ADC 采集例程

实验平台：奋斗版STM32开发板MINI、V2、V2.1、V3、V5
实验内容：板子加电后，通过串口1显示ADC1的通道11的测量结果，该实验学习了基于DMA专递方式的ADC采集软件的编制及控制流程。

预先需要掌握的知识

1 ADC介绍

12位ADC是一种逐次逼近型模拟数字转换器。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/低阀值值。

2 ADC主要特征

- 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
- 自校准
- 带内嵌数据一致的数据对齐
- 通道之间采样间隔可编程
- 规则转换和注入转换均有外部触发选项
- 间断模式
- 双重模式(带2个或以上ADC的器件)
- ADC转换时间：
  - STM32F103xx增强型产品：ADC时钟为56MHz时为1μs(ADC时钟为72MHz为1.17μs)
  - STM32F101xx基本型产品：ADC时钟为28MHz时为1μs(ADC时钟为36MHz为1.55μs)
  - STM32F102xxUSB型产品：ADC时钟为48MHz时为1.2μs
- ADC供电要求：2.4V到3.6V
- ADC输入范围：VREF- ≤ VIN ≤ VREF+
- 规则通道转换期间有DMA请求产生。

3 ADC功能描述

3.1 ADC开关控制

通过设置ADC_CR1寄存器的ADON位可给ADC上电。当第一次设置ADON位时，它将ADC从断电状态下唤醒。 ADC上电延迟一段时间后，再次设置ADON位时开始进行转换。通过清除ADON位可以停止转换，并将ADC置于断电模式。在这个模式中，ADC几乎不耗电(仅几个μA)。

3.2 ADC时钟

由时钟控制器提供的ADCCLK时钟和PCLK2(APB2时钟)同步。RCC控制器为ADC时钟提供一个专用的可编程预分频器。

3.3 通道选择

有16个多路通道。可以把转换分成两组：规则的和注入的。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。例如，可以如下顺序完成转换：通道3、通道8、通道2、通道2、通道0、通道2、通道2、通道15。

● 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写入ADC_SQR1寄存器的L[3:0]位中。

● 注入组由多达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目必须写入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，当前的转换被清除，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**/ VREFINT**内部通道

温度传感器和通道ADCx_IN16相连接，内部参照电压VREFINT和ADCx_IN17相连接。可以按注入或规则通道对这两个内部通道进行转换。

注意： 传感器和*VREFINT*只能出现在主*ADC1*中。

3.4 单次转换模式

单次转换模式下，ADC只执行一次转换。该模式既可通过设置ADC_CR2寄存器的ADON位(只适用于规则通道)启动也可通过外部触发启动(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位的ADC_DRJ1寄存器中
- JEOC(注入转换结束)标志被设置
- 如果设置了JEOCIE位，则产生中断。 然后ADC停止。

3.5 连续转换模式

在连续转换模式中，当前面ADC转换一结束马上就启动另一次转换。此模式可通过外部触发启动或通过设置ADC_CR2寄存器上的ADON位启动，此时CONT位是1。

每个转换后：

● 如果一个规则通道被转换：
- 转换数据被储存在16位的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位的ADC_DRJ1寄存器中
- JEOC(注入转换结束)标志被设置
- 如果设置了JEOCIE位，则产生中断。

3.6 模拟看门狗

如果被ADC转换的模拟电压低于低阀值或高于高阀值，AWD模拟看门狗状态位被设置。这些阀值位于在ADC_HTR和ADC_LTR寄存器的最低12个有效位中。通过设置ADC_CR1寄存器

的AWDIE位以允许产生相应中断。 阀值独立于由ADC_CR2寄存器上的ALIGN位选择的数据对齐模式。 通过配置ADC_CR1寄存器，模拟看门狗可以作用于1个或多个通道。 模拟看门狗警戒区

3.8 扫描模式:

此模式用来扫描一组模拟通道。

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(对规则通道)或ADC_JSQR(对注入通道)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，转换不会在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，在每次EOC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。

