# 8051 Tutorial: Serial Communication - 8052.com

- 

One of the 8051s many powerful features is its integrated *UART*, otherwise known as a serial port. The fact that the 8051 has an integrated serial port means that you may very easily read and write values to the serial port. If it were not for the integrated serial port, writing a byte to a serial line would be a rather tedious process requring turning on and off one of the I/O lines in rapid succession to properly "clock out" each individual bit, including start bits, stop bits, and parity bits.
However, we do not have to do this. Instead, we simply need to configure the serial ports operation mode and baud rate. Once configured, all we have to do is write to an SFR to write a value to the serial port or read the same SFR to read a value from the serial port. The 8051 will automatically let us know when it has finished sending the character we wrote and will also let us know whenever it has received a byte so that we can process it. We do not have to worry about transmission at the bit level--which saves us quite a bit of coding and processing time.

Setting the Serial Port Mode

The first thing we must do when using the 8051s integrated serial port is, obviously, configure it. This lets us tell the 8051 how many data bits we want, the baud rate we will be using, and how the baud rate will be determined.

First, lets present the "Serial Control" (SCON) SFR and define what each bit of the SFR represents:

| Bit | Name | Bit Addres | Explanation of Function |
|---|---|---|---|
| 7 | SM0 | 9Fh | Serial port mode bit 0 |
| 6 | SM1 | 9Eh | Serial port mode bit 1. |
| 5 | SM2 | 9Dh | Mutliprocessor Communications Enable (explained later) |
| 4 | REN | 9Ch | Receiver Enable. This bit must be set in order to receive characters. |
| 3 | TB8 | 9Bh | Transmit bit 8. The 9th bit to transmit in mode 2 and 3. |
| 2 | RB8 | 9Ah | Receive bit 8. The 9th bit received in mode 2 and 3. |
| 1 | TI | 99h | Transmit Flag. Set when a byte has been completely transmitted. |
| 0 | RI | 98h | Receive Flag. Set when a byte has been completely received. |

Additionally, it is necessary to define the function of SM0 and SM1 by an additional table:

| SM0 | SM1 | Serial Mode | Explanation | Baud Rate |
|---|---|---|---|---|
| 0 | 0 | 0 | 8-bit Shift Register | Oscillator / 12 |
| 0 | 1 | 1 | 8-bit UART | Set by Timer 1 (*) |
| 1 | 0 | 2 | 9-bit UART | Oscillator / 64 (*) |
| 1 | 1 | 3 | 9-bit UART | Set by Timer 1 (*) |

(*) Note: The baud rate indicated in this table is doubled if PCON.7 (SMOD) is set.

The SCON SFR allows us to configure the Serial Port. Thus, well go through each bit and review its function.

The first four bits (bits 4 through 7) are configuration bits.

Bits **SM0** and **SM1** let us set the *serial mode* to a value between 0 and 3, inclusive. The four modes are defined in the chart immediately above. As you can see, selecting the Serial Mode selects the mode of operation (8-bit/9-bit, UART or Shift Register) and also determines how the baud rate will be calculated. In modes 0 and 2 the baud rate is fixed based on the oscillators frequency. In modes 1 and 3 the baud rate is variable based on how often Timer 1 overflows. Well talk more about the various Serial Modes in a moment.

The next bit, **SM2**, is a flag for "Multiprocessor communication." Generally, whenever a byte has been received the 8051 will set the "RI" (Receive Interrupt) flag. This lets the program know that a byte has been received and that it needs to be processed. However, when SM2 is set the "RI" flag will only be triggered if the 9th bit received was a "1". That is to say, if SM2 is set and a byte is received whose 9th bit is clear, the RI flag will never be set. This can be useful in certain advanced serial applications. For now it is safe to say that you will almost always want to clear this bit so that the flag is set upon reception of *any* character.

The next bit, **REN**, is "Receiver Enable." This bit is very straightforward: If you want to receive data via the serial port, set this bit. You will almost always want to set this bit.

The last four bits (bits 0 through 3) are operational bits. They are used when actually sending and receiving data--they are not used to configure the serial port.

The **TB8** bit is used in modes 2 and 3. In modes 2 and 3, a total of nine data bits are transmitted. The first 8 data bits are the 8 bits of the main value, and the ninth bit is taken from TB8. If TB8 is set and a value is written to the serial port, the datas bits will be written to the serial line followed by a "set" ninth bit. If TB8 is clear the ninth bit will be "clear."

The **RB8** also operates in modes 2 and 3 and functions essentially the same way as TB8, but on the reception side. When a byte is received in modes 2 or 3, a total of nine bits are received. In this case, the first eight bits received are the data of the serial byte received and the value of the ninth bit received will be placed in RB8.

**TI** means "Transmit Interrupt." When a program writes a value to the serial port, a certain amount of time will pass before the individual bits of the byte are "clocked out" the serial port. If the program were to write another byte to the serial port before the first byte was completely output, the data being sent would be garbled. Thus, the 8051 lets the program know that it has "clocked out" the last byte by setting the TI bit. When the TI bit is set, the program may assume that the serial port is "free" and ready to send the next byte.

