A Simplified Guide to Using the MCS® 51 On-chip UART

Contents	Page
Overview	
Serial Port Modes	3
Baud Rate Generation Tables	
Timer 2	4
Timer 1	5
Why are some baud rates missing from the table?	6
Some common problems and questions when trying to set up the serial port in the MCS®51 Family.	
What is the purpose of using interrupts and/or polling in serial applications?	6
How does the serial interrupt and polling work?	7
When should I use polling or interrupts?	8
Common Problems	
I am viewing data on an oscilloscope and I am not seeing the data transmitted; I see other data instead. Why?	8
I am moving data into SBUF, all my registers are configured for serial communications, nothing is being transmitted. Why?	8
All of my registers are set up correctly, but when I receive data, the microcontroller never vectors to my interrupt routine. Why?	8
I am trying to transmit data and all I see on my oscilloscope is a square wave coming out of the Txd pin. Why?	8
I am receiving data and I move it to another register and read it. The value that I am reading is not the data that I received. Why?	8
Sample Programs	
M0.ASM	9
M1T1.ASM	10
M2.ASM	11

M3T2.ASM	12
M1INT.ASM	13

Overview

The MCS®-51 family contains a flexible set of microcontrollers. These 8-bit embedded controllers have different features such as on-chip program memory, data RAM and some even have integrated A/D converters. One feature that all of the microcontrollers in the MCS®-51 family have in common is an integrated UART (Universal Asynchronous Receiver Transmitter).

This guide has been designed so that any programmer with basic microcontroller experience can learn how to use the general features of the on-chip UART in a MCS®-51 microcontroller. This document has been created and designed in response to repeated inquires on the usage of the serial port. Working examples have been included and explained to ease the learning process.

The serial port can operate in 4 modes:

Mode 0: TXD outputs the shift clock. In this mode, 8 bits are transmitted ***and*** received by the same pin, RXD. The data is transmitted starting with the least significant bit first, and ending with the most significant bit. The baud rate is fixed at 1/12 the oscillator frequency.

Mode 1: Serial data enters through the RXD pin and exits through the TXD pin. In this mode, a start bit of logic level 0 is transmitted then 8 bits are transmitted with the least significant bits first up to the most significant bit; following the most significant bit is the stop bit which is a logic 1. When receiving data in this mode, the stop bit is placed into RB8 in the SFR (Special Function Register) SCON. The baud rate is variable and is controlled by either timer 1 or timer 2 reload values.

Mode 2: Serial data enters through the RXD pin and exits through the TXD pin. In this mode, a total of 11 bits are transmitted or received starting with a start bit of logic level 0, 8 bits of data with the least significant bit first, a user programmable ninth data bit, and a stop bit of logic level 1. The ninth data bit is the value of the TB8 bit inside the SCON register. This programmable bit is often used for parity information. The baud rate is programmable to either 1/32 or 1/64 of the oscillator frequency.

Mode 3: Mode three is identical to mode 2 except that the baud rate is variable and is controlled by either timer 1 or timer 2 reload values.

For more detailed information on each serial port mode, refer to the "Hardware Description of the 8051, 8052, and 80c51." in the 1993 Embedded Microcontrollers and Processors (270645).

Baud Rate Generation Using Timer Two

$$\text{Baud Rate} = \frac{F_{osc}}{(32(65536-(RCAP2H,RCAP2L)))}$$

$$(\text{RCAP2H,RCAP2L}) = 65536 - \frac{F_{osc}}{32*(BaudRate)}$$

RCAP2L and RCAP2H are 8-bit registers combined as a 16-bit entity that timer 2 uses as a reload value. Each time timer 2 overflows (goes one past FFFFH), this 16-bit reload value is placed back into the timer, and the timer begins to count up from there until it overflows again. Each time the timer overflows, it signals the processor to send a data bit out the serial port. The larger the reload value (RCAP2H, RCAP2L), the more frequently the data bits are transmitted out the serial port. This frequency of data bits transmitted or received is known as the baud rate.

