# 嵌入式系统原理及实验

顾　震

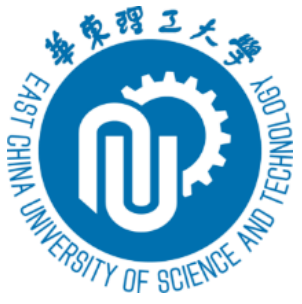
信息科学与工程学院自动化系

华东理工大学

Email: guzhen@ecust.edu.cn

# 课程大纲

1. 嵌入式系统导论
2. Cortex-M3微处理器
3. STM32最小系统及开发环境
4. 嵌入式C语言
5. 通用输入输出GPIO模块
6. 中断
7. 定时器原理与应用
8. USART通信原理及实现
9. DMA控制器
10. SPI与I2C通信原理及实现
11. 模数转换原理及实现
12. 人工智能辅助的嵌入式项目开发
13. 嵌入式应用前沿




# 9. DMA控制器

# 本章知识与能力要求

- 理解和掌握DMA的基本概念、适用场合；
- 理解STM32的DMA结构及工作原理；
- 掌握基于STM32CubeMX进行DMA开发的方法。

# 9. DMA控制器

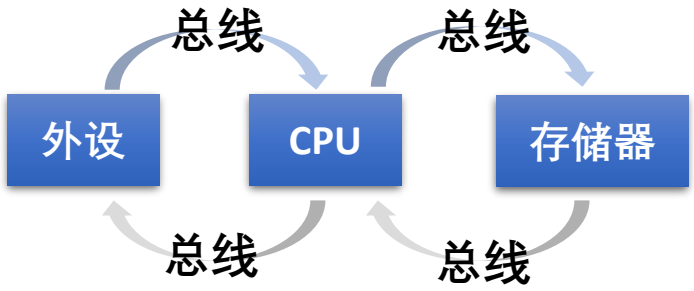
9.1 DMA基础理论知识

9.2 STM32的DMA模块

9.3 9.3 DMA模块的HAL库接口函数及应用

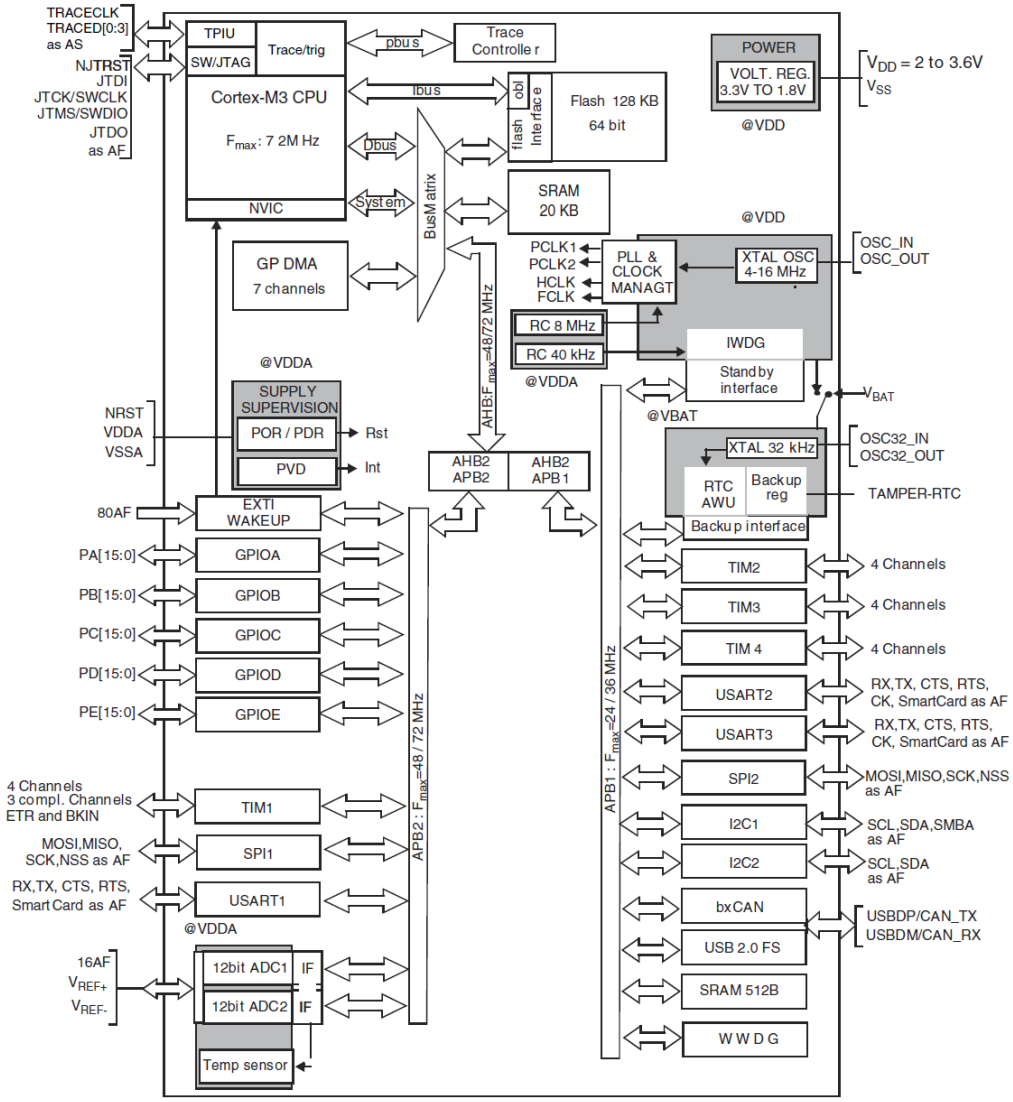
# 9.1 DMA基础理论知识

嵌入式芯片中的数据处理过程




在外设与存储器传输数据过程中，CPU全程参与

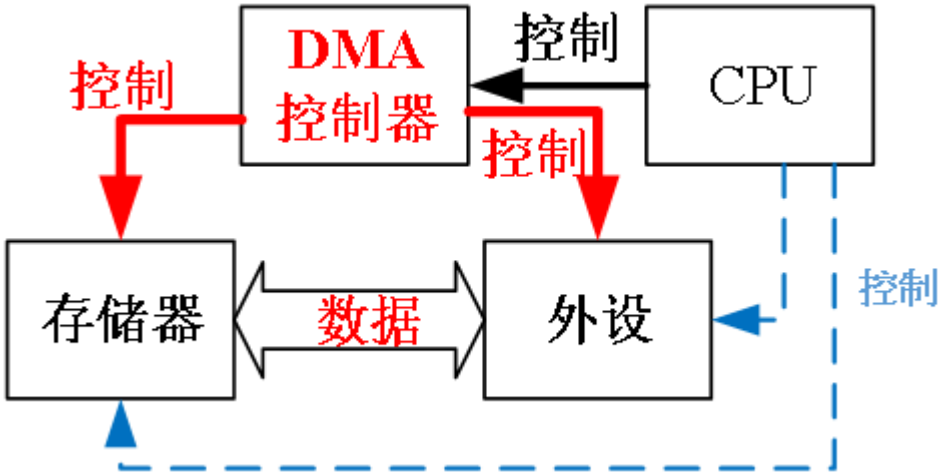
导致CPU大量的时间和资源被浪费，造成CPU效率低下。



