串口 UART

1. 实验目的

- 掌握 STC15W4K32S4 系列 MCU 串行口原理。
- 掌握 4 个 UART 串行口外设相关寄存器配置及程序设计。

2. 实验内容

- 编写程序实现单个串行口收发通信的程序设计。
- 编写程序实现多个串行口收发通信的程序设计。

3. 硬件设计

3.1. 开发板串口硬件电路

进取者 STC15 开发板上设计了 USB 转 TTL 电路（CH340），其主要作用有 3 个：

1) USB 转串口通信，可用于开发板串口通信调试。
2) USB 接口有 5V 电源，可为开发板供电（计算机 USB 口可以提供 500mA 的电流）。
3) USB 转 TTL 电路连接的是 MCU 的 P3.0 和 P3.1 引脚，这样可用于开发板程序下载。

USB 转 TTL 电路如下图所示，串口接收和发送的引脚上均连接了 LED 指示灯，收发数据或程序下载时指示灯会闪烁，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通信。电路中的 500mA 自恢复保险丝用于保护开发板和计算机 USB 口。

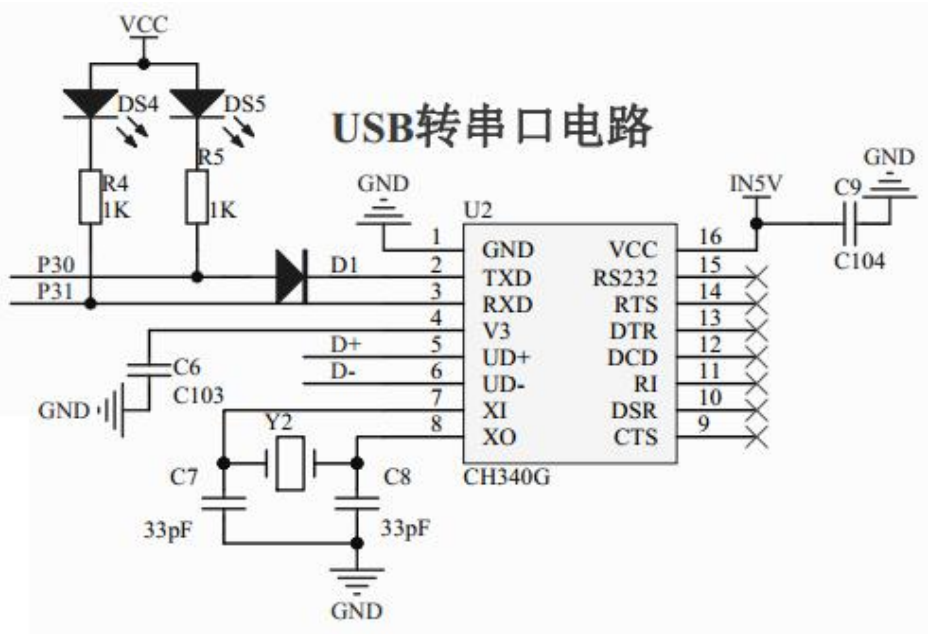

图 1：开发板 USB 转 TTL 电路

✧ USB 转 TTL 占用的单片机的引脚如下表：

表 1：串口电路引脚分配

UART	功能描述	对应 IO 口	说明
RXD	串口接收	P3.0	独立 GPIO
TXD	串口发送	P3.1	独立 GPIO

✧ 注：独立 GPIO 表示开发板没有其他的电路使用这个 GPIO。

3.2. STC15W4K32S4 系列单片机 UART 介绍

STC15W4K32S4 系列单片机有 4 个采用 UART 工作方式的全双工异步串行通信接口，每个串行口由 2 个数据缓存器、1 个移位寄存器、1 个串行控制寄存器和 1 个波特率发生器等组成。每个串行口的数据缓存器由 2 个相互独立的接收、发送缓冲器构成，因此可以同时发送和接收数据。

■ STC15W4K32S4 系列单片机的 4 个串口引脚分配：

STC15W4K32S4 系列单片机的 4 个 UART 是相互独立的，可以同时使用。但每个 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），请注意同一个 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 时必须选择一个来使用。STC15W4K32S4 系列单片机串口的引脚分配如下表。

表 2：单片机 4 个串口引脚分配

串行口	引脚名	对应 IO 口	功能描述	备注
UART1	RXD	P3.0	串口 1 接收	UART1 第一组
	TXD	P3.1	串口 1 发送	
	RXD_2	P3.6	串口 1 接收	UART1 第二组
	TXD_2	P3.7	串口 1 发送	
	RXD_3	P1.6	串口 1 接收	UART1 第三组
	TXD_3	P1.7	串口 1 发送	
UART2	RXD2	P1.0	串口 2 接收	UART2 第一组
	TXD2	P1.1	串口 2 发送	
	RXD2_2	P4.6	串口 2 接收	UART2 第二组
	TXD2_2	P4.7	串口 2 发送	
UART3	RXD3	P0.0	串口 3 接收	UART3 第一组
	TXD3	P0.1	串口 3 发送	
	RXD3_2	P5.0	串口 3 接收	UART3 第二组

	TXD3_2	P5.1	串口 3 发送	
UART4	RXD4	P0.2	串口 4 接收	UART4 第一组
	TXD4	P0.3	串口 4 发送	
	RXD4_2	P5.2	串口 4 接收	UART4 第二组
	TXD4_2	P5.3	串口 4 发送	

✧ 注：同一个串口各组之间切换是需要配置相关寄存器的相关位实现，如果没有对该部分寄存器配置，一般默认选择的都是第一组串口。

■ STC15W4K32S4 系列单片机的 4 个串口用定时器：

STC15W4K32S4 系列单片机 UART 用于波特率发生器的定时器也是可以选择的，但不是任意选择哪个定时器都可以的。针对不同 UART 可供选择的定时器如下表所示。

表 3：单片机 4 个串口波特率发生器用定时器

串行口	波特率发生器用定时器	功能描述	备注
UART1	T1	定时器 1	1、 定时器 2 是可以供 4 个串口外设同时使用。 2、 串口 2 只能使用定时器 2 作为其波特率发生器用。
	T2	定时器 2	
UART2	T2	定时器 2	
UART3	T2	定时器 2	
	T3	定时器 3	
UART4	T2	定时器 2	
	T4	定时器 4	

✧ 注：同一个串口的波特率发生器使用的定时器是需要配置相关寄存器的相关位实现，注意对寄存器按位进行操作，没有使用的位不要去配置。

3.3. 串行口 UART 工作方式

STC15W4K32S4 系列单片机 4 个 UART 均有多种工作方式，串口 1 有 4 种工作方式，其中 2 种工作方式的波特率是可变的，另 2 种工作方式的波特率是固定的，以供不同应用场合选用。串口 2、串口 3 和串口 4 都只用 2 种工作方式，这 2 种工作方式的波特率都是可变的。下面列表 4 个 UART 的工作方式。

表 4：单片机 4 个串口工作方式

串行口	工作方式	功能描述	备注
UART1	方式 0	同步移位串行方式：移位寄存器	不建议学习
	方式 1	8 位 UART，波特率可变	推荐学习
	方式 2	8 位 UART	不建议学习

	方式 3	9 位 UART，波特率可变	可以学习
UART2	方式 0	8 位 UART，波特率可变	推荐学习
	方式 1	9 位 UART，波特率可变	可以学习
UART3	方式 0	8 位 UART，波特率可变	推荐学习
	方式 1	9 位 UART，波特率可变	可以学习
UART4	方式 0	8 位 UART，波特率可变	推荐学习
	方式 1	9 位 UART，波特率可变	可以学习

✧ 注：艾克姆提供例程是按照“8 位 UART，波特率可变”方式进行配置。

■ UART1 工作方式 1 原理介绍：

“8 位 UART，波特率可变”的工作方式，其一帧信息为 10 位：1 位起始位+8 位数据位（低位在先）+1 位停止位。

发送过程：串行通信模式发送时，数据由串行发送端 TXD 输出。当主机执行一条写 SBUF 的指令就启动串行通信的发送，写“SBUF”信号还把“1”装入发送移位寄存器的第 9 位，并通知 TX 控制单元开始发送。移位寄存器将数据不断右移送 TXD 端口发送，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”，在其左边各位全为“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。

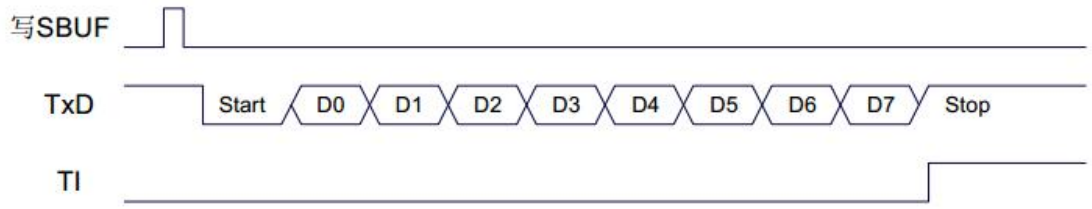

图 2：UART1 工作方式 1 发送数据示意图

接收过程：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，完后一帧信息的接收。

接收数据有效需同时满足以下两个条件：

1）RI = 0；

2）SM2 = 0 或接收到的停止位为 1。

✧ 注：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，接收器重新检测 RXD 端口上的“1”→“0”的跳变，继续下一帧信息的接收。

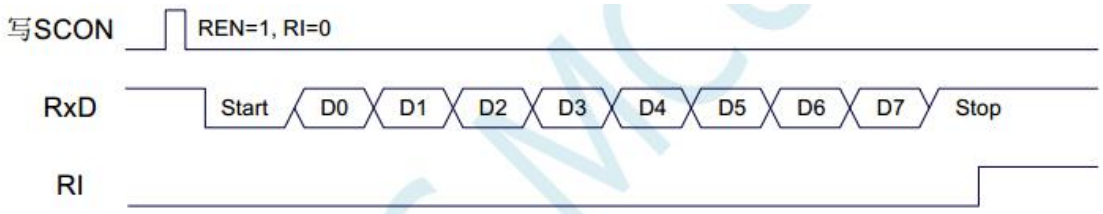

图 3：UART1 工作方式 1 接收数据示意图

下面举例介绍下串口 1 在进行工作方式选择时，需要配置的是串行控制寄存器 SCON(也可称为串口 1 控制寄存器)。该寄存器支持位寻址，该寄存器的 B6 和 B7 位便是用来选择串口 1 工作方式的，寄存器的 B5、B3 和 B2 位是 9 位 UART 时需要配置的位，寄存器的 B4 位是串行接收控制位，寄存器的 B0、B1 位是串口接收和发送中断请求标志位。

1 寄存器名

SCON：串行控制寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	name	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI

SM0/FE：当PCON寄存器中的SMOD0/PCON.6位为1时，该位用于帧错误检测。当检测到一个无效停止位时，通过UART接收器设置该位。它必须由软件清零。

当PCON寄存器中的SMOD0/PCON.6位为0时，该位和SM1一起指定串行通信的工作方式，如下表所示。

2 工作方式选择位

SM0	SM1	工作方式	功能说明	波特率
0	0	方式0	同步移位串行方式：移位寄存器	当UART_M0x6 = 0时，波特率是SYSclk/12, 当UART_M0x6 = 1时，波特率是SYSclk / 2
0	1	方式1	8位UART，波特率可变	串行口1用定时器1作为其波特率发生器且定时器1工作于模式0(16位自动重装载模式)或串行口用定时器2作为其波特率发生器时，波特率=(定时器1的溢出率或定时器T2的溢出率)/4. 注意：此时波特率与SMOD无关。 当串行口1用定时器1作为其波特率发生器且定时器1工作于模式2(8位自动重装模式)时，波特率=(2^{SMOD}/32)×(定时器1的溢出率)
1	0	方式2	9位UART	(2^{SMOD} / 64) x SYSclk系统工作时钟频率
1	1	方式3	9位UART，波特率可变	当串行口1用定时器1作为其波特率发生器且定时器1工作于模式0(16位自动重装载模式)或串行口用定时器2作为其波特率发生器时，波特率=(定时器1的溢出率或定时器T2的溢出率)/4. 注意：此时波特率与SMOD无关。 当串行口1用定时器1作为其波特率发生器且定时器1工作于模式2(8位自动重装模式)时，波特率=(2^{SMOD}/32)×(定时器1的溢出率)

3 方式2和方式3时有用

SM2：允许方式2或方式3多机通信控制位。

在方式2或方式3时，如果SM2位为1且REN位为1，则接收机处于地址帧筛选状态。此时可以利用接收到的第9位(即RB8)来筛选地址帧：若RB8=1，说明该帧是地址帧，地址信息可以进入SBUF，并使RI为1，进而在中断服务程序中再进行地址号比较；若RB8=0，说明该帧不是地址帧，应丢掉且保持RI=0。在方式2或方式3中，如果SM2位为0且REN位为1，接收收机处于地址帧筛选被禁止状态。不论收到的RB8为0或1，均可使接收到的信息进入SBUF, 并使RI=1, 此时RB8通常为校验位.

方式1和方式0是非多机通信方式，在这两种方式时，要设置SM2 应为0。

4 串行接收控制位

REN：允许/禁止串行接收控制位。由软件置位REN，即REN=1为允许串行接收状态，可启动串行接收器RxD，开始接收信息。软件复位REN，即REN=0，则禁止接收。

5 方式2和方式3时有用

TB8：在方式2或方式3，它为要发送的第9位数据，按需要由软件置位或清0。例如，可用作数据的校验位或多机通信中表示地址帧/数据帧的标志位。在方式0和方式1中，该位不用.

