

Lab4 串口实验

Notice: The English Version will be ready before class on Sep 24.

一、实验目的

1. 了解嵌入式系统开发中常用的硬件功能模块和硬件开发工具；
2. 熟练建立一个完整的工程，掌握嵌入式系统开发流程；
3. 了解串口功能模块的基本原理和开发应用。

二、实验内容

串口作为 MCU 的重要外部接口，同时也是软件开发重要的调试手段。本实验课采用的 ALIENTEK MiniSTM32 开发板最多可提供 5 路串口，有分数波特率发生器、支持同步单线通信和半双工单线通讯、支持 LIN、支持调制解调器操作、智能卡协议和 IrDA SIR ENDEC 规范、具有 DMA 等。

串口功能设置的一般步骤可以总结为如下几个步骤：

- 1) 串口时钟使能，GPIO 时钟使能
- 2) 串口复位
- 3) GPIO 端口模式设置
- 4) 串口参数初始化

5) 开启中断并且初始化 NVIC

6) 使能串口

7) 编写中断处理函数

与串口基本配置直接相关的几个固件库函数和定义主要分布在 `stm32f10x_usart.h` 和 `stm32f10x_usart.c` 文件中。

1. 串口时钟使能

串口是挂载在 APB2 下面的外设，所以使能函数为：

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1);
```

2. 串口复位

当外设出现异常的时候可以通过复位设置，实现该外设的复位，然后重新配置这个外设达到让其重新工作的目的。一般在系统刚开始配置外设的时候，都会先执行复位该外设的操作。复位的是在函数 `USART_DeInit()` 中完成：

```
void USART_DeInit(USART_TypeDef* USARTx); //串口复位
```

比如我们要复位串口 1，方法为：

```
USART_DeInit(USART1); //复位串口 1
```

3. 串口参数初始化

串口初始化是通过 `USART_Init()` 函数实现的，

```
void USART_Init(USART_TypeDef* USARTx, USART_InitTypeDef* USART_InitStruct );
```

这个函数的第一个入口参数是指定初始化的串口标号，这里选

择 USART1。

第二个入口参数是一个 USART_InitTypeDef 类型的结构体指针，这个结构体指针的成员变量用来设置串口的一些参数。一般的实现格式为：

```
USART_InitStructure.USART_BaudRate = bound; //波特率;
USART_InitStructure.USART_WordLength = USART_WordLength_8b; //字长为 8 位
数据格式
USART_InitStructure.USART_StopBits = USART_StopBits_1; //一个停止位
USART_InitStructure.USART_Parity = USART_Parity_No; //无奇偶校验位
USART_InitStructure.USART_HardwareFlowControl
= USART_HardwareFlowControl_None; //无硬件数据流控制
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx; //收发模
式
USART_Init(USART1, &USART_InitStructure); //初始化串口
```

4. 数据发送与接收

STM32 的发送与接收是通过数据寄存器 USART_DR 来实现的，这是一个双寄存器，包含了 TDR 和 RDR。当向该寄存器写数据的时候，串口就会自动发送，当收到收据的时候，也是存在该寄存器内。

STM32 库函数操作 USART_DR 寄存器发送数据的函数是：

```
void USART_SendData(USART_TypeDef* USARTx, uint16_t Data);
```

通过该函数向串口寄存器 USART_DR 写入一个数据。

STM32 库函数操作 USART_DR 寄存器读取串口接收到的数据的函数是：

```
uint16_t USART_ReceiveData(USART_TypeDef* USARTx);
```

通过该函数可以读取串口接受到的数据。

5. 串口状态

串口的状态可以通过状态寄存器 `USART_SR` 读取。`USART_SR` 的各位描述如下图：



RXNE（读数据寄存器非空），当该位被置 1 的时候，就是提示已经有数据被接收到了，并且可以读出来了。这时候我们要做的就是尽快去读取 `USART_DR`，通过读 `USART_DR` 可以将该位清零，也可以向该位写 0，直接清除。

TC（发送完成），当该位被置位的时候，表示 `USART_DR` 内的数据已经被发送完成了。如果设置了这个位的中断，则会产生中断。该位也有两种清零方式：1）读 `USART_SR`，写 `USART_DR`。2）直接向该位写 0。

在我们固件库函数里面，读取串口状态的函数是：

