下面我们将 PC0 配置成 AD1 的通道 10 为例进行讲解。

3.1 首先我们应将 PC0 设置成模拟输入：

```
#include "adc.h"
/* 为何定义 ADC1_DR_Address 为((u32)0x40012400+0x4c) ，因为存放 AD 转换结果的寄存器的地址就是 0x4001244c*/ #define ADC1_DR_Address ((u32)0x40012400+0x4c)
/*定义变量 ADC_ConvertedValue,放 AD1 通道 10 转换的数据*/ __IO uint16_t ADC_ConvertedValue;

    static void ADC1_GPIO_Config(void)
{
 GPIO_InitTypeDef GPIO_InitStructure; /* Enable ADC1 and GPIOC clock */
 RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 | RCC_APB2Periph_GPIOC,ENABLE);
 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN; GPIO_Init(GPIOC, &GPIO_InitStructure);
 }
```

3.2 设置完端口后下一步当然是对 AD 进行初始化：

这里需要补充一个知识点 DMA，DMA 就相当与 CPU 的一个秘书，他的作用就是帮 CPU 减轻负担的。说的再具体点就是帮 CPU 来转移数据的。我们都知道，AD 每次转换结束后会将转换的结果放到一个固定的寄存器里，以往我们如果想将该寄存器中的值赋给某一变量时会用到赋值语句，如果不用 DMA，则赋值语句便要 CPU 来完成，CPU 本来就要忙着处理其他事情，现在还要来解决赋值语句这么简单的问题，肯到会蛋疼。所以需要 DMA 这个秘书来帮他解决这个问题。由于 DMA 只是个秘书，所以比较笨，你只有把任务交代清楚了她才能很好的完成任务。那么怎样来给 DMA 吩咐任务呢，聪明的人肯定想到了，那当然是“DMA_Init(DMA1_Channel1, &DMA_InitStructure)”这个函数啦。下面就来一步步的来给 DMA 交代任务。

```
/* 函数名：ADC1_Mode_Config
* 描述 ： 配置 ADC1 的工作模式为 MDA 模式 * 输入 : 无 * 输出 ： 无
* 调用 ： 内部调用 */
static void ADC1_Mode_Config(void)
 {
 DMA_InitTypeDef DMA_InitStructure;
ADC_InitTypeDef ADC_InitStructure;

   /* 将与 DMA 有关的寄存器设我初始值 */
 DMA_DeInit(DMA1_Channel1);
   /*定义 DMA 外设基地址, 这里的 ADC1_DR_Address 是用户自己定义的，即为存放转换结果的寄存器 ，他的作用就是告诉 DMA 取数就到 ADC1_DR_Address 这里来取。*/
 DMA_InitStructure.DMA_PeripheralBaseAddr =ADC1_DR_Address;
   /*定义内存基地址，即告诉 DMA 要将从 AD 中取来的数放到 ADC_ConvertedValue 中 */
 DMA_InitStructure.DMA_MemoryBaseAddr =(u32)&ADC_ConvertedValue;
    /*定义 AD 外设作为数据传输的来源，即告诉 DMA 是将 AD 中的数据取出放到内存中，不能反过来*/
 DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
   /*指定 DMA 通道的 DMA 缓存的大小,即告诉 DMA 开辟几个内存空间，由于我们只取通道 10 的 AD 数据所以只需开辟一个内存空间*/
DMA_InitStructure.DMA_BufferSize = 1;
   /*设定寄存器地址固定，即告诉 DMA，只从固定的一个地方取数*/
 DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
   /*设定内存地址固定，即每次 DMA，，只将数搬到固定的内存中*/
 DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
```

