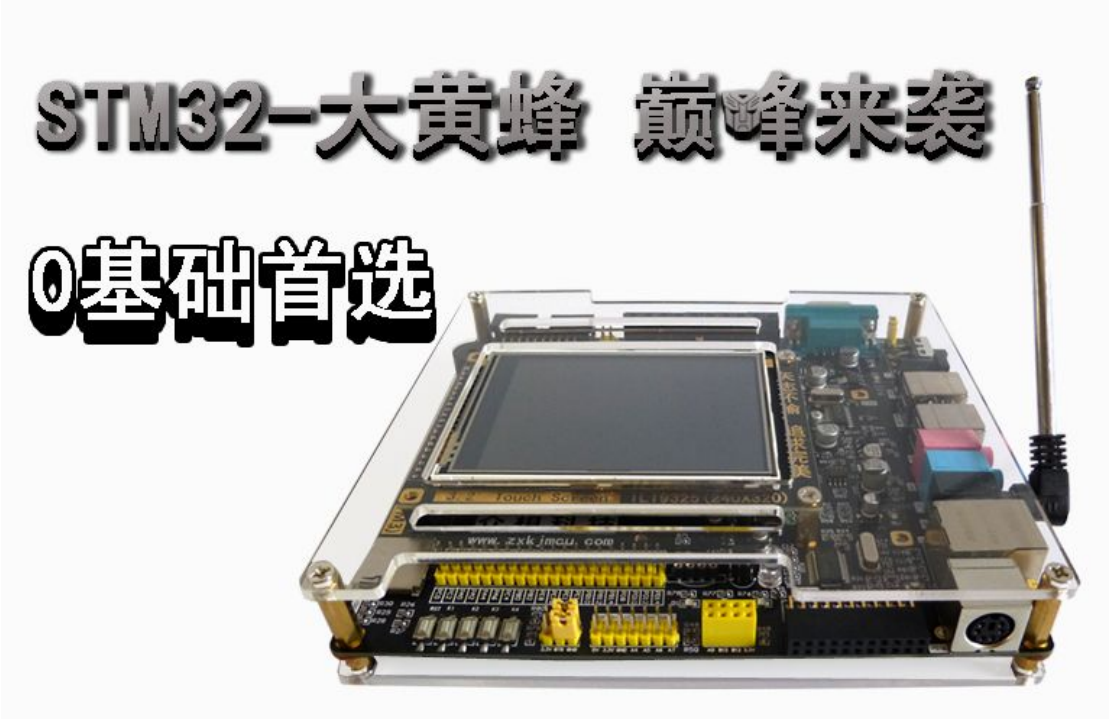
学 ARM 从 STM32 开始

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

4.12 STM32 DAC 工作原理及实验

4.12.1 概述

4.12.1.1 DAC 概念

ADC 就是数字量输入转换成模拟量输出。

随着现代科学技术的迅猛发展,特别是数字系统已广泛应用于各种学科领域及日常生活,微型计算机就是一个典型的数学系统。但是数字系统只能对输入的数字信号进行处理,其输出信号也是数字信号。而在工业检测控制和生活中的许多物理量都是连续变化的模拟量,如温度、压力、流量、速度等,这些模拟量可以通过传感器或换能器变成与之对应的电压、电流或频率等电模拟量。为了实现数字系统对这些电模拟量进行检测、运算和控制,就需要一个模拟量与数字量之间的相互转换的过程。即常常需要将数字量转换成模拟量,简称 DAC 转换,完成这种转换的电路称为数模转换器(DigitaltoAnalogConverter),简称 DAC。

4.12.1.2 STM32 DAC 模拟量输入功能

1、STM32 自带 DAC 转化功能，STM32 DAC 是 12 位数字输入，电压输出的数字/模拟转换器。DAC 可以配置为 8 位或者 12 位模式，也可以与 DMA 控制器配合使用。DAC 工作在 12 位模式时，数据可以设置成左对齐或者右对齐。DAC 模块有两个输出通道，每个通道都有单独的转化器。在双 DAC 模式下，两个通道可以独立的转换，也可以同时进行转换并同步更新两个通道的输出。DAC 可以通过引脚输入参考电压 VREF+以获得更精确的转换结果。

2、STM32 DAC 主要特性

- 2 个 DAC 转换器：每个转换器对应 1 个输出通道
- 12 位模式下数据左对齐或者右对齐
- 双 DAC 通道同时或者分别转换
- 每个通道都有 DMA 功能
- 外部触发转换
- 输入参考电压 VREF+