# 9.1 DMA基础理论知识

- **DMA （Direct Memory Access）直接内存访问**

**定义**：DMA（Direct Memory Access，直接存储器存取）是一种允许外设设备直接与存储器进行数据交换的技术，**无需CPU的全程介入。**




**运行机制**：通过专用**DMA控制器（DMAC）**接管总线控制权，实现外设与存储器、存储器不同区域之间的高速数据搬运。

# 9.1 DMA基础理论知识

## 为什么使用DMA?

传统的数据传输方式（如查询和中断方式）需要CPU全程参与每个数据的搬运，导致在处理大量数据时**CPU效率低下**。

**优势**:

**解放CPU**：数据传输过程中CPU可以执行其他任务，实现并行工作

**提高传输效率**：特别适用于高速、大批量数据传输场景

**降低系统功耗**：传输过程中CPU可以进入低功耗模式

# 9.1 DMA基础理论知识

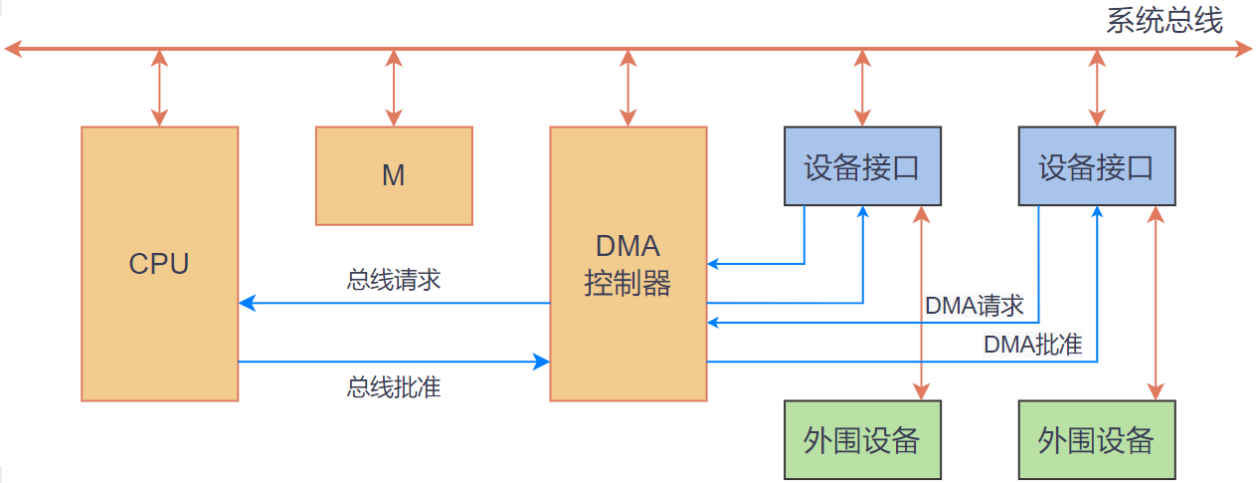
## DMA工作流程

### 1. DMA请求

外设准备好数据后，向DMAC发出DMA请求信号。

### 3. DMA传输

DMAC接管总线，发送地址和控制信号，直接完成外设与存储器间的数据传输。




### 2. DMA响应

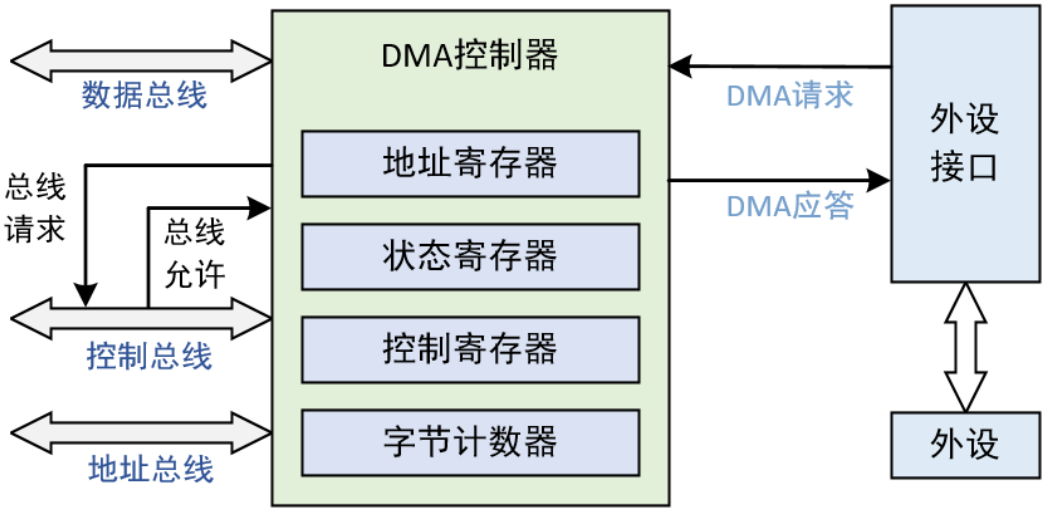
DMAC收到请求后，向CPU发出总线请求信号。CPU完成当前总线周期后，发出响应信号，并放弃总线控制权。