3.9 注入通道管理

触发注入

清除ADC_CR1寄存器的JAUTO位，并且设置SCAN位，即可使用触发注入功能。

1. 利用外部触发或通过设置ADC_CR2寄存器的ADON位，启动一组规则通道的转换。

2. 如果在规则通道转换期间产生一外部注入触发，当前转换被复位，注入通道序列被以单次扫描方式进行转换。

3. 然后，恢复上次被中断的规则组通道转换。如果在注入转换期间产生一规则事件，注入转换不会被中断，但是规则序列将在注入序列结束后被执行。图23是其定时图。

注： 当使用触发的注入转换时，必须保证触发事件的间隔长于注入序列。例如：序列长度为28个ADC时钟周期(即2个具有1.5个时钟间隔采样时间的转换)，触发之最小的间隔必须是29个ADC时钟周期。

自动注入

如果设置了JAUTO位，在规则组通道之后，注入组通道被自动转换。这可以用来转换在ADC_SQRx和ADC_JSQR寄存器中设置的多至20个转换序列。

在此模式里，必须禁止注入通道的外部触发。 如果除JAUTO位外还设置了CONT位，规则通道至注入通道的转换序列被连续执行。

3.10 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)，此转换是ADC_SQRx寄存器所选择的转换序列的一部分。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：转换的序列为 0，1，2

第二次触发：转换的序列为 3，6，7

第三次触发：转换的序列为 9，10，并产生EOC事件

第四次触发：转换的序列 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。当所有子组被转换完成，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子组的通道 0、1和2。

注入组

此模式通过设置ADC_CR1寄存器的JDISCEN位激活。在一个外部触发事件后，给模式按序转换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。

例子：

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，并且产生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，下个触发启动第1个注入通道的转换。在上述例子中，第四个触发重新转换第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 校准

ADC有一个内置自校准模式。校准可大幅减小因内部电容器组的变化而造成的准精度误差。在校准期间，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，校准码储存在ADC_DR中。

注意：

1 建议在每次上电后执行校准。

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。

5 数据对齐

ADC_CR2寄存器中的ALIGN位选择转换后数据储存的对齐方式。数据可以左对齐或右对齐。 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，不需减去偏移值，因此只有12个位有效。

6 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]位而更改。每个通道可以以不同的时间采样。总转换时间如下计算：

TCONV = 采样时间+ 12.5个周期

例如：

当ADCCLK=14MHz和1.5周期的采样时间

TCONV = 1.5 + 12.5 = 14周期 = 1μs

7 外部触发转换

转换可以由外部事件触发(例如定时器捕获，EXTI线)。如果设置了EXTTRIG控制位，则外部事件就能够触发转换。EXTSEL[2:0]和JEXTSEL2:0]控制位允许应用程序选择8个可能的事件中的某一个可以触发规则和注入组的采样。

注意： 当外部触发信号被选为ADC规则或注入转换时，只有它的上升沿可以启动转换。

ADC1和ADC2用于规则通道的外部触发

触发源	类型	EXTSEL[2:0]
定时器1的CC1输出	片上定时器 的内部信号	000
定时器1的CC2输出		001
定时器1的CC3输出		010
定时器2的CC2输出		011
定时器3的TRGO输出		100
定时器4的CC4输出		101
EXTI线11	外部管脚	110
SWSTART	软件控制位	111

TIM8_TRGO事件只存在于大容量产品 ,对于规则通道，选中EXTI线路11和TIM8_TRGO作为外部触发事件，可以通过设置 ADC1 和 ADC2 的 ADC1_ETRGREG_REMAP 位和 ADC2_ETRGREG_REMAP位实现。

ADC1和ADC2用于注入通道的外部触发

触发源	连接类型	JEXTSEL[2:0]
定时器1的TRGO输出	片上定时器 的内部信号	000
定时器1的CC4输出		001
定时器2的TRGO输出		010
定时器2的CC1输出		011
定时器3的CC4输出		100
定时器4的TRGO输出		101

EXTI线15	外部管脚	110
JSWSTART	软件控制位	111

TIM8_CC4事件只存在于大容量产品,对于规则通道，选中EXTI线路15和TIM8_CC4作为外部触发事件，可以通过设置ADC1和ADC2的ADC1_ENTRGINJ_REMAP位和ADC2_ ENTRGINJ_REMAP位实现。
ADC3用于规则通道的外部触发