3、DAC 模块管脚说明

DAC 模块管脚说明

名称	信号类型	注释
Vref+	输入，模拟参考正极	DAC 使用的高端/正极参考电压，2.4V<VREF<VDDA(3.3V)
Vdda	输入，模拟电源	模拟电源
Vssa	输入，模拟电源地	模拟电源的地线
DAC_OUTx	模拟输出信号	16 个模拟输入通道

注意：一旦使能 DACx 通道，相应的 GPIO 引脚（PA4 或者 PA5）就自动与 DAC 的模拟输出相连（DAC_OUTx）。为了避免寄生的干扰和额外的功耗，引脚 PA4 或者 PA5 在之前应当设置成模拟输入（AIN）。

4.12.1.3 STM32 DAC 通道使能

1、DAC 通道使能

将 DAC_CR 寄存器的 ENx 位置‘1’即可以打开对 DAC 通道 x 的供电。经过一段启动时间 tWAKEUP，DAC 通道 x 即被使能。

注意：ENx 位只会使能 DAC 通道 x 的模拟部分。即使该为被置‘0’，DAC 通道 x 的数字部分仍然工作。

2、使能 DAC 输出缓存

DAC 集成了 2 个输出缓存，可以用来减少输出阻抗，无需外部运放即可直接驱动外部负载。每个 DAC 通道输出缓存可以通过设置 DAC_CR 寄存器的 BOFFx

位使能或者关闭。

3、输出电压

数字输入经过 DAC 被线性地转换为模拟量电压输出，其范围为 0 到 VREF+。任意 DAC 通道引脚上的输出电压满足下面的关系：

DAC 输出=VREF*(DOR/4095)。

4.12.1.4 STM32 DAC 数据格式

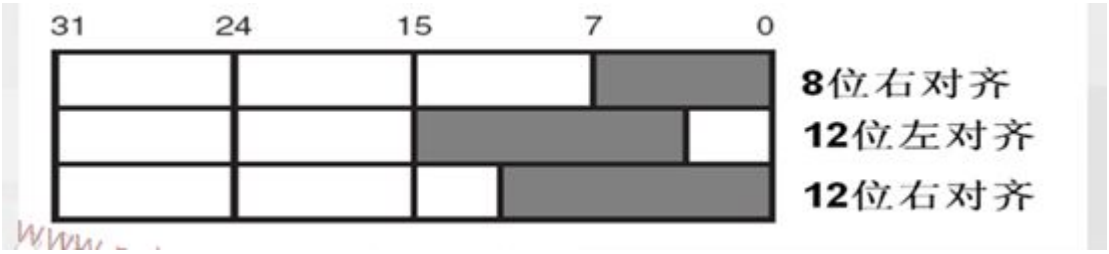
根据选择的配置模式，数据按照下文所述写入制定的寄存器：

——8 位数据右对齐：用户须将数据写入寄存器 DAC_DHR8Rx[7:0]位（实际是存入寄存器 DHRx[11:4]位）

——12 位数据左对齐：用户须将数据写入寄存器 DAC_DHR12Lx[15:4]位（实际是存入寄存器 DHRx[11:0]位）

——12 位数据右对齐：用户须将数据写入寄存器 DAC_DHR12Rx[11:0]位（实际是存入寄存器 DHRx[11:0]位）

根据对 DAC_DHRyyyx 寄存器的操作，经过相应的移位后，写入的数据被转存到 DHRx 寄存器中（DHRx 是内部的数据保存寄存器 x）。随后，DHRx 寄存器的内容或被自动地传送到 DORx 寄存器，或通过软件触发或外部事件触发被传送到 DORx 寄存器。

