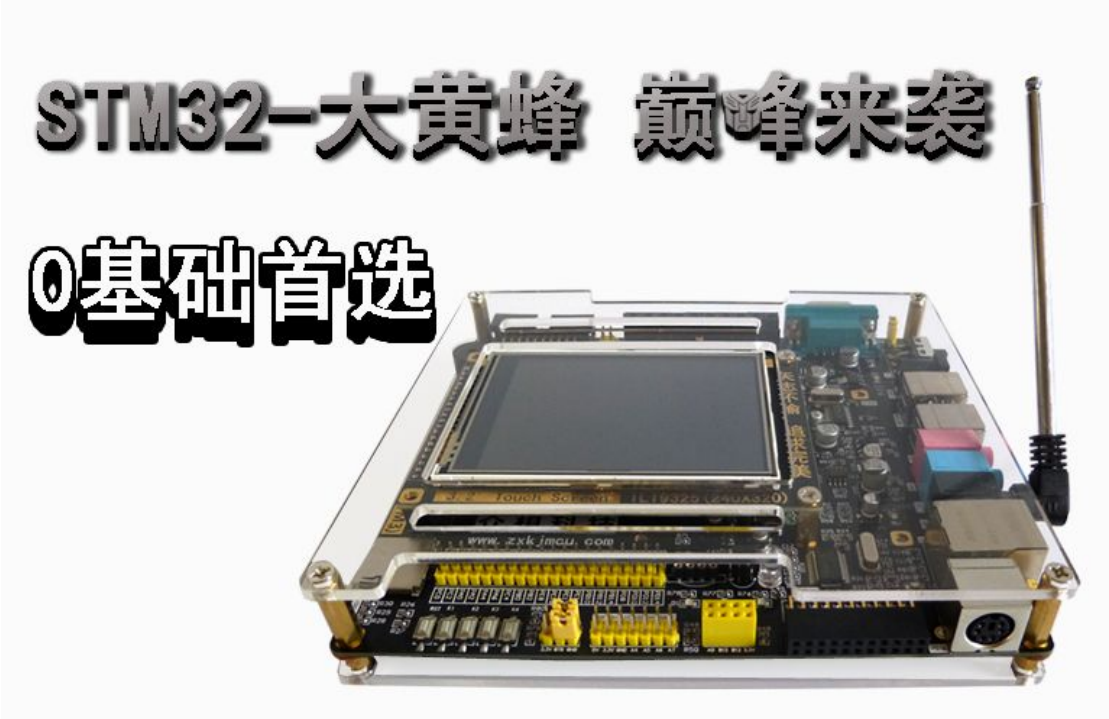
学 ARM 从 STM32 开始

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

4.11 STM32 ADC 工作原理 实验

4.11.1 概述

4.11.1.1 ADC 概念

ADC 就是模拟量输入转换成数字量。

我们先简单介绍一下逐次比较型 A/D，逐次比较型 A/D 包括 n 位逐次比较型 A/D 转换器如图 1 所示。它由控制逻辑电路、时序产生器、移位寄存器、D/A 转换器及电压比较器组成。

