第 2-5 讲：串口通信

1. 学习目的

1. 掌握 USB 转串口电路的原理和设计。
2. 学习 IAP15F2K61S2/IAP15W4K61S4 的串口通信，包括串口初始化、波特率计算、串口发送和接收。
3. 编写串口收发程序，尤其是串口接收的软件缓存处理。
4. 编写串口发送命令控制 LED 指示灯亮灭的程序。

2. 硬件电路设计

IAP15F2K61S2 共有 2 个串口，开发板上将串口 1 用于了串口通信和程序下载，剩余的 1 个串口通过排针引出。

IAP15W4K61S4 共有 4 个串口，开发板上将串口 1 用于了串口通信和程序下载，剩余的 3 个串口均通过排针引出。

开发板的串口电路如下图所示， USB 转串口电路（USB 转 UART TTL）。

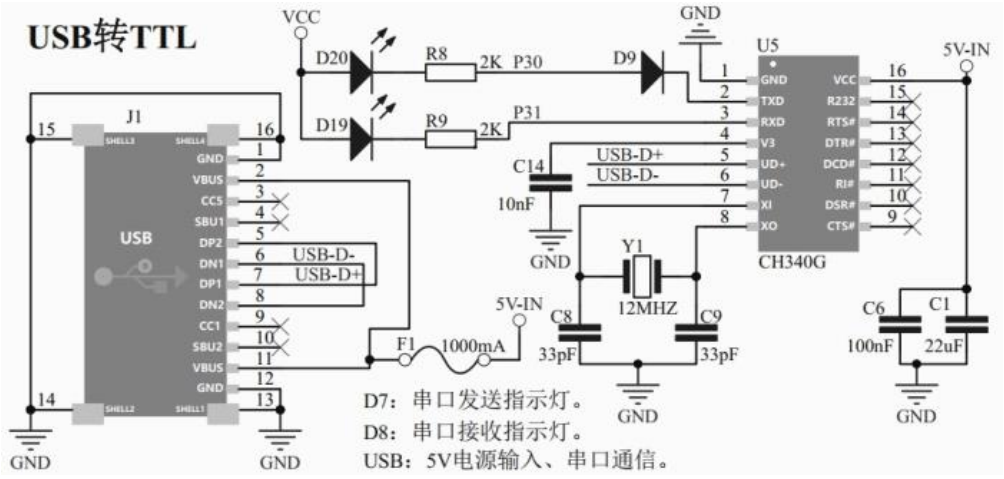


图 1：串口电路

开发板上 USB 转串口的主要作用如下：

1) USB 转串口（TTL 电平）通信：通过 USB 数据线连接到计算机的 USB 口即可使用串口通信功能。
2) 程序下载和仿真。这里要注意的是下载和仿真必须使用串口 1 的 P3.0 和 P3.1 这一组引脚（串口 1 有多组引脚，当然，同时只能配置其中的一组作为串口 1 的引脚）。
3) 开发板供电：通过 USB 可以为开发板供电（计算机 USB 可以提供约 500mA 的电流）。

在上面的电路图中，USB 转串口的接收和发送的管脚上均连接了 LED 指示灯，收发数据时指示灯会闪烁，这样，更方便我们从硬件的角度观察串口有没有在进行数据收发。

这一设计对于产品的调试和维护很方便，试想一下，当我们的产品通过串口向外发送数据，而对方没收到，这时，我们可以通过观察发送指示灯是否闪烁来快速判断数据有没有发送出去，从而方便我们定位问题。

USB 转串口电路采用的 USB 转串口芯片是 CH340，该芯片特点如下：

- 全速 USB 设备接口，兼容 USB V2.0。
- 标准 USB 打印口，用于升级原并口打印机，兼容相关的 USB 规范。
- 支持 IEEE-1284 规范的双向通信，支持单向和双向传输打印机。
- 由于是通过 USB 转换的打印口，所以只能做到应用层兼容，而无法绝对相同。
- 软件兼容 CH341，可以直接使用 CH341 的驱动程序。
- 支持 5V 电源电压和 3.3V 电源电压甚至 3V 电源电压。
- 采用无铅封装，兼容 RoHS，引脚兼容 CH341。

另外，CH340 芯片内置了 USB 上拉电阻，所以我们直接将 UD+和 UD-引脚连接 USB 总线上即可，而不用在芯片外部加上拉电阻。同时，CH340 芯片也内置了电源上电复位电路，不需要另外增加外部复位电路。

3. IAP15F2K61S2 串口

IAP15F2K61S2 共有 2 个 UART，他们是相互独立的，可以同时使用。每个 UART 会有多组引脚与之对应（具体几组还取决于芯片封装引脚数），注意同一个 UART 只能通过相关寄存器配置其中的一组使用，比如 P3.0、P3.1 是串口 1，而 P1.6、P1.7 也是串口 1，在使用串口 1 时只能选择其中一组使用，而不能同时将 P3.0、P3.1 和 P1.6、P1.7 这 2 组引脚用于串口 1。IAP15F2K61S2 单片机串口的引脚分配如下表。

表 1：IAP15F2K61S2 单片机 2 个串口引脚分配

串口	信号名称	引脚	说明
UART1	TxD	P3.1	串口 1 第 1 组引脚的发送引脚
	RxD	P3.0	串口 1 第 1 组引脚的接收引脚
	TxD_2	P3.7	串口 1 第 2 组引脚的发送引脚
	RxD_2	P3.6	串口 1 第 2 组引脚的接收引脚
	TxD_3	P1.7	串口 1 第 3 组引脚的发送引脚
	RxD_3	P1.6	串口 1 第 3 组引脚的接收引脚
UART2	TxD2	P1.1	串口 2 第 1 组引脚的发送引脚
	RxD2	P1.0	串口 2 第 1 组引脚的接收引脚

3.1. 串口引脚配置

IAP15F2K61S2 的 UART1 有多组引脚与之对应，因此，使用串口时需要配置该串口使用哪一组引脚。

串口 1 是通过“外设端口切换控制寄存器 1（P_SW1）”中的 S1_S[1:0]配置的，如下图所示。

外设端口切换控制寄存器 1（AUXR1 或 P_SW1）：

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0100,0000

串口1功能引脚选择位

P_SW1 寄存器中的 S1_S[1:0]为串口 1 功能脚选择位，如下表所示。

表 2：串口 1 功能脚选择位

S1_S[1:0]	RxD	TxD
00	P3.0	P3.1
01	P3.6	P3.7
10	P1.6	P1.7
11	无效	

3.2. 串口工作模式

IAP15F2K61S2 单片机的 2 个 UART 均有多种工作模式，其中 2 种工作模式的波特率是可变的，另 2 种工作模式的波特率是固定的，以供不同应用场合选用。串口 1 有 4 种工作模式，串口 2 只用 2 种工作模式，这 2 种工作模式的波特率都是可变的。下表列出了 2 个 UART 的工作模式。

表 3：串口工作模式

串口	工作模式	描述	备注
UART1	模式 0	同步移位串行方式：移位寄存器	不建议学习
	模式 1	8 位 UART，波特率可变	推荐学习
	模式 2	9 位 UART，波特率固定	不建议学习
	模式 3	9 位 UART，波特率可变	可以学习
UART2	模式 0	8 位 UART，波特率可变	推荐学习
	模式 1	9 位 UART，波特率可变	可以学习

1. 串口 1 工作模式

串口 1 的工作模式通过“串口 1 控制寄存器（SCON）”中的 SM0 位和 SM1 位配置。当 PCON 寄存器中的 SMOD0 位为 0 时，SM0 位和 SM1 位的组合决定了串口 1 的通信工作模式，如下表所示。