Table One

Baud Rate	***Freq (Mhz)***	***RCAP2H***	***RCAP2L***	***Baud Rate***	***Freq (Mhz)***	***RCAP2H***	***RCAP2L***
38,400	*16*	*FF*	*F3*	*56,800*	*11.059*	*FF*	*FA*
19,200	*16*	*FF*	*E6*	*38,400*	*11.059*	*FF*	*F7*
9,600	*16*	*FF*	*CC*	*19,200*	*11.059*	*FF*	*EE*
4,800	*16*	*FF*	*98*	*9,600*	*11.059*	*FF*	*DC*
2,400	*16*	*FF*	*30*	*4,800*	*11.059*	*FF*	*B8*
1,200	*16*	*FE*	*5F*	*2,400*	*11.059*	*FF*	*70*
600	*16*	*FC*	*BF*	*1,200*	*11.059*	*FE*	*E0*
300	*16*	*F9*	*7D*	*600*	*11.059*	*FD*	*C0*
110	*16*	*EE*	*3F*	*300*	*11.059*	*FB*	*80*
375,000	*12*	*FF*	*FF*	*4,800*	*6*	*FF*	*D9*
9,600	*12*	*FF*	*D9*	*2,400*	*6*	*FF*	*B2*
4,800	*12*	*FF*	*B2*	*1,200*	*6*	*FF*	*64*
2,400	*12*	*FF*	*64*	*600*	*6*	*FE*	*C8*
1,200	*12*	*FE*	*C8*	*300*	*6*	*FD*	*8F*
600	*12*	*FD*	*8F*	*110*	*6*	*F9*	*57*
300	*12*	*FB*	*1E*				

Baud Rate Generation Using Timer One

$$\text{Baud Rate} = \frac{2^{SMOD} F_{OSC}}{(384(256-TH1))}$$

$$TH1 = 256 - \frac{2^{SMOD} F_{OSC}}{BaudRate * 384}$$

Similar to timer 2, TH1 is an 8-bit register that timer 1 uses as it's reload value. The larger the number placed in TH1, the faster the baud rate. SMOD1 is bit position 7 in the PCON register. This bit is called the "Double Baud Rate Bit". When the serial port is in mode 1, 2 or 3 and timer 1 is being used as the baud rate generator, the baud rate can be doubled by setting SMOD1. For example; TH1 equals DDH and the oscillator frequency equals 16Mhz, then the baud rate equals 2400 baud if SMOD1 is set. If SMOD1 is cleared, for the same example, then the baud rate would be 1200.

Table Two

Baud Rate	**Freq (Mhz)**	**SMOD1**	**TH1**	**Baud Rate**	**Freq (Mhz)**	**SMOD1**	**TH1**
4,800	16	1	EF	56,800	11.059	1	FF

2,400	16	1	DD	19,200	11.059	1	FD
1,200	16	1	BB	9,600	11.059	1	FA
600	16	1	75	4,800	11.059	1	F4
2,400	16	0	EF	2,400	11.059	1	E8
1,200	16	0	DD	1,200	11.059	1	D0
600	16	0	BB	600	11.059	1	A0
300	16	0	75	300	11.059	1	40
4,800	12	1	F3	9,600	11.059	0	FD
2,400	12	1	E6	4,800	11.059	0	FA
1,200	12	1	CC	2,400	11.059	0	F4
600	12	1	98	1,200	11.059	0	E8
300	12	1	30	600	11.059	0	D0
2,400	12	0	F3	300	11.059	0	A0
1,200	12	0	E6	1,200	6	0	F3
600	12	0	CC	600	6	0	E6
300	12	0	98	300	6	0	CC
				110	6	0	72

Why are some baud rates missing from the table?

If you look at the table carefully, you will notice that some common baud rates are missing in certain scenarios. The reason is, certain microcontroller operating frequencies will only support specific baud rates. Just because a baud rate reload value can be calculated by the previous equations, doesn't mean that the microcontroller can accurately generate that specific baud rate. If you would like to calculate a baud rate that is not in the previous tables, or if you want to find out if a specific baud rate can be accurately generated at a specific operating frequency, follow these steps:

1. Use the appropriate equation to calculate the reload value.
2. Round off the calculated reload value to the nearest whole number.
3. Recalculate the baud rate using the rounded off reload value.
4. Calculate the percent error between the two baud rates by using the following formula:

$$error = \frac{abs(desired - calculated)}{desired} \times 100$$

5. If the percent error is less that 2%, then the rounded reload value is adequate to generate the specified baud rate. If the error is greater than 2%, this means the baud rate generated by the microcontroller would be different from the baud rate that you expect to be

transmitting and there may be a loss of data in the process.

Some common problems and questions when trying to set up the serial port in the MCS®-51 family.

The intention of this section is to provide quick answers to common problems. This has been compiled by Intel employees who technically support the MCS®-51 family of microcontrollers**.**

1. *What is the purpose of using interrupts and/or polling in serial applications?*

In serial applications, it is necessary to know when data has completed transmission or has completed reception. Whenever data has completed transmission or completed reception, there is a specific bit (flag) that is set when the process has been completed. These two specific bits are located in the SCON register and determine when an interrupt will occur or when the polling sequence should be complete. The bits are RI and TI.

