STM32L462 SDMMC DMA 多次循环读写

关键字：STM32L462，SDMMC，DMA

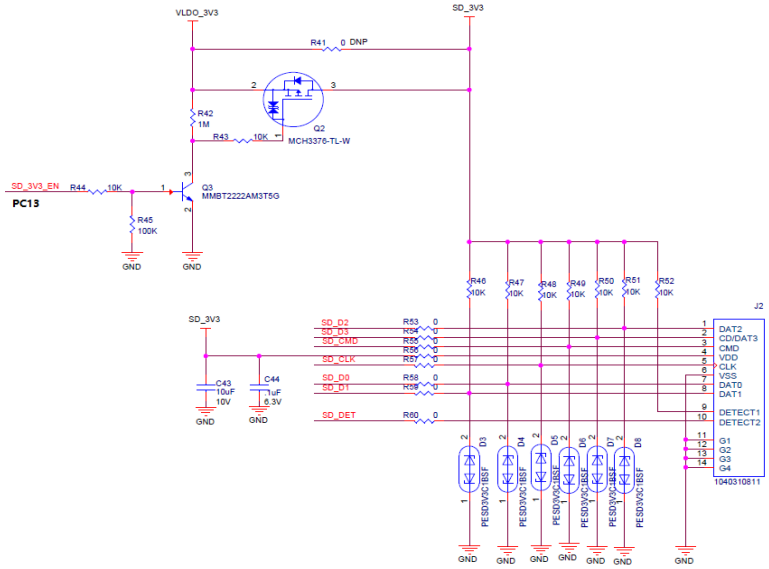
1. 引言

某客户在项目开发中用到 STM32L462 的 SDMMC 接口去进行 SD 卡的读写，发现用 SDMMC 中断、 Polling 查询的方式都可以进行连续的读写交替循环操作，但是用 DMA 的方式进行该操作时，发现不能进行多次的读写。

2. 原因分析

客户提供了它的硬件电路板和部分原理图：

图1. 客户 SDMMC 接口电路图



2.1 客户现象复现：

2.1.1 CubeMX 的配置

CubeMX V6.3.0 中关于 SDMMC 外设的配置如下：

图2. SDMMC 在 CubeMX 中的配置

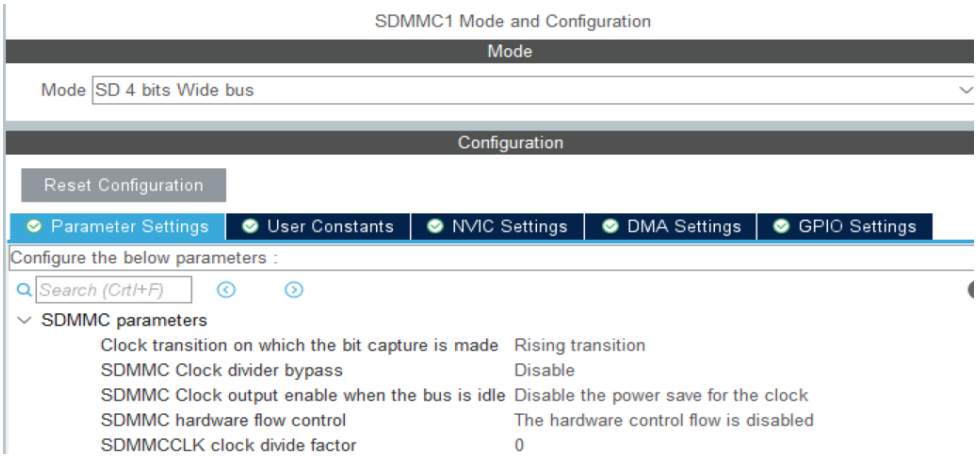
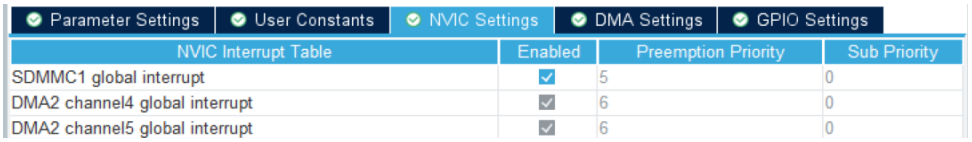