表 4：串口 1 工作模式配置

SM0	SM1	工作模式	功能描述
0	0	模式 0	同步移位串行方式：移位寄存器

0	1	模式 1	8 位 UART，波特率可变
1	0	模式 2	9 位 UART，波特率固定
1	1	模式 3	9 位 UART，波特率可变

2. 串口 2 工作模式

串口 2 的工作模式通过“串口 2 控制寄存器（S2CON）”中的 S2SM0 位配置，如下表所示。

表 5：串口 2 工作模式配置

S2SM0	工作模式	功能描述
0	模式 0	8 位 UART，波特率可变
1	模式 1	9 位 UART，波特率可变

3.3. 波特率的计算和配置

串口的波特率是指每秒传输了多少码元（二进制）的数据，单位是 bps，串口常用的波特率有 4800bps、9600 bps、19200 bps 和 115200 bps。串口通信时，发送方和接收方的波特率必须一样，否则是无法正常通信的。

3.3.1. 计算方法

串口的几种模式中，最常用的是“8 位 UART，波特率可变”的模式，因为这种模式下，定时器的值在硬件上会自动重装，无需在中断里面通过软件赋值，这样就不会因为软件参与而产生误差。

本节我们以串口 1 的“8 位 UART，波特率可变”的模式为例来说明波特率的配置。波特率配置主要涉及到波特率加倍配置；选择波特率发生器使用的定时器、设置定时器速度；计算定时器重装值这三个方面。

1. 波特率加倍配置

SMOD 是电源管理寄存器（PCON）的第 7 位，如下图所示，他是串口 1 波特率控制位。

电源管理寄存器（PCON）：

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PCON	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL

SMOD：串口 1 波特率控制位

- 0：串口 1 的各个模式的波特率都不加倍。
- 1：串口 1 模式 1（使用模式 2 的定时器 1 作为波特率发生器时有效）、模式 2、模式 3（使用模式 2 的定时器 1 作为波特率发生器时有效）的波特率加倍。

通常，我们会使用“波特率不加倍”，即 SMOD 设置为 0。

2. 选择波特率发生器使用的定时器、设置定时器速度

串口 1 的波特率是可变的，其波特率可由定时器 1 或者定时器 2 产生，通过“辅助寄存器 1（AUXR）”的位 S1ST2 可设置使用的定时器。

辅助寄存器 1（AUXR）：

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

S1ST2：串口 1 波特率发生器选择位

- 0：选择定时器 1 作为波特率发生器。
- 1：选择定时器 2 作为波特率发生器。

选择了波特率发生器使用的定时器后，还需设置定时器的模式（1T 模式或 12T 模式），定时器 1 通过“辅助寄存器 1（AUXR）”的位 T1x12 设置，定时器 2 通过“辅助寄存器 1（AUXR）”的位 T2x12 设置。当定时器采用 1T 模式时（12 倍速），相应的波特率的速度也会相应提高 12 倍。

T1x12：定时器 1 速度控制位

- 0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）。
- 1：1T 模式，即 CPU 时钟不分频（FOSC/1）。

T2x12：定时器 2 速度控制位

- 0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）。
- 1：1T 模式，即 CPU 时钟不分频（FOSC/1）。

3. 计算定时器重装值

串口 1 模式 1 的波特率计算公式如下表所示，其中 SYSclk 为系统工作频率。这里需要注意到，波特率发生器使用定时器 1 的话，可以选择 16 位重装或者 8 位重装，波特率发生器使用定时器 2 的话，只能用 16 位重装。

表 6：串口 1 模式 1 的波特率计算公式（SYSclk 为系统工作频率）

选择定时器	定时器速度	重装值计算公式	波特率
定时器2	1T	定时器2重载值 = 65536 - $\frac{SYSclk}{4\times波特率}$	波特率 = $\frac{SYSclk}{4\times(65536-定时器重装数)}$
	12T	定时器2重载值 = 65536 - $\frac{SYSclk}{12\times4\times波特率}$	波特率 = $\frac{SYSclk}{12\times4\times(65536-定时器重装数)}$
定时器1模式0	1T	定时器1重载值 = 65536 - $\frac{SYSclk}{4\times波特率}$	波特率 = $\frac{SYSclk}{4\times(65536-定时器重装数)}$
	12T	定时器1重载值 = 65536 - $\frac{SYSclk}{12\times4\times波特率}$	波特率 = $\frac{SYSclk}{12\times4\times(65536-定时器重装数)}$
定时器1模式2	1T	定时器1重载值 = 256 - $\frac{2^{SMOD}\times SYSclk}{32\times波特率}$	波特率 = $\frac{2^{SMOD}\times SYSclk}{32\times(256-定时器重装数)}$
	12T	定时器1重载值 = 256 - $\frac{2^{SMOD}\times SYSclk}{12\times32\times波特率}$	波特率 = $\frac{2^{SMOD}\times SYSclk}{12\times32\times(256-定时器重装数)}$

■ **计算举例**：系统时钟频率为 12MHz，波特率发生器使用定时器 1（1T），波特率不加倍，计算串口 1 使用模式 1，波特率为 9600bps 时的定时器重载值。

计算过程如下，由上表可以知：

$$定时器1\ 重装值 = 65536 - \frac{SYSclk}{4\times9600}$$

=65536-313

=65223

将 65223 转换为 16 进制，即 0xFEC7。所以对定时器 1 的高 8 位寄存器初始装载值为 0xFE，低 8 位寄存器初始装载值为 0x C7。

3.3.2. 使用工具软件计算

对于波特率的计算，知道原理即可。在学习过程中如果每次改动波特率都需要去计算一次，无疑是很麻烦的，而且，也会降低开发效率。宏晶科技考虑到了这一点，为方便广大开发者，宏晶科技发布了波特率计算的工具，我们只需输入相关参数，即可得到波特率配置的代码以及波特率的误差，使用起来很方便。该工具具体的使用步骤如下。

