STM32学习笔记

1. 基础：关于STM32

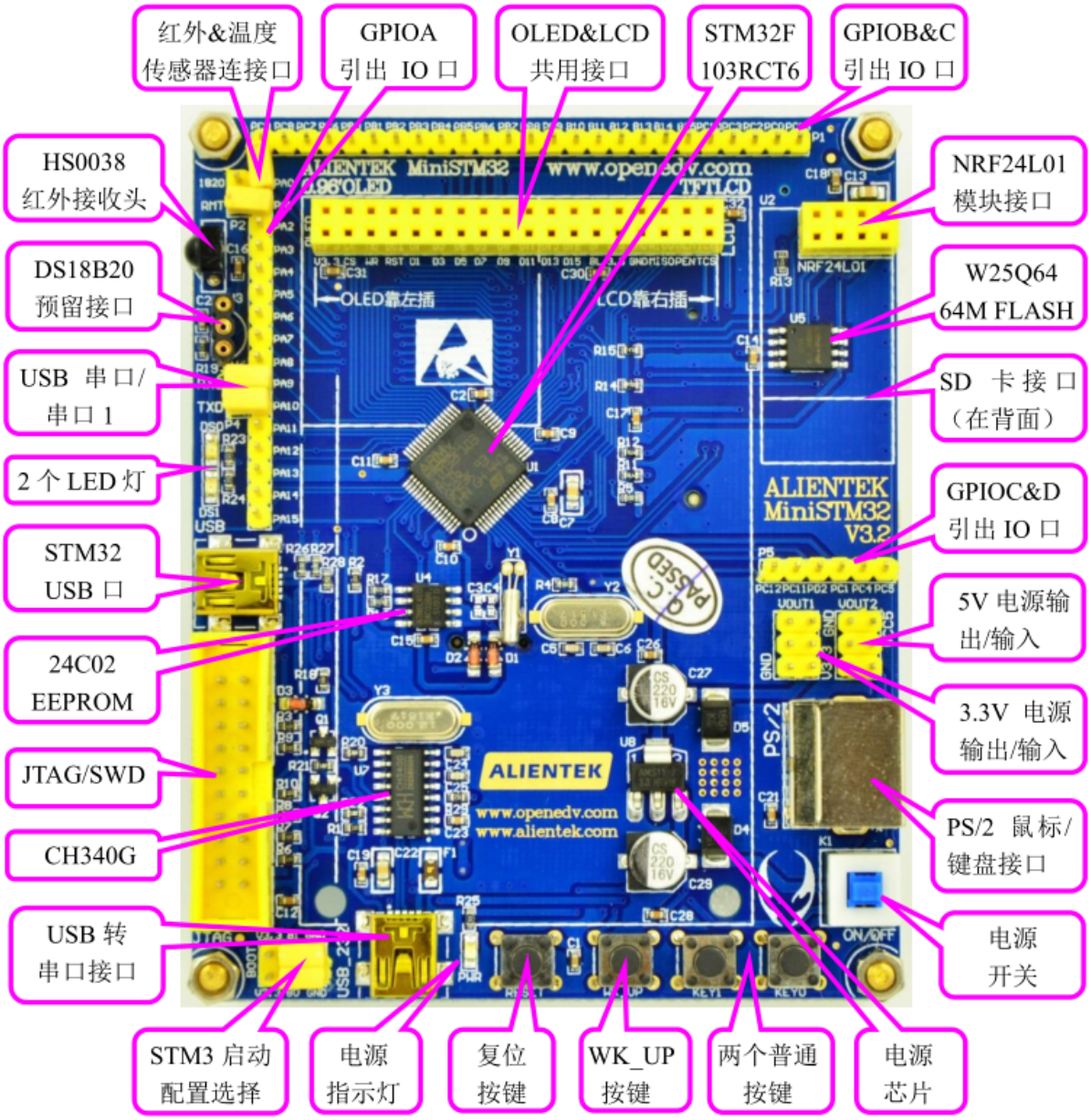

图 1.1.1 MiniSTM32 开发板资源图

1.1 开发板资源图与串口介绍

- 1 个电源指示灯（蓝色），2 个状态指示灯（`DS0`：红色，`DS1`：绿色）。
- 1 组 5V 电源供应/接入口，1 组 3.3V 电源供应/接入口。
- 1 个复位按钮，可用于复位 `MCU` 和 `LCD`，3 个功能按钮，其中 `WK_UP` 兼具唤醒功能。
- 除晶振占用的 IO 口外，其余所有 IO 口全部引出，其中`GPIOA` 和 `GPIOB` 按顺序引。
- **USB 转串口接口**：用于 USB 连接 CH340G 芯片，从而实现 USB 转串口，所以串口下载代码的时候，USB 一定是要接在这个口上的。此 MiniUSB接头也是开发板电源的主要提供口。

> 引脚分配说明：见引脚EXCEL说明

1.2 开发板原理图及设置使用方法

> ALIENTEK MiniSTM32 V3.0 版开发板选择的是 STM32F103RCT6 作为 MCU，它拥有的资源包括：48KB SRAM、256KB FLASH、2 个基本定时器、4 个通用定时器、2 个高级定时器、2 个 DMA 控制器（共 12 个通道）、3 个 SPI、2 个 IIC、5 个串口、1 个 USB、1 个 CAN、3 个12 位 ADC、1 个 12 位 DAC、1 个 SDIO 接口及 51 个通用 IO 口。

- 启动模式

BOOT0	BOOT1	启动模式	说明
0	X	用户闪存存储器	用户闪存存储器，也就是FLASH启动
1	0	系统存储器	系统存储器启动，用于串口下载
1	1	SRAM启动	SRAM启动，用于在SRAM中调试代码

1.3 开发板使用注意事项

