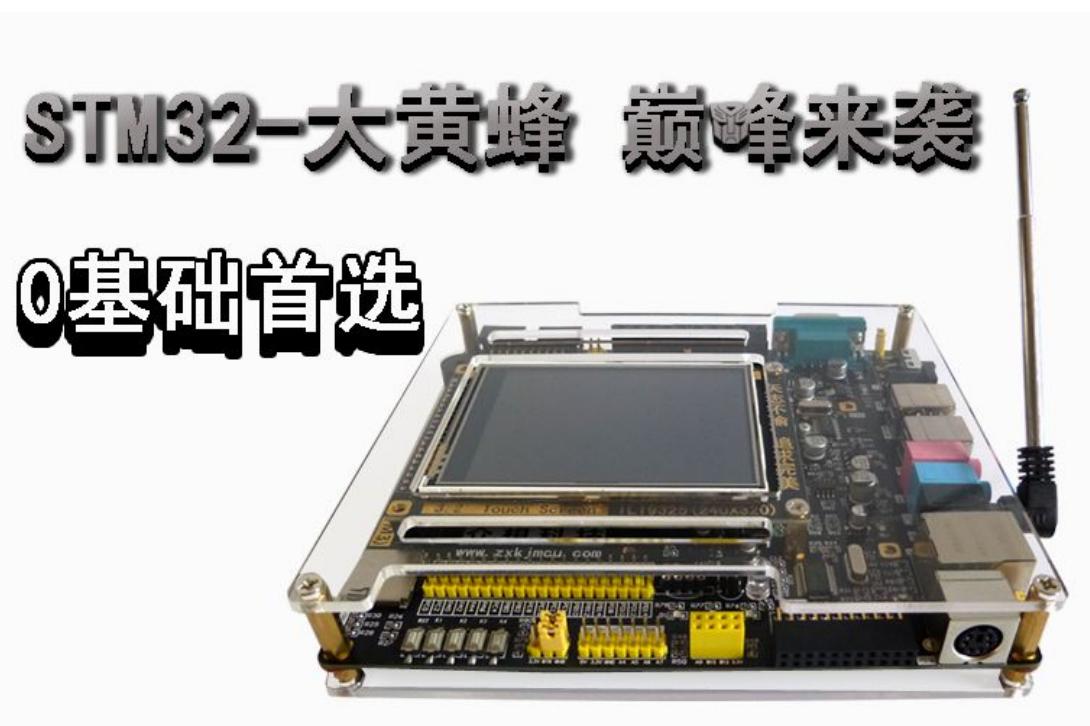


学 ARM 从 STM32 开始

STM32 开发板库函数教程—实战篇



官方网站: <http://www.zxkjmcu.com>

官方店铺: <http://zxkjmcu.taobao.com>

官方论坛: <http://bbs.zxkjmcu.com>

刘洋课堂: <http://school.zxkjmcu.com>

4.13 STM32 DMA 工作原理及程序设计

4.13.1 概述

4.13.1.1 DMA 概念

DMA（直接内存访问(Direct Memory Access)）。这是指一种高速的数据传输操作，允许在外部设备和存储器之间直接读写数据，既不通过 CPU，也不需要 CPU 干预。整个数据传输操作在一个称为“DMA 控制器”的控制下进行的。CPU 除了在数据传输开始和结束时做一点处理外，在传输过程中 CPU 可以进行其他的工作。这样，在大部分时间里，CPU 和输入输出都处于并行操作。因此，使整个计算机系统的效率大大提高。

