stm32之ADC应用实例（单通道、多通道、基于DMA）

2018年07月23日 22:25:46 脆弱的代码 阅读数 21117 更多

- **硬件：STM32F103VCT6**
- **开发工具：Keil uVision4**
- **下载调试工具：ARM仿真器**

网上资料很多，这里做一个详细的整合。（也不是很详细，但很通俗）。
所用的芯片内嵌3个12位的模拟/数字转换器(ADC)，每个ADC共用多达16个外部通道，2个内部通道。

3个：代表ADC1、ADC2、ADC3(下图是芯片固件库的截图)

```
#define ADC1             ((ADC_TypeDef *) ADC1_BASE)
#define ADC2             ((ADC_TypeDef *) ADC2_BASE)
#define TIM1             ((TIM_TypeDef *) TIM1_BASE)
#define SPI1             ((SPI_TypeDef *) SPI1_BASE)
#define TIM8             ((TIM_TypeDef *) TIM8_BASE)
#define USART1           ((USART_TypeDef *) USART1_BASE)
#define ADC3             ((ADC_TypeDef *) ADC3_BASE)
```

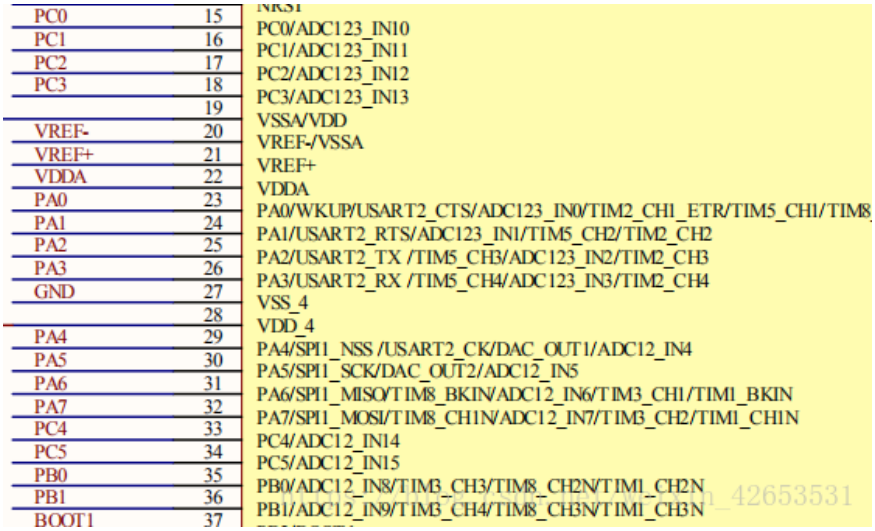
12位：也叫ADC分辨率、采样精度。先来看看二进制的12位可表示0-4095个数，也就是说转换器通过采集转换所得到的最大值是4095，如：“111111111111” =4095，那么我们怎么通过转换器转换出来的值得到实际的电压值呢？如果我们要转换的电压范围是0v-3.3v的话，转换器就会把0v-3.3v平均分成4096份。设转换器所得到的值为x，所求电压值为y。
那么就有：

$$y = \frac{x}{4096} \cdot 3.3V$$

16个外部通道：简单的说就是芯片上有16个引脚是可以接到模拟电压上进行电压值检测的。16个通道不是独立的分配给3个转换器（ADC1、ADC2、ADC3）使用，有些通道是被多个转换器共用的。首先看看16个通道在固件库的宏定义（写代码要看的）：

```
#define ADC_Channel_0                ((uint8_t)0x00)
#define ADC_Channel_1                ((uint8_t)0x01)
#define ADC_Channel_2                ((uint8_t)0x02)
#define ADC_Channel_3                ((uint8_t)0x03)
#define ADC_Channel_4                ((uint8_t)0x04)
#define ADC_Channel_5                ((uint8_t)0x05)
#define ADC_Channel_6                ((uint8_t)0x06)
#define ADC_Channel_7                ((uint8_t)0x07)
#define ADC_Channel_8                ((uint8_t)0x08)
#define ADC_Channel_9                ((uint8_t)0x09)
#define ADC_Channel_10               ((uint8_t)0x0A)
#define ADC_Channel_11               ((uint8_t)0x0B)
#define ADC_Channel_12               ((uint8_t)0x0C)
#define ADC_Channel_13               ((uint8_t)0x0D)
#define ADC_Channel_14               ((uint8_t)0x0E)
#define ADC_Channel_15               ((uint8_t)0x0F)
#define ADC_Channel_16               ((uint8_t)0x10)
#define ADC_Channel_17               ((uint8_t)0x11)
```

