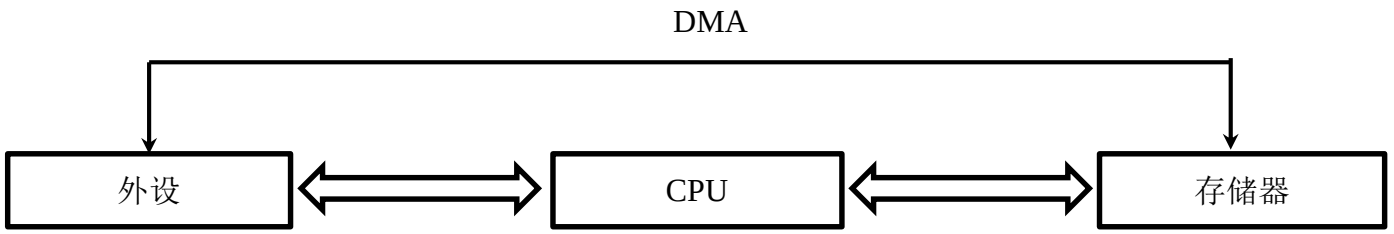
第8章 STM32直接存储器存取
DMA (Direct Memory Access)

退出

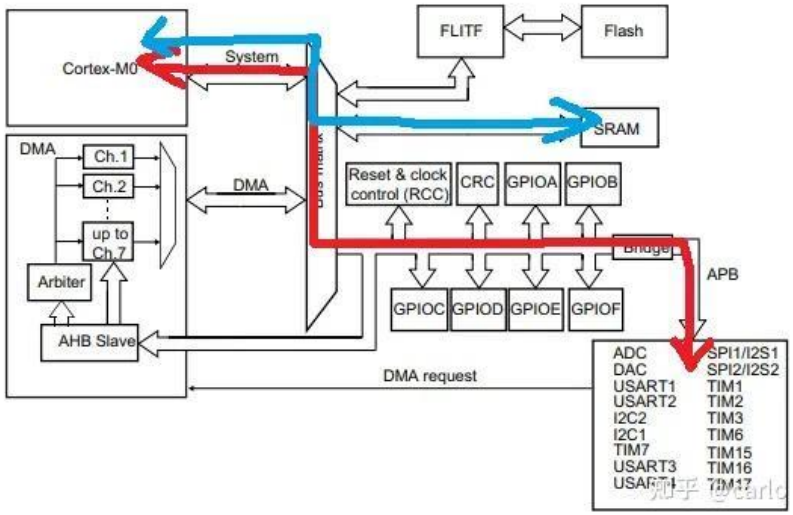
8.1 DMA简介

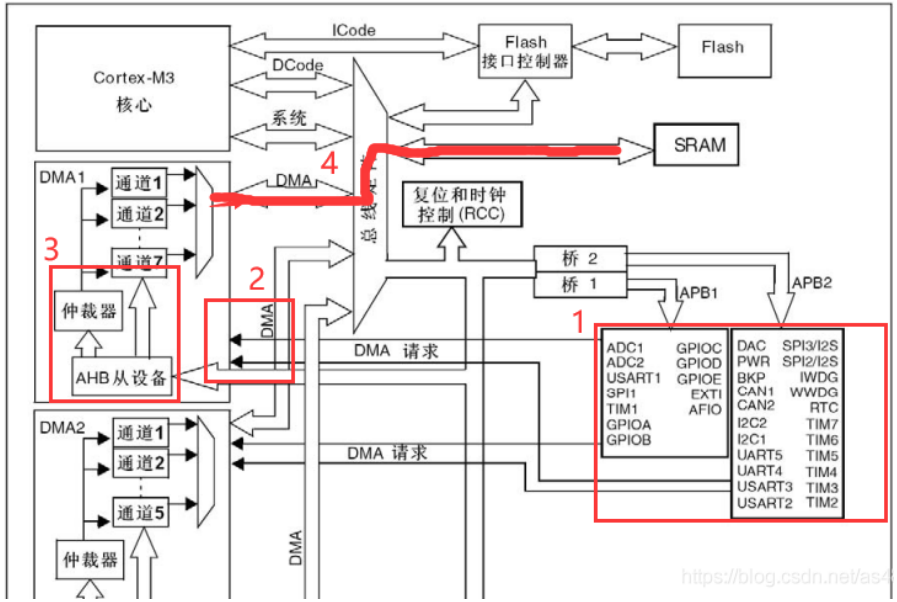
DMA定义



DMA (Direct Memory Access，直接存储器访问)是一种完全由硬件执行批量数据交换的工作方式，由DMA控制器而不是CPU控制数据传输过程。

DMA传输过程





无 **DMA**：

- 内核通过DCode经过总线矩阵协调，从AHB的获取外设ADC采集的数据；
- 然后内核再通过DCode经过总线矩阵协调把数据存放到内存SRAM中。

DMA：

- CPU对DMA控制器初始化；
- 外设对DMA控制器发出请求；
- DMA控制器收到请求，触发DMA工作；
- DMA控制器从AHB外设获取ADC采集的数据，存储到DMA通道中
- DMA控制器的DMA总线与总线矩阵协调，使用AHB把外设ADC采集的数据经由DMA通道存放到SRAM中。

