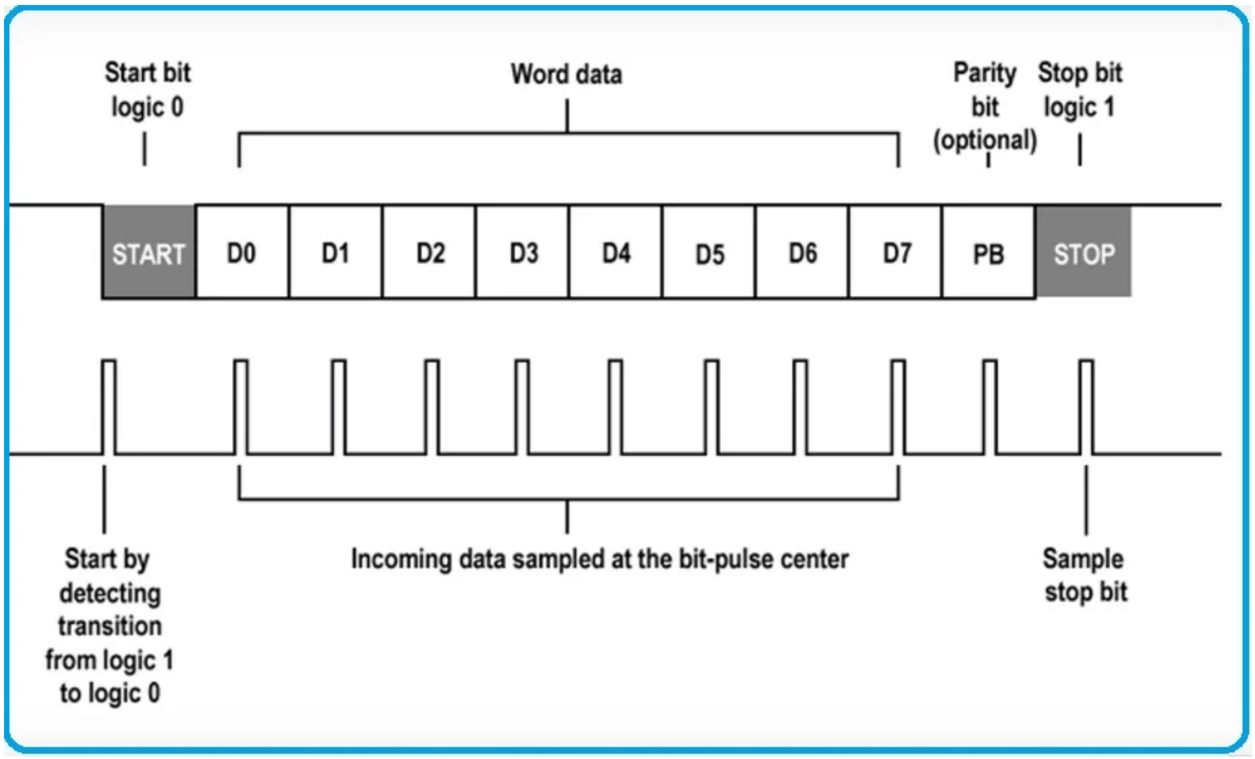
ENGR 323L: Embedded System Design

Engineering Department, Trinity College
Instructor: Professor T. Ning

Why use a UART?

- Universal Asynchronous Receiver/Transmitter
- UARTs are everywhere!
- Simple way to send data from one system to another system (using a predetermined baud rate)
- Add additional functionality to an application
- Traditional Definition: Converts parallel (8-bit) data to serial data and vice versa
- TX character frame (10-bit frame): Start, 8 data bits (D0, D1 D2 D3 D4 D5 D6 D7) and stop bit
  11-bit frame: Start bit, data bits (D0, D1 D2 D3 D4 D5 D6 D7), 9th bit (TB8), and stop bit
- RX Incoming data is received (D0, D1 D2 ,D3 D4 D5 D6 D7)



8051 Serial Communication (UART)

The 8051 CPUs are equipped with an on-chip serial interface *universal asynchronous receiver and transmitter* (UART) to allow serial data transmission and reception. The serial port of 8051 microcontrollers is full duplex, i.e., it can both transmit and receive data. It is also receiver-buffered, meaning it can commence reception of a second byte before a previously received byte has been read from the register. UART uses the *Serial Data Buffer Register* (SBUF) to hold data. The single SBUF location provides access to two physical registers – transmit register and receive register. Reading the SBUF accesses the receive register and writing to SBUF accesses the transmit register. The data communication is controlled by the *Serial Port Control Register* (SCON) and the Power Mode Control (PCON) SFR. Once configured with the desired operation mode and baud rate, one has only to write to SBUF to send or read from it to receive from the serial port.

Setting the Serial Port Mode: (SCON) SFR (098h):

Bit	**Name**	**Bit Address**	**Explanation of Function**
7	SM0	9Fh	Serial port mode bit 0
6	SM1	9Eh	Serial port mode bit 1.
5	SM2	9Dh	Multiprocessor Communications Enable
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.