- RI is the receive interrupt flag. When operating in mode 0 of the UART, this bit is set by hardware when the 8th bit is received. In all other UART operating modes, the RI bit is set by hardware upon reception halfway through the stop bit. RI bit must be cleared by software at the end of the interrupt service routine or at the end of the polling sequence.

- TI is the transmit interrupt flag. This bit operates in the same manner as RI except it is valid for transmission of data, not reception. By using either interrupts or polling, it is necessary to check to see if either of the two bits are set.

- For the case of transmitting data, it is necessary to "watch" to see if the TI bit is set. A set bit has a logic level of 1 and a cleared bit has a logic level of 0. If you try to transmit more data and your previous data has not yet fully been transmitted, you will overwrite on top of it and have data corruption. Therefore, you must only transmit the next piece of data after the transmission of the current data has been completed.

- For the case of receiving data, it is necessary to watch and see if the RI bit is set. This bit serves a similar purpose as the TI bit. Upon reception of data, it is necessary to know when data has been completely received so it can be read before more data comes and overwrites the existing data in the register.

2. *How does the serial interrupt and polling work?*

A serial interrupt will occur whenever the RI or the TI bit has been set and the serial interrupts have been enabled in the IE and SCON register. When TI or RI is set, the processor will vector to location 23H. A common serial interrupt routine would be the following:

...
org 23h
JMP *label*
...
...
label: *subroutine code*
...
RETI

After the processor vectors to 23H, it will then vector off to location *label* which has a physical location defined by the assembler. *Label* is the start of your serial interrupt subroutine which should do the following:

- Find out which bit caused the interrupt RI or TI.

- Move data into or out of the SBUF register if necessary.
- Clear the corresponding bit that caused the interrupt.

The last line of your serial interrupt subroutine should be RETI. This makes the processor vector back to the next line of code to be executed before the processor was interrupted.

Polling is easier to implement than interrupt driven routines. The technique of polling is simply to continuously check a specified bit without doing anything else. When that bit changes state, the loop should end. For the case of serial transmission, a section of sample code would be the following:

```
...
JNB TI, $ ;this code will jump onto itself until TI is set
CLR TI ;clear the TI bit
...
```

For receive polling, just replace the TI in the previous code with RI. In either case, make sure that after polling has completed, clear the bit that you were polling.

3. *When should I choose polling or interrupts?*

Polling is the simplest to use but it has a drawback; high CPU overhead. This means that while the processor is polling, it is not doing anything else, this is a waste of the CPU's time and tends to make programs slow.

Interrupts are a little more complex to use but allows the processor to do other functions. Thus, serial communication functions are executed only when needed. This makes programs run faster than programs that use polling.

Common Problems

I am viewing data on an oscilloscope and I am not seeing the data I transmitted; I see other data instead. Why?

You are not waiting for the data to be completely transmitted before you send more data out. The new data is being written on top of the old data before it exits to the serial port. See "What is the purpose of using interrupts and/or polling in serial applications" on page 6.

I am moving data into SBUF, all my registers are configured for serial communications, and nothing is being transmitted. Why?

Chances are that the timer you chose for your baud rate generator was never started or "turned on."

All of the registers are set up correctly, but when I receive data, the microcontroller never vectors to the interrupt routine. Why?

The global interrupt enable bit has not been set or the serial interrupt bit has not been set.

The address of the first line of the serial interrupt routine was not at location 23H.

I am trying to transmit data and all I see on the oscilloscope is a square wave coming out of the Txd pin. Why?

The microcontroller serial port is in mode 0. In mode 0, the Txd pin outputs the shift clock (a square wave). Data is actually transmitted and received through the Rxd pin.

I am receiving data and I move it to another register and read it. The value that I am reading is not the data that I received. Why?

The data that was received was not moved out of the buffer (SBUF) fast enough before the new data arrived. Therefore, part of the old data got overwritten before you transferred it to another register. To avoid this, see "What is the purpose of using interrupts and/or polling in serial applications?" on page 6.