### 4. DMA结束

预定数据传送完毕后，DMAC撤销HOLD信号，CPU重新控制总线。

# 9.1 DMA基础理论知识

- **DMA控制器（DMAC）**专用硬件电路，负责管理DMA传输过程




**地址寄存器**：存储并自动更新数据传输的源地址和目标地址。

**控制寄存器**：配置DMA传输的所有参数和行为模式。

**状态寄存器**：实时反映DMA控制器和传输通道的当前状态。

**字节计数器**：记录和监控DMA传输数据总量，达到预设值时发出完成信号。

# 9.1 DMA基础理论知识

## DMA适用场景

**存储设备I/O**：硬盘、软盘与内存间的大量数据交换

**高速通信通道**：网络接口、光纤通信等高速数据流处理

**数据采集系统**：波形采集、传感器数据记录等密集突发数据传输

**图像处理**：屏幕扫描操作、图像缓冲区传输

**多处理机系统**：多处理机间数据块传送

# 9.2 STM32的DMA模块

9.2.1 DMA内部结构

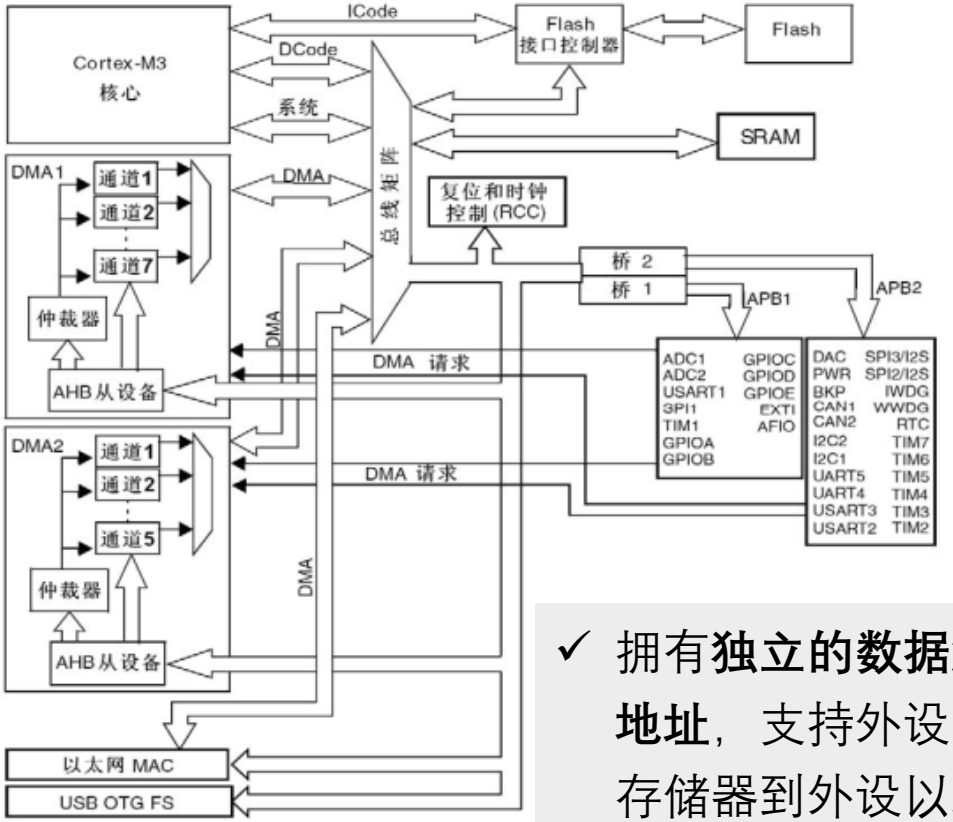
9.2.2 DMA优先权

9.2.3 DMA中断请求

# 9.2.1 DMA内部结构

- STM32F103系列包含**两个DMA控制器（DMA1和DMA2）**，共**12个独立的传输通道**（DMA1有7个，DMA2有5个）。

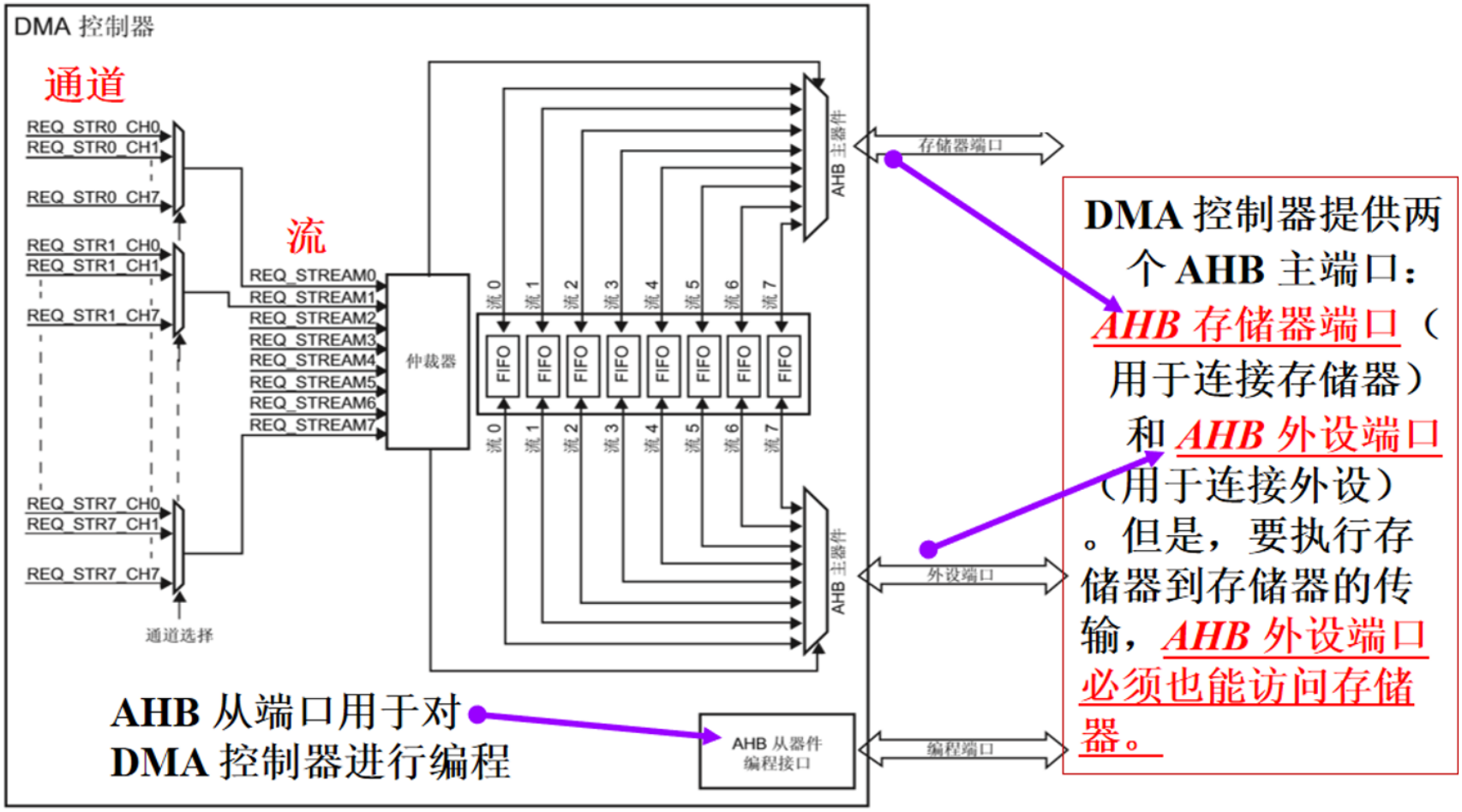
✓ 每个通道都可以配置为专门管理来自一个或多个外设的存储器访问请求




✓ 支持**循环缓冲器管理**，减轻程序员负担

✓ 拥有**独立的数据源和目标地址**，支持外设到存储器、存储器到外设以及存储器到存储器的传输。

# 9.2.1 DMA内部结构




•当多个DMA传输请求同时发生时，**仲裁器**负责根据预设的**优先级规则**，决定哪一个请求可以优先使用总线进行数据传输。

# 9.2.1 DMA内部结构

**关键配置参数**

- **传输方向**：外设到内存、内存到外设、内存到内存、外设到外设
- **数据宽度**：支持8位、16位和32位数据
- **地址递增**：可配置源地址和目的地址是否自动递增
- **传输模式**：普通模式、循环模式
- **中断能力**：传输完成、半传输、传输错误

# 9.2.1 DMA内部结构

## 传输模式

### 1. 单次模式

传输结束后(即要传输数据的数量达到零)，将不再产生DMA操作。若开始新的DMA传输，需在关闭DMA通道情况下，重新启动DMA传输。

### 2. 循环模式

可用于处理环形缓冲区和连续数据流(例如ADC扫描模式)。当活循环模式后，每轮传输结束时，要传输的数据数量将自动用设置的初始值进行加载，并继续响应DMA请求。

# 9.2.1 DMA内部结构

## DMA1各个通道的DMA请求

外设（TIMx（x=1,2,3,4）、ADC1、SPI1、IICx（x=1,2）、USARTx（x=1,2,3））产生的DMA1请求，传送到DMA1控制器，同一时刻只能有一个请求有效。

| 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/I2S_RX | SPI/I2S_TX | | |
| USART | | USART3_TX | USART3_RX | USART1_TX | USART1_RX | USART2_RX | USART2_TX |
| IIC | | | | IIC2_TX | IIC2_RX | IIC1_TX | IIC1_RX |
| TIM1 | | TIM1_CH1 | TIM1_CH2 | TIM1_TX4<br>TIM1_TRIG<br>TIM1_COM | TIM1_UP | TIM1_CH3 | |
| TIM2 | TIM2_CH3 | TIM2_UP | | | TIM2_CH1 | | TIM2_CH2<br>TIM2_CH4 |
| TIM3 | | TIM3_CH3 | TIM3_CH4<br>TIM3_UP | | | TIM3_CH1<br>TIM3_TRIG | |
| TIM4 | TIM4_CH1 | | | TIM4_CH2 | TIM4_CH3 | | TIM4_UP |