触发源	连接类型	EXTSEL[2:0]
定时器3的CC1输出	片上定时器 的内部信号	000
定时器2的CC3输出		001
定时器1的CC3输出		010
定时器8的CC1输出		011
定时器8的TRGO输出		100
定时器5的CC1输出		101
定时器5的CC3输出		110
SWSTART	软件控制位	111

ADC3用于注入通道的外部触发

触发源	连接类型	EXTSEL[2:0]
定时器1的TRGO输出	片上定时器 的内部信号	000
定时器1的CC4输出		001
定时器4的CC3输出		010
定时器8的CC2输出		011
定时器8的CC4输出		100
定时器5的TRGO输出		101
定时器5的CC4输出		110
JSWSTART	软件控制位	111

软件触发事件可以通过对寄存器ADC_CR2的SWSTART或JSWSTART位置'1'产生。 规则组的转换可以被注入触发打断。

8 DMA请求

因为规则通道转换的值储存在一个唯一的数据寄存器中，所以当转换多个规则通道时需要

使用DMA，这可以避免丢失已经存储在ADC_DR寄存器中的数据。只有在规则通道的转换结束时才产生DMA请求，并将转换的数据从ADC_DR寄存器传输到用户指定的目的地址。

注： 只有ADC1和ADC3拥有DMA功能。由ADC2转化的数据可以通过双ADC模式，利用ADC1的DMA性能来实现。

9 双ADC模式

在有2个或以上ADC的器件中，可以使用双ADC模式(见图27双ADC框图)。 在双ADC模式里，根据ADC1_CR1寄存器中DUALMOD[2:0]位所选的模式，转换的启动可以是ADC1主和ADC2从的交替触发或同时触发。

注意： 在双ADC模式里，当转换配置成由外部事件触发时，用户必须将其设置成仅触发主ADC，从ADC设置成软件触发，这样可以防止意外的触发从转换。但是，主和从ADC的外部触发必须同时被激活。

共有6种可能的模式：

— 同时注入模式
— 同时规则模式
— 快速交替模式
— 慢速交替模式
— 交替触发模式
— 独立模式

注：外部触发信号作用于ADC2，但在本图中没有显示。	* 某些双ADC模式中，在完整的32位ADC1数据寄存器(ADC1_DR)中包含了ADC1和ADC2的规则转换数据。

还有可以用下列方式组合使用上面的模式：

— 同时注入模式+同时规则模式
— 同时规则模式+交替触发模式
— 同时注入模式+交替模式

注意： 在双ADC模式里，为了从主数据寄存器上读取从转换数据，DMA位必须被使能，即使并不用它来传输规则通道数据。

9.1 同步注入模式

此模式转换一个注入通道组。外部触发源来自ADC1的注入组多路器(由ADC1_CR2寄存器的JEXTSEL[2:0]选择)。给ADC2提供同步触发。

注意： 不要在2个ADC上转换相同的通道(如果转换两个ADC的相同通道，不可能提供重叠的采样时间)。

在ADC1或ADC2的转换结束时：

● 转换的数据存储在每个ADC接口的ADC_JDRx寄存器中。
● 当所有ADC1/ADC2注入通道都被转换时，产生JEOC中断(若任一ADC接口开放了中断)。

注： 在同步模式中，必须转换具有相同长度的序列，或保证触发的间隔比2个序列中较长的序列长，否则当较长序列的转换还未完成时，具有较短序列的ADC转换会被重启。

9.2 同步规则模式

此模式在规则通道组上执行。外部触发源来自ADC1的规则组多路器(由ADC1_CR2寄存器的EXTSEL[2:0]选择)。给ADC2提供同步触发。

注意： 不要在2个ADC上转换相同的通道(如果转换两个ADC的相同通道，不可能提供重叠的采样时间)。

在ADC1或ADC2的转换结束时：

● 产生一个32位DMA传输请求(如果设置了DMA位)，传输到SRAM的32位ADC1_DR寄存器的上半个字包含ADC2的转换数据，低半个字包含ADC1的转换数据。
● 当所有ADC1/ADC2规则通道都被转换完时，产生EOC中断(如果任一ADC接口开放了中断的话)。