到这里大家可能会有疑问，每个通道到底对应哪个引脚呢？下面先给出部分引脚图：



PC0	15	PC0/ADC123_IN10
PC1	16	PC1/ADC123_IN11
PC2	17	PC2/ADC123_IN12
PC3	18	PC3/ADC123_IN13
	19	VSSA/VDD
VREF-	20	VREF-/VSSA
VREF+	21	VREF+
VDDA	22	VDDA
PA0	23	PA0/WKUP/USART2_CTS/ADC123_IN0/TIM2_CH1_ETR/TIM5_CH1/TIM8
PA1	24	PA1/USART2_RTS/ADC123_IN1/TIM5_CH2/TIM2_CH2
PA2	25	PA2/USART2_TX /TIM5_CH3/ADC123_IN2/TIM2_CH3
PA3	26	PA3/USART2_RX /TIM5_CH4/ADC123_IN3/TIM2_CH4
GND	27	VSS_4
	28	VDD_4
PA4	29	PA4/SPI1_NSS /USART2_CK/DAC_OUT1/ADC12_IN4
PA5	30	PA5/SPI1_SCK/DAC_OUT2/ADC12_IN5
PA6	31	PA6/SPI1_MISO/TIM8_BKIN/ADC12_IN6/TIM3_CH1/TIM1_BKIN
PA7	32	PA7/SPI1_MOSI/TIM8_CH1N/ADC12_IN7/TIM3_CH2/TIM1_CH1N
PC4	33	PC4/ADC12_IN14
PC5	34	PC5/ADC12_IN15
PB0	35	PB0/ADC12_IN8/TIM3_CH3/TIM8_CH2N/TIM1_CH2N
PB1	36	PB1/ADC12_IN9/TIM3_CH4/TIM8_CH3N/TIM1_CH3N
BOOT1	37	

16个通道的引脚都在上面的图中，拿其中的一个进行说明：

> **ADC123_IN10**:字母“*ADC*”不用多说，“*123*”代表它被3个（ADC1、ADC2、ADC3）转换器共用的引脚，“*10*”对应刚才那张宏定义图里面的*ADC_Channel_10*,这样就能找到每个通道对应的引脚了。