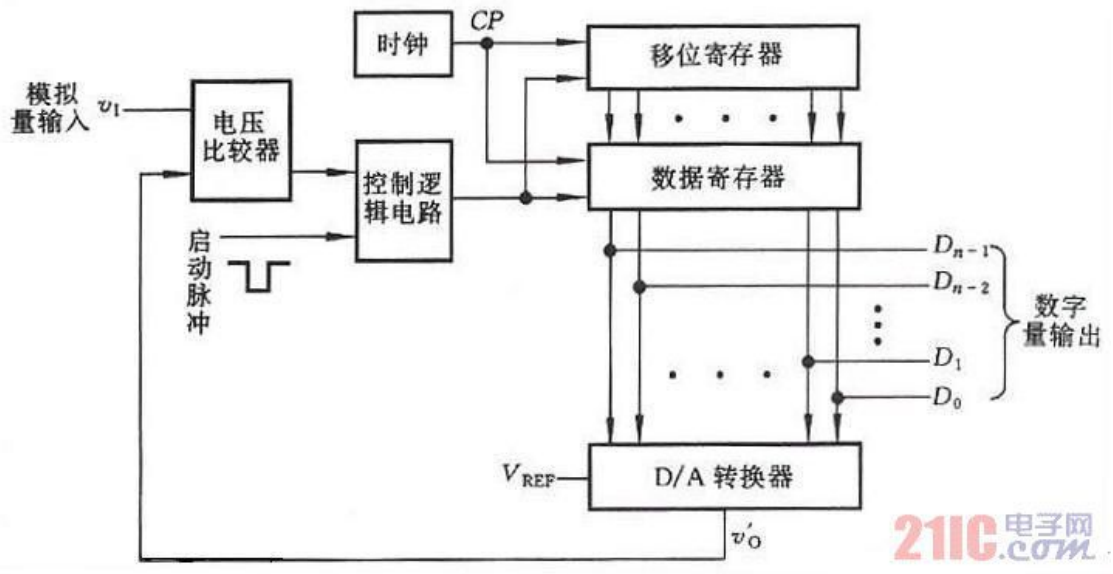


图 1 逐次比较型 A/D 转换器框图

4.11.1.2 ADC 工作原理

逐次逼近转换过程和用天平称物重非常相似。天平称重物过程是，从最重的砝码开始试放，与被称物体行进比较，若物体重于砝码，则该砝码保留，否则移去。再加上第二个次重砝码，由物体的重量是否大于砝码的重量决定第二个砝码是留下还是移去。照此一直加到最小一个砝码为止。将所有留下的砝码重量相加，就得此物体的重量。仿照这一思路，逐次比较型 A/D 转换

器，就是将输入模拟信号与不同的参考电压作多次比较，使转换所得的数字量在数值上逐次逼近输入模拟量对应值。

图 1 的电路，它由启动脉冲启动后，在第一个时钟脉冲作用下，控制电路使时序产生器的最高位置 1,其他位置 0,其输出经数据寄存器将 1000……0，送入 D/A 转换器。输入电压首先与 D/A 器输出电压(VREF/2)相比较，如 v1≥VREF/2，比较器输出为 1，若 vI< VREF/2，则为 0。比较结果存于数据寄存器的 Dn-1 位。然后在第二个 CP 作用下，移位寄存器的次高位置 1，其他低位置 0。如最高位已存 1,则此时 v0=(3/4)VREF。于是 v1 再与(3/4)VREF 相比较，如 v1≥(3/4)VREF，则次高位 Dn-2 存 1，否则 Dn-2=0;如最高位为 0,则 v0=VREF/4,与 v0 比较,如 v1≥VREF/4,则 Dn-2 位存 1,否则存 0……。以此类推，逐次比较得到输出数字量。

为了进一步理解逐次比较 A/D 转换器的工作原理及转换过程。下面用实例加以说明。

设图 1 电路为 8 位 A/D 转换器，输入模拟量 vA=6.84V，D/A 转换器基准电压 VREF=10V。 根据逐次比较 D/A 转换器的工作原理，可画出在转换过程中 CP、启动脉冲、D7～D0 及 D/A 转换器输出电压 v0 的波形，如图 11.10.2 所示。

由图.2 可见，当启动脉冲低电平到来后转换开始，在第一个 CP 作用下，数据寄存器将 D7～D0=10000000 送入 D/A 转换器，其输出电压 v0=5V，vA 与 v0 比较，vA>v0 存 1;第二个 CP 到来时，寄存器输出 D7～D0=11000000，v0 为 7.5V，vA 再与 7.5V 比较，因 vA<7.5V，所以 D6 存 0;输入第三个 CP 时，D7～D0=10100000，v0=6.25V;vA 再与 v0 比较，……如此重复比较下去，

经 8 个时钟周期，转换结束。由图中 v0 的波形可见，在逐次比较过程中，与输出数字量对应的模拟电压 v0 逐渐逼近 vA 值，最后得到 A/D 转换器转换结果 D7～D0 为 10101111。该数字量所对应的模拟电压为 6.8359375V，与实际输入的模拟电压 6.84V 的相对误差仅为 0.06%。