DMA的优点

CPU利用率高。DMA控制器的控制下完成数据传输，不需要CPU干预。

数据传输效率高。如果使用CPU实现数据传输，通常先将数据从源地址存储到某个中间变量，再将数据从中间变量转送到目标地址。

软件实现便捷。存储器地址修改、传送数据的计数等由硬件直接实现。

8.2 DMA的结构

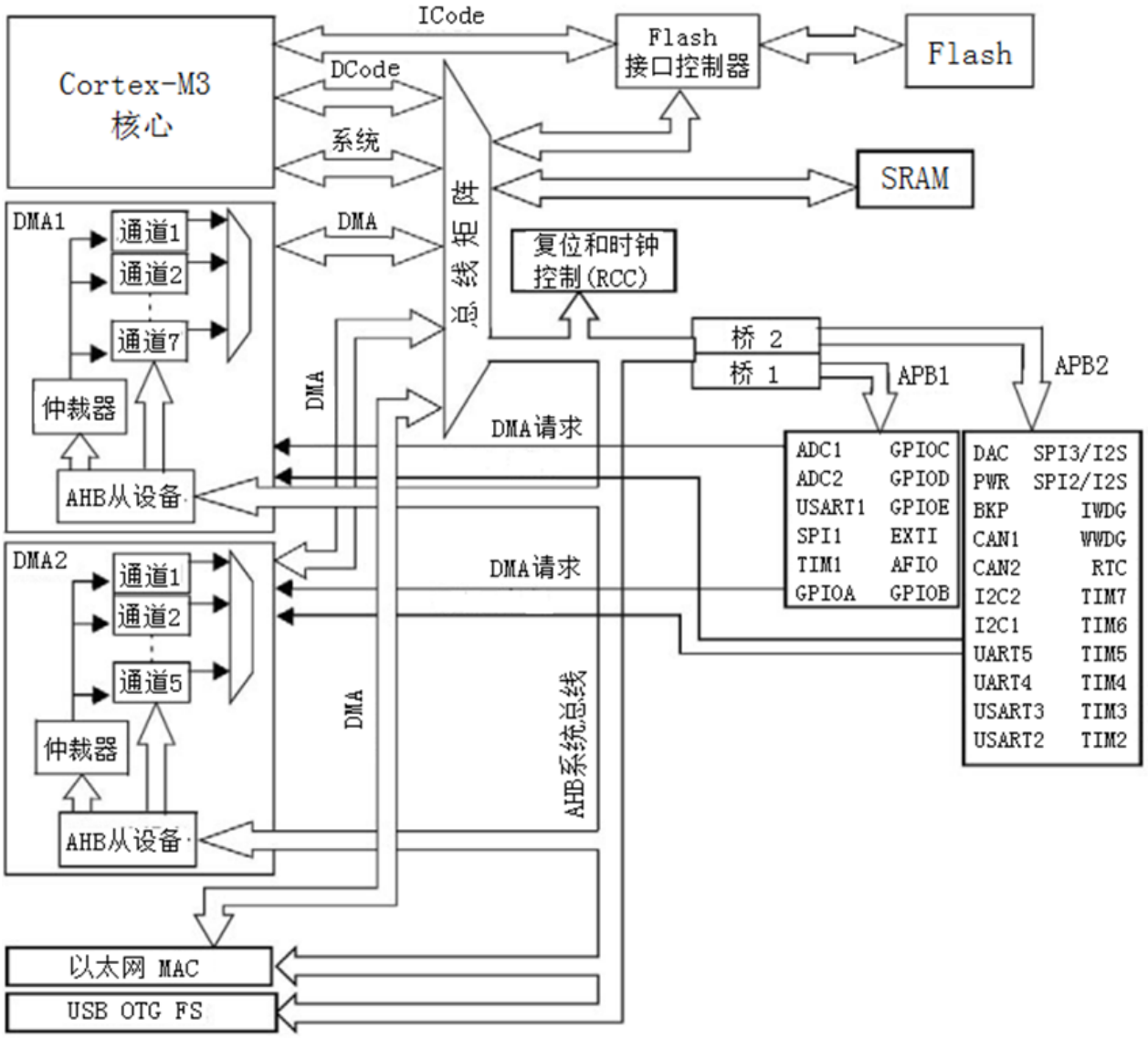
STM32F103的DMA主要特性

- 12个独立的可配置的通道(请求)： **DMA1有7个通道， DMA2有5个通道**。
- 每个通道都直接连接专用的硬件DMA请求，每个通道都同样支持软件触发。这些功能通过软件来配置。
- 在同一个DMA模块上，多个请求间的优先权可以通过软件编程设置（**共有四级：很高、高、中等和低**）。
- 独立数据源和目标数据区的传输宽度（**字节、半字、全字**），模拟打包和拆包的过程。源和目标地址必须按数据传输宽度对齐。

STM32F103的DMA主要特性

- 支持循环的缓冲器管理。
- 每个通道都有3个事件标志（**DMA半传输、 DMA传输完成和DMA传输出错**），这3个事件标志逻辑或成为一个单独的中断请求。
- 支持**存储器和存储器、外设和存储器、存储器和外设**之间的传输。
- 闪存、 SRAM、外设的SRAM、 APB1、 APB2和AHB外设均可作为访问的源和目标。
- 可编程的数据传输数目：**最大为65536**。

STM32F103的DMA内部结构



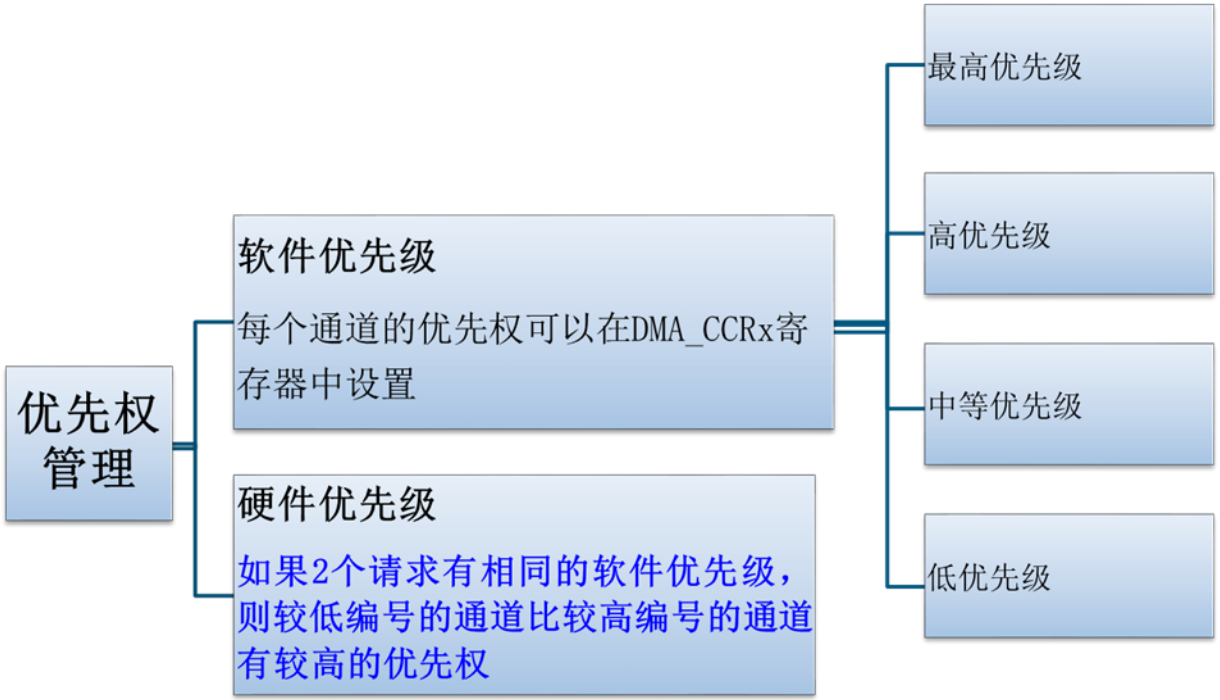
8.3 DMA工作过程

（1）DMA基本工作流程

- DMA请求：CPU对DMA控制器初始化，并向外设发出操作命令，外设提出DMA请求。
- DMA响应：DMA控制器对DMA请求判别优先级及屏蔽，向总线裁决逻辑提出总线请求；CPU释放总线控制权。
- DMA传输：DMA控制器获得总线控制权后，CPU即刻挂起或只执行内部操作；由DMA控制器输出读写命令，直接控制批量数据的DMA传输。
- DMA结束：当完成规定的批量数据传送后，DMA控制器即释放总线控制权，并向外设发出结束信号。

（2）仲裁器

仲裁器根据通道请求的优先级来启动外设/存储器的访问。仲裁器管理DMA通道请求分为两个阶段。**第一阶段属于软件阶段**，可以在DMA_CCRx寄存器中设置，**有4个等级：非常高、高、中和低四个优先级。第二阶段属于硬件阶段**，如果两个或以上的DMA通道请求优先级相同，则它们**优先级取决于通道编号，编号越低，优先级越高。**比如通道0高于通道1，在大容量产品中DMA1控制器拥有高于DMA2控制器的优先级。



（3）DMA通道

- 每个通道都可以在有固定地址的外设寄存器和存储器地址之间执行DMA传输。
- DMA传输的数据量是可编程的，**最大达到65536**。
- 包含要传输的数据项数量的寄存器，在每次传输后递减。

（4）指针自增

- 通过程序可以设置下一次传输数据的地址。
- **外设和存储器指针在每次传输后可以完成自动增量**。
- 当设置为增量模式时，下一个要传输的地址将是前一个地址加上增量值，**增量值取决于所选的数据宽度为1、2或4**。

（5）中断

- 每个DMA通道都可以在DMA传输过半、传输完成和传输错误时产生中断。
- 当传输一半的数据后，半传输标志HTIFx@DMA_ISR被置1，当设置了允许半传输中断位HTIE@DMA_CCRx时，将产生一个中断请求。
- 在数据传输结束后，传输完成标志TCIFx@DMA_ISR被置1，当设置了允许传输完成中断位TCIE@DMA_CCRx时，将产生一个中断请求。

中断事件	事件标志位	使能控制位
传输过半	HTIF	HTIE
传输完成	TCIF	TCIE
传输错误	TEIF	TEIE

（6）循环模式

- 循环模式用于处理循环缓冲区和连续的数据传输（如ADC的扫描模式）。
- 如果通道配置为非循环模式，传输结束后，即传输数据量变为0，则不再进行DMA操作。
- 当启动了循环模式，数据传输的数目变为0时，将会自动地被恢复成配置通道时设置的初值，DMA操作将会继续进行。