2个内部通道：一个是内部温度传感器，一个是内部参考电压。

在某个项目中要用到芯片里面的AD转换器，那么要怎么写应用代码？（以下是代码讲解）

芯片固件的库函数为我们提供了很多封装好的函数，只要运用它提供的函数接口就可以了，宏观上来讲就搞懂两个事情就行了：

- 初始化（设置用的哪个引脚、单通道、还是多通道同时转换、是否使用DMA等配置）？
- 怎么让转换器进行一次数据获取？

以下分别讲述三种不同方式（单通道、多通道、基于DMA的多通道采集）的ADC应用实例：

```
/*单通道的ADC采集*/
void  Adc_Config(void)
{
    /*定义两个初始化要用的结构体，下面给每个结构体成员赋值*/
        ADC_InitTypeDef ADC_InitStructure;
        GPIO_InitTypeDef GPIO_InitStructure;

        /*
          使能GPIOA和ADC1通道时钟
          注意：除了RCC_APB2PeriphClockCmd还有RCC_APB1PeriphClockCmd，那么该如何选择？
      APB2：高速时钟，最高72MHz，主要负责AD输入，I/O，串口1，高级定时器TIM
      APB1：低速时钟，最高36MHz，主要负责DA输出，串口2、3、4、5，普通定时器TIM,USB,IIC,CAN，SPI
        */
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA |RCC_APB2Periph_ADC1, ENABLE );
    RCC_ADCCLKConfig(RCC_PCLK2_Div6);  //72M/6=12，ADC的采样时钟最快14MHz

    /*配置输入电压所用的PA0引脚*/
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN; //GPIO_Mode_AIN：模拟输入（还有其他什么模式？请看下面的附录图1）
        GPIO_Init(GPIOA, &GPIO_InitStructure);


        ADC_DeInit(ADC1); //复位，将ADC1相关的寄存器设为默认值
        ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;       //工作模式：ADC1和ADC2独立工作模式   （还有其他什么模式？请看
        ADC_InitStructure.ADC_ScanConvMode = DISABLE;   //数模转换工作：扫描（多通道）模式=ENABLE、单次（单通道）模式=DISABLE
        ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;//数模转换工作：连续=ENABLE、单次=DISABLE
        ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;      //ADC转换由软件触发启动  （还有其他什么模式？
        ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;  //ADC数据右对齐    除了右就是左：ADC_DataAlign_Left
```

```
        ADC_InitStructure.ADC_NbrOfChannel = 1; //顺序进行规则转换的ADC通道的数目    范围是1-16
        ADC_Init(ADC1, &ADC_InitStructure);     //根据ADC_InitStruct中指定的参数初始化外设ADC1的寄存器

    /*为啥要设置下面这一步？
     细心的你可以发现上面初始化了一个引脚通道，初始化了一个ADC转换器，但ADC转换器并不知道你用的是哪个引脚吧？
     这一步目的是：设置指定ADC的规则组通道（引脚），设置它们的转化顺序和采样时间
     函数原型：void ADC_RegularChannelConfig(ADC_TypeDef* ADCx, u8 ADC_Channel, u8 Rank, u8 ADC_SampleTime)
     参数1 ADCx：x可以是1或者2来选择ADC外设ADC1或ADC2
     参数2 ADC_Channel：被设置的ADC通道  范围ADC_Channel_0~ADC_Channel_17
     参数3 Rank：规则组采样顺序。取值范围1到16。
     ADC_SampleTime：指定ADC通道的采样时间值  （取值范围？请看下面的附录图4）
    */
         ADC_RegularChannelConfig(ADC1, ADC_Channel_0, 1, ADC_SampleTime_239Cycles5 );

        ADC_Cmd(ADC1, ENABLE);  //使能指定的ADC  注意：函数ADC_Cmd只能在其他ADC设置函数之后被调用

    /*下面4步按流程走，走完就行*/
        ADC_ResetCalibration(ADC1);     //重置指定的ADC的校准寄存器
        while(ADC_GetResetCalibrationStatus(ADC1)); //等待上一步操作完成
        ADC_StartCalibration(ADC1);     //开始指定ADC的校准状态
        while(ADC_GetCalibrationStatus(ADC1));//等待上一步操作按成
 }
```

初始化完成之后，在主函数中：

```
void main(void)
{
  float ADC_ConvertedValue;
  float ADC_ConvertedValueLocal;

        Adc_Config();
        while(1)
        {
          ADC_SoftwareStartConvCmd(ADC1, ENABLE);               //启动转换
          while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC ));  //等待转换完成

          ADC_ConvertedValue=ADC_GetConversionValue(ADC1); //获取转换结果*ADC_ConvertedValue*
          ADC_ConvertedValueLocal=(float)ADC_ConvertedValue*(3.3/4096);   //计算出实际电压值*ADC_ConvertedValueLocal*

          //这里适当加上一些延迟
          //最好连续转换几次 取平均值  这里就省略写了 点到为止
        }
}
```

附录图1-**GPIO_Mode**值：

GPIO_Speed	描述
GPIO_Mode_AIN	模拟输入
GPIO_Mode_IN_FLOATING	浮空输入
GPIO_Mode_IPD	下拉输入
GPIO_Mode_IPU	上拉输入
GPIO_Mode_Out_OD	开漏输出
GPIO_Mode_Out_PP	推挽输出
GPIO_Mode_AF_OD	复用开漏输出
GPIO_Mode_AF_PP	复用推挽输出