1. 打开 STC-ISP 软件后，依次点击“工具→独立使用波特率计算工具(B)”，打开波特率计算工具。

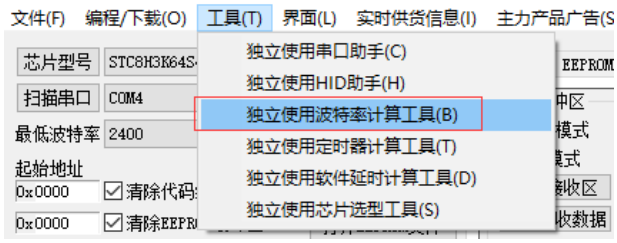


图 2：打开波特率计算工具

2. 选择相关参数后，点击“生成 C 代码”即可获取串口初始化代码，里面包含了波特率配置，同时也可以看到该配置下波特率的误差。

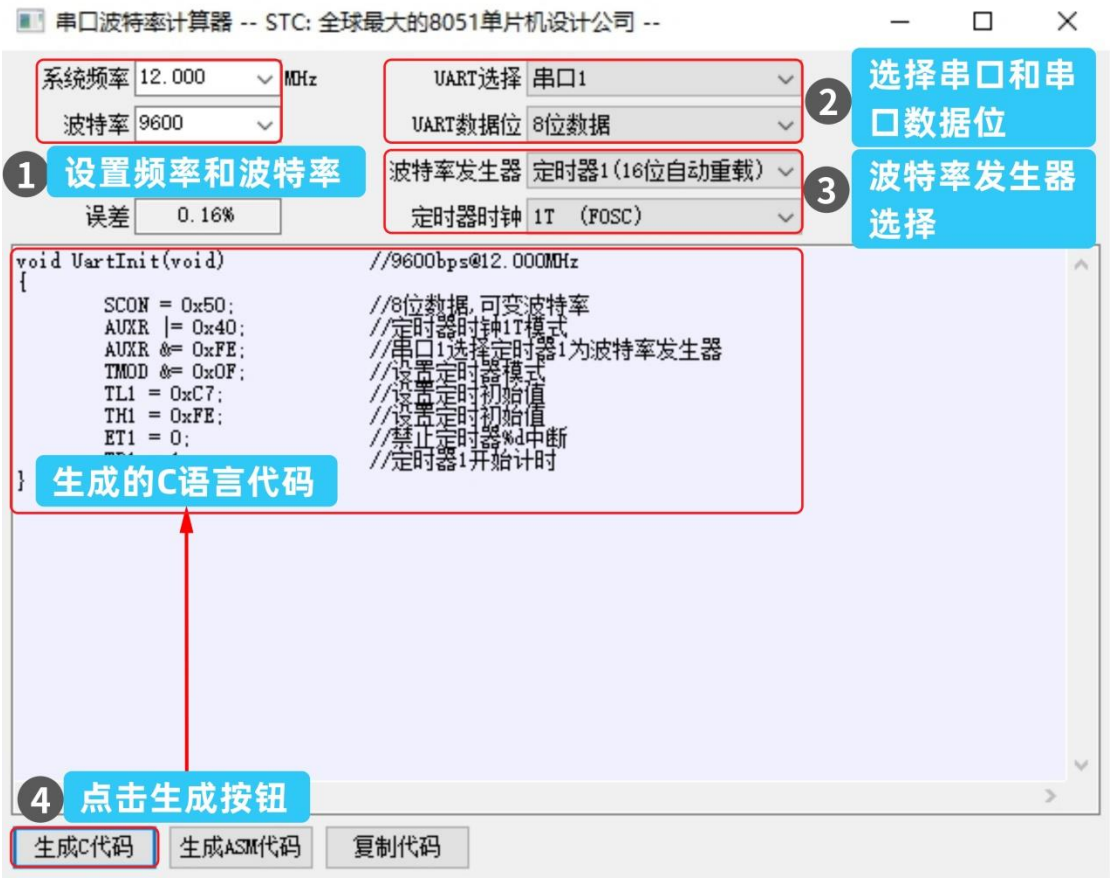


图 3：计算波特率

3.4. 串口发送

串行通信模式发送时，需要先将待发送的数据写入到 SBUF 寄存器。

■ **串口 1 数据寄存器（ SBUF）：**

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SBUF	99H								

SBUF：串口 1 数据接收/发送缓冲区。 SBUF 实际是 2 个缓冲器， 读缓冲器和写缓冲器，两个操作分别对应两个不同的寄存器，1 个是只写寄存器（写缓冲器），1 个是只读寄存器（读缓冲器）。对 SBUF 进行读操作，实际是读取串口接收缓冲区，对 SBUF 进行写操作则

是触发串口开始发送数据。

当写 SBUF 的指令执行后，待发送数据被写入到 SBUF 寄存器，并且串行通信的发送序列也被启动，同时，写“SBUF”信号还把“1”装入发送移位寄存器的第 9 位，如下图所示。

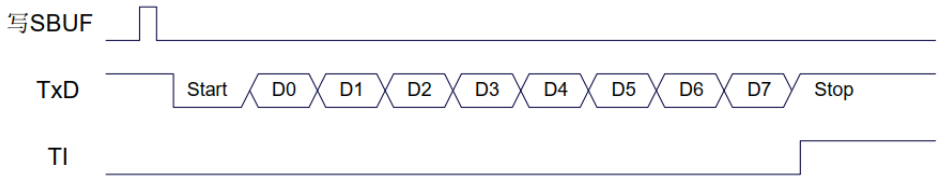


图 4：串口 1 模式 1 发送数据

发送启动后，移位寄存器将数据不断右移送 TxD 引脚将其传送到物理线路，在数据的左边不断移入“0”作补充。当数据的最高位移到移位寄存器的输出位置，紧跟其后的是第 9 位“1”，在他的左边各位全为“0”，这个状态条件，使 TX 控制单元作最后一次移位输出，然后使允许发送信号“SEND”失效，完成一帧信息的发送，并置位中断请求位 TI，即 TI=1，向主机请求中断处理。

发送中断请求位 TI 是“串口 1 控制寄存器（SCON）”的位 1，如下所示。

■ **串口 1 控制寄存器（SCON）：**

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI

TI：串口 1 发送中断请求标志位。在模式 0 中，当串口发送数据第 8 结束时，由硬件自动将 TI 置 1，向主机请求中断，响应中断后 TI 必须用软件清零。在其他模式中，则在停止位开始发送时由硬件自动将 TI 置 1，向 CPU 发请求中断，响应中断后 TI 必须用软件清零。

了解了串口发送的过程，接下来，我们再看一下，在处理串口发送时，常用的操作方式。串口发送操作有两种方式：查询方式和中断方式。

1) 查询方式：待发送数据写入“SBUF”后，通过查询中断请求位 TI 是否置位（TI=1）来判断数据是否发送完成，若 TI 置位，则表示当前数据发送完成，可以发送下一个数据。查询模式下是无需使能串口中断的。

2) 中断方式：使能串口中断和系统中断，待发送数据写入“SBUF”后，无需查询中断请求位 TI，数据发送完成后会触发串口中断，应该程序由此获知数据发送完成。

实际应用中，我们通常会使用查询方式发送数据，下面的代码是串口 1 以查询方式发送一字节数据的示例。

代码清单：串口 1 查询方式发送数据

```
1.  /*********************************************************************************
2.  功能描述：向串口 1 发送 1 字节数据，并等待发送完成
3.  参    数：dat[in]：要发送的数据
4.  返 回 值：无
```

```
5.  ********************************************************************************/
6.  void uart1_send_byte(u8 dat)
7.  {
8.      SBUF = dat;             //待发送数据写入串口 1 数据寄存器 SBUF
9.      while(TI == 0);         //查询中断请求位是否置位，由此判断数据是否发送完成
10.     TI = 0;                 //清零 TI 位（该位必须软件清零）
11. }
```

3.5. 串口接收

串口 1 模式 1 接收数据接收的时序如下图所示。

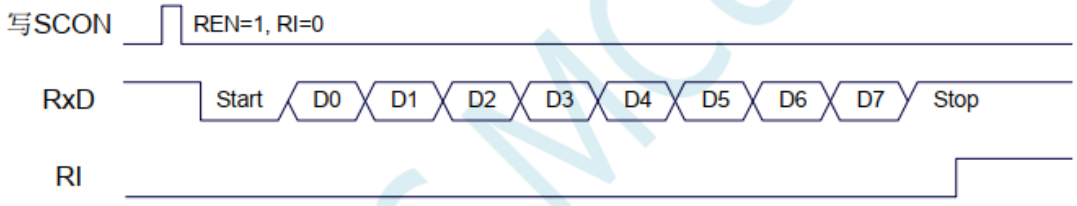


图 5：串口 1 模式 1 接收数据

当软件置位接收允许标志位 REN，即 REN=1 时，接收器便对 RxD 端口的信号进行检测，当检测到 RxD 端口发送从“1”→“0”的下降沿跳变时就启动接收器准备接收数据，并立即复位波特率发生器的接收计数器，将 1FFH 装入移位寄存器。接收的数据从接收移位寄存器的右边移入，已装入的 1FFH 向左边移出，当起始位“0”移到移位寄存器的最左边时，使 RX 控制器作最后一次移位，完成一帧的接收。

若以下两个条件同时满足：

1) RI=0；
2) SM2=0 或接收到的停止位为 1。（SM2 是多机通信控制位，串口工作于模式 1 时，通常将 SM2 设置为"0"）。

则接收到的数据有效，数据装载入 SBUF，停止位进入 RB8，RI 标志位被置位，并请求中断，若上述两条件不能同时满足，则接收到的数据作废并丢失。无论条件满足与否，接收器重又检测 RxD 端口上的"1"→"0"的跳变，继续下一帧的接收。

