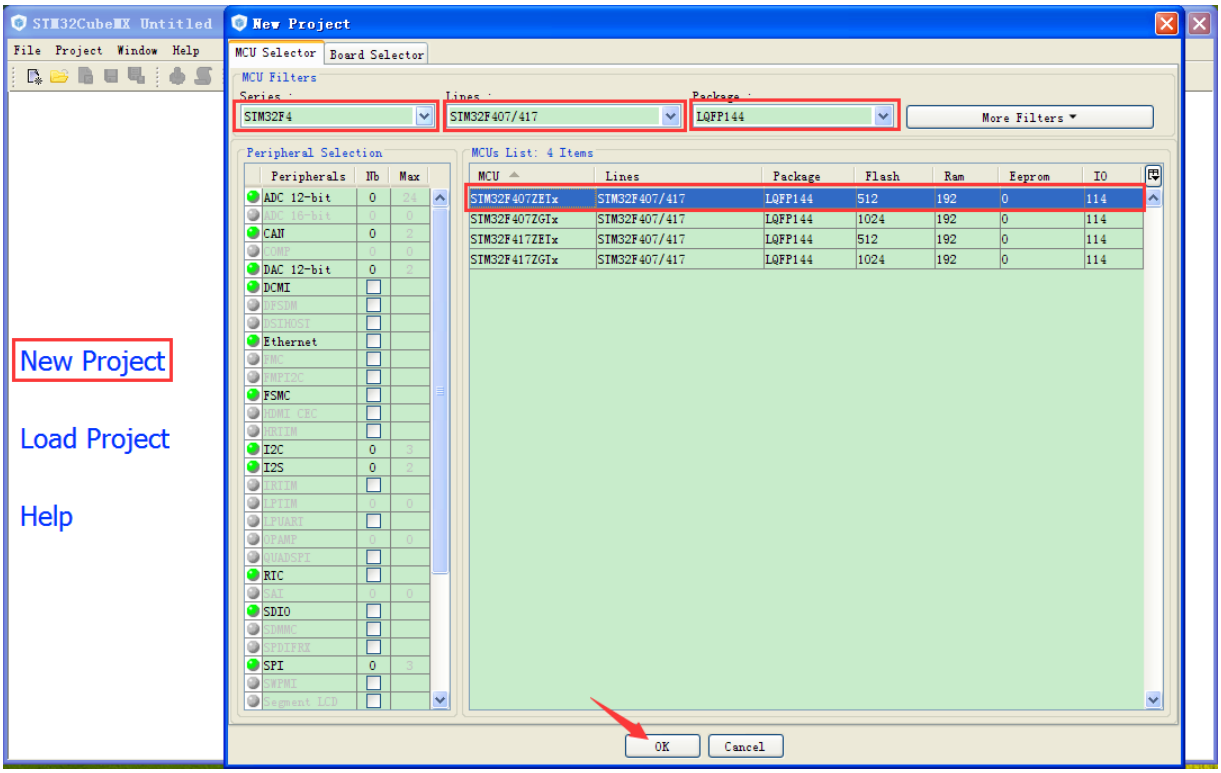
STM32Cube 学习之十四：SDIO+FATFS

前提：默认已经装好 MDK V5 和 STM32CubeMX。

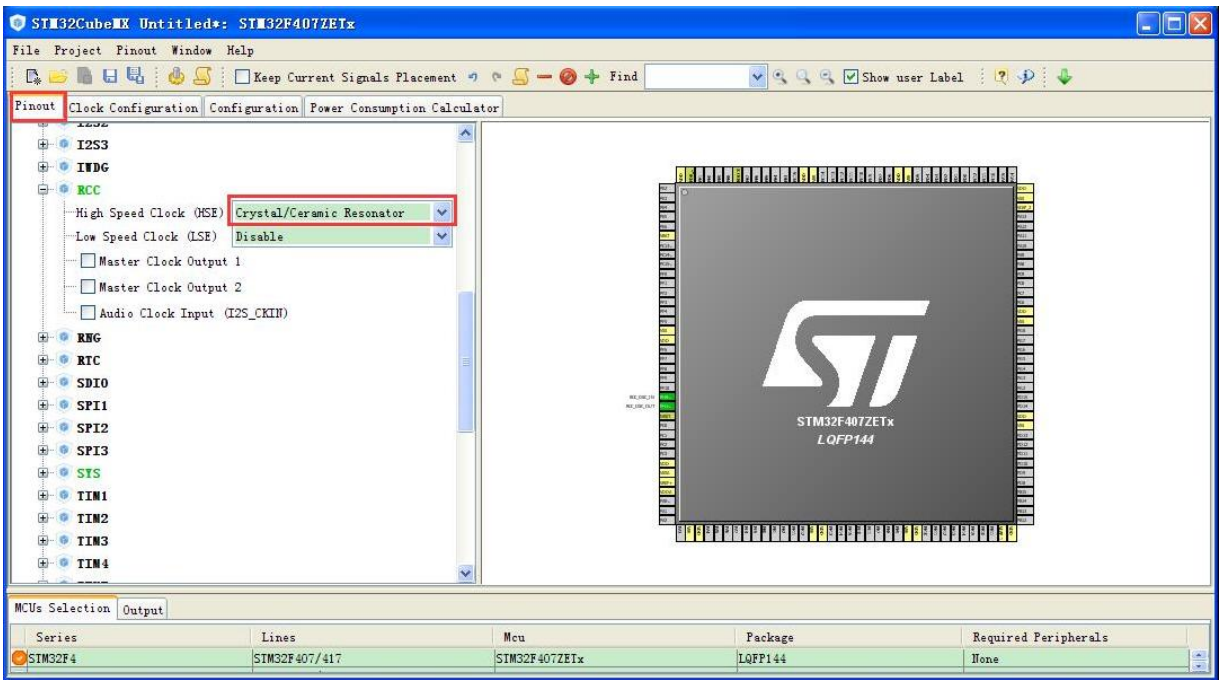
硬件平台：STM32F4xx 系列，并通过 SDIO 连接 SD 卡。