（7）存储器到存储器模式

- **DMA通道的操作可以在没有外设请求的情况下进行，这种操作就是存储器到存储器模式。**
- 当设置了DMA_CCRx寄存器中的MEM2MEM位之后，在软件设置了DMA_CCRx寄存器中的EN位启动DMA通道时，DMA传输将马上开始。
- 当DMA_CNDTRx寄存器变为0时，DMA传输结束。
- **存储器到存储器模式不能与循环模式同时使用。**

（8） 错误管理

- 读写一个保留的地址区域，将会产生DMA传输错误。
- 当在DMA读写操作时发生DMA传输错误时，硬件会自动地清除发生错误的通道所对应的通道配置寄存器(DMA_CCRx)的EN位，该通道操作被停止。
- 此时，在DMA_IFR寄存器中对应该通道的传输错误中断标志位(TEIF)将被置位，如果在DMA_CCRx寄存器中设置了传输错误中断允许位，则将产生中断。

（9）DMA请求映射

DMA1

- STM32F103的DMA1有7个触发通道
- 可以分别从外设产生的7个访问请求

  TIMx [x=1、 2、 3、 4]

  ADC1

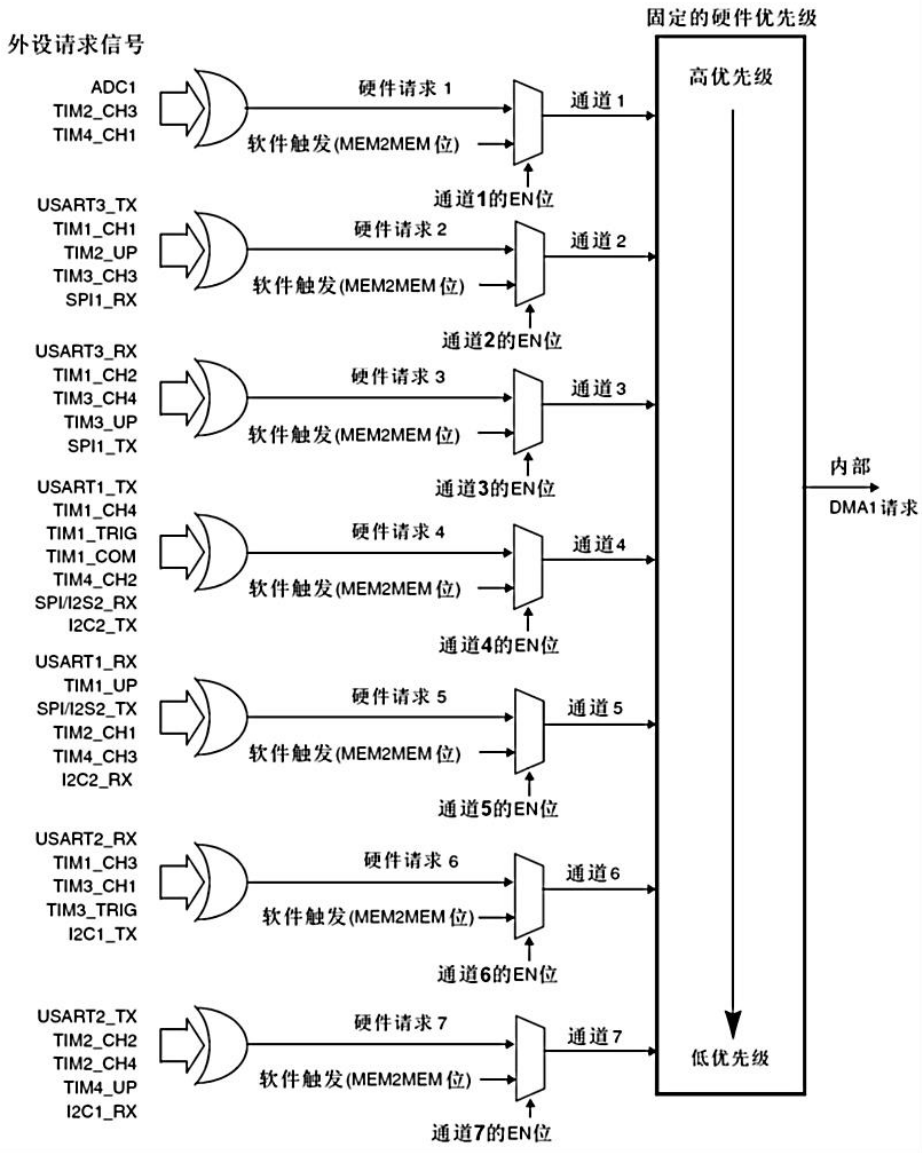
  SPI1

  SPI/I2S2

  I2Cx [x=1、 2]

  USARTx [x=1、 2、 3]

- 通过逻辑或输入到DMA1控制器，这意味着同时只能有一个请求有效。



DMA1请求映射

外设	通道1	通道2	通道3	通道4	通道5	通道6	通道7
ADC1	ADC1						
SPI/I2S		SPI1_RX	SPI1_TX	SPI/I2S2_RX	SPI/I2S2_TX		
USART		USART3_TX	USART3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
I2C				I2C2_TX	I2C2_RX	I2C1_TX	I2C1_RX
TIM1		TIM1_CH1	TIM1_CH2	TIM1_TX4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
TIM2	TIM2_CH3	TIM2_UP			TIM2_CH1		TIM2_CH2 TIM2_CH4
TIM3		TIM3_CH3	TIM3_CH4 TIM3_UP			TIM3_CH1 TIM3_TRIG	
TIM4	TIM4_CH4			TIM4_CH2	TIM4_CH3		TIM4_UP

DMA2

- DMA2控制器及相关请求仅存在于大容量STM32F103和互联型的STM32F105、STM32F107系列产品中。
- 有5个触发通道，可以分别从外设产生的5个请求

  TIMx [5、 6、 7、 8]

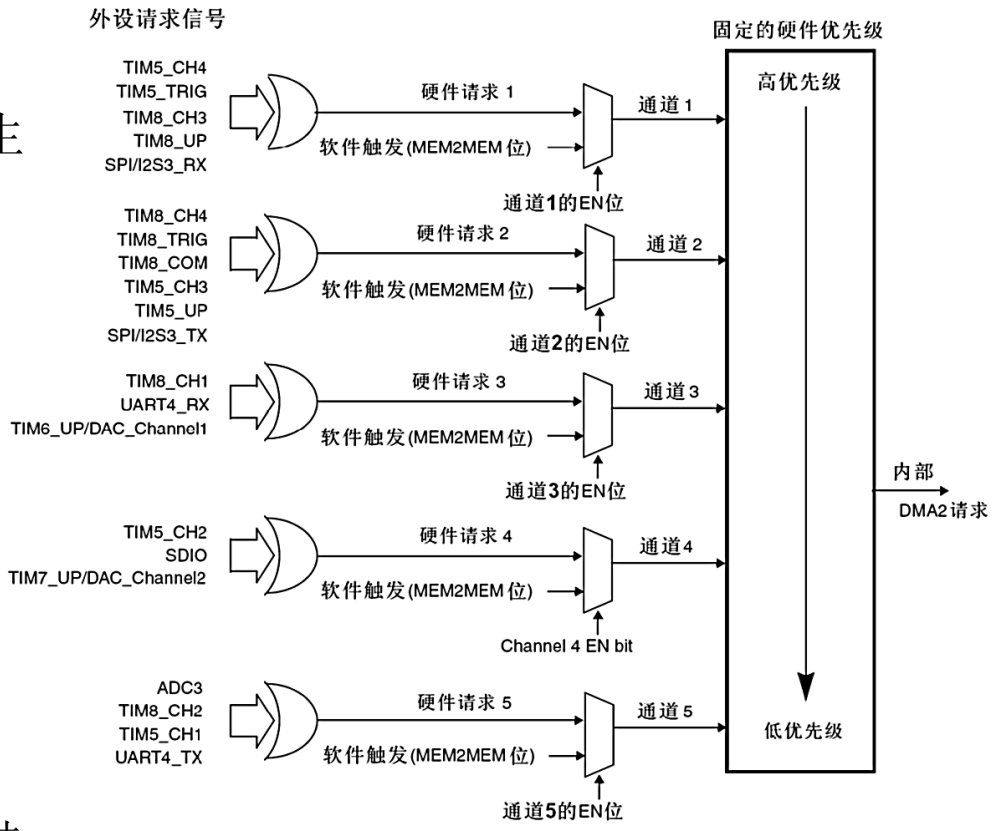
  ADC3

  SPI/I2S3

  UART4

  DAC通道1、 2

  SDIO



- 经逻辑或输入到DMA2控制器，这意味着同时只能有一个请求有效。

DMA2 请求映射

外设	通道1	通道2	通道3	通道4	通道5
ADC3					ADC3
SPI/I2S3	SPI/I2S3_RX	SPI/I2S3_TX			
UART4			UART4_RX		UART4_TX
SDIO(1)				SDIO	
TIM5	TIM5_CH4 TIM5_TRIG	TIM5_CH3 TIM5_UP		TIM5_CH2	TIM5_CH1
TIM6/ DAC通道1			TIM6_UP/ DAC通道1		
TIM7/ DAC通道2				TIM7_UP/ DAC通道2	
TIM8(1)	TIM8_CH3 TIM8_UP	TIM8_CH4 TIM8_TRIG TIM8_COM	TIM8_CH1		TIM8_CH2

