STM32的模拟接口

内容提要

- 模拟输入输出系统
- AD转换概念和原理
- STM32的ADC接口
- STM32的DAC接口

1 STM32的ADC接口

- 拥有1～3个12位ADC
  - 共有21个模拟输入通道，转换范围0—3.6V
  - 主ADC1还可以测量2个内部信号源（温度传感器、内部参考电压）
- 各个通道可以采用单次、连续、扫描或间断模式将模拟量转换为数字量
  - 12位转换结果保存于16位数据寄存器中
  - 可以选择左对齐或右对齐方式存储
- 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/低阈值

- 分辨率
  - 12位分辨率。不能直接测量负电压，所以没有符号位，即其最小量化单位LSB=$V_{ref+}/2^{12}$
- 转换时间
  - 转换时间可编程，采样一次至少要用14个ADC时钟周期，而ADC的时钟频率最高为14Mhz，也就是说，它的采样时间最短为1μs。足以胜任中、低频数字示波器的采样工作。
- ADC工作原理
  - 逐次比较型ADC

ADC相关引脚和参考电压

引脚名称	信号类型	说明
V_{REF+}	输入， 模拟参考正极	使用的高端/正极参考电压， $2.4V \le V_{REF+} \le V_{DDA}$
V_{DDA}	输入， 模拟电源	等效于V_{DD} 的模拟电源， $2.4V \le V_{DDA} \le V_{DD}$（3.6V）
V_{REF-}	输入， 模拟参考负极	ADC 使用的低端/负极参考电压，$V_{REF-} = V_{SSA}$
V_{SSA}	输入， 模拟电源地	等效于V_{SS}的模拟电源地
ADCx_IN[15:0]	模拟输入信号	16个模拟输入通道

注：一般负参考电压接地，正参考电压接3.3V。
模拟输入电压不要超过3.3V

VCC、VDD和VSS的含义

- VCC：C=circuit 表示电路的意思，即接入电路的电压；
  VDD：D=device 表示器件的意思，即器件内部的工作电压；
  VSS：S=series 表示公共连接的意思，通常指电路公共接地端电压。
- 说明：
  1、对于数字电路来说，VCC是电路的供电电压,VDD是芯片的工作电压（通常Vcc>Vdd），VSS是接地点。
  2、有些IC既有VDD引脚又有VCC引脚，说明这种器件自身带有电压转换功能。
  3、在场效应管（或COMS器件）中，VDD为漏极，VSS为源极，VDD和VSS指的是元件引脚，而不表示供电电压。

STM32F10X系列芯片ADC各通道与引脚对应关系

	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	PF6
通道5	PA5	PA5	PF7
通道6	PA6	PA6	PF8
通道7	PA7	PA7	PF9
通道8	PB0	PB0	PF10
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		