注： 在同步规则模式中，必须转换具有相同长度的序列，或保证触发的间隔比2个序列中较长的序列长，否则当较长序列的转换还未完成时，具有较短序列的ADC转换会被重启。

9.3 快速交替模式

此模式只适用于规则通道组(通常一个通道)。外部触发源来自ADC1的规则通道多路器。外部触发产生后：

● ADC2立即启动并且
● ADC1在延迟7个ADC时钟周期后启动

如果同时设置了ADC1和ADC2的CONT位，所选的两个ADC规则通道将被连续地转换。ADC1产生一EOC中断后(由EOCIE使能)，产生一个32位的DMA传输请求(如果设置了DMA位)，ADC1_DR寄存器的32位数据被传输到SRAM，ADC1_DR的上半个字包含ADC2的转换数据，低半个字包含ADC1的转换数据。

注意： 最大允许采样时间<7个ADCCLK周期，避免ADC1和ADC2转换相同通道时发生两个采样周期的重叠。 \

9.4 慢速交替模式

此模式只适用于规则通道组(通常一个通道)。外部触发源来自ADC1的规则通道多路器。外部触发产生后：

● ADC2立即启动并且
● ADC1在延迟14个ADC时钟周期后启动
● 在延迟第二个14个ADC周期后ADC2再次启动，如此循环。

注意： 最大允许采样时间<14个ADCCLK周期，以避免和下个转换重叠。 ADC1产生EOC中断后(由EOCIE使能)，产生一个32位的DMA传输请求(如果设置了DMA位)，ADC1_DR寄存器的32位数据被传输到SRAM，ADC1_DR的上半个字包含ADC2的转换数据，低半个字包含ADC1的转换数据。 在28个ADC时钟周期后自动启动新的ADC2转换。 在这个模式里不能设置CONT位，因为它将连续转换所选择的规则通道。

注意： 应用程序必须确保当使用交替模式时，不能有注入通道的外部触发产生。

9.5 交替触发模式

此模式只适用于注入通道组。外部触发源来自ADC1的注入通道多路器。

● 当第一个触发产生时，ADC1上的所有注入组通道被转换。
● 当第二个触发到达时，ADC2上的所有注入组通道被转换。
● 如此循环……

如果允许产生JEOC中断，在所有ADC1注入组通道转换后产生一个JEOC中断。 如果允许产生JEOC中断，在所有ADC2注入组通道转换后产生一个JEOC中断。 当所有注入组通道都转换完后如果产生另一个外部触发，交替触发处理从转换ADC1注入组通道重新开始。

如果ADC1和ADC2上同时使用注入间断模式：

● 当第一个触发产生时，ADC1上的第一个注入通道被转换。

● 当第二个触发到达时，ADC2上的第一个注入通道被转换。
● 如此循环……

如果允许产生JEOC中断，在所有ADC1注入组通道转换后产生一个JEOC中断。 如果允许产生JEOC中断，在所有ADC2注入组通道转换后产生一个JEOC中断。 当所有注入组通道都转换完后如果产生另一个外部触发，重新开始。

9.6 独立模式

此模式里，双ADC同步不工作，每个ADC接口独立工作。

9.7 混合的规则/注入同步模式

有可能中断规则组同步转换以启动注入组的同步转换。

注： 在混合的规则/注入同步模式中，必须转换具有相同长度的序列，或保证触发的间隔比2个序列中较长的序列长，否则当较长序列的转换还未完成时，具有较短序列的ADC转换会被重启。

9.8 混合的同步规则+交替触发模式

有可能中断规则组同步转换以启动注入组交替触发转换。图34显示了一个规则同步转换被交替触发所中断。 注入交替转换在注入事件到达后立即启动。如果规则转换已经在运行，为了在注入转换后确保同步，所有的ADC(主和从)的规则转换被停止，并在注入转换结束时同步恢复。

注： 在混合的同步规则+交替触发模式中，必须转换具有相同长度的序列，或保证触发的间隔比2个序列中较长的序列长，否则当较长序列的转换还未完成时，具有较短序列的ADC转换会被重启。