- **当你想使用某个 IO 口用作其他用处的时候，请先看看开发板的原理图，该 IO 口是否有连接在开发板的某个外设上**，如果有，该外设的这个信号是否会对你的使用造成干扰，先确定无干扰，再使用这个 IO。比如 PA0 如果和 1820 的跳线帽连接上了，那么WK_UP 按键就无法正常检测了，按键实验，也就没法做了。

1.4 软件安装与工程模板

- MDK5 集成开发环境
  - 新建文件夹Tempalte, 创建子文件夹`USER，CORE，STM32F10X，OBJ，SYSTEM，HAEDWARE及其子文件夹`。
  - 打开Keil，点击`Project->New uVersion Project，选择STM32F103RC`
  - 点击`Manage Project Items`,创建相应的文件夹
  - 添加启动代码，即`add files->.c`文件
  - 把生成的文件存放到OBJ文件夹下`Options for Target->Output->Create HEX file，Select Folder for Objects->选择OBJ文件夹`
  - 添加头文件：`Options->C/C++，添加.h文件，必须到最后一级文件夹`

2. 库函数：跑马灯实验

2.1 STM32 IO介绍

- STM32 的 IO 口可以由软件配置成如下 8 种模式
  1. 输入浮空
  2. 输入上拉
  3. 输入下拉
  4. 模拟输入
  5. 开漏输出
  6. 推挽输出
  7. 推挽式复用功能
  8. 开漏复用功能
- STM32 的每个 IO 端口都有 7 个寄存器来控制。我们常用的 IO 端口寄存器只有 4 个：**CRL、CRH、IDR、ODR**。CRL 和 CRH 控制着每个 IO 口的模式及输出速率。

2.2 跑马灯硬件设计

> 本章用到的硬件只有 LED（DS0 和 DS1）。其电路在 ALIENTEK MiniSTM32 开发板上默认是已经连接好了的。DS0 接 PA8，DS1 接 PD2。所以只需要操作PA8和PD2两个IO口即可。