DMA 是在专门的硬件（DMA）控制下，实现高速外设和主存储器之间自动成批交换数据尽量减少 CPU 干预的输入/输出操作方式。通常有三种方式：

◎停止 CPU 访问 ◎周期挪用方式 ◎DMA 与 CPU 交替访内存。

4.13.1.2 DMA 的传送数据的过程，由三个阶段组成：

◎传送前的预处理：由 CPU 完成以下步骤：

向 DMA 模块送入设备识别信号，启动设备，测试设备运行状态，送入内存地址初值，传送数据个数， DMA 的功能控制信号。

◎数据传送：在 DMA 模块控制下自动完成

◎传送结束处理

4.13.1.3 DMA 模块上应包括通用接口模块的全部组成部分，并多出如下内容：主存地址寄存器，传送字数计数器，DMA 控制逻辑，DMA 请求，DMA 响应，DMA 工作方式，DMA 优先级及排队逻辑等。

4.13.1.4 一次完整的 DMA 传送过程：

DMA 预处理，CPU 向 DMA 送命令，如 DMA 方式，主存地址，传送的字数等，之后 CPU 执行原来的程序。

4.13.1.5 DMA 控制在 I/O 设备与主存间交换数据：

准备一个数据，向 CPU 发 DMA 请求，取得总线控制权，进行数据传送，修改模块上主存地址，修改字数计数器内且检查其值是否为零，不为零则继续传送，若已为零，则向 CPU 发中断请求。

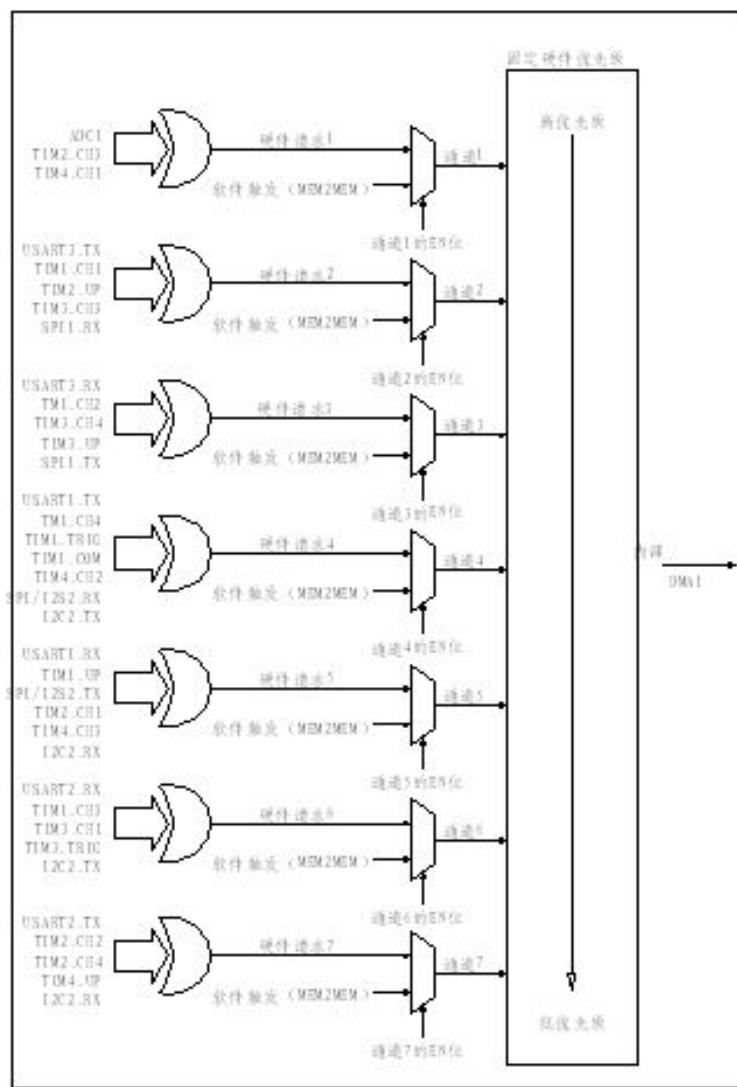
4.13.1.6 DMA 技术的弊端：

因为 DMA 允许外设直接访问内存，从而形成对总线的独占。

4.13.2 STM32 DMA 主要特征

- ◎12 个独立的可配置的通道（请求）：DMA1 有 7 个通道，DMA2 有 5 个通道；
- ◎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；
- ◎在同一个 DMA 模块上，多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、低），优先权设置相等时由硬件决定（请求 0 优先于请求 1，依此类推）；
- ◎独立数据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)，这三个事件标志逻辑或称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、存储器和外设之间的传输；