# 9.2.1 DMA内部结构

## DMA2各个通道的DMA请求

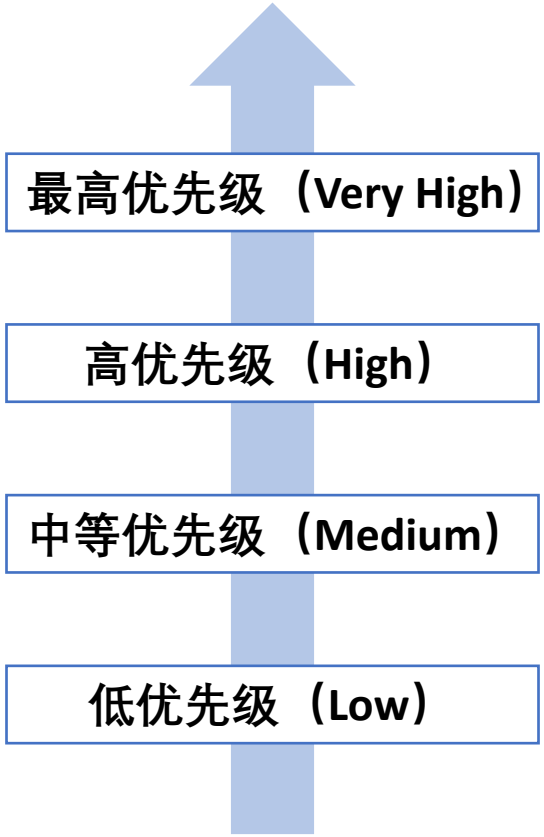
外设（TIMx（x=5,6,7,8）、ADC3、SPI/I2S3、USART4、DAC通道1、2和SDIO）产生5个通道的DMA2请求传送到DMA2控制器，同一时刻只能有一个请求有效

| 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 | | | |
| USART4 | | | USART4_RX | | USART4_TX |
| SDIO | | | | SDIO | |
| TIM5 | TIM5_CH4<br>TIM5_TRIG | TIM5_CH3<br>TIM5_UP | | TIM5_CH2 | TIM5_CH1 |
| TIM6/<br>DAC通道1 | | | TIM6_UP/<br>DMA通道1 | | |
| TIM7/<br>DAC通道2 | | | | TIM7_UP/<br>DMA通道2 | |
| TIM8 | TIM8_CH3<br>TIM8_UP | TIM8_CH4<br>TIM8_TRIG<br>TIM8_COM | TIM8_CH1 | | TIM8_CH2 |

# 9.2.2 DMA优先权

当有多个DMA请求时，DMA控制器通过内部的仲裁器进行优先权管理。

通道的优先权分为4个等级，高优先级的通道优先获得总线响应。




如果2个请求有相同的软件优先级，则较低编号的通道比较高编号的通道有较高的优先权。例如，通道2优先于通道4。

在大容量产品和互联型产品中，DMA1控制器拥有高于DMA2控制器的优先级。

# 9.2.3 DMA中断请求

- DMA的每个通道都可以在DMA传输过程中触发中断，可通过设置相应寄存器的不同位来打开这些中断。

**DMA中断事件主要有：**

- ✓ **HT（Half Transfer，传输一半）**
- ✓ **TC（Transfer Complete，传输完成）**
- ✓ **TE（Transfer Error，传输错误）**

**分别对应三个中断标志：HTIF、TCIF、TEIF，每个中断标志都有允许控制位。**

# 9.3 DMA模块的HAL库接口函数及应用

9.3.1 DMA的HAL库接口函数

9.3.2 HAL库DMA应用实例

# 9.3.1 DMA的HAL库接口函数

STM32的HAL库常用DMA接口函数

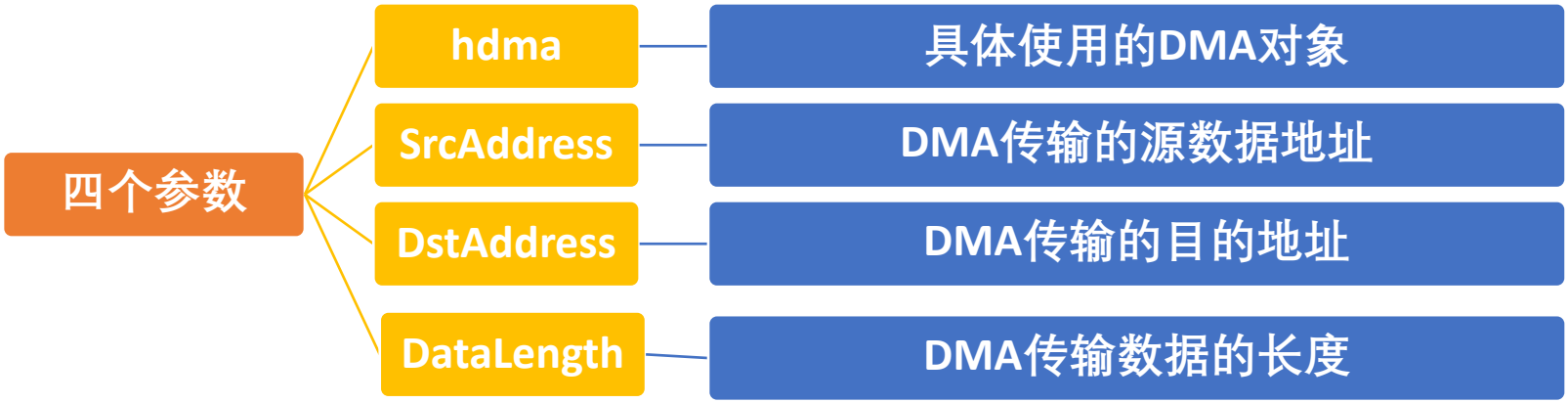
DMA模块的HAL库相关的接口函数定义在stm32f1xx_hal_dma.c源文件中，在stm32f1xx_hal_dma.h头文件中可以查看相关函数的声明以及结构体定义。

| 类型 | 函数及功能描述 |
| --- | --- |
| 初始化及复位函数 | HAL_DMA_Init(DMA_HandleTypeDef *hdma);<br>功能描述：DMA初始化函数 |
| | HAL_DMA_DeInit (DMA_HandleTypeDef *hdma);<br>功能描述：DMA复位函数 |
| 引脚功能操作函数 | HAL_DMA_Start (DMA_HandleTypeDef *hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength);<br>功能描述：启动DMA传输 |
| | HAL_DMA_Start_IT(DMA_HandleTypeDef *hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength);<br>功能描述：DMA中断开始 |
| | void HAL_DMA_IRQHandler(DMA_HandleTypeDef *hdma);<br>功能描述：DMA中断处理函数 |
| 外设状态函数 | HAL_DMA_GetState(DMA_HandleTypeDef *hdma);<br>功能描述：获取DMA状态函数 |
| | HAL_DMA_GetError(DMA_HandleTypeDef *hdma);<br>功能描述：获取DMA错误函数 |

# 9.3.1 DMA的HAL库接口函数

## DMA启动传输函数HAL_DMA_Start()

```
HAL_DMA_Start(DMA_HandleTypeDef *hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength);
```




**使用范例：**

```
HAL_DMA_Start(huart->hdmatx, (u32)pData, (uint32_t)&huart->Instance->DR, 100);
    //开启DMA的串口传输
```

# 9.3.1 DMA的HAL库接口函数

## DMA串口发送数据传输函数AL_UART_Transmit_DMA()

```
HAL_StatusTypeDef
HAL_UART_Transmit_DMA(UART_HandleTypeDef *huart,
uint8_t *pData, uint16_t Size)
{
 uint32_t *tmp;
 if (huart->gState == HAL_UART_STATE_READY)
 {
  if ((pData == NULL) || (Size == 0U))
  {
   return HAL_ERROR;
  }
   /*huart配置，上锁 */
  __HAL_LOCK(huart);
  huart->pTxBuffPtr = pData;
  huart->TxXferSize = Size;
  huart->TxXferCount = Size;
  huart->ErrorCode = HAL_UART_ERROR_NONE;
  huart->gState = HAL_UART_STATE_BUSY_TX;
   /*设置UART的DMA传输完成时调用的回调函数*/
  huart->hdmatx->XferCpltCallback = UART_DMATransmitCplt;
```