```
FlagStatus USART_GetFlagStatus(USART_TypeDef* USARTx, uint16_t USART_FLAG);
```

这个函数的第二个入口参数非常关键，它是标示要查看串口的哪种状态，比如上面讲解的 **RXNE**(读数据寄存器非空)以及 **TC**(发送完成)。例如要判断读寄存器是否非空(**RXNE**)，操作库函数的方法是：

```
USART_GetFlagStatus(USART1, USART_FLAG_RXNE);
```

要判断发送是否完成(TC)，操作库函数的方法是：

```
USART_GetFlagStatus(USART1, USART_FLAG_TC);
```

这些标识号在 MDK 里面是通过宏定义定义的：

```
#define USART_IT_PE ((uint16_t)0x0028)
#define USART_IT_TXE ((uint16_t)0x0727)
#define USART_IT_TC ((uint16_t)0x0626)
#define USART_IT_RXNE ((uint16_t)0x0525)
#define USART_IT_IDLE ((uint16_t)0x0424)
#define USART_IT_LBD ((uint16_t)0x0846)
#define USART_IT_CTS ((uint16_t)0x096A)
#define USART_IT_ERR ((uint16_t)0x0060)
#define USART_IT_ORE ((uint16_t)0x0360)
#define USART_IT_NE ((uint16_t)0x0260)
#define USART_IT_FE ((uint16_t)0x0160)
```

6. 串口使能

串口使能是通过函数 `USART_Cmd()`来实现的，这个很容易理解，使用方法是：

```
USART_Cmd(USART1, ENABLE); //使能串口
```

7. 开启串口响应中断

有些时候当我们还需要开启串口中断，那么我们还需要使能串口中断，使能串口中断的函数是：

```
void USART_ITConfig(USART_TypeDef* USARTx, uint16_t USART_IT,
FunctionalState NewState)
```

这个函数的第二个入口参数是标示使能串口的类型，也就是使能

哪种中断，因为串口的中断类型有很多种。比如在接收到数据的时候（RXNE 读数据寄存器非空），我们要产生中断，那么我们开启中断的方法是：

```
USART_ITConfig(USART1, USART_IT_RXNE, ENABLE); //开启中断，接收到数据中断
```

我们在发送数据结束的时候（TC，发送完成）要产生中断，那么方法是：

```
USART_ITConfig(USART1, USART_IT_TC, ENABLE);
```

8. 获取相应中断状态

当我们使能了某个中断的时候，当该中断发生了，就会设置状态寄存器中的某个标志位。经常我们在中断处理函数中，要判断该中断是哪种中断，使用的函数是：

```
ITStatus USART_GetITStatus(USART_TypeDef* USARTx, uint16_t USART_IT)
```

比如我们使能了串口发送完成中断，那么当中断发生了，我们便可以在中断处理函数中调用这个函数来判断到底是否是串口发送完成中断，方法是：

