**DMA**

**第一步：**了解DMA相关理论知识

直接存储器访问DMA作用：存储器与外设间提供高速数据传输。传输数据时无需占用CPU，CPU可执行其他操作。DMA通道可以访问任何存储器所映射的空间。DMA是AHB总线上的主设备，并具有灵活的配置，此外还提供硬件和软件优先级管理。

DMA三种传输模式：外设到外设、外设到存储器、存储器到外设以及存储器到存储器。

DMA支持不同外设的请求：定时器、ADC和通信外设。简化CPU负载并简化集成。

处理DMA传输的两个单元：DMA请求复用器即DMAMUX和DMA控制器。

DMA控制器将数据从源地址传输到目标地址，并管理不同通道的优先级。

DMAMUX使用户能够将请求映射到相应的DMA通道，DMAMUX还处理触发信号和同步。DMAMUX允许在STM32G0外设与DMA控制器之间实现DMA请求线的路由选择。该路由功能通过可编程的多通道DMA请求先复用器来确保实现，每个通道将无条件的配合DMAMUX同步输入事件选择唯一的DMA请求线。DMAMUX还可以用作DMA请求生成器，基于其输入触发信号的可编程事件，并生成DMA请求。DMAMUX用于将外设请求映射到7个可用的DMA通道上（这种映射是可编程的）。此外，DMAMUX内置4通道请求生成器，可将触发信号转换为DMA请求。支持的触发如下：16种外部中断、低功耗定时器1事件、两种超时事件、定时器14输出比较事件和DMAMUX自身生成的4种事件。

STM32G0中DMA和DMAMUX的特性总结：DMAMUX有57个外设请求映射到7个DMA通道。DMAMUX有4个请求生成器通道、23个触发信号输入和23个同步输入的外设请求。