SM0 and **SM1** determine the serial port mode. **Serial Port Mode:**

SM0/SM1	**Operation**
Mode - 0	Serial data enter and exit through RxD. TxD outputs the shift clock. 8 bits are transmitted/received (LSB first). The baud rate is fixed at 1/12 the oscillator frequency.
Mode -1	10 bits are transmitted (through TxD) or received (through RxD): a start bit, 8 data bits (LSB first), and a stop bit. On receive; the stop bit goes into RB8 in SCON. The baud rate is variable.
Mode -2	11 bits are transmitted (through TxD) or received (through RxD): start bit, 8 data bits (LSB first), 9th data bit, and a stop bit. On Transmit, the 9th data bit (TB8 in SCON) is assigned the value of 0 or 1 (or used for the parity bit in the PSW. On receive; the 9th data bit goes into RB8 in SCON. The baud rate is either 1/32 or 1/64 the oscillator frequency.
Mode -3	11 bits are transmitted (through TxD) or received (through RxD): a start bit, 8 data bits (LSB first), 9th data bit, and a stop bit. Mode 3 is the same as Mode 2 except the baud rate is variable. It is determined by the *PCON* register and *Timer_1 Overflow Interrupt*.

Setting Serial Port Baud Rate:

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 / 32 (*)
1	1	3	9-bit UART	Set by Timer 1 (*)

(*) Note: The baud rate indicated can be doubled if bit PCON.7 (i.e., SMOD) is set.

Once the Serial Port mode has been configured, the user must determine the serial baud rate for modes 1 and 3. The baud rate is automatically determined by the oscillator's frequency in modes 0 and 2, e.g., if a crystal frequency is 11.059 MHz, the baud rate in mode 0 will always be 921,583 baud.

In modes 1 and 3, the baud rate is determined by Timer_1 overflow interrupt. A common approach is to put Timer_1 in 8-bit auto-reload mode and initialize TH1 with a proper reload value that causes Timer_1 to overflow at a frequency to generate the desired baud rate by the following equation:

$$Baud\_rate = \frac{2^{SMOD} * crystal\_frequency}{32*12*[256-(TH1)]}$$

i.e.,

TH1 = 256 - ((crystal frequency / 384) / Baud)   *if PCON.7 is cleared or*

TH1 = 256 - ((crystal frequency / 192) / Baud)   *if PCON.7 is set.*

For example, to generate 14400 baud using an 11.059 MHz crystal,

TH1 = 256 - ((11059000 / 384) / 14400)

TH1 = 256 - 2 = 254.

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". This can be useful in certain advanced serial applications. One must clear this bit so that the flag is set upon reception of *any* character.

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.

TB8 bit is used in modes 2 and 3. In modes 2 and 3, 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 (SCON). Usually TB8 is set to indicate an address for target receiver; cleared to send data.

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 byte is sent to the serial port, a certain amount of time will pass before the individual bits of the byte are "clocked out" the serial port. The 8051 sets TI bit to indicate that it has "clocked out" the byte and the serial port is "free" to send another.

RI means "Receive Interrupt." It works similarly to "TI". It indicates that a byte has been received. The set RI alerts that the received byte in SBUF must be read before another byte received.

In all four modes, transmission is initiated by any instruction that uses SBUF as a destination register. Reception is initiated in Mode 0 by the condition RI = 0 and REN = 1. Reception is initiated in the other modes by the incoming start bit it REN = 1.

Multiprocessor Communications via UART (SM2=1):

In order to allow multiprocessor communication, the serial port must operate in mode-2 or mode-3, which has a special provision for multiprocessor communications. In these modes, 9 data bits are received. **The 9th one goes into RB8**. Then comes a stop bit. The port can be programmed such that when the stop bit is received, **the serial interrupt will be activated only if RB8 = 1.** This feature is enabled by setting bit SM2 in SCON. SM2 is a flag for "Multiprocessor communication." 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. A way to use this feature in multiprocessor systems is as follows:

> *When the master wants to transmit a block of data to one of several slaves, it first sends out an address byte which identifies the target slave. An address byte differs from a data byte in that the 9th bit is 1 in an address byte and 0 in a data byte. With SM2 = 1, no slave will be interrupted by a data byte. An address byte, however, will interrupt all slaves, so that each slave can examine the received byte and see if it is being addressed. The addressed slave will clear its SM2 bit and prepare to receive the data bytes that will be coming. The slaves that weren't being addressed leave their SM2s set and go on about their business, ignoring the coming data bytes.*

Example, transmitter (Master):

- The 9th transmit bit should be set in SCON (TB8 = 1) in order to distinguish the address byte from a data byte
- Start transmission using the SBUF register

Receiver (Slave):

- Enable Multiprocessor Communications by setting the SM2 bit in the SCON register so that the serial interrupt is triggered only when the received 9th bit (stored in RB8 in SCON) is set
- Determine if itself is among the receiver list and start receiving data if so
- During data reception the SM2 bit must cleared

Writing to the Serial Port: Once the Serial Port has been properly configured, user can transmit or receive data through the **SBUF** (99h) SFR. Transmission is initiated by any instruction that uses SBUF as a destination register. The “write to SBUF” signal also loads a 1 into the 9th position of the transmit shift register and tells the TX Control block to commence a transmission. As data bits shift out to the right, zeros come in from the left. When the MSB of the data byte is at the output position of the shift register, then the 1 that was initially loaded into the 9th position is added to the left of the MSB to indicate “stop bit”.