Step1.新建工程。

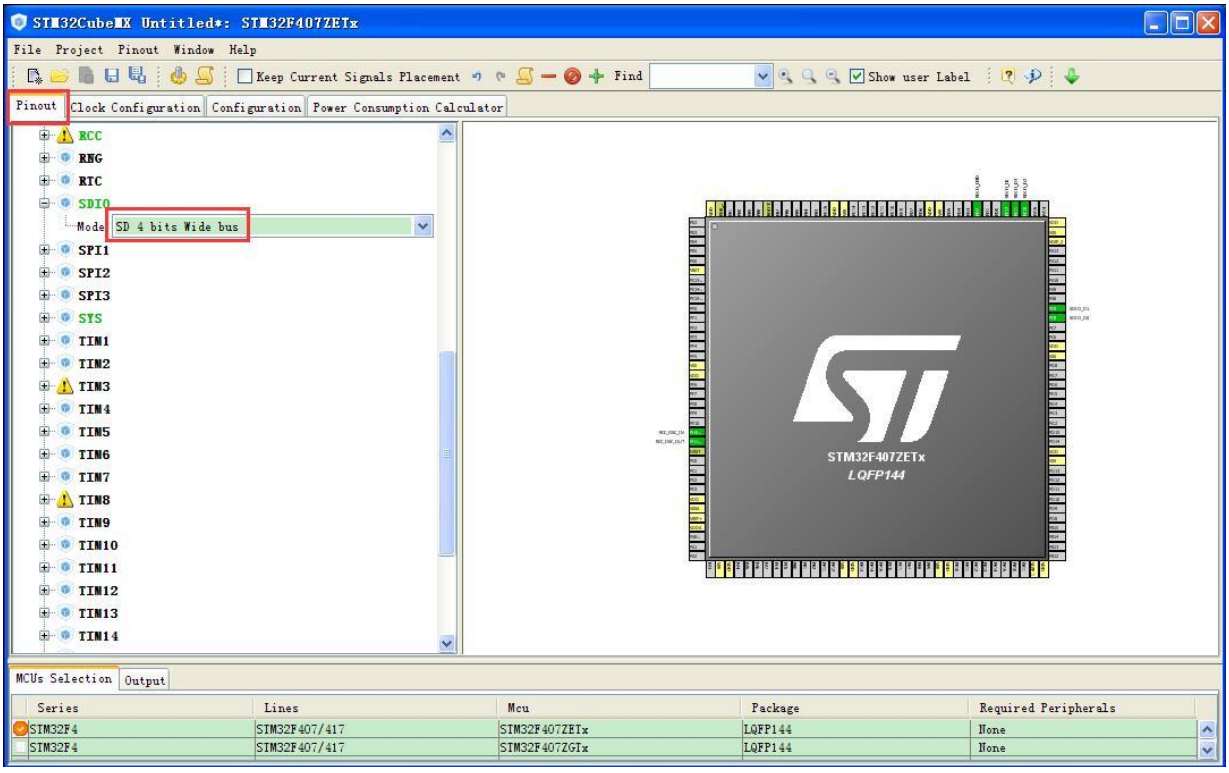
选择芯片型号。



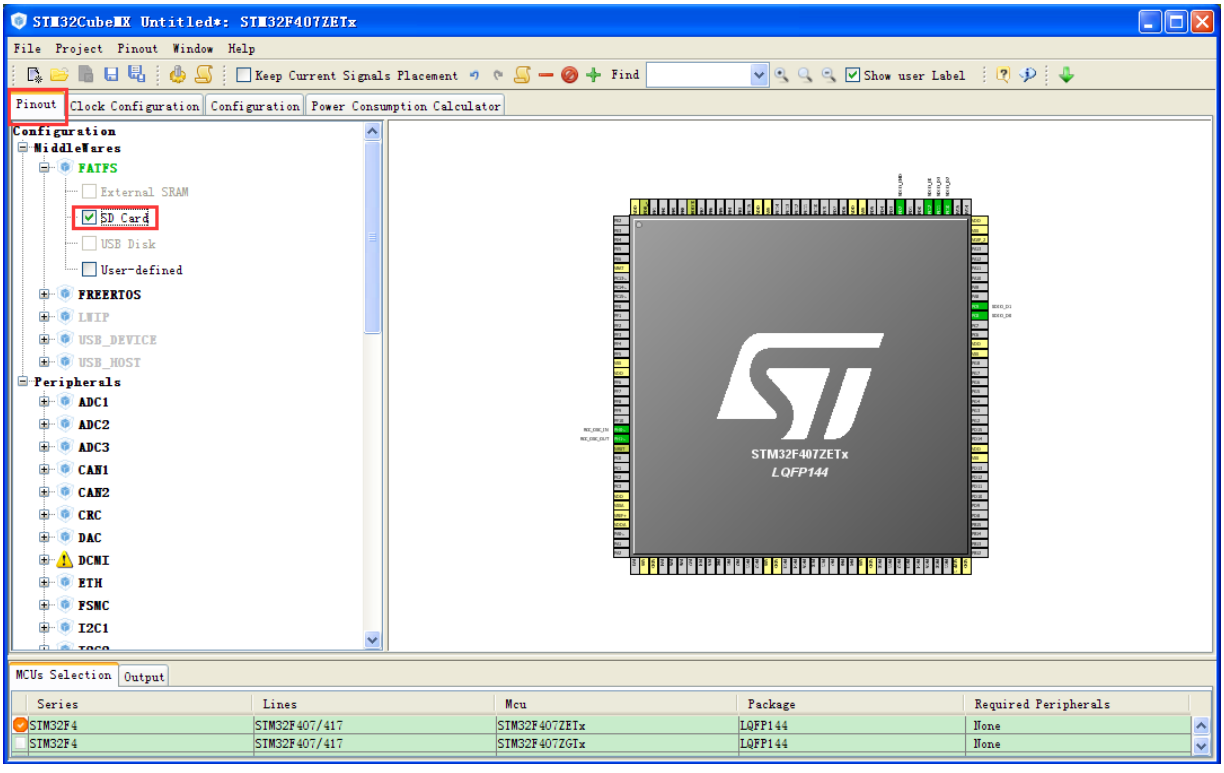
Step2.配置时钟引脚。



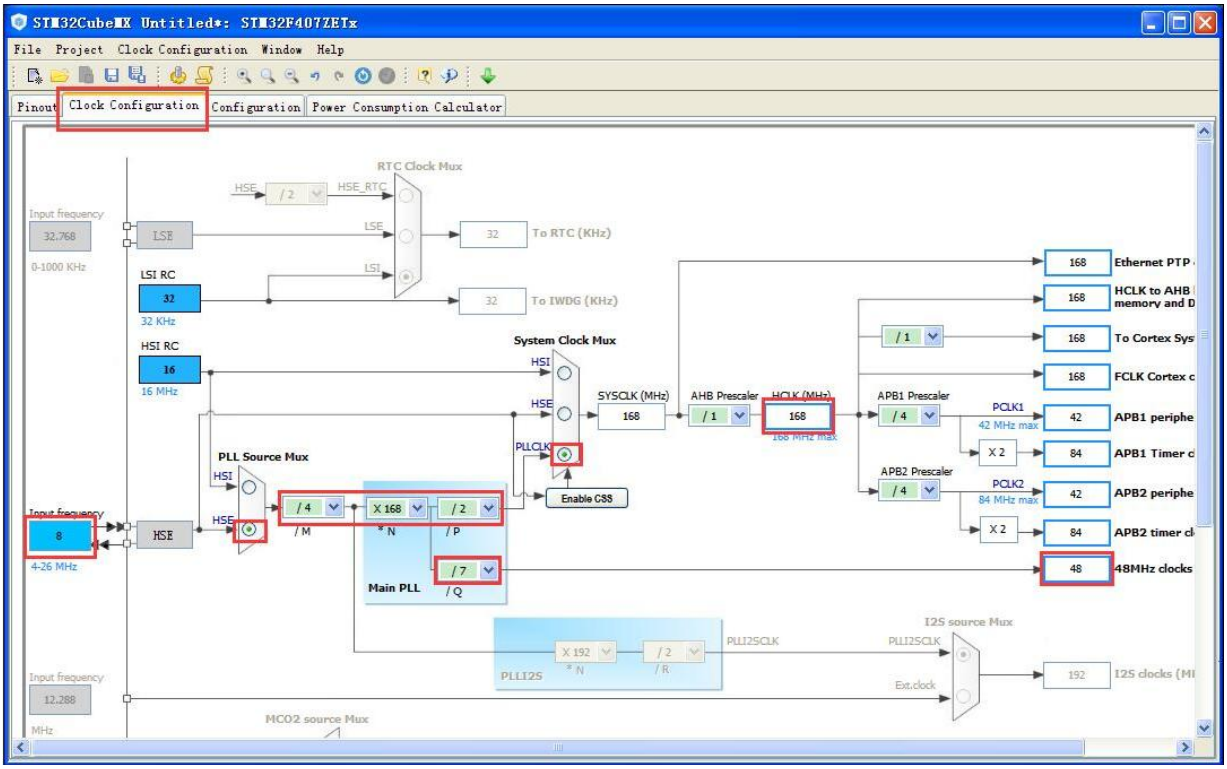
