transmitted or when a character has been received.

As a final step in initialization, the timer is started by setting bit TR2 (address 0CAh). At this point, serial communications may begin. To transmit a character, the character is written to the SBUF (byte address 099h) and other tasks are performed until an interrupt is received (in this case a tight loop). In receiving a character, no action is required until an interrupt occurs. Since either a transmit or a receive can cause a jump to the same interrupt vector, the interrupt service routine (ISR) must determine which was the cause. This is done by reading the TI (bit address 099h) and RI (bit address 098h) status bits of the SCON register. If TI is set, then a "Transmit Complete" caused the interrupt. If RI is set, then a "Receive" caused the interrupt. If a transmit caused the interrupt, the TI bit may be cleared and the ISR exited. If a receive caused the interrupt, then the RI bit must be cleared and the received character must be read from SBUF.

The software listing for example 1 below illustrates the details of implementing this mode of serial operation.

DUAL SERIAL PORT EXAMPLE

This example demonstrates the use of the two serial ports available on the DS80C320. The main purpose of the example is to illustrate how to initialize and use the second port. As discussed earlier, much of the information regarding serial port 0 applies equally to serial port 1. However, this example will help clarify any confusion about the use of this resource.

In this example, the output of port 1 (TXD1) is connected to its input (RXD1) in a "loop back" configuration. The software is written to create a closed serial loop with port 0 as input and output. A terminal or PC running a terminal emulator is connected to port 0's input (RXD0) and output (TXD0). Initially, software outputs a three-line message to the terminal on port 0, and the DS80C320 waits for an input. When the terminal sends a character to the DS80C320, the RI bit is set. When the software recognizes this bit has been set, it reads the received character and transfers it to the transmit buffer of port 1. For illustrative purposes, the character is converted to upper case (if not already) before it is transferred. Since the output of port 1 is tied to its input, the transmitted character automatically ends up in the receive buffer. The software then copies this character to port 0's transmit buffer, which causes it to be transmitted out of the processor. Finally, the character arrives at the terminal as an upper case character, thereby completing the loop.

In this software example, both serial ports are set to run from a baud clock generated from Timer 1. The timer is set for auto-reload mode, and the count and reload registers are loaded with the appropriate value. Timer 1's equation for calculating reload values is different from Timer 2's, and is shown below:

$$\text{Reload} = 256 - \frac{2^{\text{SMOD}} * \text{Oscillator Freq.}}{384 * \text{Baud rate}}$$

Using the above equation, the reload value for a desired baud rate and oscillator frequency may be calculated. It can be seen that the reload value is a function of 2 raised to the power of SMOD. Since SMOD can be either 0 or 1, this term can be either 1 or 2 ($2^0 = 1$, $2^1 = 2$). Therefore, setting SMOD to 1 has the effect of doubling the baud rate. Again for this software example, an oscillator frequency of 11.0592 MHz is assumed. Table 3 below shows the reload values calculated for several common baud rates based on this crystal frequency.

TIMER 1 RELOAD VALUES

BAUD RATE	SMOD	RELOAD
57600	1	FF
19200	1	FD
9600	0	FD
2400	0	F4
1200	0	E8

As seen in the above equation, the reload value is calculated based on the SMOD baud rate doubler bit's setting. If this bit is 0 for the desired baud rate, it is not necessary to clear it in the initialization software because this is its reset default condition. However for clarity, the instructions to clear this bit for both ports are included in the example. Since the SMOD bit is not "bit addressable" you must write to the entire PCON register. The example code shows instructions for clearing and setting the SMOD bit using a single logical instruction. The instruction not used in the program is commented out.

In the example, the timer mode is initialized, the count and reload registers are loaded, the two SMOD bits are cleared, and the timer interrupt is disabled. This completely configures the baud clock generation. After the two serial control registers are set for the desired mode, the timer is started, and serial communication begins.

This example handles serial communication differently than illustrated by the earlier example. This example uses a polled mode to monitor serial status. Since this example is more structured in its operation, this polled approach is appropriate. The basic functions that transfer characters, are GETCH and PUTCH. Both of these functions perform a tight loop waiting for the appropriate flag to be set. When it is, the program continues. This would normally be considered a waste of time, but in this application and others like it, polled operation makes sense.

The software listing for example 2 illustrates the details of implementing this mode of serial operation.

ADDRESS RECOGNITION EXAMPLE

This example demonstrates the address recognition capability of the High-Speed Micro’s serial channel. In addition, it illustrates a method of baud clock generation that does not involve the use of a timer (i.e., serial mode 2).

