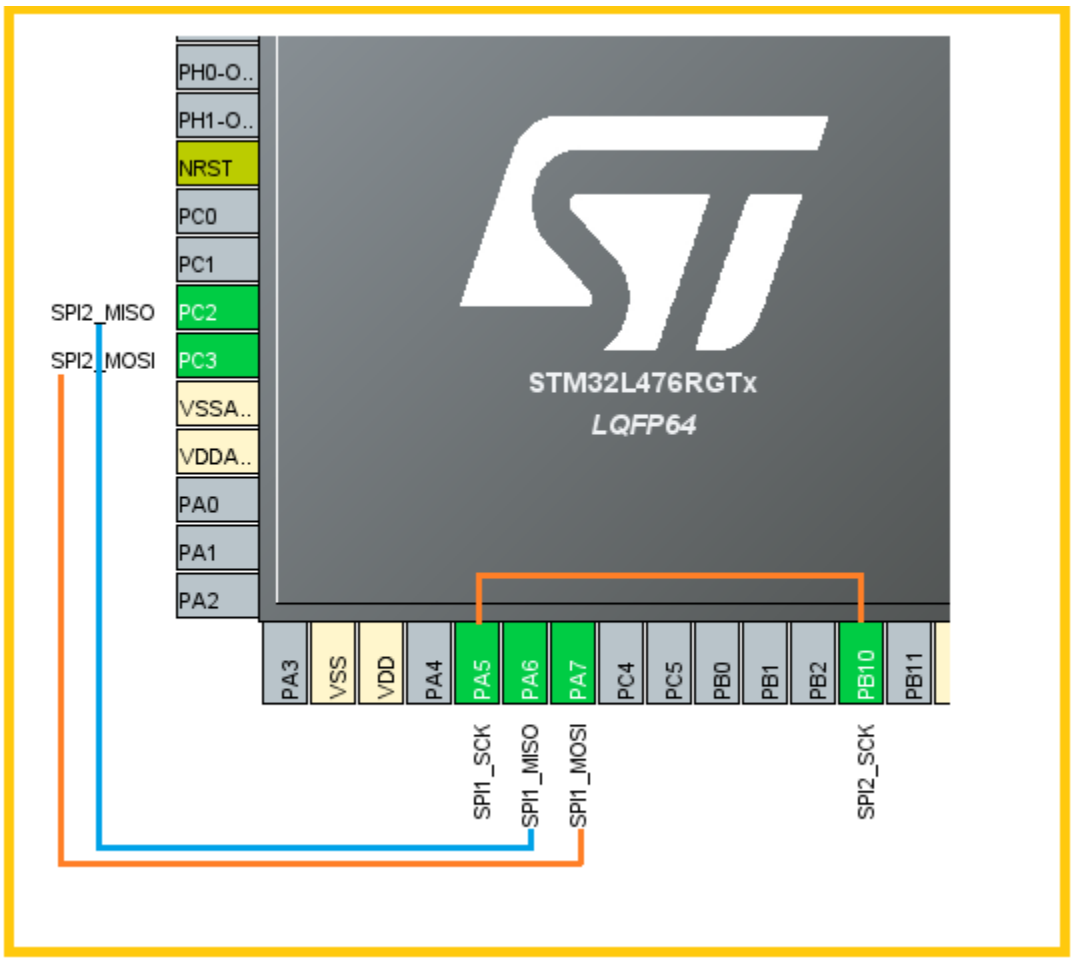
一、STM32 定时器触发 SPI 逐字收发之应用示例

我们在做 SPI 应用时，有时希望通过定时器来定时地触发 SPI 的收发，并利用 DMA 完成数据的传输。这里，我基于 STM32L476 芯片来做个演示，以供参考【为什么选用 32L476，其实没啥特别原因，只是顺便找了块 Nucleo 板】。