图3. SDMMC 中断的和优先级别的配置



SDMMC DMA 的配置如下：DMA Request Settings 发送和接收都一样，Normal 模式。

图4. SDMMC DMA 配置

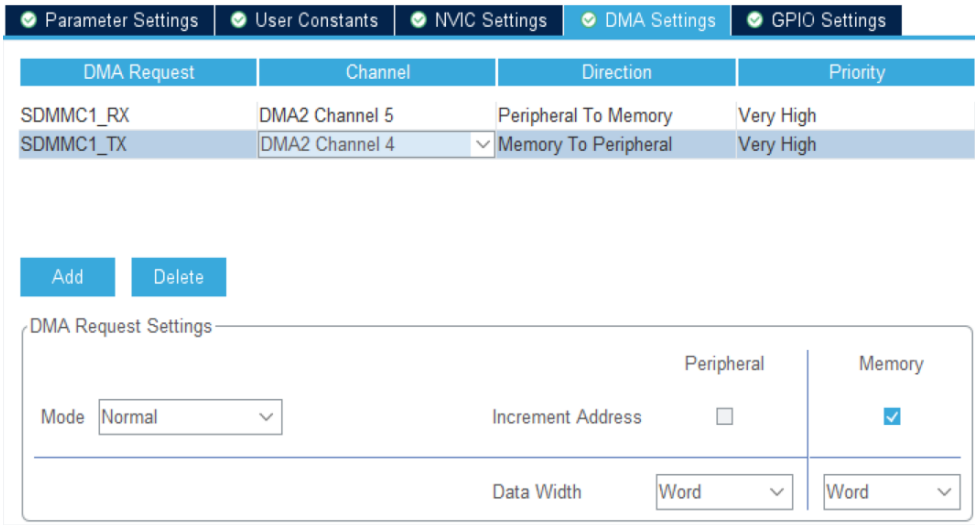
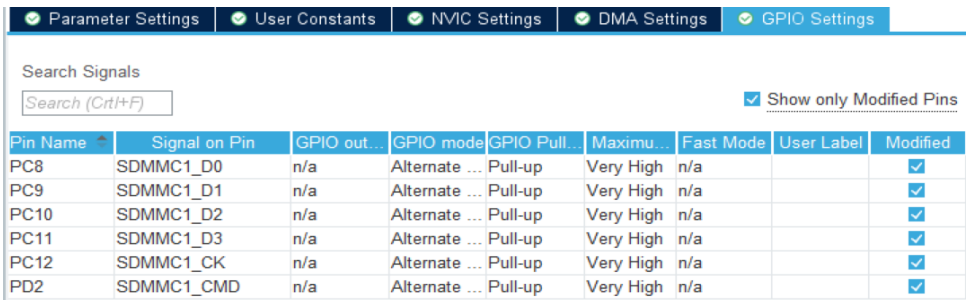


图5. SDMMC 的引脚的配置



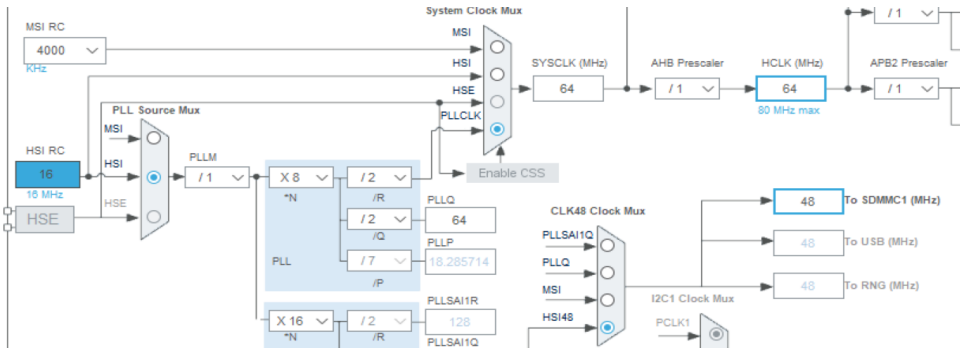
这里结合客户的原理图，还要配置一下 PC13 即给 SD 卡供电的引脚，要不然，后面调试 SD 卡时，MX_SDMMC1_SD_Init();初始化过不了。

