**EE 354** **September 29, 2015**

**Notes on the Serial Port**

For serial communications you should be familiar with the following terms:

- UART
- USART
- Baud rate
- Synchronous communication
- Asynchronous communication
- Half-Duplex
- Full-Duplex

The term UART stands for Universal Asynchronous Receiver/Transmitter and it does the logic functions to make serial communications easier. Effectively, a UART provides an 8-bit parallel interface suitable for a microcontroller bus, to a serial communications channel. The UART can take 8-bits of data from a microcontroller, convert it to a serial data stream, and send it out over a single line at a preprogrammed bit rate. It can simultaneously, in full duplex mode, receive a serial stream, at the same or different bit rate, and convert it to 8-bit parallel for input to a microcontroller bus. A block diagram for the Intel 8251 USART is shown in Figure 1 below.

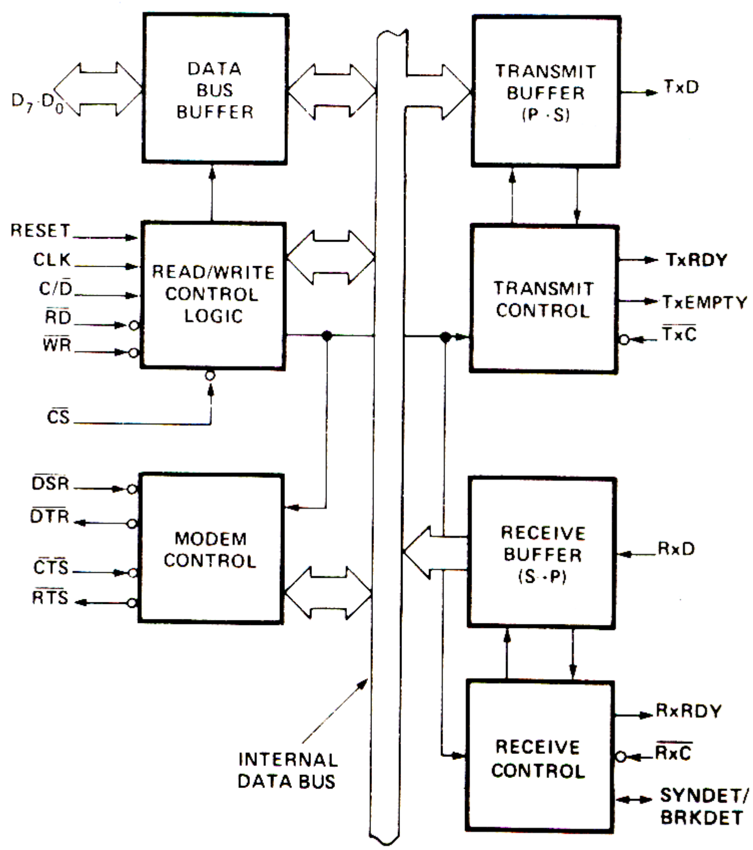


**Figure 1**

The Intel 8251 USART block diagram

A UART transmits and receives data serially in the format shown in Figure 2.

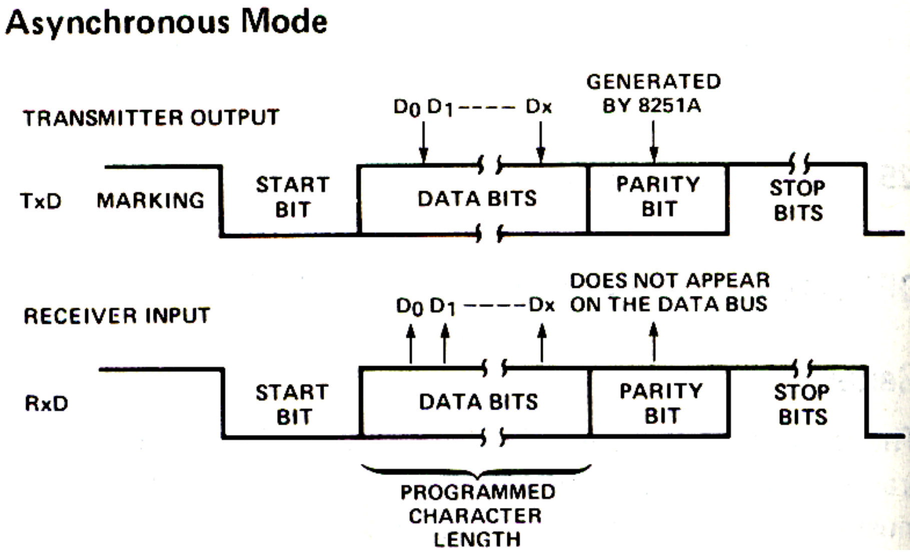


**Figure 2**

Format of serial data received and transmitted by a UART.