本示例的大致过程是这样的：

片内 SPI1 做 Master，SPI2 做 Slave,均工作在全双工模式。

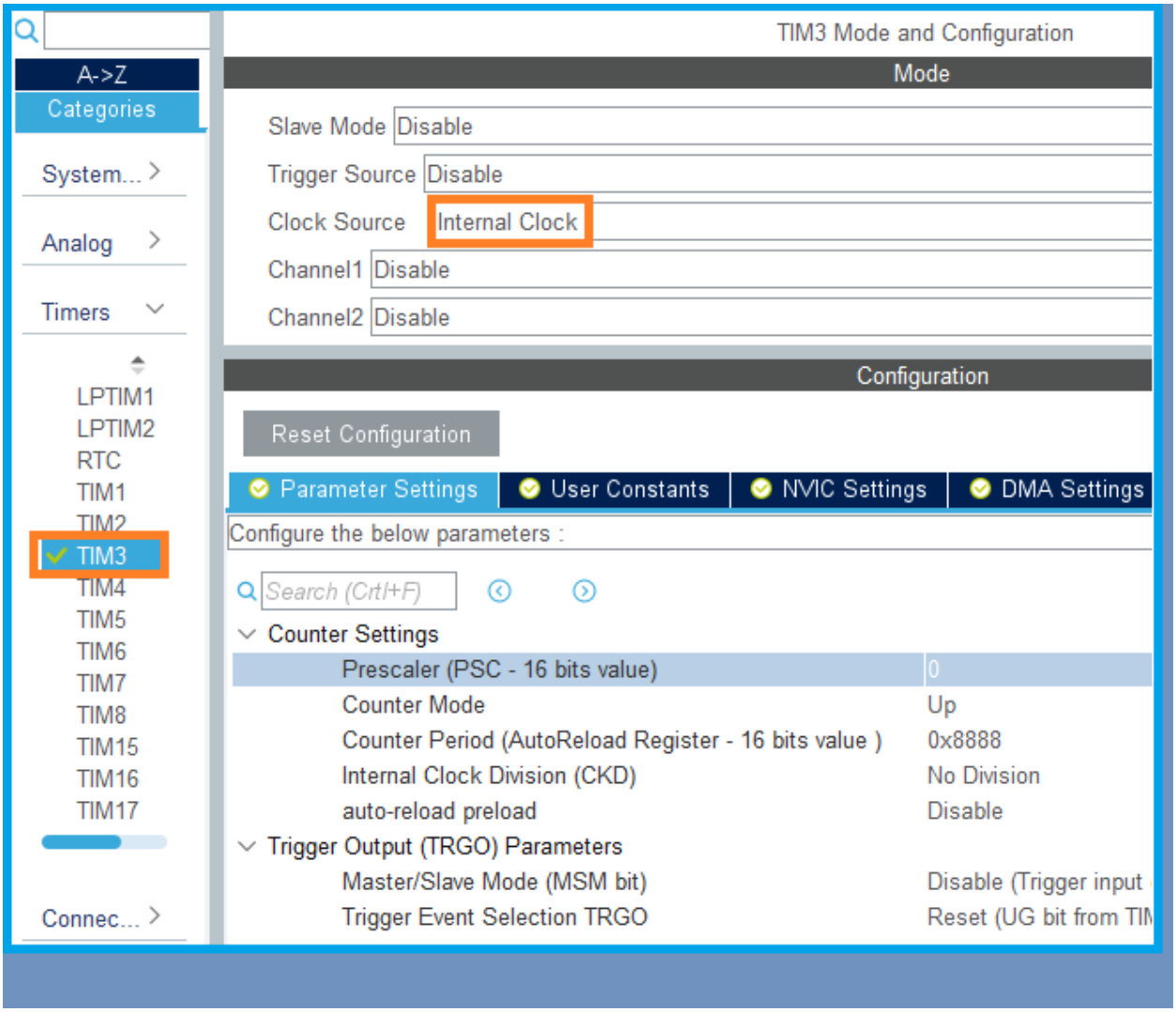


这里使用片内定时器 TIM3，通过它的更新事件触发 DMA 请求，通过 DMA 将数据给到 SPI1 的数据寄存器并发送出去，同时也开启 SPI1 接收事件的 DMA 传输。总之，SPI2 的收、发事件都启用 DMA 完成。

