实验 29　LCD-ADC-DMA 实验

本章我们将向大家介绍 STM32 的 DMA。 在本章中，我们将利用 STM32 的 DMA 来实现 ADC1 通道 1 内数据传送，并在 TFTLCD 模块上显示传送数据 AD 的电压值。本章分为以下学习目标：

1. 了解什么是 DMA

2. 懂得操作 STM32 的 DMA

1.1　STM32 DMA 简介

DMA，全称为： Direct Memory Access，即直接存储器访问， DMA 传输将数据从一个地址空间复制到另外一个地址空间。 当 CPU 初始化这个传输动作，传输动作本身是由 DMA 控制器 来实行和完成。典型的例子就是移动一个外部内存的区块到芯片内部更快的内存区。像是这样的操作并没有让处理器工作拖延，反而可以被重新排程去处理其他的工作。DMA 传输对于高效能嵌入式系统算法和网络是很重要的。DMA 传输方式无需 CPU 直接控制传输，也没有中断处理方式那样保留现场和恢复现场的过程，通过硬件为 RAM 与 I/O 设备开辟一条直接传送数据的通路， 能使 CPU 的效率大为提高。

STM32 最多有 2 个 DMA 控制器（ DMA2 仅存在大容量产品中）， DMA1 有 7 个通道。DMA2 有 5 个通道。每个通道专门用来管理来自于一个或多个外设对存储器访问的请求。还有一个仲裁起来协调各个 DMA 请求的优先权。

STM32 的 DMA 有以下一些特性：

●每个通道都直接连接专用的硬件 DMA 请求，每个通道都同样支持软件触发。这些功能通过软件来配置。

●在七个请求间的优先权可以通过软件编程设置(共有四级：很高、高、中等和低)，假如在相等优先权时由硬件决定(请求 0 优先于请求 1，依此类推) 。

●独立的源和目标数据区的传输宽度(字节、半字、全字)，模拟打包和拆包的过程。源和目标地址必须按数据传输宽度对齐。

●支持循环的缓冲器管理

●每个通道都有 3 个事件标志(DMA 半传输， DMA 传输完成和 DMA 传输出错)，这 3 个事件标志逻辑或成为一个单独的中断请求。

●存储器和存储器间的传输

●外设和存储器，存储器和外设的传输

●闪存、 SRAM、外设的 SRAM、 APB1 APB2 和 AHB 外设均可作为访问的源和目标。

●可编程的数据传输数目：最大为 65536

STM32F103ZET6 有两个 DMA 控制器， DMA1 和 DMA2， 本章我们仅针对 DMA1 进行介绍。

从外设（ TIMx、 ADC、 SPIx、 I2Cx 和 USARTx）产生的 DMA 请求，通过逻辑或输入到 DMA 控制器，这就意味着同时只能有一个请求有效。外设的 DMA 请求，可以通过设置相应的外设寄存器中的控制位，被独立地开启或关闭。

接下来我们来看一下，DMA 1 主要有哪些通道：

外设	通道1	通道2	通道3	通道4	通道5	通道6	通道7
ADC	ADC1						
SPI		SPI1_RX	SPI1_TX	SPI2_RX	SPI2_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

这里解释一下上面说的逻辑或，例如通道 1 的几个 DMA1 请求（ ADC1、 TIM2_CH3、TIM4_CH1），这几个是通过逻辑或到通道 1 的，这样我们在同一时间，就只能使用其中的一个。其他通道也是类似的。

接下来我们来看一下，DMA 2 主要有哪些通道：

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

从上图我们看出 STM32 的 DMA 很多，但大致操作时一样的，这里我们主要讲述 ADC DMA 的操作方式。

1.2 STM32 DMA 的操作

1. DMA 的初始化

DMA 配置通道的过程为：

1. 在DMA_CPARx寄存器中设置外设寄存器的地址。发生外设数据传输请求时，这个地址将是数据传输的源或目标。
2. 在DMA_CMARx寄存器中设置数据存储器的地址。发生外设数据传输请求时，传输的数据将从这个地址读出或写入这个地址。
3. 在DMA_CNDTRx寄存器中设置要传输的数据量。在每个数据传输后，这个数值递减。
4. 在DMA_CCRx寄存器的PL[1:0]位中设置通道的优先级。
5. 在DMA_CCRx寄存器中设置数据传输的方向、循环模式、外设和存储器的增量模式、外设和存储器的数据宽度、传输一半产生中断或传输完成产生中断。
6. 设置DMA_CCRx寄存器的ENABLE位，启动该通道。