此处有错误，应该是 DMA_MemoryInc_Disable;

```
    /*设定外设数据宽度，即告诉 DMA 要取的数的大小*/
  DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;


   /*设定内存的的宽度*/
 DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
    /*设定 DMA 工作再循环缓存模式，即告诉 DMA 要不停的搬运，不能偷懒*/
 DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
    /*设定 DMA 选定的通道软件优先级*/
 DMA_InitStructure.DMA_Priority = DMA_Priority_High;
  DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
DMA_Init(DMA1_Channel1, &DMA_InitStructure);
 /* Enable DMA channel1，CPU 有好几个 DMA 秘书，现在只用 DMA1_Channel1 这个秘书*/
  DMA_Cmd(DMA1_Channel1, ENABLE);

   /*设置 ADC 工作在独立模式*/
 ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
    /*规定 AD 转换工作在单次模式，即对一个通道采样*/
ADC_InitStructure.ADC_ScanConvMode = DISABLE
    /*设定 AD 转化在连续模式*/
 ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
   /*不使用外部促发转换*/
 ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
  /*采集的数据在寄存器中以右对齐的方式存放*/
 ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
 /*设定要转换的 AD 通道数目*/
 ADC_InitStructure.ADC_NbrOfChannel = 1;
 ADC_Init(ADC1, &ADC_InitStructure);
 /*配置 ADC 时钟，为 PCLK2 的 8 分频，即 9MHz*/
 RCC_ADCCLKConfig(RCC_PCLK2_Div8);
 /*配置 ADC1 的通道 11 为 55.5 个采样周期 */
ADC_RegularChannelConfig(ADC1, ADC_Channel_10, 1, ADC_SampleTime_55Cycles5);
   /* Enable ADC1 DMA */
 ADC_DMACmd(ADC1, ENABLE);
 /* Enable ADC1 */
 ADC_Cmd(ADC1, ENABLE);
/*复位校准寄存器 */
 ADC_ResetCalibration(ADC1);
/*等待校准寄存器复位完成 */
 while(ADC_GetResetCalibrationStatus(ADC1));
 /* ADC 校准 */
 ADC_StartCalibration(ADC1);
/* 等待校准完成*/
 while(ADC_GetCalibrationStatus(ADC1));
 /* 由于没有采用外部触发，所以使用软件触发 ADC 转换 */
 ADC_SoftwareStartConvCmd(ADC1, ENABLE);
}
```

配置完以上的程序，那么 AD 每转换一次，DMA 都会将转换结果搬到变量 ADC_ConvertedValue 中，而不需用每次都用赋值语句来取值 AD 转换的值。

第二部分：AD 多路采样

```
 #include "adc.h"
 #define    ADC1_DR_Address       ((u32)0x40012400+0x4c)
 /*定义数组变量 ADC_ConvertedValue[2],分别放 AD1 通道 10 和 11 转换的数据*/
    __IO uint16_t ADC_ConvertedValue[2];
/*   * 函数名：ADC1_GPIO_Config   * 描述 ：使能 ADC1 和 DMA1 的时钟,设置 PC0，PC1 为模拟输入 * 输入 ： 无 * 输出 ： 无   * 调用 ： 内部调用 */
 static void ADC1_GPIO_Config(void)
 {     GPIO_InitTypeDef    GPIO_InitStructure;
   /* Enable DMA clock */
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
   /* Enable ADC1 and GPIOC clock */
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1| RCC_APB2Periph_GPIOC,ENABLE);
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 |GPIO_Pin_1;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOC, &GPIO_InitStructure);