附录图2-**ADC_Mode**值：

ADC_Mode	描述
ADC_Mode_Independent	ADC1 和 ADC2 工作在独立模式
ADC_Mode_RegInjecSimult	ADC1 和 ADC2 工作在同步规则和同步注入模式
ADC_Mode_RegSimult_AlterTrig	ADC1 和 ADC2 工作在同步规则模式和交替触发模式
ADC_Mode_InjecSimult_FastInterl	ADC1 和 ADC2 工作在同步规则模式和快速交替模式
ADC_Mode_InjecSimult_SlowInterl	ADC1 和 ADC2 工作在同步注入模式和慢速交替模式
ADC_Mode_InjecSimult	ADC1 和 ADC2 工作在同步注入模式
ADC_Mode_RegSimult	ADC1 和 ADC2 工作在同步规则模式
ADC_Mode_FastInterl	ADC1 和 ADC2 工作在快速交替模式
ADC_Mode_SlowInterl	ADC1 和 ADC2 工作在慢速交替模式
ADC_Mode_AlterTrig	ADC1 和 ADC2 工作在交替触发模式

附录图3-**ADC_ExternalTrigConv**值:

ADC_ExternalTrigConv	描述
ADC_ExternalTrigConv_T1_CC1	选择定时器 1 的捕获比较 1 作为转换外部触发
ADC_ExternalTrigConv_T1_CC2	选择定时器 1 的捕获比较 2 作为转换外部触发
ADC_ExternalTrigConv_T1_CC3	选择定时器 1 的捕获比较 3 作为转换外部触发
ADC_ExternalTrigConv_T2_CC2	选择定时器 2 的捕获比较 2 作为转换外部触发
ADC_ExternalTrigConv_T3_TRGO	选择定时器 3 的 TRGO 作为转换外部触发
ADC_ExternalTrigConv_T4_CC4	选择定时器 4 的捕获比较 4 作为转换外部触发
ADC_ExternalTrigConv_Ext_IT11	选择外部中断线 11 事件作为转换外部触发
ADC_ExternalTrigConv_None	转换由软件而不是外部触发启动

附录图4-**ADC_SampleTime**值:

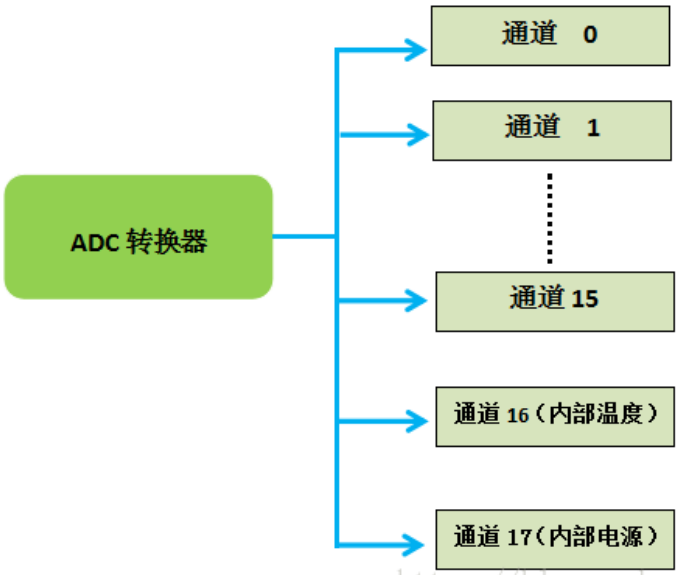
ADC_SampleTime	描述
ADC_SampleTime_1Cycles5	采样时间为 1.5 周期
ADC_SampleTime_7Cycles5	采样时间为 7.5 周期
ADC_SampleTime_13Cycles5	采样时间为 13.5 周期
ADC_SampleTime_28Cycles5	采样时间为 28.5 周期
ADC_SampleTime_41Cycles5	采样时间为 41.5 周期
ADC_SampleTime_55Cycles5	采样时间为 55.5 周期
ADC_SampleTime_71Cycles5	采样时间为 71.5 周期
ADC_SampleTime_239Cycles5	采样时间为 239.5 周期

对于一些刚接触stm32的人来说，看了上面的代码可能还会有很多疑问。

- 为什么要使能时钟？时钟到底设置多少才合适?
- 对于ADC_GetConversionValue(ADC1)这个函数参数并没有指定那个通道，如果多个通道同时使用CAN1转换器转换时怎么获取每个通道的值?