RB8：在方式2或方式3，是接收到的第9位数据，作为奇偶校验位或地址帧/数据帧的标志位。方式0中不用RB8(置SM2=0). 方式1中也不用RB8(置SM2=0, RB8是接收到的停止位)。

6 发送中断请求标志位

TI：发送中断请求标志位。在方式0，当串行发送数据第8位结束时，由内部硬件自动置位，即TI=1，向主机请求中断，响应中断后TI必须用软件清零，即TI=0。在其他方式中，则在停止位开始发送时由内部硬件置位，即TI=1, 响应中断后TI必须用软件清零。

7 接收中断请求标志位

RI：接收中断请求标志位。在方式0，当串行接收到第8位结束时由内部硬件自动置位RI=1，向主机请求中断，响应中断后RI必须用软件清零，即RI=0。在其他方式中，串行接收到停止位的中间时刻由内部硬件置位，即RI=1, 向CPU发中断申请，响应中断后RI必须由软件清零。

图 4：串行控制寄存器 SCON

✧ 注：串行控制寄存器 SCON 的 B2、B3 和 B5 位，在 UART1 工作方式 1 时配置为 0。

3.4. 串行口使用引脚切换选择

在使用 STC15W4K32S4 系列单片机的 4 个 UART 时，需要确定使用串口的哪一组引脚。这需要通过操作 P_SW1 或 P_SW2 等寄存器实现。

STC15W4K32S4 系列单片机串口 1 有 3 组串口引脚可供选择，实现引脚切换选择需要 P_SW1 寄存器（即外围设备功能切换控制寄存器 1）的 B6 和 B7 位，如下图所示。

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	0000,0000

串口1的切换选择

串口1/S1可在3个地方切换，由 S1_S0 及 S1_S1 控制位来选择		
S1_S1	S1_S0	串口1/S1可在P1/P3之间来回切换
0	0	串口1/S1在[P3.0/RxD, P3.1/TxD]
0	1	串口1/S1在[P3.6/RxD_2, P3.7/TxD_2]
1	0	串口1/S1在[P1.6/RxD_3/XTAL2, P1.7/TxD_3/XTAL1] 串口1在P1口时要使用内部时钟
1	1	无效

图 5：P_SW1 外围设备功能切换控制寄存器 1

✧ 注：这里对外围设备功能切换控制寄存器 1 命名时有 2 个名字：AUXR1 和 P_SW1，这 2 个名字使用时使用任意一个即可。

STC15W4K32S4 系列单片机串口 2、串口 3 和串口 4 均有 2 组串口引脚可供选择，实现引脚切换选择需要配置外围设备功能切换控制寄存器 2 的 B0、B1 和 B2 位，如下图所示。

P_SW2：外围设备功能切换控制寄存器2（不可位寻址）

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
P_SW2	BAH	外围设备功能切换控制寄存器2						S4_S	S3_S	S2_S	xxxx,x000

串口2/S2可在2个地方切换，由 S2_S 控制位来选择	
S2_S	S2可在P1/P4之间来回切换
0	串口2/S2在[P1.0/RxD2, P1.1/TxD2]
1	串口2/S2在[P4.6/RxD2_2, P4.7/TxD2_2]

串口2使用引脚选择

串口3/S3可在2个地方切换，由 S3_S 控制位来选择	
S3_S	S3可在P0/P5之间来回切换
0	串口3/S3在[P0.0/RxD3, P0.1/TxD3]
1	串口3/S3在[P5.0/RxD3_2, P5.1/TxD3_2]

串口3使用引脚选择

串口4/S4可在2个地方切换，由 S4_S 控制位来选择	
S4_S	S4可在P0/P5之间来回切换
0	串口4/S4在[P0.2/RxD4, P0.3/TxD4]
1	串口4/S4在[P5.2/RxD4_2, P5.3/TxD4_2]

串口4使用引脚选择

图 6：P_SW2 外围设备功能切换控制寄存器 2

✧ 注：因为串口 2、串口 3 和串口 4 只有 2 组串口引脚供选择，所以寄存器使用 1 位即可控制切换。

3.5. 串行口 1 工作方式 1 波特率计算公式

STC15W4K32S4 系列单片机 4 个 UART 在不同的工作方式下，选择不同的定时器作为波特率发生器时，波特率计算公式都是不同的。下面举例给出 UART1 工作方式 1 时的波特率计算公式。

表 5：UART1 工作方式 1 波特率计算

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

✧ 注：SYSclk 为系统工作频率，SMOD 是 PCON 寄存器最高位（用于波特率加倍选择），定时器 1 模式 0 为 16 位自动重装载模式，定时器 1 模式 2 为 8 位自动重装载模式（详见定时器部分介绍）。

举例，系统时钟频率为 11.0592MHZ，配置定时器 1 为 1T，工作模式为模式 2， PCON 寄存器 SMOD 位置为 0，波特率预设置为 9600bps，计算下定时器 1 重装载值。

1） SMOD = 0，则 2^{SMOD} * SYSclk =11059200。

2） 11059200/(32*9600) =36。

3） 256 – 36 = 220。十进制 220 对应十六进制是 DC。

4） 所以对定时器 1 的高 8 位寄存器初始装载值和低 8 位寄存器初始装载值赋值 0xDC。

5） 如果已知定时器重装载值，计算串口波特率，则是反推过来即可（建议使用软件 STC-ISP 的波特率计算器）。

3.6. 串行口中断配置步骤

针对 STC15W4K32S4 系列单片机 4 个串行口外设，软件的配置过程如下：

图 7：串行口中断软件配置步骤

✧ 注：实验例程即是按照上述配置步骤操作寄存器相关位实现，后有详述。

4. 软件设计

4.1. 串行口寄存器汇集

STC15W4K32S4 系列单片机操作串行口时会用到 18 个寄存器，如下表所示：

表 6：STC15W4K32S4 系列串行口使用寄存器汇总

序号	寄存器名	读/写	功能描述
1	AUXR	读/写	辅助寄存器。
2	PCON	读/写	电源控制寄存器。
3	P_SW1	读/写	外围设备功能切换控制寄存器 1。
4	P_SW2	读/写	外围设备功能切换控制寄存器 2。
5	SCON	读/写	串口 1 控制寄存器。
6	SBUF	读/写	串口 1 数据寄存器。
7	S2CON	读/写	串口 2 控制寄存器。
8	S2BUF	读/写	串口 2 数据寄存器。

9	S3CON	读/写	串口 3 控制寄存器。
10	S3BUF	读/写	串口 3 数据寄存器。
11	S4CON	读/写	串口 4 控制寄存器。
12	S4BUF	读/写	串口 4 数据寄存器。
13	SADDR	读/写	串口 1 从机地址寄存器。
14	SADEN	读/写	串口 1 从机地址屏蔽寄存器。
15	IE	读/写	中断允许寄存器。
16	IE2	读/写	中断允许寄存器 2。
17	IP	读/写	中断优先级控制寄存器。
18	IP2	读/写	中断优先级控制寄存器 2。

✧ 注：串口波特率发生器需要用到定时器，定时器相关的寄存器没有在上述表格中列举。

4.2. 寄存器解析

4.2.1. 中断允许寄存器 IE

外部中断允许寄存器 IE 支持位寻址，该寄存器的 B4 位是串口 1 的中断允许位。

图 8：中断允许寄存器

4.2.2. 中断允许寄存器 IE2

中断允许寄存器 IE2 不支持位寻址，该寄存器的 B0、B3 和 B4 位是串口 2、串口 3 和

串口 4 的中断允许位。因为 IE2 寄存器不支持位寻址，所以举例操作该寄存器 B0 位时，不可以直接“ES2=0;”进行操作，参考下图。

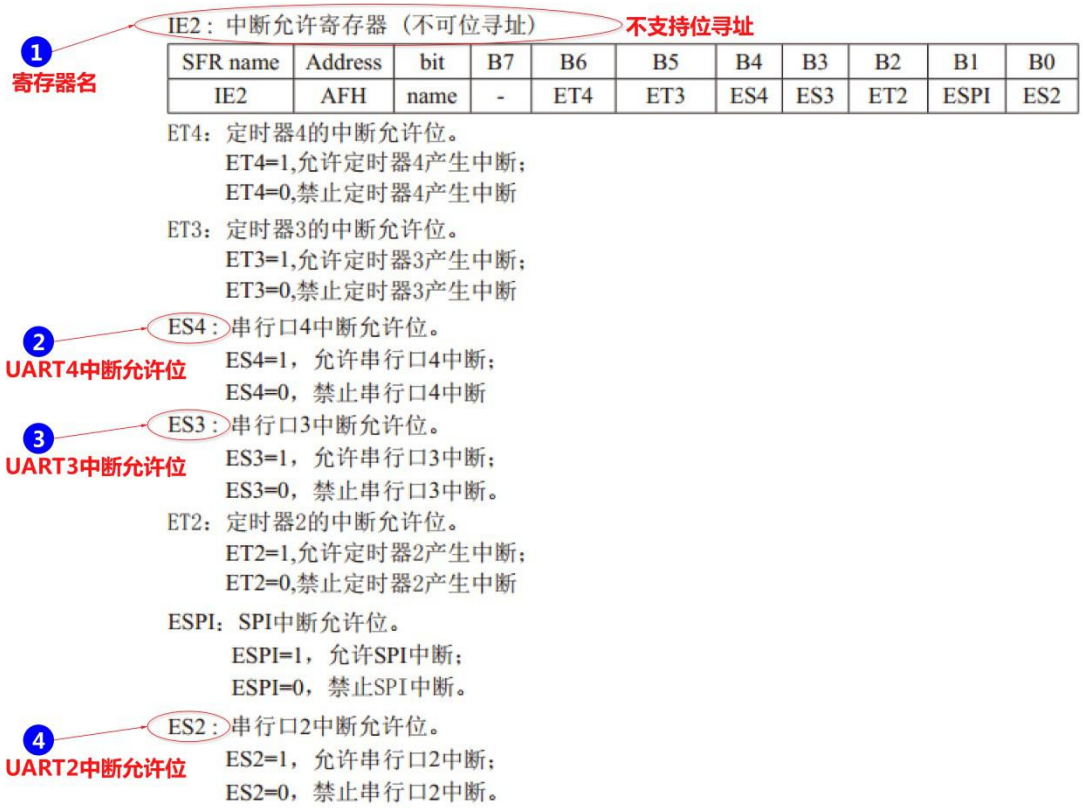

图 9：中断允许寄存器 2

4.2.3. 电源控制寄存器 PCON

电源控制寄存器 PCON 不支持位寻址，该寄存器的 B6 位和 B7 位是 UART1 的帧错误检测有效控制位和波特率选择位，具体含义如下图。

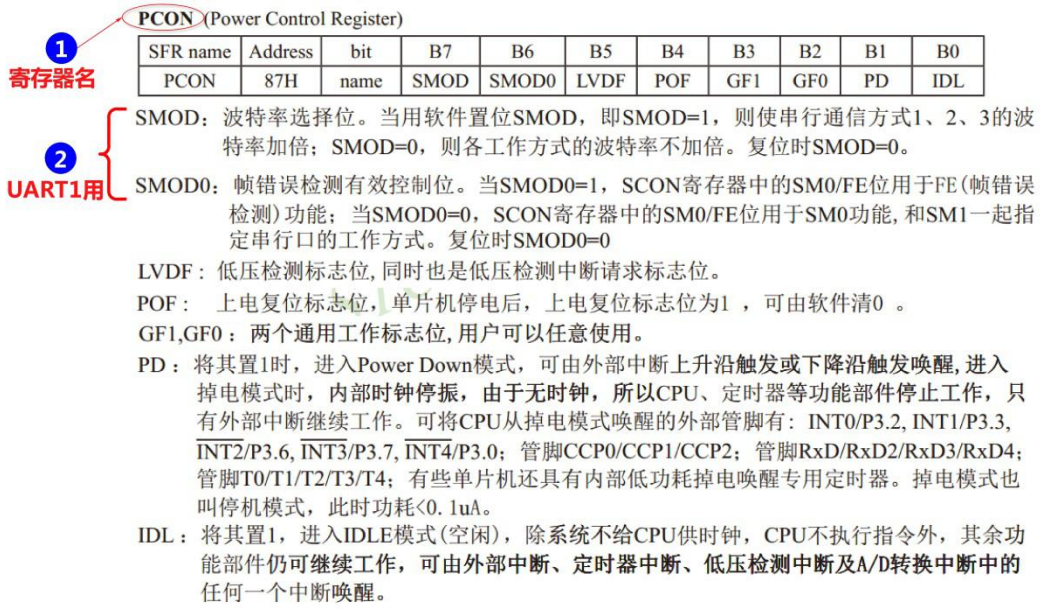

图 10：电源控制寄存器

✧ 注：PCON 寄存器的 SMOD 位和 SMOD0 位是用于串口 1 的，换句话说，串口 2、串口 3 和串口 4 没有与之对应的控制位。

4.2.4. 辅助寄存器 AUXR

辅助寄存器 AUXR 不支持位寻址，该寄存器的 B0 位是串口 1 的波特率发生器定时器选择控制位，寄存器的 B5 位是串口 1 模式 0 的通信速度设置位。

图 11：辅助寄存器

✧ 注：AUXR 寄存器的其他位用于定时器配置，操作串口时也会有对定时器的配置部分，请注意按位操作。

4.2.5. 中断优先级控制寄存器 IP

中断优先级控制寄存器 IP 支持位寻址，该寄存器的 B4 位是串口 1 中断优先级控制位。

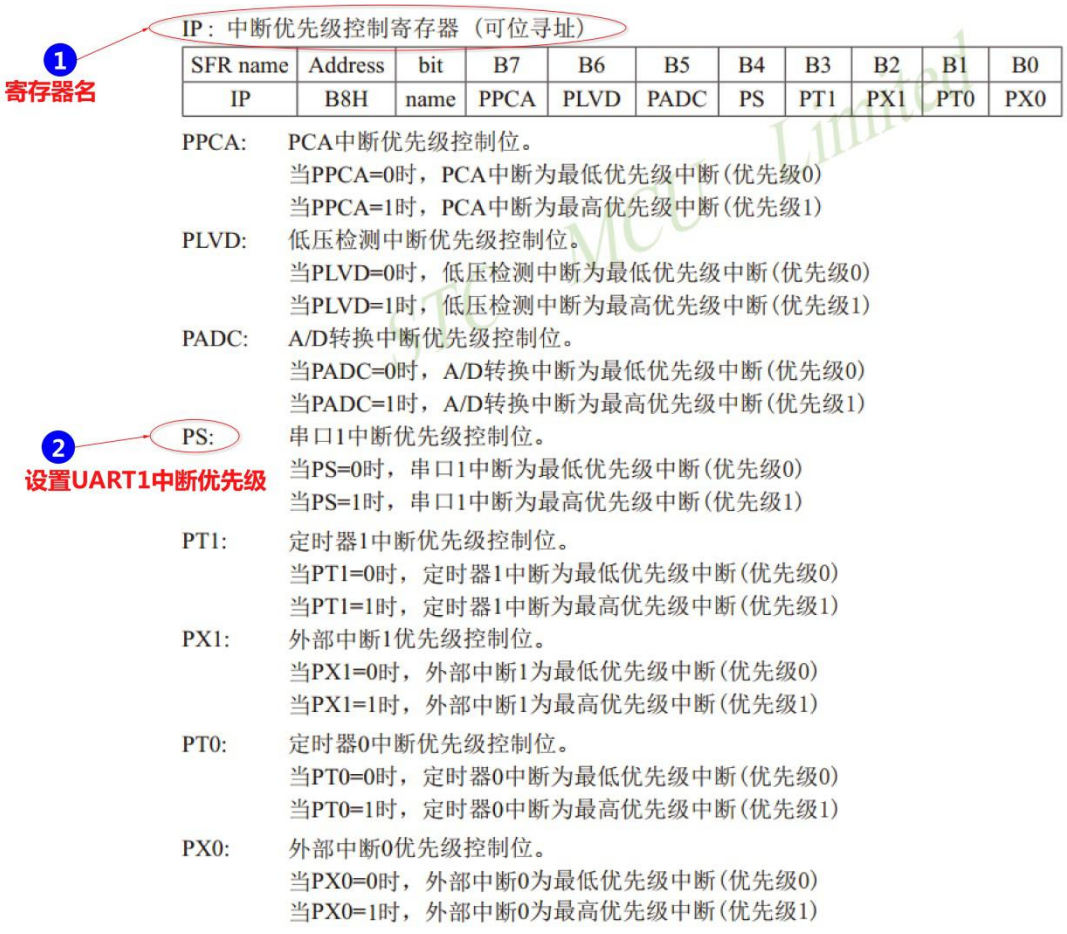

图 12：中断优先级控制寄存器

✧ 注：艾克姆例程没有对串口 1 中断优先级进行配置，可根据项目需要配置 PS 位。

4.2.6. 中断优先级控制寄存器 IP2

中断优先级控制寄存器 2 不支持位寻址，该寄存器的 B0 位是串口 2 中断优先级控制位。

图 13：中断优先级控制寄存器 2

4.2.7. 串行口 2 控制寄存器 S2CON

串行口 2 控制寄存器 S2CON 不支持位寻址，该寄存器的 B7 位是用来选择串口 2 工作方式的，寄存器的 B5、B3 和 B2 位是 9 位 UART 时需要配置的位，寄存器的 B4 位是串行接收控制位，寄存器的 B0、B1 位是串口接收和发送中断请求标志位。

S2CON：串行口2控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S2CON	9AH	name	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI

S2SM0：指定串行口2的工作方式，如下表所示。

S2SM0	工作方式	功能说明	波特率
0	方式0	8位UART，波特率可变	(定时器T2的溢出率)/4
1	方式1	9位UART，波特率可变	(定时器T2的溢出率)/4

当AUXR.2/T2x12=1时，定时器T2的溢出率 = SYSclk / (65536 - [RL_TH2,RL_TL2])
当AUXR.2/T2x12=0时，定时器T2的溢出率 = SYSclk / 12 / (65536 - [RL_TH2,RL_TL2])
式中RL_TH2是T2H的重装载寄存器，RL_TL2是T2L的重装载寄存器。

B6：保留，该位复位后为1.

