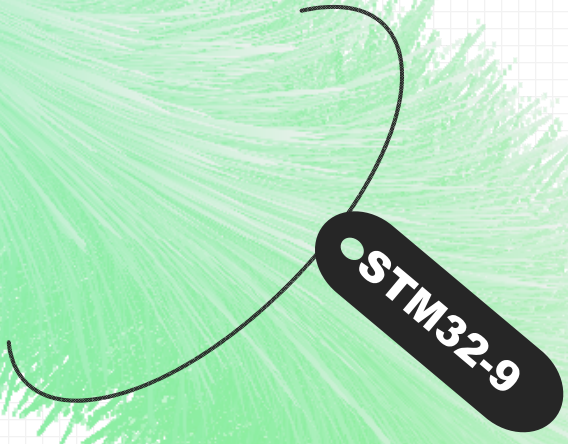
9. STM32的DMA

北京科技大学

计算机与通信工程学院

目录

CONTENTS.



直接内存访问（DMA）简介

- DMA(“Direct Memory Access”，直接内存访问) 是一种不经过CPU而直接从内存存取数据的数据交换模式。
- DMA的主要功能是可以把数据从一个地方搬到另外一个地方，而且不占用CPU。
- 在DMA模式下，CPU只须向DMA控制器下达指令，让DMA控制器来处理数据的传送，数据传送完毕再把信息反馈给CPU.
- 好处：
  - 减轻了CPU资源占有率，可以大大节省系统资源。
  - 使用 DMA 还可以保持 CPU 在低功耗模式下与外设单元之间传送数据，不需要唤醒，这就降低了整个系统的功耗。
- 例如：只需要 CPU 极少的干预，DMA 就可以将数据从诸如 ADC 或 RF 收发器等外设单元传送到存储器，从存储器传送数据到 USART，或定期在 ADC 和存储器之间传送数据样本，等等。

STM32的DMA概览

DMA1：有7个通道，可以实现 P->M，M->P，M->M （M：Memory， P:Peripherals)

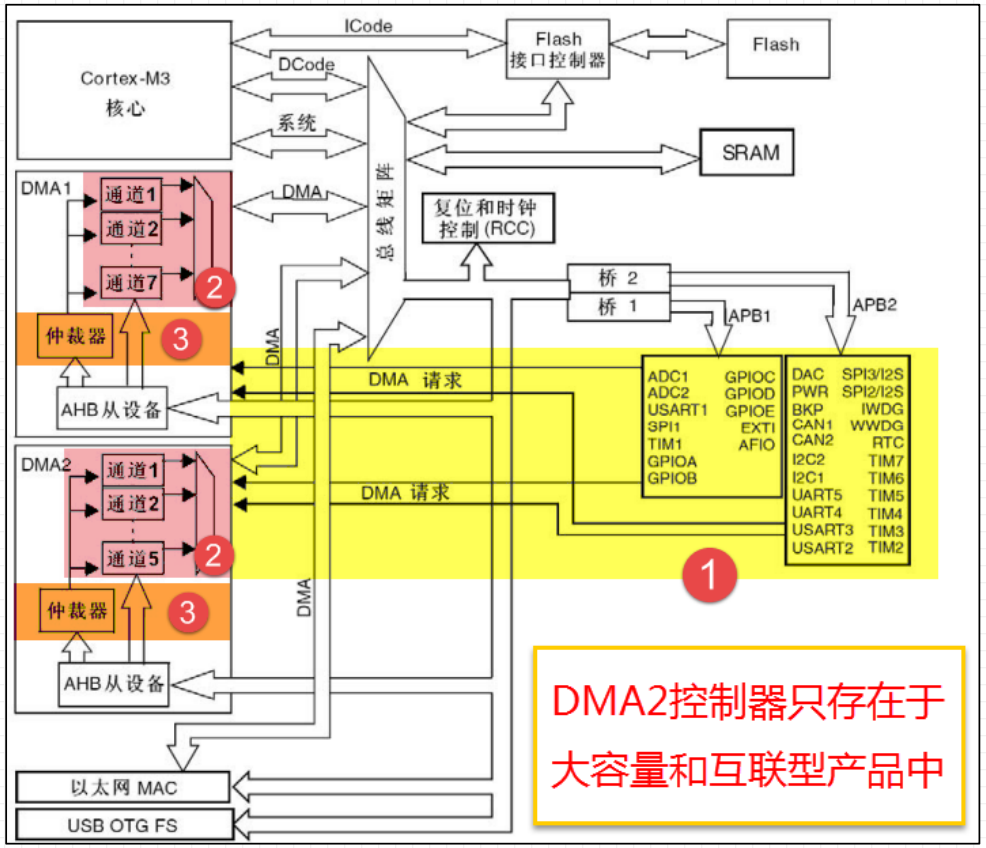
DMA2：有5个通道，可以实现 P->M，M->P，M->M（只有大容量/互联型产品才有DMA2)