- ◎闪存、SRAM、外设的SRAM、APB1、APB2、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65535；



4.13.3 STM32 DMA 请求映射

从外设 (TIM_x[x=1、2、3、4]、ADC1、SPI1、SPI/I2C2、I2Cx[X=1、2]和USARTx[x=1、2、3]) 产生的7个请求，通过逻辑或输入到DMA控制器，这意味着同时只能有一个请求有效。参见下图的DMA请求映像。外设的DMA请求，可以通过设置相应外设寄存器中的控制器，被独立地开启

或关闭。

DMA 控制器说明

DMA 控制器说明-各个通道的 DMA1 请求一览

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/I2S2_RX	SPI/I2S2_TX		
USART		USART3_TX	USART3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
I2C				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

1 数据传输的目的地和来源

DMA_DIR 规定了外设是作为数据传输的目的地还是来源。

DMA_DIR	描述
DMA_DIR_PeripheralDST	外设作为数据传输的目的地
DMA_DIR_PeripheralSRC	外设作为数据传输的来源

2、定义 DMA 通道的 DMA 缓存的大小

DMA_BufferSize 用以指定 DMA 通道的 DMA 缓存的大小，单位为数据单位。

根据传输方向，数据单位等于结构中参数 DMA_PeripheralDataSize 或者参数 DMA_MemoryDataSize 的值。

3、外设地址寄存器递增与否

DMA_PeripheralInc 用来设定外设地址寄存器递增与否。

DMA_PeripheralInc	描述
DMA_PeripheralInc_Enable	外设地址寄存器递增
DMA_PeripheralInc_Disable	外设地址寄存器不变

4、内存地址寄存器递增与否

用来设定内存地址寄存器地址寄存器递增与。

DMA_MemoryInc	描述
DMA_PeripheralInc_Enable	内存地址寄存器递增
DMA_PeripheralInc_Disable	内存地址寄存器不变

5、设定了外设数据宽度

DMA_PeripheralDataSize 设定了外设数据宽度。

DMA_PeripheralDataSize	描述
DMA_PeripheralDataSize_Byte	数据宽度为 8 位
DMA_PeripheralDataSize_HalfWord	数据宽度为 16 位
DMA_PeripheralDataSize_Word	数据宽度为 32 位

6、DMA_MemoryDataSize 设定了内存数据宽度

DMA_MemoryDataSize	描述
DMA_MemoryDataSize_Byte	数据宽度为 8 位
DMA_MemoryDataSize_HalfWord	数据宽度为 16 位
DMA_MemoryDataSize_Word	数据宽度为 32 位

7、DMA_Mode 设置了 DMA 的工作模式

DMA_Mode	描述
DMA_Mode_Circular	工作在循环缓存模式
DMA_Mode_Normal	工作在正常缓存模式

8、DMA 通道的软件优先级

DMA_Mode	描述
DMA_Priority_VeryHigh	DMA 通道 x 拥有非常高优先级
DMA_Priority_High	DMA 通道 x 拥有高优先级
DMA_Priority_Medium	DMA 通道 x 拥有中优先级
DMA_Priority_Low	DMA 通道 x 拥有低优先级

9、使能 DMA 通道的内存到内存传输

DMA_M2M	描述
DMA_M2M_Enable	DMA 通道 x 设置为内存到内存传输
DMA_M2M_Disable	DMA 通道 x 没有设置为内存到内存传输

