ADC 实验

作者	fire
E-Mail	firestm32@foxmail.com
QQ	313303034
博客	firestm32.blog.chinaunix.net
硬件平台	野火 STM32 开发板
库版本	ST3.0.0

实验描述：串口 1(USART1)向电脑的超级终端以 1s 为时间间隔打印当前 ADC1 的转换电压值。

硬件连接：PC1 - ADC1 连接外部电压(通过一个滑动变阻器分压而来)。

库文件 ：startup/start_stm32f10x_hd.c

CMSIS/core_cm3.c

CMSIS/system_stm32f10x.c

FWlib/stm32f10x_gpio.c

FWlib/stm32f10x_rcc.c

FWlib/stm32f10x_usart.c

FWlib/stm32f10x_adc.c

FWlib/stm32f10x_dma.c

FWlib/stm32f10x_flash.c

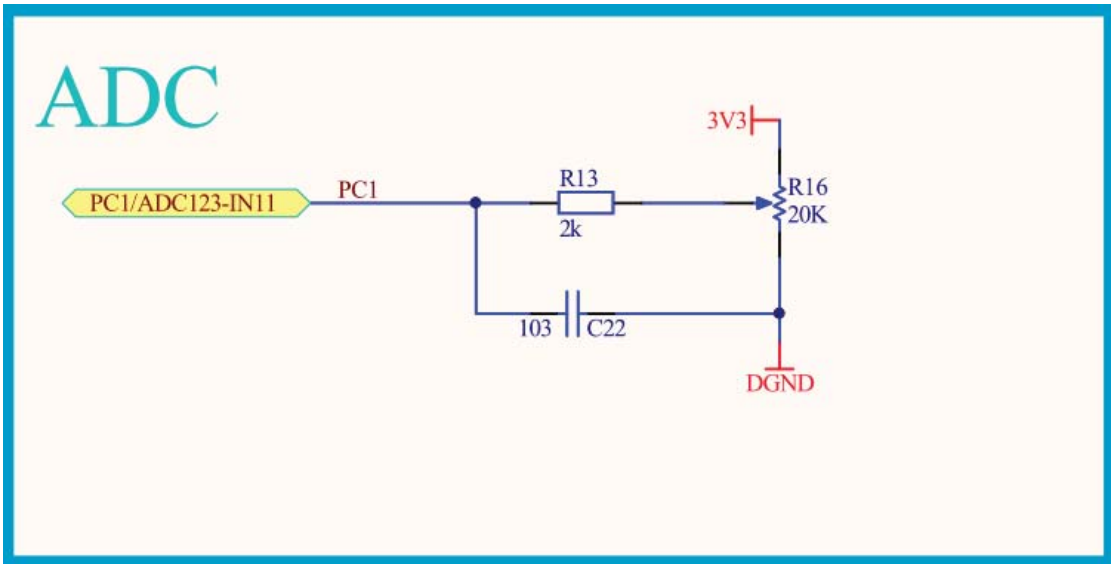
用户文件：USER/main.c

USER/stm32f10x_it.c

USER/usart1.c

USER/adc.c

野火 **STM32** 开发板 **ADC** 硬件原理图：



ADC 简介->

STM32F103xC、STM32F103xD 和 STM32F103xE 增强型产品，内嵌 3 个 12 位的模拟/数字转换器(ADC)，每个 ADC 共用多达 21 个外部通道，可以实现单次或多次扫描转换。STM32 开发板用的是 STM32F103VET6，属于增强型的 CPU。它有 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、扫描或间断模式执行。ADC 的结果可以左对齐或右对齐方式存储在 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 可以使用 DMA(data memory access)方式操作。

本实验用的是 ADC1 的通道 11，采用 DMA 的方式操作。

实验讲解->

首先要添加用的库文件，在工程文件夹下 Fwlib 下我们需添加以下库文件：

```
1.  stm32f10x_gpio.c
2.  stm32f10x_rcc.c
3.  stm32f10x_usart.c
4.  stm32f10x_adc.c
5.  stm32f10x_dma.c
6.  stm32f10x_flash.c
```