DMA功能框图讲解

1-DMA请求

2-通道

3-仲裁器



如果外设要想通过DMA来传输数据，必须先给DMA控制器发送DMA请求，DMA收到请求信号之后，控制器会给外设一个应答信号，当外设应答后且DMA控制器收到应答信号之后，就会启动DMA的传输，直到传输完毕。

DMA请求+通道

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/I^2S		SPI1_RX	SPI1_TX	SPI/I2S2_RX	SPI/I2S2_TX		
USART		USART3_TX	USART3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
I^2C				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_CH1			TIM4_CH2	TIM4_CH3		TIM4_UP

注： M->M可以使用任意通道

DMA请求+通道

DMA2请求映射

外设	通道1	通道2	通道3	通道4	通道5
ADC3(1)					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

ADC3/SDIO/TIM8 的DMA请求只有大容量的单片机才有

仲裁器

多个DMA请求一起来，怎么办?

- 1、软件阶段，DMA_CCRx：PL[1:0]。

位13:12	**PL[1:0]**：通道优先级 (Channel priority level) 这些位由软件设置和清除。 00：低 01：中 10：高 11：最高

- 2、硬件阶段，通道编号小的优先级大，DMA1的优先级高于DMA2的优先级。

目录

CONTENTS.

STM32-9

DMA初始化结构体

初始化结构体在固件库头文件中：stm32f10x_dma.h

- **DMA_ InitTypeDef 初始化结构体**

```
typedef struct
{
    uint32_t DMA_PeripheralBaseAddr;    // 外设地址
    uint32_t DMA_MemoryBaseAddr;        // 存储器地址
    uint32_t DMA_DIR;                   // 传输方向
    uint32_t DMA_BufferSize;            // 传输数目
    uint32_t DMA_PeripheralInc;         // 外设地址增量模式
    uint32_t DMA_MemoryInc;             // 存储器地址增量模式
    uint32_t DMA_PeripheralDataSize;    // 外设数据宽度
    uint32_t DMA_MemoryDataSize;        // 存储器数据宽度
    uint32_t DMA_Mode;                  // 模式选择
    uint32_t DMA_Priority;              // 通道优先级
    uint32_t DMA_M2M;                   // 存储器到存储器模式
} DMA_InitTypeDef;
```

使用DMA，最核心就是配置要传输的数据，包括数据从哪里来，要到哪里去，传输的数据的单位是什么，要传多少数据，是一次传输还是循环传输等等

DMA初始化结构体

一、数据从哪里来，要到哪里去

```
uint32_t DMA_PeripheralBaseAddr;    // 外设地址
uint32_t DMA_MemoryBaseAddr;        // 存储器地址
uint32_t DMA_DIR;                   // 传输方向
```