接收中断请求标志是“串口 1 控制寄存器（SCON）”的位 0，如下所示。RI 标志置位后，硬件不会自动清零，必须由软件清零。

■ **串口 1 控制寄存器（SCON）：**

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI

RI：串口 1 接收中断请求标志位。在模式 0 中，当串口接收第 8 位数据结束时，由硬件自动将 RI 置位，向主机请求中断，响应中断后 RI 必须用软件清零。在其他模式中，串行接收到停止位的中间时刻由硬件自动将 RI 置 1，向 CPU 发中断申请，响应中断后 RI 必须由软件清零。

串口接收数据时，和发送一样，操作方式也有查询方式接收和中断方式接收。

1) 查询方式：通过查询接收中断请求标志位 RI 是否置位（RI=1）来判断串口是否接收到数据。

2) 中断方式：使能串口中断和系统中断，串口接收到数据后会触发中断，在中断服务函数中完成数据接收。

实际应用中，我们通常会使用中断方式接收数据，很少会用查询方式去接收数据。下面的代码是串口 1 以中断方式接收数据的示例。

代码清单：串口 1 中断方式接收数据

```
/*******************************************************************************
 * 描  述 : 串口 1 中断服务函数
 * 入  参 : 无
 * 返回值 : 无
 *******************************************************************************/
void Uart1() interrupt 4 using 1
{
    ES = 0;                     //串口 1 中断关闭

    if (RI)                     //是接收中断（接收中断请求标志位为 1）
    {
       RI = 0;                  //清零 RI 位（该位必须软件清零）
       ......
       uart_rx[i++] = SBUF;     //读取串口接收的数据
       ......
     }
     ES =  1;                   //串口 1 中断打开
}
```

4. IAP15W4K61S4 串口

IAP15W4K61S4 共有 4 个 UART，他们是相互独立的，可以同时使用。每个 UART 会有多组引脚与之对应（具体几组还取决于芯片封装引脚数），注意同一个 UART 只能通过相关寄存器配置其中的一组使用，比如 P3.0、P3.1 是串口 1，而 P1.6、P1.7 也是串口 1，在使用串口 1 时只能选择其中一组使用，而不能同时将 P3.0、P3.1 和 P1.6、P1.7 这 2 组引脚用于串口 1。IAP15W4K61S4 单片机串口的引脚分配如下表。

表 7：IAP15W4K61S4 单片机 4 个串口引脚分配

串口	信号名称	引脚	说明
UART1	TxD	P3.1	串口 1 第 1 组引脚的发送引脚
	RxD	P3.0	串口 1 第 1 组引脚的接收引脚
	TxD_2	P3.7	串口 1 第 2 组引脚的发送引脚

	RxD_2	P3.6	串口 1 第 2 组引脚的接收引脚
	TxD_3	P1.7	串口 1 第 3 组引脚的发送引脚
	RxD_3	P1.6	串口 1 第 3 组引脚的接收引脚
UART2	TxD2	P1.1	串口 2 第 1 组引脚的发送引脚
	RxD2	P1.0	串口 2 第 1 组引脚的接收引脚
	TxD2_2	P4.2	串口 2 第 2 组引脚的发送引脚
	RxD2_2	P4.0	串口 2 第 2 组引脚的接收引脚
UART3	TxD3	P0.1	串口 3 第 1 组引脚的发送引脚
	RxD3	P0.0	串口 3 第 1 组引脚的接收引脚
	TxD3_2	P5.1	串口 3 第 2 组引脚的发送引脚
	RxD3_2	P5.0	串口 3 第 2 组引脚的接收引脚
UART4	TxD4	P0.3	串口 4 第 1 组引脚的发送引脚
	RxD4	P0.2	串口 4 第 1 组引脚的接收引脚
	TXD4_2	P5.3	串口 4 第 2 组引脚的发送引脚
	RxD4_2	P5.2	串口 4 第 2 组引脚的接收引脚

✧ **说明：**串口 2、串口 3 和串口 4，针对 DIP40 封装是没有第 2 组引脚的。上述表格是针对该系列单片机其他封装来描述的，比如 LQFP64 封装。

4.1. 串口引脚配置

IAP15W4K61S4 的 4 个 UART 均有多组引脚与之对应，因此，使用串口时需要配置该串口使用哪一组引脚。

串口 1 是通过“外设端口切换控制寄存器 1（P_SW1）”中的 S1_S[1:0]配置的，如下图所示。

外设端口切换控制寄存器 1（AUXR1 或 P_SW1）：

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0100,0000

串口1功能引脚选择位

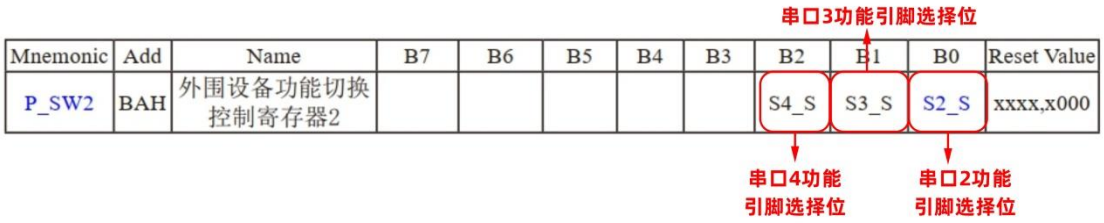
P_SW1 寄存器中的 S1_S[1:0]为串口 1 功能脚选择位，如下表所示。

表 8：串口 1 功能脚选择位

S1_S[1:0]	RxD	TxD
00	P3.0	P3.1
01	P3.6	P3.7
10	P1.6	P1.7
11	无效	

串口 2、3、4 是通过“外设端口切换控制寄存器 2（P_SW2）”中的 S2_S、S3_S 和 S4_S 配置的，如下图所示。

外设端口切换控制寄存器 2（P_SW2）：



- S2_S：串口 2 功能脚选择位，如下表所示。

表 9：串口 2 功能脚选择位

S2_S	RxD	TxD
0	P1.0	P1.1
1	P4.0	P4.2

- S3_S：串口 3 功能脚选择位，如下表所示。

表 10：串口 3 功能脚选择位

S3_S	RxD	TxD
0	P0.0	P0.1
1	P5.0	P5.1

- S4_S：串口 4 功能脚选择位，如下表所示。

表 11：串口 4 功能脚选择位

S4_S	RxD	TxD
0	P0.2	P0.3
1	P5.2	P5.3

4.2. 串口工作模式

IAP15W4K61S4 单片机的 4 个 UART 均有多种工作模式，其中 2 种工作模式的波特率是可变的，另 2 种工作模式的波特率是固定的，以供不同应用场合选用。串口 1 有 4 种工作模式，串口 2、串口 3 和串口 4 均只用 2 种工作模式，这 2 种工作模式的波特率都是可变的。下表列出了 4 个 UART 的工作模式。

表 12：串口工作模式

串口	工作模式	描述	备注
UART1	模式 0	同步移位串行方式：移位寄存器	不建议学习
	模式 1	8 位 UART，波特率可变	推荐学习
	模式 2	9 位 UART，波特率固定	不建议学习
	模式 3	9 位 UART，波特率可变	可以学习

UART2	模式 0	8 位 UART，波特率可变	推荐学习
	模式 1	9 位 UART，波特率可变	可以学习
UART3	模式 0	8 位 UART，波特率可变	推荐学习
	模式 1	9 位 UART，波特率可变	可以学习
UART4	模式 0	8 位 UART，波特率可变	推荐学习
	模式 1	9 位 UART，波特率可变	可以学习

3. 串口 1 工作模式

串口 1 的工作模式通过“串口 1 控制寄存器（SCON）”中的 SM0 位和 SM1 位配置。当 PCON 寄存器中的 SMOD0 位为 0 时，SM0 位和 SM1 位的组合决定了串口 1 的通信工作模式，如下表所示。

表 13：串口 1 工作模式配置

SM0	SM1	工作模式	功能描述
0	0	模式 0	同步移位串行方式：移位寄存器
0	1	模式 1	8 位 UART，波特率可变
1	0	模式 2	9 位 UART，波特率固定
1	1	模式 3	9 位 UART，波特率可变