S2SM2：允许方式1多机通信控制位。
在方式1时，如果S2SM2位为1且S2REN位为1，则接收机处于地址帧筛选状态。此时可以利用接收到的第9位(即S2RB8)来筛选地址帧：若S2RB8=1，说明该帧是地址帧，地址信息可以进入S2BUF，并使S2RI为1，进而在中断服务程序中再进行地址号比较；若S2RB8=0，说明该帧不是地址帧，应丢掉且保持S2RI=0。在方式1中，如果S2SM2位为0且S2REN位为1，接收收机处于地址帧筛选被禁止状态。不论收到的S2RB8为0或1，均可使接收到的信息进入S2BUF，并使S2RI=1，此时S2RB8通常为校验位.
方式0是非多机通信方式，在这种方式时，要设置S2SM2 应为0。

S2REN：允许/禁止串行口2接收控制位。由软件置位S2REN，即S2REN=1为允许串行接收状态，可启动串行接收器RxD2，开始接收信息。软件复位S2REN，即S2REN=0，则禁止接收。

S2TB8：在方式1，S2TB8为要发送的第9位数据，按需要由软件置位或清0。例如，可用作数据的校验位或多机通信中表示地址帧/数据帧的标志位。在方式0中，该位不用.

S2RB8：在方式1，S2RB8是接收到的第9位数据，作为奇偶校验位或地址帧/数据帧的标志位。方式0中不用S2RB8(置S2SM2=0，S2RB8是接收到的停止位)。

S2TI：发送中断请求标志位。在停止位开始发送时由S2TI内部硬件置位，即S2TI=1，响应中断后S2TI必须用软件清零。

S2RI：接收中断请求标志位。在串行接收到停止位的中间时刻S2RI由内部硬件置位，即S2RI=1，向CPU发中断申请，响应中断后S2RI必须由软件清零。

图 14：串行口 2 控制寄存器

4.2.8. 串行口 3 控制寄存器 S3CON

串行口 3 控制寄存器 S3CON 不支持位寻址，该寄存器的 B7 位是用来选择串口 3 工作方式的，寄存器的 B5、B3 和 B2 位是 9 位 UART 时需要配置的位，寄存器的 B4 位是串行接收控制位，寄存器的 B0、B1 位是串口接收和发送中断请求标志位。

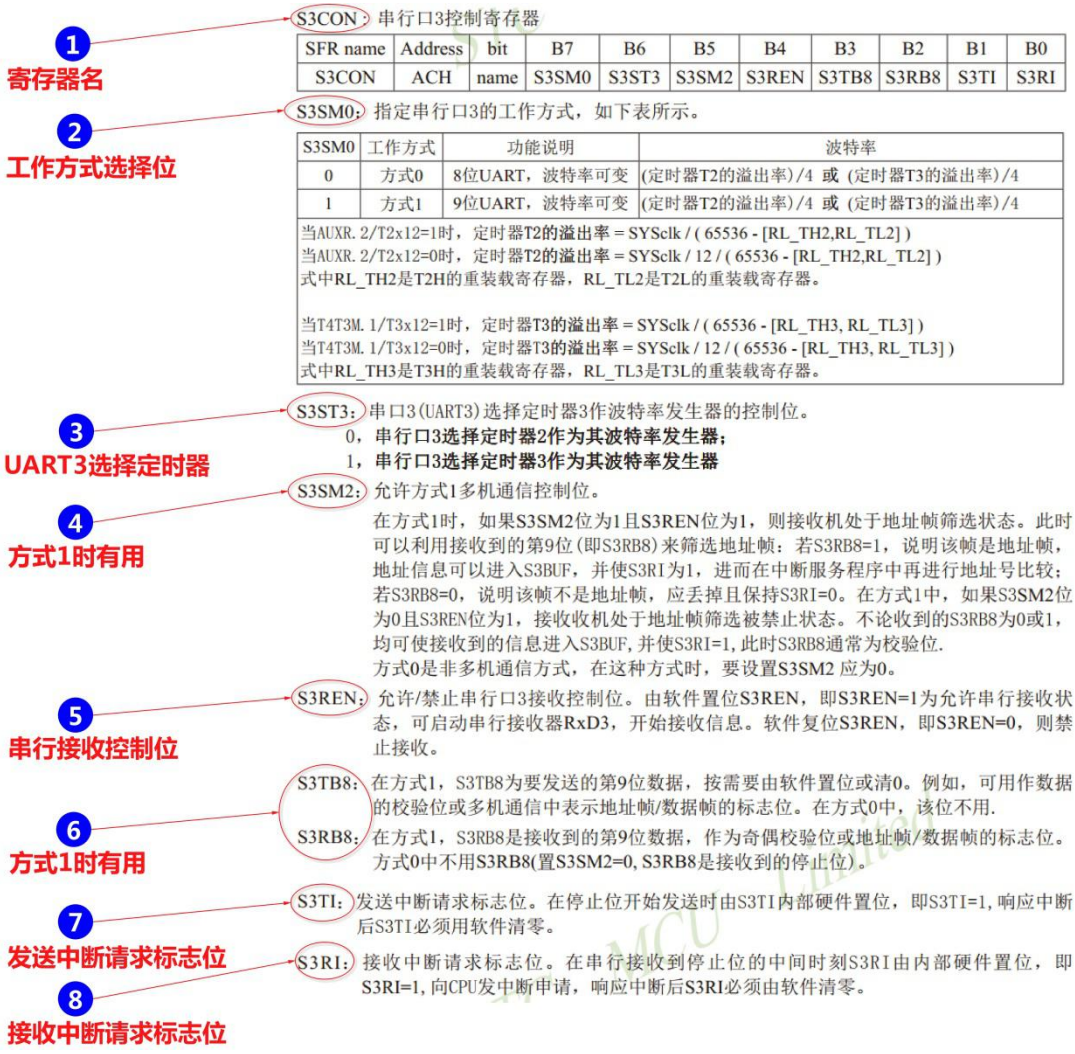

图 15：串行口 3 控制寄存器

4.2.9. 串行口 4 控制寄存器 S4CON

串行口 4 控制寄存器 S4CON 不支持位寻址，该寄存器的 B7 位是用来选择串口 4 工作方式的，寄存器的 B5、B3 和 B2 位是 9 位 UART 时需要配置的位，寄存器的 B4 位是串行接收控制位，寄存器的 B0、B1 位是串口接收和发送中断请求标志位。

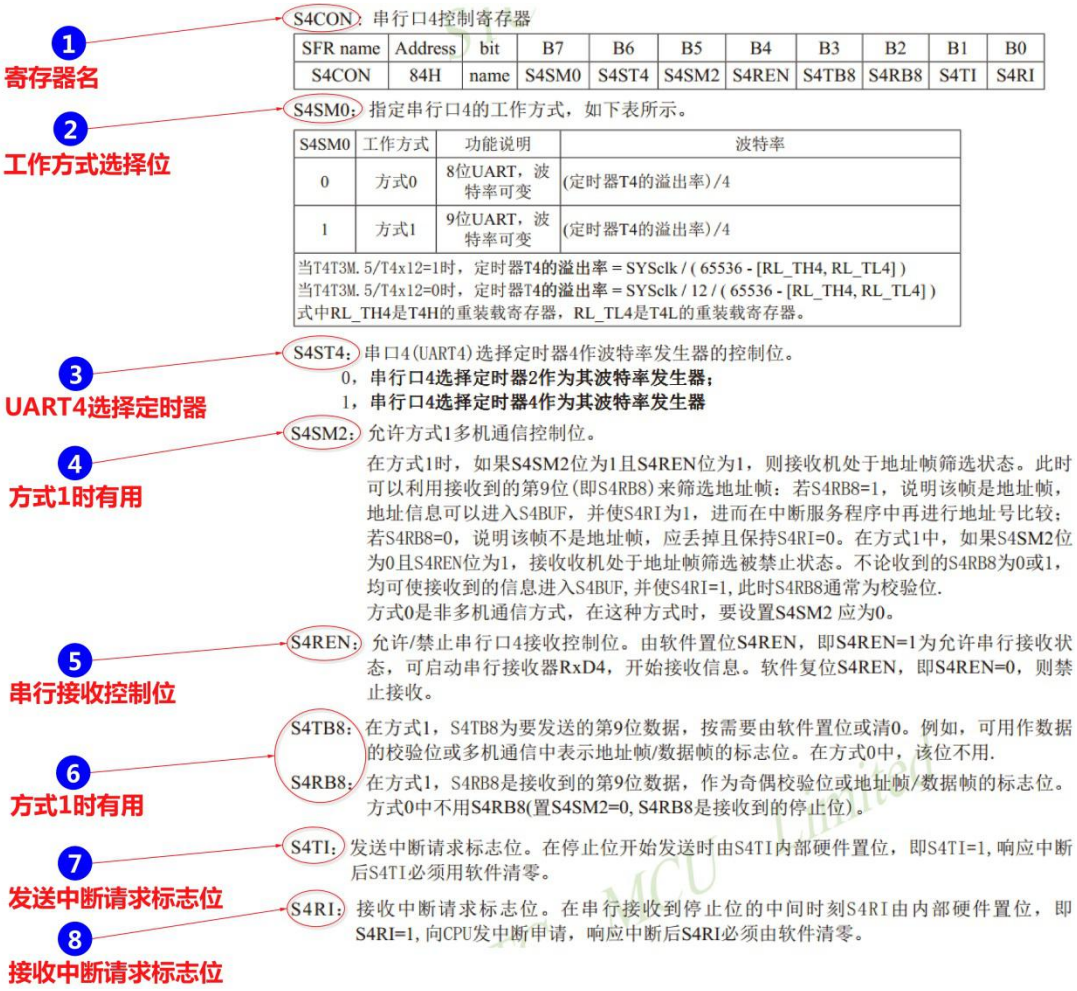

图 16：串行口 4 控制寄存器

4.3. 串口 1 收发实验（P3.0 和 P3.1）

✧ 注：本节的实验源码是在“实验 2-5-1：定时器 0 定时”的基础上修改。本节对应的实验源码是：“实验 2-8-1：串口 1 收发实验（P3.0 和 P3.1）”。

4.3.1. 工程需要用到的 c 文件

本例需要用到的 c 文件如下表所示，工程需要添加下表中的 c 文件。

表 7：实验需要用到的 c 文件

序号	文件名	后缀	功能描述
1	uart	.c	外部串行口有关的用户自定义函数。
2	delay	.c	包含用户自定义延时函数。

4.3.2. 头文件引用和路径设置

■ 需要引用的头文件

```
1. #include "delay.h"
2. #include "uart.h"
```

■ 需要包含的头文件路径

本例需要包含的头文件路径如下表：

表 8：头文件包含路径

序号	路径	描述
1	..\ Source	uart.h 和 delay.h 头文件在该路径，所以要包含。
2	..\User	15W4KxxS4.h 头文件在该路径，所以要包含。

MDK 中点击魔术棒，打开工程配置窗口，按照下图所示添加头文件包含路径。

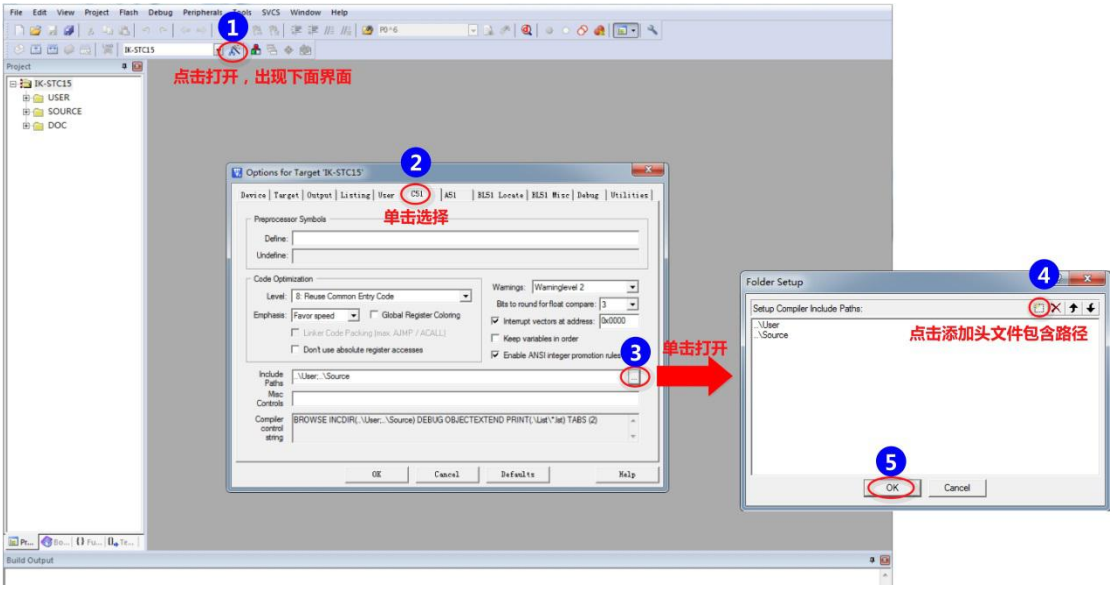

图 17：添加头文件包含路径

4.3.3. 编写代码

首先，在 uart.c 文件中编写串口 1 的初始化函数 Uart1_Init，代码如下。

程序清单：串口 1 初始化函数

```
1. /*********************************************************************************
2.  * 描  述 : 串口 1 初始化函数
3.  * 入  参 : 无
4.  * 返回值 : 无
5. 备注：波特率 9600bps   晶振 11.0592MHz
6.  *********************************************************************************/
7. void Uart1_Init(void)
```

```
{
    PCON &= 0x3f;          //波特率不倍速，串行口工作方式由 SM0、SM1 决定
    SCON = 0x50;           //8 位数据,可变波特率，启动串行接收器
    AUXR |= 0x40;          //定时器 1 时钟为 Fosc,即 1T
    AUXR &= 0xfe;          //串口 1 选择定时器 1 为波特率发生器
    TMOD &= 0x0f;          //清除定时器 1 模式位
    TMOD |= 0x20;          //设定定时器 1 为 8 位自动重装方式
    TL1 = 0xDC;            //设定定时初值
    TH1 = 0xDC;            //设定定时器重装值
    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
}
```