Sample Programs

The following programs have been designed to aid in the understanding of the general setup and transmission of serial applications.

```
;FILE: M0.ASM
;
;THIS PROGRAM TRANSMITS THE HEX VALUE AA REPETITIVELY ACROSS THE SERIAL PORT
;OF A MCS®-51 MICROCONTROLLER IN MODE 0
;
;DETAILS:
;
;MODE 0: SERIAL DATA EXITS AND ENTERS THROUGH THE RXD PIN.  THE
;TXD PIN OUTPUTS THE SHIFT CLOCK.  IN MODE 0, 8 BITS ARE TRANSMITTED/RECEIVED
;STARTING WITH THE LEAST SIGNIFICANT BIT.  THE BAUD RATE IS FIXED TO 1/12 THE
;OSCILLATOR FREQUENCY.
;
;
      ORG 00H
      JMP MAIN
MAIN: MOV SCON,#00H           ;SET UP FOR MODE 0
      CLR TI                  ;READY TO TRANSMIT
LOOP: MOV SBUF,#0AAH          ;TRANSMIT AAH
      JNB TI,$                ;WAIT FOR END OF TRANSMISSION
      CLR TI                  ;CLEAR TRANSMIT FLAG
      JMP LOOP                ;DO IT ALL AGAIN
      END
```

```
;FILE: M1T1.ASM
;
;THIS PROGRAM TRANSMITS THE HEX VALUE AA REPETITIVELY ACROSS THE SERIAL PORT
;OF A MCS®-51 IN MODE 1 USING TIMER 1 AT A RATE OF 1200 BAUD
;
;DETAILS:
;
;MODE 1: 10 BITS ARE TRANSMITTED THROUGH TXD OR RECEIVED THROUGH RXD WITH THE
;START BIT FIRST (0), 8 DATA BITS WITH THE LEAST SIGNIFICANT BIT FIRST, AND A
;STOP BIT (1).  ON RECEIVE, THE STOP BIT GOES INTO RB8 IN SPECIAL FUNCTION
;REGISTER SCON.  THE BAUD RATE IS VARIABLE.
;
;
      ORG 00H
      JMP MAIN
MAIN: MOV SCON,#40H           ;SET SERIAL PORT FOR MODE 1 OPERATION
      MOV TMOD,#20H           ;SET TIMER 1 TO AUTO RELOAD
      MOV TH1,#0DDH           ;LOAD RELOAD VALUE FOR 1200 BAUD AT 16MHZ
      MOV TCON,#40H           ;START TIMER 1
      CLR TI
LOOP: MOV SBUF,#0AAH          ;TRANSMIT AA HEX OUT THE TXD LINE
      JNB TI,$                ;WAIT UNTIL TRANSMISSION COMPLETED
      CLR TI                  ;READY TO TRANSMIT ANOTHER
      JMP LOOP                ;DO IT ALL OVER AGAIN
      END
```

```
;FILE: M2.ASM
;
;THIS PROGRAM TRANSMITS THE HEX VALUE AA REPETITEVELY ACROSS THE SERIAL PORT
;OF A MCS®-51 IN MODE 2 AT A RATE OF 1/32 THE OSCILLATOR FREQUENCY
;
;DETAILS:
;
;MODE 2:  11 BITS ARE TRANSMITTED THROUGH TXD OR RECEIVED THROUGH RXD.
;STARTING WITH A START BIT (0), 8 DATA BITS WITH THE LEAST SIGNIFICANT BIT
;FIRST, A PROGRAMMABLE 9th DATA BIT, AND A STOP BIT (1).  ON TRANSMIT, THE 9th
;DATA BIT, TB8 IN SCON, CAN BE ASSIGNED A VALUE OF 0 OR 1.  FOR EXAMPLE THE
;PARITY BIT, P FROM PSW, COULD BE MOVED INTO TB8.  ON RECEIVE, THE NINTH DATA
;BIT GOES INTO RB8 IN SCON WHILE THE STOP BIT IS IGNORED.  (THE VALIDITY OF
;THE STOP BIT CAN BE CHECKED WITH FRAMING ERROR DETECTION.  THE BAUD RATE IS
;PROGRAMMABLE TO EITHER 1/32 OR 1/64 THE OSCILLATOR FREQUENCY.  IF SMOD1 BIT
;IN THE PCON REGISTER IS 0, THEN THE BAUD RATE IS 1/64 THE OSCILLATOR
;FREQUENCY, IF SMOD1 IS 1, THE THE BAUD RATE IS 1/32 THE OSCILLATOR FREQUENCY.
;
;
;
        PCON EQU 87H

        ORG 00H
        JMP MAIN
MAIN:   MOV SCON,#80H                   ;SET UP FOR MODE 2
        MOV PCON,#80H                   ;BAUD RATE EQUALS 1/32 OSC. FREQ
        CLR TI                          ;READY TO TRANSMIT
LOOP:   MOV SBUF,#0AAH                  ;TRANSMIT AAH
        JNB TI,$                        ;WAIT FOR END OF TRANSMISSION