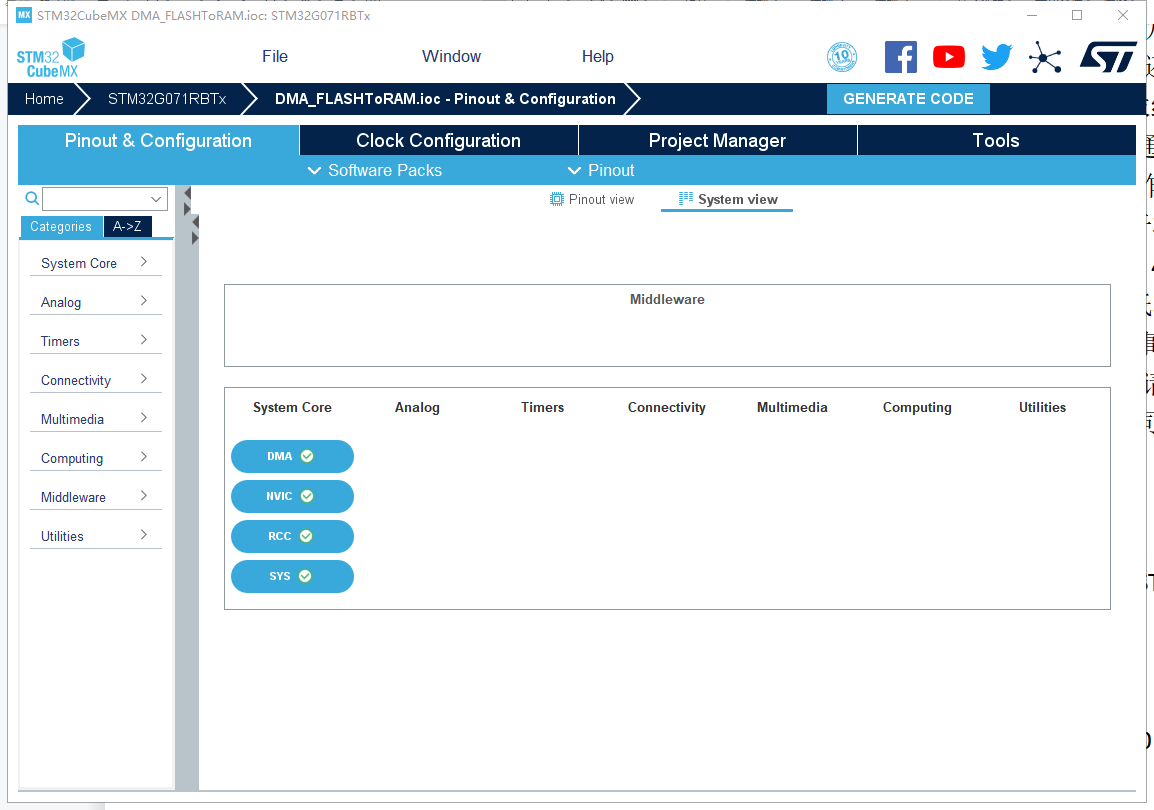
【Flash到内部SRAM的DMA传输】

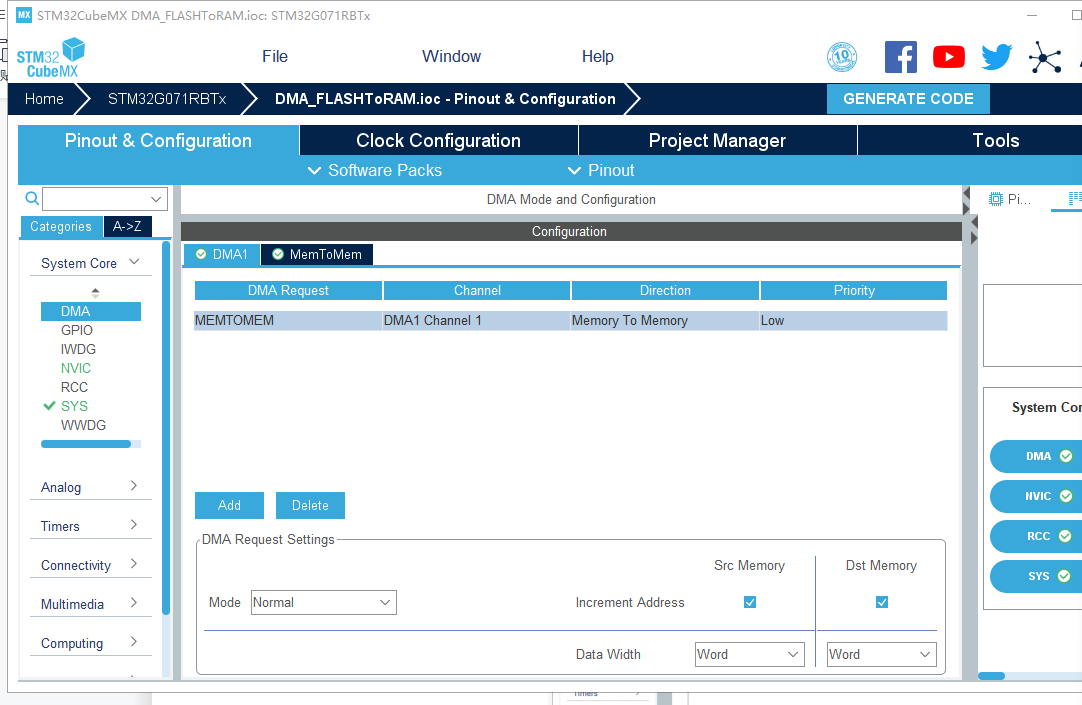
**第二步：**分析DMA样例程序初始配置

①打开样例程序，路径：C:\Users\李天凌\STM32Cube\Repository\STM32Cube\_FW\_G0

\_V1.4.0\Projects\NUCLEO-G071RB\Examples\DMA\DMA\_FLASHToRAM

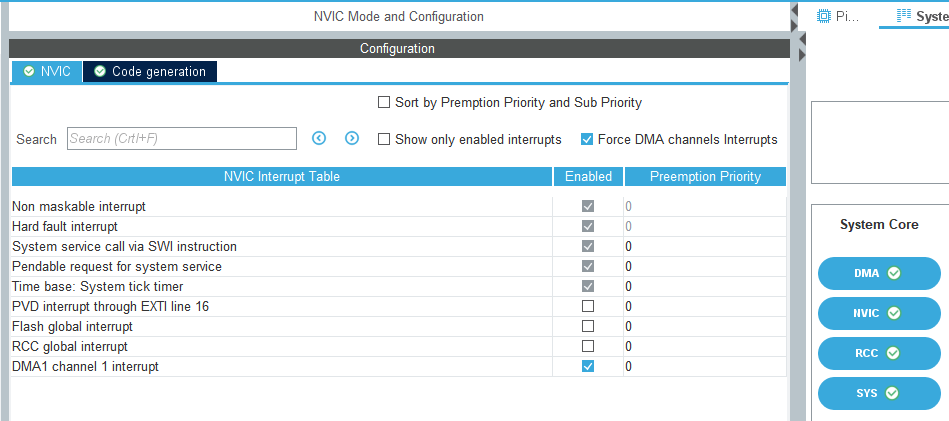
②在系统试图下点击DMA查看相关配置





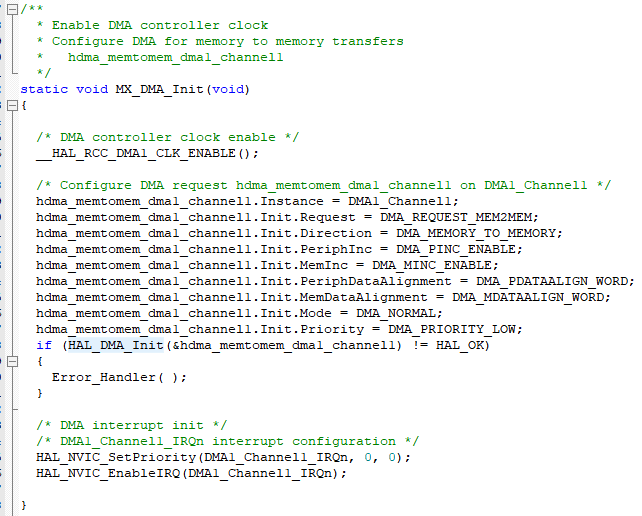
如上图所示，所用通道为DMA1通道1，方向是存储器到存储器，优先级为低。模式为正常，每次传输后“源存储器”和“目标存储器”的地址均递增，传输的“数据宽度”为字宽即32位。