然后，编写串口 1 发送数据函数，把要发送的字节存放于数据缓存寄存器中，直到数据发送完成，代码如下。

程序清单：数据发送函数函数

```
/*----------------------------------------------------------------------------
 * 描   述 : 串口 1 发送数据函数
 * 入   参 : uint8 数据
 * 返回值 : 无
 ----------------------------------------------------------------------------*/
void SendDataByUart1(uint8 dat)
{
    SBUF = dat;                 //写数据到 UART 数据寄存器
    while(TI == 0);             //在停止位没有发送时，TI 为 0 即一直等待
    TI = 0;                     //清除 TI 位（该位必须软件清零）
}
```

之后，编写串口 1 的中断服务函数，将接收的数据存放到用户自定义变量 uart1temp 中，代码如下。

程序清单：中断服务函数

```
/*----------------------------------------------------------------------------
 * 描   述 : 串口 1 中断服务函数
 * 入   参 : 无
 * 返回值 : 无
 ----------------------------------------------------------------------------*/
void Uart1() interrupt UART1_VECTOR using 1
{
    ES = 0;                       // 串口 1 中断关闭
    Flag=TRUE;                    //接收到数据,接收标识符有效
```

```
    if (RI)                          //串行接收到停止位的中间时刻时，该位置 1
  {
      RI = 0;                  //清除 RI 位 （该位必须软件清零）
      uart1temp = SBUF;
   }
   if (TI)                     //在停止位开始发送时，该位置 1
   {
      TI = 0;                  //清除 TI 位（该位必须软件清零）
   }
     ES =  1;                    // 串口 1 中断打开
}
```

最后，用户定义一个自定义函数 UART1_Tx_Puts，该函数将接收的数据原样返回去并加上回车符。主函数 main 在主循环中调用该函数。具体代码如下。

代码清单：用户函数 UART1_Tx_Puts

```
/*******************************************************************************
 * 描   述 : 串口 1 接收到数据后发送回去
 * 入   参 : 无
 * 返回值 : 无
 *******************************************************************************/
void UART1_Tx_Puts(void)
{
    if(Flag)                                //有新数据通过串口被接收到
    {
        ES = 0;                             //串口 1 中断关闭
       SendDataByUart1(uart1temp);          //发送字符
       SendDataByUart1(0x0D);               //发送换行符
       SendDataByUart1(0x0A);               //发送换行符
        ES = 1;                             //串口 1 中断打开
        Flag=FALSE;                         //清除接收标识符
    }
}
```

代码清单：主函数

```
int main()
{
//////////////////////////////////////////////
//注意: STC15W4K32S4 系列的芯片,上电后所有与 PWM 相关的 IO 口均为
//       高阻态,需将这些口设置为准双向口或强推挽模式方可正常使用
```

```
//相关 IO: P0.6/P0.7/P1.6/P1.7/P2.1/P2.2
//         P2.3/P2.7/P3.7/P4.2/P4.4/P4.5
////////////////////////////////////////////////
    P1M1 &= 0x3F;   P1M0 &= 0x3F;     //设置 P1.6~P1.7 为准双向口

    Uart1_Init();          //串口 1 初始化
    EA = 1;                //总中断打开

    while(1)
    {
        UART1_Tx_Puts();   //串口接收到一个字符后返回该字符
    }
}
```

4.3.4. 硬件连接

本实验需要将 USB 线连接至 PC，因为开发板板载 CH340 电路连接的就是单片机 P3.0 和 P3.1 口，所以无需外接 USB 转 TTL 模块做该实验。

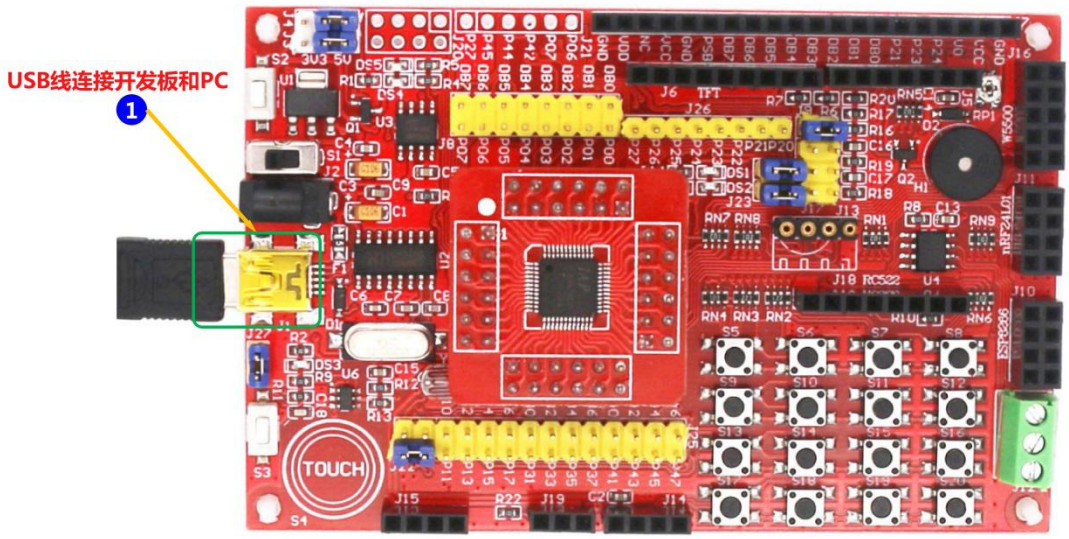

图 18：串口 1 实验连接图

4.3.5. 实验步骤

1. 解压“…\第 3 部分：配套例程源码\ 1 - 基础实验程序\”目录下的压缩文件“实验 2-8-1：串口 1 收发实验（P3.0 和 P3.1）”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC15”。
2. 启动 Keil C51。
3. 在 Keil C51 中执行“Project→Open Project”打开“…\UART1\projec”目录下的工程“UART1.uvproj”。

4. 点击编译按钮编译工程。注意查看编译输出栏，观察编译的结果，如果有错误，修改程序，直到编译成功为止。编译后生成的 HEX 文件“UART1.hex”位于工程目录下的“Output”文件夹中。
5. 打开 STC-ISP 软件下载程序 。下载使用内部 IRC 时钟，IRC 频率选择为 11.0592MHZ。
6. 程序运行后，打开串口调试助手，波特率设置为 9600，在发送区发送什么字符在接收区就会收到什么字符（注意是字符不是字符串）。

4.4. 串口 2 收发实验（P4.6 和 P4.7）

✧ 注：本节的实验源码是在“实验 2-8-1：串口 1 收发实验（P3.0 和 P3.1）”的基础上修改。本节对应的实验源码是：“实验 2-8-2：串口 2 收发实验（P4.6 和 P4.7）”。

4.4.1. 工程需要用到的 c 文件

本实验需要用到的头文件以及添加头文件包含路径的方法请参考“实验 2-8-1：串口 1 收发实验（P3.0 和 P3.1）”部分。

4.4.2. 编写代码

首先，在 uart.c 文件中编写串口 2 的初始化函数 Uart2_Init，代码如下。

程序清单：串口 2 初始化函数

```
/*******************************************************************************
 * 描   述 : 串口 2 初始化函数
 * 入   参 : 无
 * 返回值 : 无
备注：波特率 9600bps    晶振 11.0592MHz
 *******************************************************************************/
void Uart2_Init(void)
{
    P_SW2|=S2_S;    //选择 P46 P47 为串口 2

    S2CON = 0x50;        //8 位数据,可变波特率，启动串行接收器
    AUXR |= 0x04;        //定时器 2 时钟为 Fosc,即 1T
    T2L = 0xE0;        //设定定时初值
    T2H = 0xFE;      //设定定时初值
    AUXR |= 0x10;    //启动定时器 2
    IE2 |= 0x01;     //串口 2 中断打开
}
```

然后，编写串口 2 发送数据函数，把要发送的字节存放于数据缓存寄存器中，直到数据发送完成，代码如下。

程序清单：数据发送函数函数

```
/*******************************************************************************
```

```
2.   * 描   述 : 串口 2 发送数据函数
3.   * 入   参 : uint8 数据
4.   * 返回值 : 无
5.   ********************************************************************************/
6.  void SendDataByUart2(uint8 dat)
7.  {
8.      S2BUF = dat;                  //写数据到 UART 数据寄存器
9.      while(!(S2CON&S2TI));         //在停止位没有发送时，S2TI 为 0 即一直等待
10.     S2CON&=~S2TI;                 //清除 S2CON 寄存器对应 S2TI 位（该位必须软件清零）
11. }
```

之后，编写串口 2 的中断服务函数，将接收的数据存放到用户自定义变量 uart2temp 中，代码如下。

程序清单：中断服务函数

```
1.  /********************************************************************************
2.   * 描   述 : 串口 2 中断服务函数
3.   * 入   参 : 无
4.   * 返回值 : 无
5.   ********************************************************************************/
6.  void Uart2() interrupt UART2_VECTOR using 1
7.  {
8.      IE2 &= 0xFE;                          // 串口 2 中断关闭
9.      Flag=TRUE;                          //接收到数据,接收标识符有效
10.     if (S2CON & S2RI)                   //串行接收到停止位的中间时刻时，该位置 1
11.     {
12.       S2CON &= ~S2RI;                   //清除 S2CON 寄存器对应 S2RI 位(该位必须软件清零)
13.       uart2temp = S2BUF;
14.     }
15.     if (S2CON & S2TI)                   //在停止位开始发送时，该位置 1
16.     {
17.       S2CON &= ~S2TI;                  //清除 S2CON 寄存器对应 S2TI 位(该位必须软件清零)
18.      }
19.      IE2 |= 0x01;                       // 串口 2 中断打开
20. }
```

最后，用户定义一个自定义函数 UART2_Tx_Puts，该函数将接收的数据原样返回去并加上回车符。主函数 main 在主循环中调用该函数。具体代码如下。

代码清单：用户函数 UART2_Tx_Puts

```
1.  /********************************************************************************
2.   * 描   述 : 串口 2 接收到数据后发送出去
```

```
3.   * 入  参 : 无
4.   * 返回值 : 无
5.   ******************************************************************************/
6.  void UART2_Tx_Puts(void)
7.  {
8.    if(Flag)       //有新数据通过串口被接收到
9.    {
10.       IE2 &= 0xFE;                            // 串口 2 中断关闭
11.       SendDataByUart2(uart2temp);            //发送字符
12.       SendDataByUart2(0x0D);                 //发送换行符
13.       SendDataByUart2(0x0A);                 //发送换行符
14.       IE2 |= 0x01;                           // 串口 2 中断打开
15.       Flag=FALSE;                            //清除接收标识符
16.   }
17. }
```

代码清单：主函数

```
1.  int main()
2.  {
3.  ////////////////////////////////////////////////
4.  //注意: STC15W4K32S4 系列的芯片,上电后所有与 PWM 相关的 IO 口均为
5.  //       高阻态,需将这些口设置为准双向口或强推挽模式方可正常使用
6.  //相关 IO: P0.6/P0.7/P1.6/P1.7/P2.1/P2.2
7.  //         P2.3/P2.7/P3.7/P4.2/P4.4/P4.5
8.  ////////////////////////////////////////////////
9.      P4M1 &= 0x3F;   P4M0 &= 0x3F;      //设置 P4.6~P4.7 为准双向口
10.
11.     Uart2_Init();          //串口 2 初始化
12.     EA = 1;                //总中断打开
13.
14.     while(1)
15.     {
16.         UART2_Tx_Puts();    //串口接收到一个字符后返回该字符
17.     }
18. }
```

4.4.3. 硬件连接

本实验需要外接 USB 转 TTL 模块连接到开发板串口 2 上，该实验串口 2 选择的 P4.6 和 P4.7，具体接线图如下。

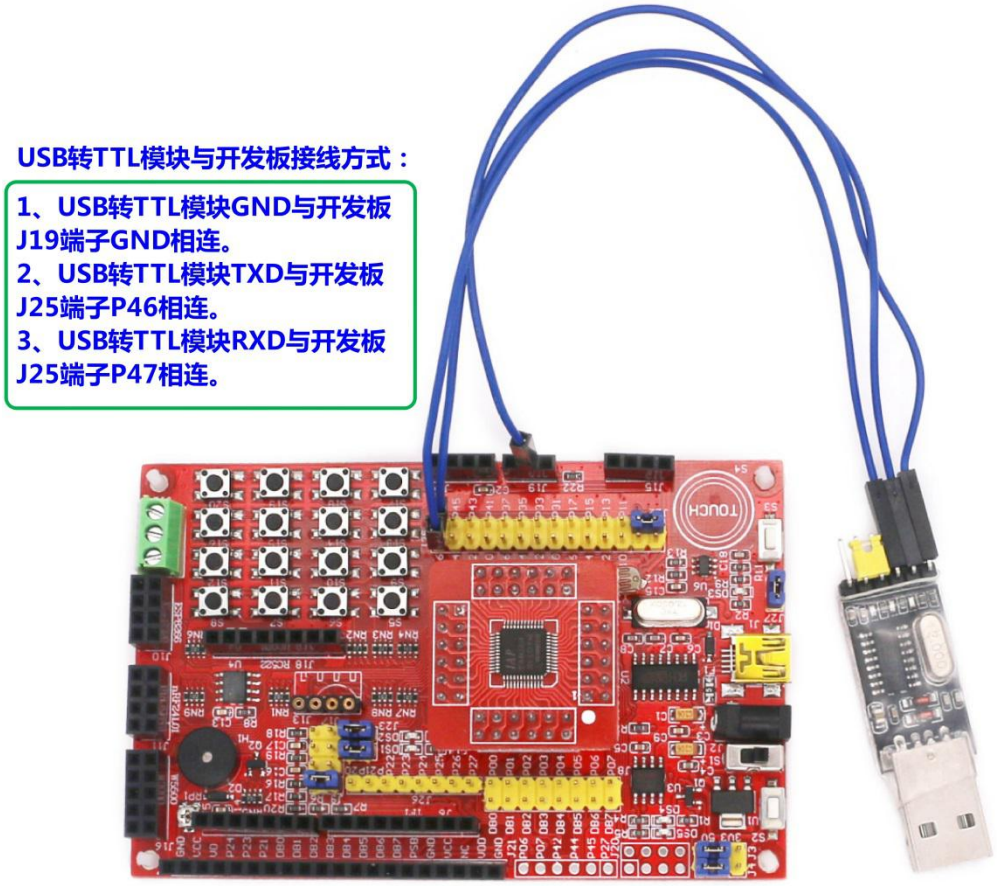

图 19：串口 2 实验连接图

4.4.4. 实验步骤

1. 解压“…\第 3 部分：配套例程源码\ 1 - 基础实验程序\”目录下的压缩文件“实验 2-8-2：串口 2 收发实验（P4.6 和 P4.7）”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC15”。
2. 启动 Keil C51。
3. 在 Keil C51 中执行“Project→Open Project”打开“…\UART2\projec”目录下的工程“UART2.uvproj”。
4. 点击编译按钮编译工程。注意查看编译输出栏，观察编译的结果，如果有错误，修改程序，直到编译成功为止。编译后生成的 HEX 文件“UART2.hex”位于工程目录下的“Output”文件夹中。
5. 打开 STC-ISP 软件下载程序 。下载使用内部 IRC 时钟，IRC 频率选择为 11.0592MHZ。
6. 程序运行后，打开串口调试助手，选择 USB 转 TTL 模块对应的串口号，波特率设置为 9600，在发送区发送什么字符在接收区就会收到什么字符（注意是字符不是字符串）。