图6.　PC13 引脚的配置

Pin Name	Signal on Pin	GPIO outp...	GPIO mode	GPIO Pull-...	Maximum ...	Fast Mode	User Label	Modified
PC13	n/a	High	Output Pu...	No pull-up ...	Low	n/a	SD_3V3_E...	☑

其他的配置保持默认，系统时钟源选择 HSI 经过 PLL 锁相环，SYSCLK=64MHz，SDMMC1 的时钟为 48MHz，分频系数 SDMMCCLK clock divide factor 为 0，但是参考手册公式，SDMMC_CK = SDMMCCLK / [CLKDIV + 2]。所以 SDMMC_CK 频率这里是 24MHz。

图7.　时钟树配置



CubeMX 配置好后，直接生成代码。

2.1.2 添加测试代码

代码生成后，添加如下的测试代码及相关调用函数：

图8.　相关的代码

main.c

```
65
66  /* Private user code ---------------------------------------------------------*/
67  /* USER CODE BEGIN 0 */
68  #define BLOCK_SIZE            512          // SD卡块大小
69  #define NUMBER_OF_BLOCKS      1           //每条记录的BLOCK数量
70  #define WRITE_READ_ADDRESS    0x00000000   //测试读写地址
71
72  #define   SD_TRANSFER_OK                ((uint8_t)0x00)
73  #define   SD_TRANSFER_BUSY              ((uint8_t)0x01)
74  #define   SD_TRANSFER_ERROR             ((uint8_t)0x02)
75
76  __attribute__ ((aligned (4))) uint8_t Buffer_Block_WR[BLOCK_SIZE*NUMBER_OF_BLOCKS]; // 写数据缓存
77  __attribute__ ((aligned (4))) uint8_t Buffer_Block_RD[BLOCK_SIZE*NUMBER_OF_BLOCKS]; // 读数据缓存
78
79  uint32_t Temp_Offset = 0;
80
81  HAL_StatusTypeDef  sdSta;
82
83  __IO uint32_t writestatus, readstatus = 0;
84
85  volatile uint32_t main_count = 0;
86
87  void Fill_Buffer(uint8_t *pBuffer, uint32_t BufferLength, uint32_t Offset);
88  uint8_t BSP_SD_ReadBlocks_DMA(uint32_t *pData, uint32_t ReadAddr, uint32_t NumOfBlocks);
89  uint8_t BSP_SD_WriteBlocks_DMA(uint32_t *pData, uint32_t WriteAddr, uint32_t NumOfBlocks);
90  uint8_t BSP_SD_GetCardState(void);
91  HAL_StatusTypeDef SD_DMAConfigRx(SD_HandleTypeDef *hsd);
92  HAL_StatusTypeDef SD_DMAConfigTx(SD_HandleTypeDef *hsd);
93
94  /* USER CODE END 0 */
```

main.c

MX_GPIO_Init()

```
294
295  /* USER CODE BEGIN 4 */
296  //填充缓冲区数组
297  void Fill_Buffer(uint8_t *pBuffer, uint32_t BufferLength, uint32_t Offset)
298  {
299    uint32_t index = 0;
300    /* 填充数据 */
301    for (index = 0; index < BufferLength; index++ )
302    {
303      pBuffer[index] = index + Offset;
304    }
305  }
306  /* USER CODE END 4 */
```