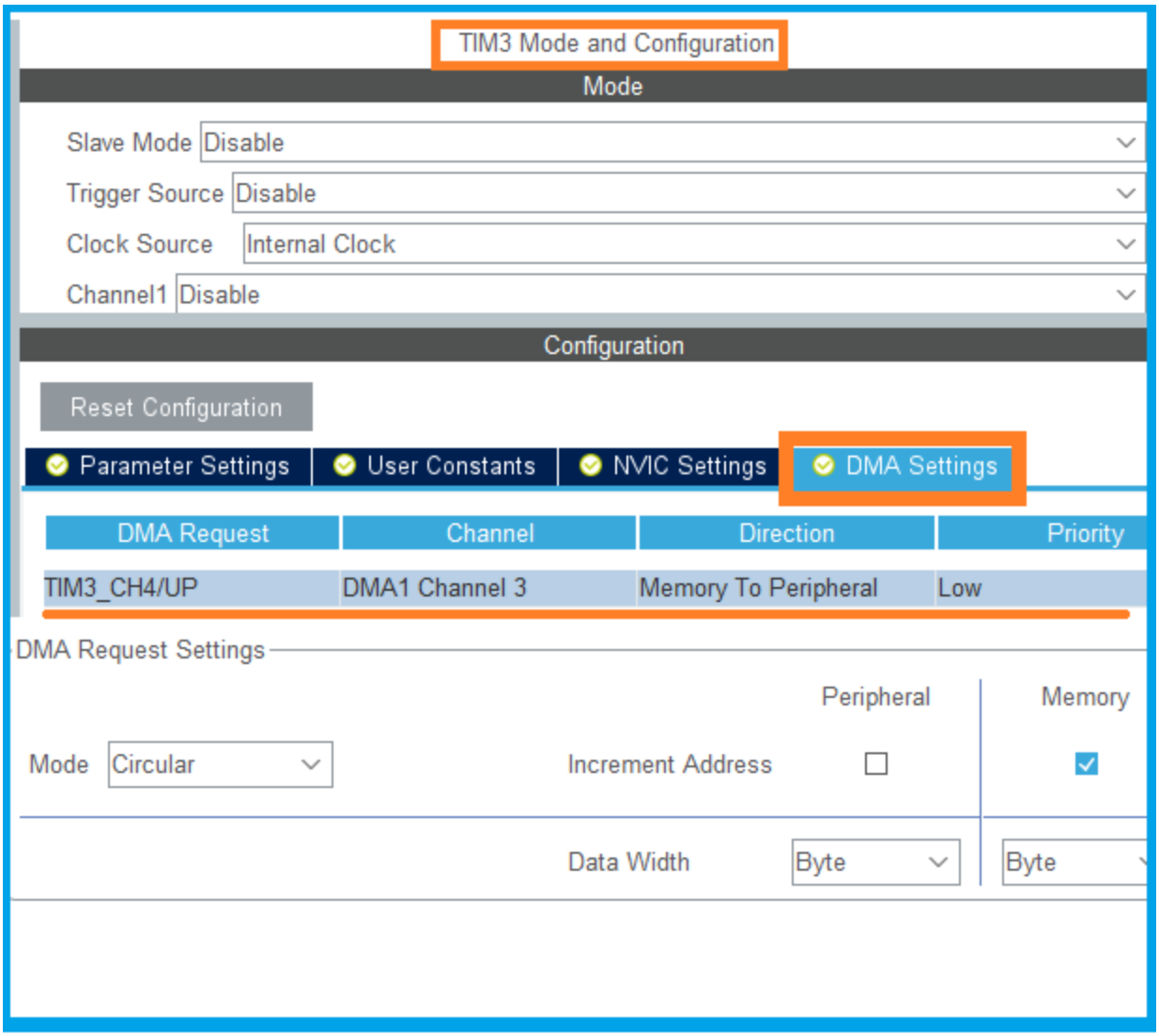
TIM3 的更新事件周期控制两个 SPI 的收发节奏，即定时器每产生一次更新事件，SPI1/SPI2 这两个主从通信模块就进行一个数据的收发。我们可以通过调整定时器的计时周期来调整数据收发的快慢。

