扩展点5——通过DMA加快数据传输

知识笔记

1. DMA功能

CPU作为整个芯片的核心，负责各个组件的协调和交互，其处理的工作量是很大的。搬运数据只是CPU很次要的一种工作。CPU最重要的工作是进行数据运算，负责中断申请和响应等。 而数据搬运却可能占用大部分的CPU资源，成为了降低CPU的工作效率的主要原因之一。于是需要一种硬件结构分担CPU这一功能——DMA。

2. DMA转运流程

从数据搬运的角度看，如果要把存储地址A的数值赋给另外一个地址上B的变量，CPU实现过程为首先读出A地址上的数据存储在一个中间变量，然后再转送到B地址的变量上。使用DMA则不需要中间变量，直接将A地址的数值传送到B地址的变量里。

3. DMA资源

STM32 DMA资源：12个独立可配置的通道： DMA1（7个通道），DMA2（5个通道） STM32F103C8T6 DMA资源：DMA1（7个通道）

4. DMA对象

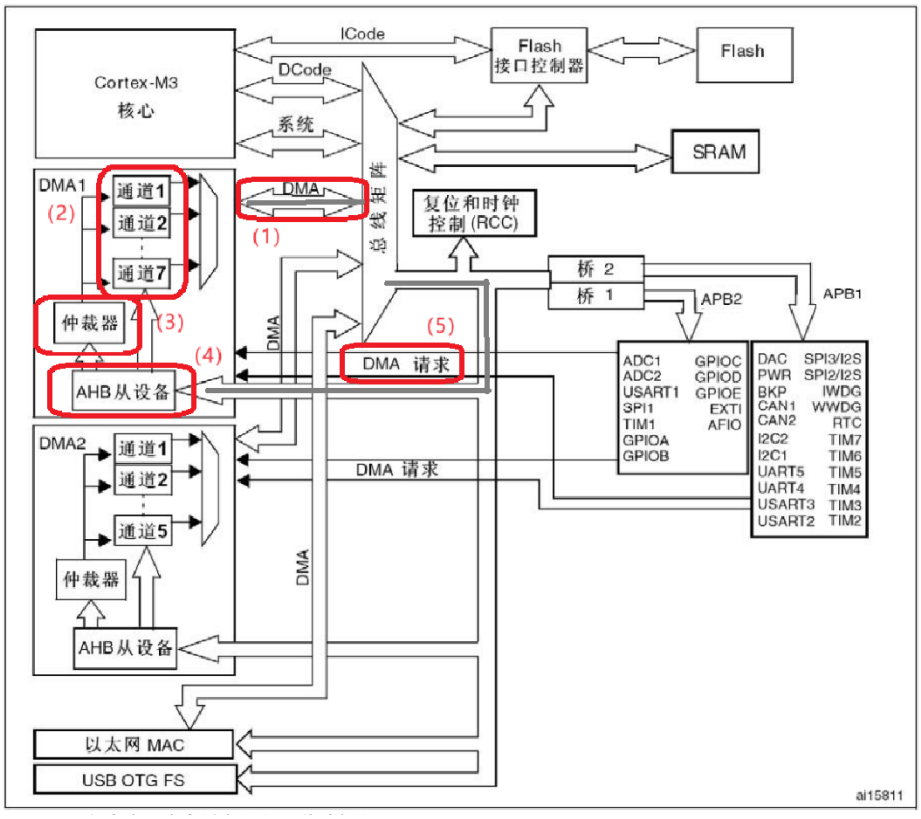
两类：外设寄存器和存储器，存储器和存储器

存储器包括运行内存SRAM、程序存储器Flash和寄存器等

外设寄存器和存储器一般用硬件触发，存储器和存储器一般用软件触发。

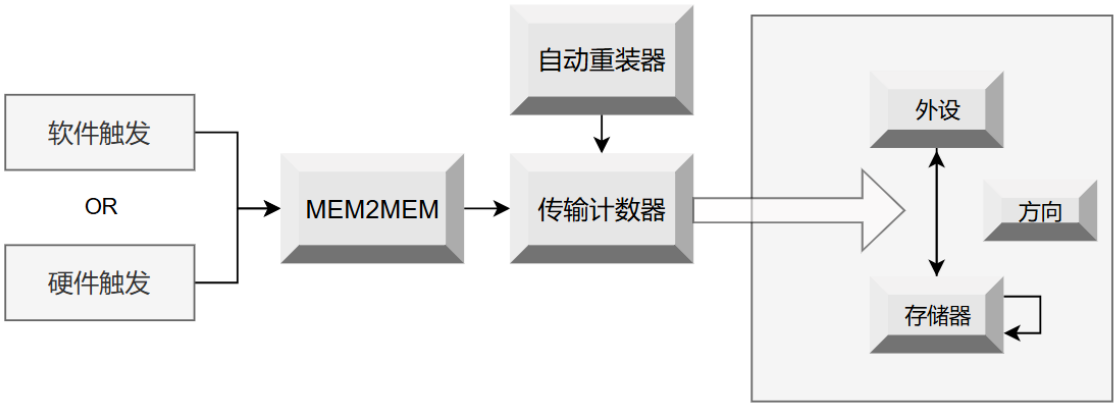
5. DMA结构关系

图21　DMA框图



- 总线矩阵左边：主动单元，可读写右边的存储器（包括寄存器） 总线矩阵右边：被动单元，只能被左边单元读写 DMA模块连接了左边，证明了它有读写存储器和寄存器的访问权限。 DMA模块同时连接了右边，也证明了它作为一个外设，也有相应的需要被读写配置的寄存器。
- 系统结构中与DMA有关的五个部分 (1) 用于访问存储器的DMA总线
  (2) 内部可独立转运的多通道
  (3) 仲裁多个通道的优先级，避免冲突 (4) 配置DMA参数 (5) 硬件触发DMA数据转运