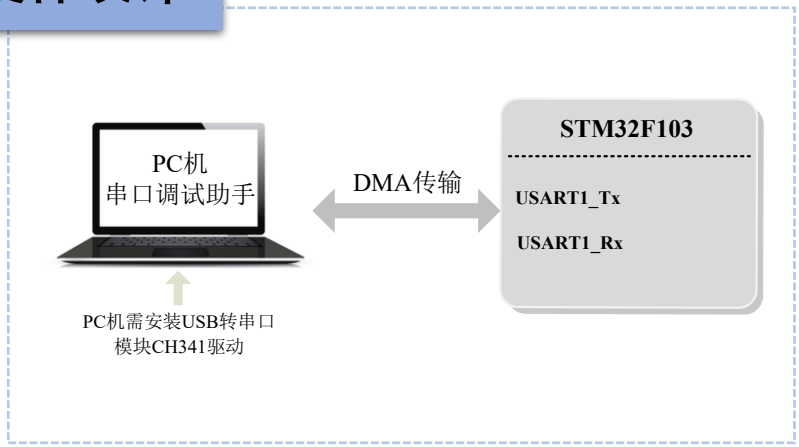
```
   /*设置UART的DMA传输完成一半时的回调函数 */
  huart->hdmatx->XferHalfCpltCallback =
UART_DMATxHalfCplt;
   /*设置DMA传输错误的回调函数 */
  huart->hdmatx->XferErrorCallback = UART_DMAError;
   /*置DMA传输中止的回调函数*/
  huart->hdmatx->XferAbortCallback = NULL;
   /*设置存储器到外设的地址，开启中断，并使能UART
的DMA传输的DMA通道*/
  tmp = (uint32_t *)&pData;
  HAL_DMA_Start_IT(huart->hdmatx, *(uint32_t *)tmp,
(uint32_t)&huart->Instance->DR, Size);
   /*清除串口TC中断标志位 */
  __HAL_UART_CLEAR_FLAG(huart, UART_FLAG_TC);
   /* huart解锁 */
  __HAL_UNLOCK(huart);
   /* 通过配置UART的CR3寄存器，使能DMA发送*/
  SET_BIT(huart->Instance->CR3, USART_CR3_DMAT);
  return HAL_OK;
 }
 else
 {   return HAL_BUSY;   }
}
```

# 9.3.2 HAL库DMA应用实例

## 功能

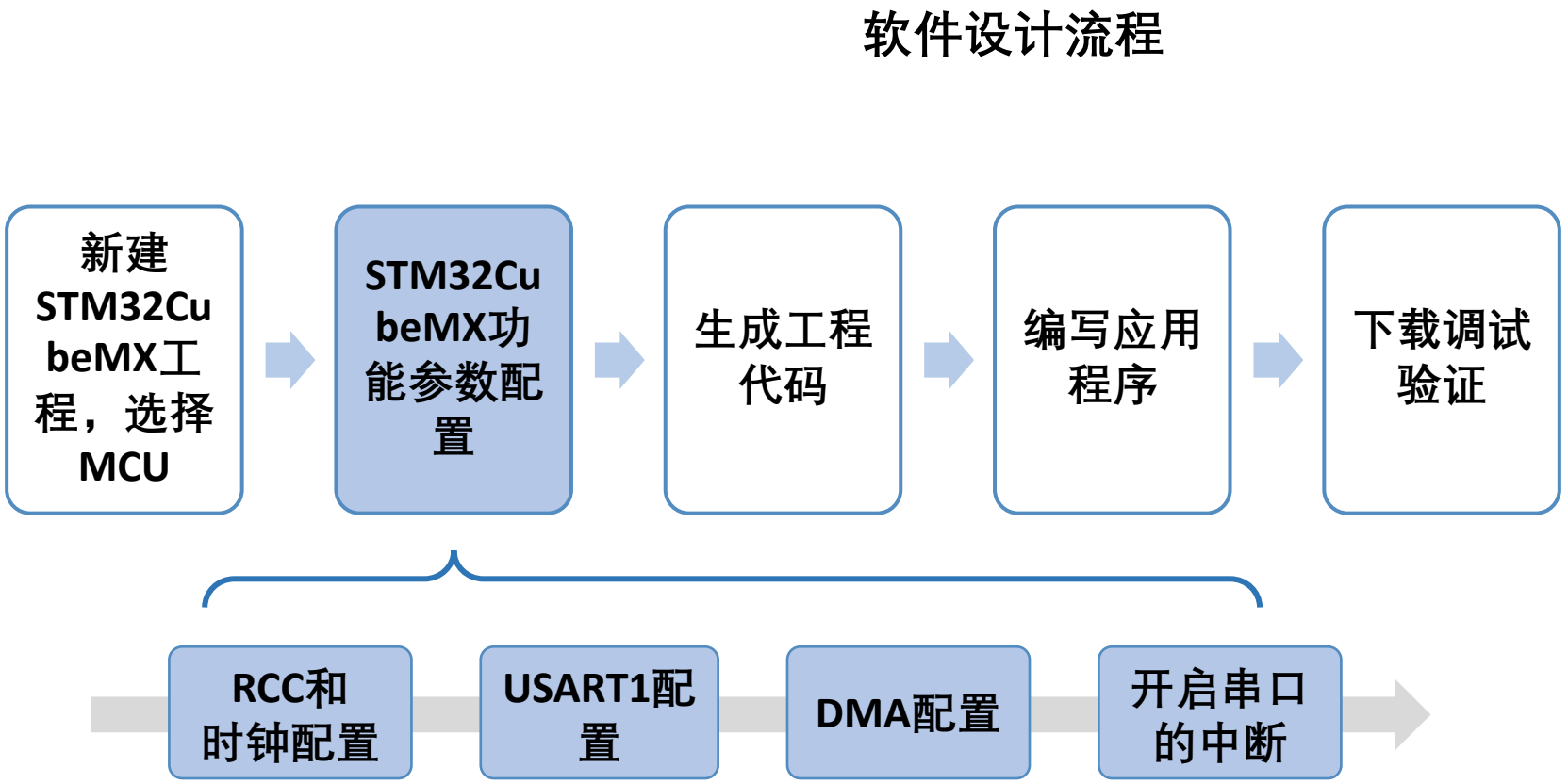
基于HAL库采用DMA方式实现USART串口收发数据，用于需要大批量数据的通信与信息交换的场合。