8.4 DMA的常用库函数

函数 DMA_Init

函数名	DMA_Init
函数原形	void DMA_Init(DMA_Channel_TypeDef* DMAy_Channelx, DMA_InitTypeDef* DMA_InitStruct);
功能描述	根据 DMA_InitStruct 中指定的参数初始化 DMAy 的通道 x 寄存器
输入参数 1	DMAy_Channelx：DMAy的通道x 其中y 可以是1或2 对于DMA1，x可以是1~7，对于DMA2，x可以是1~5
输入参数 2	DMA_InitStruct：指向结构 DMA_InitTypeDef 的指针，包含了 DMAy 通道 x 的配置信息
输出参数	无
返回值	无

DMA_InitTypeDef定义于文件“stm32f10x_dma.h”：

```
typedef struct
{
    u32 DMA_PeripheralBaseAddr;     // DMA 外设基地址
    u32 DMA_MemoryBaseAddr;         // DMA内存基地址
    u32 DMA_DIR;                    //规定了外设是作为数据传输的目的地还是来源
    u32 DMA_BufferSize;  // 定义指定 DMA 通道的 DMA 缓存的大小，单位为数据单位
    u32 DMA_PeripheralInc;          // 用来设定外设地址寄存器递增与否
    u32 DMA_MemoryInc;              // 设定内存地址寄存器递增与否
    u32 DMA_PeripheralDataSize;     // 设定外设数据宽度
    u32 DMA_MemoryDataSize;         // 设定存储器数据宽度
    u32 DMA_Mode;                   // 设置 DMA 的工作模式：循环/正常
    u32 DMA_Priority;               // 设定 DMA 通道 x 的软件优先级
    u32 DMA_M2M;                    // 使能 DMA 通道的内存到内存传输
} DMA_InitTypeDef;
```

DMA_PeripheralBaseAddr： 该参数用以定义 DMA 外设基地址

DMA_MemoryBaseAddr： 该参数用以定义 DMA 内存基地址

DMA_DIR： 规定了外设是作为数据传输的目的地还是来源。

DMA_DIR	**描述**
DMA_DIR_PeripheralDST	外设作为数据传输的目的地
DMA_DIR_PeripheralSRC	外设作为数据传输的来源

DMA_BufferSize： 用以定义指定 DMA 通道的 DMA 缓存的大小，单位为数据单位。根据传输方向，数据单位等于结构中参数 DMA_PeripheralDataSize 或者参数 DMA_MemoryDataSize 的值。

DMA_PeripheralInc：用来设定外设地址寄存器递增与否

DMA_PeripheralInc	描述
DMA_PeripheralInc_Enable	外设地址寄存器递增
DMA_PeripheralInc_Disable	外设地址寄存器不变

DMA_MemoryInc：用来设定内存地址寄存器递增与否

DMA_MemoryInc	描述
DMA_MemoryInc_Enable	内存地址寄存器递增
DMA_MemoryInc_Disable	内存地址寄存器不变

DMA_PeripheralDataSize：设定了外设数据宽度。

DMA_PeripheralDataSize	描述
DMA_PeripheralDataSize_Byte	数据宽度为 8 位
DMA_PeripheralDataSize_HalfWord	数据宽度为 16 位
DMA_PeripheralDataSize_Word	数据宽度为 32 位

DMA_MemoryDataSize：设定了存储器数据宽度。

DMA_MemoryDataSize	描述
DMA_MemoryDataSize_Byte	数据宽度为 8 位
DMA_MemoryDataSize_HalfWord	数据宽度为 16 位
DMA_MemoryDataSize_Word	数据宽度为 32 位

DMA_Mode：设置了 DMA 的工作模式。

DMA_Mode	**描述**
DMA_Mode_Circular	工作在循环缓存模式
DMA_Mode_Normal	工作在正常缓存模式

DMA_Priority：设定 DMA 通道 x 的软件优先级

DMA_Mode	**描述**
DMA_Priority_VeryHigh	DMA 通道 x 拥有非常高优先级
DMA_Priority_High	DMA 通道 x 拥有高优先级
DMA_Priority_Medium	DMA 通道 x 拥有中优先级
DMA_Priority_Low	DMA 通道 x 拥有低优先级

DMA_M2M：使能 DMA 通道的内存到内存传输

DMA_M2M	描述
DMA_M2M_Enable	DMA 通道 x 设置为内存到内存传输
DMA_M2M_Disable	DMA 通道 x 没有设置为内存到内存传输

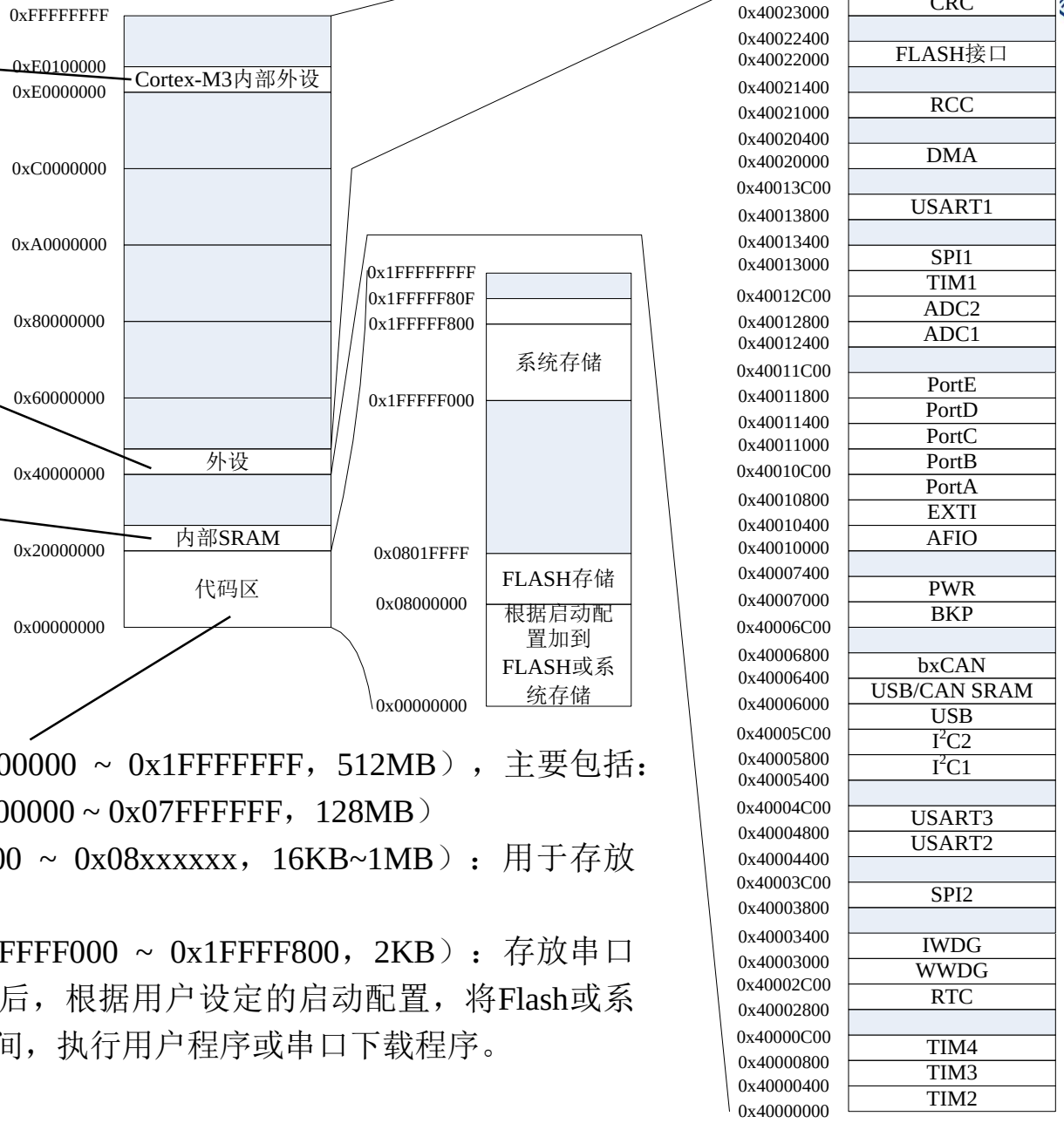
```
/* Initialize the DMA Channel1 according to the DMA_InitStructure members */
DMA_InitTypeDef DMA_InitStructure;

DMA_InitStructure.DMA_PeripheralBaseAddr = 0x40005400;
DMA_InitStructure.DMA_MemoryBaseAddr = 0x20000100;
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
DMA_InitStructure.DMA_BufferSize = 256;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
DMA_InitStructure.DMA_Priority = DMA_Priority_Medium;
DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;

DMA_Init(DMA_Channel1, &DMA_InitStructure);
```