4. 串口 2 工作模式

串口 2 的工作模式通过“串口 2 控制寄存器（S2CON）”中的 S2SM0 位配置，如下表所示。

表 14：串口 2 工作模式配置

S2SM0	工作模式	功能描述
0	模式 0	8 位 UART，波特率可变
1	模式 1	9 位 UART，波特率可变

5. 串口 3 工作模式

串口 3 的工作模式通过“串口 3 控制寄存器（S3CON）”中的 S3SM0 位配置，如下表所示。

表 15：串口 3 工作模式配置

S3SM0	工作模式	功能描述
0	模式 0	8 位 UART，波特率可变
1	模式 1	9 位 UART，波特率可变

6. 串口 4 工作模式

串口 4 的工作模式通过“串口 4 控制寄存器（S2CON）”中的 S4SM0 位配置，如下表

所示。

表 16：串口 4 工作模式配置

S4SM0	工作模式	功能描述
0	模式 0	8 位 UART，波特率可变
1	模式 1	9 位 UART，波特率可变

✧ **说明：**关于单片机 IAP15W4K61S4 的波特率计算和配置、串口接收和发送，请参考单片机 IAP15F2K61S2 部分讲解，原理都是一样的。

5. 软件设计

5.1. 串口应用步骤

串口应用的步骤主要包括串口初始化、数据发送和数据接收，这里面串口初始化内容比较固定，容易掌握。至于数据发送，因为 IAP15F2K61S2/IAP15W4K61S4 没有硬件 FIFO，因此大多采样查询的方式发送，也容易理解。相对来说，最难的部分是数据接收，因为数据接收存在随机性，接收方通常不知道对方什么时候发数据过来，也不知道对方一次会发多少数据过来，因此，处理起来相对复杂一些。

1. 串口初始化

串口初始化包含设置串口使用的引脚、串口的工作模式、波特率以及中断的开启（如果仅仅使用串口发送，通常使用查询方式发送，可以不使能中断；如果使用了串口接收功能，通常使用中断接收，需要使能串口中断）。

下面的代码是串口 1 初始化的示例，代码中对串口 1 进行了如下配置。

- 串口 1 使用的引脚为：RxD：P3.0；TxD：P3.1。
- 串口 1 的工作模式为：模式 1，此模式为 8 位 UART 格式，一帧信息为 10 位：1 位起始位、 8 位数据位（低位在先）和 1 位停止位。
- 波特率：9600bps，波特率发生器使用定时器 1，16 位重装。
- 串口中断：使能，因为我们需要使用串口接收功能（中断方式接收）。

代码清单：串口 1 初始化

```
1.  /*******************************************************************************
2.  功能描述：初始化串口 1，设置为 8 位数据位、波特率 9600bps （系统时钟使用 24MHz）
3.  参    数：无
4.  返 回 值：无
5.  *******************************************************************************/
6.  void uart1_init(void)
7.  {
8.      ES = 0;             //初始化前关闭 UART1 中断
9.      P_SW1 &= 0xFC;      //设置串口 1 使用的引脚为：RxD--P3.0;TxD--P3.1
10.     PCON &= 0x3f;       //波特率不倍速，串行口工作方式由 SM0、SM1 决定
11.     SCON = 0x50;        //8 位数据,可变波特率（SM0=0,SM1=1）
```

```
12.     AUXR |= 0x40;       //定时器时钟 1T 模式
13.     AUXR &= 0xFE;       //串口 1 选择定时器 1 为波特率发生器
14.
15.     TMOD &= 0x0F;       //设置定时器 1 模式：16 位自动重装方式
16.     TL1 = 0x8F;         //设置定时初始值
17.     TH1 = 0xFD;         //设置定时初始值
18.
19.     ET1 = 0;            //禁止定时器 1 中断
20.     TR1 = 1;            //启动定时器 1
21.     ES = 1;             //串口 1 中断打开
22. }
```

2. 数据发送

前文中，我们给出了使用查询方式发送一个字节数据的代码示例，当我们有多个数据需要发送的时候，使用循环语句调用该函数发送数据，直到数据全部发送完成即可。

3. 数据接收

大多数情况下，接收数据都存在不确定性，就比如我们下面要做的串口收发的例子，开发板无法知道串口调试助手什么时候给他发数据，也无法确定一次发多少个字节的数据。在这种情况下，接收方为了保证完整地接收搭配数据，应该怎么做？

通常，我们会做一个软件缓存，在串口中断中接收数据存入到软件缓存，并定义一个变量用于记录串口接收的字节数（接收计数器），应用程序中可以通过查询接收计数器从而判断串口是否接收到数据，如接收到数据，则从缓存中取出数据进行处理。该方式的软件流程如下图所示，具体实现的代码参见下一节中的例子。

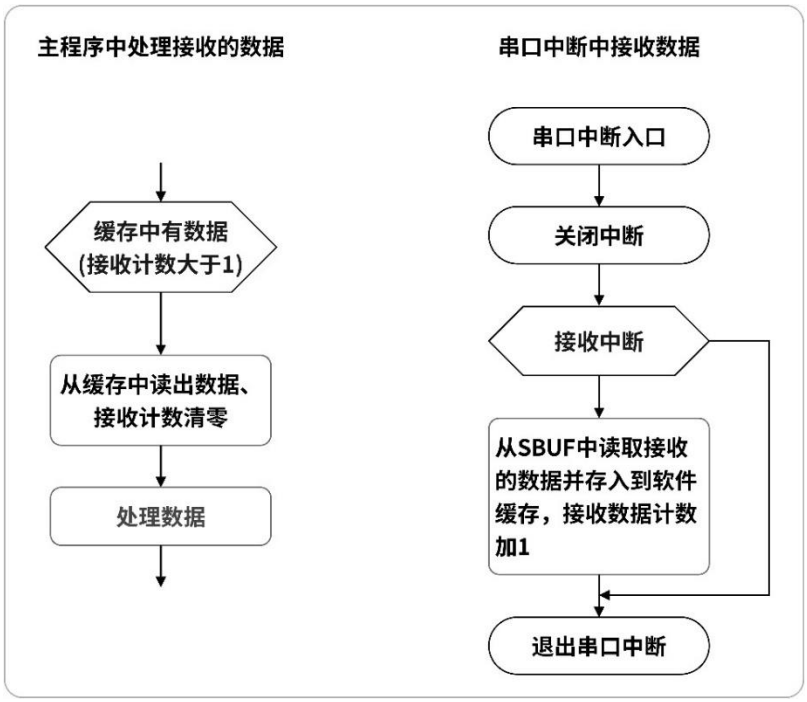


图 6：串口接收处理流程

5.2. 串口数据收发实验

✧ 注：本节的实验是在“实验 2-3-1：独立按键检测”的基础上修改，本节对应的实验源码是：“实验 2-5-1：串口 1 数据收发实验”。

5.2.1. 实验内容

1. 配置串口 1 使用的引脚为：RxD 为 P3.0，TxD 为 P3.1。8 位数据位，波特率 9600bps。
2. 串口 1 接收电脑端串口调试助手发过来的数据，并将接收到的数据原样返回。

5.2.2. 代码编写

1. 新建一个名称为“uart.c”的文件及其头文件“uart.h”并保存到工程的“Source”文件夹，并将“uart.c”加入到 Keil 工程中的“SOURCE”组。
2. 引用头文件

因为在“main.c”文件中使用了“uart.c”文件中的函数，所以需要引用下面的头文件“uart.h”。

代码清单：引用头文件

```
1.  //引用串口的头文件
2.  #include    "uart.h"
```

3. 串口初始化

串口 1 初始化函数代码清单如下，该函数中设置了串口 1 使用的引脚：RxD 为 P3.0，TxD 为 P3.1。串口 1 配置为 8 位数据位、波特率 9600bps。同时，初始化函数中也初始化了串口接收软件缓存（软件缓存在下文的串口接收部分描述）。