4.13.4 STM32 DMA 实验目的

DMA 不是一个独立功能模块，需要和其他功能外设配合使用，目的是增强其他功能的传输性能，降低 CPU 的占有率。本次试验中要采用 ADC 和 DMA

的配合使用，实现 ADC 的数据采集等转换功能。

4.13.5 硬件设计

利用实验板上的 ADC 模拟量转换电路，通过软件设计实现这个功能。

4.13.6 DMA 模拟量转换软件设计

ADC 采集进来的数据，通过 DMA 把数据放入内存中，再通过主程序把内存的数据转送到串口输出。

4.13.6.1 STM32 库函数文件

```
stm32f10x_gpio.c  
stm32f10x_rcc.c  
Misc.c // 中断控制字（优先级设置）库函数  
stm32f10x_exti.c // 外部中断库处理函数  
stm32f10x_tim.c // 定时器库处理函数  
stm32f10x_usart.c // 串口通讯函数  
stm32f10x_dac.c // DAC 模拟量转换函数  
stm32f10x_dma.c // DMA
```

本节实验及以后的实验我们都是用到库文件，其中 `stm32f10x_gpio.h` 头文件包含了 GPIO 端口的定义。`stm32f10x_rcc.h` 头文件包含了系统时钟配置函数以及相关的外设时钟使能函数，所以我们要把这两个头文件对应的 `stm32f10x_gpio.c` 和 `stm32f10x_rcc.c` 加到工程中；`Misc.c` 库函数主要包含了中断优先级的设置，`stm32f10x_exti.c` 库函数主要包含了外部中断设置参数，`stm32f10x_tim.c` 库函数主要包含定时器设置，`stm32f10x_usart.c` 库函数主要包含串行通讯设置，`stm32f10x_dac.c` 库函数主要包含 DAC 模拟量转换设置，`stm32f10x_dma.c` 库函数主要包含 DMA 功能模块设置，这些函数也要添加到函数库中。以上库文件包含了本次实验所有要用到的函数使用功能。

4.13.6.2 自定义头文件

pbdata.h
pbdata.c

同时我们自己也创建了两个公共的文件，这两个文件主要存放我们自定义的公共函数和全局变量，以方便以后每个功能模块之间传递参数。

4.13.6.3 pbdata.h 文件里的内容是

```
#ifndef _pbdata_H
#define _pbdata_H

#include "stm32f10x.h"
#include "misc.h"
#include "stm32f10x_exti.h"
#include "stm32f10x_tim.h"
#include "stm32f10x_usart.h"
#include "stm32f10x_dac.h"
#include "stm32f10x_dma.h"
#include "stdio.h"

extern u8 dt;//定义变量

void RCC_HSE_Configuration(void); //定义函数
void delay(u32 nCount);
void delay_us(u32 nus);
void delay_ms(u16 nms);

#endif
```

语句 #ifndef、#endif 是为了防止 pbdata.h 文件被多个文件调用时出现错误提示。如果不加这两条语句，当两个文件同时调用 pbdata 文件时，会提示重复调用错误。