③在系统试图下点击NVIC查看相关配置



从上图中，可以看到，它的DMA1通道1被使能。

④以上图形化配置可在工程文件主函数下找到其DMA初始化函数：

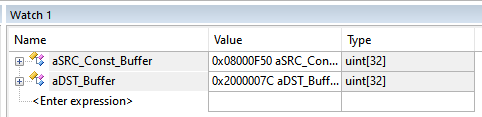


由上图可以看出，DMA的相关设置与预设值相符。

**第三步：**编译调试样例程序

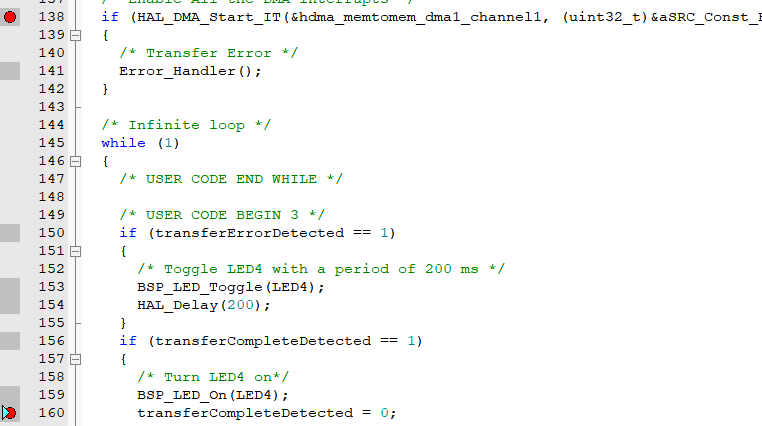
①在调试界面将源存储器aSRC\_Const\_Buffer和目标存储器aDST\_Buffer添加到可视化窗口。方法：找到此变量名，选中，右键单击，选择添加到Watch1窗口。

添加完毕后效果如下图所示：



②在main主函数添加两个断点，如下图所示

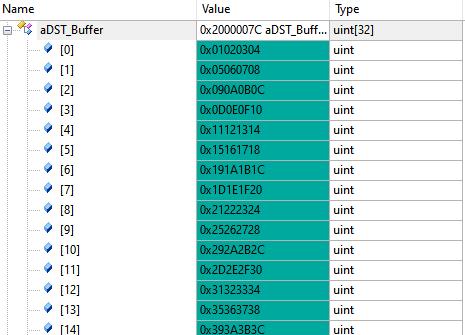
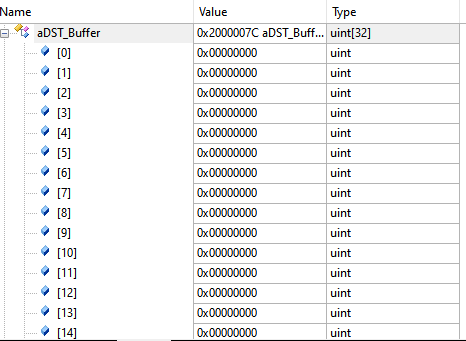
注：第一个断点在传输完成之前，第二个断点在传输完成之后。