The address recognition feature of the High-Speed Micro is frequently used for multi-processor communications. Each processor on the bus can be assigned a unique address. When configured, the processors will not recognize any serial communications unless its address is matched. Full details of this mode of operation may be found in the High-Speed Microcontroller User’s Guide.

In this example, the address is set to recognize a control C character (03h). Any character received before a control C is ignored. However, when a control C is received, a character string is immediately printed, and subsequent characters received on RXD0 are echoed out TXD0.

As mentioned, the illustrated mode of generating a baud rate does not involve a timer (i.e., serial mode 2). The baud rate is selectable as shown in the following equation:

$$\text{baud rate} = \frac{2^{\text{SMOD}} * \text{Oscillator Freq.}}{64}$$

As can be seen in the equation, the selection of crystal frequencies is much more restricted for this serial mode than others if a particular baud rate is desired. In fact, there are relatively few crystal frequencies available that will produce a standard baud rate. In this example, an oscillator frequency of 7.372 MHz is assumed, which will result in a rate of 115,200 baud with SMOD cleared to 0. It is quite common to see an oscillator of 12.0 MHz that will produce a rate of 187,500 baud with SMOD cleared to 0.

The software listing for example 3 illustrates the details of implementing this mode of serial operation.

CODE EXAMPLE # 1 : 10-BIT ASYNCHRONOUS MODE

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * * * * * * * * * * * * Program ASYNC10B.ASM * * * * * * * * * * * *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; This program sets up the DS80C320's Port 0 serial port to operate
; in 10-bit asynchronous format. Timer 2 is used as the baud rate
; generator by setting it in auto-reload mode. A crystal value of
; 11.0529 MHz is assumed for the baud rate reload values.
;
;
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;               Interrupt Vectors
;
          ORG        0000h          ; RESET VECTOR
          AJMP       START          ; Jump to start of program
;
          ORG        0023h          ; SERIAL PORT 0 VECTOR
          AJMP       SERINT         ; Jump to serial interrupt routine
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;         Initialization
;
          ORG        0100h
;
START:    MOV        IE, #0         ; Disable ALL interrupts
;
;                    Set up timer 2
;
          MOV        T2CON, #030h   ; Timer 2 : auto-reload mode for
                                    ; both receive & transmit baud clocks
          MOV        RCAP2L, #T2RL96L ; Establish 16-bit reload value for 9600
          MOV        RCAP2H, #T2RL96H
          MOV        TL2, #T2RL96L  ; Make first timeout correct
          MOV        TH2, #T2RL96H
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;
;         Set up Serial Port 0
;
          MOV        SCON, #050h    ; Mode 1, set receive enable
          SETB       ES0            ; Enable Serial Port interrupt
          ;
          SETB       EA             ; Set Global Interrupt enable
          SETB       TR2            ; Start Timer
;
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * * * * * * * * * Test Code to exercise the serial port * * * * *
;
          MOV        SBUF, #'D'     ; Put character 'D' in buffer (send it)
          CALL       WAIT           ; Wait until flag cleared from interrupt
                                    ; service routine.
          MOV        SBUF, #'S'
          CALL       WAIT
          MOV        SBUF, #'8'
          CALL       WAIT
          MOV        SBUF, #'0'
          CALL       WAIT
          MOV        SBUF, #'C'
          CALL       WAIT
          MOV        SBUF, #'3'
          CALL       WAIT
          MOV        SBUF, #'2'
          CALL       WAIT
          MOV        SBUF, #'0'
          CALL       WAIT
          MOV        SBUF, #00Dh
          CALL       WAIT
          MOV        SBUF, #00Ah
          CALL       WAIT
          ;
          SJMP       $
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * * * * * * * * * * * Subroutine to wait for a serial interrupt * *
WAIT:     MOV        R1, #0FFh
HERE:     CJNE       R1, #0000h, HERE
          RET
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * * * * * * * * * * * * * * Serial Interrupt Service Routine * * * *
```