4.13.6.4 pbdata.c 文件里的内容是

```
#include "pbdata.h"

u8 dt=0;

void RCC_HSE_Configuration(void) //HSE 作为 PLL 时钟，PLL 作为 SYSCLK
{
    RCC_DeInit(); /*将外设 RCC 寄存器重设为缺省值 */
```

```
RCC_HSEConfig(RCC_HSE_ON); /*设置外部高速晶振 (HSE) HSE 晶振打开(ON)*/  
  
    if(RCC_WaitForHSEStartUp() == SUCCESS) { /*等待 HSE 起振， SUCCESS: HSE 晶  
振稳定且就绪*/  
        RCC_HCLKConfig(RCC_SYSCLK_Div1);/*设置 AHB 时钟 (HCLK) RCC_SYSCLK_Div1——  
AHB 时钟 = 系统时*/  
        RCC_PCLK2Config(RCC_HCLK_Div1); /*设置高速 AHB 时钟 (PCLK2) RCC_HCLK_Div1——  
APB2 时钟 = HCLK*/  
        RCC_PCLK1Config(RCC_HCLK_Div2); /*设置低速 AHB 时钟 (PCLK1) RCC_HCLK_Div2——  
APB1 时钟 = HCLK / 2*/  
        RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);/*设置 PLL 时钟源及倍频  
系数*/  
        RCC_PLLCmd(ENABLE); /*使能 PLL */  
        while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET); /*检查指定的 RCC 标志  
位 (PLL 准备好标志) 设置与否*/  
        RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK); /*设置系统时钟 (SYSCLK) */  
        while(RCC_GetSYSCLKSource() != 0x08); /*0x08: PLL 作为系统时钟 */  
    }  
}  
  
void delay(u32 nCount)  
{  
    for(;nCount!=0;nCount--);  
}  
  
/*********************************************  
*  
* 名 称: delay_us(u32 nus)  
* 功 能: 微秒延时函数  
* 入口参数: u32 nus  
* 出口参数: 无  
* 说 明:  
* 调用方法: 无  
*****  
/  
void delay_us(u32 nus)  
{  
    u32 temp;  
    SysTick->LOAD = 9*nus;  
    SysTick->VAL=0X00;//清空计数器  
    SysTick->CTRL=0X01;//使能, 减到零是无动作, 采用外部时钟源  
    do  
    {  
        temp=SysTick->CTRL;//读取当前倒计数值
```

```
    }while((temp&0x01)&&(! (temp&(1<<16))));//等待时间到达

    SysTick->CTRL=0x00; //关闭计数器
    SysTick->VAL =0X00; //清空计数器
}

//*****************************************************************************
/*
* 名      称: delay_ms(u16 nms)
* 功      能: 毫秒延时函数
* 入口参数: u16 nms
* 出口参数: 无
* 说      明:
* 调用方法: 无
*****
*/
void delay_ms(u16 nms)
{
    //注意 delay_ms 函数输入范围是 1-1864
    //所以最大延时为 1.8 秒

    u32 temp;
    SysTick->LOAD = 9000*nms;
    SysTick->VAL=0X00;//清空计数器
    SysTick->CTRL=0X01;//使能, 减到零是无动作, 采用外部时钟源
    do
    {
        temp=SysTick->CTRL;//读取当前倒计数值
    }while((temp&0x01)&&(! (temp&(1<<16))));//等待时间到达
    SysTick->CTRL=0x00; //关闭计数器
    SysTick->VAL =0X00; //清空计数器
}
```

4.13.7 STM32 系统时钟配置 SystemInit()

每个工程都必须在开始时配置并启动 STM32 系统时钟。

4.13.8 GPIO 引脚时钟使能

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE); //设置串口 1 时钟
使能
RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE); //功能复用 IO 时钟
使能
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE); //PC 端口时钟使能
```

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE); //ADC1 模拟量转换时  
钟使能  
RCC_ADCCLKConfig(RCC_PCLK2_Div6); //12M 最大 14M  
  
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE); //DMA1 功能模块时钟使  
能  
}
```

本节实验用到了 PA 和 PC 端口，所以要把 PA 和 PC 端口的时钟打开；串口 1 时钟打开；因为要与外部芯片通讯，所以要打开功能复用时钟；ADC 模拟量转换时钟打开，DMA 功能模块时钟打开。

4.13.9 GPIO 管脚电平控制函数

在主程序中采用 while(1) 循环语句，采用查询的方式等待 ADC 模拟量转换完毕，初始化完成以后要在主程序中采集模拟量，加入滤波、取平均值等措施然后转换送出打印。下面是 while(1) 语句中详细的内容。

```
while(1)  
{  
    da=0; //临时中间变量，先清空。  
  
    for(i=0;i<50;) //从 0V 开始输出，一共采样 50 次，每次都有电压变化  
    {  
        ad=ad+ADCDATA[i]; //把 50 次的采样值全部相加  
    }  
    ad=ad/50; //取 50 次的采样值的平均值  
    printf("ad =%f\r\n", 3.3/4095*ad); //实际电压值  
  
    delay_ms(1000);  
    delay_ms(1000);  
    delay_ms(1000);  
    delay_ms(1000);  
    delay_ms(1000);  
}  
}
```