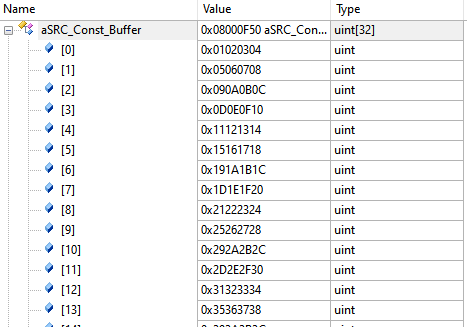
③调试查看结果

当运行到第一个断点时，目标缓冲区的值都将被清零。如下图（左）所示：

当运行到第二个断点时，目标缓冲区将被填充。如下图（右）所示：



现在，查看源缓冲区的值，如下图：



综上所述：DMA传输将源缓冲区的数据拷贝到了目标缓冲区。即实现了Flash缓冲区到RAM缓冲区的传输。

**第四步：**自己创建DMA工程

①在图形化配置界面按上述过程，分别配置DMA和NVIC。此外将PA5引脚设置为GPIO\_OUTPUT模式，用以指示数据传输过程。

②生成代码

③定义缓存区域大小 #define BUFFER\_SIZE 32

④定义并初始化源缓存区，定义目标缓存区，设定标志位

static const uint32\_t aSRC\_Const\_Buffer[BUFFER\_SIZE] =

{

0x01020304, 0x05060708, 0x090A0B0C, 0x0D0E0F10,

0x11121314, 0x15161718, 0x191A1B1C, 0x1D1E1F20,

0x21222324, 0x25262728, 0x292A2B2C, 0x2D2E2F30,

0x31323334, 0x35363738, 0x393A3B3C, 0x3D3E3F40,

0x41424344, 0x45464748, 0x494A4B4C, 0x4D4E4F50,

0x51525354, 0x55565758, 0x595A5B5C, 0x5D5E5F60,

0x61626364, 0x65666768, 0x696A6B6C, 0x6D6E6F70,

0x71727374, 0x75767778, 0x797A7B7C, 0x7D7E7F80

};

static uint32\_t aDST\_Buffer[BUFFER\_SIZE];

static \_\_IO uint32\_t transferErrorDetected; // 传输错误置一

static \_\_IO uint32\_t transferCompleteDetected; // 传输完成置一

⑤声明（主函数之前）并编写（主函数之后）函数设置其标志位

static void TransferComplete(DMA\_HandleTypeDef \*hdma\_memtomem\_dma1\_channel1);

static void TransferError(DMA\_HandleTypeDef \*hdma\_memtomem\_dma1\_channel1);

static void TransferComplete(DMA\_HandleTypeDef \*hdma\_memtomem\_dma1\_channel1)

{

transferCompleteDetected = 1;

}

static void TransferError(DMA\_HandleTypeDef \*hdma\_memtomem\_dma1\_channel1)

{

transferErrorDetected = 1;

}

⑥编写主函数非循环部分

transferErrorDetected = 0; //初始标志为0没有传输错误

transferCompleteDetected = 0; //初始标志为0没有传输完成

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_5,0); //LD4初始状态为灭——未开始传输

HAL\_DMA\_RegisterCallback(&hdma\_memtomem\_dma1\_channel1, HAL\_DMA\_XFER\_CPLT\_CB\_ID, TransferComplete);

HAL\_DMA\_RegisterCallback(&hdma\_memtomem\_dma1\_channel1, HAL\_DMA\_XFER\_ERROR\_CB\_ID, TransferError);

if (HAL\_DMA\_Start\_IT(&hdma\_memtomem\_dma1\_channel1, (uint32\_t)&aSRC\_Const\_Buffer, (uint32\_t)&aDST\_Buffer, BUFFER\_SIZE) != HAL\_OK)

Error\_Handler();

⑦编写主函数循环部分

if (transferErrorDetected == 1)

{

//传输出错LD4灯闪烁（间隔0.2s）

HAL\_GPIO\_TogglePin(GPIOA,GPIO\_PIN\_5);

HAL\_Delay(200);

}

if (transferCompleteDetected == 1)

{

//传输完成LD4保持常亮

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_5,1);

transferCompleteDetected = 0;

}

**第五步：**编译调试

步骤：编译→打开debug→设置2个断点→将两个缓存区添加到可视化窗口→调试

**第六步：**附录——函数执行过程

初始LD4灭，if处跳转

433 HAL\_DMA\_Start\_IT

1069 DMA\_SetConfig

457 HAL\_DMA\_Start\_IT

{483\_\_HAL\_DMA\_ENABLE(hdma);} //执行完后目标缓存区数据传输完成

跳回主函数while(1)循环，LD4保持常亮。