好，先基于 STM32CubeMx 进行初始化配置。

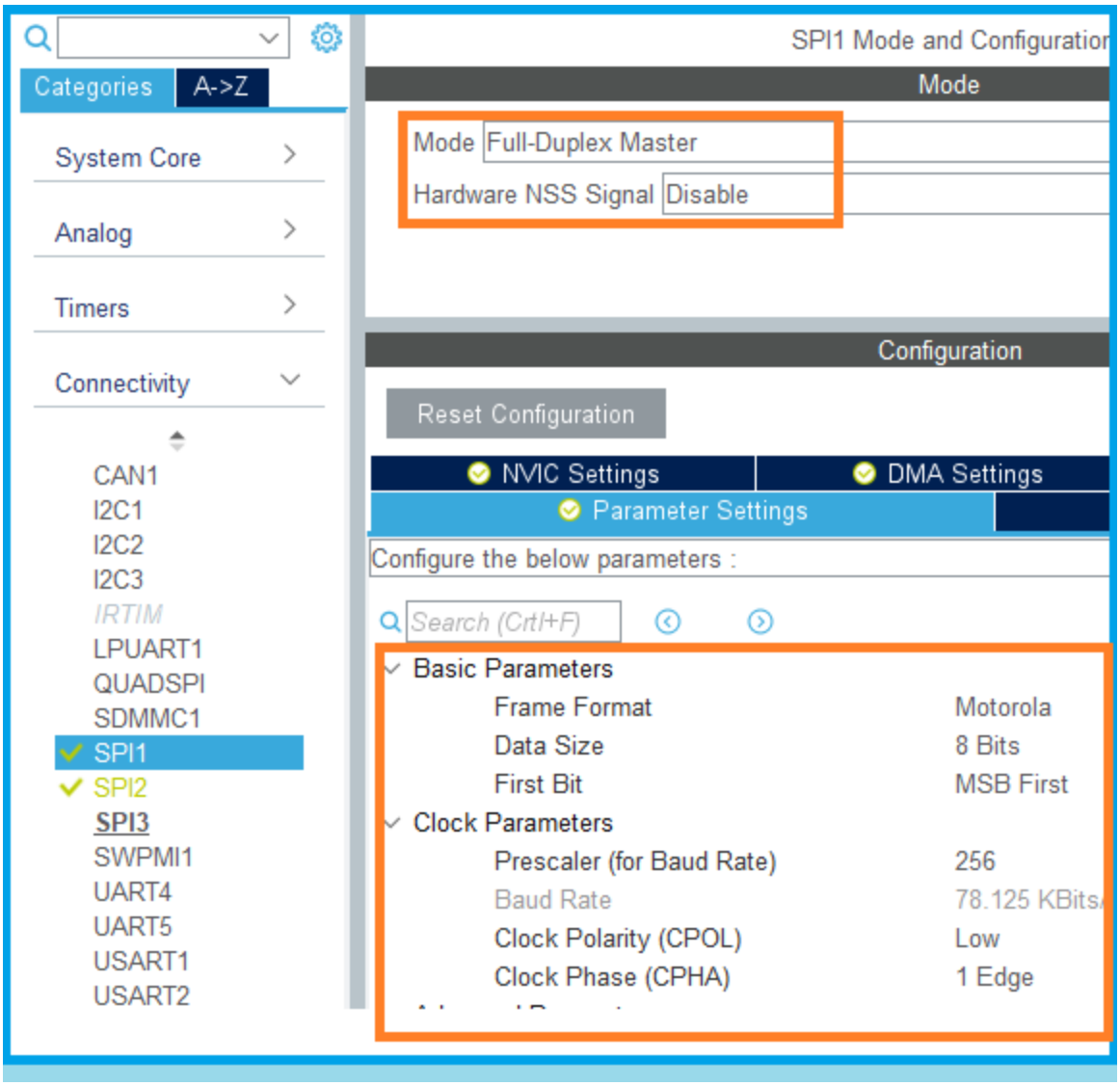
1、对 TIM3 做基本配置。选择时钟源，先大致估算个定时器计时周期，调试时我们可以自行灵活调整。