还要在 stm32f10x_conf.h 中将相应头文件的注释去掉：

```
1.  /* Uncomment the line below to enable peripheral header file inclusion */
2.  #include "stm32f10x_adc.h"
3.  /* #include "stm32f10x_bkp.h" */
4.  /* #include "stm32f10x_can.h" */
5.  /* #include "stm32f10x_crc.h" */
6.  /* #include "stm32f10x_dac.h" */
7.  /* #include "stm32f10x_dbgmcu.h" */
8.  #include "stm32f10x_dma.h"
9.  /* #include "stm32f10x_exti.h" */
10. #include "stm32f10x_flash.h"
11. /* #include "stm32f10x_fsmc.h" */
12. #include "stm32f10x_gpio.h"
13. /* #include "stm32f10x_i2c.h" */
14. /* #include "stm32f10x_iwdg.h" */
15. /* #include "stm32f10x_pwr.h" */
16. #include "stm32f10x_rcc.h"
17. /* #include "stm32f10x_rtc.h" */
18. /* #include "stm32f10x_sdio.h" */
19. /* #include "stm32f10x_spi.h" */
20. /* #include "stm32f10x_tim.h" */
21. #include "stm32f10x_usart.h"
22. /* #include "stm32f10x_wwdg.h" */
23. /*#include "misc.h"*/  /* High level functions for NVIC and SysTick (add-
    on to CMSIS functions) */
```

配置好所需的库文件之后，我们就从 main 函数开始分析：

```
1.  /**
2.    * @brief  Main program.
3.    * @param  None
4.    * @retval : None
5.    */
6.
7.  int main(void)
8.  {
9.      /* config the sysclock to 72M */
10.   SystemInit();
11.
12.   /* USART1 config */
13.     USART1_Config();
14.
15.     /* enable adc1 and config adc1 to dma mode */
16.     ADC1_Init();
17.
18.     printf("\r\n -------这是一个 ADC 实验------\r\n");
```

```
19.
20.   while (1)
21.   {
22.       ADC ConvertedValueLocal = ADC ConvertedValue; // 读取转换的 AD 值
23.       Delay(0xffffee);                              // 延时
24.       printf("\r\n The current AD value = 0x%04X \r\n", ADC_ConvertedValueLocal
    );
25.   }
26. }
```

系统库函数 SystemInit(); 将系统时钟设置为 72M， USART1_Config(); 配置串口，关于这两个函数的具体讲解可以参考前面的教程，这里不再详述。

ADC1_Init(); 函数使能了 ADC1，并使 ADC1 工作于 DMA 方式。ADC1_Init();这个函数是由我们用户在 adc.c 文件中实现的应用程序：

```
1.  /*
2.   * 函数名：ADC1_Init
3.   * 描述  ：无
4.   * 输入  ：无
5.   * 输出  ：无
6.   * 调用  ：外部调用
7.   */
8.  void ADC1_Init(void)
9.  {
10.     ADC1 GPIO Config();
11.     ADC1_Mode_Config();
12. }
```

ADC1_Init();调用了 ADC1_GPIO_Config();和 ADC1_Mode_Config();这两个函数的作用分别是配置好 ADC1 所用的 I/O 端口、配置它的工作模式为 MDA 模式。

```
1.  /*
2.   * 函数名：ADC1_GPIO_Config
3.   * 描述  ：使能 ADC1 和 DMA1 的时钟，初始化 PC.01
4.   * 输入  ：无
5.   * 输出  ：无
6.   * 调用  ：内部调用
7.   */
8.  static void ADC1_GPIO_Config(void)
9.  {
10.     GPIO_InitTypeDef GPIO_InitStructure;
11.     /* Enable DMA clock */
12.     RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
13.
14.     /* Enable ADC1 and GPIOC clock */
15.     RCC APB2PeriphClockCmd(RCC APB2Periph ADC1 | RCC APB2Periph GPIOC, ENABLE);
16.
17.     /* Configure PC.01  as analog input */
18.     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
19.     GPIO InitStructure.GPIO Mode = GPIO Mode AIN;
20.     GPIO_Init(GPIOC, &GPIO_InitStructure);        // PC1,输入时不用设置速率
21. }
```