第一个问题，所有的外设都要使能时钟，时钟源分为外部时钟和内部时钟，外部时钟比如接8MHz晶振，内部时钟就在芯片内部集成，时钟源为所有的时序电路提供基本的脉冲信号。时钟源好比是一颗跳动的心脏，它按照一定的频率在跳动，所有的器官（外设）要跟心脏（时钟源）桥接起来才能工作，但不同的外设需要的频率不同，所以在时钟源跟外设之中常常还会有一些分频器或者倍频器，以实现对频率的衰减或增强。还想了解更多专业的解释可以去研究stm32的时钟树图。

第二个问题，回答这个问题那么就等于开始介绍多通道转换怎么实现了，看下图



由图理解，一个ADC转换器只能选择转换一个通道，那么对比单通道我们只需做一下改变（以双通道为例）：
1.在void Adc_Config(void)函数里面添加：

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
GPIO_Init(GPIOA, &GPIO_InitStructure);
```

配置多一个IO（PA1）口， 也就是通道1。

2.在void Adc_Config(void)函数里面添加：

```
ADC_RegularChannelConfig(ADC1, ADC_Channel_0, 1, ADC_SampleTime_239Cycles5 );

```

先不指定ADC转换通道。

3.在主函数循环里改为：

```
    while(1)
        {
          /*先采集通道1数据*/
          ADC_RegularChannelConfig(ADC1, ADC_Channel_0, 1, ADC_SampleTime_239Cycles5 );
          ADC_SoftwareStartConvCmd(ADC1, ENABLE);
          while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC ));
          ADC_ConvertedValue=ADC_GetConversionValue(ADC1);
          ADC_ConvertedValueLocal=(float)ADC_ConvertedValue*(3.3/4096);

          /*再采集通道2数据*/
          ADC_RegularChannelConfig(ADC1, ADC_Channel_1, 1, ADC_SampleTime_239Cycles5 );
      ADC_SoftwareStartConvCmd(ADC1, ENABLE);
          while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC ));
          ADC_ConvertedValue=ADC_GetConversionValue(ADC1);
          ADC_ConvertedValueLocal=(float)ADC_ConvertedValue*(3.3/4096);

          //加入适当延时
        }

```

完成以上三步就能把单通道扩展到双通道(或者更多个通道)。不过还有一种基于DMA的多通道转换更加合适。

首先简单介绍DMA，DMA(Direct Memory Access，直接内存存取)，用来提供在外设和存储器之间或者存储器和存储器之间的高速数据传输。无需CPU干预，节省CPU资源；ADC转换出来的值直接赋值给定义好的变量中。配置好的DMA可以不停地将ADC转换值写到该变量中，在主函数直接判断该变量就知道此时的AD值，也就是说在主函数中不需要调用ADC_GetConversionValue()函数来获取转换值。

DMA跟其他外设一样需要进行配置通道，使能时钟等参数。
下面直接看代码分析：

```
/*基于DMA的ADC多通道采集*/

volatile uint16 ADCConvertedValue[10][3];//用来存放ADC转换结果，也是DMA的目标地址,3通道，每通道采集10次后面取平均数

void DMA_Init(void)
{

    DMA_InitTypeDef DMA_InitStructure;