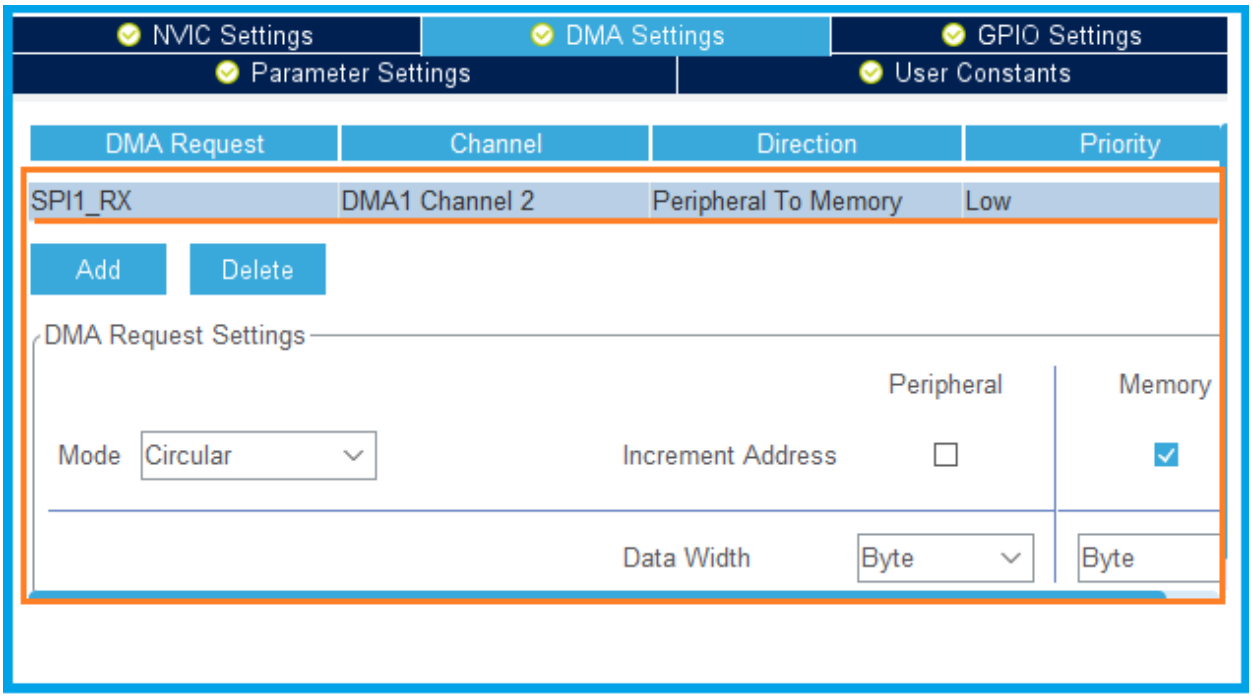


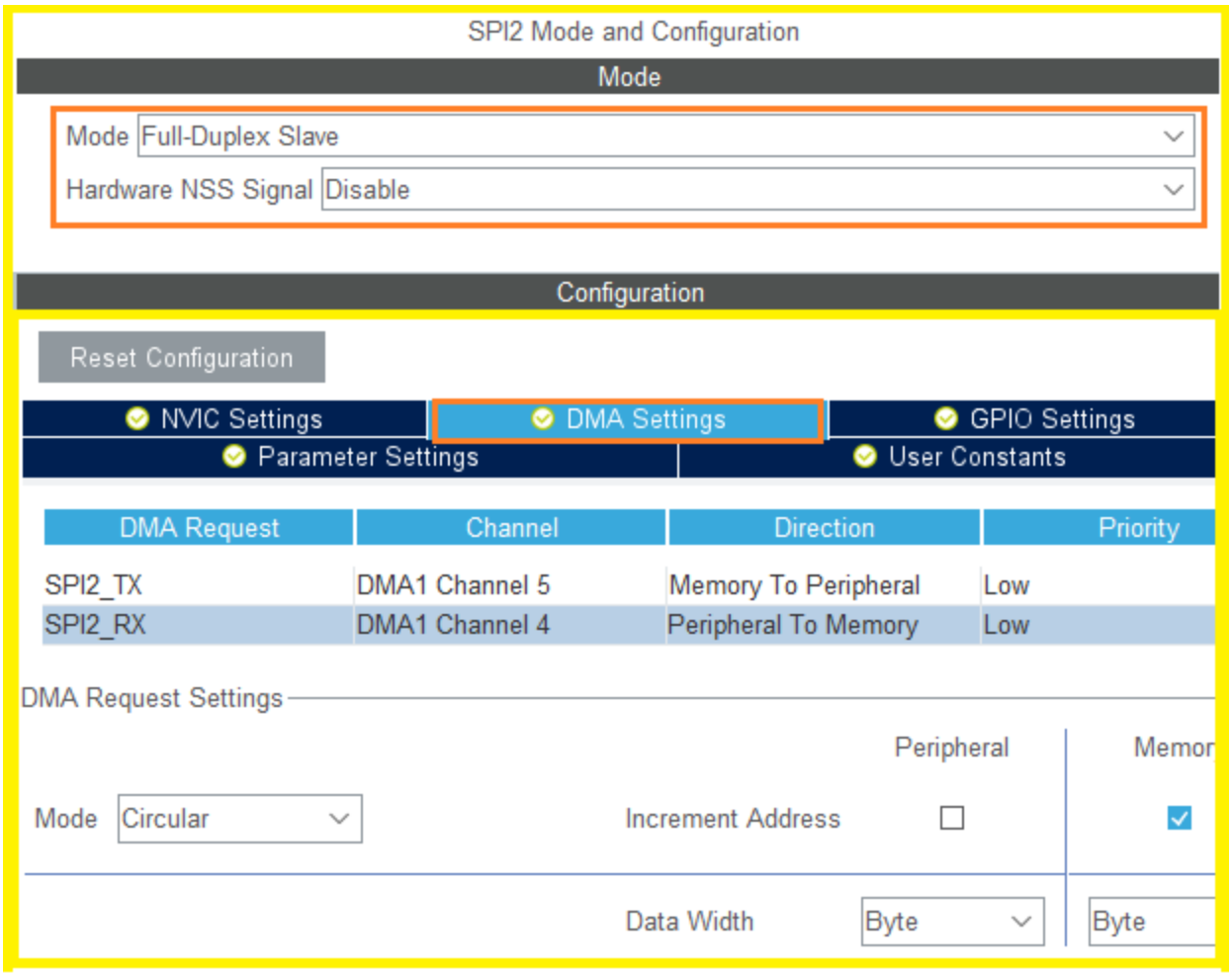
开启基于 TIM3 更新事件的 DMA 配置。传输方向是从 Memory 到 外设 SPI1，即将内存数据传输到 SPI1 的数据寄存器进行数据发送，这里选用循环模式，以便测试。