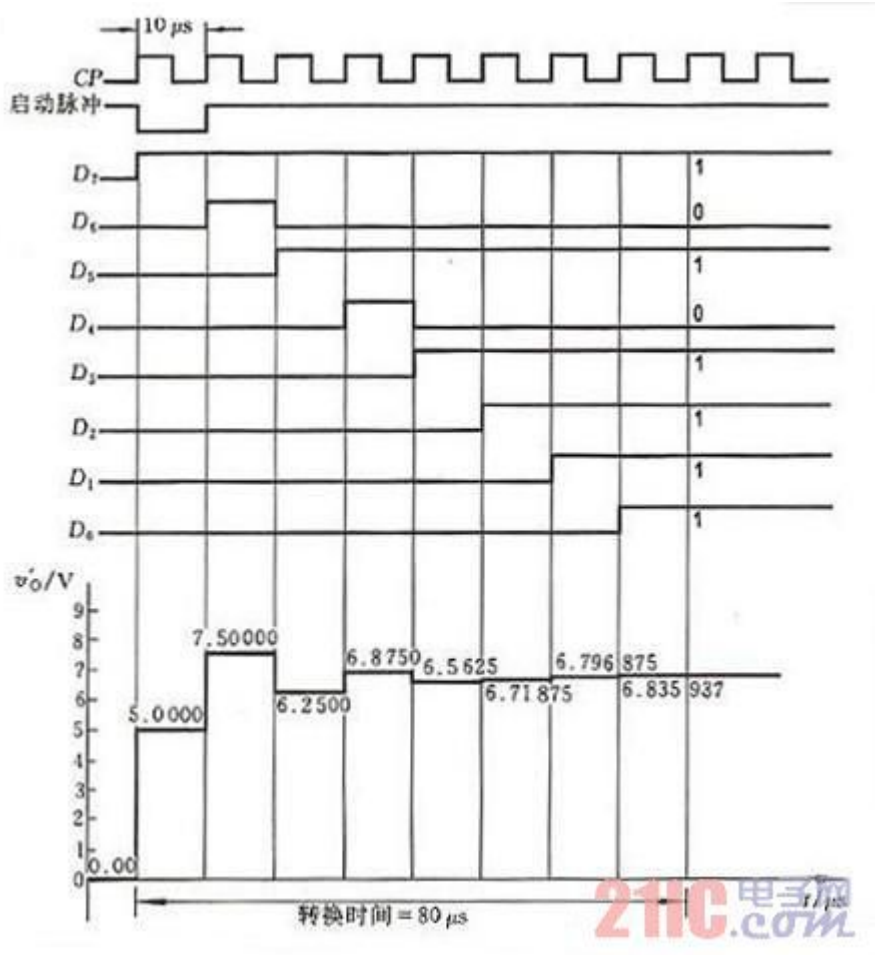


图 2　8 位逐次比较型 A/D 转换器波形图

4.11.1.3 STM32 ADC 模拟量输入功能

1、STM32 单片机自带 ADC 转换，STM32 ADC 是 12 位逐次逼近型模拟数字转换器。它有多达 18 个通道，可测量 16 个外部和 2 个内部信号源。各通道的 A/D 转换可以单次、连续、扫描或者间断模式执行。ADC 的结果可以左对齐或者右对齐方式存储在 16 位数据寄存器中。模拟看门狗特性允许应用程序

检测输入电压是否超出用户定义的高/低阀值。**ADC 的输入时钟不得超过 14MHz，它是由 PCLK2 经分频产生。**

2、STM32 ADC 主要特性

- 12 位分辨率
- 转换结束、注入转换结束和发生模拟看门狗事件时产生中断
- 单次和连续转换模式
- 从通道 0 到通道 n 的自动扫描模式
- 间断模式执行
- 自校准
- 带内嵌数据一致性的数据对齐
- 采样间隔可以按通道分别编成
- 规律转换和注入转换均有外部触发选项
- 双重模式（带 2 个或者以上 ADC 的器件）

3、ADC 转换时间

- STM32F103xx 增强型：时钟为 56MHz 时，转换时间为 1μs（时钟 72MHz 时为 1.17μs）；
- STM32F101xx 基本型：时钟为 28MHz 时，转换时间为 1μs（时钟 36MHz 时为 1.55μs）；
- STM32F102xxUSB 型：时钟为 48MHz 时，转换时间为 1.2μs；
- STM32F105xx 和 STM32F107xx 型：时钟为 56MHz 时，转换时间为 1μs（时钟 72MHz 时为 1.17μs）；
- ADC 供电要求：2.4V~3.6V

- ADC 输入范围：VREF-<=VIN<=VREF+；
- 规则通道转换期间有 DMA 请求产生；

4、ADC 引脚说明

ADC 模拟量输入说明

名称	信号类型	注释
Vref+	输入，模拟参考正极	ADC 使用的高端/正极参考电压，2.4V<VREF<VDOA
Vdao	输入，模拟电源	
Vref-	输入，模拟参考负极	
Vssa	输入，模拟电源地	等效于 VSS 的模拟电源地
ADCx_IN[15：0]	模拟输入信号	16 个模拟输入通道