4.5. 串口 3 收发实验（P0.0 和 P0.1）

✧ 注：本节的实验源码是在“实验 2-8-1：串口 1 收发实验（P3.0 和 P3.1）”的基础上修改。本节对应的实验源码是：“实验 2-8-3：串口 3 收发实验（P0.0 和 P0.1）”。

4.5.1. 工程需要用到的 c 文件

本实验需要用到的头文件以及添加头文件包含路径的方法请参考“实验 2-8-1：串口 1 收发实验（P3.0 和 P3.1）”部分。

4.5.2. 编写代码

首先，在 uart.c 文件中编写串口 3 的初始化函数 Uart3_Init，代码如下。

程序清单：串口 3 初始化函数

```
/*******************************************************************************
 * 描   述 : 串口 3 初始化函数
 * 入   参 : 无
 * 返回值 : 无
备注：波特率 9600bps    晶振 11.0592MHz
 *******************************************************************************/
void Uart3_Init(void)
{
    S3CON |= 0x10;          //启动串行接收器
    S3CON &= 0x30;          //8 位数据,可变波特率,串口 3 选择定时器 2 为波特率发生器
    AUXR |= 0x04;           //定时器 2 时钟为 Fosc,即 1T
    T2L = 0xE0;             //设定定时初值
    T2H = 0xFE;             //设定定时初值
    AUXR |= 0x10;           //启动定时器 2
    IE2 |= 0x08;             // 串口 3 中断打开
}
```

然后，编写串口 3 发送数据函数，把要发送的字节存放于数据缓存寄存器中，直到数据发送完成，代码如下。

程序清单：数据发送函数函数

```
/*******************************************************************************
 * 描   述 : 串口 3 发送数据函数
 * 入   参 : uint8 数据
 * 返回值 : 无
 *******************************************************************************/
void SendDataByUart3(uint8 dat)
{
    S3BUF = dat;                //写数据到 UART 数据寄存器
    while(!(S3CON&S3TI));       //在停止位没有发送时，S3TI 为 0 即一直等待
    S3CON&=~S3TI;               //清除 S3CON 寄存器对应 S3TI 位（该位必须软件清零）
```

```
11. }
```

之后，编写串口 3 的中断服务函数，将接收的数据存放到用户自定义变量 uart3temp 中，代码如下。

程序清单：中断服务函数

```
1.  /*********************************************************************************
2.   * 描   述 : 串口 3 中断服务函数
3.   * 入   参 : 无
4.   * 返回值 : 无
5.   *********************************************************************************/
6.  void Uart3() interrupt UART3_VECTOR using 1
7.  {
8.      IE2 &= 0xF7;                        // 串口 3 中断关闭
9.      Flag=TRUE;                         //接收到数据,接收标识符有效
10.     if (S3CON & S3RI)                  //串行接收到停止位的中间时刻时，该位置 1
11.   {
12.       S3CON &= ~S3RI;                //清除 S3CON 寄存器对应 S3RI 位(该位必须软件清零)
13.       uart3temp = S3BUF;
14.    }
15.   if (S3CON & S3TI)                   //在停止位开始发送时，该位置 1
16.    {
17.       S3CON &= ~S3TI;                //清除 S3CON 寄存器对应 S3TI 位(该位必须软件清零)
18.    }
19.     IE2 |= 0x08;                      // 串口 3 中断打开
20. }
```

最后，用户定义一个自定义函数 UART3_Tx_Puts，该函数将接收的数据原样返回去并加上回车符。主函数 main 在主循环中调用该函数。具体代码如下。

代码清单：用户函数 UART3_Tx_Puts

```
1.  /*********************************************************************************
2.   * 描   述 : 串口 3 接收到数据后发送出去
3.   * 入   参 : 无
4.   * 返回值 : 无
5.   *********************************************************************************/
6.  void UART3_Tx_Puts(void)
7.  {
8.    if(Flag)       //有新数据通过串口被接收到
9.      {
10.         IE2 &= 0xF7;                                    //串口 3 中断关闭
11.         SendDataByUart3(uart3temp);              //发送字符
```

```
    SendDataByUart3(0x0D);                  //发送换行符
    SendDataByUart3(0x0A);                  //发送换行符
    IE2 |= 0x08;                            //串口 3 中断打开
    Flag=FALSE;                             //清除接收标识符
  }
}
```

代码清单：主函数

```
int main()
{
/////////////////////////////////////////////////
//注意: STC15W4K32S4 系列的芯片,上电后所有与 PWM 相关的 IO 口均为
//       高阻态,需将这些口设置为准双向口或强推挽模式方可正常使用
//相关 IO: P0.6/P0.7/P1.6/P1.7/P2.1/P2.2
//         P2.3/P2.7/P3.7/P4.2/P4.4/P4.5
/////////////////////////////////////////////////
  P0M1 &= 0xFC; P0M0 &= 0xFC;     //设置 P0.0~P0.1 为准双向口

    Uart3_Init();           //串口 3 初始化
    EA = 1;                 //总中断打开

    while(1)
    {
        UART3_Tx_Puts();    //串口接收到一个字符后返回该字符
    }
}
```

4.5.3. 硬件连接

本实验需要外接 USB 转 TTL 模块连接到开发板串口 3 上，该实验串口 3 选择的 P0.0 和 P0.1，具体接线图如下。

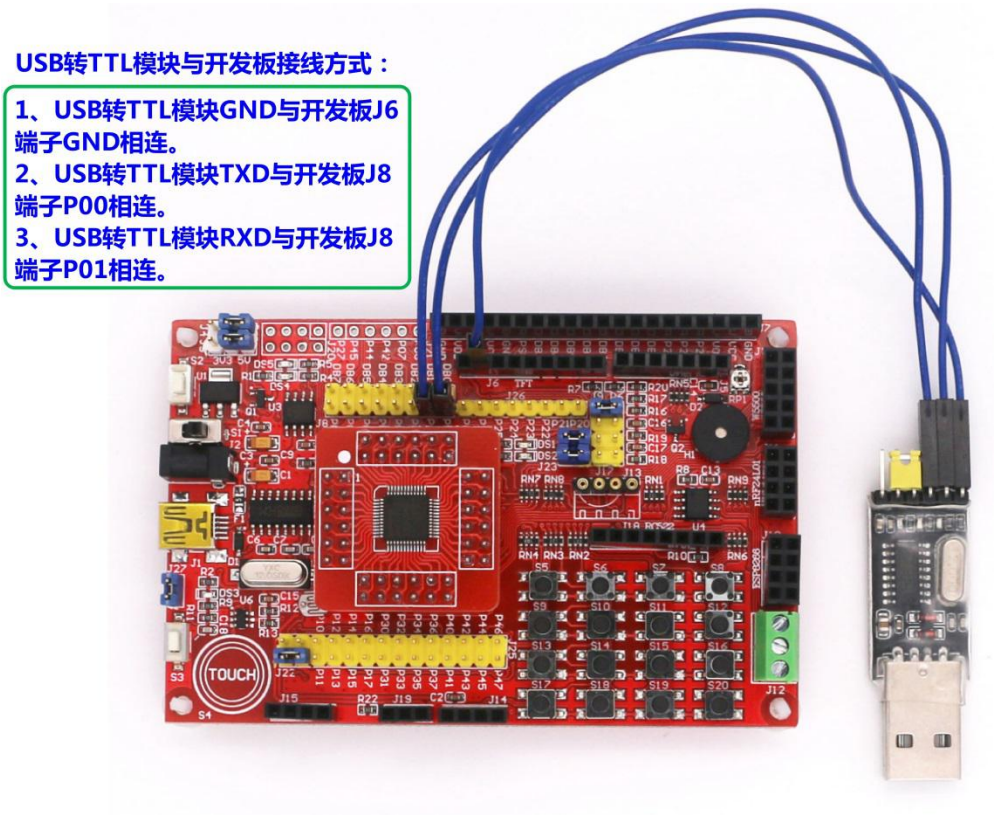

图 20：串口 3 实验连接图

4.5.4. 实验步骤

1. 解压“…\第 3 部分：配套例程源码\ 1 - 基础实验程序\”目录下的压缩文件“实验 2-8-3：串口 3 收发实验（P0.0 和 P0.1）”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC15”。
2. 启动 Keil C51。
3. 在 Keil C51 中执行“Project→Open Project”打开“…\UART3\projec”目录下的工程“UART3.uvproj”。
4. 点击编译按钮编译工程。注意查看编译输出栏，观察编译的结果，如果有错误，修改程序，直到编译成功为止。编译后生成的 HEX 文件“UART3.hex”位于工程目录下的“Output”文件夹中。
5. 打开 STC-ISP 软件下载程序 。下载使用内部 IRC 时钟，IRC 频率选择为 11.0592MHZ。
6. 程序运行后，打开串口调试助手，选择 USB 转 TTL 模块对应的串口号，波特率设置为 9600，在发送区发送什么字符在接收区就会收到什么字符（注意是字符不是字符串）。

4.6. 串口 4 收发实验（P0.2 和 P0.3）

✧ 注：本节的实验源码是在“实验 2-8-1：串口 1 收发实验（P3.0 和 P3.1）”的基础上修改。本节对应的实验源码是：“实验 2-8-4：串口 4 收发实验（P0.2 和 P0.3）”。

4.6.1. 工程需要用到的 c 文件

本实验需要用到的头文件以及添加头文件包含路径的方法请参考“实验 2-8-1：串口 1 收发实验（P3.0 和 P3.1）”部分。

4.6.2. 编写代码

首先，在 uart.c 文件中编写串口 4 的初始化函数 Uart4_Init，代码如下。

程序清单：串口 4 初始化函数

```
/*************************************************************************
 * 描  述 : 串口 4 初始化函数
 * 入  参 : 无
 * 返回值 : 无
备注：波特率 9600bps    晶振 11.0592MHz
 *************************************************************************/
void Uart4_Init(void)
{
    S4CON |= 0x10;       //启动串行接收器
    S4CON &= 0x30;       //8 位数据,可变波特率，串口 4 选择定时器 2 为波特率发生器
    AUXR |= 0x04;          //定时器 2 时钟为 Fosc,即 1T
    T2L = 0xE0;          //设定定时初值
    T2H = 0xFE;          //设定定时初值
    AUXR |= 0x10;          //启动定时器 2
    IE2 |= 0x10;       // 串口 4 中断打开
}
```

然后，编写串口 4 发送数据函数，把要发送的字节存放于数据缓存寄存器中，直到数据发送完成，代码如下。

程序清单：数据发送函数函数

```
/*************************************************************************
 * 描  述 : 串口 4 发送数据函数
 * 入  参 : uint8 数据
 * 返回值 : 无
 *************************************************************************/
void SendDataByUart4(uint8 dat)
{
    S4BUF = dat;                 //写数据到 UART 数据寄存器
    while(!(S4CON&S4TI));        //在停止位没有发送时，S4TI 为 0 即一直等待
    S4CON&=~S4TI;                //清除 S4CON 寄存器对应 S4TI 位（该位必须软件清零）
}
```

之后，编写串口 4 的中断服务函数，将接收的数据存放到用户自定义变量 uart4temp 中，

代码如下。

程序清单：中断服务函数

```
/*************************************************************************
 * 描  述 : 串口 4 中断服务函数
 * 入  参 : 无
 * 返回值 : 无
 *************************************************************************/
void Uart4() interrupt UART4_VECTOR using 1
{
    IE2 &= 0xEF;                        // 串口 4 中断关闭
    Flag=TRUE;                          //接收到数据,接收标识符有效
    if (S4CON & S4RI)                   //串行接收到停止位的中间时刻时，该位置 1
  {
      S4CON &= ~S4RI;                 //清除 S4CON 寄存器对应 S4RI 位(该位必须软件清零)
      uart4temp = S4BUF;
   }
  if (S4CON & S4TI)                   //在停止位开始发送时，该位置 1
   {
      S4CON &= ~S4TI;                   //清除 S4CON 寄存器对应 S4TI 位(该位必须软件清零)
   }
     IE2 |= 0x10;                        // 串口 4 中断打开
}
```

最后，用户定义一个自定义函数 UART4_Tx_Puts，该函数将接收的数据原样返回去并加上回车符。主函数 main 在主循环中调用该函数。具体代码如下。

代码清单：用户函数 UART4_Tx_Puts

```
/*************************************************************************
 * 描  述 : 串口 4 接收到数据后发送出去
 * 入  参 : 无
 * 返回值 : 无
 *************************************************************************/
void UART4_Tx_Puts(void)
{
  if(Flag)      //有新数据通过串口被接收到
    {
        IE2 &= 0xEF;                                    // 串口 4 中断关闭
        SendDataByUart4(uart4temp);             //发送字符
       SendDataByUart4(0x0D);                  //发送换行符
       SendDataByUart4(0x0A);                  //发送换行符
        IE2 |= 0x10;                                    // 串口 4 中断打开
```

```
        Flag=FALSE;                                    //清除接收标识符
    }
}
```

代码清单：主函数

```
int main()
{
//////////////////////////////////////////////////
//注意: STC15W4K32S4 系列的芯片,上电后所有与 PWM 相关的 IO 口均为
//       高阻态,需将这些口设置为准双向口或强推挽模式方可正常使用
//相关 IO: P0.6/P0.7/P1.6/P1.7/P2.1/P2.2
//         P2.3/P2.7/P3.7/P4.2/P4.4/P4.5
//////////////////////////////////////////////////
   P0M1 &= 0xF3; P0M0 &= 0xF3;     //设置 P0.2~P0.3 为准双向口

    Uart4_Init();          //串口 4 初始化
    EA = 1;                //总中断打开

    while(1)
    {
        UART4_Tx_Puts();   //串口接收到一个字符后返回该字符
    }
}
```