不过呢，我们使用 V3.5 库函数操作的话，我们只需要调用 DMA_Init()函数就可以了。

DMA_Init()函数有两个一个输入参数：

1) DMA_Channelx：用来选择要初始化的通道参数。我们这里以 ADC1 为例，查看上面的 DMA1 的通道表，我们知道 ADC 1 的 DMA 通道是是通道 1，所以打开 ADC1 的通道：DMA1_Channel1。

2) DMA_InitStruct：这个是用来设置 DAM 参数的结构，它一共有 11 个成员，它的成员如下：

- DMA_PeripheralBaseAddr：外设基地址，也就是你 DMA 要传输数据的目的地，我们是要使用到 ADC1，所以设置为 ADC1 的地址：(u32)&ADC1->DR（注意是地址）。
- DMA_MemoryBaseAddr：内存基地址，也就是你 DMA 要搬运数据的存放的首地址，如果你是定义了一组数组发送，那么就设置为你的数据首地址。
- DMA_BufferSize：DMA 通道的缓存大小。即设置一次传输数据量的大小，这个很容易理解。
- DMA_PeripheralInc：是否使能外设地址递增。我们直接使用 ADC1 的发送寄存器，地址不递增，所以设置为：DMA_PeripheralInc_Disable。
- DMA_MemoryInc：是否使能内存地址递增。我们发送的数据发送完一位，肯定是发送下一位，所以这里肯定要使能：DMA_MemoryInc_Enable。

■ DMA_PeripheralDataSize：设置外设数据宽度，ADC 保存数组内数据是 16 位，所以设置为：DMA_PeripheralDataSize_HalfWord。

■ DMA_MemoryDataSize：内存数据宽度，这里也设置为：DMA_MemoryDataSize_HalfWord。

■ DMA_Mode：传送模式。DMA 的工作模式又两种，第 1 种是工作在循环缓存模式，也就是将数据一直重复发送，第 2 种工作在正常缓存模式，也就是只将数据传送一次。我们这里设置为循环缓存模式：DMA_Mode_Circular。

■ DMA_Priority：DMA 的传输顺序是有优先级的，当有两个 DMA 同时使用的时候，优先级高的先发送。而 DMA 的等级一共有四种等级：

a) 最高优先级

b) 高优先级

c) 中优先级

d) 低优先级

我们这里只使用一个 DMA，随便设置就好。

■ DMA_M2M：是否使用存储器到存储器模式，我们设置为：DMA_M2M_Disable。

2. 初始化代码

```
void dma_init()          //DMA 初始化
{
    //结构体定义
    GPIO_InitTypeDef GPIO_InitStructure;
    ADC_InitTypeDef ADC_InitStructure;
    DMA_InitTypeDef DMA_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_AFIO|RCC_APB2Periph_ADC1,ENABLE);
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1,ENABLE);
    RCC_ADCCLKConfig(RCC_PCLK2_Div6);//12M  最大 14M 设置 ADC 时钟（ADCCLK）
    ADC_DeInit(ADC1);
```

```
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_1;//ADC    //1
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AIN;     //模拟输入
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
    GPIO_Init(GPIOA,&GPIO_InitStructure);           //A


    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
    ADC_InitStructure.ADC_ScanConvMode = DISABLE;
    ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
    ADC_InitStructure.ADC_NbrOfChannel = 1;
    ADC_Init(ADC1, &ADC_InitStructure);

    //设置指定 ADC 的规则组通道，设置它们的转化顺序和采样时间
    ADC_RegularChannelConfig(ADC1,ADC_Channel_1,1,ADC_SampleTime_239Cycles5);
//1

    //内部温度传感器是在 ADC1 通道 16 的。
//  ADC_RegularChannelConfig(ADC1,ADC_Channel_16,1,ADC_SampleTime_239Cycles5);
//  ADC_TempSensorVrefintCmd(ENABLE);//打开内部温度传感器使能

    ADC_DMACmd(ADC1,ENABLE);//将 ADC 与 DMA 链接在一起
    ADC_Cmd(ADC1,ENABLE);

    ADC_ResetCalibration(ADC1);//重置指定的 ADC 的校准寄存器
    while(ADC_GetResetCalibrationStatus(ADC1));//获取 ADC 重置校准寄存器的状态
    ADC_StartCalibration(ADC1);//开始指定 ADC 的校准状态
    while(ADC_GetCalibrationStatus(ADC1));//获取指定 ADC 的校准程序
```