```
;
;
SERINT:   CLR       ES0             ; Disable serial interrupts
          ;
          JNB       TI, SER_A       ; If TI=0, must be receive complete interrupt
          CLR       TI              ; Else must be transmit complete interrupt
          SJMP      SER_X
SER_A:
          CLR       RI              ; Clear the receive bit
          MOV       A, SBUF         ; Get the character and
          MOV       SBUF, A         ; echo it
SER_X:
          MOV       R1, #00h        ; Indicate interrupt has been serviced
                                    ; (i.e. exit wait)
          SETB      ES0             ; Re-enable serial interrupts
          RETI                      ; Return from interrupt
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;
          END ; Program ASYNC10B.ASM
```

CODE EXAMPLE #2 : DUAL SERIAL PORT

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * * * * * * * * * * * * * * * * Program Tst2Ser * * * * * * * * *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; This program demonstrates the simultaneous use of both of the DS80C320's
; serial ports. Both serial channels run off of timer 1 which is set for
; 9600 KBaud with SMOD = 0, and an 11.059 MHz crystal. The program
; assumes that the transmit and receive pins of port 1 are connected
; together (i.e., TXD1 tied to RXD1). When the program starts, the
; DS80C320 sends a three line message out on TXD0. It then waits for
; a character to come in on RXD0 from an external terminal. This
; received character is converted to upper case if it is not already.
; The upper case character is then transferred to the transmit pin of
; port 1, TXD1, and the processor then waits for a character to be
; received on port 1's receive pin, RXD1. The received character is
; then transferred to port 0's transmit pin, TXD0, which completes the
; loop with the terminal.
;
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;
;              Initialization
;
          RG          100h
START:
          MOV         IE, #0          Disable all interrupts
;
;                     Set up Timer 1
;
          MOV         TMOD, #020h     Timer 1: Mode 2 (8–bit reload)
          MOV         TH1, #REL96     Reload value for 9600, SMOD = 0
          MOV         TL1, #REL96     Make first time–out correct
;
          ANL         PCON, #07Fh     Clear SMOD bit for port 0
          ANL         WDCON, #07Fh    Clear SMOD bit for port 1
;
          CLR         ET1            ; Disable timer 1 interrupts
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;
;                     Set up serial ports
;
          MOV         SCON, #050h    ; Timer 1 : Mode 1 (vari baud)
                                     ; and set receive enable
          MOV         SCON1, #050h   ; Timer 2 : Mode 1 (vari baud)
                                     ; and set receive enable
          SETB        TR1            ; Start timer 1
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;
;                     Configuration Demonstration
;
;
;         Output 3 Message Lines
          MOV         DPTR, #MSG1    ; Point to first message
          CALL        PUTS           ; and call routine to output
          MOV         DPTR, #MSG2    ; Point to second message
          CALL        PUTS           ; and call routine to output
          MOV         DPTR, #MSG3    ; Point to third message
          CALL        PUTS           ; and call routine to output
          ;
GETCH:
          JNB         RI, GETCH      ; Wait here for received character
          CLR         RI             ; Get ready for next character
          MOV         A, SBUF        ; Put new character in Acc
          ;
          ; If lower case character, convert to upper case
          CJNE A, #'a', GETCH_A      ; If character
GETCH_A:  JC GETCH_CONT              ; is between 'a'
          CJNE A, #'z'+1, GETCH_B    ; and 'z' +1
GETCH_B: JNC GETCH_CONT              ; it is lower case
          CLR         C              ; so subtract 020h
          SUBB        A, #020h       ; from it
          ;
GETCH_CONT:
          CALL        PUTCH1         ; Output character to port 1
          ;
GETCH1:
          JNB         RI1, GETCH1    ; Wait here for received character
          CLR         RI1            ; Get ready for next character
          MOV         A, SBUF1       ; Put new character in Acc
GETCH1_CONT:
          CALL        PUTCH          ; Output character to port 0
          ;
```

```
          JMP        GETCH          ; Repeat cycle
;
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * * * * * * * * * * * * * * * * * * * * * Subroutines * * * * * * * *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;
;                    ; Output a character on port 0
PUTCH:
          MOV        SBUF, A        ; Copy Acc to SBUF 0
          JNB        TI, $          ; Wait here until TI bit set
          CLR        TI             ; Clear TI for next character
          RET                       ; Return
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;
;                    ; Output a string on port 0
PUTS:
          CLR        A              ; Make address offset zero
          MOVC       A, @A+DPTR     ; Get next char to output
          JZ         PUTS_A         ; If zero, end of string
          CALL       PUTCH          ; Output character pointed to
          INC        DPTR           ; Point to next character
          JMP        PUTS           ; Repeat procedure
PUTS_A:
          RET                       ; Exit from routine
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;
;                    ; Output a character on port 1
PUTCH1:
          MOV        SBUF1, A       ; Copy Acc to SBUF 1
          JNB        TI1, $         ; Wait here until TI bit set
          CLR        TI1            ; Get ready for next character
          RET                       ; Return
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;                    ; Output a string on port 1
PUTS1:
          CLR        A              ; Make address offset zero
          MOVC       A, @A+DPTR     ; Get next char to output
          JZ         PUTS1_A        ; If zero, end of string
          CALL       PUTCH1         ; Output character pointed to
          INC        DPTR           ; Point to next character
          JMP        PUTS1          ; Repeat procedure
PUTS1_A:
RET ; Exit from routine
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;
msg1:     db         'Dallas Semiconductor DS80C320', 0dh, 0ah, 0
msg2:     db         'Serial Port Tests.', 0dh, 0ah, 0
msg3:     db         'Characters typed in after this are converted to uppercase.'
          db         0dh, 0ah, 0
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;
          END        ; Program Tst2Ser
```