main.c

```
309  /**
310    * @brief  Gets the current SD card data status.
311    * @param  None
312    * @retval Data transfer state.
313    */
314  uint8_t BSP_SD_GetCardState(void)
315  {
316    HAL_SD_CardStateTypedef card_state;
317    card_state = HAL_SD_GetCardState(&hsd1);
318
319    if (card_state == HAL_SD_CARD_TRANSFER)
320    {
321      return (SD_TRANSFER_OK);
322    }
323    else if ((card_state == HAL_SD_CARD_SENDING) ||
324             (card_state == HAL_SD_CARD_RECEIVING) ||
325             (card_state == HAL_SD_CARD_PROGRAMMING))
326    {
327      return (SD_TRANSFER_BUSY);
328    }
329    else
330    {
331      return (SD_TRANSFER_ERROR);
332    }
333  }
334
335  /**
336    * @brief Tx Transfer completed callback
337    * @param hsd: SD handle
338    * @retval None
339    */
340  void HAL_SD_TxCpltCallback(SD_HandleTypeDef *hsd)
341  {
342    writestatus = 1;
343  }
344
345  /**
346    * @brief Rx Transfer completed callback
347    * @param hsd: SD handle
348    * @retval None
349    */
350  void HAL_SD_RxCpltCallback(SD_HandleTypeDef *hsd)
351  {
352     readstatus = 1;
353  }
```

另外在中断函数的里面，可以添加如下代码，加一个判断。

图9.　中断函数添加的代码

main.c stm32l4xx_it.c

```
221  void DMA2_Channel4_IRQHandler(void)
222  {
223    /* USER CODE BEGIN DMA2_Channel4_IRQn 0 */
224
225    /* USER CODE END DMA2_Channel4_IRQn 0 */
226    if((hsdl.Context == (SD_CONTEXT_DMA | SD_CONTEXT_WRITE_SINGLE_BLOCK)) ||
227            (hsdl.Context == (SD_CONTEXT_DMA | SD_CONTEXT_WRITE_MULTIPLE_BLOCK)))
228    {
229        HAL_DMA_IRQHandler(&hdma_sdmmc1_tx);
230    }
231    /* USER CODE BEGIN DMA2_Channel4_IRQn 1 */
232
233    /* USER CODE END DMA2_Channel4_IRQn 1 */
234  }
235
236  /**
237    * @brief This function handles DMA2 channel5 global interrupt.
238    */
239  void DMA2_Channel5_IRQHandler(void)
240  {
241    /* USER CODE BEGIN DMA2_Channel5_IRQn 0 */
242
243    /* USER CODE END DMA2_Channel5_IRQn 0 */
244    if((hsdl.Context == (SD_CONTEXT_DMA | SD_CONTEXT_READ_SINGLE_BLOCK)) ||
245       (hsdl.Context == (SD_CONTEXT_DMA | SD_CONTEXT_READ_MULTIPLE_BLOCK)))
246    {
247       HAL_DMA_IRQHandler(&hdma_sdmmc1_rx);
248    }
249    /* USER CODE BEGIN DMA2_Channel5_IRQn 1 */
250
251    /* USER CODE END DMA2_Channel5_IRQn 1 */
252  }
```