ADC_SoftwareStartConvCmd(ADC1, ENABLE);//使能或者失能指定的 ADC 的软件转换启动功能

DMA_DeInit(DMA1_Channel1);

DMA_InitStructure.DMA_PeripheralBaseAddr =(u32)&ADC1->DR;//DMA 外设地址

DMA_InitStructure.DMA_MemoryBaseAddr=(u32)&adc_data;//DMA 内存地址

DMA_InitStructure.DMA_DIR=DMA_DIR_PeripheralSRC;//外设作为数据传输的来源

DMA_InitStructure.DMA_BufferSize=1;//指定 DMA 通道的 DMA 缓存的大小

DMA_InitStructure.DMA_PeripheralInc=DMA_PeripheralInc_Disable;//外设地址寄存器递增

DMA_InitStructure.DMA_MemoryInc=DMA_PeripheralInc_Enable;//内存地址寄存器递增

DMA_InitStructure.DMA_PeripheralDataSize=DMA_PeripheralDataSize_HalfWord;//外设数据宽度 16

DMA_InitStructure.DMA_MemoryDataSize=DMA_MemoryDataSize_HalfWord;//存储数据宽度 16

DMA_InitStructure.DMA_Mode=DMA_Mode_Circular;//工作在循环缓存模式

DMA_InitStructure.DMA_Priority=DMA_Priority_High;//DMA 通道 x 拥有高优先级

DMA_InitStructure.DMA_M2M=DMA_M2M_Disable;//DMA 通道 x 没有设置为内存到内存传输

DMA_Init(DMA1_Channel1,&DMA_InitStructure); //ADC1 在 DMA1 通道 1 内

DMA_Cmd(DMA1_Channel1,ENABLE);//使能 DMA1

}

3. 主函数代码

```
/*******************************************************************************
**
* Function Name  : main
* Description    : Main program.
* Input          : None
* Output         : None
```

```
* Return         : None
*******************************************************************************
*/
int main()
{
    u16 value;
    u8 i,j,dat[6];
    float ad;
    dma_init();   //DMA 初始化
    TFT_Init();  //TFT 彩屏初始化
    LED_Init();   //端口初始化
    printf_init();  //printf 初始化
    TFT_ClearScreen(BLACK);    //清屏
    GUI_Show12ASCII(10,10,"This is a ADC1-DMA1 Check!",YELLOW,BLACK);
    GUI_Show12ASCII(10,27,"PA1 is AD Input!",YELLOW,BLACK);
    GUI_Show12ASCII(10,100,"The AD Volage is:",YELLOW,BLACK);
    while(1)
    {
        value=0;
        for(i=0;i<10;i++)//读取 10 次的 AD 数值取其平均数较为准确
        {
            value=value+adc_data[0];
        }
        delay_ms(500);
        value=value/10;
        ad=value*3.3/4095;
        value=(u16)(ad*100);
        dat[0]=value/100+0x30;
        dat[1]='.';
        dat[2]=value%100/10+0x30;
```

```
            dat[3]=value%100%10+0x30;
            dat[4]='V';
            dat[5]='\0';
            GUI_Show12ASCII(160,100,dat,YELLOW,BLACK);
            if(j>1)
            {
                j=0;
                GPIO_SetBits(GPIOC,GPIO_Pin_0);
            }
            else
            {
                j++;
                GPIO_ResetBits(GPIOC,GPIO_Pin_0);
            }

        }
    }
```