CODE EXAMPLE # 3 : 11–BIT ADDRESS RECOGNITION MODE

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; * * * * * * * * * * * * * * * * Program ADRECMD.ASM * * * * * * * * * * * *
;
;
; This program sets up the DS80C320’s Port 0 serial channel to operate
; in Multi–Processor Communications mode. In this mode, serial
; characters received that do not match a masked address are ignored
; (no interrupt generated). The address and mask are initially set to
; recognize a <cntl>C character (03h). When a <cntl>C is received, a message
; is sent out, and characters input are “echoed” out unchanged.
;
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; Interrupt Vectors
;
       ORG           000h                    ; RESET VECTOR
       AJMP          START                   ; Jump to start of program
;
       ORG           0023h                   ; SERIAL PORT 0 VECTOR
       AJMP          SERINT                  ; Jump to serial interrupt routine
;
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;         Initialization
;
          ORG 0100h
START:
          MOV         IE, #0                  ; Disable ALL interrupts
          MOV         P1, #0                  ; Clear ports P1
;
;         Clear SMOD for divide by 64
          ANL         PCON, #07Fh             ; Clear SMOD bit
;
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;
;         Set up Serial Port 0
;
          MOV         SCON, #0B8h             ; Mode 2, set receive enable,
                                              ; set multi–cpu communication, and set
                                              ; TB8 to be a 1 (consistent with input)
          SETB        ES0                     ; Enable Serial Port interrupt
          ;
          MOV         SADDR0, #03h            ; Set address register to <cntl>C (03h)
          MOV         SADEN0, #0FFh           ; Set mask to all 1s (no mask)
          ;
          SETB        EA                      ; Global Interrupt enable
;
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; * * * * * * * * * * * * * * * * Code to exercise the serial port * * * * * * *
;
;
LOOP:
          MOV         A, SADEN0               ; If <cntl>C has not been received
          CJNE        A, #00, LOOP            ; continue waiting here.
          CALL        WAIT                    ; <cntl>C will be echoed, wait till
                                              ; transmit complete.
          ; Output a message
          MOV         SBUF, #’D’              ; Put character ’D’ in buffer (send it)
          CALL        WAIT                    ; Wait until flag cleared from interrupt
                                              ; service routine.
          MOV         SBUF, #’S’              ; Repeat for ’S’
          CALL        WAIT
          MOV         SBUF, #’8’              ; Repeat for ’8’
          CALL        WAIT
          MOV         SBUF, #’0’              ; Repeat for ’0’
          CALL        WAIT
          MOV         SBUF, #’C’              ; Repeat for ’C’
          CALL        WAIT
          MOV         SBUF, #’3’              ; Repeat for ’3’
          CALL        WAIT
          MOV         SBUF, #’2’              ; Repeat for ’2’
          CALL        WAIT
          MOV         SBUF, #’0’              ; Repeat for ’0’
          CALL        WAIT
          MOV         SBUF, #00Dh             ; Repeat for <cr>
          CALL        WAIT
          MOV         SBUF, #00Ah             ; Repeat for <lf>
          CALL        WAIT
          ;
          SJMP        $                       ; Continuous loop. Only serial
                                              ; interrupts exit loop. Characters are
                                              ; echoed.
;
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; * * * * * * * * * * * * * Subroutine to wait for a serial interrupt * * * *
WAIT:     MOV         R1, #0FFh               ; Serial ISR clears R1 when complete
HERE:     CJNE        R1, #0000h, HERE
```