代码清单：串口初始化

```
1.  /*******************************************************************************
2.  功能描述：初始化串口 1，设置为 8 位数据位、波特率 9600bps （系统时钟使用 12MHz）
3.  参    数：无
4.  返 回 值：无
5.  *******************************************************************************/
6.  void uart1_init(void)
7.  {
8.      ES = 0;             //初始化前关闭 UART1 中断
9.
10.     P_SW1 &= 0xFC;      //设置串口 1 使用的引脚为：RxD--P3.0;TxD--P3.1
11.     uart1_init_fifo();  //初始化软件缓存
12.
13.     PCON &= 0x3f;       //波特率不倍速，串行口工作方式由 SM0、SM1 决定
14.     SCON = 0x50;        //8 位数据,可变波特率（SM0=0,SM1=1）
15.     AUXR |= 0x40;       //定时器时钟 1T 模式
16.     AUXR &= 0xFE;       //串口 1 选择定时器 1 为波特率发生器
17.     TMOD &= 0x0F;       //设置定时器 1 模式：16 位自动重装方式
18.     TL1 = 0XC7;         //设置定时初始值
```

```
    TH1 = 0xFE;          //设置定时初始值
    
    ET1 = 0;             //禁止定时器 1 中断
    TR1 = 1;             //启动定时器 1
    ES = 1;              //串口 1 中断打开
}
```

4. 串口数据发送

串口发送采用查询方式，待发送数据写入 SBUF 后，一直等待中断请求位 TI 置位，TI 置位后，软件将其清零。串口发送函数代码清单如下。

代码清单：串口发送函数

```
/*********************************************************************************
功能描述：向串口 1 发送 1 字节数据，并等待发送完成
参    数：dat[in]：要发送的数据
返 回 值：无
*********************************************************************************/
void uart1_send_byte(u8 dat)
{
    SBUF = dat;          //待发送数据写入串口 1 数据寄存器 SBUF
    while(TI == 0);      //查询中断请求位是否置位，由此判断数据是否发送完成
    TI = 0;              //清零 TI 位（该位必须软件清零）
}
```

5. 串口数据接收

串口接收相对麻烦一些，这里，我们做了一个软件缓存用于存储接收到的数据。该软件缓存是基于循环队列的原理实现的，具体代码实现方式如下。

定义一个数组用来存放串口接收到数据，定义一个变量用来记录缓存中写的位置，称为“写指针”，定义一个变量用来记录缓存中读的位置，称为“读指针”，同时定义一个变量用来记录数组中数据个数，称为“计数器”，代码清单如下。

代码清单：软件缓存相关变量定义

```
#define UART_BUF_LEN   32             //串口接收软件缓存大小，如果接收的数据较多，可以修改缓存大小

static u8 idata uart_rx[UART_BUF_LEN]; //定义串口接收缓存数据
static u8 uart_rx_wp, uart_rx_rp;      //分别用来记录在软件缓存中写数据和读数据的位置
static u8 uart_rx_cnt;                 //缓存数据计数器
```

接收软件缓存在使用前需要初始化，即将用于记录的各个变量清零，初始化函数代码清单如下。

代码清单：软件缓存初始化

```
/*********************************************************************************
功能描述：初始化软件缓存
```

```
3.  参    数：无
4.  返 回 值：无
5.  ********************************************************************************/
6.  void uart1_init_fifo(void)
7.  {
8.      uart_rx_wp = 0;
9.      uart_rx_rp = 0;
10.     uart_rx_cnt = 0;
11. }
```

串口中断服务函数中，将接收到的串口数据写入缓存，同时计数器加 1。这里需要注意，写入数据时需要判断写指针是否到达缓存尾，如果到达缓存尾，则翻转，重新指向缓存头。

代码清单：串口 1 中断服务函数

```
1. /********************************************************************************
2.  * 描   述 : 串口 1 中断服务函数
3.  * 入   参 : 无
4.  * 返回值 : 无
5.  ********************************************************************************/
6. void uart1_isr() interrupt 4 using 1
7. {
8.     ES = 0;                          //串口 1 中断关闭
9.
10.    if (RI)                          //是接收中断（接收中断请求标志位为 1）
11.    {
12.            RI = 0;                  //清零 RI 位（该位必须软件清零）
13.
14.            if (uart_rx_cnt < UART_BUF_LEN)
15.            {
16.                uart_rx[uart_rx_wp] = SBUF; //数据写入缓存
17.                uart_rx_wp = (uart_rx_wp + 1) % UART_BUF_LEN;//写指针加 1，若到达缓存尾，则从头开始
18.                uart_rx_cnt++;//串口接收计数器加 1
19.            }
20.    }
21.    ES =  1;                    //串口 1 中断打开
22.}
```

数据存入软件缓存后，我们还需提供一个读取函数用于主程序从缓存中读取数据。读取函数的代码清单如下，这里同样需要注意，读数据时需要判断读指针是否到达缓存尾，如果到达缓存尾，则翻转，重新指向缓存头。

代码清单：从软件缓存中读取一个字节数据

```
1.  /********************************************************************************
2.  功能描述：从软件缓存中读取一个字节数据
3.  参    数：无
```

```
4.  返 回 值：读取的数据
5.  ******************************************************************************/
6.  u8 uart1_fifo_getbyte(void)
7.  {
8.    u8 ch;
9.
10.   ES = 0;                    //读取时关闭串口中断
11.   ch = uart_rx[uart_rx_rp]; //读取一个字节数据
12.   uart_rx_rp = (uart_rx_rp + 1) % UART_BUF_LEN;//读指针加 1，若到达缓存尾，则从头开始
13.   uart_rx_cnt--;//数据计数器减一
14.   ES = 1;       //读取完成后使能中断
15.   return ch;
16. }
```

6. 串口重定向 printf

在程序开发调试过程中，我们习惯使用 printf 函数来打印数据或调试信息，但在默认情况下，使用 printf 函数并不能直接通过串口输出数据，因此，我们需要重定向 printf 函数。重定向 printf 函数只需要改写 putchar 函数即可，本例中，我们将 printf 函数重定向到串口 1，代码清单如下：

代码清单：重定向 printf 函数到串口 1

```
1.  /*********************************************************************
2.   * 描  述 : 重定向 c 库函数 printf 到串口 1
3.   * 入  参 : char dat
4.   * 返回值 : char
5.   *********************************************************************/
6.  char putchar(char dat)
7.  {
8.      uart1_send_byte(dat);
9.      return dat;
10. }
```

printf 函数是 C 语言标准库函数，定义于头文件 <stdio.h>，因此，在使用 printf 函数的文件中需要引用该头文件，如下所示。

代码清单：引用 stdio.h 头文件

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

7. 主函数

主函数中，调用串口 1 初始化“uart1_init()”完成串口 1 初始化，并且使能了总中断。之后在主循环中查询接收软件缓存中是否有数据，如有数据，读出数据，并调用串口 1 发送函数“uart1_send_byte()”将读取的数据发送，由此完成串口接收数据的回环。

代码清单：主函数

```
1. /*********************************************************************
```

```
2. 功能描述：主函数
3. 入口参数：无
4. 返 回 值：int 类型
5. *******************************************************************************/
6. int main(void)
7. {
8.     u8 uart1_rece_length;
9.
10.    P3M1 &= 0xFE;   P3M0 &= 0xFE;     //设置 P3.0 为准双向口（串口 1 的 RxD）
11.    P3M1 &= 0xFD;   P3M0 |= 0x02;     //设置 P3.1 为推挽输出（串口 1 的 TxD）
12.    P2M1 &= 0x1F;   P2M0 |= 0xE0;     //设置 P2.5、P2.6、P2.7 为推挽输出
13.    P0M1 &= 0x00;   P0M0 |= 0xFF;     //设置 P0.0 ~ P0.7 为推挽输出
14.
15.    SEG_off();          //控制 8 位数码管/点阵不显示
16.    ULN2003_off();      //控制步进电机、蜂鸣器、继电器等不工作
17.    leds_off();         //熄灭 D1~D8 指示灯
18.    delay_ms(10);       //延时
19.
20.    uart1_init();       //串口 1 初始化
21.    EA = 1;             //使能总中断
22.
23.    printf("uart example started\r\n");
24.
25.    while(1)
26.    {
27.       uart1_rece_length = uart1_get_fifo_datalen(); //查询串口接收软件缓存中是否有数据
28.       if(uart1_rece_length > 0)                     //如有数据
29.       {
30.          uart1_send_byte(uart1_fifo_getbyte());    //从串口接收软件缓存中读取数据并发送
31.       }
32.    }
33.}
```