```
USART_GetITStatus(USART1, USART_IT_TC)
```

返回值是 SET，说明是串口发送完成中断发生。

三、实验前准备

1. 先导要求

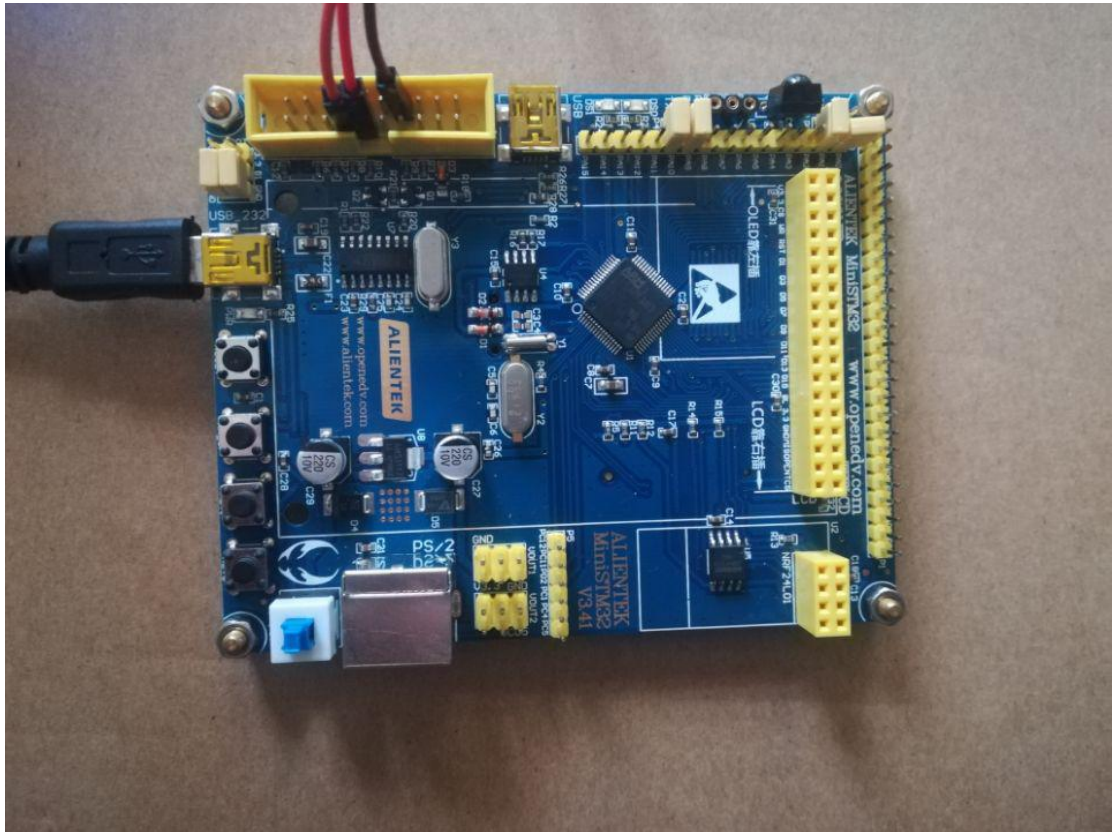
熟练掌握 Keil5 Project 建立及软硬件调试测试的开发流程。（希望

大家逐步熟悉，不要每次都要对照文档才会建立工程)

2. 硬件准备

(1) J-Link or STLink 连接;

(2) USB-COM 连接: 连接 USB-232 端口, 如下图:



3. 软件及环境准备

(1) USB 串口驱动安装, CH340 驱动(USB 串口驱动), 请注意电脑的操作系统版本, 可自行从网上下载安装。

































(2) 串口调试器, 若系统没有自带 COM 调试器, 推荐 SSCOM。

四、实验过程

1. 工程建立

具体过程略。

注意：设置运行环境需要选择：CMSIS->CORE, Device->Startup, StdPeriph Drivers->Framework, GPIO,RCC,USART【比之前增加了 usart】。

	CMSIS		
	CORE	<input checked="" type="checkbox"/>	3.40.0
	DSP	<input type="checkbox"/>	1.4.2
	RTOS (API)		1.0
	CMSIS Driver		
	Device		
	DMA	<input type="checkbox"/>	1.2
	GPIO	<input checked="" type="checkbox"/>	1.3
	Startup	<input checked="" type="checkbox"/>	1.0.0
	StdPeriph Drivers		
	ADC	<input type="checkbox"/>	3.5.0
	BKP	<input type="checkbox"/>	3.5.0
	CAN	<input type="checkbox"/>	3.5.0
	CEC	<input type="checkbox"/>	3.5.0
	CRC	<input type="checkbox"/>	3.5.0
	DAC	<input type="checkbox"/>	3.5.0
	DBGMCU	<input type="checkbox"/>	3.5.0
	DMA	<input type="checkbox"/>	3.5.0
	EXTI	<input type="checkbox"/>	3.5.0
	FSMC	<input type="checkbox"/>	3.5.0
	Flash	<input type="checkbox"/>	3.5.0
	Framework	<input checked="" type="checkbox"/>	3.5.1
	GPIO	<input checked="" type="checkbox"/>	3.5.0
	I2C	<input type="checkbox"/>	3.5.0
	IWDG	<input type="checkbox"/>	3.5.0
	PWR	<input type="checkbox"/>	3.5.0
	RCC	<input checked="" type="checkbox"/>	3.5.0
	RTC	<input type="checkbox"/>	3.5.0
	SDIO	<input type="checkbox"/>	3.5.0
	SPI	<input type="checkbox"/>	3.5.0
	TIM	<input type="checkbox"/>	3.5.0
	USART	<input checked="" type="checkbox"/>	3.5.0


2. 主要功能源码