## 硬件设计




通过STM32F103目标板上的串口1（USART1）采用DMA方式实现与上位机PC的数据传输，可在PC机的串口助手查看测试结果。

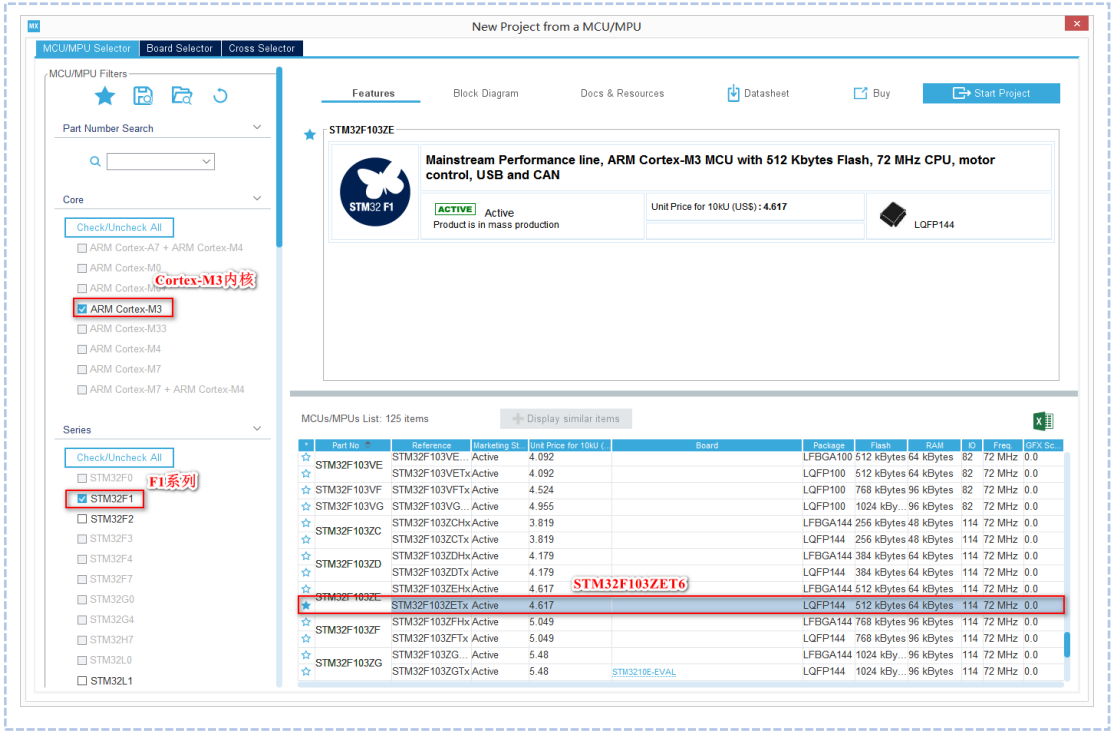
# 9.3.2 HAL库DMA应用实例



# 9.3.2 HAL库DMA应用实例

## 软件设计——新建STM32CubeMX工程，选择MCU

新建一个STM32CubeMx工程，选择选择MCU，这里选择STM32F103ZETx芯片，读者可根据自己的目标板选择相应的芯片

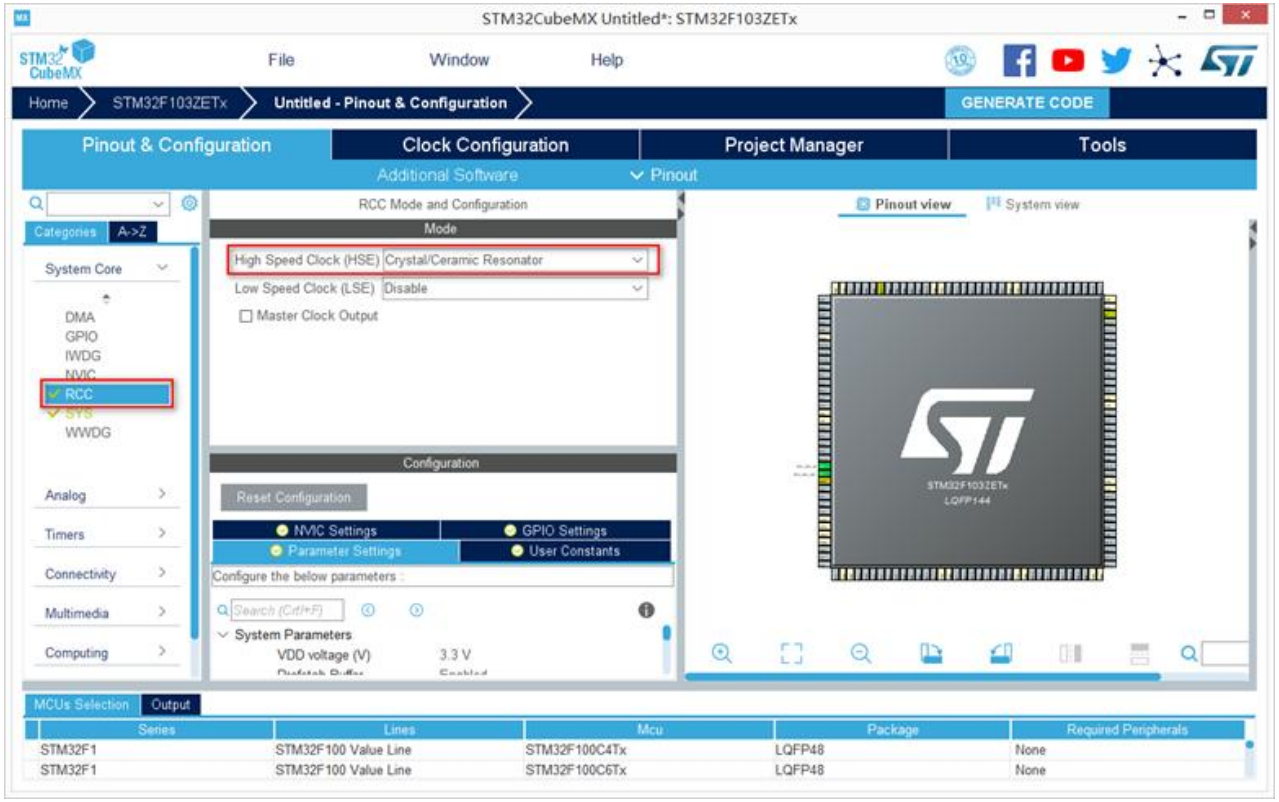


# 9.3.2 HAL库DMA应用实例

## 软件设计——STM32CubeMX功能参数配置

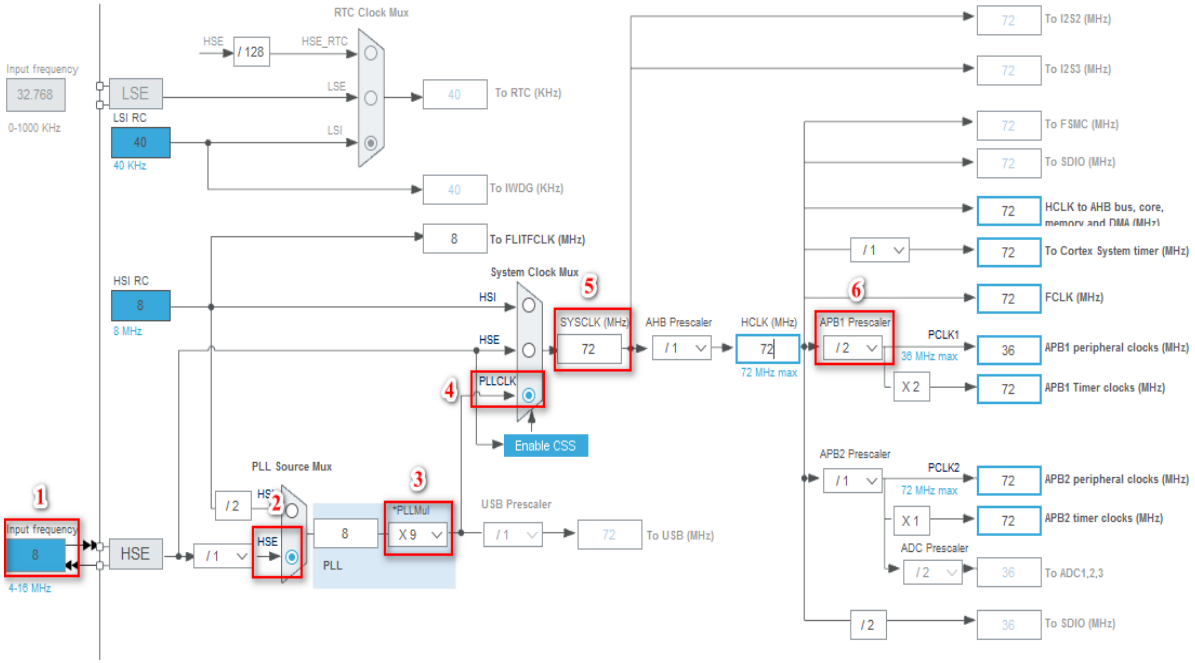
RCC配置

HSE选择“Crystal/Ceramic Resonator”（晶振/陶瓷谐振器



# 9.3.2 HAL库DMA应用实例

## 软件设计——STM32CubeMX功能参数配置

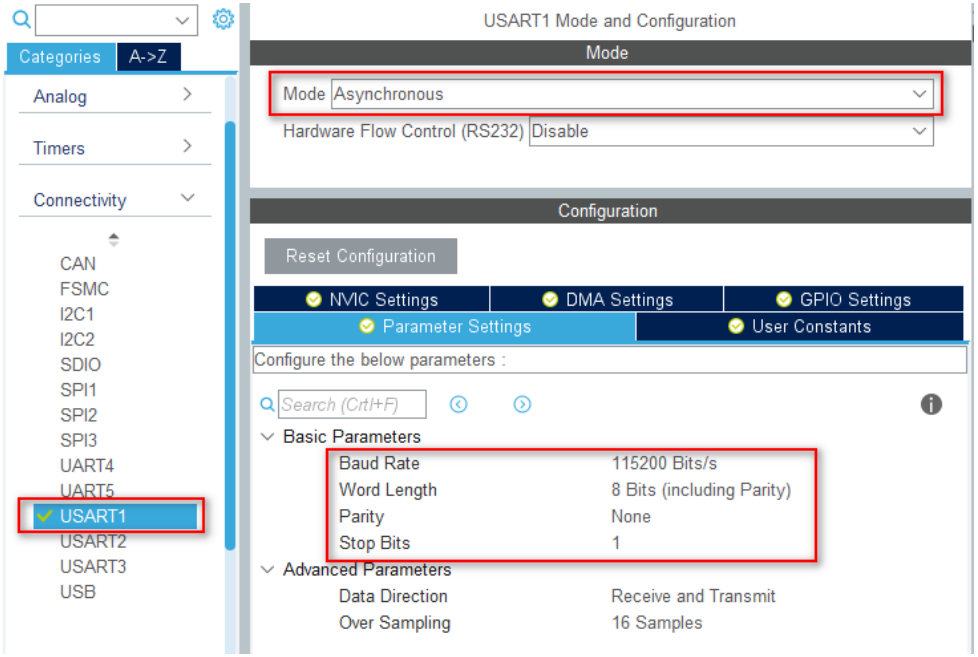