   }
```

```
    /* 函数名：ADC1_Mode_Config   * 描述 ：配置 ADC1 的工作模式为 MDA 模式 * 输入 ： 无 * 输出 ：无   * 调用 ： 内部调用 */
    static void ADC1_Mode_Config(void)
   {
    DMA_InitTypeDef    DMA_InitStructure;
    ADC_InitTypeDef    ADC_InitStructure;

/* DMA channel1 configuration */
    DMA_DeInit(DMA1_Channel1);

     /*定义 DMA 外设基地址,即为存放转换结果的寄存器*/
   DMA_InitStructure.DMA_PeripheralBaseAddr =ADC1_DR_Address;
     /*定义内存基地址*/
   DMA_InitStructure.DMA_MemoryBaseAddr =(u32)&ADC_ConvertedValue;

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

   /*指定 DMA 通道的 DMA 缓存的大小,即需要开辟几个内存空间，本实验有两个转换通道，所以开辟两个*/
   DMA_InitStructure.DMA_BufferSize = 2;

   /*设定寄存器地址固定*/
   DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;

   /*设定内存地址递加，即每次 DMA 都是将该外设寄存器中的值传到两个内存空间中*/
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
```

```
/*设定外设数据宽度*/
 DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
/*设定内存的的宽度*/
 DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
/*设定 DMA 工作再循环缓存模式*/
DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
   /*设定 DMA 选定的通道软件优先级*/
 DMA_InitStructure.DMA_Priority = DMA_Priority_High;
DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
DMA_Init(DMA1_Channel1, &DMA_InitStructure);
/* Enable DMA channel1 */
DMA_Cmd(DMA1_Channel1, ENABLE);

/*设置 ADC 工作在独立模式*/

ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;

/*规定 AD 转换工作在扫描模式，即对多个通道采样*/
ADC_InitStructure.ADC_ScanConvMode = ENABLE ；


/*设定 AD 转化在连续模式*/
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;

/*不使用外部促发转换*/

ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;

/*采集的数据在寄存器中以右对齐的方式存放*/
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;

/*设定要转换的 AD 通道数目*/
ADC_InitStructure.ADC_NbrOfChannel = 2;


ADC_Init(ADC1, &ADC_InitStructure);
/*配置 ADC 时钟，为 PCLK2 的 8 分频，即 9MHz*/
RCC_ADCCLKConfig(RCC_PCLK2_Div8);
/*配置 ADC1 的通道 10 和 11 的转换先后顺序以及采样时间为为 55.5 个采样周期 */
ADC_RegularChannelConfig(ADC1,ADC_Channel_10,1,ADC_SampleTime_55Cycles5);
ADC_RegularChannelConfig(ADC1, ADC_Channel_11, 2, ADC_SampleTime_55Cycles5);
/* Enable ADC1 DMA */
ADC_DMACmd(ADC1, ENABLE);
  /* Enable ADC1 */
  ADC_Cmd(ADC1, ENABLE);
/*复位校准寄存器 */
ADC_ResetCalibration(ADC1);
  /*等待校准寄存器复位完成 */
while(ADC_GetResetCalibrationStatus(ADC1));
```

```
/* ADC 校准 */
ADC_StartCalibration(ADC1);
 /* 等待校准完成*/
 while(ADC_GetCalibrationStatus(ADC1));
 /* 由于没有采用外部触发，所以使用软件触发 ADC 转换 */
 ADC_SoftwareStartConvCmd(ADC1, ENABLE); }
```