ADC 模拟量输入通道列表

	ADC1	ADC2	ADC3
通道 0	PA0	PA0	PA0
通道 1	PA1	PA1	PA1
通道 2	PA2	PA2	PA2
通道 3	PA3	PA3	PA3
通道 4	PA4	PA4	
通道 5	PA5	PA5	
通道 6	PA6	PA6	
通道 7	PA7	PA7	
通道 8	PB0	PB0	
通道 9	PB1	PB1	
通道 10	PC0	PC0	PC0
通道 11	PC1	PC1	PC1
通道 12	PC2	PC2	PC2
通道 13	PC3	PC3	PC3
通道 14	PC4	PC4	
通道 15	PC5	PC5	
通道 16	温度传感器		
通道 17	内部参考电压		

4.11.1.4 STM 32 ADC 通道选择说明

有 16 个多路通道，可以把转换组织成两组：规则组和注入组。在任意多个通道上以任意顺序进行的一系列转换构成成组转换。

- 规则组由多达 16 个转换组成。规则通道和它们的转换顺序在 ADC_SQRx 寄存器中选择。规则组中转换的总数应写入 ADC_SQR1 寄存器的 L[3:0]

位中。

- 注入组由 4 个转换组成。注入通道和它们的转换顺序在 ADC_JSQR 寄存器中选择，注入组里的转换总数应写入 ADC_JSQR 寄存器的 L[1:0]位中。

如果 ADC_SQRx 或 ADC_JSQR 寄存器在转换期间被更改，当前的转换被清 0，一个新的启动脉冲将发送到 ADC 以转换新选择的组。

温度传感器和通道 ADC1_IN16 相连接，内部参照电压 VREFINT 和 ADC1_IN17 相连接。可以按照注入或者规则通道对这两个内部通道进行转换。

注意：温度传感器和 VREFINT 只能出现在主 ADC1 中。

4.11.1.5 STM 32 ADC 转换模式

1、单次转换模式

单次转换模式下，ADC 只执行一次转换。该模式即可通过设置 ADC_CR2 寄存器的 ADON 位（仅仅适用于规则通道）启动，也可以通过外部触发启动（适用于规则通道或注入通道），这时 CONT 位为 0。

一旦选择通道的转换完成：

- 如果一个规则通道被转换：

——转换数据被储存在 16 位 ADC_DR 寄存器中

——EOC（转换结束）标志被设置

——如果设置了 EOCIE，则产生中断。

- 如果一个注入通道被转换：

——转换数据被储存在 16 位 ADC_DRJ1 寄存器中

——JEOC（转换结束）标志被设置

——如果设置了 EOCIE，则产生中断，然后 ADC 停止。

2、连续转换模式

连续转换模式下，当前面的 ADC 转换一结束就马上启动另一次转换。此模式可以通过外部触发启动或者设置 ADC_CR2 寄存器上的 ADON 位启动，这时 CONT 位为 1。

每次转换完成后：

- 如果一个规则通道被转换：

  ——转换数据被储存在 16 位 ADC_DR 寄存器中

  ——EOC（转换结束）标志被设置

  ——如果设置了 EOCIE，则产生中断。

- 如果一个注入通道被转换：

  ——转换数据被储存在 16 位 ADC_DRJ1 寄存器中

  ——JEOC（转换结束）标志被设置

  ——如果设置了 JEOCIE 位，则产生中断。

3、扫描模式

扫描模式可通过设置 ADC_CR1 寄存器的 SCAN 位来选择。一旦这个位被设置，ADC 扫描所有被 ADC_SQRX 寄存器（）或者 ADC_JSQR()选中的所有通道。在每个组的每个通道上执行单次转换。在每个转换结束时，同一组的下一个通道被自动转换。如果设置了 CONT 位，转换不会再选择组的最后一个通道上停止，而是再次从选择组的第一个通道继续转换。

如果设置了 DMA 位，在没次 ECO 后，DMA 控制器把规则组通道的转换数据传输到 SRAM 中。而注入通道转换的数据总是存储在 ADC_JDRx 寄存器中。

4、间断模式

- 规则组

此模式通过设置 ADC_CR1 寄存器上的 DISCEN 位激活。它可以用来执行一个短序列的 n 次转换（n<=8），此次转换是 ADC_SQRx 寄存器所选择的转换序列的一部分。数值 n 由 ADC_CR1 寄存器的 DISCNUM[2:0]位给出。

一个外部触发信号可以启动 ADC_SQRx 寄存器中描述的下一轮 n 次转换，直到此序列所有的转换完成为止。总的序列长度由 ADC_SQR1 寄存器的 L[3:0]定义。

举例：

n=3，被转换的通道=0、1、2、3、6、7、9、10

第一次触发：转换的序列位 0、1、2

第二次触发：转换的序列位 3、6、7

第三次触发：转换的序列位 9、10，并产生 EOC 事件

第四次触发：转换的序列位 0、1、2

注意：当以间断模式转换一个规则组时，转换序列结束后不自动从头开始。当所有子组转换完成后，下一次触发启动第一个子组的转换。在上面的例子中，第四次触发重新转换第一组的 0、1、2.