2、对 SPI1/SPI2 进行基本配置。细节请参看下面三幅截图。

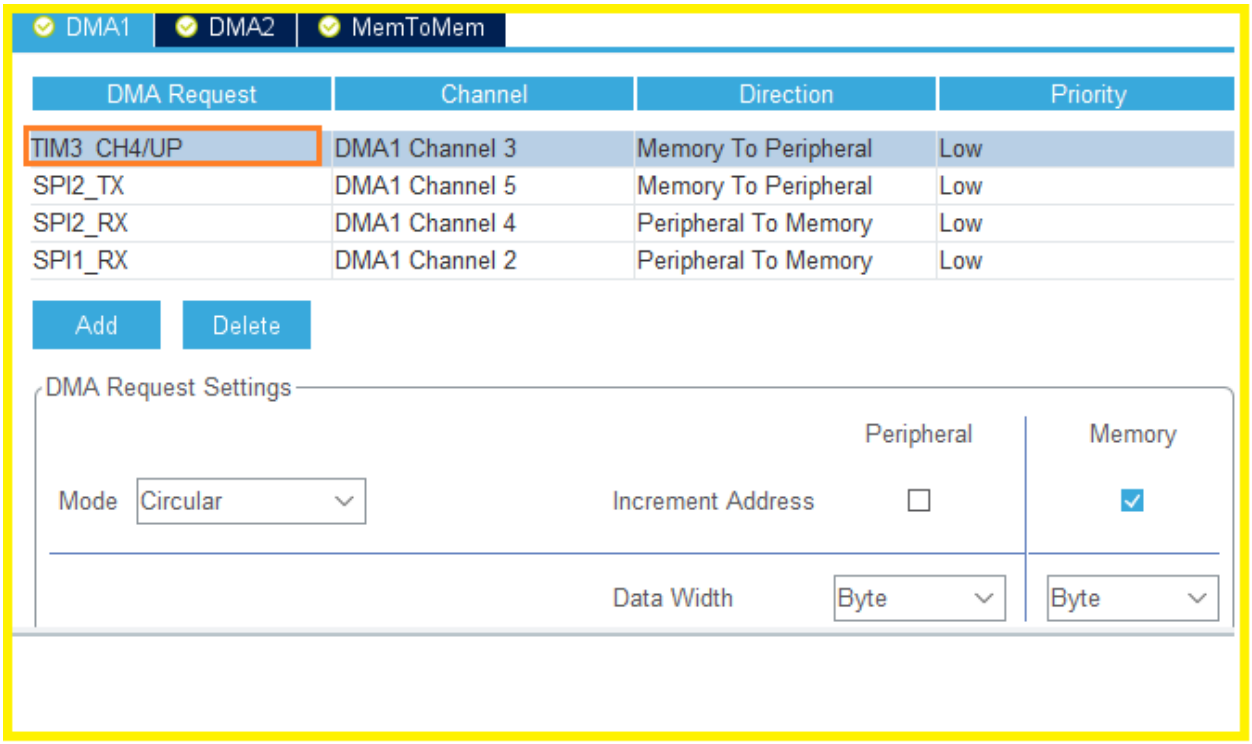

Categories
A->Z
System Core
Analog
Timers
Connectivity
CAN1
I2C1
I2C2
I2C3
IRTIM
LPUART1
QUADSPI
SDMMC1
SPI1
SPI2
SPI3
SWPMI1
UART4
UART5
USART1
USART2
SPI1 Mode and Configuration
Mode
Mode Full-Duplex Master
Hardware NSS Signal Disable
Configuration
Reset Configuration
NVIC Settings
DMA Settings
Parameter Settings
Configure the below parameters :
Search (Crtl+F)
Basic Parameters
Frame Format Motorola
Data Size 8 Bits
First Bit MSB First
Clock Parameters
Prescaler (for Baud Rate) 256
Baud Rate 78.125 KBits
Clock Polarity (CPOL) Low
Clock Phase (CPHA) 1 Edge


NVIC Settings
DMA Settings
GPIO Settings
Parameter Settings
User Constants
DMA Request
Channel
Direction
Priority
SPI1_RX
DMA1 Channel 2
Peripheral To Memory
Low
Add
Delete
DMA Request Settings
Peripheral
Memory
Mode
Circular
Increment Address
Data Width
Byte
Byte



3、DMA 的配置情况。

在 TIM3 和 SPI1/SPI2 外设配置中，开启了相关事件的 DMA 请求，汇总如下图。



4、准备用户代码。

当完成基于 STM32CubeMx 的初始化配置并生产初始化代码后，我们准备相应的用户代码。这里准备了 4 个内存数组，分别用于存放 SPI1/SPI2 的收发数据。