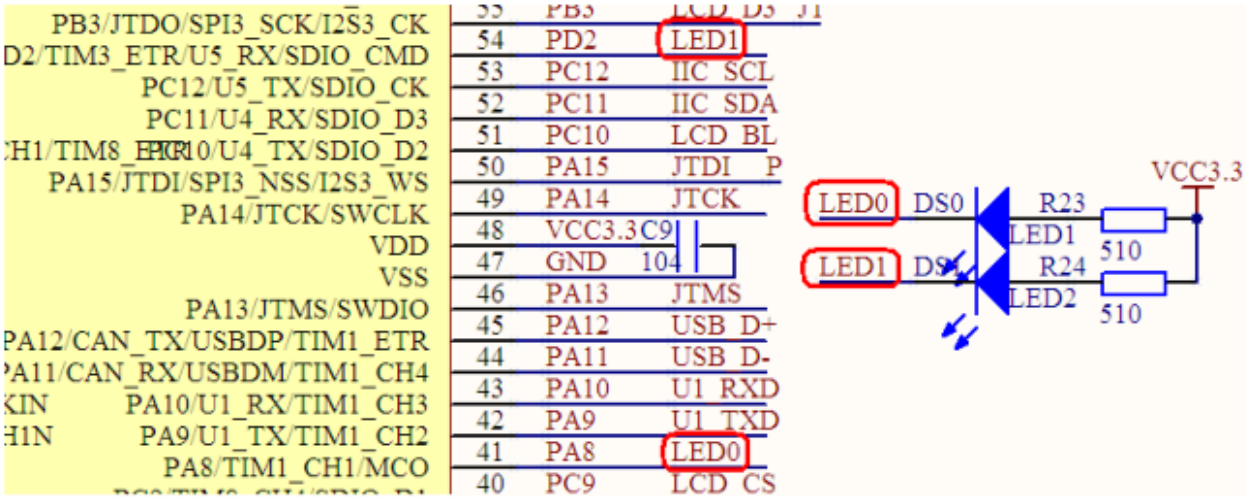

图 6.2.1 LED 与 STM32 连接原理图

2.3 软件设计

- 怎样通过固件库设置 GPIO的相关参数和输出？

```
void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct);
//通过初始化结构体初始化 GPIO 的常用格式
GPIO_InitTypeDef GPIO_InitStructure;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;  //LED0-->PB.5 端口配置
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; //推挽输出
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//速度 50MHz
GPIO_Init(GPIOB, &GPIO_InitStructure);//根据设定参数配置 GPIO

//在固件库中操作 IDR 寄存器读取 IO 端口数据是通过 GPIO_ReadInputDataBit 函数实现的：
uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);

void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);  //置为高电平
void GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin); //置为低电平
```

- 跑马灯代码

```
//LED initialize
#include "stm32f10x.h"
#include "led.h"
void LED_Init(void)
{
GPIO_InitTypeDef GPIO_InitStructure;
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|
RCC_APB2Periph_GPIOD, ENABLE); //使能 PA,PD 端口时钟
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8; //LED0-->PA.8 端口配置
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; //推挽输出
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz; //IO 口速度为 50MHz
GPIO_Init(GPIOA, &GPIO_InitStructure); //初始化 GPIOA.8
```

```
GPIO_SetBits(GPIOA,GPIO_Pin_8); //PA.8 输出高
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2; //LED1-->PD.2 端口配置，推挽输出
GPIO_Init(GPIOD, &GPIO_InitStructure); //推挽输出 ，IO 口速度为 50MHz
GPIO_SetBits(GPIOD,GPIO_Pin_2); //PD.2 输出高
//LED 头文件
#ifndef __LED_H
#define __LED_H
#include "sys.h"
#define LED0 PAout(8) // PA8 端口定义
#define LED1 PDout(2) // PD2
void LED_Init(void);//初始化
#endif
```

3. 串口实验

3.1 串口简介

- 串口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

   `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。

   `void USART_DeInit(USART_TypeDef* USARTx);//eg. USART_DeInit(USART1)`

3. GPIO 端口模式设置

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

   第一个参数为指定初始化的串口编号，第二个为USART_InitTypeDef类型的结构体：

   ```
   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. 串口参数初始化

   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。

   ```
   void USART_SendData(USART_TypeDef* USARTx, uint16_t Data);//发送数据
   uint16_t USART_ReceiveData(USART_TypeDef* USARTx);//接收数据
   ```

5. 开启中断并且初始化 NVIC（如果需要开启中断才需要这个步骤）

