青风手把手教你学 stm32f030 系列教程

---------- 库函数操作版本

出品论坛： www.qfv8.com 青风电子社区

作者： 青风
出品论坛: www.qfv8.com
购买地址：http://qfv5.taobao.com
QQ 技术群：241364123
硬件平台：QF-STM32F030 开发板

2.10 串口通信的实现

2.10.1 原理分析

串口通信也称为异步串行通信，学过 51 的同学都会知道串口通信。串口是计算机上一种非常通用设备通信的协议。大多数计算机包含两个基于 RS232 的串口。串口同时也是仪器仪表设备通用的通信协议；很多 GPIB 兼容的设备也带有 RS-232 口。同时，串口通信协议也可以用于获取远程采集设备的数据。串口通信的概念非常简单，串口按位（bit）发送和接收字节。

通信使用 3 根线完成：（1）地线，（2）发送，（3）接收。由于串口通信是异步的，端口能够在一根线上发送数据同时在另一根线上接收数据。其他线用于握手，但是不是必须的。串口通信最重要的参数是波特率、数据位、停止位和奇偶校验。

对于两个进行通信的端口，这些参数必须匹配：

a，波特率：这是一个衡量通信速度的参数。它表示每秒钟传送的 bit 的个数。例如 300 波特表示每秒钟发送 300 个 bit。当我们提到时钟周期时，我们就是指波特率例如如果协议需要 4800 波特率，那么时钟是 4800Hz。这意味着串口通信在数据线上的采样率为 4800Hz。通常电话线的波特率为 14400，28800 和 36600。波特率可以远远大于这些值，但是波特率和距离成反比。高波特率常常用于放置的很近的仪器间的通信，波特率除数（baud-rate divisor）是一个 22 位数，它由 16 位整数和 6 位小数组成。波特率发生器使用这两个值组成的数字来决定位周期。通过带有小数波特率的除法器，在足够高的系统时钟速率下，UART 可以产生所有标准的波特率，而误差很小。

波特率除数公式：

$$BRD = BRDI.BRDF = SystemClock / (16 \times BaudRate)$$

其中：

BRD 是 22 位的波特率除数，由 16 位整数和 6 位小数组成

BRDI 是 BRD 的整数部分

BRDF 是 BRD 的小数部分

SystemClock 是系统时钟（UART 模块的时钟直接来自 SystemClock）

BaudRate 是波特率（9600，38400，115200 等）

b，数据位：这是衡量通信中实际数据位的参数。当计算机发送一个信息包，实际的数据不会是 8 位的，标准的值是 5、7 和 8 位。如何设置取决于你想传送的信息。比如，标准的 ASCII 码是 0～127（7 位）。扩展的 ASCII 码是 0～255（8 位）。如果数据使用简单的文本（标准 ASCII 码），那么每个数据包使用 7 位数据。每个包是指一个字节，包括开始/停止位，数据位和奇偶校验位。由于实际数据位取决于通信协议的选取，术语“包”指任何通信的情况。

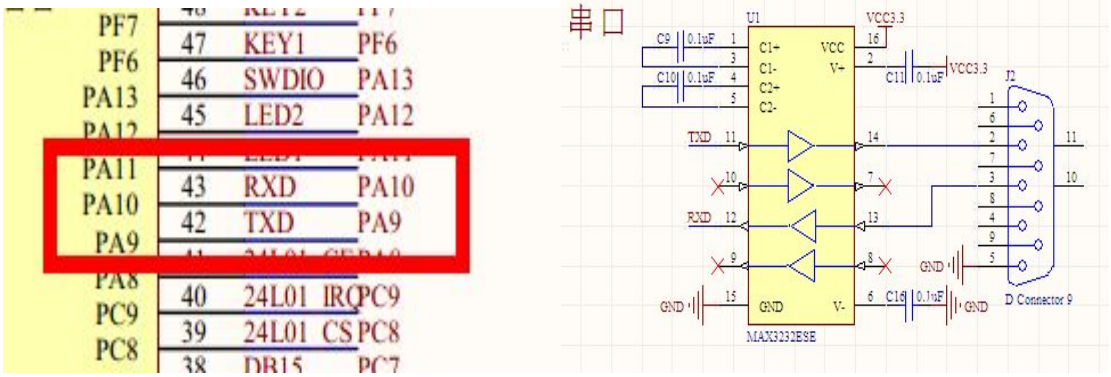
c，停止位：用于表示单个包的最后一位。典型的值为 1，1.5 和 2 位。由于数据是在传输线上定时的，并且每一个设备有其自己的时钟，很可能在通信中两台设备间出现了小小的不同步。因此停止位不仅仅是表示传输的结束，并且提供计算机校正时钟同步的机会。适用于停止位的位数越多，不同时钟同步的容忍程度越大，但是数据传输率同时也越慢。