4.13.10 stm32f10x_it.c 文件里的内容是

在中断处理 stm32f10x_it.c 文件里中串口 1 子函数非空，进入中断处理函

数后，先打开串口 1，和外部设备联络好，让后通过 CAN 通讯子函数把数据发送到总线上。

```
#include "stm32f10x_it.h"
#include "stm32f10x_exti.h"
#include "stm32f10x_rcc.h"
#include "misc.h"
#include "pbdata.h"

void NMI_Handler(void)
{
}

void USART1_IRQHandler(void)
{
    if(USART_GetITStatus(USART1, USART_IT_RXNE) !=RESET)
    {
        USART_SendData(USART1, USART_ReceiveData(USART1));
        while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)==RESET);
    }
}
```

4.13.11 main.c 文件里的内容是

大家都知道 `prinif` 重定向是把需要显示的数据打印到显示器上。在这个试验中 ADC 模拟量采集来的数据通过 DMA 模块送到内存的数组中，接着要把从外部得到的模拟量转换成数字信号，通过 `prinif` 重定向打印到串口精灵上；再采用万用表测量，和显示器上的数据做对比，检查多大的误差。

```
#include "pbdata.h"

void RCC_Configuration(void);
void GPIO_Configuration(void);
void NVIC_Configuration(void);
void USART_Configuration(void);
void ADC_Configuration(void);
void DMA_Configuration(void); //声明 DMA 函数

u16 ADCData[50]; //定义一个 16 位数组
```

```
int fputc(int ch, FILE *f)//打印到显示器函数
{
    USART_SendData(USART1, (u8)ch);
    while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)==RESET);
    return ch;
}

int main(void)
{
    U32 ad=0;
    u8 i=0;
    RCC_Configuration(); //系统时钟初始化
    GPIO_Configuration(); //端口初始化
    USART_Configuration();
    NVIC_Configuration();
    ADC_Configuration(); //调用 ADC 配置
    DMC_Configuration(); //调用 DMA 功能配置

    while(1)
    {
        ad=0;
        for(i=0;i<50;i++)
        {
            ad=ad+ADCData[i];
        }

        ad=ad/50;

        printf("ad =%f\r\n", 3.3/4095*ad); //实际电压值

        delay_ms(1000);
        delay_ms(1000);
        delay_ms(1000);
        delay_ms(1000);
        delay_ms(1000);
    }
}

void RCC_Configuration(void)
{
    SystemInit(); //72m
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
}
```

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
RCC_ADCCLKConfig(RCC_PCLK2_Div6); //12M 最大 14M

RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
}

void GPIO_Configuration(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    //LED
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_9;//TX
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_10;//RX
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_0;//AD
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AIN;
    GPIO_Init(GPIOC, &GPIO_InitStructure);
}

void NVIC_Configuration(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;

    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);

    NVIC_InitStructure.NVIC IRQChannel = USART1_IRQn;
    NVIC_InitStructure.NVIC IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC IRQChannelSubPriority = 1;
    NVIC_InitStructure.NVIC IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}

void USART_Configuration(void)
{
    USART_InitTypeDef USART_InitStructure;

    USART_InitStructureUSART_BaudRate=9600;
```

```
USART_InitStructure. USART_WordLength=USART_WordLength_8b;
USART_InitStructure. USART_StopBits=USART_StopBits_1;
USART_InitStructure. USART_Parity=USART_Parity_No;
USART_InitStructure. USART_HardwareFlowControl=USART_HardwareFlowControl_None;
USART_InitStructure. USART_Mode=USART_Mode_Rx|USART_Mode_Tx;