1、外设地址，DMA_CPAR ：设置为外设的数据寄存器地址，
2、存储器地址，DMA_CMAR
3、传输方向，DMA_CCR:DIR

位4	**DIR**：数据传输方向 (Data transfer direction) 该位由软件设置和清除。 0：从外设读 1：从存储器读

0 表示从外设到存储器，1 表示从存储器到外设

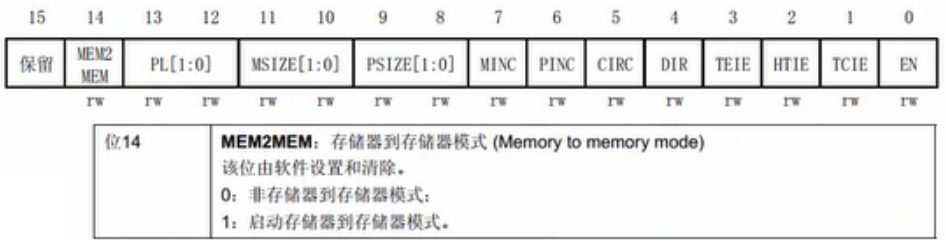
```
#define DMA_DIR_PeripheralDST          ((uint32_t)0x00000010)
#define DMA_DIR_PeripheralSRC          ((uint32_t)0x00000000)
```

DMA初始化结构体

一、数据从哪里来，要到哪里去

M->M

```
uint32_t DMA_M2M;                    // 存储器到存储器模式
```



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留	MEM2MEM	PL[1:0]		MSIZE[1:0]		PSIZE[1:0]		MINC	PINC	CIRC	DIR	TEIE	HTIE	TCIE	EN
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位14	**MEM2MEM**：存储器到存储器模式 (Memory to memory mode) 该位由软件设置和清除。 0：非存储器到存储器模式； 1：启动存储器到存储器模式。

1、外设地址，DMA_CPAR ：

设置为其中一个存储器地址。

2、存储器地址，DMA_CMAR

3、传输方向，DMA_CCR:DIR

```
DMA_InitStruct.DMA_PeripheralBaseAddr = (uint32_t)aSRC_Buffer;
DMA_InitStruct.DMA_MemoryBaseAddr = (uint32_t)aDST_Buffer;
DMA_InitStruct.DMA_DIR = DMA_DIR_PeripheralSRC;
DMA_InitStruct.DMA_M2M = DMA_M2M_Enable;

DMA_InitStruct.DMA_PeripheralBaseAddr = (uint32_t)aDST_Buffer;
DMA_InitStruct.DMA_MemoryBaseAddr = (uint32_t)aSRC_Buffer;
DMA_InitStruct.DMA_DIR = DMA_DIR_PeripheralDST              ;
DMA_InitStruct.DMA_M2M = DMA_M2M_Enable;
```

DMA初始化结构体

二、数据要传多少，传的单位是什么

```
uint32_t DMA_BufferSize;            // 传输数目
uint32_t DMA_PeripheralInc;         // 外设地址增量模式
uint32_t DMA_MemoryInc;             // 存储器地址增量模式
uint32_t DMA_PeripheralDataSize;    // 外设数据宽度
uint32_t DMA_MemoryDataSize;        // 存储器数据宽度
```

- 1、传输数目，DMA_CNDTR
- 2、外设地址是否递增，DMA_CCRx:PINC
- 3、存储器地址是否递增，DMA_CCRx:MINC
- 4、外设数据宽度， DMA_CCRx:PSIZE
- 5、存储器数据宽度， DMA_CCRx:MSIZE

DMA初始化结构体

三、什么时候传输结束

```
uint32_t DMA_Mode;                    // 模式选择
```

1、模式选择，DMA_CCRx:CIRC

- 一次传输: 即是传输一次之后就停止，要想再传输的话，必须关断 DMA 使能后再重新配置后才能继续传输。
- 循环传输: 则是一次传输完成之后又恢复第一次传输时的配置循环传输，不断重复。

2、每个通道有4个状态标志位：传输过半，传输完成，传输出错，DMA_ISR(DMA全局中断)

- 数据什么时候传输完成，通过查询标志位或者通过中断的方式来鉴别.每个 DMA 通道在 DMA 传输过半、传输完成和传输错误时都会有相应的标志位，如果使能了该类型的中断后，则会产生中断。