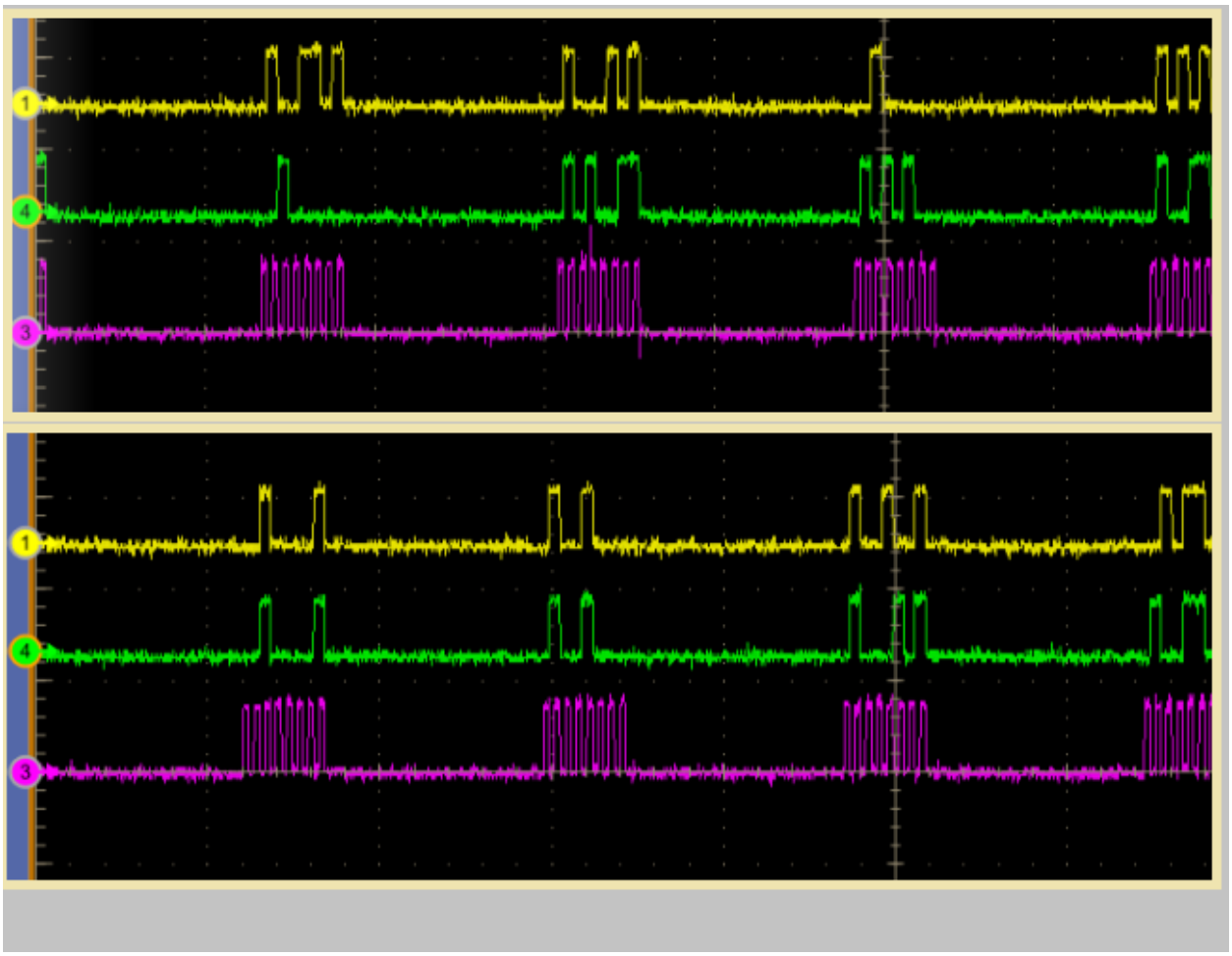
在定时器的触发下，Master SPI1 逐字的向 Slave SPI2 发送“Hello! I AM STM32!”,Slave SPI2 也逐字的向 Master 回应“HI,MASTER,ME TOO!”,这样循环操作。下面两幅截图是本示例中使用到的用户代码，是基于 STM32Cube 固件库而编写的。应该说简单明了，无须过多解释。

```
#define Length (17)

uint8_t  Master_DATA_TX[]={"HELLO!I AM STM32!"};

uint8_t  Slave_DATA_TX[]={"HI,MASTER,ME TOO!"};

uint8_t  Master_DATA_RX[Length]={0};

uint8_t  Slave_DATA_RX[Length]={0};
```