```
          RET
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; * * * * * * * * * * * * * * * * * * Serial Interrupt Service Routine * * * *
;
SERINT:   CLR        ES0                   ; Disable serial interrupts
          ;
          JNB        TI, SER_A             ; If TI=0, must be receive complete interrupt
          CLR        TI                    ; Else must be transmit complete interrupt
          SJMP       SER_X
          ;
SER_A:
          CLR        RI                    ; Clear the receive bit
          MOV        A, SBUF               ; Get the character
          MOV        SADEN0, #0            ; Clear address mask register
          MOV        P1, A                 ; Display character on port 1
          MOV        SBUF, A               ; and echo it
SER_X:
          MOV        R1, #00h              ; Indicate interrupt has been serviced
                                           ; (i.e. exit wait)
          SETB       ES0                   ; Re-enable serial interrupts
          RETI                             ; Return from interrupt
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;
          END ; Program ADRECMD
```

APPENDIX A : CONSTANT DEFINITIONS

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * * * * * * * * * * * * * * * * * * * DEFINITIONS * * * * * * * *
;          Define Register Addresses
PCON       EQU      087h                   ; Contains SMOD baud rate doubler bit
TCON       EQU      088h                   ; Control for timer 1 and 0
TMOD       EQU      089h                   ; Mode register for timers 1 and 0
TL1        EQU      08Bh                   ; Timer 1 low (count) byte
TH1        EQU      08Dh                   ; Timer 1 high (reload) byte
CKCON      EQU      08Eh                   ; Clock Control register
P1         EQU      090H                   ;
SCON       EQU      098h                   ; Control for serial port 0
SBUF       EQU      099h                   ; Data buffer for serial port 0
IE         EQU      0A8h                   ; Interrupt enables
SADDR0     EQU      0A9h                   ;
SADEN0     EQU      0B9H                   ;
SCON1      EQU      0C0h                   ; Control for serial port 1
SBUF1      EQU      0C1h                   ; Data buffer for serial port 1
T2CON      EQU      0C8h                   ; Timer 2 control register
RCAP2L     EQU      0CAh                   ; Timer 2 low byte of reload register
RCAP2H     EQU      0CBh                   ; Timer 2 high byte of reload register
TL2        EQU      0CCh                   ; Timer 2 low byte of count register
TH2        EQU      0CDh                   ; Timer 2 high byte of count register
WDCON      EQU      0D8h                   ; Watchdog Control register
;
;          Define bit addresses
TR1        EQU      08Eh                   ; Timer 1 enable bit
RI         EQU      098h                   ; Serial port 0 receive interrupt flag
TI         EQU      099h                   ; Serial port 0 transmit interrupt flag
REN        EQU      09Ch                   ; Serial port 0 receive enable bit
SM2        EQU      09Dh                   ; Serial port 0 mode bit 2
SM1        EQU      09Eh                   ; Serial port 0 mode bit 1
SM0FE      EQU      09Fh                   ; Serial port 0 mode bit 0
;
RI1        EQU      0C0h                   ; Serial port 1 receive interrupt flag
TI1        EQU      0C1h                   ; Serial port 1 transmit interrupt flag
REN1       EQU      0C4h                   ; Serial port 1 receive enable bit
SM21       EQU      0C5h                   ; Serial port 1 mode bit 2
SM11       EQU      0C6h                   ; Serial port 1 mode bit 1
SM0FE1     EQU      0C7h                   ; Serial port 1 mode bit 0
;
TR2        EQU      0CAh                   ; Timer 2 enable bit
;
ET1        EQU      0ABh                   ; Timer 1 interrupt enable
ES0        EQU      0ACh                   ; Serial port 0 interrupt enable
ES1        EQU      0AEh                   ; Serial port 1 interrupt enable
EA         EQU      0AFh                   ; Global interrupt enable
;
;          Define Constants
; Timer 1 Reload Values
REL576     EQU      0FFh                   ; Reload value for 57600 baud, SMOD = 1
REL96      EQU      0FDh                   ; Reload value for 9600 baud, SMOD = 0
REL24      EQU      0F4h                   ; Reload value for 2400 baud, SMOD = 0
REL12      EQU      0E7h                   ; Reload value for 2400 baud, SMOD = 0
; Timer 2 Reload Values
T2RL57H    EQU      0FFh                   ; 16-bit Reload value for 57600 baud
T2RL57L    EQU      0FAh
T2RL96H    EQU      0FFh                   ; 16-bit Reload value for 9600 baud
T2RL96L    EQU      0DCh
T2RL24H    EQU      0FFh                   ; 16-bit Reload value for 2400 baud
T2RL24L    EQU      070h
T2RL12H    EQU      0FEh                   ; 16-bit Reload value for 1200 baud
T2RL12L    EQU      0E0h
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

APPENDIX B: MAXIMUM BAUD RATES

It is frequently asked what the maximum possible baud rate is for a particular serial mode of the DS80C320. The following table shows the maximum possible baud rates for the four primary serial modes.

MODE	MAXIMUM BAUD, 25 MHz	MAXIMUM BAUD, 33 MHz	MAXIMUM BAUD, 40 MHz
0	6,250,000	8,250,000	10,000,000
1	390,625	515,625	625,000
2	781,250	1,031,250	1,250,000
3	390,625	515,625	625,000