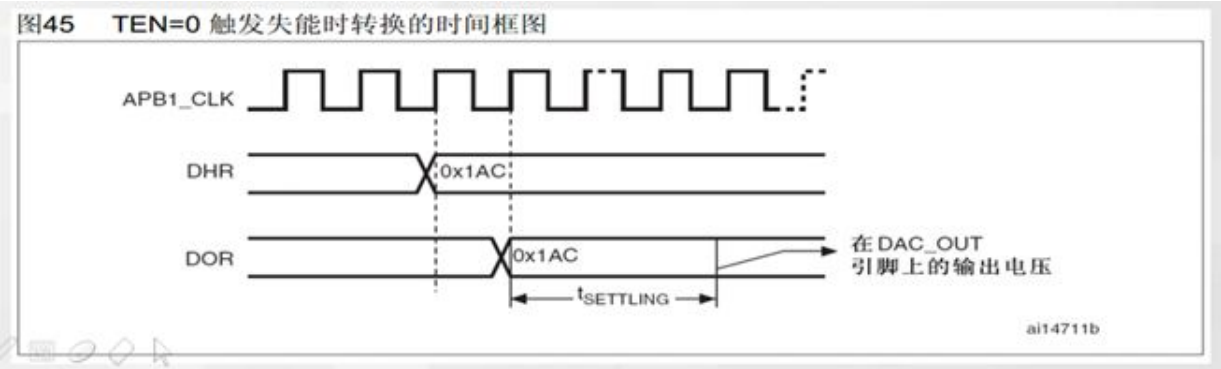


4.12.2 STM32 DAC 转换

不能直接对寄存器 DAC_DORx 写入数据，任何输出到 DAC 通道 x 的数据都必须写入 DAC_DHRx 寄存器（数据实际写入 DAC_DHR8Rx、DAC_DHR12Lx、DAC_DHR12Rx、DAC_DHR8RD、DAC_DHR12LD、或者 DAC_DHR12RD 寄存器）。

如果没有选中硬件触发（寄存器 DAC_CR1 的 TENx 位置‘0’），存入寄存器 DAC_DORx 的数据会在一个 APB1 时钟周期后自动传至寄存器 DAC_DORx。如果选中硬件触发（寄存器 DAC_CR1 的 TENx 位置‘1’），数据传输在触发发生以后 3 个 APB1 时钟周期后完成。一旦数据从 DAC_DHRx 寄存器，在经过时间 tSETTLING 之后，输出即有效，这段时间的长短依电源电压和模拟输出负载的不同会有所变化。

图45 TEN=0 触发失能时转换的时间框图



4.12.3 选择 DAC 触发

如果 TENx 位被置‘1’，DAC 转换可以由某外部事件触发（定时器计数器、外部中断线）。配置控制位 TSELx[]可以选择 8 个触发事件之一触发 DAC 转换。

DAC 外部触发条件列表

触发源	类型	TSELx[2:0]
定时器 6 TRGO 事件	来自片上定时器的内部信号	PA0
互联型产品为定时器 3 TROG 事件 或大容量产品为定时器 8 TROG 事件		PA1
定时器 7 TRGO 事件		PA2
定时器 5 TRGO 事件		PA3
定时器 2 TRGO 事件		PA4

定时器 4 TRGO 事件		PA5
EXTI 线路 9	外部引脚	PA6
SWTRIG(软件触发)	软件控制位	PA7

每次 DAC 接口侦测到来自选中的定时器 TROG 输出，或者外部中断线 9 的上升沿，最近存放在寄存器 DAC_DHRx 中的数据会被传送到寄存器 DAC_DORx 中。在 3 个 APB1 时钟周期后，寄存器 DAC_DORx 更新为新值。如果选择软件触发，一旦 SWTRIG 位置‘1’，转换即开始。在数据从 DAC_DHRx 寄存器传送到 DAC_DORx 寄存器后，SWTRIG 位由硬件自动清‘0’。

4.12.4 实验目的

因为数字信号由实验板上的主芯片 STM32（CPU）直接转换成模拟量输出到端子，所以我们采用万用表可以很轻松的测量出来输出变化。从这个实验中学习者要熟悉 DAC 转换方式和程序设计构架。

4.12.5 硬件设计

利用实验板上的 DAC 模拟量转换电路，可以很方便的实现这个功能。

4.12.6 DAC 模拟量转换软件设计

和 ADC 程序设计一样，软件设计在 prinif 重定向应用程序上修改。下面对软件的设计做一次说明：

1、打开 DAC 时钟；

2、要对主频进行分频；

3、配置 DAC 相关设置参数；

4.12.6.1 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
```

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，tm32f10x_tim.c 库函数主要包含定时器设置，tm32f10x_usart.c 库函数主要包含串行通讯设置，tm32f10x_dac.c 库函数主要包含 DAC 模拟量转换设置，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。

4.12.6.2 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。

4.12.6.3 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 "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。

4.12.6.4 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
}
```