4.6.3. 硬件连接

本实验需要外接 USB 转 TTL 模块连接到开发板串口 4 上，该实验串口 4 选择的 P0.2 和 P0.3，具体接线图如下。

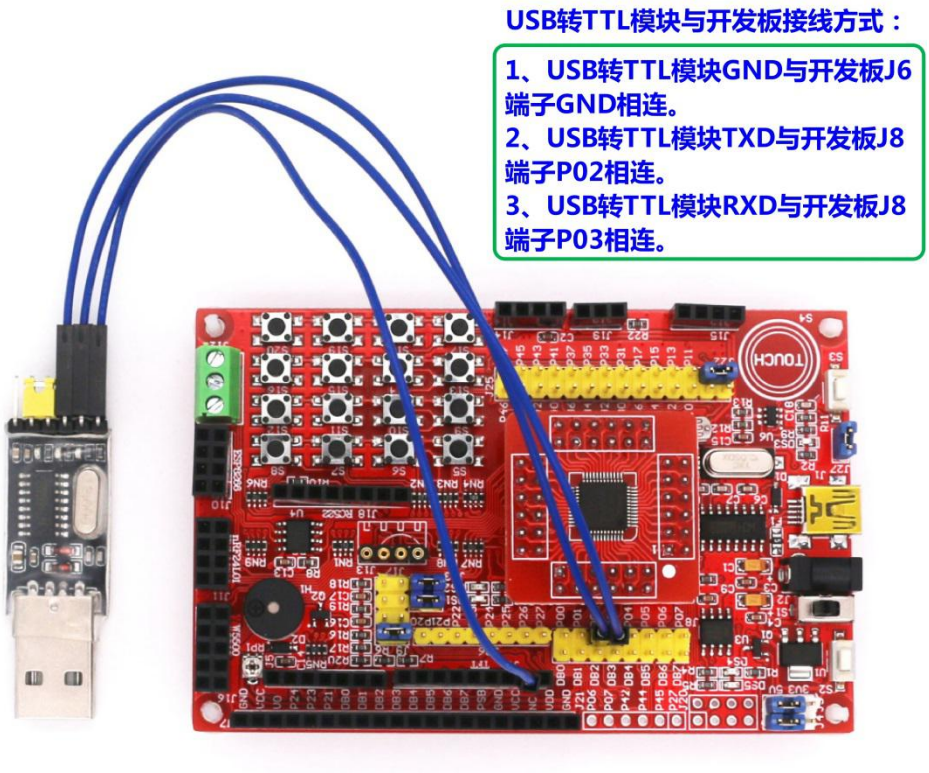

图 21：串口 4 实验连接图

4.6.4. 实验步骤

1. 解压“…\第 3 部分：配套例程源码\ 1 - 基础实验程序\”目录下的压缩文件“实验 2-8-4：串口 4 收发实验（P0.2 和 P0.3）”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC15”。
2. 启动 Keil C51。
3. 在 Keil C51 中执行“Project→Open Project”打开“…\UART4\projec”目录下的工程“UART4.uvproj”。
4. 点击编译按钮编译工程。注意查看编译输出栏，观察编译的结果，如果有错误，修改程序，直到编译成功为止。编译后生成的 HEX 文件“UART4.hex”位于工程目录下的“Output”文件夹中。
5. 打开 STC-ISP 软件下载程序 。下载使用内部 IRC 时钟，IRC 频率选择为 11.0592MHZ。
6. 程序运行后，打开串口调试助手，选择 USB 转 TTL 模块对应的串口号，波特率设置为 9600，在发送区发送什么字符在接收区就会收到什么字符（注意是字符不是字符串）。

4.7. 串口 1 串口 2 同时收发字符串实验

✧ 注：本节的实验源码是在“实验 2-8-1：串口 1 收发实验（P3.0 和 P3.1）”的基础上修改。本节对应的实验源码是：“实验 2-8-5：串口 1 串口 2 同时收发字符串实验”。

4.7.1. 工程需要用到的 c 文件

本例需要用到的 c 文件如下表所示，工程需要添加下表中的 c 文件。

表 9：实验需要用到的 c 文件

序号	文件名	后缀	功能描述
1	uart	.c	外部串行口有关的用户自定义函数。
2	delay	.c	包含用户自定义延时函数。
3	led	.c	包含与用户 led 控制有关的用户自定义函数。

4.7.2. 头文件引用和路径设置

■ 需要引用的头文件

```
1. #include "delay.h"
2. #include "uart.h"
```

■ 需要包含的头文件路径

本例需要包含的头文件路径如下表：

表 10：头文件包含路径

序号	路径	描述
1	..\ Source	Led.h、uart.h 和 delay.h 头文件在该路径，所以要包含。
2	..\User	15W4KxxS4.h 头文件在该路径，所以要包含。

MDK 中点击魔术棒，打开工程配置窗口，按照下图所示添加头文件包含路径。

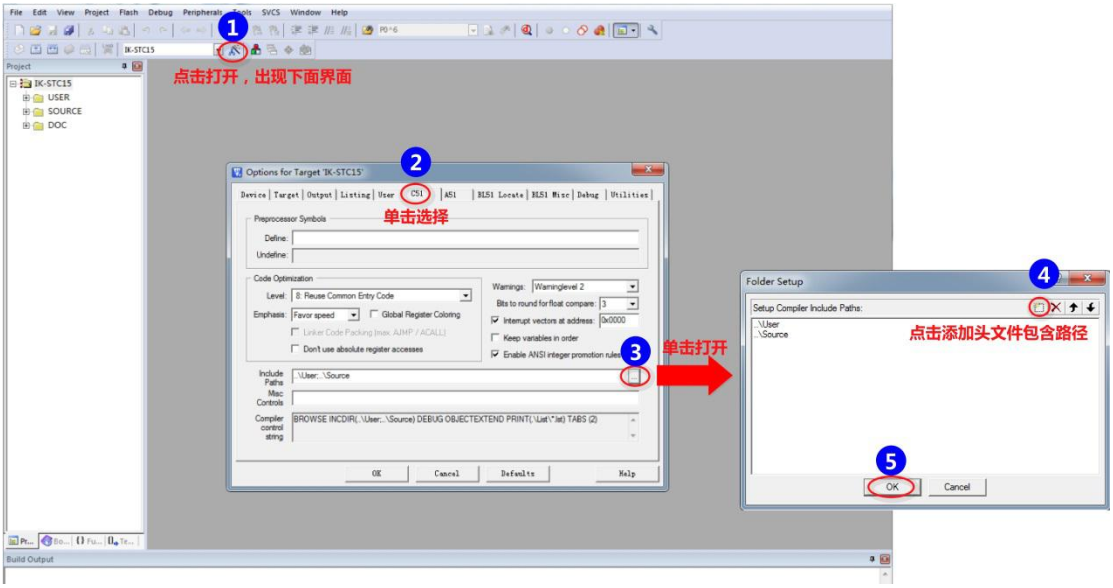

图 22：添加头文件包含路径

4.7.3. 编写代码

首先，在 uart.c 文件中编写串口 1 的初始化函数 Uart1_Init 和串口 1 发送单个字符函数 SendDataByUart1，这两个函数不介绍了。下面介绍下串口 1 清缓存函数 CLR_Buf1 和发送字符串函数 SendStringByUart1，代码如下。

程序清单：串口 1 发送字符串函数

```
/*******************************************************************************
 * 描    述 : 串口 1 发送字符串函数
 * 入    参 : 字符串
 * 返回值 : 无
 *******************************************************************************/
void SendStringByUart1(uint8 *s)
{
    while(*s)
    {
        SendDataByUart1(*s++);        //将字符串中的字符一个一个发送
    }
}
```

程序清单：串口 1 清缓存函数

```
/*******************************************************************************
功能描述：清除串口 1 缓存内容函数
入口参数：无
返回值：无
*******************************************************************************/
void CLR_Buf1(void)
{
    uint8 k;
    for(k=0;k<Buf1_Max;k++)       //将串口 1 缓存数组的值都清为零
    {
        Rec_Buf1[k] = 0;
    }
    i = 0;                        //清零串口 1 接收数据个数变量
}
```

然后，串口 2 的初始化函数 Uart2_Init 和串口 2 发送单个字符函数 SendDataByUart2，这两个函数不介绍了。下面介绍下串口 2 清缓存函数 CLR_Buf2 和发送字符串函数 SendStringByUart2，代码如下。

程序清单：串口 2 发送字符串函数

```
/*******************************************************************************
 * 描  述 : 串口 2 发送字符串函数
 * 入  参 : 字符串
 * 返回值 : 无
 *******************************************************************************/
void SendStringByUart2(uint8 *s)
{
    while (*s)                  //检测字符串结束标志
  {
    SendDataByUart2(*s++);          //发送当前字符
  }
}
```

程序清单：串口 2 清缓存函数

```
/*******************************************************************************
功能描述：清除串口 2 缓存内容函数
入口参数：无
返回值：无
*******************************************************************************/
void CLR_Buf2(void)
{
    uint8 k;
  for(k=0;k<Buf2_Max;k++)       //将串口 2 缓存数组的值都清为零
    {
        Rec_Buf2[k] = 0;
  }
  j = 0;                            //清零串口 2 接收数据个数变量
}
```

之后，编写串口 1 和串口 2 的中断服务函数，将接收的数据存放到用户自定义数组 Rec_Buf1 和 Rec_Buf2 中，代码如下。

程序清单：串口 1 中断服务函数

```
/*******************************************************************************
 * 描  述 : 串口 1 中断服务函数
 * 入  参 : 无
 * 返回值 : 无
 *******************************************************************************/
```

```
void Uart1() interrupt UART1_VECTOR using 1
{
    uint8 temp;
    ES = 0;                          // 串口 1 中断关闭
    if (RI)                         //串行接收到停止位的中间时刻时，该位置 1
   {
      RI = 0;                  //清除 RI 位 （该位必须软件清零）
          temp = SBUF;             //接收到的数赋值给临时变量
          if(temp !='\n')          //没有接收到换行符
            {
                Rec_Buf1[i] = temp;      //接收到的数存到接收数组中
                i++;                     //串口 1 接收数据个数变量累加
            }
            else                      //接收到结束符
            {
                Buf1_Length = i;       //接收数据长度赋值
                i = 0;                 //清零串口 1 接收数据个数变量
                Buf1_Flag=TRUE;        //接收完数据,接收标识符有效
                led_toggle(LED_1);     //翻转蓝色指示灯 DS1，方便观察实验现象
            }
    }
    if (TI)                     //在停止位开始发送时，该位置 1
    {
       TI = 0;                  //清除 TI 位（该位必须软件清零）
    }
      ES =  1;                    // 串口 1 中断打开
}
```

程序清单：串口 2 中断服务函数

```
/*********************************************************************
 * 描  述 : 串口 2 中断服务函数
 * 入  参 : 无
 * 返回值 : 无
 *********************************************************************/
void Uart2() interrupt UART2_VECTOR using 1
{
    uint8 temp;
    IE2 &= 0xFE;                              //串口 2 中断关闭
    if (S2CON & S2RI)                        //串行接收到停止位的中间时刻时,该位置 1
    {
      S2CON &= ~S2RI;                //清除 S2CON 寄存器对应 S2RI 位（该位必须软件清零）
```

```
13.       temp = S2BUF;                                  //接收到的数赋值给临时变量
14.       if(temp !='\n')                                //没有接收到换行符
15.       {
16.          Rec_Buf2[j] = temp;                          //接收到的数存到接收数组中
17.          j++;                                         //串口 2 接收数据个数变量累加
18.       }
19.       else                                           //接收到结束符
20.       {
21.          Buf2_Length = j;                             //接收数据长度赋值
22.          j = 0;                                       //清零串口 2 接收数据个数变量
23.          Buf2_Flag=TRUE;                              //接收完数据,接收标识符有效
24.          led_toggle(LED_2);                  //翻转红色指示灯 DS2，方便观察实验现象
25.       }
26.    }
27.    if (S2CON & S2TI)                                 //在停止位开始发送时，该位置 1
28.    {
29.       S2CON &= ~S2TI;              //清除 S2CON 寄存器对应 S2TI 位（该位必须软件清零）
30.    }
31.    IE2 |= 0x01;                                      //串口 2 中断打开
32. }
```

最后，用户定义一个自定义函数 UART1_2_Tx_Puts，该函数将接收的字符串原样返回去并加上回车符。主函数 main 在主循环中调用该函数。具体代码如下。

代码清单：用户函数 UART1_2_Tx_Puts

```
1.  /*********************************************************************
2.   * 描   述 : 串口 1 和串口 2 接收到字符串后发送出去
3.   * 入   参 : 无
4.   * 返回值 : 无
5.   *********************************************************************/
6.  void UART1_2_Tx_Puts(void)
7.  {
8.    if(Buf1_Flag)                 //串口 1 接收一组字符串完成
9.      {
10.         ES = 0;                                    //串口 1 中断关闭
11.         SendStringByUart1(Rec_Buf1);       //发送字符
12.         SendDataByUart1(0x0D);             //发送换行符
13.         SendDataByUart1(0x0A);             //发送换行符
14.         ES = 1;                           //串口 1 中断打开
15.         CLR_Buf1();                         //清除串口 1 缓存
16.         Buf1_Flag=FALSE;                  //清除接收标识符
```

```
17.   }
18.     if(Buf2_Flag)                   //串口 2 接收一组字符串完成
19.     {
20.         IE2 &= 0xFE;                                //串口 2 中断关闭
21.         SendStringByUart2(Rec_Buf2);        //发送字符
22.         SendDataByUart2(0x0D);              //发送换行符
23.         SendDataByUart2(0x0A);              //发送换行符
24.         IE2 |= 0x01;                        //串口 2 中断打开
25.         CLR_Buf2();                         //清除串口 2 缓存
26.         Buf2_Flag=FALSE;                    //清除接收标识符
27.   }
28. }
```

代码清单：主函数

```
1.  int main()
2.  {
3.  ////////////////////////////////////////////////
4.  //注意: STC15W4K32S4 系列的芯片,上电后所有与 PWM 相关的 IO 口均为
5.  //       高阻态,需将这些口设置为准双向口或强推挽模式方可正常使用
6.  //相关 IO: P0.6/P0.7/P1.6/P1.7/P2.1/P2.2
7.  //         P2.3/P2.7/P3.7/P4.2/P4.4/P4.5
8.  ////////////////////////////////////////////////
9.      P0M1 &= 0x3F;   P0M0 &= 0x3F;     //设置 P0.6~P0.7 为准双向口
10.     P3M1 &= 0xFC;   P3M0 &= 0xFC;     //设置 P3.0~P3.1 为准双向口
11.     P4M1 &= 0x3F;   P4M0 &= 0x3F;     //设置 P4.6~P4.7 为准双向口
12.
13.     Uart1_Init();                   //串口 1 初始化
14.     Uart2_Init();                   //串口 2 初始化
15.     EA = 1;                         //总中断打开
16.
17.     CLR_Buf1();                     //清除串口 1 缓存
18.     CLR_Buf2();                     //清除串口 2 缓存
19.
20.     while(1)
21.     {
22.         UART1_2_Tx_Puts();          //UART1 和 UART2 接收到字符串后发送出去
23.     }
24. }
```