Note that:

1. The data is transmitted with the LSB first in time.
2. The serial line is normally high and a low going signal indicates a start bit for transmission.
3. A high going signal after 8 data bits indicates a stop bit.
4. Typical serial data consists of the 7-bit ASCII code plus a parity bit so that characters are sent in 10-bit packets consisting of a start bit, 7-bits of data, a parity bit, and a stop bit.

For serial transmission the baud rate is the same as the serial bit rate. The lowest standard baud rate is 110 baud which typically has two stop bits instead of one making an information packet have 11-bits instead of 10. All other standard baud rates have only 10-bits/packet. These are: 600, 1200, 2400, 4800, 7200, 9600, 14400, 19200, 28800, 38400, 57600, 115200, and 230400.

To transmit data we need an oscillator which will run at the bit rate we want to send and receive at. This is typically done by way of one of the on-board timers for a microcontroller. For example, at 1200 baud we would need a timer that runs at 1/1200 = 0.83333 milliseconds.

Since the clock is part of the signal for asynchronous transmission, it must be retrieved on the receiving end. So, for data sent at 1200 baud, the receiver waits for the start bit and half way through the start bit it starts its own 1200 bit/second oscillator. This oscillator clocks in the incoming bits until a stop bit is received. If the stop bit is not high, a "framing" error occurs.

The transmitter's clock and the receiver's clock are independent of one another but must be at approximately the same frequency. From the beginning of the start bit to the end of the stop bit there are 10 bit times. The two oscillators must be within half of a bit time of each other over this period or a framing error will occur. One-half of a bit in 10-bits is about 5%. In general, the clock rates of the transmitter and receiver are set to within ±2.5% of the agreed upon baud rate so that in the worst case the two clocks will be with 5% of one another.

The AT89C51CC03 has two sources for the baud rate clock: It can use timer 1 or timer 2. For information on how to set up the baud rate on timer 1 see the user's manual. The discussion below explains how to use timer 2 to generate the baud rate.

The AT89C51CC03 UART operates in one of four modes:

MODE 0: This is actually a synchronous mode where RxD is both receive line and a send line and the clock is placed on the TxD line. The clock in this case is always $F_{xtal}/12$ (or $F_{xtal}/6$ for the x2 mode).

MODE1: This is the standard UART mode. It can send and receive 10-bit packets over TxD and RxD respectively. The baud rate is set by the baud rate generator or by the timer 1 or timer 2 overflow rate.

MODE2 and MODE3: These two modes both send and receive 11 bits with 9-bits of data, one start bit, and one stop bit. The baud rate in mode 2 is fixed and in mode 3 it can be determined by the baud rate generator or by the timer 1 overflow rate.

In this class we will use MODE1 exclusively. To set up timer 2 as the baud rate generator for the serial port we need to set up timer 2 to overflow at the agreed upon baud rate. For this example we will transmit and receive at 19,200 baud which corresponds to 52.0833µsec. Using timer 2 in the 16-bit mode means we must load two capture registers with the proper overflow rate. See Figure 3.

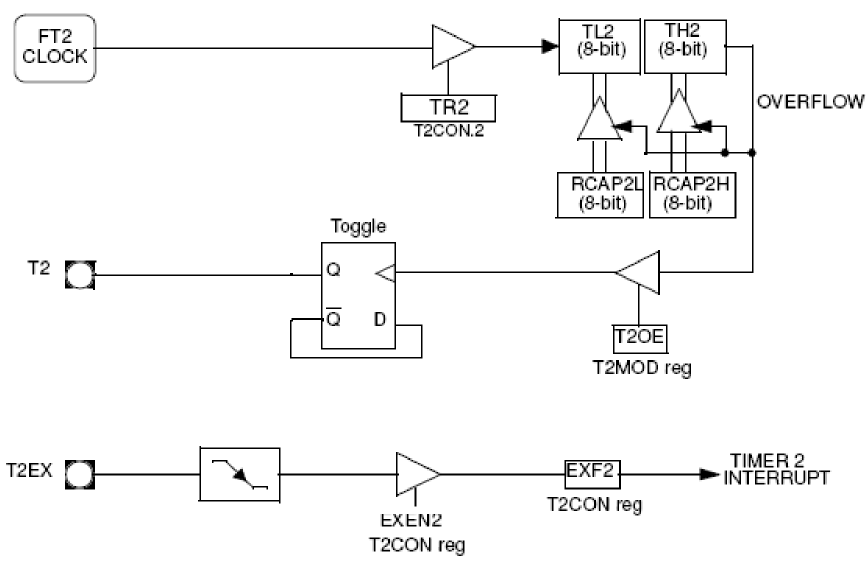


**Figure 3**

Timer 2 in the 16-bit mode. RCAP2L and RCAP2H determine the overflow rate.