代码非常简单，大家就自己花点心思看看吧。这两个函数都用 static 关键字修饰了，都属于内部调用，我们只向其他用户提供了这个 ADC1_Init();接口，先得更方便简洁。

```
/* 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);
  DMA_InitStructure.DMA_PeripheralBaseAddr = ADC1_DR_Address;
  DMA_InitStructure.DMA_MemoryBaseAddr = (u32)&ADC_ConvertedValue;
  DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
  DMA_InitStructure.DMA_BufferSize = 1;
  DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
  DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Disable;
  DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
  DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
  DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
  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);

  /* ADC1 configuration */
  ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
  ADC_InitStructure.ADC_ScanConvMode = ENABLE;
  ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
  ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
  ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfChannel = 1;
  ADC_Init(ADC1, &ADC_InitStructure);

  /* ADC1 regular channel11 configuration */
  ADC_RegularChannelConfig(ADC1, ADC_Channel_11, 1, ADC_SampleTime_55Cycles5);

  /* Enable ADC1 DMA */
  ADC_DMACmd(ADC1, ENABLE);

  /* Enable ADC1 */
  ADC_Cmd(ADC1, ENABLE);

  /* Enable ADC1 reset calibaration register */
  ADC_ResetCalibration(ADC1);
  /* Check the end of ADC1 reset calibration register */
  while(ADC_GetResetCalibrationStatus(ADC1));

  /* Start ADC1 calibaration */
  ADC_StartCalibration(ADC1);
  /* Check the end of ADC1 calibration */
  while(ADC_GetCalibrationStatus(ADC1));

  /* Start ADC1 Software Conversion */
  ADC_SoftwareStartConvCmd(ADC1, ENABLE);
}
```

现在我们再来认识两个变量：

```
// ADC1 转换的电压值通过 MDA 方式传到 flash
extern __IO u16 ADC_ConvertedValue;

// 局部变量，用于存从 flash 读到的电压值
__IO u16 ADC_ConvertedValueLocal;
```

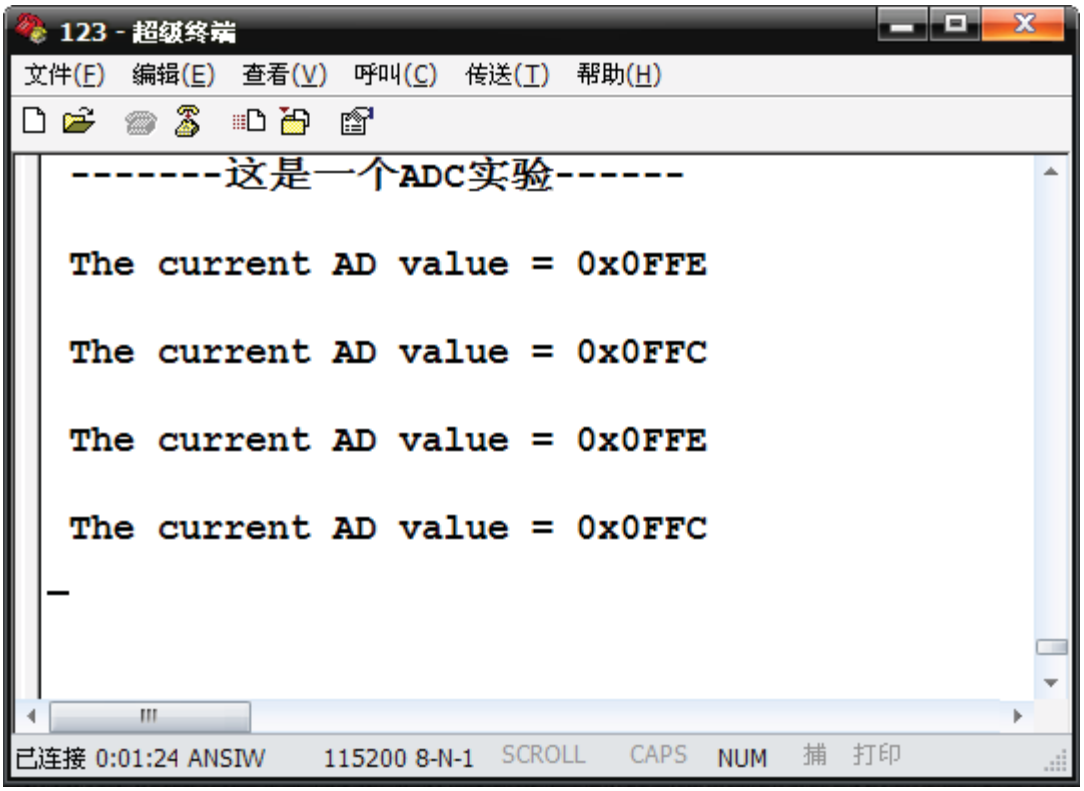
ADC_ConvertedValue 在 adc.c 中定义，ADC_ConvertedValueLocal 在 main.c 中定义，关于这两个变量的作用可看代码的描述。这里要注意一点的是，这两个变量都要用 volatile 关键字来修饰，为的是编译器优化这个变量，这样每次用到这两个变量时都要回到相应变量的内存中去取值，因为这两个变量的值随时都是可变的，而 volatile 字面意思就是“可变的，不确定的”。有关 volatile 关键字的详细用法大家可去查与 C 语言有关的书，这里推荐一个 C 语言的小册子《C 语言深度解剖-陈正冲编著》，里面对 volatile 这个关键字的讲解就非常好，还有其他有关 C 语言的知识也讲得非常好，挺值得大家看看。