4.7.4. 硬件连接

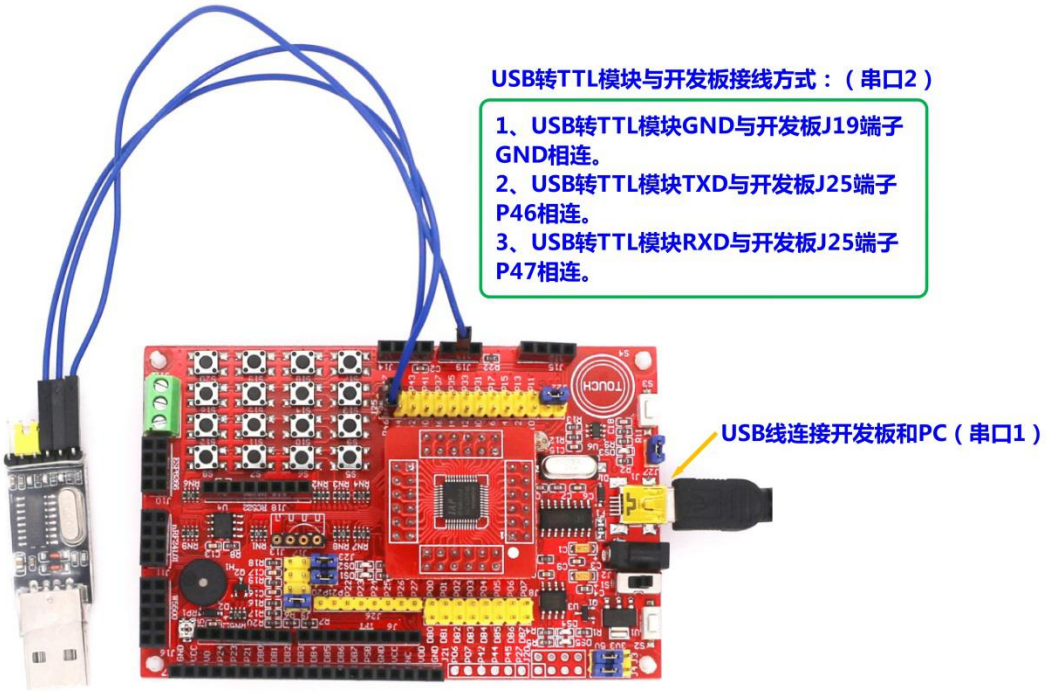

图 23：串口 1 串口 2 实验连接图

4.7.5. 实验步骤

1. 解压“…\第 3 部分：配套例程源码\ 1 - 基础实验程序\”目录下的压缩文件“实验 2-8-5：串口 1 串口 2 同时收发字符串实验”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC15”。
2. 启动 Keil C51。
3. 在 Keil C51 中执行“Project→Open Project”打开“…\UART1_UART2 \projec”目录下的工程“UART1_UART2.uvproj”。
4. 点击编译按钮编译工程。注意查看编译输出栏，观察编译的结果，如果有错误，修改程序，直到编译成功为止。编译后生成的 HEX 文件“UART1_UART2.hex”位于工程目录下的“Output”文件夹中。
5. 打开 STC-ISP 软件下载程序 。下载使用内部 IRC 时钟，IRC 频率选择为 11.0592MHZ。
6. 程序运行后，打开串口调试助手，波特率设置为 9600，选择开发板 CH340 对应的串口号，打勾选中发送新行，在发送区发送什么字符串在接收区就会收到什么字符串。如下图所示。

图 24：串口 1 实验现象截图

7. 再次打开串口调试助手，波特率设置为 9600，选择连接串口 2 的 USB 转 TTL 模块对应的串口号，打勾选中发送新行，在发送区发送什么字符串在接收区就会收到什么字符串。

4.8. 串口 3 串口 4 同时收发字符串实验

✧ 注：本节的实验源码是在“实验 2-8-5：串口 1 串口 2 同时收发字符串实验”的基础上修改。本节对应的实验源码是：“实验 2-8-6：串口 3 串口 4 同时收发字符串实验”。

4.8.1. 工程需要用到的 c 文件

本实验需要用到的头文件以及添加头文件包含路径的方法请参考“实验 2-8-5：串口 1 串口 2 同时收发字符串实验”部分。

4.8.2. 编写代码

首先，在 uart.c 文件中编写串口 3 的初始化函数 Uart3_Init 和串口 3 发送单个字符函数 SendDataByUart3，这两个函数不介绍了。下面介绍下串口 3 清缓存函数 CLR_Buf3 和发送字符串函数 SendStringByUart3，代码如下。

程序清单：串口 3 发送字符串函数

```
1.  /*********************************************************************************
2.   * 描   述 : 串口 3 发送字符串函数
3.   * 入   参 : 字符串
4.   * 返回值 : 无
5.   *********************************************************************************/
6.  void SendStringByUart3(uint8 *s)
```

```
{
    while (*s)                          //检测字符串结束标志
    {
      SendDataByUart3(*s++);          //发送当前字符
    }
}
```

程序清单：串口 3 清缓存函数

```
/*********************************************************************************
功能描述：清除串口 3 缓存内容函数
入口参数：无
返回值：无
*********************************************************************************/
void CLR_Buf3(void)
{
    uint8 k;
   for(k=0;k<Buf3_Max;k++)       //将串口 3 缓存数组的值都清为零
    {
        Rec_Buf3[k] = 0;
    }
  i = 0;                         //清零串口 3 接收数据个数变量
}
```

然后，串口 4 的初始化函数 Uart4_Init 和串口 4 发送单个字符函数 SendDataByUart4，这两个函数不介绍了。下面介绍下串口 4 清缓存函数 CLR_Buf4 和发送字符串函数 SendStringByUart4，代码如下。

程序清单：串口 4 发送字符串函数

```
/*********************************************************************************
 * 描   述 : 串口 4 发送字符串函数
 * 入   参 : 字符串
 * 返回值 : 无
 *********************************************************************************/
void SendStringByUart4(char *s)
{
    while (*s)                          //检测字符串结束标志
   {
    SendDataByUart4(*s++);          //发送当前字符
   }
}
```

程序清单：串口 4 清缓存函数

```
/*******************************************************************************
功能描述：清除串口 4 缓存内容函数
入口参数：无
返回值：无
*******************************************************************************/
void CLR_Buf4(void)
{
    uint8 k;
  for(k=0;k<Buf4_Max;k++)       //将串口 4 缓存数组的值都清为零
    {
        Rec_Buf4[k] = 0;
  }
  j = 0;                              //清零串口 4 接收数据个数变量
}
```

之后，编写串口 3 和串口 4 的中断服务函数，将接收的数据存放到用户自定义数组 Rec_Buf3 和 Rec_Buf4 中，代码如下。

程序清单：串口 3 中断服务函数

```
/*******************************************************************************
 * 描  述 : 串口 3 中断服务函数
 * 入  参 : 无
 * 返回值 : 无
 *******************************************************************************/
void Uart3() interrupt UART3_VECTOR using 1
{
    uint8 temp;
    IE2 &= 0xF7;                                //串口 3 中断关闭
    if (S3CON & S3RI)                    //串行接收到停止位的中间时刻时，该位置 1
    {
      S3CON &= ~S3RI;                  //清除 S3CON 寄存器对应 S3RI 位(该位必须软件清零)
        temp = S3BUF;                     //接收到的数赋值给临时变量
        if(temp !='\n')                   //没有接收到换行符
          {
            Rec_Buf3[i] = temp;          //接收到的数存到接收数组中
            i++;                                //串口 3 接收数据个数变量累加
          }
          else                                 //接收到结束符
          {
```

```
                Buf3_Length = i;            //接收数据长度赋值
                i = 0;                      //清零串口 3 接收数据个数变量
                Buf3_Flag=TRUE;             //接收完数据,接收标识符有效
                led_toggle(LED_1);          //翻转蓝色指示灯 DS1，方便观察实验现象
            }
    }
    if (S3CON & S3TI)                   //在停止位开始发送时，该位置 1
    {
       S3CON &= ~S3TI;                  //清除 S3CON 寄存器对应 S3TI 位(该位必须软件清零)
    }
    IE2 |= 0x08;                        //串口 3 中断打开
}
```

程序清单：串口 2 中断服务函数

```
/*********************************************************************
 * 描  述 : 串口 4 中断服务函数
 * 入  参 : 无
 * 返回值 : 无
 *********************************************************************/
void Uart4() interrupt UART4_VECTOR using 1
{
    uint8 temp;
    IE2 &= 0xEF;                                    //串口 4 中断关闭
    if(S4CON & S4RI)                            //串行接收到停止位的中间时刻时，该位置 1
    {
        S4CON &= ~S4RI;                     //清除 S4CON 寄存器对应 S4RI 位（该位必须软件清零）
        temp = S4BUF;                               //接收到的数赋值给临时变量
        if(temp !='\n')                             //没有接收到换行符
        {
            Rec_Buf4[j] = temp;                     //接收到的数存到接收数组中
            j++;                                    //串口 4 接收数据个数变量累加
         }
         else                                       //接收到结束符
         {
             Buf4_Length = j;                       //接收数据长度赋值
             j = 0;                                 //清零串口 4 接收数据个数变量
             Buf4_Flag=TRUE;                        //接收完数据,接收标识符有效
             led_toggle(LED_2);                 //翻转红色指示灯 DS2，方便观察实验现象
          }
    }
    if(S4CON & S4TI)                            //在停止位开始发送时，该位置 1
```

```
    {
        S4CON &= ~S4TI;                 //清除 S4CON 寄存器对应 S4TI 位（该位必须软件清零）
    }
    IE2 |= 0x10;                                    //串口 4 中断打开
}
```

最后，用户定义一个自定义函数 UART3_4_Tx_Puts，该函数将接收的字符串原样返回去并加上回车符。主函数 main 在主循环中调用该函数。具体代码如下。

代码清单：用户函数 UART3_4_Tx_Puts

```
/*********************************************************************
 * 描   述 : 串口 3 和串口 4 接收到字符串后发送出去
 * 入   参 : 无
 * 返回值 : 无
 *********************************************************************/
void UART3_4_Tx_Puts(void)
{
  if(Buf3_Flag)                 //串口 3 接收一组字符串完成
  {
      IE2 &= 0xF7;                          //串口 3 中断关闭
      SendStringByUart3(Rec_Buf3);          //发送字符
      SendDataByUart3(0x0D);                //发送换行符
      SendDataByUart3(0x0A);                //发送换行符
      IE2 |= 0x08;                          //串口 3 中断打开
      CLR_Buf3();                           //清除串口 3 缓存
      Buf3_Flag=FALSE;                      //清除接收标识符
  }
  if(Buf4_Flag)                 //串口 4 接收一组字符串完成
  {
      IE2 &= 0xEF;                          //串口 4 中断关闭
      SendStringByUart4(Rec_Buf4);          //发送字符
      SendDataByUart4(0x0D);                //发送换行符
      SendDataByUart4(0x0A);                //发送换行符
      IE2 |= 0x10;                          //串口 4 中断打开
      CLR_Buf4();                             //清除串口 4 缓存
      Buf4_Flag=FALSE;                      //清除接收标识符
  }
}
```

代码清单：主函数

```
int main()
{
//////////////////////////////////////////////////
//注意: STC15W4K32S4 系列的芯片,上电后所有与 PWM 相关的 IO 口均为
//       高阻态,需将这些口设置为准双向口或强推挽模式方可正常使用
//相关 IO: P0.6/P0.7/P1.6/P1.7/P2.1/P2.2
//         P2.3/P2.7/P3.7/P4.2/P4.4/P4.5
//////////////////////////////////////////////////
    P0M1 &= 0x30;   P0M0 &= 0x30;     //设置 P0.0~P0.3、P0.6~P0.7 为准双向口

    Uart3_Init();                     //串口 3 初始化
    Uart4_Init();                     //串口 4 初始化
    EA = 1;                           //总中断打开

    CLR_Buf3();                       //清除串口 3 缓存
    CLR_Buf4();                       //清除串口 4 缓存

    while(1)
    {
        UART3_4_Tx_Puts();    //UART3 和 UART4 接收到字符串后发送出去
    }
}
```

4.8.3. 硬件连接

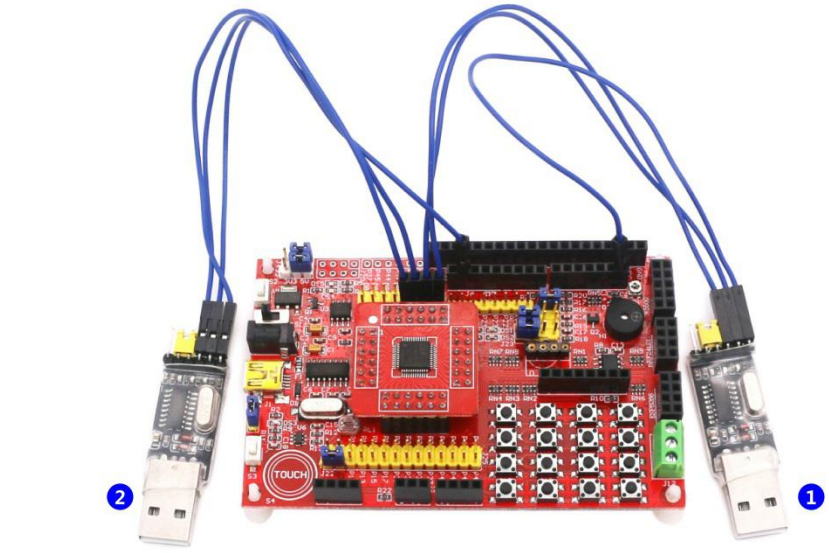

图 25：串口 3 串口 4 实验连接图

4.8.4. 实验步骤

1. 解压“…\第 3 部分：配套例程源码\ 1 - 基础实验程序\”目录下的压缩文件“实验 2-8-6：串口 3 串口 4 同时收发字符串实验”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC15”。
2. 启动 Keil C51。
3. 在 Keil C51 中执行“Project→Open Project”打开“…\UART3_UART4 \projec”目录下的工程“UART3_UART4.uvproj”。
4. 点击编译按钮编译工程。注意查看编译输出栏，观察编译的结果，如果有错误，修改程序，直到编译成功为止。编译后生成的 HEX 文件“UART3_UART4.hex”位于工程目录下的“Output”文件夹中。
5. 打开 STC-ISP 软件下载程序 。下载使用内部 IRC 时钟，IRC 频率选择为 11.0592MHZ。
6. 程序运行后，打开串口调试助手，波特率设置为 9600，选择连接串口 3 的 USB 转 TTL 模块对应的串口号，打勾选中发送新行，在发送区发送什么字符串在接收区就会收到什么字符串。如下图所示。

图 26：串口 3 实验现象截图

7. 再次打开串口调试助手，波特率设置为 9600，选择连接串口 4 的 USB 转 TTL 模块对应的串口号，打勾选中发送新行，在发送区发送什么字符串在接收区就会收到什么字符串。

4.9. 串口 1 串口 2 串口 3 串口 4 同时收发实验

✧ 注：本节的实验源码是在“实验 2-8-5：串口 1 串口 2 同时收发字符串实验”的基础上修改。本节对应的实验源码是:“实验 2-8-7:串口 1 串口 2 串口 3 串口 4 同时收发实验”。

4.9.1. 工程需要用到的 c 文件

本实验需要用到的头文件以及添加头文件包含路径的方法请参考“实验 2-8-5：串口 1 串口 2 同时收发字符串实验”部分。

4.9.2. 编写代码

首先，在 uart.c 文件中编写串口 1、串口 2、串口 3 和串口 4 的初始化函数 Uart1234_Init，代码如下。

程序清单：串口初始化函数

```
/*************************************************************************
 * 描  述 : 串口 1/2/3/4 初始化函数
 * 入  参 : 无
 * 返回值 : 无
备注：波特率 9600bps    晶振 11.0592MHz
 *************************************************************************/
void Uart1234_Init(void)
{
    P_SW2|=S2_S;          //选择 P46 P47 为串口 2

    //串口 1 配置
    PCON &= 0x3f;            //串口 1 波特率不倍速，串行口工作方式由 SM0、SM1 决定
    SCON = 0x50;             //串口 1 的 8 位数据,可变波特率，启动串行接收器
    AUXR |= 0x01;            //串口 1 选择定时器 2 为波特率发生器
    //串口 2 配置
    S2CON = 0x50;            //串口 2 的 8 位数据,可变波特率
    //串口 3 配置
    S3CON |= 0x10;         //串口 3 启动串行接收器
    S3CON &= 0x30;         //串口 3 选择定时器 2 为波特率发生器，8 位数据,可变波特率
    //串口 4 配置
    S4CON |= 0x10;         //启动串行接收器
    S4CON &= 0x30;         //8 位数据,可变波特率，串口 4 选择定时器 2 为波特率发生器

    //定时器 2 配置
    AUXR |= 0x04;            //定时器 2 时钟为 Fosc,即 1T
    T2L = 0xE0;            //设定定时初值
    T2H = 0xFE;          //设定定时初值
    AUXR |= 0x10;        //启动定时器 2