9.9 混合同步注入+交替模式

可以用一个注入事件来中断一个交替转换。这种情况下，交替转换被中断，注入转换被启动，在注入序列转换结束时，交替转换被恢复。0是这种情况的一个例子。

注： 当ADC时钟预分频系数设置为4时，交替模式不会均匀地分配采样时间，采样间隔是8个ADC时钟周期与6个ADC时钟周期轮替，而不是均匀的7个ADC时钟周期。

9.10 温度传感器

温度传感器可以用来测量器件周围的温度(TA)。 温度传感器在内部和ADCx_IN16输入通道相连接，此通道把传感器输出的电压转换成数字值。温度传感器模拟输入推荐采样时间是17.1μs。 当没有被使用时，传感器可以置于关电模式。

注意：必须设置TSVREFE位激活内部通道：ADCx_IN16(温度传感器)和ADCx_IN17(VREFINT)的转换。

主要特征

● 支持的温度范围：-40到125度
● 精确度：+/- 1.5° C

1. 选择ADCx_IN16输入通道
2. 选择采样时间大于2.2 μs
3. 设置ADC控制寄存器2(ADC_CR2)的TSVREFE位，以唤醒关电模式下的温度传感器
4. 通过设置ADON位启动ADC转换(或用外部触发)
5. 读ADC数据寄存器上的VSENSE 数据结果
6. 利用下列公式得出温度

温度(° C) = {(V25 - VSENSE) / Avg_Slope} + 25
这里：

V25 = VSENSE在25° C时的数值

Avg_Slope ＝ 温度与VSENSE曲线的平均斜率(单位为mV/° C 或 μV/ ° C)

参考电气特性章节中V25 和Avg_Slope的实际值。

注意： 传感器从关电模式唤醒后到可以输出正确水平的VSENSE前，有一个建立时间。ADC在上电后也有一个建立时间，为了缩短延时，应该同时设置ADON和TSVREFE位。

9.11 ADC中断

规则和注入组转换结束时能产生中断，当模拟看门狗状态位被设置时也能产生中断。它们都有独立的中断使能位。

注： ADC1和ADC2的中断映射在同一个中断向量上，而ADC3的中断有自己的中断向量。

ADC_SR寄存器中有2个其他标志，但是它们没有相关联的中断：

- JSTRT(注入组通道转换的启动)
- STRT(规则组通道转换的启动)

中断事件	事件标志	使能控制位
规则组转换结束	EOC	EOCIE
注入组转换结束	JEOC	JEOCIE
设置模拟看门狗状态位	AWD	AWDIE

1. 应用实例

用到的几处需要关注的知识:

DMA：在这个例程用到了ADC转换结果采用DMA传递方式。直接存储器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。无须CPU任何干预，通过DMA数据可以快速地移动。这就节省了CPU的资源来做其他操作。

ADC规则组： STM32，ADC通道选择有两种方式，一种是规则组，一种是注入组，简单来讲就是规则组设置后，可以按照设置的通道顺序对各通道进行依次采集。方便于对多路ADC通道的自动采集。 注入组最多设置4个通道，简单来讲就是需要触发才能采集设置的通道ADC值。 本例程选择了采用规则组，设置了一个通道进行自动采集。

1.1. 设计要求

板子加电后，通过串口1延时间隔显示ADC1的通道11的测量结果。

1.2 硬件电路设计

ADC通道10接口：

V5板----XS8座的pin14（PC1）

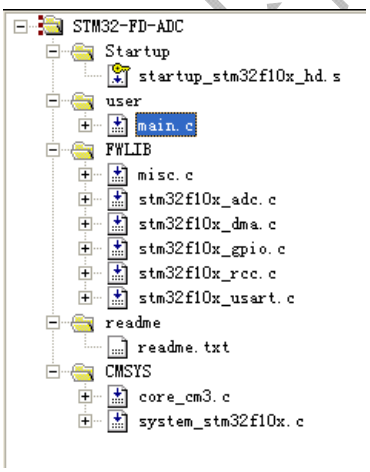
MINI----XS5座的pin9(PC1)