### 时钟配置

通过图形化方式直观地对系统时钟进行配置，系统时钟采用外部高速时钟，配置STM32F103系列芯片最大时钟为72MHz，配置APB2为72MHz，配置APB1为36MHz。

# 9.3.2 HAL库DMA应用实例

## 软件设计——STM32CubeMX功能参数配置

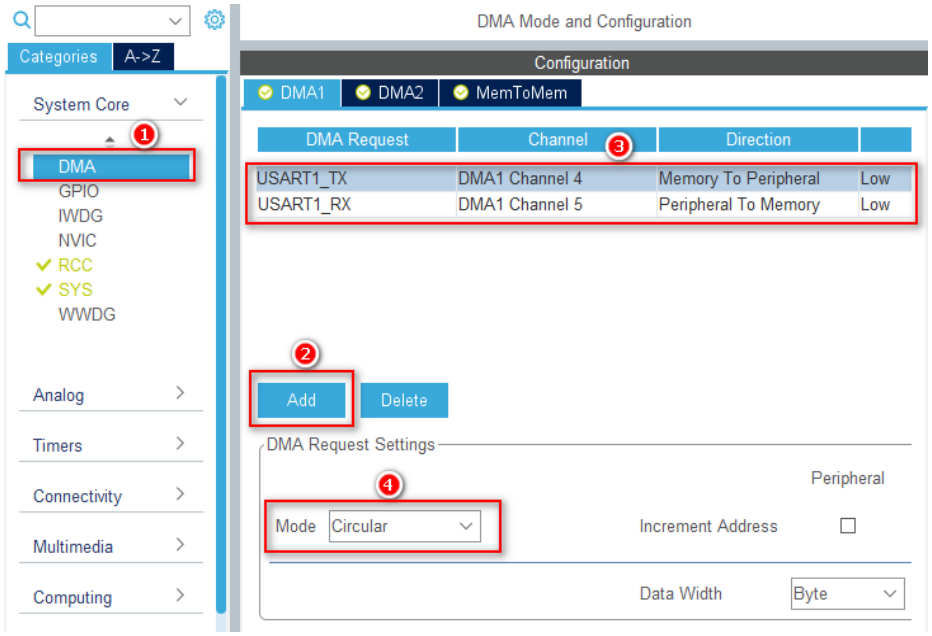



### 串口USART1配置

- 在“Connectivity”中选择“USART1”；
- 设置“MODE”为“Asynchronous”；
- 设置USART1的相关参数配置为默认的：115200、8、None和1

# 9.3.2 HAL库DMA应用实例

## 软件设计——STM32CubeMX功能参数配置

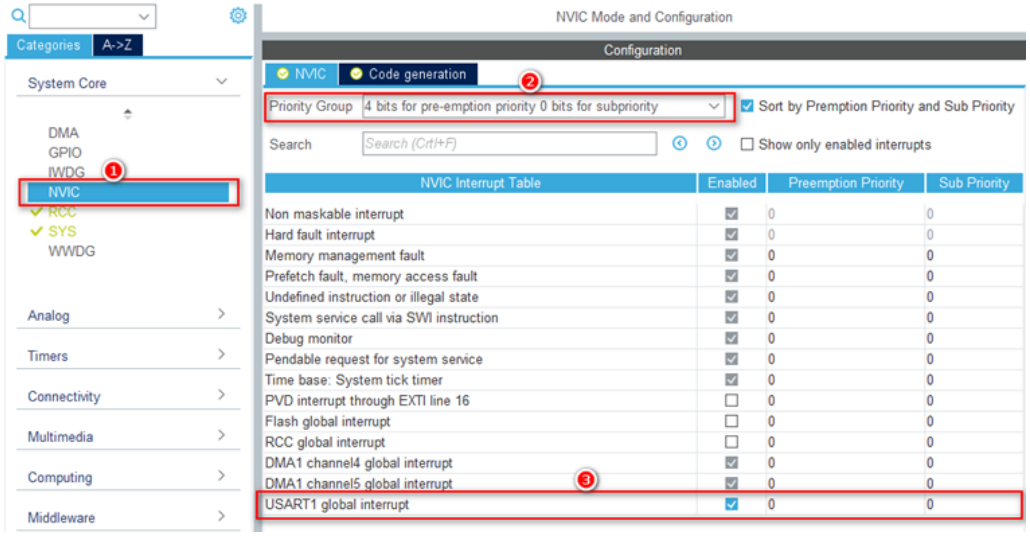



### DMA配置

- 在“System Core”中单击“DMA”
- 在右侧的配置页面“Configuration”中，单击“Add”，添加“USART1_TX”；
- “Mode”选择为“Circular”；
- “Channel”和“Direction”会自动选择；
- “Data Width”用默认的“Byte”；
- 按同样的操作方式添加“USART1_RX”