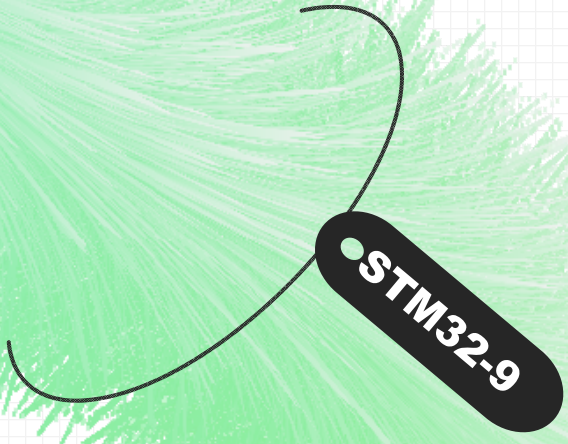
DMA库函数

```
406 /** @defgroup DMA_Exported_Functions
407   * @{
408   */
409
410 void DMA_DeInit(DMA_Channel_TypeDef* DMAy_Channelx);
411 void DMA_Init(DMA_Channel_TypeDef* DMAy_Channelx, DMA_InitTypeDef* DMA_InitStruct);
412 void DMA_StructInit(DMA_InitTypeDef* DMA_InitStruct);
413 void DMA_Cmd(DMA_Channel_TypeDef* DMAy_Channelx, FunctionalState NewState);
414 void DMA_ITConfig(DMA_Channel_TypeDef* DMAy_Channelx, uint32_t DMA_IT, FunctionalState NewState);
415 void DMA_SetCurrDataCounter(DMA_Channel_TypeDef* DMAy_Channelx, uint16_t DataNumber);
416 uint16_t DMA_GetCurrDataCounter(DMA_Channel_TypeDef* DMAy_Channelx);
417 FlagStatus DMA_GetFlagStatus(uint32_t DMAy_FLAG);
418 void DMA_ClearFlag(uint32_t DMAy_FLAG);
419 ITStatus DMA_GetITStatus(uint32_t DMAy_IT);
420 void DMA_ClearITPendingBit(uint32_t DMAy_IT);
```

```
  *     @arg DMA2_FLAG_GL5: DMA2 Channel5 global flag.
  *     @arg DMA2_FLAG_TC5: DMA2 Channel5 transfer complete flag.
  *     @arg DMA2_FLAG_HT5: DMA2 Channel5 half transfer flag.
  *     @arg DMA2_FLAG_TE5: DMA2 Channel5 transfer error flag.
  * @retval The new state of DMAy_FLAG (SET or RESET).
  */
FlagStatus DMA_GetFlagStatus(uint32_t DMAy_FLAG)
{
  FlagStatus bitstatus = RESET;
  uint32_t tmpreg = 0;
```

目录

CONTENTS.



实验设计

1-M to M：FLASH to SRAM，把内部FLASH的数据传输到内部的SRAM。

2-M to P：SRAM to 串口，同时LED灯闪烁，演示DMA传数据不需要占用CPU。

M To M 编程要点

- 1-在FLASH中定义好要传输的数据，在SRAM中定义好用来接收FLASH数据的变量。
- 2-初始化DMA，主要是配置DMA初始化结构体。
- 3-编写比较函数。
- 4-编写main函数。

1-在FLASH中定义好要传输的数据，在SRAM中定义好用来接收FLASH数据的变量。

bsp_dma_mtm.h

```
#ifndef __BSP_DMA_MTM_H
#define __BSP_DMA_MTM_H

#include "stm32f10x.h"

// 要发送的数据大小
#define BUFFER_SIZE     32

#define  MTM_DMA_CLK       RCC_AHBPeriph_DMA1
#define  MTM_DMA_CHANNEL   DMA1_Channel6
#define  MTM_DMA_FLAG_TC   DMA1_FLAG_TC6

void MtM_DMA_Config(void);
uint8_t Buffercmp(const uint32_t* pBuffer,  uint32_t* pBuffer1, uint16_t BufferLength);

#endif  /* __BSP_DMA_MTM_H */
```