5. DMA内部结构
按钮型的部分是库函数配置DMA时的重点



6. DMA转运双方的三大参数
起始地址，数据宽度，地址是否自增

- 起始地址决定了DMA转运的发送地和接受地
- 数据宽度影响了数据接受的方式：有字节（8bit），半字（16bit），全字（32bit）
- 地址是否自增决定了1.源端是否重复发送值 2.接收端是否覆盖接收值

7. 存储器映射
可以通过取地址观察存储器的映射地址，其中变量存在运行内存SRAM中。 要想得到某外设寄存器的基地址，可以先查询外设的初始地址，再加上偏移即可。

存储器类型	起始地址(0x)	存储器
ROM	0800 0000	程序存储器Flash
ROM	1FFF F000	系统存储器
ROM	1FFF F800	选项字节
RAM	2000 0000	运行内存SRAM
RAM	4000 0000	外设寄存器
RAM	E000 0000	内核外设寄存器

实现步骤

DMA工作的三个条件：触发源有触发信号，传输计数器大于零，DMA使能，对应步骤23.

1. AHB开启时钟（DMA是挂载在AHB总线上的设备）

```
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
```

2. 初始化DMA：对应上面的DMA内部结构图，定义初始化结构体，配置成员变量（与GPIO初始化类似）

```
函数：void MyDMA_Init(uint32_t AddrA, uint32_t AddrB, uint16_t Size)
//定义初始化结构体
DMA_InitTypeDef DMA_InitStructure;
//外设站点的三大参数：起始地址，数据宽度，地址是否自增
DMA_InitStructure.DMA_PeripheralBaseAddr = AddrA;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Enable;
//存储器站点的三大参数：起始地址，数据宽度，地址是否自增
DMA_InitStructure.DMA_MemoryBaseAddr = AddrB;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Byte;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
//DMA传输方向：外设站点时源端还是目的地，DST or SRC
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
//传输计数器：范围0~65535
DMA_InitStructure.DMA_BufferSize = Size;
//是否自动重装
DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
//选择触发方式："memory to memory" 存储器到存储器
DMA_InitStructure.DMA_M2M = DMA_M2M_Enable;
//优先级
DMA_InitStructure.DMA_Priority = DMA_Priority_Medium;
DMA_Init(DMA1_Channel1, &DMA_InitStructure);
```

注意事项：

- 自动重装和软件触发不能同时使用，否则DMA无休止工作

3. DMA使能

```
DMA_Cmd(DMA1_Channel1, DISABLE);
```

现在已经可以实现从A到B的一次转运工作了，如果想让B的值持续跟随A变化，需要进行以下步骤。

4. 写DMA传输函数，重新传入传输计数值

```
void MyDMA_Transfer(void)
{
    DMA_Cmd(DMA1_Channel1, DISABLE);
    DMA_SetCurrDataCounter(DMA1_Channel1, MyDMA_Size);
    DMA_Cmd(DMA1_Channel1, ENABLE);
    //至此DMA满足工作的三个条件
    while (DMA_GetFlagStatus(DMA1_FLAG_TC1) == RESET);  //等待转运完成
    DMA_ClearFlag(DMA1_FLAG_TC1);  //清除完成标志
```

```
}
```

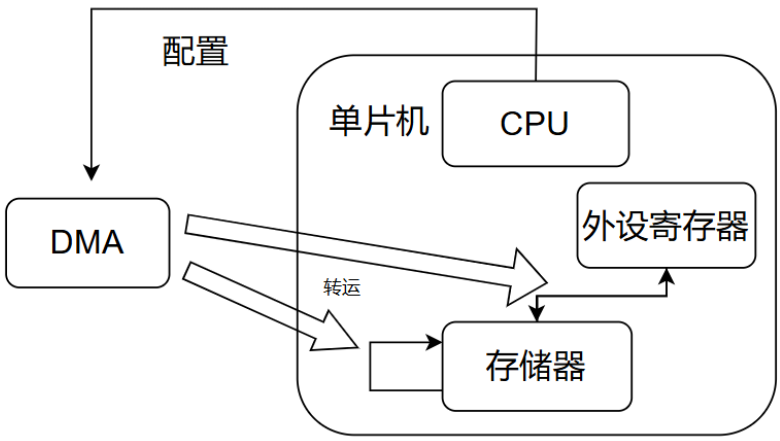
- 重写传输计数器时要确保DMA开关为关闭状态，即DMA失能

5. 使用DMA进行转运：先初始化（带参数，两个地址，传输计数器值），随后再在需要的地方使用MyDMA_Transfer()即可，用一次转运一次。

```
//举例
MyDMA_Init((uint32_t)DataA, (uint32_t)DataB, 4);
...
void MyDMA_Transfer();
...
```

6. 实际应用：
   数据规模要大才能体现出DMA的优势。
   在MPU6050姿态角解算的情景下，要用到数据转运的可以有：
   - MPU6050读取的原始数据AX等转运到待转化的变量AX_Convert中
   - 姿态角pitc，roll，yaw转运到舵机角度变量Angle中
     …
     以上都是从存储器到存储器的转运过程

框图



注：实际上DMA属于单片机的一部分，此处为了表达方便

照片

略（本节现象和MPU6050输出姿态角，驱动舵机转动没有任何区别）