In the normal mode the baud rate is given by

$$baud\ rate = f_{crystal}/(32*(65536 - (RCAP2H, RCAP2L)))$$

If system is double clocked we need to multiply the crystal rate by 2. For a double clocked system the baud rate is given by

$$baud\ rate = f_{crystal}/(16*(65536 - (RCAP2H, RCAP2L)))$$

For a baud rate of 19,200 a crystal of 28.2076MHz double clocked we get:

$$19200 = 28.2076\text{ MHz} /(16*(65536 - (RCAP2H, RCAP2L)))$$

This gives 65536 – RCAP2H, RCAP2L = 92 (as an integer. In hexadecimal, we set RCAP2H, RCAP2L = 0xFFA4.

**Table 25.** SCON Register

SCON (S:98h)
Serial Control Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| FE/SM0 | SM1 | SM2 | REN | TB8 | RB8 | TI | RI |

| Bit Number | Bit Mnemonic | Description |
|---|---|---|
| 7 | FE | **Framing Error bit (SMOD0=1)**<br>Clear to reset the error state, not cleared by a valid stop bit.<br>Set by hardware when an invalid stop bit is detected. |
| | SM0 | **Serial port Mode bit 0 (SMOD0=0)**<br>Refer to SM1 for serial port mode selection. |
| 6 | SM1 | **Serial port Mode bit 1**<br><u>SM0</u> <u>SM1</u> <u>Mode</u> <u>Baud Rate</u><br>0 0 Shift Register $F_{XTAL}/12$ (or $F_{XTAL}/6$ in mode X2)<br>0 1 8-bit UART Variable<br>1 0 9-bit UART $F_{XTAL}/64$ or $F_{XTAL}/32$<br>1 1 9-bit UART Variable |
| 5 | SM2 | **Serial port Mode 2 bit/Multiprocessor Communication Enable bit**<br>Clear to disable multiprocessor communication feature.<br>Set to enable multiprocessor communication feature in mode 2 and 3. |
| 4 | REN | **Reception Enable bit**<br>Clear to disable serial reception.<br>Set to enable serial reception. |
| 3 | TB8 | **Transmitter Bit 8/Ninth bit to transmit in modes 2 and 3**<br>Clear to transmit a logic 0 in the 9th bit.<br>Set to transmit a logic 1 in the 9th bit. |
| 2 | RB8 | **Receiver Bit 8/Ninth bit received in modes 2 and 3**<br>Cleared by hardware if 9th bit received is a logic 0.<br>Set by hardware if 9th bit received is a logic 1. |
| 1 | TI | **Transmit Interrupt flag**<br>Clear to acknowledge interrupt.<br>Set by hardware at the end of the 8th bit time in mode 0 or at the beginning of the stop bit in the other modes. |
| 0 | RI | **Receive Interrupt flag**<br>Clear to acknowledge interrupt.<br>Set by hardware at the end of the 8th bit time in mode 0, see Figure 33. and Figure 34. in the other modes. |

Reset Value = 0000 0000b
Bit addressable

**Figure 4**

The serial control register SCON for the AT89C51CC03.

**Table 36.** T2CON Register

T2CON (S:C8h)
Timer 2 Control Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| TF2 | EXF2 | RCLK | TCLK | EXEN2 | TR2 | C/T2# | CP/RL2# |