Finally, the **RI** bit means "Receive Interrupt." It funcions similarly to the "TI" bit, but it indicates that a byte has been received. That is to say, whenever the 8051 has received a complete byte it will trigger the RI bit to let the program know that it needs to read the value quickly, before another byte is read.

Setting the Serial Port Baud Rate

Once the Serial Port Mode has been configured, as explained above, the program must configure the serial ports baud rate. This only applies to Serial Port modes 1 and 3. The Baud Rate is determined based on the oscillators frequency when in mode 0 and 2. In mode 0, the baud rate is always the oscillator frequency divided by 12. This means if youre crystal is 11.059Mhz, mode 0 baud rate will always be 921,583 baud. In mode 2 the baud rate is always the oscillator frequency divided by 64, so a 11.059Mhz crystal speed will yield a baud rate of 172,797.

In modes 1 and 3, the baud rate is determined by how frequently timer 1 overflows. The more frequently timer 1 overflows, the higher the baud rate. There are many ways one can cause timer 1 to overflow at a rate that determines a baud rate, but the most common method is to put timer 1 in 8-bit auto-reload mode (timer mode 2) and set a reload value (TH1) that causes Timer 1 to overflow at a frequency appropriate to generate a baud rate.

To determine the value that must be placed in TH1 to generate a given baud rate, we may use the following equation (assuming PCON.7 is clear).

TH1 = 256 - ((Crystal / 384) / Baud)

If PCON.7 is set then the baud rate is effectively doubled, thus the equation becomes:

TH1 = 256 - ((Crystal / 192) / Baud)

For example, if we have an 11.059Mhz crystal and we want to configure the serial port to 19,200 baud we try plugging it in the first equation:

TH1 = 256 - ((Crystal / 384) / Baud)
TH1 = 256 - ((11059000 / 384) / 19200 )
TH1 = 256 - ((28,799) / 19200)

TH1 = 256 - 1.5 = 254.5

As you can see, to obtain 19,200 baud on a 11.059Mhz crystal wed have to set TH1 to 254.5. If we set it to 254 we will have achieved 14,400 baud and if we set it to 255 we will have achieved 28,800 baud. Thus were stuck...
But not quite... to achieve 19,200 baud we simply need to set PCON.7 (SMOD). When we do this we double the baud rate and utilize the second equation mentioned above. Thus we have:

TH1 = 256 - ((Crystal / 192) / Baud)
TH1 = 256 - ((11059000 / 192) / 19200)
TH1 = 256 - ((57699) / 19200)
TH1 = 256 - 3 = 253

Here we are able to calculate a nice, even TH1 value. Therefore, to obtain 19,200 baud with an 11.059MHz crystal we must:

1. Configure Serial Port mode 1 or 3.
2. Configure Timer 1 to timer mode 2 (8-bit auto-reload).
3. Set TH1 to 253 to reflect the correct frequency for 19,200 baud.
4. Set PCON.7 (SMOD) to double the baud rate.

Writing to the Serial Port
Once the Serial Port has been propertly configured as explained above, the serial port is ready to be used to send data and receive data. If you thought that configuring the serial port was simple, using the serial port will be a breeze.

To write a byte to the serial port one must simply write the value to the **SBUF** (99h) SFR. For example, if you wanted to send the letter "A" to the serial port, it could be accomplished as easily as:

**MOV SBUF,#A**

Upon execution of the above instruction the 8051 will begin transmitting the character via the serial port. Obviously transmission is not instantaneous--it takes a measureable amount of time to transmit. And since the 8051 does not have a serial output buffer we need to be sure that a character is completely transmitted before we try to transmit the next character.
The 8051 lets us know when it is done transmitting a character by setting the **TI** bit in SCON. When this bit is set we know that the last character has been transmitted and that we may send the next character, if any. Consider the following code segment:

**CLR TI** ;Be sure the bit is initially clear
**MOV SBUF,#A** ;Send the letter A to the serial port
**JNB TI,$** ;Pause until the TI bit is set.

The above three instructions will successfully transmit a character and wait for the TI bit to be set before continuing. The last instruction says "Jump if the TI bit is not set to $"--$, in most assemblers, means "the same address of the current instruction." Thus the 8051 will pause on the JNB instruction until the TI bit is set by the 8051 upon successful transmission of the character.

Reading the Serial Port

Reading data received by the serial port is equally easy. To read a byte from the serial port one just needs to read the value stored in the **SBUF** (99h) SFR after the 8051 has automatically set the **RI** flag in SCON.

For example, if your program wants to wait for a character to be received and subsequently read it into the Accumulator, the following code segment may be used:

**JNB RI,$** ;Wait for the 8051 to set the RI flag
**MOV A,SBUF** ;Read the character from the serial port

The first line of the above code segment waits for the 8051 to set the RI flag; again, the 8051 sets the RI flag automatically when it receives a character via the serial port. So as long as the bit is not set the program repeats the "JNB" instruction continuously.

Once the RI bit is set upon character reception the above condition automatically fails and program flow falls through to the "MOV" instruction which reads the value.