主函数的最后以一个无限循环不断地往串口打印 ADC1 转换的电压值：

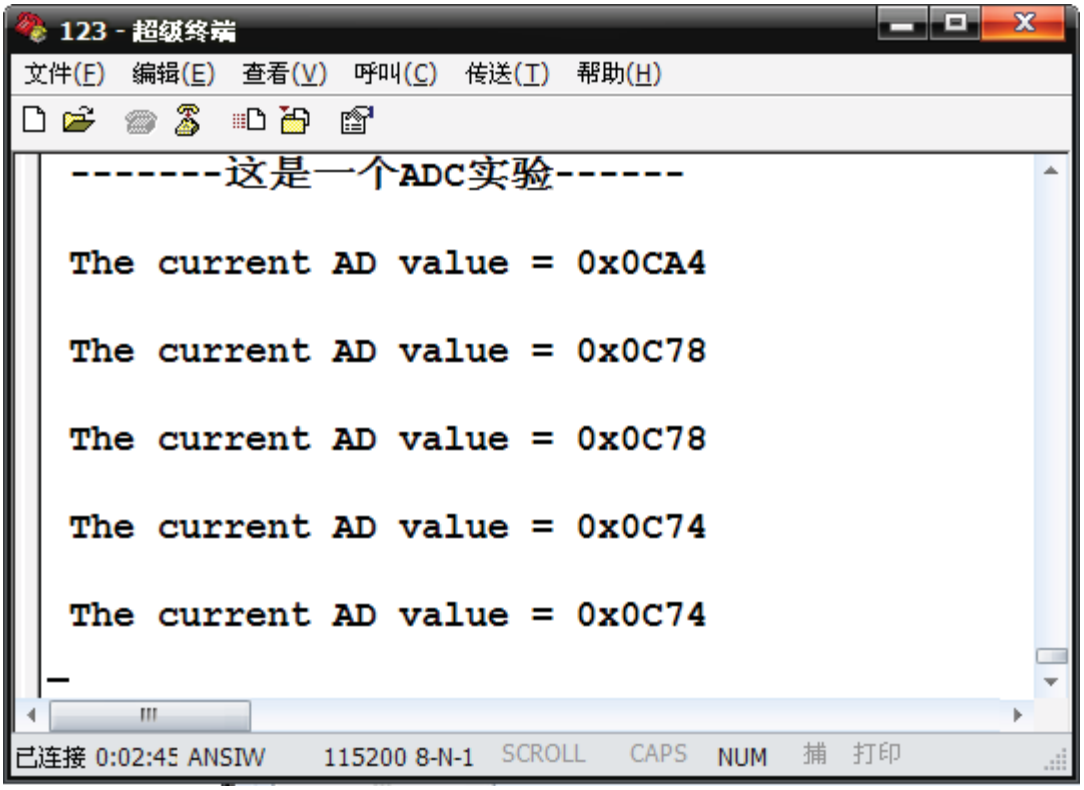
```
1.  while (1)
2.    {
3.      ADC_ConvertedValueLocal = ADC_ConvertedValue; // 读取转换的 AD 值
4.      Delay(0xffffee);                              // 延时
5.      printf("\r\n The current AD value = 0x%04X \r\n", ADC_ConvertedValueLocal);
6.    }
```

实验现象->

将野火 STM32 开发板供电(DC5V)，插上 JLINK，插上串口线(两头都是母的交叉线)，打开超级终端，配置超级终端为 115200 8-N-1，将编译好的程序下载到开发板，即可看到超级终端打印出如下信息：


123 - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
-------这是一个ADC实验------
The current AD value = 0x0FFE
The current AD value = 0x0FFC
The current AD value = 0x0FFE
The current AD value = 0x0FFC
已连接 0:01:24 ANSIW
115200 8-N-1
SCROLL
CAPS
NUM
捕
打印

当旋转开发板开发板上的滑动变阻器时，ADC1 转换的电压值则会改变。板载的是 20K 的精密电阻，旋转的圈数要多点才能看到 AD 值的明显变化。



实验讲解完毕，野火祝大家学习愉快^_^。