    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);//使能时钟

    DMA_DeInit(DMA1_Channel1);    //将通道一寄存器设为默认值
    DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)&(ADC1->DR);//该参数用以定义DMA外设基地址
    DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)&ADCConvertedValue;//该参数用以定义DMA内存基地址(转换结果保存的地址)
    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;//该参数规定了外设是作为数据传输的目的地还是来源，此处是作为来源
    DMA_InitStructure.DMA_BufferSize = 3*10;//定义指定DMA通道的DMA缓存的大小,单位为数据单位。这里也就是ADCConvertedValue的大小
    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;//设定外设地址寄存器递增与否,此处设为不变 Disable
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;//用来设定内存地址寄存器递增与否,此处设为递增，Enable
    DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;//数据宽度为16位
    DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;//数据宽度为16位
    DMA_InitStructure.DMA_Mode = DMA_Mode_Circular; //工作在循环缓存模式
    DMA_InitStructure.DMA_Priority = DMA_Priority_High;//DMA通道拥有高优先级 分别4个等级 低、中、高、非常高
    DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;//使能DMA通道的内存到内存传输
    DMA_Init(DMA1_Channel1, &DMA_InitStructure);//根据DMA_InitStruct中指定的参数初始化DMA的通道

    DMA_Cmd(DMA1_Channel1, ENABLE);//启动DMA通道一
}
```

下面是ADC的初始化，可以将它与上面的对比一下有啥不同，重复的就不解析了

```
void Adc_Init(void)
{
    ADC_InitTypeDef ADC_InitStructure;
    GPIO_InitTypeDef GPIO_InitStructure;
    /*3个IO口的配置（PA0、PA1、PA2）*/
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    /*IO和ADC使能时钟*/
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1|RCC_APB2Periph_GPIOA,ENABLE);
    RCC_ADCCLKConfig(RCC_PCLK2_Div6);

    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
    ADC_InitStructure.ADC_ScanConvMode = ENABLE;
    ADC_InitStructure.ADC_ContinuousConvMode = ENABLE; //连续转换
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
```

```
    ADC_InitStructure.ADC_NbrOfChannel = 3;
    ADC_Init(ADC1, &ADC_InitStructure);

    ADC_RegularChannelConfig(ADC1,ADC_Channel_0,1,ADC_SampleTime_239Cycles5);//通道一转换结果保存到ADCConvertedValue[0~10]
        ADC_RegularChannelConfig(ADC1,ADC_Channel_1,2,ADC_SampleTime_239Cycles5););//通道二转换结果保存到ADCConvertedValue
        ADC_RegularChannelConfig(ADC1,ADC_Channel_2,3,ADC_SampleTime_239Cycles5); );//通道三转换结果保存到ADCConvertedValu


    ADC_DMACmd(ADC1, ENABLE);//开启ADC的DMA支持
    ADC_Cmd(ADC1, ENABLE);

    ADC_ResetCalibration(ADC1);
    while(ADC_GetResetCalibrationStatus(ADC1));
    ADC_StartCalibration(ADC1);
    while(ADC_GetCalibrationStatus(ADC1));

}
```

做完这两步，ADCConvertedValue数组的值就会随输入的模拟电压改变而改变，在主函数中最好取多几次的平均值，再通过公式换算成电压单位。下面是主函数：

```
int main(void)
{
        int sum;
        u8 i,j;
        float ADC_Value[3];//用来保存经过转换得到的电压值
        ADC_Init();
        DMA_Init();

        ADC_SoftwareStartConvCmd(ADC1, ENABLE);//开始采集

        while(1)
        {
                for(i=0;i<3;i<++)
                {
                        sum=0;
                        for(j=0;j<10;j++)
                        {
                                sum+=ADCConvertedValue[j][i];
                        }
                        ADC_Value[i]=(float)sum/(10*4096)*3.3;//求平均值并转换成电压值
                        //打印（略）
                }
                //延时（略）
        }
}
```

ADCConvertedValue的定义用了volatile修饰词，因为这样可以保证每次的读取都是从绝对地址读出来的值，不会因为被会编译器进行优化导致读取到的值不是实时的AD值。

最后提醒一下，接线测试的时候记得接上基准电压，就是VREF+和VREF-这两个引脚。如果不想外接线测试就将内部通道的电压读出来，这样就不用配置IO口了。

水平有限，仅供参考，错误之处以及不足之处还望多多指教。