4.12.7 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。

4.12.8 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_APB1Periph_DAC,ENABLE);//DAC 模拟量转换时钟
使能
```

本节实验用到了 PA 端口，所以要把 PA 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；DAC 模拟量时钟打开。

4.12.9 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<=10;i++)    //从 0V 开始输出，一共输出 10 次，每次都有电压变化
    {
      da=i*400;//数字量最大裸数据是 4095，当 i=10 时，裸数据时 4000，不会超出
```

```
最大裸数据，符合要求

        DAC_SetChannel1Data(DAC_Align_12b_R,da);//12 位 右对齐 PA4 端口输出

        printf("da=%f v\r\n",3.3*(da/4095));//转换成浮点打印到显示器，输出真实
的电压值
        //延时 5 秒，就是每隔 5 秒输出一次，一共输出 10 次
        delay_ms(1000);
        delay_ms(1000);
        delay_ms(1000);
        delay_ms(1000);
        delay_ms(1000);
    }
  }
```

采用查询方式设计程序，就不需要中断处理函数。

4.12.10 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 DAC 模拟量输入程序，是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 DAC_Configuration(void);

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)
{
```

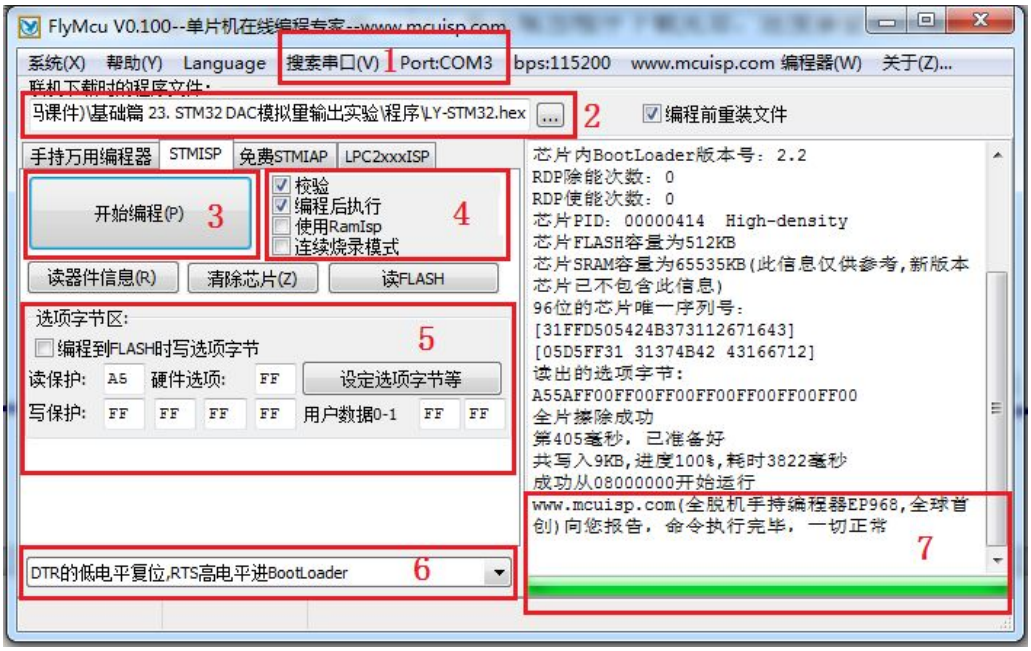
```
    u8 i=0;
    float da=0;

    RCC_Configuration(); //系统时钟初始化
    GPIO_Configuration();//端口初始化
    USART_Configuration();
    NVIC_Configuration();
    DAC_Configuration();//调用 DAC 配置

    while(1)
    {
      da=0;
     for(i=0;i<=10;i++)
     {
       da=i*400;
       DAC_SetChannel1Data(DAC_Align_12b_R,da);//12 位 右对齐 PA4 端口输
出
       printf("da=%f v\r\n",3.3*(da/4095));

       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_APB1PeriphClockCmd(RCC_APB1Periph_DAC,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_4;//TX
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AIN;
    GPIO_Init(GPIOA,&GPIO_InitStructure);
    GPIO_SetBits(GPIOA,GPIO_Pin_4);//输出高
}

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_InitStructure.USART_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_HardwareFlowContr
ol_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 DAC_Configuration(void)
    {
       DAC_InitTypeDef DAC_InitStructure;//结构体

       DAC_InitStructure.DAC_Trigger=DAC_Trigger_None;//不使用触发功能
       DAC_InitStructure.DAC_WaveGeneration=DAC_WaveGeneration_None;//不使用
三角波
       //屏蔽 幅值设置

DAC_InitStructure.DAC_LFSRUnmask_TriangleAmplitude=DAC_LFSRUnmask_Bit0;

       //关闭缓存
       DAC_InitStructure.DAC_OutputBuffer=DAC_OutputBuffer_Disable;