图10. 主函数中测试代码

main.c

main()

```
125
126   while (1)
127   {
128     Fill_Buffer(Buffer_Block_WR,BLOCK_SIZE*NUMBER_OF_BLOCKS, Temp_Offset++);
129
130     if(HAL_SD_Erase(&hsd1,0x00000000,WRITE_READ_ADDRESS+1000) != HAL_OK){
131       while(1);
132     }
133     /* Wait until SD card is ready to use for new operation */
134     while (BSP_SD_GetCardState() != SD_TRANSFER_OK){}  //delay for erase finished
135
136     // if(SD_DMAConfigTx(&hsd1) == HAL_OK)  //comment here for repeat customer issue
137     {
138       if(HAL_SD_WriteBlocks_DMA(&hsd1,Buffer_Block_WR,WRITE_READ_ADDRESS, 1) != HAL_OK){
139         while(1);
140       }
141     }
142     /* Wait for Tx Transfer completion */
143     while(writestatus == 0){}
144     writestatus = 0;
145
146     /* Wait until SD card is ready to use for new operation */
147     while (BSP_SD_GetCardState() != SD_TRANSFER_OK){}
148
149     // if(SD_DMAConfigRx(&hsd1) == HAL_OK)   //comment here for repeat customer issue
150     {
151       if(HAL_SD_ReadBlocks_DMA(&hsd1,Buffer_Block_RD, WRITE_READ_ADDRESS, 1) != HAL_OK){
152         while(1);
153       }
154     }
155     while (readstatus == 0){}
156     readstatus = 0;
157
158     /* Wait until SD card is ready to use for new operation */
159     while (BSP_SD_GetCardState() != SD_TRANSFER_OK){}
160
161     HAL_Delay(2000);
162     main_count++;  //test for main loop
163
164   }
```

Live Watch

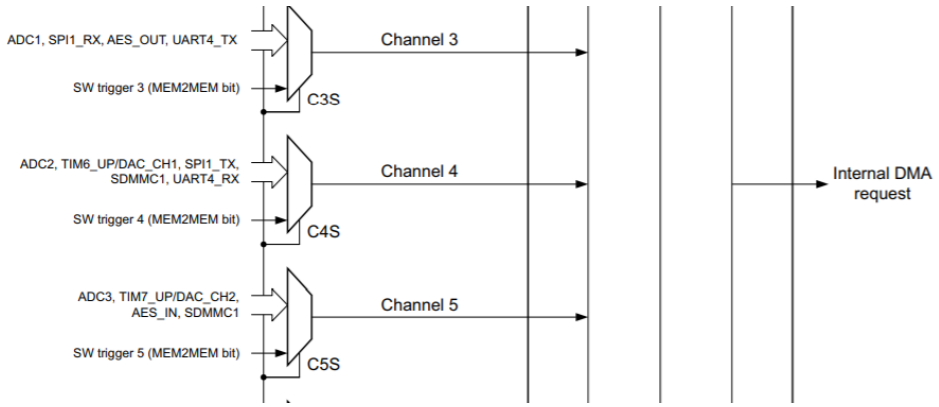
Expression	Value
main_count	0
Buffer_Block_RD	<array>""
[0]	'\0' (0x00)
[1]	'\0' (0x00)
[2]	'\0' (0x00)
[3]	'\0' (0x00)
[4]	'\0' (0x00)
[5]	'\0' (0x00)
[6]	'\0' (0x00)
[7]	'\0' (0x00)
[8]	'\0' (0x00)
[9]	'\0' (0x00)
[10]	'\0' (0x00)
[11]	'\0' (0x00)
[12]	'\0' (0x00)
[13]	'\0' (0x00)
[14]	'\0' (0x00)
[15]	'\0' (0x00)
[16]	'\0' (0x00)
[17]	'\0' (0x00)
[18]	'\0' (0x00)
[19]	'\0' (0x00)
[20]	'\0' (0x00)
[21]	'\0' (0x00)
[22]	'\0' (0x00)
[23]	'\0' (0x00)
[24]	'\0' (0x00)
[25]	'\0' (0x00)

添加好编译测试，会发现代码卡在如上图的 while 循环等待中，main_count=0，也就是一次的读写操作都还没有完成。即复现了客户的现象。但是连续多次的单独写可以顺利执行，连续多次的单独读可以顺利执行(如果不行的话，请注意 SDMMC DMA 中断函数里面的断言是不是判断反了，打断点看看稍微调试即可)。