- 注入组

此模式通过设置 ADC_CR1 寄存器上的 JDISCEN 位激活。在一个外部触发事件后，该模式按通道顺序逐个转换 ADC_JSQR 寄存器中选择的序列。

一个外部触发信号可以启动 ADC_JSQR 寄存器选择的下一个通道序列的转换，直到序列中所有的转换完成为止。总的序列长度由 ADC_JSQR 寄存器的 JL[1:0]位定义。

举例：

n=1，被转换的通道=1、2、3

第一次触发：通道 1 被转换

第二次触发：通道 2 被转换

第三次触发：通道 3 被转换，并产生 EOC 和 JEOC 事件

第四次触发：通道 1 被转换

注意：

1、当完成所有注入通道转换，下一次触发启动第 1 个注入通道的转换。在上面的例子中，第四次触发重新转换第 1 个注入通道 1。

2、不能同时使用自动注入和间断模式。

3、必须避免同时为规则和注入组设置间断模式。间断模式只能作用于一组转换。

4.11.1.6 STM 32 ADC 时钟配置

Void RCC_ADCCLKConfing(uint32_t RCC_PCLK2) 分频函数。

输入参数范围：

```
#define RCC_PCLK2_Div2      ((uint32_t)0x00000000)
#define RCC_PCLK2_Div4      ((uint32_t)0x00004000)
#define RCC_PCLK2_Div6      ((uint32_t)0x00008000)
#define RCC_PCLK2_Div8      ((uint32_t)0x0000C000)
```

STM32 的 ADC 最大的转换速率时 1MHz，也就是转换时间为 1us（在 ADCCLK=14M,采样周期为 1.5 个 ADC 时钟下得到），不要让 ADC 的时钟超过 14M，否则将导致结果准确率下降。

4.11.1.7 ADC 的采样时间

可编程的通道采样时间

ADC 使用若干个 ADC_CLK 周期对输入电压采样，采样周期数目可以通过 ADC_SMPR1 和 ADC_SMPR2 寄存器中的 SMP[2:0]位更改。每个通道可以分别用不同的时间采样。

总转换时间如下计算：TCONV=采样时间+12.5 个周期

例如：当 ADCCLK=14MHz，采样时间为 1.5 周期

TCON=1.5+12.5=14 周期=1us

常常使用的周期：

1.5 周期、7.5 周期、13.5 周期、28.5 周期、41.5 周期、55.5 周期、71.5 周期、239.5 周期。

4.11.1.8 ADC 的数据对齐

ADC_CR2 寄存器中的 ALIGN 位选择转换后数据存储的对齐方式。数据可以左对齐或右对齐，如图 29 和图 30 所示。

注入组通道转换的数据值已经减去了在 ADC_JOFRx 寄存器中定义的偏移量，因此结果可以是一个负值。SEXT 位是扩展的符号值。

对于规则组通道，不需要减去偏移值，因此只有 12 个位有效。

图29 数据右对齐

注入组

SEXT	SEXT	SEXT	SEXT	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

规则组

0	0	0	0	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

图30 数据左对齐

注入组

SEXT	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0

规则组

D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0	0

我们在试验中用到的是规则组右对齐。

4.11.2 ADC 的数据校准

ADC 有一个内置自校准模式，校准可以大幅减少因内部电容器组的变化而早晨的准精度误差。在校准期间，在每个电容器上都会计算出一个误差修正码（数字值），这个码用于消除在随后的转换中每个电容器上产生的误差。通过设置 ADC_CR2 寄存器的 CAL 位启动校准。一旦校准结束，CAL 位被硬件复位，可以开始正常转换。建议在上电时执行一次 ADC 校准。校准阶段结束后，校准码存储在 ADC_DR 中。