Step3.配置 SDIO 为 SD 4bit 宽度总线。



Step4.使用 FATFS 中间件。

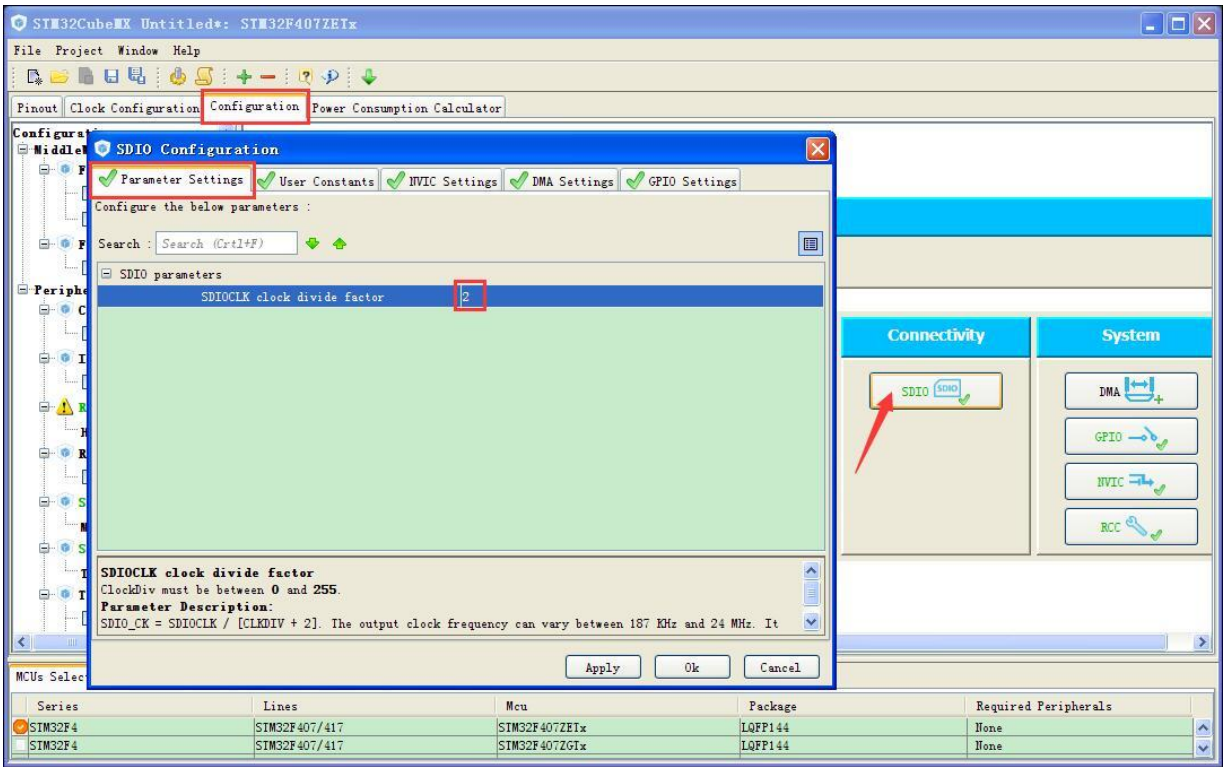


Step5.配置时钟树，SDIO 模块输入要求为 48MHz 时钟

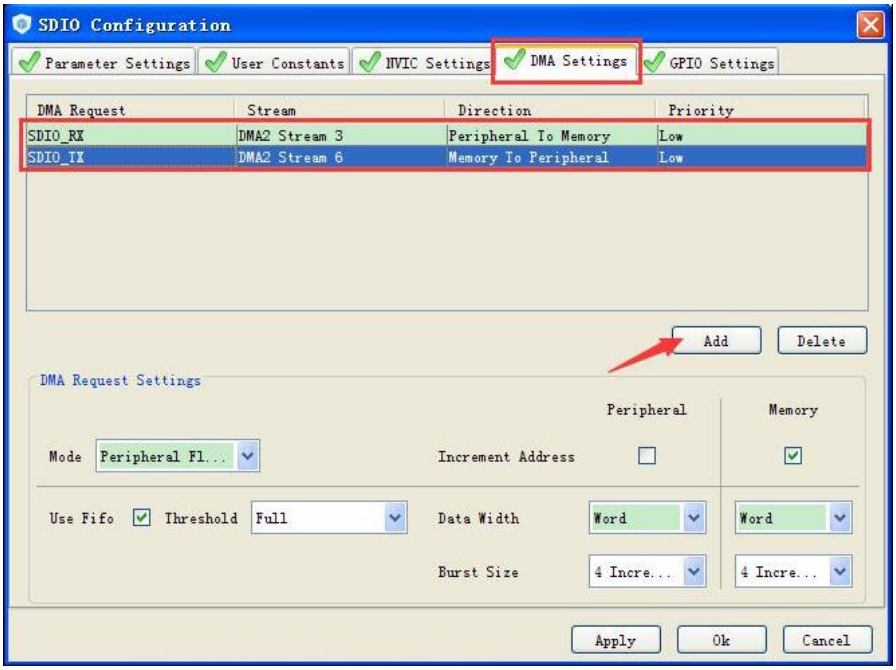


Step6.配置 SDIO 时钟分频系数 CLKDIV。计算公式为 SDIO_CK = 48MHz/(CLKDIV+2)。

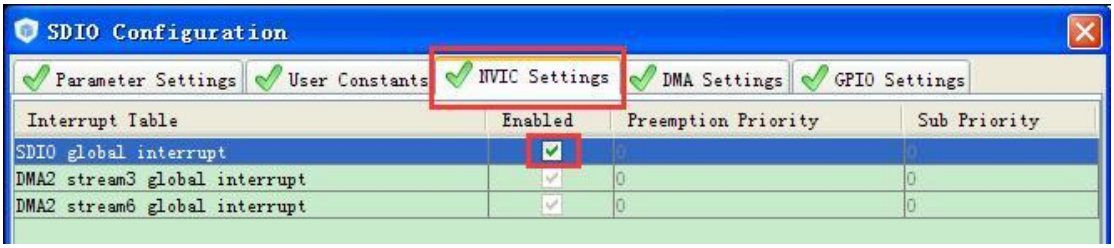
如下图，CLKDIV=2，则 SDIO 时钟 SDIO_CK=48MHz/(2+2)=12MHz。