1-在FLASH中定义好要传输的数据，在SRAM中定义好用来接收FLASH数据的变量。

bsp_dma_mtm.c

```
#include "bsp_dma_mtm.h"

/* 定义aSRC_Const_Buffer数组作为DMA传输数据源
 * const关键字将aSRC_Const_Buffer数组变量定义为常量类型
 * 表示数据存储在内部的FLASH中 */
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};

/* 定义DMA传输目标存储器存储在内部的SRAM中 */
uint32_t aDST_Buffer[BUFFER_SIZE];
```

2-初始化DMA，主要是配置DMA初始化结构体。

bsp_dma_mtm.c

```
void MtM_DMA_Config(void) {
        DMA_InitTypeDef DMA_InitStruct;

        RCC_AHBPeriphClockCmd(MTM_DMA_CLK, ENABLE);          // 开启DMA时钟
        DMA_InitStruct.DMA_PeripheralBaseAddr = (uint32_t)aSRC_Const_Buffer; // 源数据地址
        DMA_InitStruct.DMA_MemoryBaseAddr = (uint32_t)aDST_Buffer; // 目标地址
        DMA_InitStruct.DMA_DIR = DMA_DIR_PeripheralSRC; // 方向：外设到存储器
        DMA_InitStruct.DMA_BufferSize = BUFFER_SIZE; // 传输大小
        DMA_InitStruct.DMA_PeripheralInc = DMA_PeripheralInc_Enable; // 外设地址递增
        DMA_InitStruct.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word; // 外设数据单位
        DMA_InitStruct.DMA_MemoryInc = DMA_MemoryInc_Enable; // 内存地址递增
        DMA_InitStruct.DMA_MemoryDataSize = DMA_MemoryDataSize_Word; // 内存数据单位
        DMA_InitStruct.DMA_Mode = DMA_Mode_Normal; // DMA模式，一次(非循环)模式
        DMA_InitStruct.DMA_Priority = DMA_Priority_High; // 优先级：高
        DMA_InitStruct.DMA_M2M = DMA_M2M_Enable; // 使能内存到内存的传输

        DMA_Init(MTM_DMA_CHANNEL, &DMA_InitStruct);  // 配置DMA通道
        DMA_ClearFlag(MTM_DMA_FLAG_TC); //清除TC标志
        DMA_Cmd(MTM_DMA_CHANNEL, ENABLE);  } // 使能DMA
```

3-编写比较函数。

bsp_dma_mtm.c

```
1  uint8_t Buffercmp(const uint32_t* pBuffer,
2                    uint32_t* pBuffer1, uint16_t BufferLength)
3  {
4      /* 数据长度递减 */
5      while (BufferLength--) {
6          /* 判断两个数据源是否对应相等 */
7          if (*pBuffer != *pBuffer1) {
8              /* 对应数据源不相等马上退出函数，并返回 0 */
9              return 0;
10         }
11         /* 递增两个数据源的地址指针 */
12         pBuffer++;
13         pBuffer1++;
14     }
15     /* 完成判断并且对应数据相对 */
16     return 1;
17 }
```

4-编写main函数。

```c
int main(void)
{
    /* 定义存放比较结果变量 */
    uint8_t TransferStatus;

    /* LED 端口初始化 */
    LED_GPIO_Config();

    /* 设置 RGB 彩色灯为紫色 */
    LED_PURPLE;

    /* 简单延时函数 */
    Delay(0xFFFFFF);

    /* DMA 传输配置 */
    DMA_Config();

    /* 等待 DMA 传输完成 */
    while (DMA_GetFlagStatus(DMA_FLAG_TC)==RESET)
    {

    }

    /* 比较源数据与传输后数据 */
    TransferStatus=Buffercmp(aSRC_Const_Buffer, aDST_Buffer, BUFFER_SIZE);

    /* 判断源数据与传输后数据比较结果*/
    if (TransferStatus==0)
    {
        /* 源数据与传输后数据不相等时 RGB 彩色灯显示红色 */
        LED_RED;
    }
    else
    {
        /* 源数据与传输后数据相等时 RGB 彩色灯显示蓝色 */
        LED_BLUE;
    }

    while (1)
    {
    }
}
```