2.1.3 问题原因分析

通过查阅参考手册中关于 DMA 章节内容，Figure 26. DMA2 request mapping， 发现 SDMMC 的 DMA 中断并不像 UART4_TX 和 UART4_RX 请求一样，分别各对应一条 DMA 通道，SDMMC 只有 Channe 4 或 Channel 5 可用且不能同时使用。

图11. DMA 配置参考



2.1.4 问题解决

基于上述的分析后，我们的 SDMMC 的读或写都使用同一个通道，或者两个通道不同时使用，实施代码如下即可。

图12. 实时代码

main.c

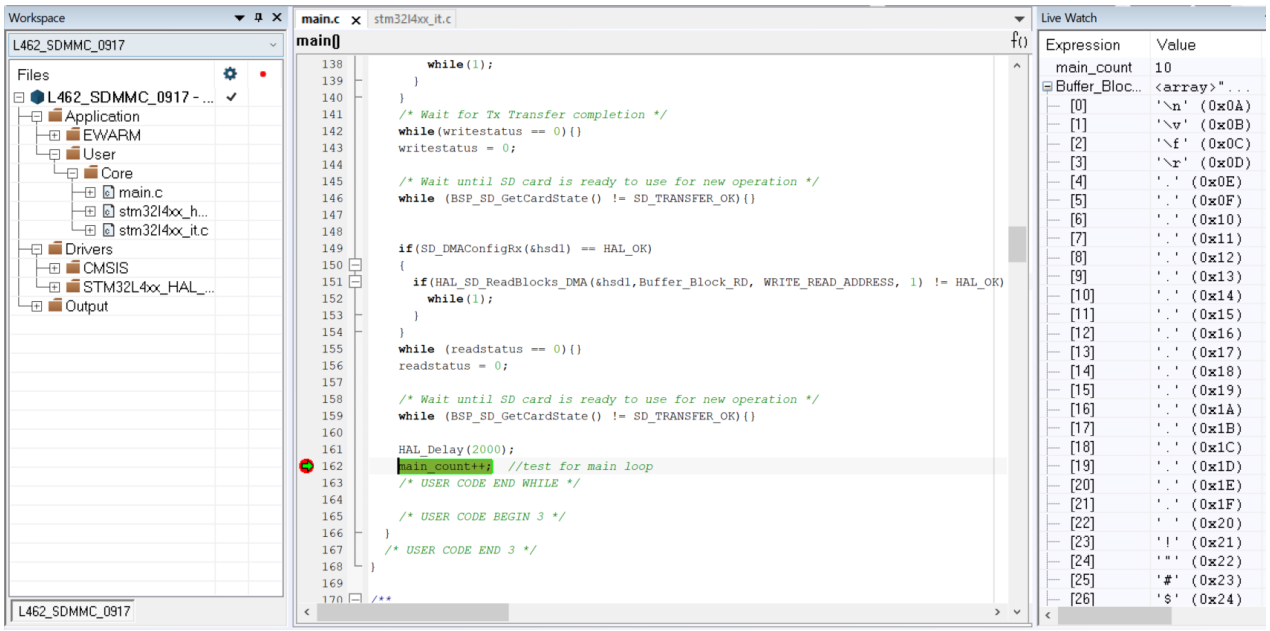
SD_DMAConfigRx(SD_HandleTypeDef *)

```
363  HAL_StatusTypeDef SD_DMAConfigRx(SD_HandleTypeDef *hsd)
364  {
365    HAL_StatusTypeDef status = HAL_ERROR;
366    /* Configure DMA Rx parameters */
367    hdma_sdmmc1_rx.Init.Request             = DMA_REQUEST_7;
368    hdma_sdmmc1_rx.Init.Direction           = DMA_PERIPH_TO_MEMORY;
369    hdma_sdmmc1_rx.Init.PeriphInc           = DMA_PINC_DISABLE;
370    hdma_sdmmc1_rx.Init.MemInc              = DMA_MINC_ENABLE;
371    hdma_sdmmc1_rx.Init.PeriphDataAlignment = DMA_PDATAALIGN_WORD;
372    hdma_sdmmc1_rx.Init.MemDataAlignment    = DMA_MDATAALIGN_WORD;
373    hdma_sdmmc1_rx.Init.Priority            = DMA_PRIORITY_VERY_HIGH;
374
375    hdma_sdmmc1_rx.Instance = DMA2_Channel5;
376
377    /* Associate the DMA handle */
378    __HAL_LINKDMA(hsd, hdmarx, hdma_sdmmc1_rx);
379
380    /* Stop any ongoing transfer and reset the state*/
381    HAL_DMA_Abort(&hdma_sdmmc1_rx);
382
383    /* Deinitialize the Channel for new transfer */
384    HAL_DMA_DeInit(&hdma_sdmmc1_rx);
385
386      /* Stop any ongoing transfer and reset the state*/
387    HAL_DMA_Abort(&hdma_sdmmc1_tx);  //add here to deint
388    HAL_DMA_DeInit(&hdma_sdmmc1_tx); //add here to deint
389
390    /* Configure the DMA Channel */
391    status = HAL_DMA_Init(&hdma_sdmmc1_rx);
392
393    /* NVIC configuration for DMA transfer complete interrupt */
394    HAL_NVIC_SetPriority(DMA2_Channel5_IRQn, 6, 0);
395    HAL_NVIC_EnableIRQ(DMA2_Channel5_IRQn);
396
397    return (status);
398  }
```