Step7.使用 DMA 传输。



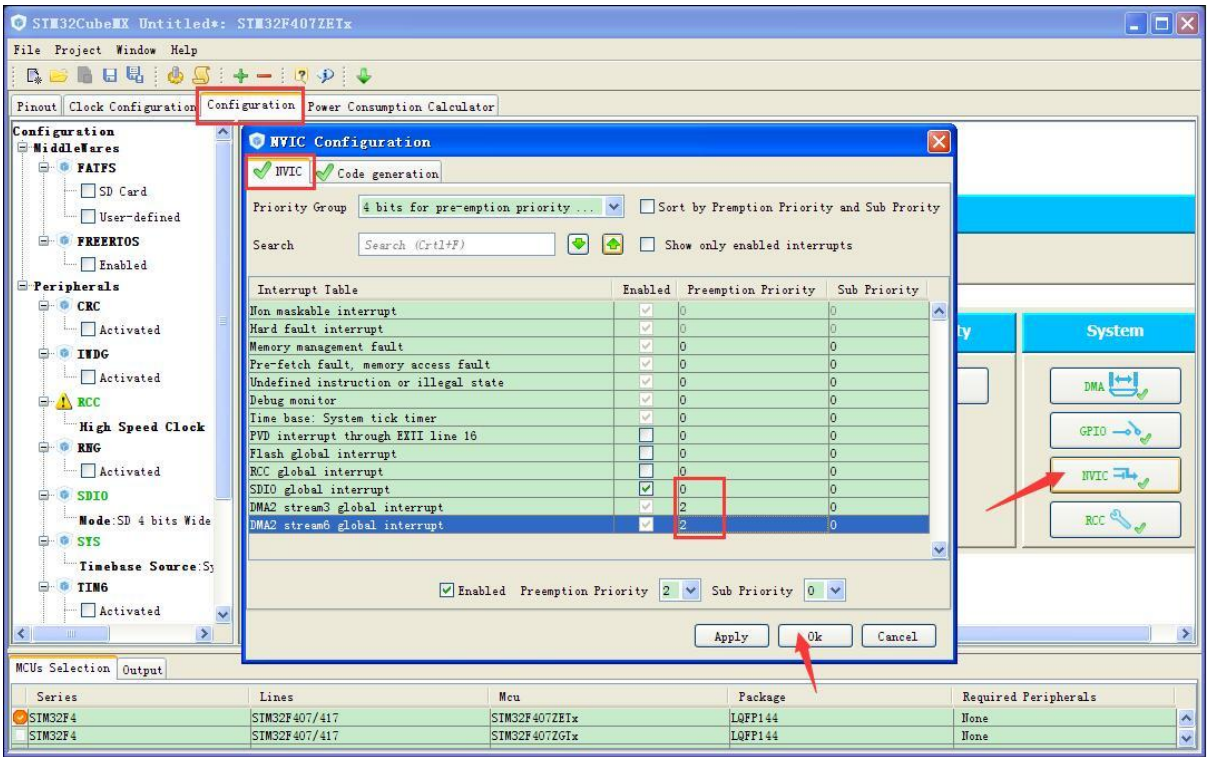
Step8.使能 SDIO 中断。



Step9.配置 NVIC。

注意，此处要求 SDIO 中断优先级必须高于 DMA2 stream3 和 DMA2 stream6 的中断优先级。

因此，将 DMA2 stream3 和 DMA2 stream6 的中断优先级都将为 2(或者更低)。

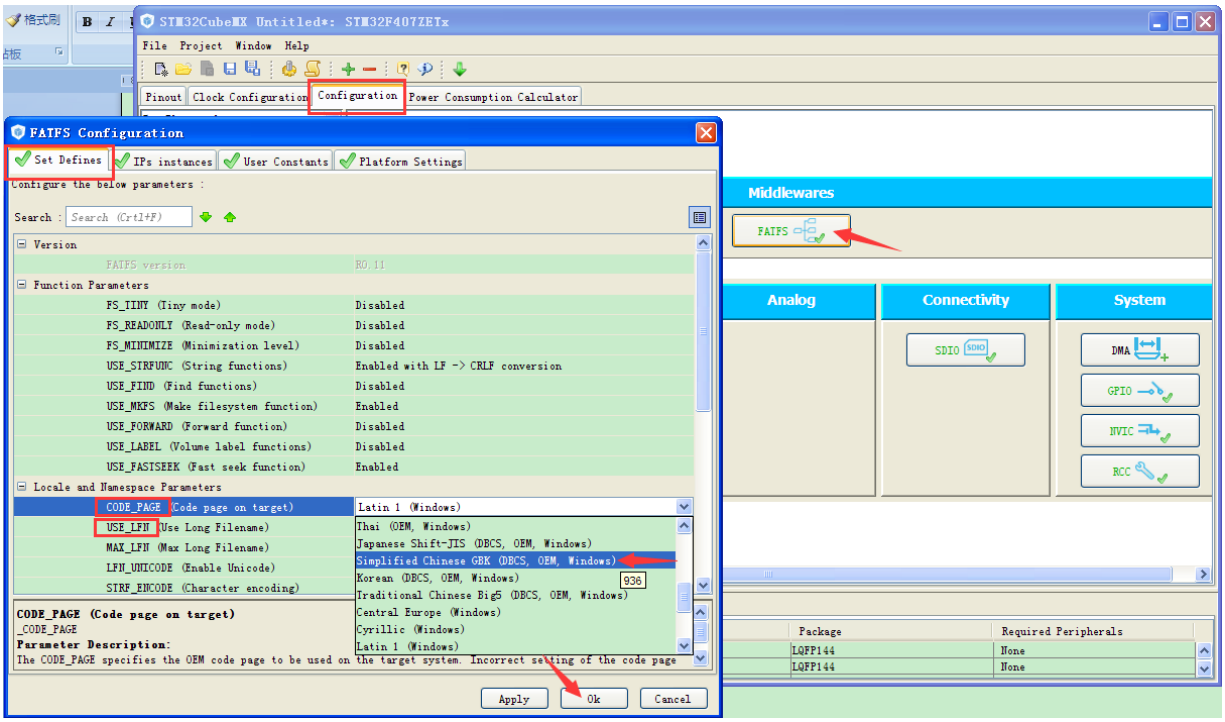