嵌入式系统

内部外设区：用于调试组件等私有外设。例如：FPB, DWT, ITM, ETM, TPIU, ROM表等。

外设区：外设寄存器地址空间。

内部SRAM（0x20000000 ~ 0x200xxxxx，6~96KB）：保护程序运行时产生的临时数据的随机存储器。



代码区：（0x00000000 ~ 0x1FFFFFFF，512MB），主要包括：

启动空间（0x00000000 ~ 0x07FFFFF，128MB）

Flash（0x08000000 ~ 0x08xxxxxx，16KB~1MB）：用于存放用户编写的程序

系统存储区（0x1FFFF000 ~ 0x1FFFF800，2KB）：存放串口下载程序，当系统上电后，根据用户设定的启动配置，将Flash或系统存储区映射到启动空间，执行用户程序或串口下载程序。

函数DMA_Cmd

函数名	DMA_Cmd
函数原形	void DMA_Cmd(DMA_Channel_TypeDef* DMAy_Channelx, FunctionalState NewState);
功能描述	使能或者失能指定的通道 x
输入参数 1	DMAy_Channelx：选择 DMAy 通道 x
输入参数 2	NewState： DMA 通道 x 的新状态 这个参数可以取： ENABLE 或者 DISABLE
输出参数	无
返回值	无

函数 DMA_GetCurrDataCounter

函数名	DMA_GetCurrDataCounter
函数原形	u16 DMA_GetCurrDataCounter(DMA_Channel_TypeDef* DMAy_Channelx);
功能描述	返回当前 DMAy 通道 x 剩余的待传输数据数目
输入参数	DMAy_Channelx：选择 DMAy 通道 x
输出参数	无
返回值	当前 DMA 通道 x 剩余的待传输数据数目

```
/* Get the number of remaining data units in the current DMA Channel2 transfer */
 u16 CurrDataCount;
CurrDataCount = DMA_GetCurrDataCounter(DMA_Channel2);
```

函数DMA_GetFlagStatus

函数名	DMA_GetFlagStatus
函数原形	FlagStatus DMA_GetFlagStatus(uint32_t DMAy_FLAG);
功能描述	检查指定的 DMAy 通道 x 标志位设置与否
输入参数	DMAy_FLAG：待检查的DMAy 通道 x标志位 参阅 Section： DMA_FLAG 查阅更多该参数允许取值范围
输出参数	无
返回值	DMA_FLAG 的新状态（SET 或者 RESET）

```
/* Test if the DMA Channel6 half transfer interrupt flag is set or not */

FlagStatus Status;

Status = DMA_GetFlagStatus(DMA_FLAG_HT6);
```

DMAy_FLAG:参数 DMA_FLAG 定义了待检查的标志位类型

DMA_FLAG	描述
DMA1_FLAG_GL1	DMA1通道 1 全局标志位
DMA1_FLAG_TC1	DMA1通道 1 传输完成标志位
DMA1_FLAG_HT1	DMA1通道 1 传输过半标志位
DMA1_FLAG_TE1	DMA1通道 1 传输错误标志位
DMA1_FLAG_GL2	DMA1通道 2 全局标志位
DMA1_FLAG_TC2	DMA1通道 2 传输完成标志位
DMA1_FLAG_HT2	DMA1通道 2 传输过半标志位
DMA1_FLAG_TE2	DMA1通道 2 传输错误标志位
…… 依次类推	…… 依次类推

函数DMA_ClearFlag

函数名	DMA_ClearFlag
函数原形	void DMA_ClearFlag(u32 DMAy_FLAG)
功能描述	清除 DMAy 通道 x 待处理标志位
输入参数	DMAy_FLAG：待清除的 DMA 标志位，使用操作符“|”可以同时选中多个DMA 标志位
输出参数	无
返回值	无

函数DMA_ITConfig

函数名	DMA_ITConfig
函数原形	void DMA_ITConfig(DMA_Channel_TypeDef* DMAy_Channelx, u32 DMA_IT, FunctionalState NewState);
功能描述	使能或者失能指定的 DMAy 通道 x 中断
输入参数 1	DMAy_Channelx：选择 DMAy 通道 x
输入参数 2	DMA_IT：待使能或者失能的 DMA 中断源，使用操作符“|”可以同时选中多个 DMA 中断源
输入参数 3	NewState： DMA 通道 x 中断的新状态 这个参数可以取：ENABLE 或者 DISABLE

DMA_IT:

输入参数 DMA_IT 使能或者失能 DMA通道 x 的中断。

可以取下表的一个或者多个取值的组合作为该参数的值。

DMA_IT	描述
DMA_IT_TC	传输完成中断屏蔽
DMA_IT_HT	传输过半中断屏蔽
DMA_IT_TE	传输错误中断屏蔽

函数 DMA_GetITStatus

函数名	DMA_GetITStatus
函数原形	ITStatus DMA_GetITStatus(uint32_t DMAy_IT);
功能描述	检查指定的 DMAy 通道 x 中断发生与否
输入参数	DMAy_IT：待检查的 DMAy 的通道 x 中断源
输出参数	无
返回值	DMA_IT 的新状态（SET 或者 RESET）

DMAy_IT: 定义了待检查的 DMA 中断

DMAy_IT	描述
DMA1_IT_GL1	通道 1 全局中断
DMA1_IT_TC1	通道 1 传输完成中断
DMA1_IT_HT1	通道 1 传输过半中断
DMA1_IT_TE1	通道 1 传输错误中断
DMA1_IT_GL2	通道 2 全局中断
DMA1_IT_TC2	通道 2 传输完成中断
DMA1_IT_HT2	通道 2 传输过半中断
DMA1_IT_TE2	通道 2 传输错误中断
…… 依次类推	…… 依次类推

8.5 DMA使用流程

① 使能DMA时钟

RCC_AHBPeriphClockCmd();

② 初始化DMA通道参数

DMA_Init();

③ 使能DMA1通道，启动传输。

DMA_Cmd();