USART_Init(USART1, &USART_InitStructure);
USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
USART_Cmd(USART1, ENABLE);
USART_ClearFlag(USART1, USART_FLAG_TC);
}

void ADC_Configuration(void)
{
    ADC_InitTypeDef ADC_InitStructure;

    ADC_InitStructure. ADC_Mode=ADC_Mode_Independent;
    ADC_InitStructure. ADC_ScanConvMode=DISABLE;
    ADC_InitStructure. ADC_ContinuousConvMode=ENABLE;//循环采集方式
    ADC_InitStructure. ADC_ExternalTrigConv=ADC_ExternalTrigConv_None;
    ADC_InitStructure. ADC_DataAlign=ADC_DataAlign_Right;
    ADC_InitStructure. ADC_NbrOfChannel=1;

    ADC_Init(ADC1, &ADC_InitStructure);

    ADC-RegularChannelConfig(ADC1, ADC_Channel_10, 1, ADC_SampleTime_239Cycles5);

    ADC_DMACmd(ADC1, ENABLE); //重要，有这条语句才能把 ADC 采集和 DMA 功能模块捆绑在一起。

    ADC_Cmd(ADC1, ENABLE); //ADC 外设使能

    ADC_ResetCalibration(ADC1);
    while(ADC_GetResetCalibrationStatus(ADC1));

    ADC_StartCalibration(ADC1);
    while(ADC_GetCalibrationStatus(ADC1));

    ADC_SoftwareStartConvCmd(ADC1, ENABLE);
}

void DMA_Configuration(void)
```

```

{
    DMA_InitTypeDef DMA_InitStructure;//创建结构体
    DMA_DeInit(DMA1_Channel1); //DMA1 通道 1 初始化，因为在 DMA 通道变量表中
ADC1 必须走通道 1

    DMA_InitStructure.DMA_PeripheralBaseAddr=(u32)&ADC1->DR;//定义 DMA 的
外设基础地址，并把地址数据转换成 32 位地址数据，&是地址符号，把地址送到
“ADC1->DR” 寄存器中。
    DMA_InitStructure.DMA_MemoryBaseAddr=(u32)ADCDATA;//定义 DMA 的内存地
址，并把内存地址数据转换成 32 位内存地址数据，数组地址

    DMA_InitStructure.DMA_DIR=DMA_DIR_PeripheralSRC;//外设作为数据的来源
    DMA_InitStructure.DMA_BufferSize=50;//设置缓冲区大小，应该和数组的值一
样大，注意这点

    DMA_InitStructure.DMA_PeripheralInc=DMA_PeripheralInc_Disable;//外设
地址不变
    DMA_InitStructure.DMA_MemoryInc=DMA_MemoryInc_Enable;//内存地址是递增
的
    //外设和内存的数据宽度要设置成一样宽度
    DMA_InitStructure.DMA_PeripheralDataSize=DMA_PeripheralDataSize_HalfW
ord;//设置外设的数据宽度，设置为 16 位
    DMA_InitStructure.DMA_MemoryDataSize=DMA_MemoryDataSize_HalfWord;//设
置内存的数据宽度，设置为 16 位

    DMA_InitStructure.DMA_Mode=DMA_Mode_Circular;//工作模式时连续不断的
    DMA_InitStructure.DMA_Priority=DMA_Priority_High;//优先级
    DMA_InitStructure.DMA_M2M=DMA_M2M_Disable;//设置成非内存到内存