Step10.配置 FATFS 文件系统。

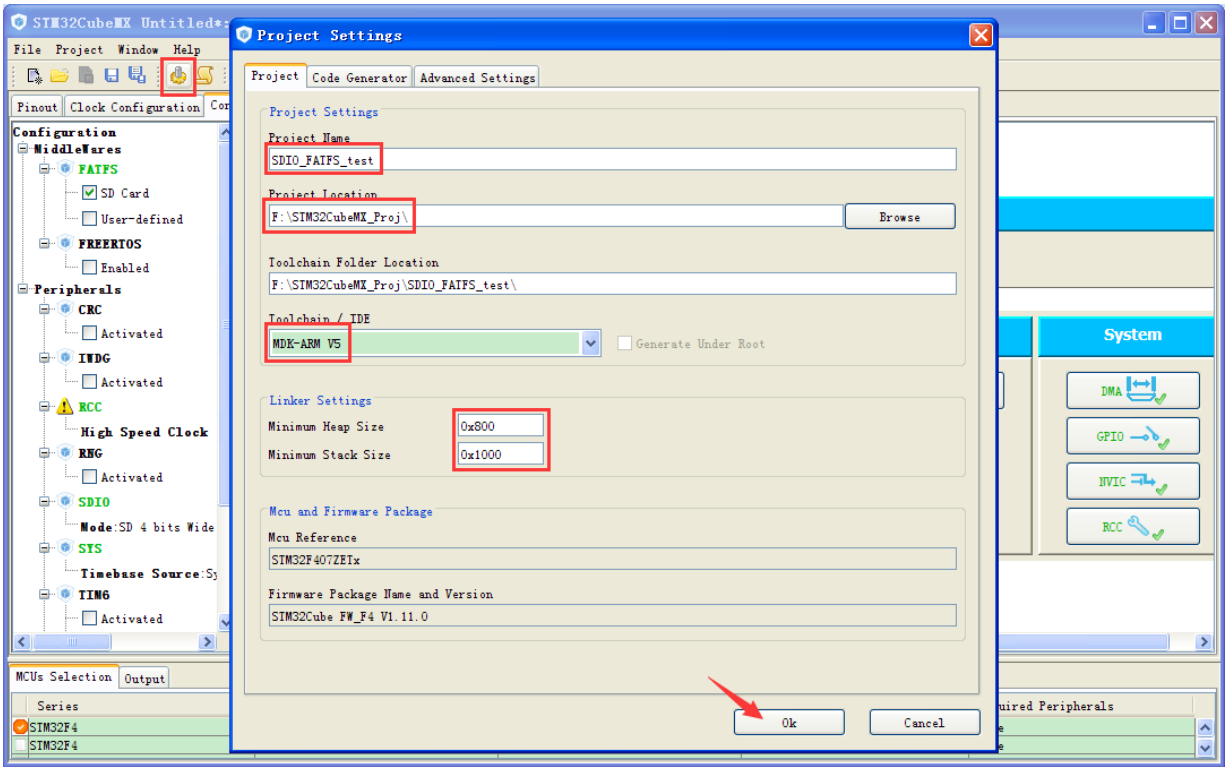
如果要支持中文文件名，则要配置 PAGE_CODE 项为中文。

如果要支持长文件名，要使能 USE_LFN。

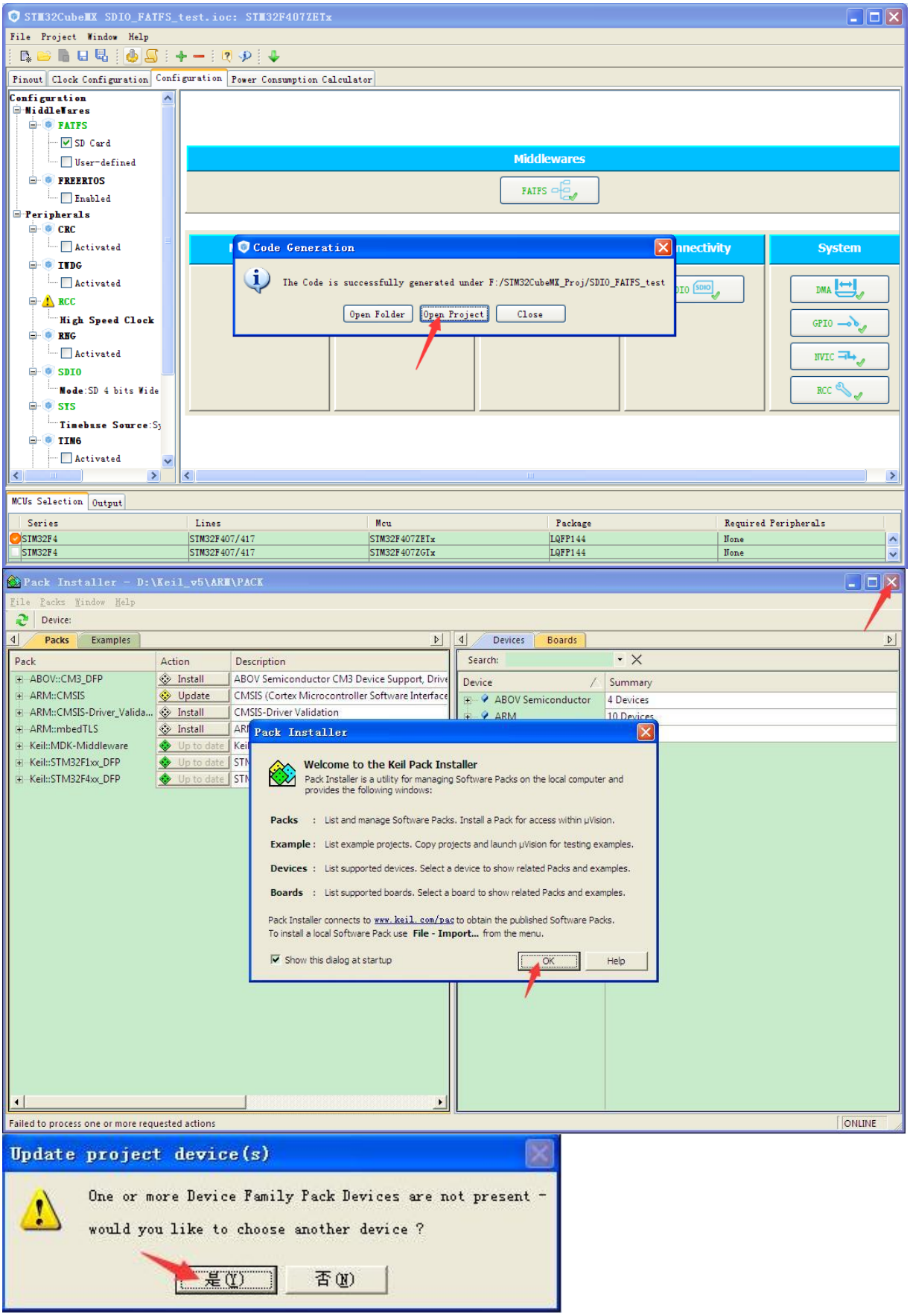
本程序全部使用默认值。



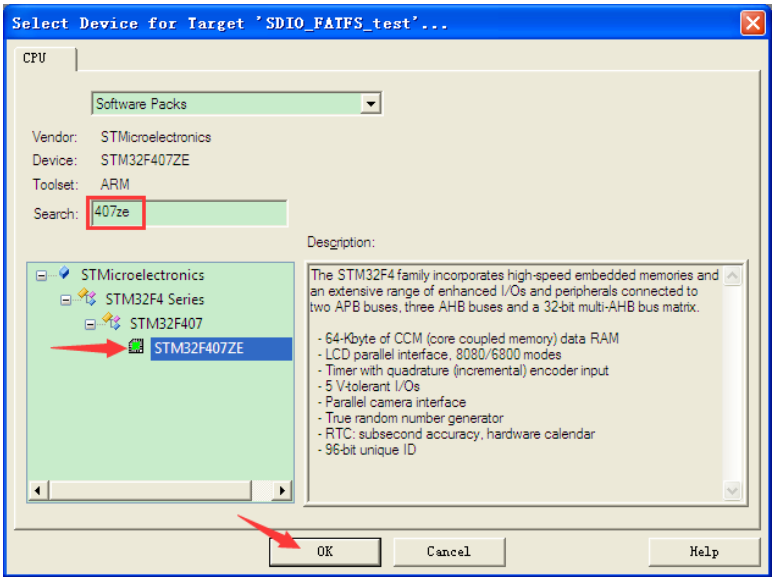
Step11. 生成 MDK 工程及代码。特别注意，一定要加大堆栈大小，默认的堆栈大小不够用。