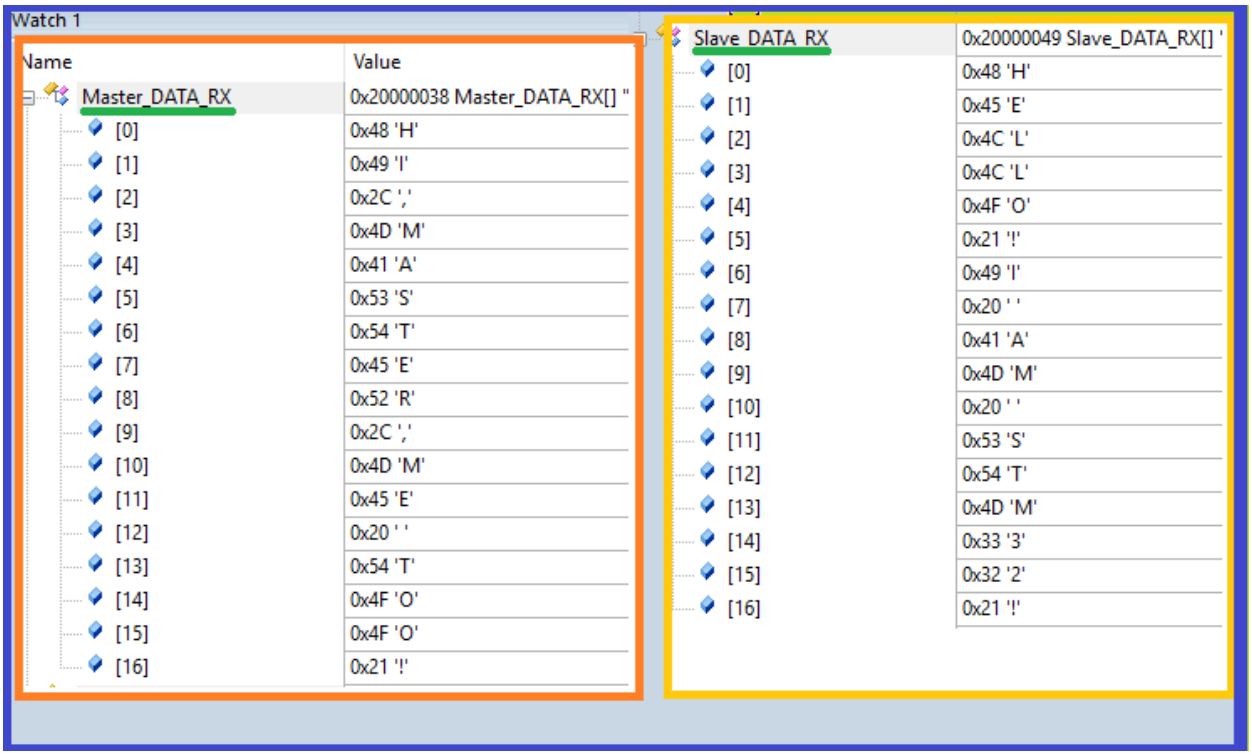
```
 TIM3->CNT = 0;
__HAL_TIM_CLEAR_IT(&htim3, TIM_IT_UPDATE);   A

HAL_DMA_Start(&hdma_spi1_rx, (uint32_t)&SPI1->DR, (uint32_t)Master_DATA_RX, Length);

HAL_DMA_Start(&hdma_spi2_rx, (uint32_t)&SPI2->DR,(uint32_t)Slave_DATA_RX,  Length);
                                                                                  B
HAL_DMA_Start(&hdma_spi2_tx, (uint32_t)Slave_DATA_TX,(uint32_t)&SPI2->DR,  Length);

HAL_DMA_Start(&hdma_tim3_ch4_up, (uint32_t)Master_DATA_TX,(uint32_t)&SPI1->DR, Length)

SET_BIT(SPI2->CR2, SPI_CR2_RXDMAEN);

SET_BIT(SPI2->CR2, SPI_CR2_TXDMAEN);          C

SET_BIT(SPI1->CR2, SPI_CR2_RXDMAEN);

__HAL_TIM_ENABLE_DMA(&htim3, TIM_DMA_UPDATE);

__HAL_SPI_ENABLE(&hspi2);

__HAL_SPI_ENABLE(&hspi1);                     D

 __HAL_TIM_ENABLE(&htim3);
```

5、结果验证。

下面的截图是两个不同时刻 SPI 通信时的信号波形图。其中，紫色的是时钟信号，绿色、黄色是数据信号。两个数据信号间的时间间隔由定时器的更新周期决定。



下面的截图是在调试状态下的通过观察窗口得到的 SPI1/SPI2 分别从对方收到的数据，即 SPI2 收到的数据是“HELLO,I AM STM32!”,SPI1 收到的数据则是“HI,MASTER,ME TOO!”



整体上讲，上述应用的实现不难，可能稍微有点综合性。

要实现上述应用，首先要求我们对 DMA 传输的原理有清晰的了解，触发事件，传输源、传输目标几个概念及关系要弄清楚。

另外，即使我们基于 STM32 固件库开发，不一定能找到完整的现存例程，我们可能需要基于现有驱动代码自行组织用户程序。

还有，在上面示例代码中，我没有开启 DMA 的中断事件，我们在具体应用中可以根据情况来决定是否启用 DMA 中断，比方开启传输完成中断等。

最后顺便提醒下，这里我们基于定时器事件的 DMA 请求而自行指定 DMA 的源端和目的端，一定要保证是该触发事件所请求的 DMA 可以到达的地方。建议编程设计前最好查看下相关芯片数据手册里的芯片模块及总线框架图，不然的话，有时你可能遇到你指定的 DMA 根本就不正常运作的情况。