M To P 编程要点

- 1-初始化外设（串口）（从现有的例程移植过来）
- 2-配置DMA初始化结构体。
- 3-编写主函数（开启外设(串口)发送DMA请求,

  DMA传输同时CPU可以运行其他任务）。

1-初始化外设

bsp_usart_dma.h

```
#include "stm32f10x.h"
#include <stdio.h>

// 串口工作参数宏定义
#define  DEBUG_USARTx                   USART1
#define  DEBUG_USART_CLK                RCC_APB2Periph_USART1
#define  DEBUG_USART_APBxClkCmd         RCC_APB2PeriphClockCmd
#define  DEBUG_USART_BAUDRATE           115200

// USART GPIO 引脚宏定义
#define  DEBUG_USART_GPIO_CLK           (RCC_APB2Periph_GPIOA)
#define  DEBUG_USART_GPIO_APBxClkCmd    RCC_APB2PeriphClockCmd

#define  DEBUG_USART_TX_GPIO_PORT       GPIOA
#define  DEBUG_USART_TX_GPIO_PIN        GPIO_Pin_9
#define  DEBUG_USART_RX_GPIO_PORT       GPIOA
#define  DEBUG_USART_RX_GPIO_PIN        GPIO_Pin_10

// 串口对应的DMA请求通道
#define  USART_TX_DMA_CHANNEL     DMA1_Channel4
// 外设寄存器地址
#define  USART_DR_ADDRESS        (USART1_BASE+0x04)
```

```
#define USART1_BASE
(APB2PERIPH_BASE + 0x3800)


// 一次发送的数据量
#define  SENDBUFF_SIZE          5000


void USART_Config(void);
void USARTx_DMA_Config(void);

#endif /* __USARTDMA_H */
```

25.6.2　数据寄存器(USART_DR)

地址偏移：0x04

1-初始化外设

bsp_usart_dma.c

```
void USART_Config(void){
        GPIO_InitTypeDef GPIO_InitStructure;
        USART_InitTypeDef USART_InitStructure;

        DEBUG_USART_GPIO_APBxClkCmd(DEBUG_USART_GPIO_CLK, ENABLE); // 打开串口GPIO的时钟

        DEBUG_USART_APBxClkCmd(DEBUG_USART_CLK, ENABLE); // 打开串口外设的时钟

        // 将USART Tx的GPIO配置为推挽复用模式
        GPIO_InitStructure.GPIO_Pin = DEBUG_USART_TX_GPIO_PIN;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_Init(DEBUG_USART_TX_GPIO_PORT, &GPIO_InitStructure);

          // 将USART Rx的GPIO配置为浮空输入模式
        GPIO_InitStructure.GPIO_Pin = DEBUG_USART_RX_GPIO_PIN;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
        GPIO_Init(DEBUG_USART_RX_GPIO_PORT, &GPIO_InitStructure);

        // 配置串口的工作参数
        USART_InitStructure.USART_BaudRate = DEBUG_USART_BAUDRATE; // 配置波特率
        USART_InitStructure.USART_WordLength = USART_WordLength_8b; // 配置帧数据字长
        USART_InitStructure.USART_StopBits = USART_StopBits_1; // 配置停止位
        USART_InitStructure.USART_Parity = USART_Parity_No ; // 配置校验位
        USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None; // 配置硬件流控制

        USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx; // 配置工作模式，收发一起
        USART_Init(DEBUG_USARTx, &USART_InitStructure); // 完成串口的初始化配置
        USART_Cmd(DEBUG_USARTx, ENABLE);     // 使能串口
}
```