       DAC_Init(DAC_Channel_1,&DAC_InitStructure);//初始化 DAC 通道 1
       DAC_Cmd(DAC_Channel_1,ENABLE);//使能 DAC1
       DAC_SetChannel1Data(DAC_Align_12b_R,0);//12 位 右对齐，给它一个初始数据
0
    }
```

4.12.11 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 COM3。（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。

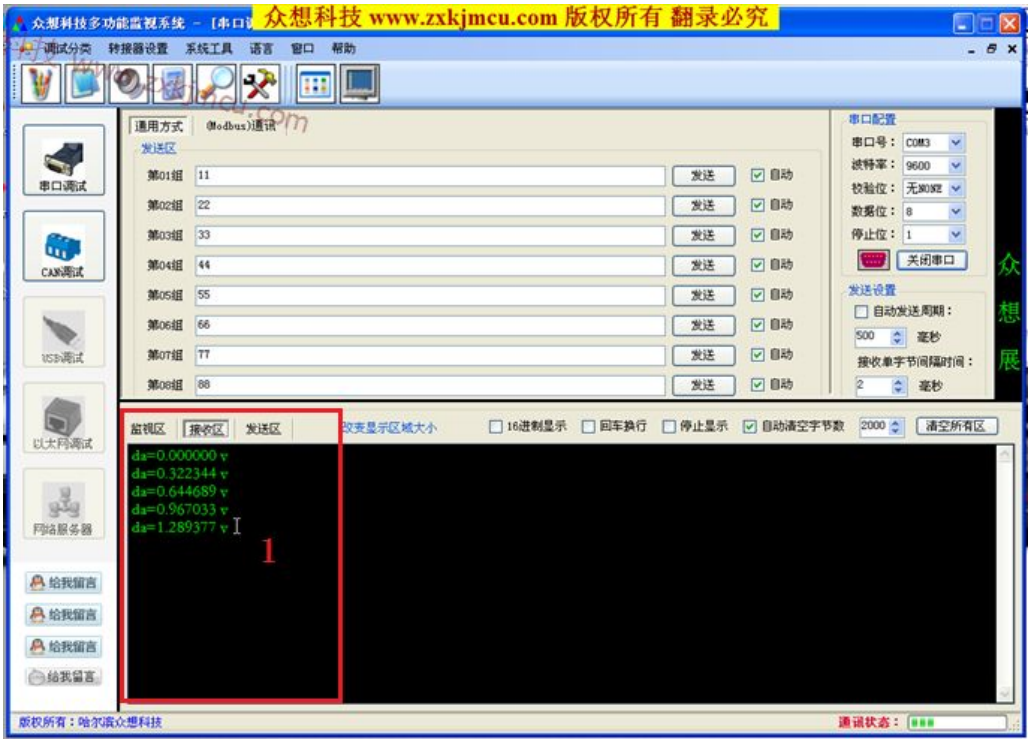


本节实验的源代码在光盘中：（LY-STM32 光盘资料\1.课程\1,基础篇\基础篇 23. STM32 DAC 模拟量输出实验\程序）

4.12.12 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，在接收区可以看到 CPU 发送出来的模拟量电压值，按照程序设计在不停的增长。使用万用表对地实际测量大黄蜂实验板 DAC 输出端子的电压值，基本上和接收区显示的数据很贴近，说明程序设计成功，输出正常。

这个程序设计和串口通讯涉程序相差无几，串口程序掌握了这个就很快掌握。


众想科技 www.zxkjmcu.com 版权所有 翻录必究
众想科技多功能监视系统
调试分类
转接器设置
系统工具
语言
窗口
帮助
通用方式
(Modbus)通讯
发送区
第01组 11
第02组 22
第03组 33
第04组 44
第05组 55
第06组 66
第07组 77
第08组 88
发送
自动
串口配置
串口号： COM3
波特率： 9600
校验位： 无校验
数据位： 8
停止位： 1
关闭串口
发送设置
自动发送周期：
500
毫秒
接收单字节间隔时间：
2
毫秒
众想展
串口调试
CAN调试
USB调试
以太网调试
网络服务器
给我留言
给我留言
给我留言
给我留言
监视区
接收区
发送区
改变显示区域大小
16进制显示
回车换行
停止显示
自动清空字节数
2000
清空所有区
da=0.000000 v
da=0.322344 v
da=0.644689 v
da=0.967033 v
da=1.289377 v
1
版权所有：哈尔滨众想科技
通讯状态：