!!!!! 单通道采样与多通道采样的不同点都在第二段程序中用红色标出来了，注意比较。 总结：DMA 就是一个无私奉献的搬运工，想将外设寄存器中的值放入内存中原本需要 CPU 来完成，现在 DMA 来帮 CPU 完成，这在一定程度上解放了 CPU

STM32 ADC 多通道转换

描述：用 ADC 连续采集 11 路模拟信号，并由 DMA 传输到内存。ADC 配置为扫描并且连续转换模式，ADC 的时钟配置为 12MHZ。在每次转换结束后，由 DMA 循环将转换的数据传输到内存中。ADC 可以连续采集 N 次求平均值。最后通过串口传输出最后转换的结果。

程序如下：

```
#include "stm32f10x.h"          //这个头文件包括 STM32F10x 所有外围寄存器、位、内存映射的定义
#include "eval.h"               //头文件（包括串口、按键、LED 的函数声明）
#include "SysTickDelay.h"
#include "UART_INTERFACE.h"
#include <stdio.h>

#define   N     50          //每通道采 50 次
#define   M    12           //为 12 个通道

vu16   AD_Value[N][M];     //用来存放 ADC 转换结果，也是 DMA 的目标地址
vu16   After_filter[M];      //用来存放求平均值之后的结果
int     i;


/*GPIO 管脚的配置
选用 ADC 的通道 0   1   2   8   9   10   11   12   13   14   15，分别对应的管脚为 PA0   PA1
PA2   PB0   PB1   PC0   PC1   PC2   PC3   PC4   PC5
串口使用 USART1 其中 TX 为 PA9，RX 为 PA10 */
void GPIO_Configuration(void)
     {
     GPIO_InitTypeDef    GPIO_InitStructure;
    /* Configure USART1 Tx (PA.09) as alternate function push-pull */
     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;    //因为 USART1 管脚是以复用的形式接到 GPIO 口上的，所以使用复用推挽式输出
     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
     GPIO_Init(GPIOA, &GPIO_InitStructure);

     /* Configure USART1 Rx (PA.10) as input floating */
     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
     GPIO_Init(GPIOA, &GPIO_InitStructure);
```

```
    //PA0/1/2  作为模拟通道输入引脚
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0| GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;          //模拟输入引脚
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    //PB0/1  作为模拟通道输入引脚
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0|GPIO_Pin_1;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;          //模拟输入引脚
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    //PC0/1/2/3/4/5  作为模拟通道输入引脚
    GPIO_InitStructure.GPIO_Pin                                                    =
GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3|GPIO_Pin_4|GPIO_Pin_5;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;          //模拟输入引脚
    GPIO_Init(GPIOC, &GPIO_InitStructure);
    }

  }



/*配置系统时钟,使能各外设时钟*/
void RCC_Configuration(void)
    {
     ErrorStatus    HSEStartUpStatus;

     RCC_DeInit();                                  //RCC  系统复位
     RCC_HSEConfig(RCC_HSE_ON);           //开启 HSE
     HSEStartUpStatus = RCC_WaitForHSEStartUp();    //等待 HSE 准备好
     if(HSEStartUpStatus == SUCCESS)
     {
        FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable); //Enable Prefetch Buffer
        FLASH_SetLatency(FLASH_Latency_2);          //Set 2 Latency cycles
        RCC_HCLKConfig(RCC_SYSCLK_Div1);     //AHB clock    = SYSCLK
        RCC_PCLK2Config(RCC_HCLK_Div1);      //APB2 clock = HCLK
        RCC_PCLK1Config(RCC_HCLK_Div2);      //APB1 clock = HCLK/2
        RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_6); //PLLCLK = 12MHz * 6 = 72
MHz
        RCC_PLLCmd(ENABLE);                  //Enable PLL
        while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);    //Wait till PLL is ready
        RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK); //Select PLL as system clock source
        while(RCC_GetSYSCLKSource() != 0x08);      //Wait till PLL is used as system clock source
```

```
    /*使能各个外设时钟*/
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB
        |  RCC_APB2Periph_GPIOC  |RCC_APB2Periph_ADC1  |  RCC_APB2Periph_AFIO
|RCC_APB2Periph_USART1, ENABLE );   //使能 ADC1 通道时钟，各个管脚时钟
    /* Configure ADCCLK such as ADCCLK = PCLK2/6 */
    RCC_ADCCLKConfig(RCC_PCLK2_Div6);   //72M/6=12,ADC 最大时间不能超过 14M
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);   //使能 DMA 传输

    }
  }



/*配置 ADC1*/
void ADC1_Configuration(void)
    {
     ADC_InitTypeDef   ADC_InitStructure;

    ADC_DeInit(ADC1);  //将外设 ADC1 的全部寄存器重设为缺省值

    /* ADC1 configuration ------------------------------------------------------*/
    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent; //ADC 工作模式:ADC1 和 ADC2
工作在独立模式
    ADC_InitStructure.ADC_ScanConvMode =ENABLE;    //模数转换工作在扫描模式
    ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;  //模数转换工作在连续转换模
式
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None; //外部触发转换
关闭