④ 查询DMA传输状态

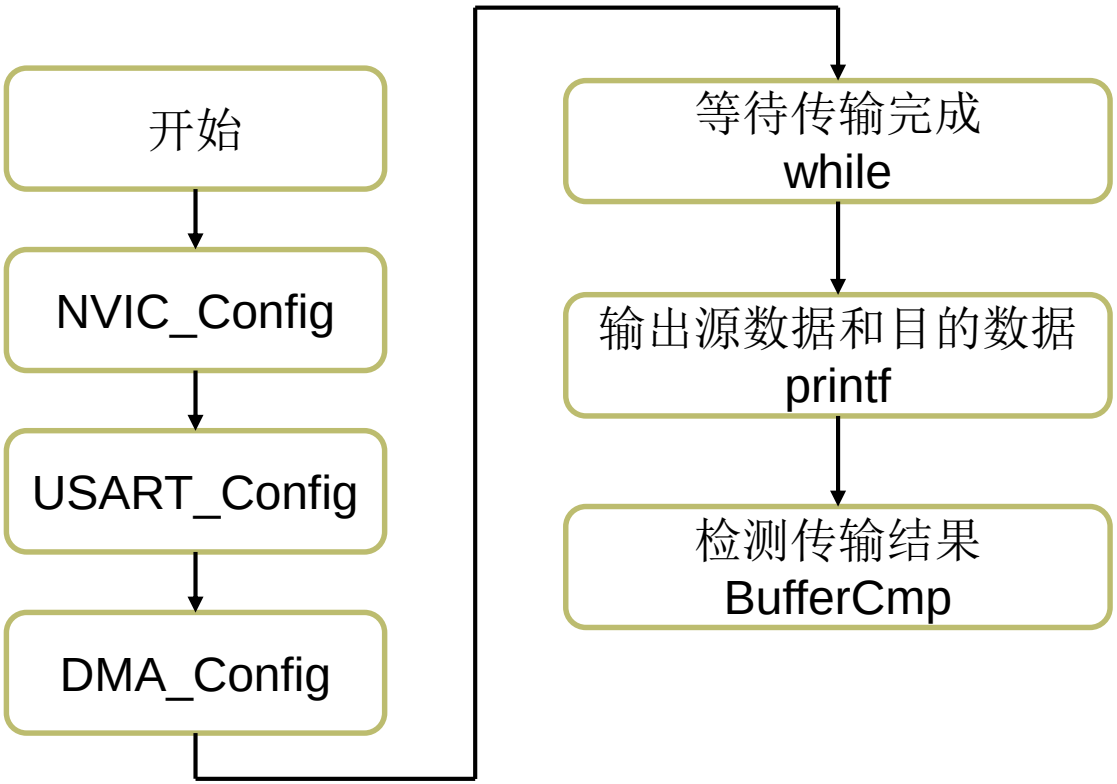
DMA_GetFlagStatus();

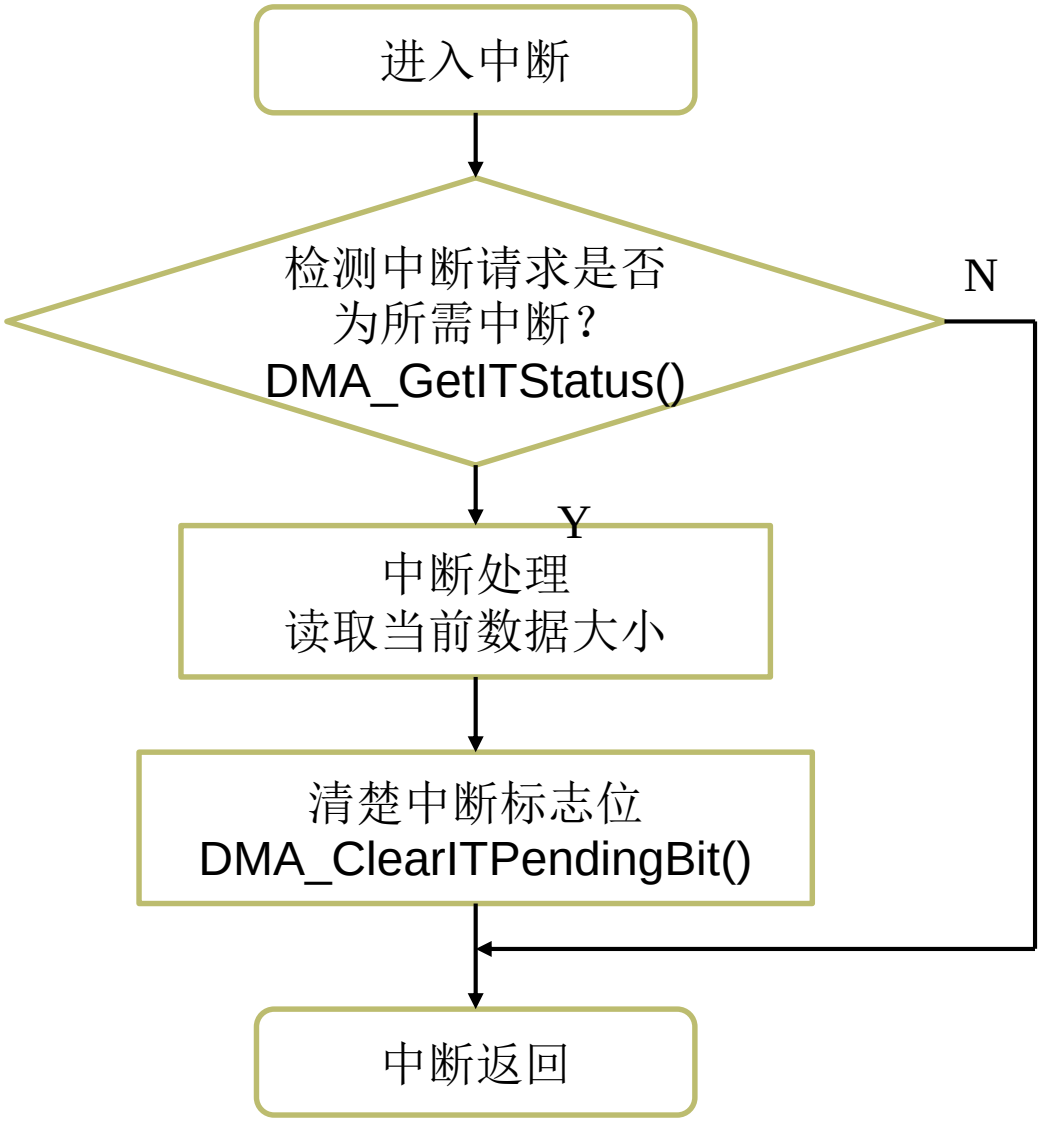
⑤ 获取/设置通道当前剩余数据量：

DMA_GetCurrDataCounter();

8.6 DMA应用设计实例：片内Flash到片内RAM数据传输

- 设置DMA通道6，实现Flash到RAM的DMA传输。
- 通过串口将传输的状态及内容输出。
- 启动DMA，传输结束后比较源数据和目的数据，检测传输结果。




进入中断
检测中断请求是否
为所需中断？
DMA_GetITStatus()
N
Y
中断处理
读取当前数据大小
清楚中断标志位
DMA_ClearITPendingBit()
中断返回

main.c 文件

```
#include “stm32f10x.h”
#include <stdio.h>
#define BufferSize 32
typedef enum{
    FAILED = 0,
    PASSED = ! FAILED
} TestStatus;

__IO uint32_t CurrDataCounterBegin = 0;       //传输前通道数据量
__IO uint32_t CurrDataCounterEnd = 0x01;      //传输后通道数据量
const uint32_t SRC_Const_Buffer[BufferSize] = {
0x01020304, 0x05060708, 0x090A0B0C, 0x0D0E0f=F10,
…
0x71727374, 0x75767778, 0x797A7B7C, 0x7D7E7F80 };
uint_32t DST_Buffer[BufferSize];
void NVIC_Configure(void);
void DMA_Configure(void);
void USART_Configure(void);
void USART_GPIO_Configure(void);
TestStatus_Buffercmp( const uint32_t* Buffer_SRC, uint32_t* Buffer_DST, uint16_t BufferLength);
void delay_ms(int32_t ms);
```

main.c 文件

```
int main( )
{
    uint32_t count;     //数据计数
    TestStatus TransferStatus = FAILED;    // 发送状态标志
    NVIC_Configure();
    USART_Configure();
    DMA_Configure();
    delay_ms(1000);      //延时1s，便于观察打印数据
     printf("\n ---------------DMA Test---------------");
     printf("\n -----------Complete Initialization----------");
     while(CurrDataCounterEnd != 0)
           ;                              //等待传送结束
     printf("\n ---------------Complete Transmission---------------");
     打印源数据内容
     打印目的数据内容
     TransferStatus = Buffercmp(SRC_Const_Buffer, DST_Buffer, BufferSize);
      串口打印测试结果
     while(1)
        ;        /                       /无限循环
}
```

main.c 文件

```
void NVIC_Configure(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
    //允许DMA1通道6中断
    NVIC_InitStructure.NVIC_IRQChannel = DMA1_Channel6_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

main.c 文件