1.3 软件程序设计

根据任务要求，程序内容主要包括：

1. 初始化串口1，用于ADC1通道11采集值的输出显示。
2. ADC1通道11的初始化配置。
3. 间隔时间将采集到的ADC值通过串口1发送出去。

整个工程包含4类源文件：



ASM--startup_stm32f10x_hd.s 由于奋斗板采用的是STM32F103大存储器芯片，因此采用STM32标准库自带的大存储器芯片启动代码，这个文件已经配置好了初始状态，以及中断向量表。可以直接在工程里使用，如果你在以后的应用中采用了中存储器或者小

存储器STM32芯片，可以将启动代码换为startup_stm32f10x_md.s 或者

startup_stm32f10x_ld.s。

FWLIB--stm32f10x_gpio.c ST公司的标准库，包含了关于对通用IO口设置的函数。

stm32f10x_rcc.c ST公司的标准库，包含了关于对系统时钟设置的函数。

stm32f10x_dma.c ST公司的标准库，包含了关于对于DMA设置的函数。

stm32f10x_adc.c ST公司的标准库，包含了和ADC有关的函数。

stm32f10x_USART.c ST公司的标准库，包含了关于对USART设置的函数。

Misc.c ST公司的标准库，包含了关于中断设置的函数。

CMSYS—是关于CORETEX-M3平台的系统函数及定义

USER—main.c 例程的主函数。

```
/*******************************************************************************
* 名    称：int main(void)
* 功    能：主函数
* 入口参数：无
* 出口参数：无
* 说    明：
* 调用方法：
*******************************************************************************/
int main(void)
{
  RCC_Configuration();                  //设置内部时钟及外设时钟使能
  Usart1_Init();                        //串口1初始化
  ADC_Configuration();                  //ADC初始化
  USART_OUT(USART1,"\r\n USART1 print AD_value -------------------------- \r\n");
  while(1)
  {
     if (ticks++ >= 900000) {           //间隔时间显示转换结果
        ticks  = 0;
        Clock1s = 1;
     }
     if (Clock1s) {
       Clock1s = 0;
       USART_OUT(USART1,"The current AD value = %d \r\n", ADC_ConvertedValue);
     }
  }
}
```

RCC_Configuration(void)用于配置系统时钟设置，及外设时钟使能。

```
/*******************************************************************************
* 名    称：void RCC_Configuration(void)
* 功    能：系统时钟配置为72MHZ，  外设时钟配置
* 入口参数：无
* 出口参数：无
* 说    明：
* 调用方法：无
*******************************************************************************/
void RCC_Configuration(void){

  SystemInit();
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 , ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB | RCC_APB2Periph_GPIOC
                       | RCC_APB2Periph_GPIOD| RCC_APB2Periph_GPIOE , ENABLE);
}
```