注意：

1、建议在每次上电后执行一次校准。

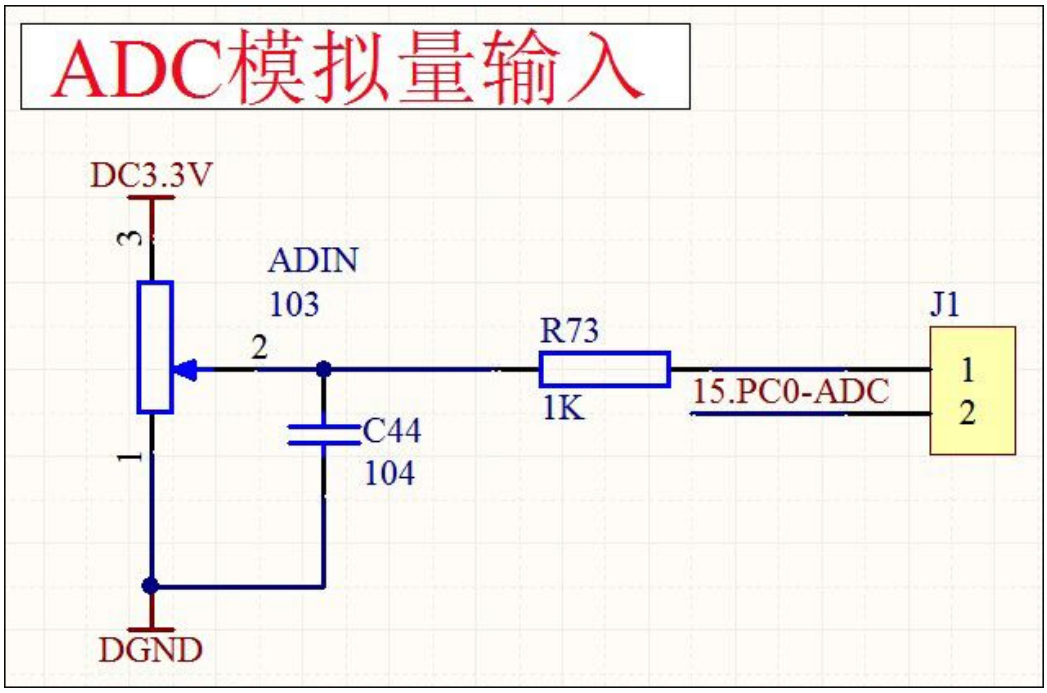
2、启动校准前，ADC 必须处于关电状态（ADON=‘0’）超过至少两个 ADC 时钟周期。

4.11.3 实验目的

通过串口实时打印出模拟量通道 PC0 采集的电压数值。

4.11.4 硬件设计

利用实验板上的 ADC 模拟量转换电路，可以很方便的实现这个功能。



图一 ADC 模拟量转换应用电路

4.11.5 ADC 模拟量转换软件设计

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 ADC 时钟；

2、要对主频进行分频；

3、配置 ADC 相关设置参数

4.11.5.1 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_adc.c  // ADC 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_adc.c 库函数主要包含 ADC 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。

4.11.5.2 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。

4.11.5.3 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_adc.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。

4.11.5.4 pbdata.c 文件里的内容是

```
#include "pbdata.h" //很重要，引用这个头文件

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)
{
   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
}
```

4.11.6 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。

4.11.7 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);//设置分频，6 分频，主频是 72M，分频后
是 12M，符合要求，不大于 14M。
```

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 模拟量时钟打开；设置分频，6 分频，主频是 72M，分频后是 12M，符合要求，要求是不大于 14M。

4.11.8 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)
  {
    ad=0;
   for(i=0;i<50;i++)
    {
      ADC_SoftwareStartConvCmd(ADC1,ENABLE);
      while(!ADC_GetFlagStatus(ADC1,ADC_FLAG_EOC));
      ad=ad+ADC_GetConversionValue(ADC1);
    }

    ad=ad/50;

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

```
    //注意 delay_ms 函数输入范围是 1-1863
    //所以最大延时为 1.8 秒
    delay_ms(1000);
    delay_ms(1000);
    delay_ms(1000);
}
```

4.11.9 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);
   }
}
```