| Bit Number | Bit Mnemonic | Description |
|---|---|---|
| 7 | TF2 | **Timer 2 Overflow Flag**<br>TF2 is not set if RCLK=1 or TCLK = 1.<br>Must be cleared by software.<br>Set by hardware on timer 2 overflow. |
| 6 | EXF2 | **Timer 2 External Flag**<br>Set when a capture or a reload is caused by a negative transition on T2EX pin if EXEN2=1.<br>Set to cause the CPU to vector to timer 2 interrupt routine when timer 2 interrupt is enabled.<br>Must be cleared by software. |
| 5 | RCLK | **Receive Clock bit**<br>Clear to use timer 1 overflow as receive clock for serial port in mode 1 or 3.<br>Set to use timer 2 overflow as receive clock for serial port in mode 1 or 3. |
| 4 | TCLK | **Transmit Clock bit**<br>Clear to use timer 1 overflow as transmit clock for serial port in mode 1 or 3.<br>Set to use timer 2 overflow as transmit clock for serial port in mode 1 or 3. |
| 3 | EXEN2 | **Timer 2 External Enable bit**<br>Clear to ignore events on T2EX pin for timer 2 operation.<br>Set to cause a capture or reload when a negative transition on T2EX pin is detected, if timer 2 is not used to clock the serial port. |
| 2 | TR2 | **Timer 2 Run Control bit**<br>Clear to turn off timer 2.<br>Set to turn on timer 2. |
| 1 | C/T2# | **Timer/Counter 2 Select bit**<br>Clear for timer operation (input from internal clock system: $F_{OSC}$).<br>Set for counter operation (input from T2 input pin). |
| 0 | CP/RL2# | **Timer 2 Capture/Reload bit**<br>If RCLK=1 or TCLK=1, CP/RL2# is ignored and timer is forced to auto-reload on timer 2 overflow.<br>Clear to auto-reload on timer 2 overflows or negative transitions on T2EX pin if EXEN2=1.<br>Set to capture on negative transitions on T2EX pin if EXEN2=1. |

Reset Value = 0000 0000b
Bit addressable

**Figure 5**

This is the control register for Timer 2. Setting bits 4 and 5 allow timer 2 to be used as a baud rate generator for the UART.

**Table 28.** SBUF Register

SBUF (S:99h)
Serial Data Buffer

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| Bit Number | Bit Mnemonic | Description |
|---|---|---|
| 7-0 | | **Data sent/received by Serial I/O Port** |

Reset Value = 0000 0000b
Not bit addressable

**Figure 6**

SBUF is the data register for the serial port.

The power control register (PCON) has two bits which are used for serial communications. The power control register is shown in Figure 7. The PCON register's reset values need not be changed unless we need to double the baud rate when for example, the processor changes x2 modes.

**Table 29.** PCON Register

PCON (S:87h)
Power Control Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| SMOD1 | SMOD0 | – | POF | GF1 | GF0 | PD | IDL |

| Bit Number | Bit Mnemonic | Description |
|---|---|---|
| 7 | SMOD1 | **Serial port Mode bit 1**<br>Set to select double baud rate in mode 1, 2 or 3. |
| 6 | SMOD0 | **Serial port Mode bit 0**<br>Clear to select SM0 bit in SCON register.<br>Set to select FE bit in SCON register. |
| 5 | - | **Reserved**<br>The value read from this bit is indeterminate. Do not set this bit. |
| 4 | POF | **Power-Off Flag**<br>Clear to recognize next reset type.<br>Set by hardware when VCC rises from 0 to its nominal voltage. Can also be set by software. |
| 3 | GF1 | **General-purpose Flag**<br>Cleared by user for general-purpose usage.<br>Set by user for general-purpose usage. |
| 2 | GF0 | **General-purpose Flag**<br>Cleared by user for general-purpose usage.<br>Set by user for general-purpose usage. |
| 1 | PD | **Power-Down mode bit**<br>Cleared by hardware when reset occurs.<br>Set to enter power-down mode. |
| 0 | IDL | **Idle mode bit**<br>Clear by hardware when interrupt or reset occurs.<br>Set to enter idle mode. |

Reset Value = 00X1 0000b
Not bit addressable

**Figure 7**

PCON is the Power Control register. It has two bits which set the serial port's mode.

The AT89C51CC03 UART also has a serial address register so that the serial port can be assigned an address. When this address appears on the serial line, the UART processes the data and can interrupt the processor. For this class we will only use a single CPU and UART so that the serial address register will not be needed.

To initialize the timer and set up the UART in the AT89C51CC03 to transmit messages at 19,200 baud we need to complete the following steps:

1. Set up the system in x2 mode with F2Clock the same as the crystal.
```
CKCON = 0x01;
```
2. Set up the serial control register for mode 1 with an 8-bit UART
```
SCON = 0x40;
```
3. Turn on the receive and transmit clocks in T2CON.
```
RCLK = 1;
TCLK = 1;
```
4. Set up the capture registers in Timer 2
```
RCAP2H = 0xFF;
RCAP2L = 0xA4;
```
5. Turn on the Timer 2 run flag.
```
TR2 = 1;
```
6. Turn on the receive enable bit in SCON.
```
REN = 1;
```
7. Clear the receive interrupt flag.  This gets set when a bit is received.
```
RI = 0;
```
8. Turn on the global interrupt flag and the serial interrupt mask bit.
```
EA = 1;
ES = 1;
```