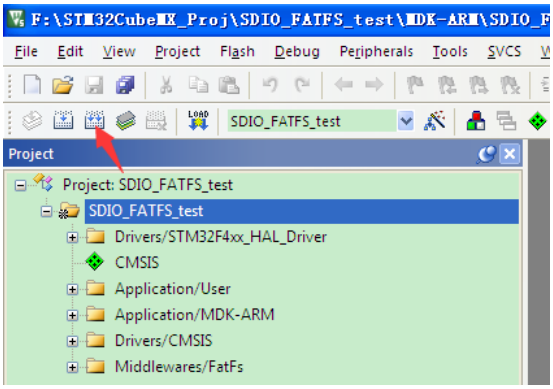
Step12.打开 MDK 工程。



Step13.选择芯片型号



Step14.编译工程。



Step15.添加代码。

这时，就需要参考 STM32CubeF4 的例程了。

解压 stm32cubef4.zip 支持包，可以得到如 STM32Cube_FW_F4_V1.9.0 的文件夹。其中就包含了 STM32CubeF4 的使用例程。

SD+FATFS 的例程在

STM32Cube_FW_F4_V1.9.0\Projects\STM324xG_EVAL\Applications\FatFs\FatFs_uSD 目录下。

打开其中的 MDK-ARM 目录下的工程，参考其中 main.c 的代码。

直接复制变量定义的代码到 main.c 文件中。

```
47  /* USER CODE BEGIN PV */
48  /* Private variables ---------------------------------------------------------*/
49  FATFS SDFatFs;  /* File system object for SD card logical drive */
50  FIL MyFile;     /* File object */
51  char SDPath[4]; /* SD card logical drive path */
52  /* USER CODE END PV */
69  int main(void)
70  {
71
72    /* USER CODE BEGIN 1 */
73    FRESULT res;                                          /* FatFs function common result code */
74    uint32_t byteswritten, bytesread;                     /* File write/read counts */
75    uint8_t wtext[] = "This is STM32 working with FatFs"; /* File write buffer */
76    uint8_t rtext[100];                                   /* File read buffer */
77    /* USER CODE END 1 */
```

复制例程中的 2~10 步骤的代码到 main 函数的/* USER CODE BEGIN 2 */和/* USER CODE END 2 */注释行之间，并修改为如下代码。

```
  MX_DMA_Init();
  MX_SDIO_SD_Init();
  MX_FATFS_Init();

  /* USER CODE BEGIN 2 */
   if(f_mount(&SDFatFs, (TCHAR const*)SDPath, 0) != FR_OK) {                  /*##-2-*/
     while(1);/* FatFs Initialization Error */
   } else {
      if(f_open(&MyFile, "STM32.TXT", FA_CREATE_ALWAYS | FA_WRITE) != FR_OK) {/*##-4-*/
        while(1);/* 'STM32.TXT' file Open for write Error */
      } else {
        res = f_write(&MyFile, wtext, sizeof(wtext), (void *)&byteswritten);  /*##-5-*/
        if((byteswritten == 0) || (res != FR_OK)) {
          while(1);/* 'STM32.TXT' file Write or EOF Error */
        } else {
          f_close(&MyFile);                                                   /*##-6-*/
          if(f_open(&MyFile, "STM32.TXT", FA_READ) != FR_OK) {                /*##-7-*/
            while(1);/* 'STM32.TXT' file Open for read Error */
          } else {
            res = f_read(&MyFile, rtext, sizeof(rtext), (UINT*)&bytesread);   /*##-8-*/
            if((bytesread == 0) || (res != FR_OK)) {
              while(1);/* 'STM32.TXT' file Read or EOF Error */
            } else {
              f_close(&MyFile);                                               /*##-9-*/
              if((bytesread != byteswritten)) {                               /*##-10-*/
                while(1);/* Read data is different from the expected data */
              } else {
                //while(1);/* Success of the demo: no error occurrence */
              }
            }
          }
        }
      }
   }
  /* USER CODE END 2 */
```