2-配置DMA初始化结构体。

bsp_usart_dma.c

```
void USARTx_DMA_Config(void)
{
    DMA_InitTypeDef DMA_InitStructure;

    // 开启DMA时钟
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
    // 设置DMA源地址：串口数据寄存器地址*/
    DMA_InitStructure.DMA_PeripheralBaseAddr = USART_DR_ADDRESS;
    // 内存地址(要传输的变量的指针)
    DMA_InitStructure.DMA_MemoryBaseAddr = (u32)SendBuff;
    // 方向：从内存到外设
    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralDST;
    // 传输大小
    DMA_InitStructure.DMA_BufferSize = SENDBUFF_SIZE;
    // 外设地址不增
    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
    // 内存地址自增
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
    // 外设数据单位
    DMA_InitStructure.DMA_PeripheralDataSize =
    DMA_PeripheralDataSize_Byte;
    // 内存数据单位
    DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Byte;
    // DMA模式，一次或者循环模式
    DMA_InitStructure.DMA_Mode = DMA_Mode_Normal ;
    //DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
    // 优先级：中
    DMA_InitStructure.DMA_Priority = DMA_Priority_Medium;
    // 禁止内存到内存的传输
    DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
    // 配置DMA通道
    DMA_Init(USART_TX_DMA_CHANNEL, &DMA_InitStructure);
    // 使能DMA
    DMA_Cmd (USART_TX_DMA_CHANNEL,ENABLE);
}
```

3-编写主函数

main.c

```c
int main(void)
{
  uint16_t i;
  /* 初始化USART */
  USART_Config();

  /* 配置使用DMA模式 */
  USARTx_DMA_Config();

  /* 配置RGB彩色灯 */
  LED_GPIO_Config();

  //printf("\r\n USART1 DMA TX 测试 \r\n");

  /*填充将要发送的数据*/
  for(i=0;i<SENDBUFF_SIZE;i++)
  {
    SendBuff[i]  = 'P';

  }

  /*为演示DMA持续运行而CPU还能处理其它事情，持续使用DMA发送数据，量非常大，
  *长时间运行可能会导致电脑端串口调试助手会卡死，鼠标乱飞的情况，
  *或把DMA配置中的循环模式改为单次模式*/

  /* USART1 向 DMA发出TX请求 */
  USART_DMACmd(DEBUG_USARTx, USART_DMAReq_Tx, ENABLE);

  /* 此时CPU是空闲的，可以干其他的事情 */
  //例如同时控制LED
  while(1)
  {
    LED1_TOGGLE
    Delay(0xFFFFF);
  }
}
```

Reference

stm32f10x_usart.c

```
430   }
431 }
432
433 /**
434   * @brief  Enables or disables the USART指 DMA interface.
435   * @param  USARTx: Select the USART or the UART peripheral.
436   *   This parameter can be one of the following values:
437   *   USART1, USART2, USART3, UART4 or UART5.
438   * @param  USART_DMAReq: specifies the DMA request.
439   *   This parameter can be any combination of the following values:
440   *     @arg USART_DMAReq_Tx: USART DMA transmit request
441   *     @arg USART_DMAReq_Rx: USART DMA receive request
442   * @param  NewState: new state of the DMA Request sources.
443   *   This parameter can be: ENABLE or DISABLE.
444   * @note The DMA mode is not available for UART5 except in the STM32
445   *       High density value line devices(STM32F10X_HD_VL).
446   * @retval None
447   */
448 void USART_DMACmd(USART_TypeDef* USARTx, uint16_t USART_DMAReq, FunctionalState NewState)
449 {
450   /* Check the parameters */
451   assert_param(IS_USART_ALL_PERIPH(USARTx));
452   assert_param(IS_USART_DMAREQ(USART_DMAReq));
453   assert_param(IS_FUNCTIONAL_STATE(NewState));
454   if (NewState != DISABLE)
455   {
456     /* Enable the DMA transfer for selected requests by setting the DMAT and/or
457        DMAR bits in the USART CR3 register */
458     USARTx->CR3 |= USART_DMAReq;
459   }
```