5.2.3. 硬件连接

本实验程序的编写都是基于 IO 模式，所以 J24 端子需要使用短路帽将该端子第 1 引脚和第 2 引脚短接，即选择为 IO 模式。同时将 USB 线接插到开发板 J1 可做串口 1 通信。

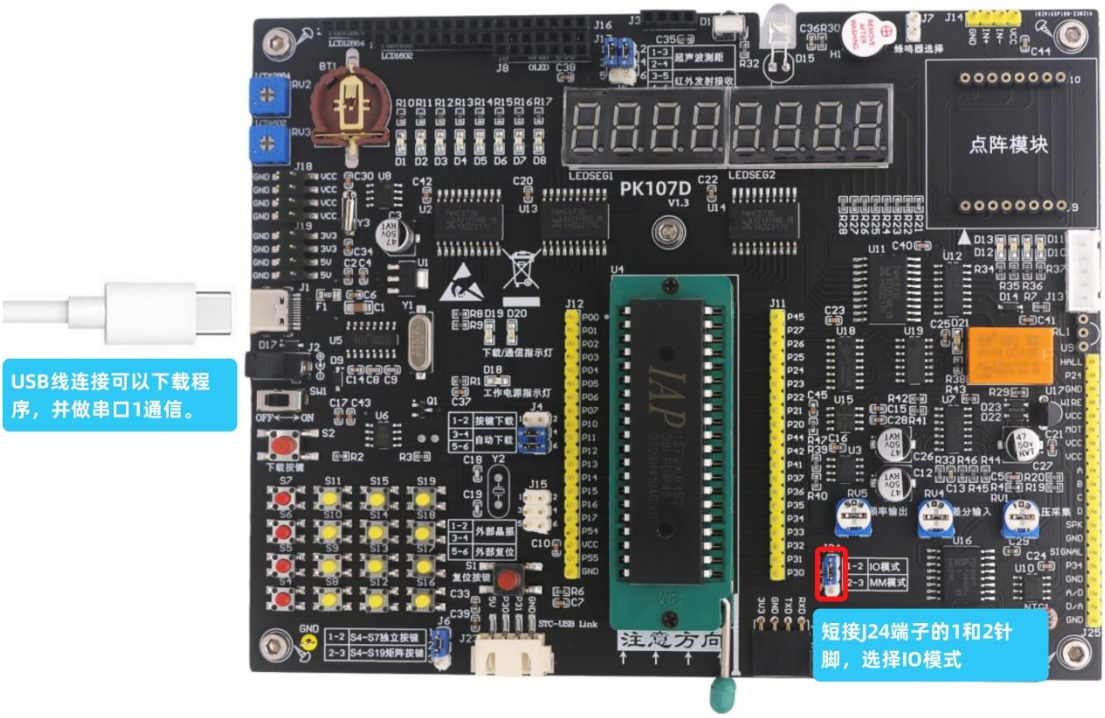


图 7：实验短接图

5.2.4. 实验步骤

1) 解压“…\第 3 部分：配套例程源码”目录下的压缩文件“实验 2-5-1：串口 1 数据收发实验”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC15”（这样做的目的是为了防止中文路径或者工程存放的路径过深导致打开工程出现问题）。
2) 双击“…\uart1_echo\project”目录下的工程文件“uart1_echo.uvproj”。
3) 点击编译按钮编译工程，编译成功后生成的 HEX 文件“uart1_echo.hex”位于工程的“…\uart1_echo\Project\Object”目录下。
4) 打开 STC-ISP 软件下载程序，下载使用内部 IRC 时钟，IRC 频率选择：12MHz。
5) 电脑上打开串口调试助手，选择开发板对应的串口号，将波特率设置为 9600bps，之后输入发送的数据，点击发送按钮发送数据。

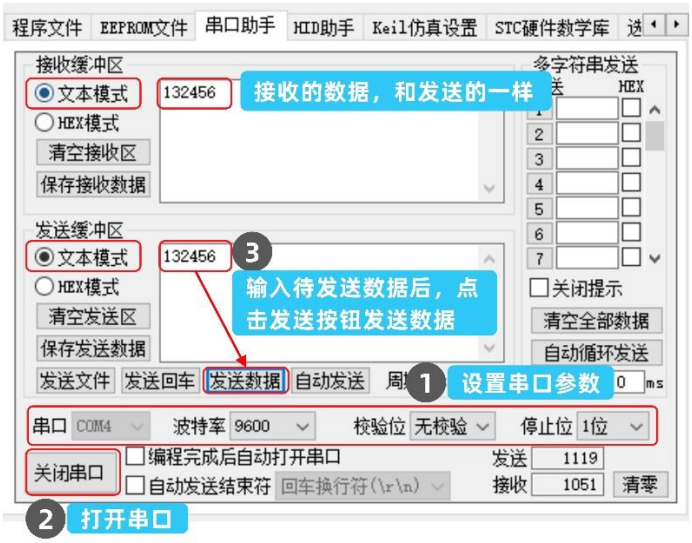


图 8：串口调试助手收发数据

6) 观察串口接收的数据，应和发送的数据一样。

5.3. 串口控制 LED 实验

✧ 注：本节的实验是在“实验 2-5-1：串口 1 数据收发实验”的基础上修改，本节对应的实验源码是：“实验 2-5-2：串口控制 LED 实验”。

5.3.1. 实验内容

串口数据收发中我们学习了串口如何收发数据，这一节我们来看一下如何解析简单命令。我们定义的 LED 指示灯控制的命令格式如下：

- 指示灯控制命令格式：‘#’+‘D’+ 指示灯标号（1、2、3、4、5、6、7、8）。
- 命令示例：控制指示灯 D1 点亮“#D1”。

开发板接收到一个命令包后，对命令包进行解析，并根据解析结果点亮相应的 LED 指示灯。

5.3.2. 代码编写

串口解析接收数据的代码清单如下，当串口接收到字符“#”，若此时接收数据长度 uart_rx_cnt 为 0，则认为接收到了命令包的起始字节。此后，当接收字节数达到 3 的时候，认为一个命令包接收完成。接着，从命令包中解析出要点亮的指示灯的编号，即将命令包中的第 3 个字节减去 48，得到 ASCII 对应的十进制数值，由此得到指示灯的编号。最后，调用指示灯操作函数点亮对应的指示灯即可。