ADC_Configuration函数用于配置ADC1的通道11，因为只用了ADC1所以采用了ADC独立模式，设置通道11进入规则组，规则组里的通道只有1个，就是通道1，转换用了扫描方式，软件触发，转换结果采用DMA方式传递到2字节长度的缓存区里(ADC_ConvertedValue)，默认的ADCCLK为36MHz，采样周期是55.5+12.5时钟周期，相当于采样时间是间隔(68/36)us。

```
/*******************************************************************************
* 名    称：void ADC_Configuration(void)
* 功    能：ADC 配置函数
* 入口参数：无
* 出口参数：无
* 说    明：
* 调用方法：
*******************************************************************************/
void ADC_Configuration(void)
{
    ADC_InitTypeDef ADC_InitStructure;
    GPIO_InitTypeDef GPIO_InitStructure;
    DMA_InitTypeDef DMA_InitStructure;

    //设置AD模拟输入端口为输入 1路AD 规则通道
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOC, &GPIO_InitStructure);
    /* Enable DMA clock */
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);

   /* Enable ADC1 and GPIOC clock */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 , ENABLE);

    /* DMA channel1 configuration ----------------------------------------------*/
    //使能DMA
    DMA_DeInit(DMA1_Channel1);
    DMA_InitStructure.DMA_PeripheralBaseAddr = ADC1_DR_Address;                    //DMA通道1的地址
    DMA_InitStructure.DMA_MemoryBaseAddr = (u32)&ADC_ConvertedValue;               //DMA传送地址
    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;                             //传送方向
    DMA_InitStructure.DMA_BufferSize = 1;                                          //传送内存大小，100个16位
    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;                        //传送内存地址递增
    DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;    //ADC1转换的数据是16位
    DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;            //传送的目的地址是16位宽度
    DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;                                //循环
    DMA_InitStructure.DMA_Priority = DMA_Priority_High;
    DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
    DMA_Init(DMA1_Channel1, &DMA_InitStructure);

    /* 允许DMA1通道1传输结束中断 */
    //DMA_ITConfig(DMA1_Channel1,DMA_IT_TC, ENABLE);


    //使能DMA通道1
    DMA_Cmd(DMA1_Channel1, ENABLE);


    //ADC配置
    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;  //ADC1工作在独立模式
    ADC_InitStructure.ADC_ScanConvMode = ENABLE;        //模数转换工作在扫描模式（多通道）还是单次（单通道）模式
    ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;  //模数转换工作在扫描模式（多通道）还是单次（单通道）模式
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;//转换由软件而不是外部触发启动
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;//ADC数据右对齐
    ADC_InitStructure.ADC_NbrOfChannel = 1;               //规定了顺序进行规则转换的ADC通道的数目。这个数目的取值范围是1到16
    ADC_Init(ADC1, &ADC_InitStructure);

    /* ADC1 regular channels configuration [规则模式通道配置]*/

    //ADC1 规则通道配置
    ADC_RegularChannelConfig(ADC1, ADC_Channel_11, 1, ADC_SampleTime_55Cycles5);      //通道11采样时间 55.5周期


    //使能ADC1 DMA
    ADC_DMACmd(ADC1, ENABLE);
    //使能ADC1
    ADC_Cmd(ADC1, ENABLE);

    // 初始化ADC1校准寄存器
    ADC_ResetCalibration(ADC1);
    //检测ADC1校准寄存器初始化是否完成
    while(ADC_GetResetCalibrationStatus(ADC1));

    //开始校准ADC1
    ADC_StartCalibration(ADC1);
    //检测是否完成校准
    while(ADC_GetCalibrationStatus(ADC1));

    //ADC1转换启动
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);
}
```

void Usart1_Init(void)用于配置串口1。

```
/*******************************************************************************
* 名    称：void Usart1_Init(void)
* 功    能：串口1初始化函数
* 入口参数：无
* 出口参数：无
* 说    明：
* 调用方法：无
*******************************************************************************/
void Usart1_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;
  USART_InitTypeDef USART_InitStructure;

  RCC_APB2PeriphClockCmd( RCC_APB2Periph_USART1 , ENABLE);    //使能串口1时钟

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;                   //USART1 TX
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;             //复用推挽输出
  GPIO_Init(GPIOA, &GPIO_InitStructure);                      //A端口

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;                  //USART1 RX
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;       //复用开漏输入
  GPIO_Init(GPIOA, &GPIO_InitStructure);                      //A端口

  USART_InitStructure.USART_BaudRate = 115200;                        //速率115200bps
  USART_InitStructure.USART_WordLength = USART_WordLength_8b;         //数据位8位
  USART_InitStructure.USART_StopBits = USART_StopBits_1;              //停止位1位
  USART_InitStructure.USART_Parity = USART_Parity_No;                 //无校验位
  USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;   //无硬件流控
  USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;                    //收发模式

  /* Configure USART1 */
  USART_Init(USART1, &USART_InitStructure);                           //配置串口参数函数
   /* Enable the USART1 */
  USART_Cmd(USART1, ENABLE);

}
```

2 运行过程

按  编译工程， 完成后会提示如下。

```
Build target 'STM32-FD-ADC'
linking...
Program Size: Code=2376 RO-data=336 RW-data=28 ZI-data=1028
FromELF: creating hex file...
".\obj\STM32-FD-ADC.axf" - 0 Error(s), 0 Warning(s).
```

2.1. 通过JLINK V8或者串口将代码写入板子，具体的烧写步骤，参考奋斗板文档目录下的《奋斗版STM32开发板JTAG下载步骤》或者《奋斗版STM32开发板串口下载步骤》。

2.2使用串口线将板子与PC连接起来， 运行串口助手软件。 设置如下：

2.3 通过USB线给板子加电。串口助手软件会收到如下的信息。

用户可根据例程编写自己的应用例程。