    DMA_Init(DMA1_Channel1, &DMA_InitStructure); //参数传递，初始化
    DMA_Cmd(DMA1_Channel1, ENABLE); //启动外设
}

```

ADC_DMACmd(ADC1, ENABLE); //重要，用这条语句才能把 ADC 采集和 DMA 功能模块捆绑在一起。

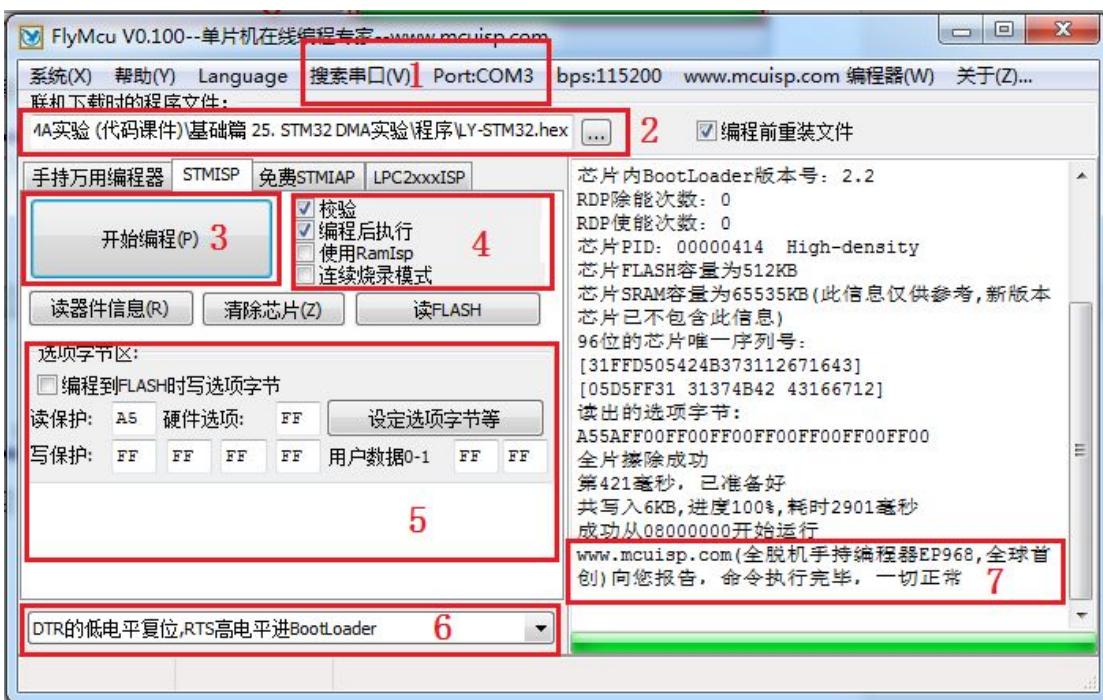
表 1 函数 DMA_Init

函数名称	DMA_Init
------	----------

函数原型	Void DMA_Init(DMA_Channel1_TypeDef*DMA_Channelx, DMA_InitTypeDef*DMA_InitStruct)
功能描述	根据 DMA_InitStruct 中指定的参数初始化外设 DMA 的通道 x 寄存器
输入参数 1	DMA_Channelx: x 可以是 1, 2... 或者 7 来选择 DMA 通道 x
输入参数 2	DMA_InitStruct: 指向结构 DMA_InitTypeDef 的指针, 包含了 DMA 通道 x 的配置信息 参阅: DMA_InitTypeDef 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

4.13.12 程序下载

请根据下图所指向的 7 个重点区域配置。其中（1）号区域根据自己机器的实际情况选择，我的机器虚拟出来的串口号是 COM2。（2）号区域请自己选择程序所在的文件夹。（7）号区域当程序下载完后，进度条会到达最右边，并且提示一切正常。（4、5、6）号区域一定要按照上图显示的设置。当都设置好以后就可以直接点击（3）号区域的开始编程按钮下传程序了。



本节实验的源代码在光盘中：(LY-STM32 光盘资料\1. 课程\1. 基础篇\基础

篇 25. STM32 DMA 实验\程序)

4.13.13 实验效果图

程序写入实验板后，使用公司开发的多功能监视系统，在串口调试界面中的接收区就会接收到程序定时发送过来的数据。和 ADC 数据采集程序相差无几。具体的实验效果参考下图，红色框图 1 区域是在调节精密电位器，使输出改变，在红色框图 2 区域的监视区可以看到模拟电压值在不停的變化，符合程序设计要求。

