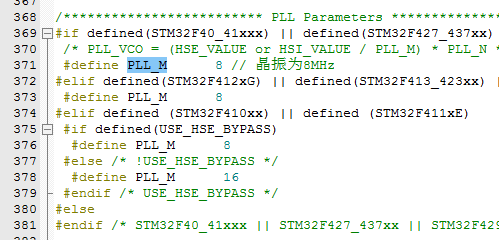
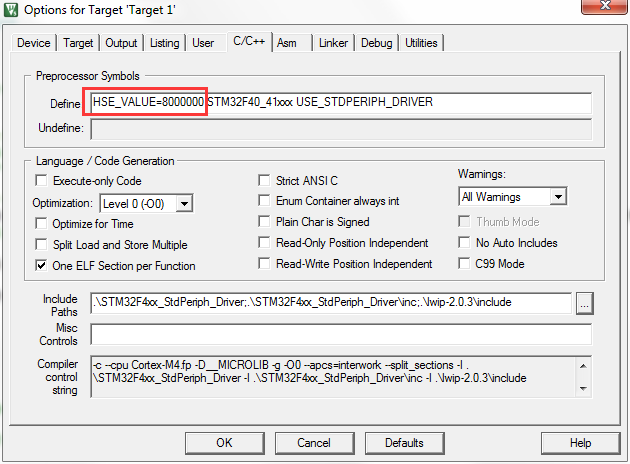
**88W8801程序从F1移植到F4的过程**

本程序是从[STM32F103ZE单片机驱动的88W8801程序（20180807版）](https://blog.csdn.net/ZLK1214/article/details/81489731)修改而来的。

1. 开发板用的是8MHz晶振。而system\_stm32f4xx.c里面默认的晶振大小为25MHz，所以必须将该文件中PLL\_M的值改为8。



在项目属性中，定义HSE\_VALUE宏，其值为8000000，表示晶振的大小。如果使用25MHz晶振，则不需要定义这个宏。



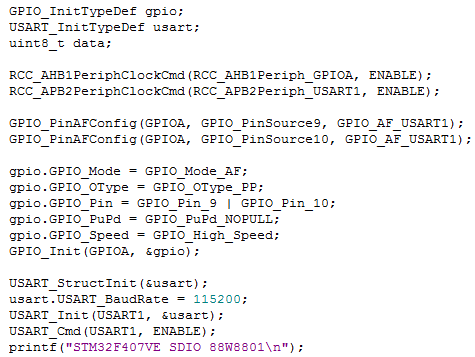
STM32F40\_41xxx宏确定了单片机的类型，USE\_STDPERIPH\_DRIVER表示项目用到了库函数。

值得注意的是，库函数压缩包STM32F4xx\_DSP\_StdPeriph\_Lib\_V1.8.0里面的Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\arm目录下有两个适用于STM32F407的启动文件：startup\_stm32f40\_41xxx.s和startup\_stm32f40xx.s。打开startup\_stm32f40xx.s文件后会看到里面有这样一句话：Same as startup\_stm32f40\_41xxx.s and maintained for legacy purpose。这说明，这两个文件实质上是一样的，用哪个都可以。

1. STM32F4单片机的头文件是stm32f4xx.h。



打开main.c文件，修改main函数中的GPIO和USART串口初始化代码。



1. 修改sys\_now函数的实现。STM32F1的RTC外设是一个32位的计数器，而STM32F4的RTC外设则是一个真正的日历，没有计数器。虽然可以用C库的mktime函数将日期转换为UNIX时间戳，然后再调用RTC\_GetSubSecond函数读取毫秒值，两者拼在一起形成毫秒计数值，但是这样做非常麻烦，效率也比较低。考虑到STM32F4单片机中有两个32位的定时器：TIM2和TIM5，所以程序选择使用TIM5实现sys\_now的功能。

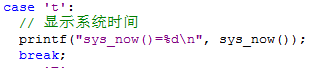
TIM5的时钟频率是84MHz，最大分频系数为65536。84000000÷65536≈1281.7>1000，光靠TIM5本身无法分频得到1kHz的频率，所以需要再级联一个TIM4定时器。TIM4先将84MHz分频为1MHz，然后接到TIM5上，再进行1000分频，最终得到1kHz的频率。



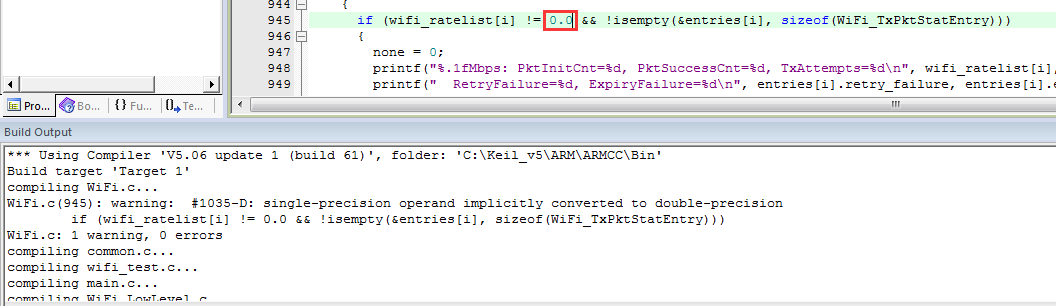
然后，在main函数中调用sys\_now\_init函数，初始化毫秒计数器：



wifi\_test.c中的display\_time函数演示了如何在STM32F1单片机下将32位的RTC秒计数器的值转换为日期时间格式，而STM32F4中的RTC有完整的日历功能，这个函数已经没有意义了，所以程序中删除了这个函数，并将main.c中串口命令t的作用改为了直接输出sys\_now毫秒计数器的值。