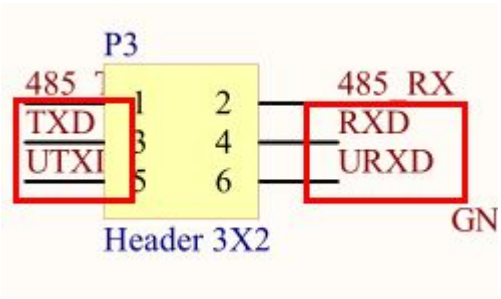
d，奇偶校验位：在串口通信中一种简单的检错方式。有四种检错方式：偶、奇、高和低。当然没有校验位也是可以的。对于偶和奇校验的情况，串口会设置校验位（数据位后面的一位），用一个值确保传输的数据有偶个或者奇个逻辑高位。例如，如果数据是 011，那么对于偶校验，校验位为 0，保证逻辑高的位数是偶数个。如果是奇校验，校验位为 1，这样就有 3 个逻辑高位。高位和低位不真正的检查数据，简单置位逻辑高或者逻辑低校验。

下面来看下 STM32F030 是如何实现串口功能的：

2.10.2 硬件准备：

Stm32f030 开发板串口连接入下图所示：





由于开发板 485 和 232 公用一个端口，因此使用串口 232 时，首先需要把 P3 跳线跳到串口 232 模式，此时：

RXD--PA10 接收数据线　　TXD--PA9 发送数据线

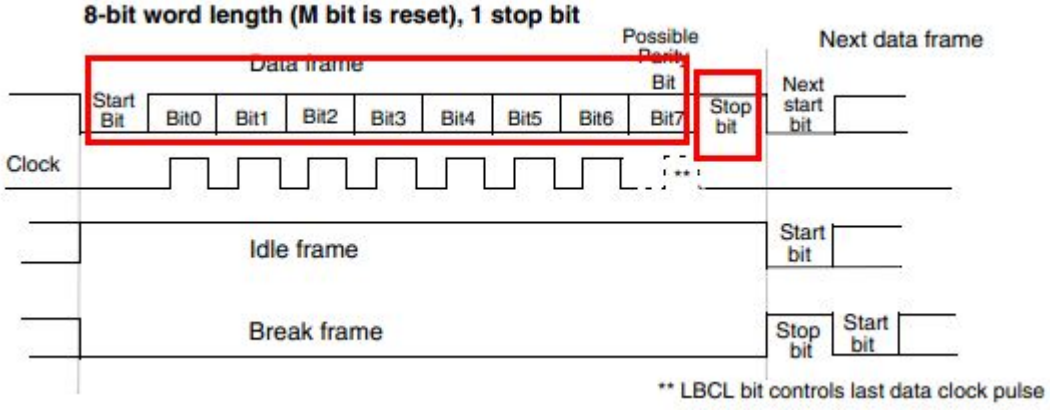
2.10.3 软件准备：

软件配置采用直接调用 stm 库函数，工程配置入下图所示，开发者只需要编写 UART.C 子驱动函数就可以在 main 中直接调用了：

串口的库函数 stm32f0xx_usart.c 文件中，用于配置串口的关键参数被写成一个结构体的形式,如下代码所示：

```
01. typedef struct
02. {
03.    uint32_t USART_BaudRate;       //串口波特率
04.    uint32_t USART_WordLength;     //数据位宽
05.    uint32_t USART_StopBits;       //停止位宽
06.    uint32_t USART_Parity;         //效验位宽
07.    uint32_t USART_Mode;           //工作模式
08.    uint32_t USART_HardwareFlowControl; //流控制
09. } USART_InitTypeDef;
```

上面的结构体设置了串口通信的几个数据流参数，包括传输方式：单工，全双工，半双工,下图为 8BIT 数据位的帧格式：



在 uart.c 驱动函数中，时钟串口初始化可以按下面来设置，当然在这之前需要设置串口所使用的 IO 端口，由于 IO 端口属于复用功能，采用 GPIO_PinAFConfig 进行复用设置，复用数可以参考 stm32f030r8 数据手册，如下图所示：