# 9.3.2 HAL库DMA应用实例

## 软件设计——STM32CubeMX功能参数配置

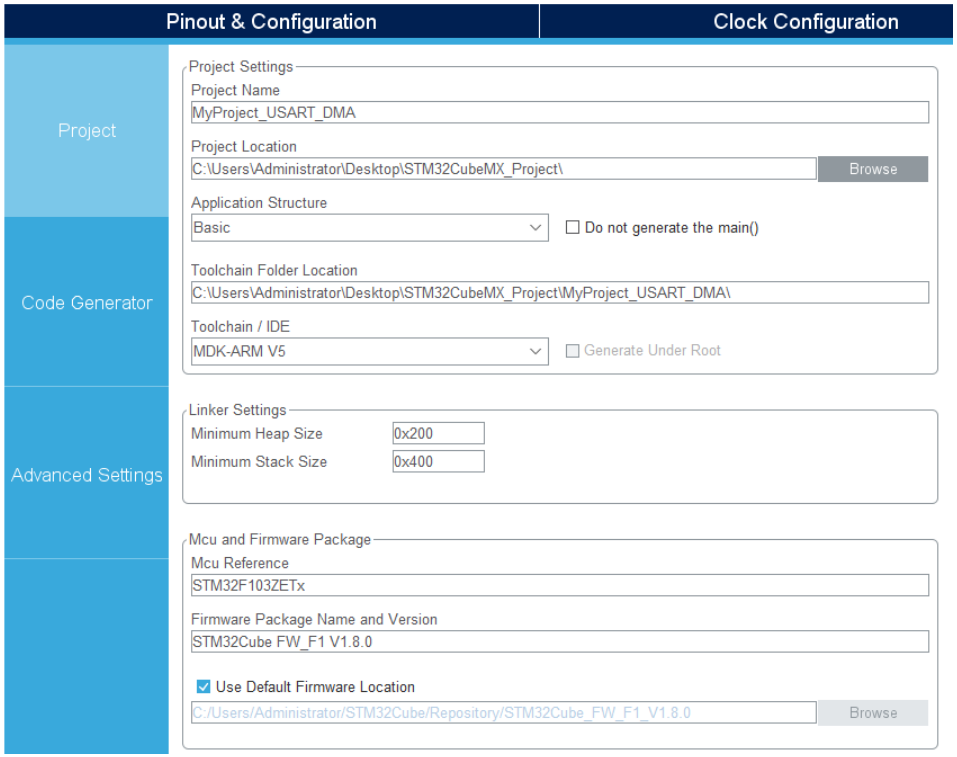



## 开启串口USART1的中断

- 在“System Core”中点击“NVIC”，在右侧的配置页面“Configuration”中，可以设置“Priority Group”优先级分组，默认为“4 bits for pre-emption priority 0 bits for subpriority”；
- 勾选“USART1 global interrupt”，“Priority Group”优先级分组可根据具体需要选择合适的分组。

# 9.3.2 HAL库DMA应用实例

## 软件设计——生成工程代码

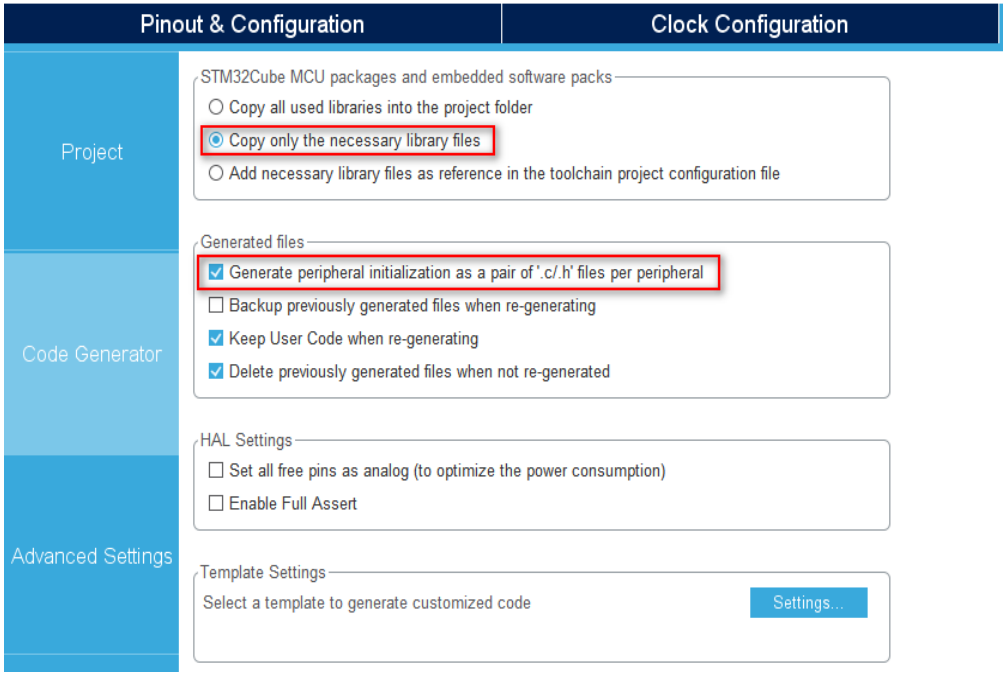



**配置keil工程名称和存放位置**

**配置工程名称、工程保存位置、选择“MDK-ARM V5”编译器等**

# 9.3.2 HAL库DMA应用实例

## 软件设计——生成工程代码

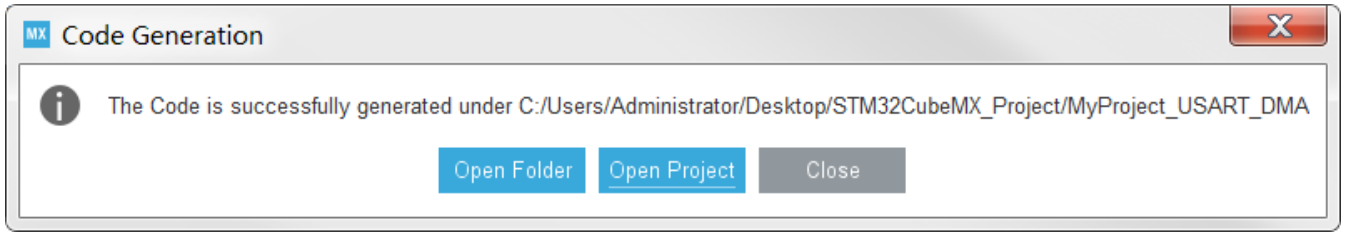



在“Code Generator”选项栏中找到“Generated files”框，勾选“Generate peripheral initialization as a pair of ’.c/.h’ files per IP”，将外设初始化的代码设置生成为独立的.c源文件和.h头文件。

# 9.3.2 HAL库DMA应用实例

## 软件设计——生成工程代码

通过STM32CubeMX的菜单栏中的“Generate Code”生成工程代码，生成代码后，会提示是否打开该工程窗口




# 9.3.2 HAL库DMA应用实例

## 软件设计——编写应用程序

在main.c文件中添加代码：

- 先定义两个数组DMA_RxBuffer和DMA_TxBuffer，用于接收数据和发送数据
- 将此代码放在main.c文件中的/* USER CODE BEGIN PV */和/* USER CODE END PV */之间。

```
/* Private variables ---------------------*/
/* USER CODE BEGIN PV */
/* Private variables ---------------------*/
uint8_t DMA_RxBuffer[10];
uint8_t DMA_TxBuffer[] = " This is the sending code with DMA  \r\n ";
/* USER CODE END PV */
```

# 9.3.2 HAL库DMA应用实例

## 软件设计——编写应用程序

在int main(void)函数中的/* USER CODE BEGIN 2 */和 /* USER CODE END 2 */之间添加代码：

```
/* USER CODE BEGIN 2 */
HAL_UART_Receive_DMA(&huart1,DMA_RxBuffer,10);// 开启DMA接收数据
HAL_UART_Transmit_DMA(&huart1,(uint8_t*)DMA_TxBuffer,sizeof(DMA_TxBuffer));
/* USER CODE END 2 */
```

# 9.3.2 HAL库DMA应用实例

## 软件设计——编写应用程序

在/* USER CODE BEGIN 4 */和/* USER CODE END 4 */之间添加如下代码：

```
    /* USER CODE BEGIN 4 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
  HAL_UART_Transmit_DMA(&huart1,DMA_RxBuffer,10);
  HAL_UART_Receive_DMA(&huart1,DMA_RxBuffer,10);  //重新打开DMA接收数据
}
    /* USER CODE END 4 */
```

# 9.3.2 HAL库DMA应用实例

**实验：**

4.3 尝试修改数据实现通过发送2个字符，来对LED的亮暗进行控制，并写出相应的回调函数代码。

5.3 尝试将上位机通过串口发送给开发板的所有数据都发送回上位机，并且上位机一次发送的数据大于100个byte（可发送任意字符）。

# 本章小结

9.1 DMA基础理论知识

9.2 STM32的DMA模块

DMA内部结构

DMA优先权

DMA中断请求

9.3 DMA模块的HAL库接口函数及应用

接口函数

应用实例