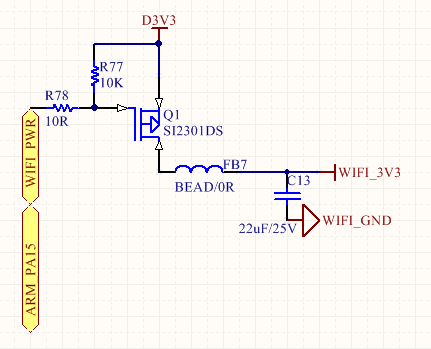


1. 为了避免在STM32F4环境下编译时出现的double转float浮点数导致精度丢失的警告，将WiFi.c中相应行的0.0改为0.0f。

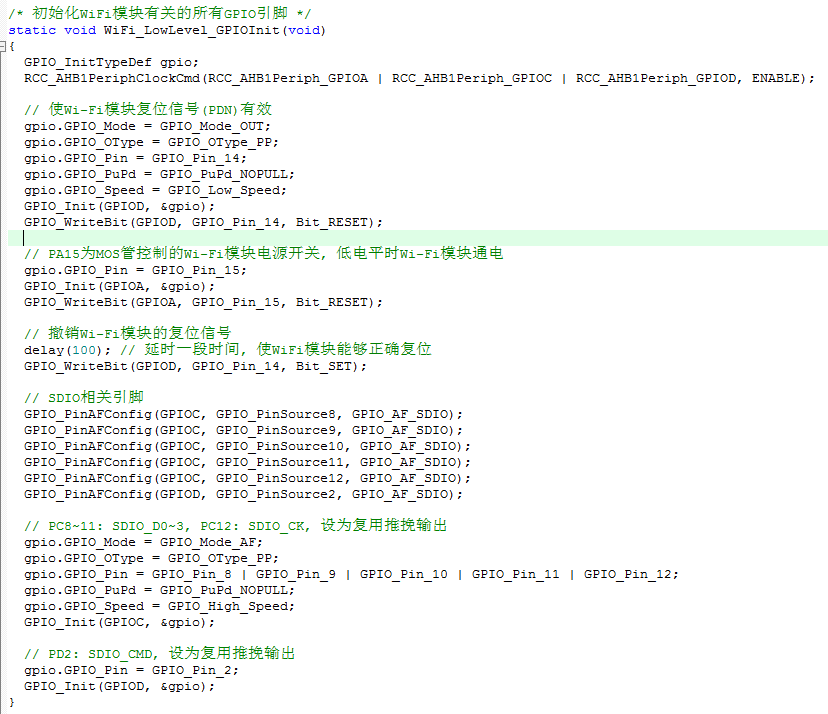


1. Wi-Fi模块引脚连接情况：CMD接PD2，CLK接PC12，**RESET引脚悬空，PDN（复位引脚）接PD14**。

笔者所用的开发板上，VCC不是直接接到电源上的，而是通过一个场效应管接到电源正极的，通断由PA15引脚控制，所以程序中将PA15设为了低电平，目的是使场效应管导通。**实际应用中，这个场效应管是可接可不接的，VCC可以直接接到电源上，复位由PDN引脚负责。**

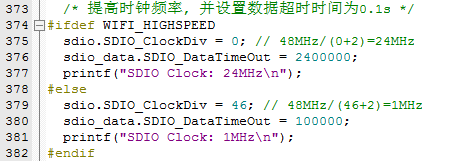


Wi-Fi模块GPIO引脚的初始化代码如下：



1. 修改SDIO接口的初始化函数WiFi\_LowLevel\_SDIOInit。





STM32F1单片机的SDIO频率是72MHz，而STM32F4中只有48MHz，所以在同样的频率下，需要使用不同的分频系数。

在STM32F4单片机中用DMA方式收发SDIO数据时，必须由SDIO（外设）来控制DMA的数据传输量，不能由dma.DMA\_BufferSize（SxNDTR寄存器）指定。否则，数据收发将不能正常进行，发送数据时会产生Underrun错误。