4.11.10 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 ADC 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。

```
#include "pbdata.h"

void RCC_Configuration(void);
void GPIO_Configuration(void);
void NVIC_Configuration(void);
void USART_Configuration(void);
void ADC_Configuration(void);

int fputc(int ch,FILE *f)//屏幕打印
{
    USART_SendData(USART1,(u8)ch);
    while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
    return ch;
}

int main(void)
{
   u32 ad=0;//定义大的变量，存储采样值
   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();

   while(1)
   {
     ad=0;
    for(i=0;i<50;i++)
    {
      ADC_SoftwareStartConvCmd(ADC1,ENABLE);//启动 ADC 转换
      while(!ADC_GetFlagStatus(ADC1,ADC_FLAG_EOC));//判断是否转化完成
      ad=ad+ADC_GetConversionValue(ADC1);//把 50 次采样值累加
    }
     ad=ad/50;//把 50 次采样值累加值除以 50，得到平均值

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

     //注意 delay_ms 函数输入范围是 1-1863
     //所以最大延时为 1.8 秒
     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

}

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;//RX
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AIN;//模拟量输入
    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_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 ADC_Configuration(void)
    {
        ADC_InitTypeDef ADC_InitStructure;//结构体

        ADC_InitStructure.ADC_Mode=ADC_Mode_Independent;//模式
        ADC_InitStructure.ADC_ScanConvMode=DISABLE;//单通道
        ADC_InitStructure.ADC_ContinuousConvMode=DISABLE;//单次循环
        ADC_InitStructure.ADC_ExternalTrigConv=ADC_ExternalTrigConv_None;//软
件启动方式
        ADC_InitStructure.ADC_DataAlign=ADC_DataAlign_Right;//右对齐方式
        ADC_InitStructure.ADC_NbrOfChannel=1;//通道数目，我们用 1 个通道

        ADC_Init(ADC1,&ADC_InitStructure);//把结构体变量参数传递

        ADC_RegularChannelConfig(ADC1,ADC_Channel_10,1,ADC_SampleTime_239Cycl
es5);//规则组

        ADC_Cmd(ADC1,ENABLE);
        //ADC 校准
        ADC_ResetCalibration(ADC1);
        while(ADC_GetResetCalibrationStatus(ADC1));//等待 ADC 是否复位完成

        ADC_StartCalibration(ADC1);//ADC 开始校准
```

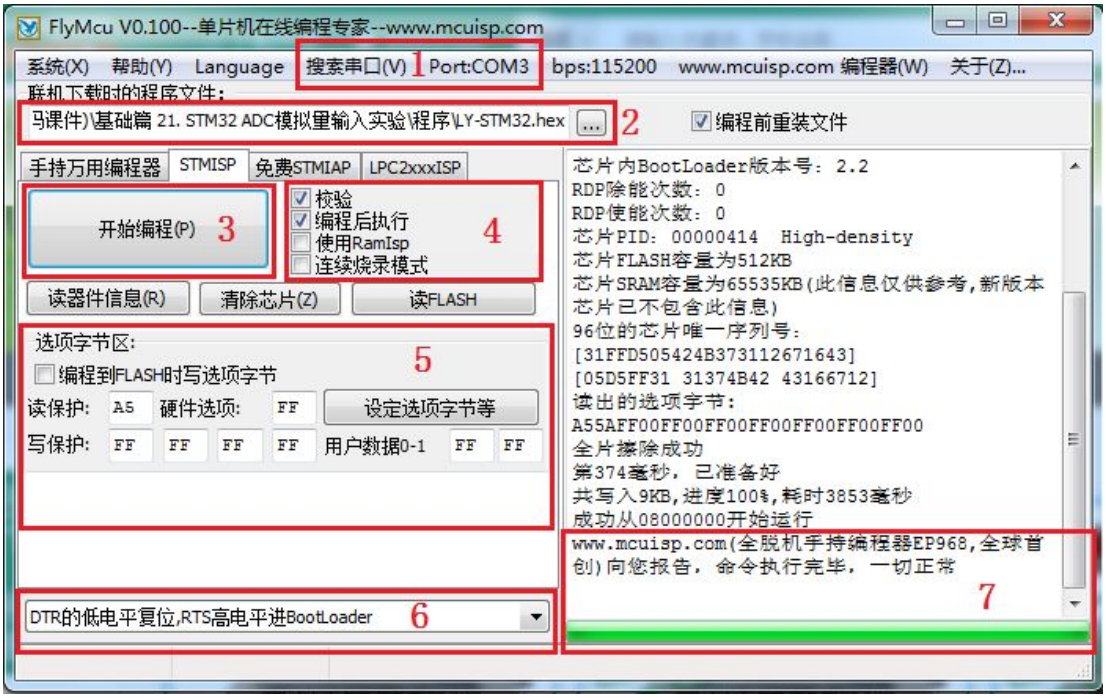
```
        while(ADC_GetCalibrationStatus(ADC1));//ADC 开始校准

        ADC_SoftwareStartConvCmd(ADC1,ENABLE);//等待 ADC 校准是否完成后，启动
ADC

    }
```