```
void USART_Configure()
{
    USART_InitTypeDef  USART_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE):
    USART_GPIO_Configure();
    //配置波特率、数据位、停止位、奇偶校验、硬件流控制模式
    USART_InitStructure.USART_BaudRate = 9600;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl =  USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;  //接收和发送
    USART_Init(USART1, &USART_InitStructure);
    //允许接收中断
    ?? USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
    //使能USART1
    USART_Cmd(USART1, ENABLE);
}
```

main.c 文件

```
void USART_GPIO_Configure(void)
{
    GPIO_InitTypeDef  GPIO_InitStructure;
    //打开GPIOA、AFIO和USART1时钟
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA |
                RCC_APB2Periph_AFIO, ENABLE);
    //配置PA9（USART_Tx)为复用推挽输出
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    //配置PA10（USART_Rx）为浮空输入
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
}
```

main.c 文件

```
//将C库中printf函数重定向到USART, fputc函数包含于stdio.h
int fputc(int ch, FILE* f)
{
    USART_SendData(USART1, (u8)ch); // 串口发送数据
    while(!(USART_GetFlagStatus(USART1, USART_FLAG_TXE == SET))
        ;               //等待发送完成
    return ch;
}

// 延时函数
void delay_ms()
{
    int32_t i;
    while(ms--)
    {
        i =7500;        //晶振8MHz时的经验值
        while(i--);
    }
}
```

main.c 文件

```
void DMA_Configure()
{
    DMA_Init_TypeDef DMA_InitStructure;
    //打开DMA时钟
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
    //Flash外设基地址为SRC_Const_Buffer
     DMA_InitStructure.DMA_PeripheralBaseAddr=(uint32_t)SRC_Const_Buffer;
     //RAM基地址为DST_Buffer
     DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t) DST_Buffer;
     //传输方向，FLASH -> RAM
     DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
     //缓冲区大小，0~65536范围，此处为32
     DMA_InitStructure.DMA_BufferSize = BufferSize;
      //外设和RAM地址自增1
     DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_ Enable;
     DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
      //外设和RAM数据宽度，32位
     DMA_InitStructure.DMA_PeripheralDataSize=DMA_PeripheralDataSize_Word;
     DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
```

main.c 文件

```
    //传输模式
    DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
    //优先级高
    DMA_InitStructure.DMA_Priority = DMA_Priority_High;
     //内存到内存传输
    DMA_InitStructure.DMA_M2M = DMA_M2M_Enable;
     //完成配置
    DMA_Init(DMA1_Channel6， &DMA_InitStructure);
     //允许传输完成中断
    DMA_ITConfig(DMA1_Channel6, DMA_IT_TC, ENABLE);
     //传输前数据量
    CurrDataCounterBegin = DMA_GetCurrDataCounter(DMA1_Channel6);
     //开启DMA
    DMA_Cmd(DMA1_Channel6, ENABLE);
}
```

Stm32f10x_it.c 文件

```
//DMA1通道6中断服务程序
void DMA1_Channel6_IRQHandler(void)
{
    //检测DMA1 Channel6 传输完成中断
    if(DMA_GetITStatus(DMA1_IT_TC6))
    {
        //传输完成后得到当前计数
        CurrDataCounterEnd = DMA_GetCurrDataCounter(DMA1_Channel6);
        //清除中断
        DMA_ClearITPendingBit(DMA1_IT_GL6);
     }
}
```

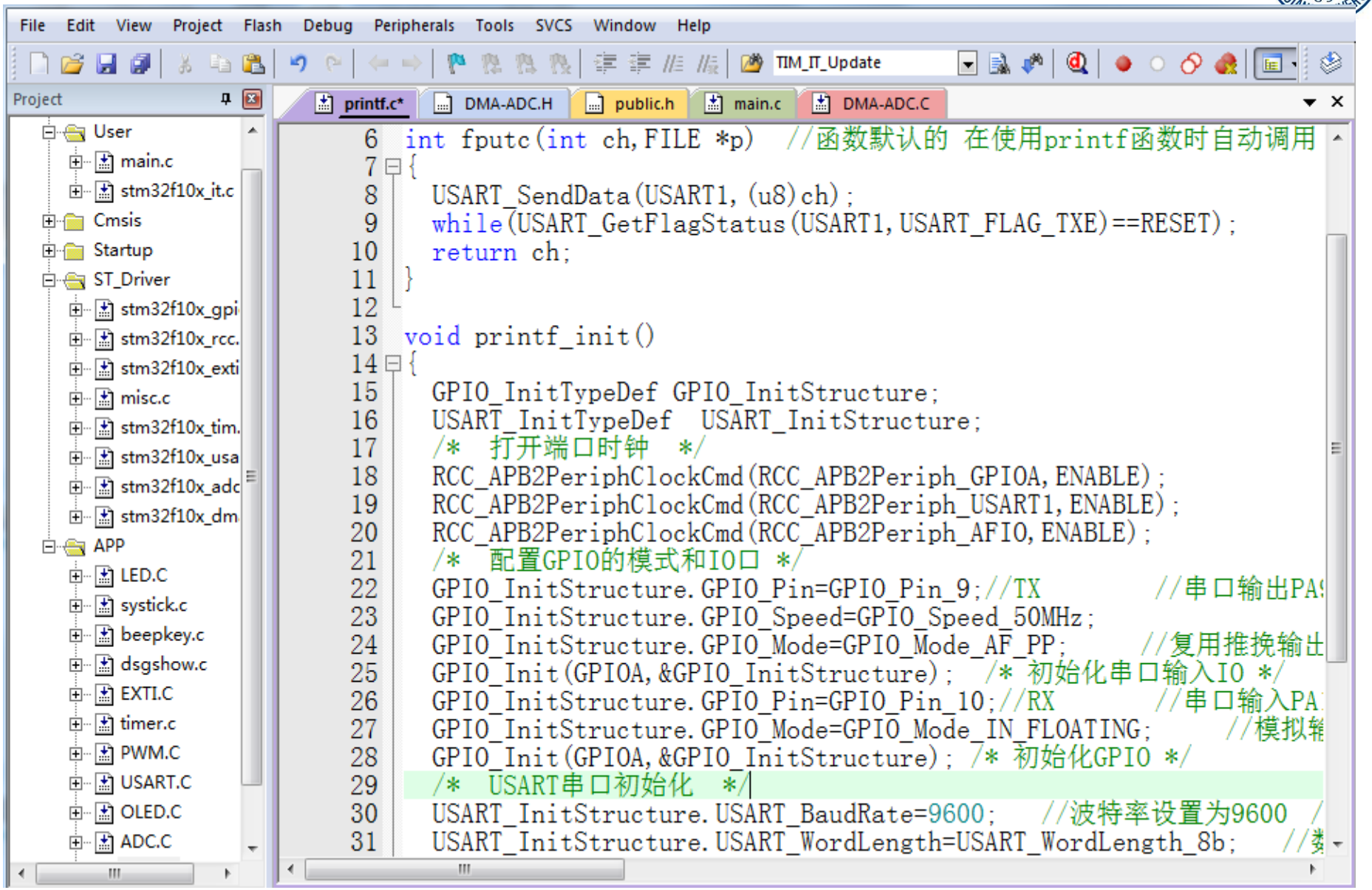
DMA_IT	描述
DMA_IT_TC	传输完成中断屏蔽
DMA_IT_HT	传输过半中断屏蔽
DMA_IT_TE	传输错误中断屏蔽

main.c 文件