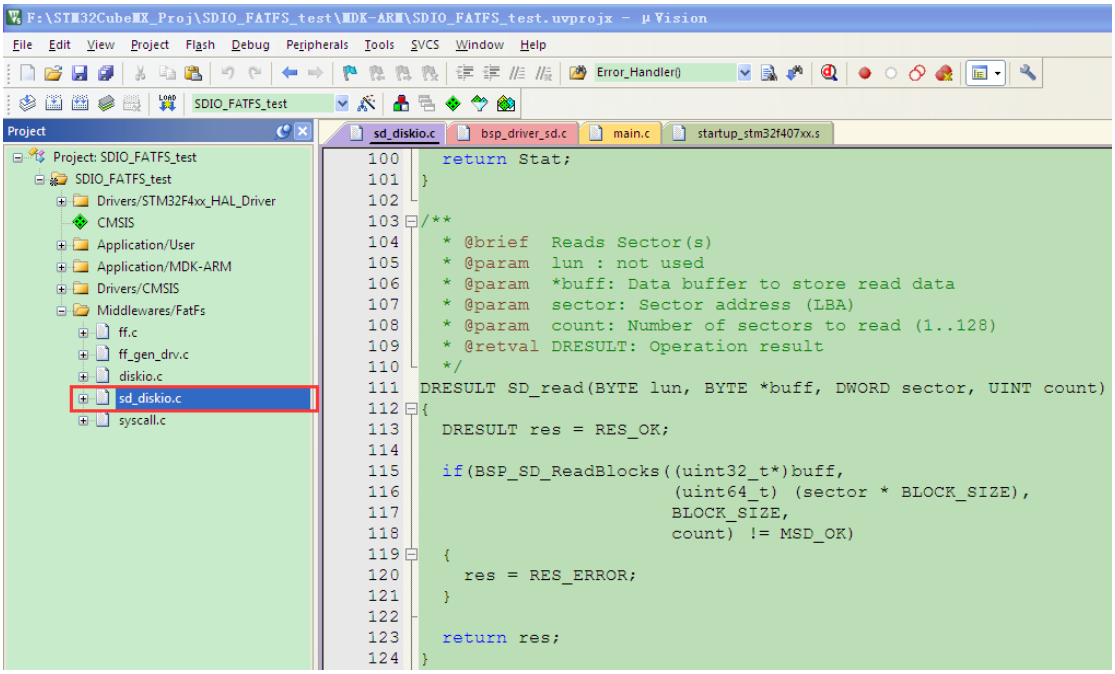
注意：

和例程代码相比，本程序省略了步骤 1。因为步骤 1 的 FATFS_LinkDriver()是链接 SD 底层驱动函数到 FATFS 文件系统，这个在 MX_FATFS_Init()函数中已经完成。

本程序也省略了步骤 3，因为该步骤是对 SD 卡进行格式化！显然是没必要的。

生成的代码中，FATFS 文件系统的 SD 卡的底层读写操作函数并没有使用 DMA。需要手动修改。【这一步非常关键】

打开 sd_diskio.c 文件，找到第 111 行。



读/写原函数：

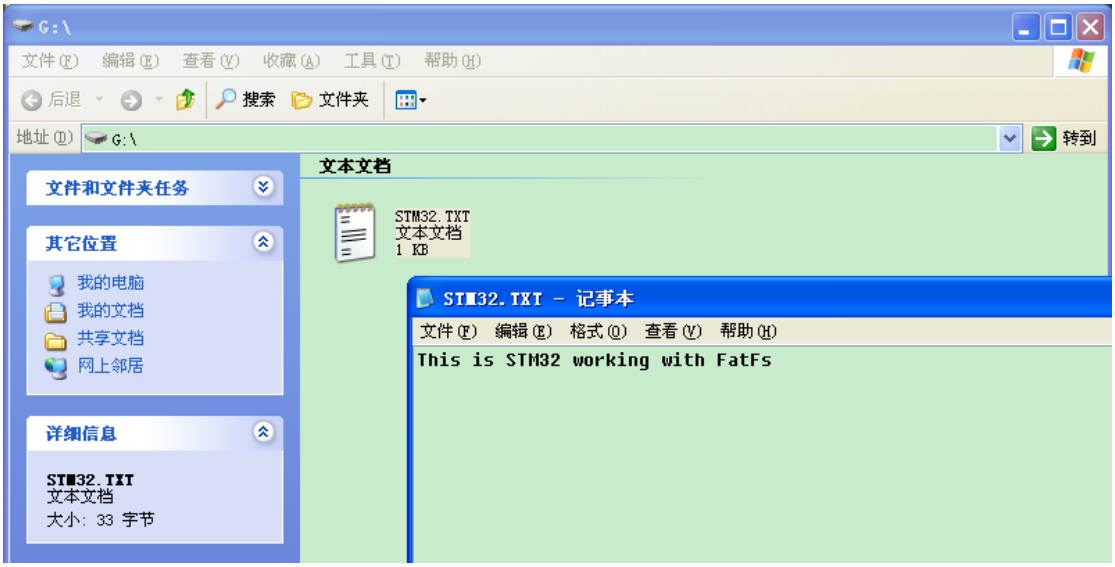
```
111  DRESULT SD_read(BYTE lun, BYTE *buff, DWORD sector, UINT count)
112  {
113    DRESULT res = RES_OK;
114
115    if(BSP_SD_ReadBlocks((uint32_t*)buff,
116                         (uint64_t) (sector * BLOCK_SIZE),
117                         BLOCK_SIZE,
118                         count) != MSD_OK)
119    {
120      res = RES_ERROR;
121    }
122
123    return res;
124  }
135  DRESULT SD_write(BYTE lun, const BYTE *buff, DWORD sector, UINT count)
136  {
137    DRESULT res = RES_OK;
138
139    if(BSP_SD_WriteBlocks((uint32_t*)buff,
140                          (uint64_t)(sector * BLOCK_SIZE),
141                          BLOCK_SIZE, count) != MSD_OK)
142    {
143      res = RES_ERROR;
144    }
145
146    return res;
147  }
```

改为：

```
111  DRESULT SD_read(BYTE lun, BYTE *buff, DWORD sector, UINT count)
112  {
113    DRESULT res = RES_OK;
114
115    if(BSP_SD_ReadBlocks_DMA((uint32_t*)buff,
116                          (uint64_t) (sector * BLOCK_SIZE),
117                          BLOCK_SIZE,
118                          count) != MSD_OK)
119    {
120      res = RES_ERROR;
121    }
122
123    return res;
124  }
125
135  DRESULT SD_write(BYTE lun, const BYTE *buff, DWORD sector, UINT count)
136  {
137    DRESULT res = RES_OK;
138
139    if(BSP_SD_WriteBlocks_DMA((uint32_t*)buff,
140                          (uint64_t)(sector * BLOCK_SIZE),
141                          BLOCK_SIZE, count) != MSD_OK)
142    {
143      res = RES_ERROR;
144    }
145
146    return res;
147  }
```

Step16.编译下载运行。

使用 J-Link 进行在线仿真调试，运行后，看看程序停止在哪一个 while(1)。如果是停止在主函数最后一个 while(1)，则成功，取出 SD 卡，用电脑查看其内容，会发现多了一个 STM32.TXT 文件，内容如下图。如果程序运行后，停止在其他的 while(1)，请根据错误说明找原因。



本例程序构建了 FATFS 管理 SD 卡文件的基本环境，包含了文件系统的读写基本操作。同样支持如文件的创建、删除以及 SD 卡格式化等操作，调用相关函数即可实现。

另外，本例中的结构体变量 SDFatFs 在调用 f_mout()函数之后，就获得了当前 SD 卡总容量、剩余容量等的相关信息。

S.D.Lu 于 深圳
2016 年 10 月