```

```
    //打开串口中断
    ES = 1;                                 //串口 1 中断打开
    IE2 |= 0x01;                            //串口 2 中断打开
    IE2 |= 0x08;                            //串口 3 中断打开
    IE2 |= 0x10;                            //串口 4 中断打开
}
```

然后，介绍下 4 个串口的握手函数，代码如下。（串口 1、串口 2、串口 3 和串口 4 的发送函数、清缓存函数不作介绍）

程序清单：串口 1 握手函数

```
/*************************************************************************
功能描述：握手成功与否函数
入口参数：uint8 *a
返回值：位
*************************************************************************/
bit Hand1(uint8 *a)
{
    if(strstr(Rec_Buf1,a)!=NULL)     //判断字符串 a 是否是字符串 Rec_Buf1 的子串
        return 1;                    //如果字符串 a 是字符串 Rec_Buf1 的子串
    else
        return 0;                    //如果字符串 a 不是字符串 Rec_Buf1 的子串
}
```

程序清单：串口 2 握手函数

```
/*************************************************************************
功能描述：握手成功与否函数
入口参数：uint8 *a
返回值：位
*************************************************************************/
bit Hand2(uint8 *a)
{
  if(strstr(Rec_Buf2,a)!=NULL)       //判断字符串 a 是否是字符串 Rec_Buf2 的子串
      return 1;                      //如果字符串 a 是字符串 Rec_Buf2 的子串
  else
       return 0;                     //如果字符串 a 不是字符串 Rec_Buf2 的子串
}
```

程序清单：串口 3 握手函数

```
/*******************************************************************************
功能描述：握手成功与否函数
入口参数：uint8 *a
返回值：位
*******************************************************************************/
bit Hand3(uint8 *a)
{
    if(strstr(Rec_Buf3,a)!=NULL)     //判断字符串 a 是否是字符串 Rec_Buf3 的子串
        return 1;                    //如果字符串 a 是字符串 Rec_Buf3 的子串
    else
        return 0;                    //如果字符串 a 不是字符串 Rec_Buf3 的子串
}
```

程序清单：串口 4 握手函数

```
/*******************************************************************************
功能描述：握手成功与否函数
入口参数：uint8 *a
返回值：位
*******************************************************************************/
bit Hand4(uint8 *a)
{
  if(strstr(Rec_Buf4,a)!=NULL)       //判断字符串 a 是否是字符串 Rec_Buf4 的子串
        return 1;                    //如果字符串 a 是字符串 Rec_Buf4 的子串
  else
        return 0;                    //如果字符串 a 不是字符串 Rec_Buf4 的子串
}
```

之后，编写串口 1、串口 2、串口 3 和串口 4 的中断服务函数，将接收的数据存放到用户自定义数组 Rec_Buf1、Rec_Buf2、Rec_Buf3 和 Rec_Buf4 中，代码如下。

程序清单：串口 1 中断服务函数

```
/*******************************************************************************
 * 描  述 : 串口 1 中断服务函数
 * 入  参 : 无
 * 返回值 : 无
 *******************************************************************************/
void Uart1() interrupt UART1_VECTOR using 1
```

```
{
    ES = 0;                          // 串口 1 中断关闭
    if (RI)                          //串行接收到停止位的中间时刻时，该位置 1
  {
      RI = 0;                  //清除 RI 位 （该位必须软件清零）
      Rec_Buf1[i] = SBUF;      //接收到的数存到接收数组中
      i++;                     //串口 1 接收数据个数变量累加
      if(i>Buf_Max)            //判断接收数据个数是否超限
      {
          i = 0;                 //清零接收数据个数
       }
   }
   if (TI)                     //在停止位开始发送时，该位置 1
   {
      TI = 0;                  //清除 TI 位（该位必须软件清零）
   }
   ES =  1;                    // 串口 1 中断打开
}
```

程序清单：串口 2 中断服务函数

```
/*************************************************************************
 * 描  述 : 串口 2 中断服务函数
 * 入  参 : 无
 * 返回值 : 无
 *************************************************************************/
void Uart2() interrupt UART2_VECTOR using 1
{
    IE2 &= 0xFE;                          // 串口 2 中断关闭
    if (S2CON & S2RI)                     //串行接收到停止位的中间时刻时，该位置 1
    {
      S2CON &= ~S2RI;              //清除 S2CON 寄存器对应 S2RI 位（该位必须软件清零）
      Rec_Buf2[j] = S2BUF;     //把串口 2 缓存 SBUF 寄存器数据依次存放到数组 Rec_Buf2 中
      j++;                      //串口 2 接收数据个数变量累加
      if(j>Buf_Max)           //接收数大于定义接收数组最大个数时，覆盖接收数组之前值
      {
          j = 0;              //清零串口 2 接收数据个数变量
       }
     }
     if (S2CON & S2TI)                      //在停止位开始发送时，该位置 1
     {
        S2CON &= ~S2TI;     //清除 S2CON 寄存器对应 S2TI 位（该位必须软件清零）
```

```
22.       }
23.      IE2 |= 0x01;                                    //串口 2 中断打开
24. }
```

程序清单：串口 3 中断服务函数

```
1.  /*********************************************************************
2.   * 描   述 : 串口 3 中断服务函数
3.   * 入   参 : 无
4.   * 返回值 : 无
5.   *********************************************************************/
6.  void Uart3() interrupt UART3_VECTOR using 1
7.  {
8.      IE2 &= 0xF7;                                  // 串口 3 中断关闭
9.      if (S3CON & S3RI)                            //串行接收到停止位的中间时刻时，该位置 1
10.     {
11.       S3CON &= ~S3RI;                          //清除 S3CON 寄存器对应 S3RI 位(该位必须软件清零)
12.       Rec_Buf3[m] = S3BUF;    //把串口 3 缓存 SBUF 寄存器数据依次存放到数组 Rec_Buf3 中
13.       m++;                                      //串口 3 接收数据个数变量累加
14.       if(m>Buf_Max)           //接收数大于定义接收数组最大个数时，覆盖接收数组之前值
15.       {
16.            m = 0;                              //清零串口 3 接收数据个数变量
17.       }
18.    }
19.   if (S3CON & S3TI)                             //在停止位开始发送时，该位置 1
20.    {
21.       S3CON &= ~S3TI;       //清除 S3CON 寄存器对应 S3TI 位（该位必须软件清零）
22.    }
23.    IE2 |= 0x08;                                  //串口 3 中断打开
24. }
```

程序清单：串口 4 中断服务函数

```
1.  /*********************************************************************
2.   * 描   述 : 串口 4 中断服务函数
3.   * 入   参 : 无
4.   * 返回值 : 无
5.   *********************************************************************/
6.  void Uart4() interrupt UART4_VECTOR
7.  {
8.      IE2 &= 0xEF;                                  //串口 4 中断关闭
```

```
    if(S4CON & S4RI)            //串行接收到停止位的中间时刻时，该位置 1
    {
        S4CON &= ~S4RI;          //清除 S4CON 寄存器对应 S4RI 位（该位必须软件清零）
        Rec_Buf4[n] = S4BUF;  //把串口 4 缓存 SBUF 寄存器数据依次存放到数组 Rec_Buf4 中
        n++;                  //串口 4 接收数据个数变量累加
      if(n>Buf_Max)           //接收数大于定义接收数组最大个数时，覆盖接收数组之前值
        {
                n = 0;              //清零串口 4 接收数据个数变量
        }
    }
    if(S4CON & S4TI)            //在停止位开始发送时，该位置 1
    {
        S4CON &= ~S4TI;          //清除 S4CON 寄存器对应 S4TI 位（该位必须软件清零）
    }
     IE2 |= 0x10;               //串口 4 中断打开
}
```

最后，主函数 main 在主循环中判断对应串口接收到的字符串是不是规定的字符串，然后再发送另一串字符串。具体代码如下。

代码清单：主函数

```
int main()
{
//////////////////////////////////////////////////
//注意: STC15W4K32S4 系列的芯片,上电后所有与 PWM 相关的 IO 口均为
//       高阻态,需将这些口设置为准双向口或强推挽模式方可正常使用
//相关 IO: P0.6/P0.7/P1.6/P1.7/P2.1/P2.2
//         P2.3/P2.7/P3.7/P4.2/P4.4/P4.5
//////////////////////////////////////////////////
    P0M1 &= 0x30;   P0M0 &= 0x30;     //设置 P0.0~P0.3、P0.6~P0.7 为准双向口
    P3M1 &= 0xFC;   P3M0 &= 0xFC;     //设置 P3.0~P3.1 为准双向口
    P4M1 &= 0x3F;   P4M0 &= 0x3F;     //设置 P4.6~P4.7 为准双向口

    Uart1234_Init();                                    //串口 1/2/3/4 初始化
    EA = 1;                                             //总中断打开
    CLR_Buf1();                                         //清除串口 1 缓存
    CLR_Buf2();                                         //清除串口 2 缓存
    CLR_Buf3();                                         //清除串口 3 缓存
    CLR_Buf4();                                         //清除串口 4 缓存

    while(1)
    {
```

```
22.         if(Hand1("UART1"))                                  //串口 1 收到字符串 UART1
23.         {
24.             CLR_Buf1();                     //将串口 1 缓存数组的值都清为零
25.             ES = 0;                                                 //串口 1 中断关闭
26.             SendStringByUart1("UART1 CHECK OK!\r\n");        //串口 1 发送字符串
27.             ES = 1;                                     //串口 1 中断打开
28.         }
29.         if(Hand2("UART2"))                                  //串口 2 收到字符串 UART2
30.         {
31.             CLR_Buf2();                     //将串口 2 缓存数组的值都清为零
32.             IE2 &= 0xFE;                                            //串口 2 中断关闭
33.             SendStringByUart2("UART2 CHECK OK!\r\n");        //串口 2 发送字符串
34.             IE2 |= 0x01;                        //串口 2 中断打开
35.         }
36.         if(Hand3("UART3"))                                  //串口 3 收到字符串 UART3
37.         {
38.             CLR_Buf3();                       //将串口 3 缓存数组的值都清为零
39.             IE2 &= 0xF7;                        //串口 3 中断关闭
40.             SendStringByUart3("UART3 CHECK OK!\r\n");        //串口 3 发送字符串
41.             IE2 |= 0x08;                              //串口 3 中断打开
42.         }
43.         if(Hand4("UART4"))                                  //串口 4 收到字符串 UART4
44.         {
45.             CLR_Buf4();                       //将串口 4 缓存数组的值都清为零
46.             IE2 &= 0xEF;                        //串口 4 中断关闭
47.             SendStringByUart4("UART4 CHECK OK!\r\n");        //串口 4 发送字符串
48.             IE2 |= 0x10;                        //串口 4 中断打开
49.         }
50.     }
51. }
```

4.9.3. 硬件连接

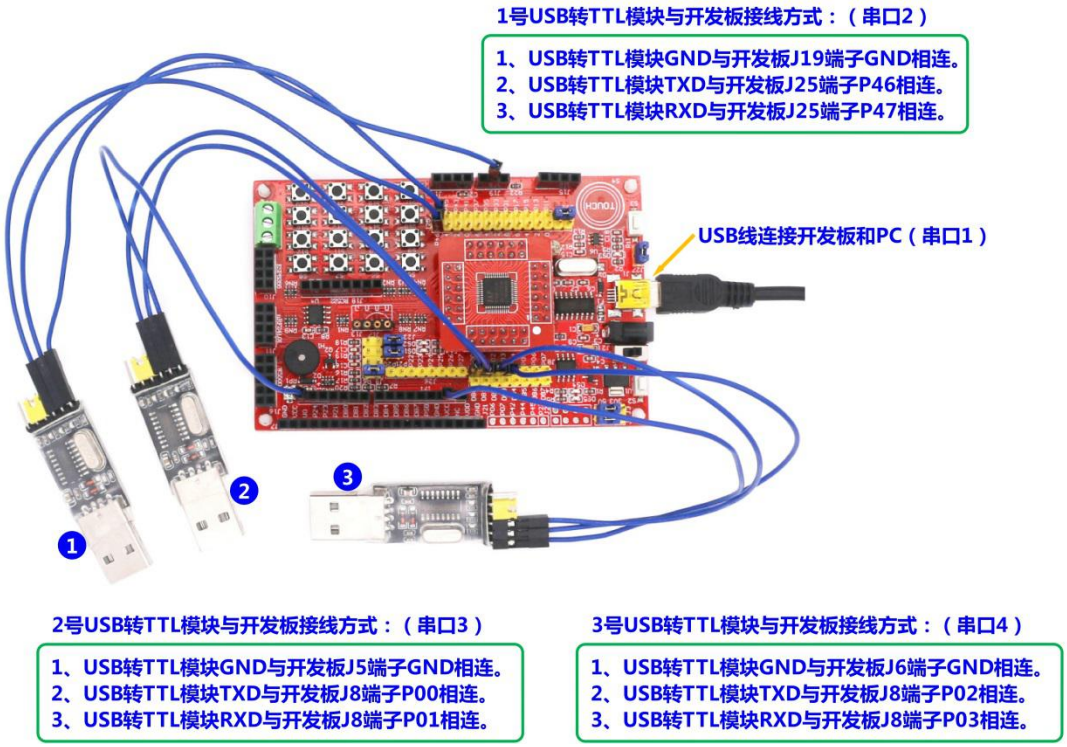

图 27：4 个串口同时通信实验连接图

4.9.4. 实验步骤

1. 解压“…\第 3 部分：配套例程源码\ 1 - 基础实验程序\”目录下的压缩文件“实验 2-8-7：串口 1 串口 2 串口 3 串口 4 同时收发实验”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC15”。
2. 启动 Keil C51。
3. 在 Keil C51 中执行“Project→Open Project”打开“…\ UART1~UART4 \projec”目录下的工程“UART1~UART4.uvproj”。
4. 点击编译按钮编译工程。注意查看编译输出栏，观察编译的结果，如果有错误，修改程序，直到编译成功为止。编译后生成的 HEX 文件“UART1~UART4.hex”位于工程目录下的“Output”文件夹中。
5. 打开 STC-ISP 软件下载程序 。下载使用内部 IRC 时钟，IRC 频率选择为 11.0592MHZ。
6. 程序运行后，打开串口调试助手，波特率设置为 9600，选择连接开发板 CH340 对应的串口号，在发送区发送字符串“UART1”，可在接收区中收到字符串“UART1 CHECK OK!”。如下图所示。

图 28：串口 1 实验现象截图

7. 打开串口调试助手，波特率设置为 9600，选择连接串口 2 的 USB 转 TTL 模块对应的串口号，在发送区字符串“UART2”，可在接收区中收到字符串“UART2 CHECK OK!”。
8. 打开串口调试助手，波特率设置为 9600，选择连接串口 3 的 USB 转 TTL 模块对应的串口号，在发送区字符串“UART3”，可在接收区中收到字符串“UART3 CHECK OK!”。
9. 打开串口调试助手，波特率设置为 9600，选择连接串口 4 的 USB 转 TTL 模块对应的串口号，在发送区字符串“UART4”，可在接收区中收到字符串“UART4 CHECK OK!”。