```
#include "led.h"
#include "delay.h"
#include "sys.h"
#include "usart.h"
//实验（四）串口实验
int main(void)
{
    u8 t;
    u8 len;
    u16 times=0;
    delay_init();           //延时函数初始化
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2); // 设置中断优先级分组 2
    uart_init(9600);        //串口初始化为 9600
    LED_Init();             //初始化与 LED 连接的硬件接口
    while(1)
    {
        if(USART_RX_STA&0x8000)
        {
            len=USART_RX_STA&0x3fff; //得到此次接收到的数据长度
            printf("\r\n 您发送的消息为:\r\n");
            for(t=0;t<len;t++)
            {
                USART1->DR=USART_RX_BUF[t];
                while((USART1->SR&0X40)==0); //等待发送结束
            }
            printf("\r\n\r\n"); //插入换行
            USART_RX_STA=0;
        }else
        {
            times++;
            if(times%5000==0)
            {
                printf("\r\n 实验（四）串口实验\r\n");
                printf("CSE@SUSTech\r\n\r\n\r\n\r\n");
            }
            if(times%1000==0) printf("请输入数据,以回车键结束\r\n");
            if(times%30==0) LED0=!LED0; //闪烁 LED,提示系统正在运行.
            delay_ms(10);
        }
    }
}
```

3. 编译及下载到开发板

略。

4. 串口调试

打开 SSCOM 软件，选择串口号（不同机器串口号可能不同，可通过我的电脑-设备管理器查看  USB-SERIAL CH340 (COM5) ），选择波特率（与源码中要求的相同，默认 9600），打开串口，选择发送新行。然后在字符串输入下的编辑框中输入内容，回车或点击“发送”即可。



五、常见问题

1. 工程编译报错类

内容略。可参考之前实验的相关文档。

2. 程序 load

问题：串口实验编译 load 成功后，在调试助手中无消息显示，led 灯闪烁。

疑似原因：先在调试助手中打开串口，再 load，实际 load 失败，led 灯闪烁实则之前实验的效果。

测试过程：在初始化后 while(1)前，加入某个 led 灯常亮设置（其他地方不要对 led 进行控制），load 后虽然 keil 中提示成功，但 led 不亮，说明实际 load 不成功。

解决：load 之后再打开调试助手，或者关闭串口连接再 load，load 成功后再打开串口。

3. 串口调试助手

（1） 启动软件报错或提示找不到串口

原因：计算机自身无串口，且未安装 USB-COM 驱动，或安装驱动后从未连接过串口连接线。

辅助现象：在我的电脑“设备管理器”中无“端口（COM 和 LPT）”项。

解决：安装驱动，并正确连接开发板。

（2） writefile function failed win error code 6

原因：未打开串口，试图在显示区域内输入。

解决：打开串口，在字符输入框内输入字符串。

（3） 发送按钮为灰色

原因：未打开串口

（4） 输入字符串，点击发送或回车无反应

原因：未选择“发送新行”；

补充：由于样例程序要求以回车结束输入，未输入回车（新行）视为输入尚未结束，不会在显示窗口输出内容。

（5） 输出内容为乱码

原因：调试助手设置的波特率与开发板中波特率要求的不一致

六、作业

1. 功能 1 实现：从串口调试助手输入自己的姓名，返回“Hello, xxx”。
2. 功能 2 实现：从串口调试助手输入命令：led0 on/led0 off/led1 on/led1 off 四种命令，可控制 led 灯的亮灭。
3. 功能 3 实现：按键输入时，串口调试助手输出提示“XX pressed.”
4. 作业提交：包括 main.c 主要代码，及功能 1 调试成功并运行结果截图，DDL: Oct 9.
5. 作业检查：功能 2、3 运行状态。