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. The 8051 will set the **TI** bit in SCON when the last character has been transmitted. Consider the following code segment:

```
CLR  TI         ;clear transmitter bit
MOV SBUF, #A    ;send character A to the serial port
JNB TI, $       ;Pause until the TI bit is set.
```

These three instructions will transmit character “A” and wait for the TI bit to be set.

Reading from the Serial Port: Reception is initiated by the condition **REN = 1** and **RI = 0**. **RI** flag in SCON is set if a byte (with the 9th bit if possible) is received . For example,

```
JNB RI, $       ;wait until RI flag set
MOV A, SBUF     ;read from the serial port
```

Note that RI flag needs to be manually cleared by the user in some 8051 derivatives such as the Silicon lab C8051F120.

General Procedure for Multiprocessor Communication:

- Configure Serial Port Control Register (SCON) to operate in multiprocessor variable mode (Mode 3)
- Serial Interrupt (ES0), Timer_0 Overflow Interrupt (TF0) and External Interrupt_0 (IE0) should be activated
- Configure the target baud rate by setting Timer_1 into 8-bit auto-reload mode. Based on the target crystal frequency, load the most accurate value into TH1 for auto-reload.
- The SMOD bit (bit 7 in PCON) can be set to double the existing baud rate.
- The special function registers IE, IP, TCON, TMOD, SCON should be properly configured upon initialization

As a Transmitter (Master):

- Invoke another interrupt service routine to specify the address of the designated receiver(s) within the network. Note that the 9th transmit bit should be set in SCON (TB8)

As a Receiver (Slave):

- Enable Multiprocessor Communications by setting SM2 =1 (in SCON) so that the serial interrupt is invoked only if received 9th bit (RB8 in SCON) is set. This is to select target receiver(s) to receive the following data.
- If identified as a target receiver, the slave begins receiving data (by setting SM2=0) and moving received byte from SBUF to allocated RAM.
- Non-target receivers will maintain SM2=1.

UART Serial Communication Example

```
#include <C8051F120.h>
#include <stdio.h>

// define variables
        enter variable declarations

//define function patterns
void putc(unsigned int c);
unsigned int getc(void);

/*
   External interrupt 0 is used to receive input from the "Record/Send" button
   Press the first time: record data
   afterwards, send data thru UART
*/
void SetupEx0(){
// high to low transition
   IT0 = 1;
   EX0 = 1;
}//SetupEX0

// ISR for external interrupt 0
void interrupt(0x03) ISR_EX0(){
   … enter statements to be executed
} //

// TDM for the four 7-segment displays
void SetupT1(void){
#prag
; --- initialize timer 1 ---
  setb TR1       ; timer 1 run control
  orl tmod, #10h ; mode 1: 16-bit counter
  mov TL1, #0C0h ; E0C0h + 8000d - 1= FFFFh
  mov TH1, #0E0h
  setb ET1       ; enable timer 1
#endasm
}

/*
   Serial communication -
   Baud rate: 9600 bps
*/
void SetupSerial(){

   // mode 3: multiprocessor communication
   SM0 = 1;
   SM1 = 1;
   SM2 = 0;
   RI = 0;
   TI = 0;
   REN = 1;   // enable receive
   ES = 1;    // enable serial interrupt
}// SetupSerial()
/*
   ISR to handle serial interrupt
   if it is receiving data
   - reset SM2 if get correct address,
   -- save data points if getting data point (SM = 0)
*/
void interrupt (0x23) ISR_Serial(){
   unsigned int temp;
   if (RI){
         temp = getc();

         // if SM2 = 0: expecting data from serial port
         if (SM2 == 0){
            …
            }

         // SM2 = 1, expecting address.

}//

// send out a character through serial port
void putc(char c){
   ES = 0;
   SBUF = c;
   while(!TI);
   TI = 0;
   ES = 1;
}

// receive a character through serial port
unsigned int getc(void){
   while (!RI);
   RI = 0;
   return (SBUF);
}//

void main(){
unsign char i;
   // initialize variables
   count = 0;
   T0_count = 0;
   display_count = 0;
   receive_count = 0;
   for (i = 0; i<256; i++){
      s_data[i] = i;
   }
```

```
    // set up timer, external interrupt and serial communication

    SetupT0();
    SetupSerial();
    SM2 = 1;
    TB8 = 1;

}// main
```