表 1　函数 ADC_Init

函数名称	ADC_Init
函数原型	Void ADC_Init(ADC_T Ype*ADCx,ADC_TypeDef*ADC_InitStruct)
功能描述	根据 ADC_InitStruct 中指定的参数初始化外设 ADCx 的寄存器
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输入参数 2	ADC_InitStruct：指向结构 ADC_InitTypeDef 的指针，包含了指定外设 ADC 的配置信息，参阅：ADC_StrucInit 获得 ADC_InitStruct 值的完整描述
输出参数	无
返回值	无
先决条件	无
被调用函数	无

4.11.11 程序下载

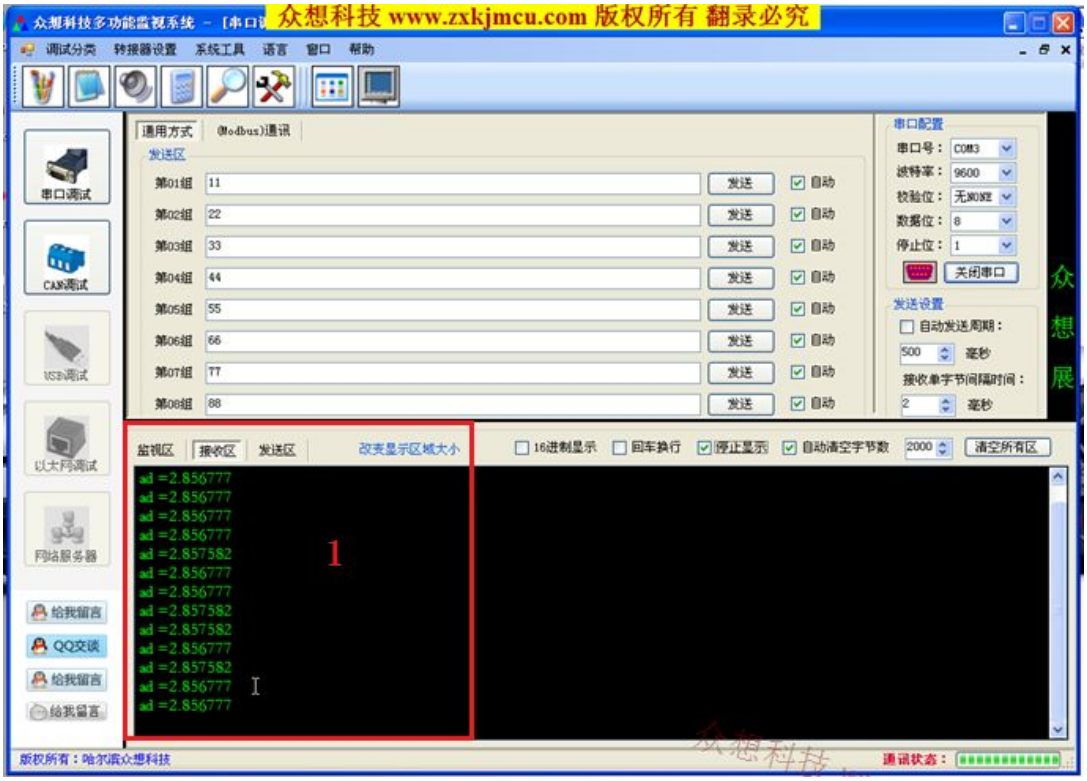
请根据下图所指向的 7 个重点区域配置。其中（1）号区域根据自己机器的实际情况选择，我的机器虚拟出来的串口号是 COM3。（2）号区域请自己选择程序所在的文件夹。（7）号区域当程序下载完后，进度条会到达最右边，并且提示一切正常。（4、5、6）号区域一定要按照上图显示的设置。当都设置好以后就可以直接点击（3）号区域的开始编程按钮下传程序了。



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 21. STM32 ADC 模拟量输入实验\程序）

4.11.12 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，在串口调试界面中的接收区就会就会接收到程序定时发送过来的数据。在 1#红色框图区域是接收到的 AD 模拟量的电压值。此时调节大黄蜂实验板上的电位器，输出电压就会逐渐变大（变小），电压值最大能达到电源电压 3.3V，最小能归零。这个程序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
串口调试
CAN调试
USB调试
以太网调试
网络服务器
给我留言
QQ交谈
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NONE
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
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
ad =2.856777
ad =2.857582
1
版权所有：哈尔滨众想科技
通讯状态：