```
//比较传输结果

TestStatus Buffercmp(const uint32_t* Buffer_SRC, uint32_t* Buffer_DST, uint16_t
BufferLength)
{
    //逐位比较，有不同则返回FAILED
    while(BufferLength--)
    {
        if(*Buffer_SRC != *Buffer_DST)
        {
            return FAILED;
         }
         Buffer_SRC++;
         Buffer_DST++;
     }
     return PASSED;
}
```

重写printf函数

- 由于需要将结果打印到PC机的串口，所以将标准C的printf重写，利用微控制器的串口将数据发送至PC机，数据发送方法和我们在标准C语言中使用printf函数向显示屏打印信息是一样的。
- 在printf.c文件中输入如下源程序，在程序中首先包含printf.h头文件，然后重写fputc函数，并对串口USART1进行初始化。


File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
TIM_IT_Update
Project
User
main.c
stm32f10x_it.c
Cmsis
Startup
ST_Driver
stm32f10x_gpi
stm32f10x_rcc.
stm32f10x_exti
misc.c
stm32f10x_tim.
stm32f10x_usa
stm32f10x_adc
stm32f10x_dm
APP
LED.C
systick.c
beepkey.c
dsgshow.c
EXTI.C
timer.c
PWM.C
USART.C
OLED.C
ADC.C
printf.c*
DMA-ADC.H
public.h
main.c
DMA-ADC.C
6 int fputc(int ch,FILE *p) //函数默认的 在使用printf函数时自动调用
7 {
8 USART_SendData(USART1,(u8)ch);
9 while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
10 return ch;
11 }
12
13 void printf_init()
14 {
15 GPIO_InitTypeDef GPIO_InitStructure;
16 USART_InitTypeDef USART_InitStructure;
17 /* 打开端口时钟 */
18 RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
19 RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);
20 RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);
21 /* 配置GPIO的模式和IO口 */
22 GPIO_InitStructure.GPIO_Pin=GPIO_Pin_9;//TX //串口输出PA
23 GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
24 GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AF_PP; //复用推挽输出
25 GPIO_Init(GPIOA,&GPIO_InitStructure); /* 初始化串口输入IO */
26 GPIO_InitStructure.GPIO_Pin=GPIO_Pin_10;//RX //串口输入PA
27 GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IN_FLOATING; //模拟输
28 GPIO_Init(GPIOA,&GPIO_InitStructure); /* 初始化GPIO */
29 /* USART串口初始化 */
30 USART_InitStructure.USART_BaudRate=9600; //波特率设置为9600
31 USART_InitStructure.USART_WordLength=USART_WordLength_8b;

在printf.h文件中输入如下源程序，其中条件编译格式不变，只要更改一下预定义变量名称即可，需要将我们刚定义函数的声明加到头文件当中。

printf.c | DMA-ADC.H | public.h | main.c

```
/*****************************************
#ifndef _printf_H
#define _printf_H
#include "stm32f10x.h"
#include "stdio.h"
int fputc(int ch,FILE *p) ;
void printf_init(void);
#endif
```

在public.h文件的中间部分添加“#include "printf.h"”语句，即包含printf.h头文件。

例2 存储器和外设之间的传输

- 以DMA方式将Flash中的数据发送至USART1 。

```
const uint32_t SRC_Const_Buffer[BufferSize] = {
0x01020304, 0x05060708, 0x090A0B0C, 0x0D0E0f=F10,
…
0x71727374, 0x75767778, 0x797A7B7C, 0x7D7E7F80 };

void NVIC_Configure(void);
void DMA_Configure(void);
void USART_Configure(void);
void USART_GPIO_Configure(void);
```

```
int main( )
{
    NVIC_Configure();
    USART_Configure();
    DMA_Configure();

    // 使能USART1的DMA发送请求
    USART_DMACmd(USART1, USART_DMAReq_Tx, ENABLE);
    DMA_Cmd(DMA1_Channel4, ENABLE);
    while(1)
        ;
}
```

Table 723. 函数 USART_ DMACmd

函数名	USART_ DMACmd
函数原形	USART_DMACmd(USART_TypeDef* USARTx, FunctionalState NewState)
功能描述	使能或者失能指定 USART 的 DMA 请求
输入参数 1	USARTx：x 可以是 1，2 或者 3，来选择 USART 外设
输入参数 2	USART_DMAreq：指定 DMA 请求 参阅 Section：USART_DMAreq 查阅更多该参数允许取值范围
输入参数 3	NewState: USARTx DMA 请求源的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无

```
void NVIC_Configure(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
    //允许DMA1通道4中断
    NVIC_InitStructure.NVIC_IRQChannel = DMA1_Channel4_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

外设	通道1	通道2	通道3	通道4	通道5	通道6	通道7
ADC1	ADC1						
SPI/I2S		SPI1_RX	SPI1_TX	SPI/I2S2_RX	SPI/I2S2_TX		
USART		USART3_TX	USART3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
I2C				I2C2_TX	I2C2_RX	I2C1_TX	I2C1_RX
TIM1		TIM1_CH1	TIM1_CH2	TIM1_TX4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
TIM2	TIM2_CH3	TIM2_UP			TIM2_CH1		TIM2_CH2 TIM2_CH4
TIM3		TIM3_CH3	TIM3_CH4 TIM3_UP			TIM3_CH1 TIM3_TRIG	
TIM4	TIM4_CH4			TIM4_CH2	TIM4_CH3		TIM4_UP

main.c 文件

```
void USART_Configure()
{
    USART_InitTypeDef  USART_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE):
    USART_GPIO_Configure();
    //配置波特率、数据位、停止位、奇偶校验、硬件流控制模式
    USART_InitStructure.USART_BaudRate = 9600;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl =  USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Tx;   //仅发送
    USART_Init(USART1, &USART_InitStructure);

    USART_Cmd(USART1, ENABLE);
}
```

```
void USART_GPIO_Configure(void)
{
    GPIO_InitTypeDef  GPIO_InitStructure;
    //打开GPIOA、AFIO和USART1时钟
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA |
                RCC_APB2Periph_AFIO, ENABLE);
    //配置PA9（USART_Tx)为复用推挽输出
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
}
```

```
void DMA_Configure()
{
    DMA_Init_TypeDef DMA_InitStructure;
    //打开DMA时钟
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
    //Flash外设基地址为USART1的DR寄存器地址
     DMA_InitStructure.DMA_PeripheralBaseAddr=(uint32_t) &USART1->DR;
     //Flash基地址为SRC_Const_Buffer
     DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t) SRC_Const_Buffer;
     //传输方向，FLASH -> USART
     DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralDST;
     //缓冲区大小，0~65536范围，此处为32
     DMA_InitStructure.DMA_BufferSize = BufferSize;
      //外设和RAM地址自增1
     DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
     DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
      //外设和RAM数据宽度，32位
     DMA_InitStructure.DMA_PeripheralDataSize=DMA_PeripheralDataSize_Word;
     DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
```

```
    //传输模式
    DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
    //优先级高
    DMA_InitStructure.DMA_Priority = DMA_Priority_High;
     //内存到内存传输
    DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;

    //完成配置
    DMA_Init(DMA1_Channel4，  &DMA_InitStructure);
     //允许传输完成中断
    DMA_ITConfig(DMA1_Channel4, DMA_IT_TC, ENABLE);
}
```

Stm32f10x_it.c 文件

```
//DMA1通道4中断服务程序
void DMA1_Channel4_IRQHandler(void)
{
    //检测DMA1 Channel4 传输完成中断
    if(DMA_GetITStatus(DMA1_IT_TC4))
    {
        USART_DMACmd(USART1, USART_DMAReq_Tx, DISABLE);
        DMA_ClearITPendingBit(DMA1_IT_TC4);
     }
}
```