With this set up you can receive and transmit at 19,200 baud.  A complete program example is attached.

```
//SerialIO.c
// This project illustrates serial IO on the 89C51CC03 processor.
//   It is meant to be run on the simulator.
//Assumes a 12MHz crystal.
//To run this program on the simulator open P1 and use the view menu to open
//  the serial port (Use the Window menu to tile the windows).
//  Type data into the serial port and it will be echoed, entered
//  into the buffer, and displayed on P1.
#include<AT89C51CC03.h>
//
void SerialInt();
unsigned char buffer[33];
unsigned char bIndex = 0;
//
void main (void)
  {int i, pos;
   TMOD = 0x20;              //Timer 1 mode 2
   TH1 = 0xE6;               //1200 baud at 12 MHz with some error
   TCON = 0x40;              //Start the baud clock
   SCON = 0x50;              //Enable receiver
   IEN0 = 0x90;                //Enable the serial interrupt
   for(i=0;i<32;i++)
     buffer[i] = 0;
   TI = 0;                   //Transmit interrupt off -> receive only
   while(1)
     {for(pos=0;pos<32;pos++)
        if(buffer[pos] != 0)  //Send nonzero positions to port 1
            {P1 = buffer[pos];
          }
     }
  }
//
//
void SerialInt() interrupt 4 using 1
  {unsigned char c;
   c = SBUF & 0x7F;              //strip off parity flag
   buffer[bIndex] = c;
   bIndex = (bIndex + 1) % 32;
   SBUF = c;                     //Echo
   while(TI == 0);               //Wait here until transmit complete
   RI = 0;                       //Turn off receive interrupt flag
   TI = 0;
  }
```

```
//SerialGPSIO.c
/* Gets 5 inputs from the serial receive port at 9600 baud and stores
     these in buffer.  It then transmits these five inputs on the
       serial out port at 1200 baud.  The process repeats endlessly.

   To run in the simulator open Serial Port 1 from the View menu.  Run
     and enter 5 characters.
*/
#include <at89c51cc03.h>

unsigned char buffer[33];
unsigned char bIndex = 0;
void Get5Inputs();
void Send5Outputs();

void main (void)
  {CKCON = 0x01;      // x2 mode, T2 clock is same as crystal
   while(1)
     {Get5Inputs();
          Send5Outputs();
         }
  }
//
void SerialInt() interrupt 4 using 1
  {unsigned char c;
   c = SBUF & 0x7F;               //strip off parity flag
   buffer[bIndex] = c;
   bIndex = (bIndex + 1) % 32;
   RI = 0;                        //Turn off receive interrupt flag
   TI = 0;
  }
//
  void Get5Inputs()
  {int i;
   //9600 = 28.2076 x 10^6/(16(65536 - RCAP2H,RCAP2L)
   //  RCAP2H,RCAP2L = 65536 - 184 = 65352    = FF48h
   RCAP2H = 0xFF;            //9600 baud at 28.2076 MHz with some error
   RCAP2L = 0x48;
   T2CON = 0x20;              //Start the baud clock
   RCLK = 1;
   SCON = 0x50;              //Enable receiver
   IEN0 = 0x90;              //Enable the serial interrupt
   for(i=0;i<32;i++)         //Erase buffer
     buffer[i] = 0;
   TI = 0;                   //Transmit interrupt off -> receive only
   TR2 = 1;
   bIndex = 0;
   while(bIndex < 5);        //Wait for 5 inputs
   IEN0 = 0;
  }
 //
```

```
void Send5Outputs()
  {unsigned char i;
   SCON = 0X40;       // Mode 2, 8 bit uart transmit only, uses T2
   RCLK=1;            // Turn on receive clock in T2CON
   TCLK=1;            // Turn on transmit clock in T2CON
   //Baud rate = fCrystal/(32*(65536 - (RCAP2H, RCAP2L)) for x1 mode
   //Baud rate = fCrystal/(16*(65536 - (RCAP2H, RCAP2L)) for x2 mode
   //1200 = 28.2076 x 10^6/(16(65,536 = (RCAP2H, RCAP2L))
   //  RCAP2H, RCAP2L = 64067 = FA43h
   RCAP2H=0XFA;       //1200 baud @ 28.2076 Mhz
   RCAP2L=0x43;       //
   TR2=1;             // TCON bit to start Timer 2
   REN=0;             // Transmit only
   RI = 0;            // Clear the receive interrupt flag
   i = 0;
   while(i < 5) //Char string ends in 0
     {TI = 0;
      SBUF = buffer[i];
      i++;
      while (TI == 0);   // Wait for write to be done
     }
  }
```