main.c

```
405  HAL_StatusTypeDef SD_DMAConfigTx(SD_HandleTypeDef *hsd)
406  {
407    HAL_StatusTypeDef status;
408    /* Configure DMA Tx parameters */
409    hdma_sdmmc1_tx.Init.Request             = DMA_REQUEST_7;
410    hdma_sdmmc1_tx.Init.Direction           = DMA_MEMORY_TO_PERIPH;
411    hdma_sdmmc1_tx.Init.PeriphInc           = DMA_PINC_DISABLE;
412    hdma_sdmmc1_tx.Init.MemInc              = DMA_MINC_ENABLE;
413    hdma_sdmmc1_tx.Init.PeriphDataAlignment = DMA_PDATAALIGN_WORD;
414    hdma_sdmmc1_tx.Init.MemDataAlignment    = DMA_MDATAALIGN_WORD;
415    hdma_sdmmc1_tx.Init.Priority            = DMA_PRIORITY_VERY_HIGH;
416
417    hdma_sdmmc1_tx.Instance = DMA2_Channel4;
418
419    /* Associate the DMA handle */
420    __HAL_LINKDMA(hsd, hdmatx, hdma_sdmmc1_tx);
421
422    /* Stop any ongoing transfer and reset the state*/
423    HAL_DMA_Abort(&hdma_sdmmc1_tx);
424
425    /* Deinitialize the Channel for new transfer */
426    HAL_DMA_DeInit(&hdma_sdmmc1_tx);  //add here to deint
427    HAL_DMA_Abort(&hdma_sdmmc1_rx);   //add here to deint
428
429    /* Deinitialize the Channel for new transfer */
430    HAL_DMA_DeInit(&hdma_sdmmc1_rx);
431
432
433    /* Configure the DMA Channel */
434    status = HAL_DMA_Init(&hdma_sdmmc1_tx);
435
436    /* NVIC configuration for DMA transfer complete interrupt */
437    HAL_NVIC_SetPriority(DMA2_Channel4_IRQn, 6, 0);
438    HAL_NVIC_EnableIRQ(DMA2_Channel4_IRQn);
439
440    return (status);
441  }
```

图13. 运行结果



成功！另外，感兴趣的同学，也可以参考如下路径的例程。

……\STM32Cube_FW_L4_V1.17.0\Projects\32L496GDISCOVERY\Applications\FatFs\FatFs_uSD_DMA_RTOS\

版本历史

日期	版本	变更
2021 年 12 月 14 日	1.0	首版发布