```
FlagStatus USART_GetFlagStatus(USART_TypeDef* USARTx, uint16_t USART_FLAG);
//如判断寄存器是否非空：USART_GetFlagStatus(USART1,USART_FLAG_RXNE);
//判断发送是否完成TC:USART_GetFlagStatus(USART1,USART_FLAG_TC);
```

6. 使能串口

`USART_Cmd(USART1, ENABLE);`

7. 编写中断处理函数

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

8. 获取相应中断状态

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

3.2 硬件设计

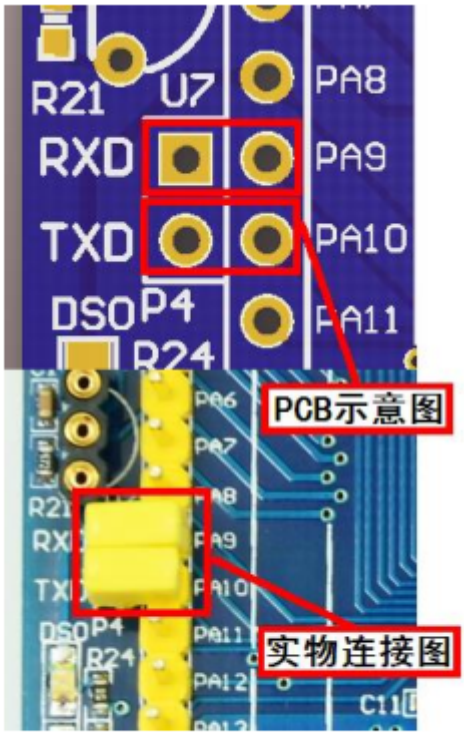

图 8.2.1 硬件连接图示意图

3.3 软件设计

> 串口初始化代码和接收代码在SYSTEM文件夹下，打开`usart.c`文件即可看到各个函数代码。

```
#include "sys.h"
#include "usart.h" //含有void usart_init(u32 bound)函数
#include "delay.h" //含有void delay_ms(u32 ms)等函数
```

```
#include "led.h"
int main(void)
{
u8 t;
u8 len;
u16 times=0;
delay_init();  //延时函数初始化
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2); //设置中断分组
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\nALIENTEK MiniSTM32 开发板 串口实验\r\n");
printf("正点原子@ALIENTEK\r\n\r\n\r\n");
}
if(times%200==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);
}
}
}
```

4. 外部中断实验

> 外部 IO 口的中断功能，代码主要分布在固件库的 `stm32f10x_exti.h` 和 `stm32f10x_exti.c` 文件中。

- 线 0~15：对应外部 IO 口的输入中断。线 16：连接到 PVD 输出。线 17：连接到 RTC 闹钟事件。线 18：连接到 USB 唤醒事件。
- GPIO 的管教 `GPIOx.0~GPIOx.15(x=A,B,C,D,E，F,G)` 分别对应中断线 15~0。
- 配置 GPIO 与中断线的映射关系的函数 GPIO_EXTILineConfig()来实现的： `void GPIO_EXTILineConfig(uint8_t GPIO_PortSource, uint8_t GPIO_PinSource)`
- 中断线上中断的初始化是通过函数 EXTI_Init()实现的。EXTI_Init()函数的定义是：

```
void EXTI_Init(EXTI_InitTypeDef* EXTI_InitStruct);

EXTI_InitTypeDef EXTI_InitStructure;
EXTI_InitStructure.EXTI_Line=EXTI_Line4;//中断线的标号，取值范围为EXTI_Line0~EXTI_Line15
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;//中断模式，可选值为中断
EXTI_Mode_Interrupt 和事件 EXTI_Mode_Event
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;//触发方式，可以是下降沿触发
EXTI_Trigger_Falling，上升沿触发 EXTI_Trigger_Rising
EXTI_InitStructure.EXTI_LineCmd = ENABLE;
EXTI_Init(&EXTI_InitStructure); //根据 EXTI_InitStruct 中指定的
//参数初始化外设 EXTI 寄存器
```