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right; //ADC 数据右对齐
    ADC_InitStructure.ADC_NbrOfChannel = M; //顺序进行规则转换的 ADC 通道的数目
    ADC_Init(ADC1, &ADC_InitStructure);  //根据 ADC_InitStruct 中指定的参数初始化外设
ADCx 的寄存器

    /* ADC1 regular channel11 configuration */
    //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
    //ADC1,ADC 通道 x,规则采样顺序值为 y,采样时间为 239.5 周期
    ADC_RegularChannelConfig(ADC1, ADC_Channel_0, 1, ADC_SampleTime_239Cycles5 );
    ADC_RegularChannelConfig(ADC1, ADC_Channel_1, 2, ADC_SampleTime_239Cycles5 );
    ADC_RegularChannelConfig(ADC1, ADC_Channel_2, 3, ADC_SampleTime_239Cycles5 );
    ADC_RegularChannelConfig(ADC1, ADC_Channel_3, 4, ADC_SampleTime_239Cycles5 );
    ADC_RegularChannelConfig(ADC1, ADC_Channel_8, 5, ADC_SampleTime_239Cycles5 );
    ADC_RegularChannelConfig(ADC1, ADC_Channel_9, 6, ADC_SampleTime_239Cycles5 );
    ADC_RegularChannelConfig(ADC1, ADC_Channel_10, 7, ADC_SampleTime_239Cycles5 );
    ADC_RegularChannelConfig(ADC1, ADC_Channel_11, 8, ADC_SampleTime_239Cycles5 );
```

```
    ADC_RegularChannelConfig(ADC1, ADC_Channel_12, 9, ADC_SampleTime_239Cycles5 );
    ADC_RegularChannelConfig(ADC1, ADC_Channel_13, 10, ADC_SampleTime_239Cycles5 );
    ADC_RegularChannelConfig(ADC1, ADC_Channel_14, 11, ADC_SampleTime_239Cycles5 );
    ADC_RegularChannelConfig(ADC1, ADC_Channel_15, 12, ADC_SampleTime_239Cycles5 );

     // 开启 ADC 的 DMA 支持（要实现 DMA 功能，还需独立配置 DMA 通道等参数）
    ADC_DMACmd(ADC1, ENABLE);

    /* Enable ADC1 */
     ADC_Cmd(ADC1, ENABLE);       //使能指定的 ADC1
     /* Enable ADC1 reset calibaration register */
    ADC_ResetCalibration(ADC1);   //复位指定的 ADC1 的校准寄存器
    /* Check the end of ADC1 reset calibration register */
    while(ADC_GetResetCalibrationStatus(ADC1));    //获取 ADC1 复位校准寄存器的状态,设置状态则等待

    /* Start ADC1 calibaration */
    ADC_StartCalibration(ADC1);       //开始指定 ADC1 的校准状态
    /* Check the end of ADC1 calibration */
    while(ADC_GetCalibrationStatus(ADC1));          //获取指定 ADC1 的校准程序,设置状态则等待


}



/*配置 DMA*/
void DMA_Configuration(void)
    {
    /* ADC1    DMA1 Channel Config */
    DMA_InitTypeDef DMA_InitStructure;
    DMA_DeInit(DMA1_Channel1);    //将 DMA 的通道 1 寄存器重设为缺省值
    DMA_InitStructure.DMA_PeripheralBaseAddr =   (u32)&ADC1->DR;   //DMA 外设 ADC 基地址
    DMA_InitStructure.DMA_MemoryBaseAddr = (u32)&AD_Value;   //DMA 内存基地址
    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;   //内存作为数据传输的目的地
    DMA_InitStructure.DMA_BufferSize = N*M;   //DMA 通道的 DMA 缓存的大小
    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;   //外设地址寄存器不变
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;   //内存地址寄存器递增
    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 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);   //根据 DMA_InitStruct 中指定的参数初始化 DMA 的通道

    }


//配置所有外设
void Init_All_Periph(void)
    {

    RCC_Configuration();

    GPIO_Configuration();

    ADC1_Configuration();

    DMA_Configuration();

    //USART1_Configuration();
    USART_Configuration(9600);


    }


/*获取 ADC 的值，将二进制换算为十进制*/
u16 GetVolt(u16 advalue)

  {

   return (u16)(advalue * 330 / 4096);     //求的结果扩大了 100 倍，方便下面求出小数

  }

/*求平均值函数*/
void filter(void)
{
    int    sum = 0;
```

```
    u8    count;
    for(i=0;i<12;i++)

      {

          for ( count=0;count<N;count++)

           {

             sum += AD_Value[count][i];

           }

           After_filter[i]=sum/N;

           sum=0;
      }

 }



int main(void)
    {

     u16       value[M];

    init_All_Periph();
    SysTick_Initaize();
    /* Start ADC1 Software Conversion */
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);
    DMA_Cmd(DMA1_Channel1, ENABLE);   //启动 DMA 通道
    while(1)
        {
        while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);//等待传输完成否则
第一位数据容易丢失

           filter();
           for(i=0;i<12;i++)
            {
              value[i]= GetVolt(After_filter[i]);

              printf("value[%d]:\t%d.%dv\n",i,value[i]/100,value[i]%100) ;
```

```
            delay_ms(100);
          }
        }

  }
```

总结

该程序中的两个宏定义，M 和 N，分别代表有多少个通道，每个通道转换多少次，可以修改其值。

曾出现的问题：配置时钟时要知道外部晶振是多少，以便准确配置时钟。将转换值由二进制转换为十进制时，要先扩大 100 倍，方便显示小数。最后串口输出时在 printf 语句之前加这句代码，防止输出的第一位数据丢失：

```
while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
```