串口的基本使用

RT-Thread 评估板 RealTouch 裸机例程

版本号：1.0.0

日期：2012/8/8

修订记录

日期	作者	修订历史
2012/8/8	Heyuanjie87	添加例程说明

串口

串口在嵌入式系统中是一个非常重要的外设，它通信方式简单在软件开发阶段常用作调试工具。本示例中我们只实现串口的输出功能，同时还会实现一个具有 printf 功能输出接口。这样咱们以后的例程中就有了一个简单的调试工具。

1. STM32 串口简介

STM32 的串口功能非常丰富，它可以支持双全工异步通信、LIN、IrDA、智能卡协议、单线半双工通信、支持调制解调器操作。

接下来我们将对使用 STM32 的串口应该进行的哪些设置给予简单说明，并对需要设置的寄存器给予简单介绍。要使用串口除了应对串口的波特率等进行配置外还需要对串口用到 I/O 进行设置，下面将分步进行介绍：

1） 串口时钟使能

STM32 可以对每个外设进行单独的时钟控制，因此配置串口前需要打开串口的时钟。开启时钟前查看一下具体外设在哪个总线上是非常有必要的，通过查看 STM32 的参考手册第一章的 1.2 节得知串口 1 和 6（USART1 和 6）在 APB2 上其它串口都在 APB1 上。

2） 设置波特率

串口波特率的设置是在 USART_BRR 中配置的，寄存器描述如下：

波特率配置寄存器各位描述

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV_Mantissa												DIV_Fraction			

STM32 的波特率发生器可以使用分数作为分频值，DIV_Fraction 表示小数部分 DIV_Mantissa 表示整数部分。分频值 USARTDIV = DIV_Mantissa + DIV_Fraction / （8 * （2 - OVER8））。波特率 Baud = APBxCLK / (8 * (2 - OVER8) * USARTDIV)。其中 OVER8 是 USART_CR1 中的一位取值 0 或 1（当 OVER8 为 1 时只有 DIV_Fraction[2:0]有效，DIV_Fraction[3]应设置为 0）。举例说明一下这个分频值的计算：假设 OVER8 为 0，USART_BRR 为 0x1BC 则 Mantissa = 27，Fraction = 12。分频值为 27+12/16 = 27.75。

3） 串口控制

串口控制寄存器 1 各位描述

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OVER8	-	UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	TCIE	TE	RE	RWU	SBK

STM32 共有三个控制寄存器在这里我们仅对控制寄存器 1 中要用到的位给予说明。OVER8 是设置过采样，1 表示 8 倍过采样 0 表示 16 倍过采样。UE 为串口使能。M 为字长选择，0 表示 8 位数据位 1 表示 9 位数据位。PCE 为校验控制使能，1 表示开启校验 0 表示无需校验。PS 为校验选择，0 表示偶校验 1 表示奇校验。PEIE 为校验中断使能。TE 为发送使能，1 表示允许发送。

4） 数据传输

STM32 的数据发送与接收都是通过寄存器 USART_DR 来实现的。它是一个包含发送（TDR）和接收（RDR）的 32 位双功能寄存器，但只有 9 位有效位。

5） 串口状态

串口状态寄存器各位描述

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE

在这里我们只需关注 TXE、TC 这两个位。串口发送数据时是先把数据放在发送缓冲区，然后再放到移位寄存器中一位一位的将数据发送出去。TXE 为 1 时就表示发送缓冲区 TDR 为空，TC 为 1 则表示传输完成即移位寄存器中也没有数据。只有当 TXE 为 1 时我们才能将下一个数据写到 USART_DR 中。

2.本例程所用硬件资源介绍

本示例使用了串口 3（在扩展板上标有 UART 字样），PB10 对应发送引脚，PB11 对应接收引脚但本例程只使用了发送引脚。

3. 例程

代码 1　初始化串口

```
  /* uart.h */

  void uart_init(void)
  {
     USART_InitTypeDef USART_InitStructure;
     GPIO_InitTypeDef GPIO_InitStructure;

     /* 开启 GPIO_B 的时钟 */
     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
     /* 开启串口 3 的时钟 */
     RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3, ENABLE);

     GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AF;
     GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
     GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_UP;
     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
     GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_10;

     GPIO_Init(GPIOB, &GPIO_InitStructure);

     /* 将 PB10 作为复用功能中的 USART3 通信引脚使用 */
     GPIO_PinAFConfig(GPIOB, GPIO_PinSource10, GPIO_AF_USART3);


     USART_InitStructure.USART_BaudRate   = 115200;
     USART_InitStructure.USART_WordLength = USART_WordLength_8b;
     USART_InitStructure.USART_StopBits   = USART_StopBits_1;
     USART_InitStructure.USART_Parity    = USART_Parity_No;
     USART_InitStructure.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;
     USART_InitStructure.USART_Mode  = USART_Mode_Tx;

              USART_Init(USART3, &USART_InitStructure);

     /* 使能串口 3 */
     USART_Cmd(USART3, ENABLE);
  }
```

这里要注意的就是引脚配置为复用功能模式后，还需要选择具体使用哪个复用功能，其余的初始化比较简单就没什么说的了。

代码 2　发送数据

```
 /* uart.c */

static void _send(const char *str, unsigned int size)
{
    int pos = 0;

    while(size)
    {
        if (str[pos] == '\0')
            break;

        /* 等待缓冲区空 */
        while(!(USART3->SR & 0x80));
        /* 发送数据 */
        USART3->DR = str[pos];

        pos ++;
        size --;
    }
}

void debug(const char* fmt,...)
{
    va_list ap;
    char string[65];

    string[64]='\0';
    va_start(ap,fmt);
    vsprintf(string,fmt,ap);
    va_end(ap);
    _send(string,64);
}
```

在这里我实现了一个可以格式输出的接口在以后的例程中可以把它作为调试工具了。