Table 14. Alternate functions selected through GPIOA_AFR registers for port A

Pin name	AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7
PA0		USART2_CTS	TIM2_CH1_ETR	TSC_G1_IO1				COMP1_OUT
PA1	EVENTOUT	USART2_RTS	TIM2_CH2	TSC_G1_IO2				
PA2	TIM15_CH1	USART2_TX	TIM2_CH3	TSC_G1_IO3				COMP2_OUT
PA3	TIM15_CH2	USART2_RX	TIM2_CH4	TSC_G1_IO4				
PA4	SPI1_NSS/I2S1_WS	USART2_CK		TSC_G2_IO1	TIM14_CH1			
PA5	SPI1_SCK/I2S1_CK	CEC	TIM2_CH1_ETR	TSC_G2_IO2				
PA6	SPI1_MISO/I2S1_MCK	TIM3_CH1	TIM1_BKIN	TSC_G2_IO3		TIM16_CH1	EVENTOUT	COMP1_OUT
PA7	SPI1_MOSI/I2S1_SD	TIM3_CH2	TIM1_CH1N	TSC_G2_IO4	TIM14_CH1	TIM17_CH1	EVENTOUT	COMP2_OUT
PA8	MCO	USART1_CK	TIM1_CH1	EVENTOUT				
PA9	TIM15_BKIN	USART1_TX	TIM1_CH2	TSC_G4_IO1				
PA10	TIM17_BKIN	USART1_RX	TIM1_CH3	TSC_G4_IO2				

```
10.     GPIO_PinAFConfig(GPIOA,GPIO_PinSource9,GPIO_AF_1);//按上表所示，复用功为 1
11.     GPIO_PinAFConfig(GPIOA,GPIO_PinSource10,GPIO_AF_1);
12.     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9|GPIO_Pin_10;
13.     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF; //设置端口复用
14.     GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
15.     GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
16.     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
17.     GPIO_Init(GPIOA, &GPIO_InitStructure);
18.     USART_InitStructure.USART_BaudRate = 115200;//设置串口波特率
19.     USART_InitStructure.USART_WordLength = USART_WordLength_8b;//设置数据位
20.     USART_InitStructure.USART_StopBits = USART_StopBits_1;//设置停止位
21.     USART_InitStructure.USART_Parity = USART_Parity_No;//设置效验位
22.    USART_InitStructure.USART_HardwareFlowControl =
    USART_HardwareFlowControl_None;//设置流控制
23.     USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    //设置工作模式
24.     USART_Init(USART1, &USART_InitStructure); //配置入结构体
25.     USART_Cmd(USART1, ENABLE);//使能串口 1
```

那么上面的代码就对串口进行了初始化，归纳一下串口初始化的大体步骤，实际上其他的 API 功能采用库函数的话，都是相同的初始化方式：

1.初始化 IO 端口，使用外部 API 功能需要端口复用 GPIO_Mode_AF

2.设置 AF 的数，范围为 0—7；

3.设置 API 的相应参数，比如串口上面提到了参数结构体，ST 库函数已经配置好。

初始化后，需要些一些小的功能函数,包括串口直接的发送和接收，代码可以配置如下：

```
26.  void UART_send_byte(uint8_t byte) //发送 1 字节数据
```

```
{
  while(!((USART1->ISR)&(1<<7)));//等待发送完
  USART1->TDR=byte; //发送一个字节
}

void UART_Send(uint8_t *Buffer, uint32_t Length)
{
        while(Length != 0)
        {
                while(!((USART1->ISR)&(1<<7)));//等待发送完
                USART1->TDR= *Buffer;
                Buffer++;
                Length--;
        }
}
uint8_t UART_Recive(void)
{
        while(!(USART1->ISR & (1<<5)));//等待接收到数据
        return(USART1->RDR);                    //读出数据
}
```

由于使用寄存器配置功能函数比较简单，我直接使用寄存器 ISR,TDR 和 RDR 进行编写，其中 ISP 表示串口状态标志位，其中第 7 位和第 5 位如下表所示：

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	Res	Res	Res	Res	Res	Res	Res	Res	REACK	TEACK	WUF	RWU	SBKF	CMF	BUSY
									r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ABRF	ABRE	Res	EOBF	RTOF	CTS	CTSIF	LBDF	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
r	r		r	r	r	r	r	r	r	r	r	r	r	r	r

TXE 表示发送数据是否为空标志位，取 0 表示数据没有转换到移位寄存器内，取 1 表示已经转移，为空，此时就可以判断是否发送完成，可以表示为：!((USART1->ISR)&(1<<7))。

RXNE 表示接收寄存器是否为空，取 0 表示没有读取数据，取 1 表示接收数据，此时可以表示为：!((USART1->ISR)&(1<<7))。

TDR 和 RDR 分别表示接收数据寄存器和发送数据寄存器，分别只取低 8 位。

那么用户需要编写的串口驱动子函数已经完成，这部分很硬件是密切相关的，我们在写主函数的时候只需要调用上面的驱动就可以进行数据的发送和接收了。下面主函数就演示一个串口发送数据的例子，代码如下：

```
  int main(void)
{
/* USART1 config 115200 8-N-1 */
  USART_Configuration();//首先配置串口的数据参数，包括工作方式和帧格式，IO 端口的复用
  while (1)
  {
```

```
53.         UART_Send( Tx_Buffer, countof(Tx_Buffer)-1);//然后发送一个数组
54.             delay(10000000);
55.     }
56. }
```

实验现象：

1.下载：D:\青风 STM32F0\源代码\实验十：usart 串口\user 代码进入开发板内。

2.打开串口调试助手，设置串口参数。

3.按下开发板的复位键，此时串口就可以接收到 CPU 发送的数据信息了，如下图所示：