        CLR TI                          ;READY TO TRANSMIT
        JMP LOOP                        ;DO IT ALL AGAIN
        END
```

```
;FILE: M3T2.ASM
;
;THIS PROGRAM TRANSMITS THE HEX VALUE AA REPETITIVELY ACROSS THE SERIAL PORT
;OF A MCS®-51 IN MODE 3 USING TIMER 2 AS A BAUD RATE GENERATOR TO GENERATE A
;BAUD RATE OF 2400 BAUD AT 16MHZ WITH A PARITY BIT
;
;DETAILS:
;
;MODE 3:  11 BITS ARE TRANSMITTED THROUGH TXD OR RECEIVED THROUGH RXD
;TRANSMISSION STARTS WITH A START BIT (0), EIGHT DATA BITS WITH THE LEAST
;SIGNIFICANT BIT FIRST, A PROGRAMMABLE 9TH DATA BIT, AND A STOP BIT (1). MODE
;3 IS THE SAME AS MODE 2 EXCEPT THAT MODE 3 HAS A VARIABLE BAUD RATE
;
;
        RCAP2H EQU 0CBH
        RCAP2L EQU 0CAH
        T2CON  EQU 0C8H

        ORG 00H
        JMP MAIN
MAIN:   MOV SCON,#0C0H      ;SET UP FOR SERIAL MODE 3
        MOV RCAP2H,#0FFH    ;LOAD HIGH BYTE TO GENERATE 2400 BAUD AT 16MHZ
        MOV RCAP2L,#30H     ;LOAD LOW BYTE TO GENERATE 2400 BAUD AT 16MHZ
        MOV T2CON,#14       ;TIMER 2 BAUD RATE GENERATOR AND START TIMER
        MOV A,#0AAH         ;PUT THE VALUE TO BE TRANSMITTED IN THE ACC
        MOV C,P             ;PARITY INFORMATION TO CARRY FLAG
        MOV TB8,C           ;PARITY INFO FROM CARRY TO PROGRAMMABLE BIT *
;                            *NOTE: THE CONTENTS OF THE CARRY FLAG IN THE
;                                   PSW MAY BE ALTERED
        CLR TI              ;READY TO TRANSMIT
LOOP:   MOV SBUF,A          ;TRANSMIT AAH
        JNB TI,$            ;WAIT UNTIL DONE TRANSMITTING
        CLR TI              ;READY TO TRANSMIT
        JMP LOOP            ;DO IT ALL OVER AGAIN
        END
```

```
;
;THIS PROGRAM RECEIVES A VALUE ENTERING INTO THE SERIAL PORT PIN RXD AND PUTS
;THE DATA OUT TO PORT 1.
;
;DETAILS:
;
;THE PROGRAM IS DESIGNED TO BE IN A CONTINUOUS NEVER ENDING LOOP UNTIL A BYTE
;OF DATA HAS BEEN COMPLETELY RECEIVED.  THE LOOP IS EXITED BECAUSE OF THE
;OCCURANCE OF A SERIAL INTERRUPT.  AFTER THE INTERRUPT HAS BEEN SERVICED, THE
;PROGRAM GOES BACK INTO IT'S ENDLESS LOOP UNTIL ANOTHER INTERRUPT OCCURS
;
;
      PCON EQU 87H              ;DEFINE REGISTER LOCATION
;
      ORG 00H
      JMP MAIN

      ORG 023H                 ;STARTING ADDRESS OF SERIAL INTERRUPT
      JMP SERIAL_INT
;
MAIN: MOV SCON, #50H           ;SET UP SERIAL PORT FOR MODE 1 WITH RECEIVE
                               ;ENABLED
      MOV TMOD, #20H           ;SET UP TIMER 1 AS AUTO-RELOAD 8-BIT TIMER
      MOV TH1, #0DDH           ;BAUD RATE EQUALS 2400 BAUD AT 16Mhz
      MOV PCON, #80H           ;SET THE DOUBLE BAUD RATE BIT
      MOV IE, #90H             ;ENABLE THE SERIAL PORT & GLOBAL INTERRUPT BITS
      MOV TCON, #40H           ;START TIMER 1
      CLR RI                   ;ENSURE THAT THE RECEIVE INTERRUPT FLAG IS
                               ;CLEAR
LOOP:  JMP LOOP                ;ENDLESS LOOP (UNLESS INTERRUPT OCCURS)
;
SERIAL_INT:                    ;SERIAL INTERRUPT ROUTINE
      CLR RI                   ;CLEAR THE RI BIT (SINCE WE KNOW THAT WAS THE
                               ;BIT THAT CAUSED THE INTERRUPT)
      MOV P1, SBUF             ;MOVE THE RECEIVED DATA OUT TO PORT 1
      RETI                     ;EXIT THE SERIAL INTERRUPT ROUTINE
      END
```