代码清单：主函数

```
1.  /*******************************************************************************
2.  功能描述：主函数
3.  入口参数：无
4.  返 回 值：int 类型
5.  *******************************************************************************/
6.  int main(void)
7.  {
```

```
8.      u8 uart1_rece_length,i,cr;
9.
10.     P3M1 &= 0xFE;   P3M0 &= 0xFE;     //设置 P3.0 为准双向口（串口 1 的 RxD）
11.     P3M1 &= 0xFD;   P3M0 |= 0x02;     //设置 P3.1 为推挽输出（串口 1 的 TxD）
12.     P2M1 &= 0x1F;   P2M0 |= 0xE0;     //设置 P2.5、P2.6、P2.7 为推挽输出
13.     P0M1 &= 0x00;   P0M0 |= 0xFF;     //设置 P0.0 ~ P0.7 为推挽输出
14.
15.     SEG_off();          //控制 8 位数码管/点阵不显示
16.     ULN2003_off();      //控制步进电机、蜂鸣器、继电器等不工作
17.     leds_off();         //熄灭 D1~D8 指示灯
18.     delay_ms(10);       //延时
19.
20.     uart1_init();       //串口 1 初始化
21.     EA = 1;             //使能总中断
22.
23.     while(1)
24.     {
25.       uart1_rece_length = uart1_get_fifo_datalen(); //查询串口接收软件缓存中是否有数据
26.
27.             for(i=0; i<uart1_rece_length;i++)       //处理数据
28.             {
29.                 cr = uart1_fifo_getbyte();//从串口接收软件缓存中读取数据
30.
31.
32.                 if(start_flag == 0)         //数据包开始标志为 false
33.                 {
34.                     if(cr == '#')           //如果接收的数据是'#'，表示接收到数据包起始字节
35.                     {
36.                         if(uart_rx_cnt == 0)//接收到起始字节后，如果 uart_rx_cnt 也等于 0，表示当前流
    程是对的
37.                         {
38.                             uart_rece_buf[uart_rx_cnt++] = cr;
39.                             start_flag = 1;
40.                         }
41.                         else//清零接收计数
42.                         {
43.                             uart_rx_cnt = 0;
44.                         }
45.                     }
46.                 }
47.                 else//数据包开始标志为真
48.                 {
49.                     uart_rece_buf[uart_rx_cnt++] = cr;
```

```
50.                     if(uart_rx_cnt == UART_RECE_PACK_LEN)//接收命令完成
51.                     {
52.                         //检查数据是否合法
53.                         if((uart_rece_buf[1] == 'D') || (uart_rece_buf[1] == 'd'))
54.                         {
55.                             //减去 48，得到 ASCII 对应的十进制数值
56.                             switch(uart_rece_buf[2]-48)
57.                             {
58.                                 case 1:
59.                                     leds_off();       //熄灭所有 LED 指示灯
60.                                     led_on(LED_1);    //点亮 D1 指示灯
61.                                     break;
62.
63.                                 case 2:
64.                                     leds_off();       //熄灭所有 LED 指示灯
65.                                     led_on(LED_2);    //点亮 D2 指示灯
66.                                     break;
67.
68.                                 case 3:
69.                                     leds_off();      //熄灭所有 LED 指示灯
70.                                     led_on(LED_3);   //点亮 D3 指示灯
71.                                     break;
72.
73.                                 case 4:
74.                                     leds_off();      //熄灭所有 LED 指示灯
75.                                     led_on(LED_4);   //点亮 D4 指示灯
76.                                     break;
77.                                 case 5:
78.                                     leds_off();       //熄灭所有 LED 指示灯
79.                                     led_on(LED_5);    //点亮 D5 指示灯
80.                                     break;
81.
82.                                 case 6:
83.                                     leds_off();       //熄灭所有 LED 指示灯
84.                                     led_on(LED_6);    //点亮 D6 指示灯
85.                                     break;
86.
87.                                 case 7:
88.                                     leds_off();      //熄灭所有 LED 指示灯
89.                                     led_on(LED_7);   //点亮 D7 指示灯
90.                                     break;
91.
92.                                 case 8:
```

```
93.                                 leds_off();       //熄灭所有 LED 指示灯
94.                                 led_on(LED_8);    //点亮 D8 指示灯
95.                                 break;
96.
97.                             default:
98.                                 break;
99.                         }
100.                    }
101.                    uart_rx_cnt = 0;      //清零接收计数
102.                    start_flag = 0;       //清零数据包开始标志
103.                }
104.            }
105.        }
106.    }
107. }
```

5.3.3. 硬件连接

本实验需要使用 USB 转串口和 8 个 LED 指示灯，硬件连接和“实验 2-5-1：串口 1 数据收发实验”一样。

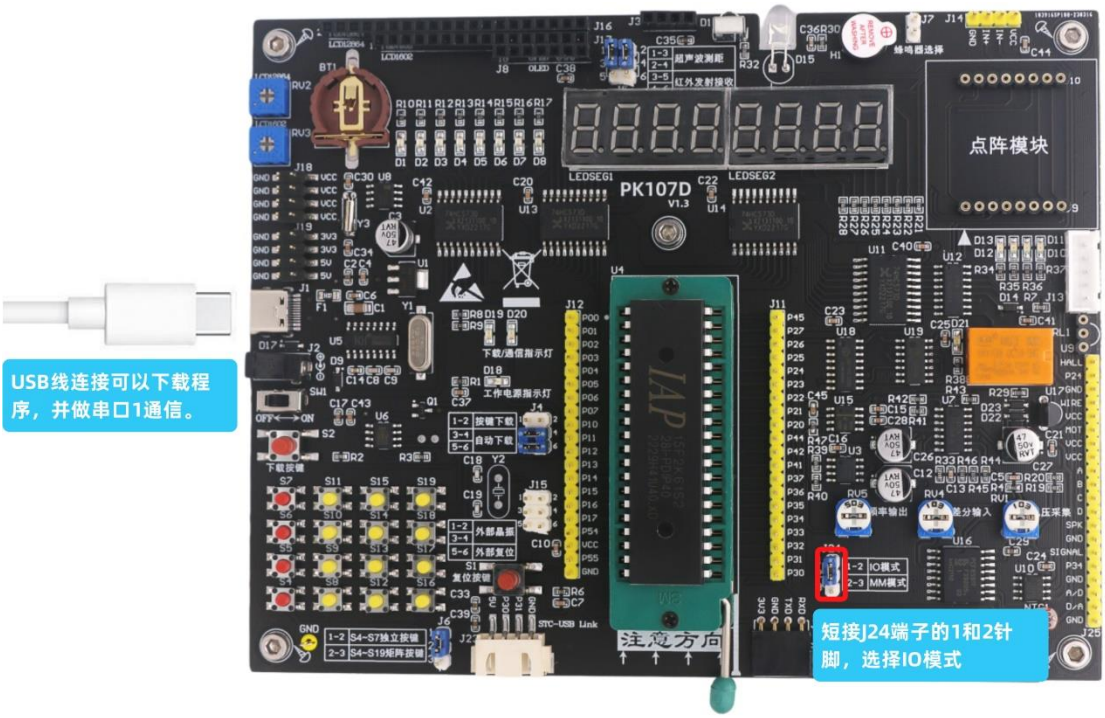


图 9：实验连接图

5.3.4. 实验步骤

1) 解压“…\第 3 部分：配套例程源码”目录下的压缩文件“实验 2-5-2：串口控制 LED 实验”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC15”（这样做的目的是

为了防止中文路径或者工程存放的路径过深导致打开工程出现问题）。

2) 双击“…\uart1_led\project”目录下的工程文件“uart1_led.uvproj”。
3) 点击编译按钮编译工程，编译成功后生成的 HEX 文件“uart1_led.hex”位于工程的“…\uart1_led\Project\Object”目录下。
4) 打开 STC-ISP 软件下载程序，下载使用内部 IRC 时钟，IRC 频率选择：12MHz。
5) 电脑上打开串口调试助手，选择开发板对应的串口号，将波特率设置为 9600bps。
6) 程序运行后，串口调试助手发送栏输入“#D1”、“#D2”、“#D3” 、“#D4”、“#D5” 、“#D6”、“#D7”或“#D8”，点击发送按钮发送数据，观察开发板上的指示灯的亮灭，应和发送的指示灯编号一致。

✧ **说明：**串口 2 的操作和串口 1 类似，读者可以尝试在学习了串口 1 的基础上编写一下串口 2 的收发程序。

我们也编写好了串口 2 的收发例子，这些例子在资料的“…\第 3 部分：配套例程源码”目录下，他们的实验名称如下，读者在编写的过程中可以参考一下。

- 实验 2-5-3：串口 2 数据收发实验。