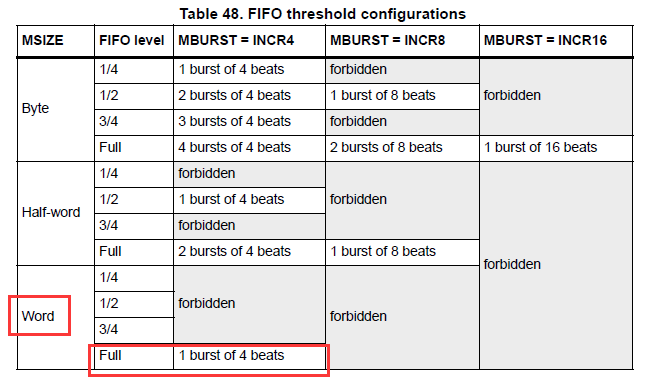
同时还需要注意，必须先执行SDIO\_DMACmd函数，后执行SDIO\_SetSDIOOperation函数，顺序不能颠倒，否则SDIO\_DMACmd有可能执行失败。

**这里要强调一下，在STM32F4单片机中，要使用SDIO外设，就必须打开PLL倍频器，即使系统时钟用的是HSI或HSE，没有用到PLL。不打开PLL倍频器，SDIO的时钟引脚根本就无法产生时钟信号。**

1. 修改数据发送函数WiFi\_LowLevel\_WriteData和数据接收函数WiFi\_LowLevel\_ReadData的DMA部分的代码。

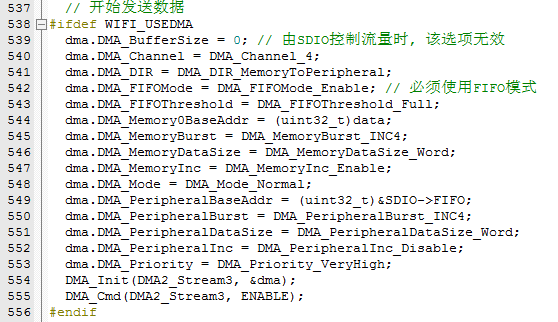
收发数据时，不能使用direct mode，必须使用FIFO mode，FIFO的阈值为Full。

根据STM32F4参考手册，当MSIZE=32位（Word）时，突发模式只能使用INC4，不能使用INC8或16的选项。

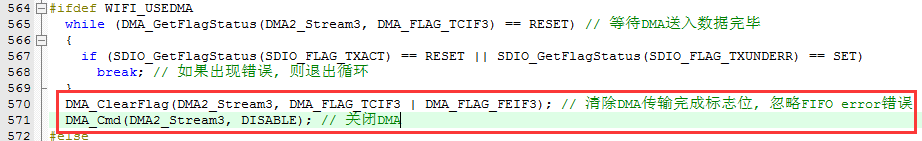


DMA必须设为由SDIO（外设）控制数据传输量，DMA\_BufferSize的值将被忽略。

DMA数据发送的配置如下：

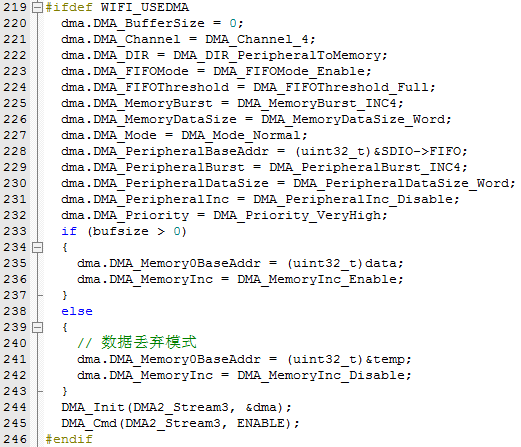


数据发送过程中产生的FIFO error错误属于正常现象，发送完毕后只需要将FEIF的标志位清除掉就行了。该错误的产生原因是DMA本身的FIFO暂时出现了underrun/overrun的错误，此时DMA不会对SDIO的DMA请求发出确认，这是一个可自恢复的、不会影响数据传输的错误。



虽然F4在数据发送成功后会自动关闭相应的DMA通道，但是为了考虑出错的情况，遇到某些错误时DMA通道不会自动关闭，程序中保留了关闭DMA的语句。

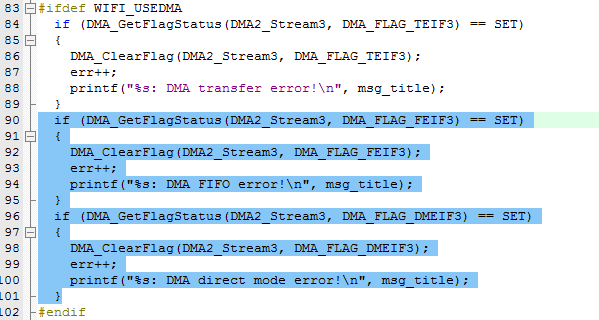
DMA数据接收的配置与发送是类似的：



只不过接收数据不会产生FIFO error错误，不需要清除这个标志位。



错误信息提示函数WiFi\_LowLevel\_CheckError需要添加F4新增的两个错误标志位。



1. 最后，修改lwip的NetBIOS计算机名和DHCP客户端名称（路由器管理页面中可看到）。

main函数中：



ethernetif\_init函数中：