21个外部输入通道：

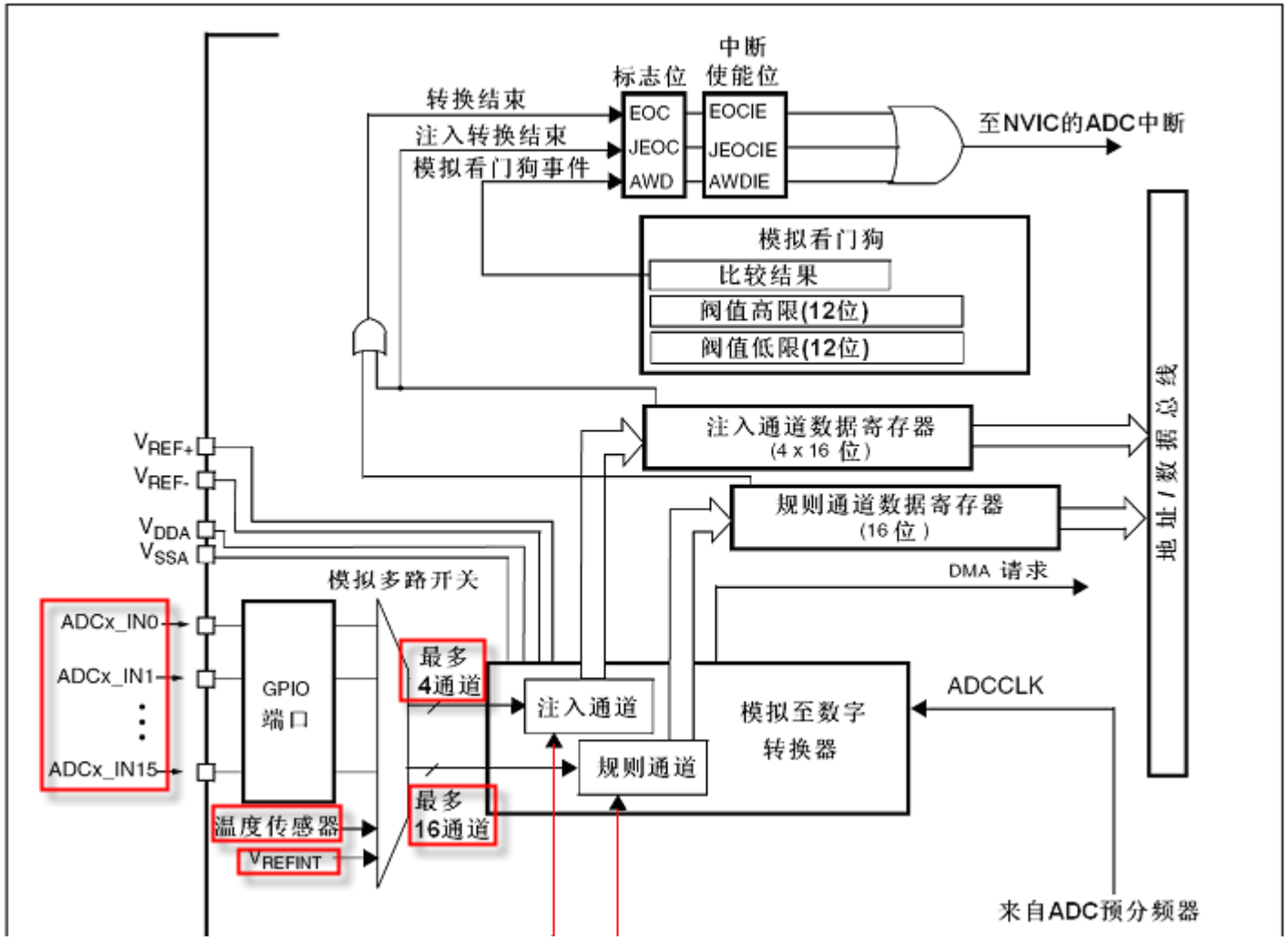
PA0—PA7

PB0—PB1

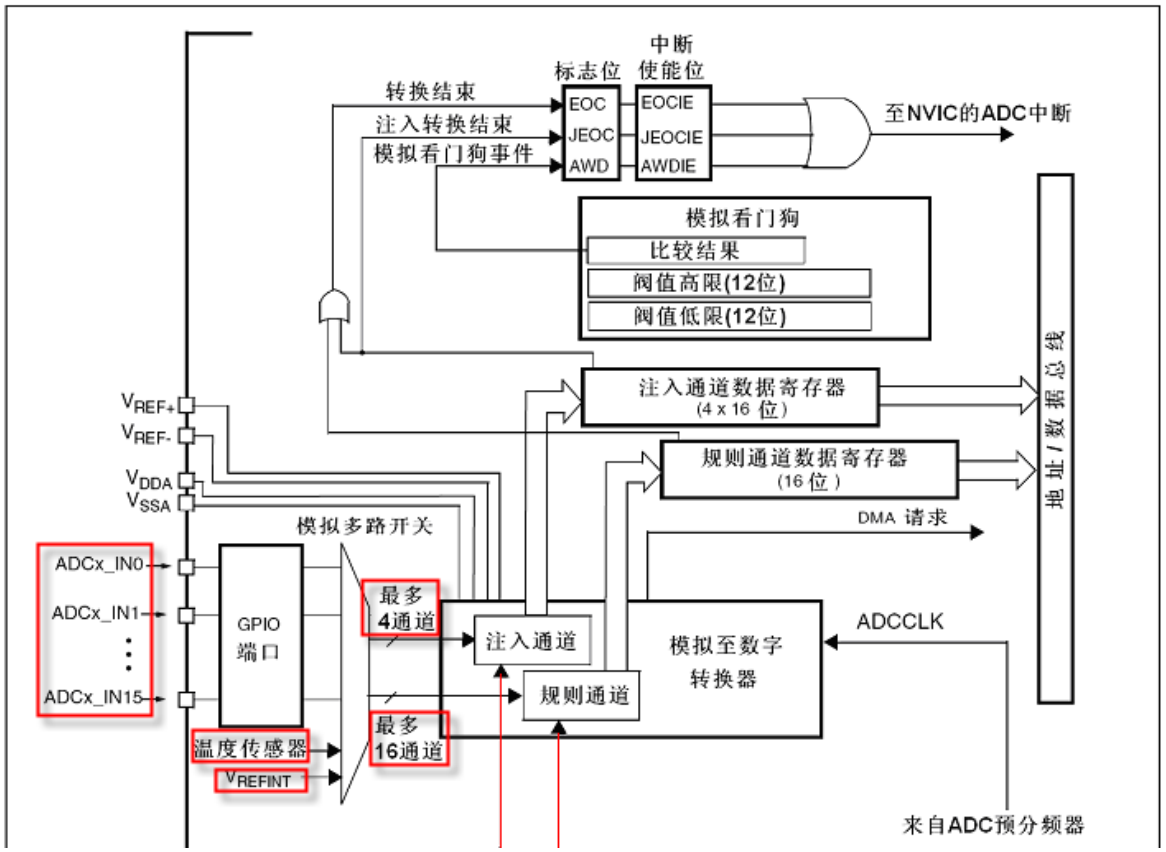
PC0—PC5

PF6—PF10

ADC模块框图



- 图中省略了用于产生ADC转换的触发信号，即图中的两根红线所示。
- 如果使用软件触发ADC转换，将和触发信号部分没有关系。

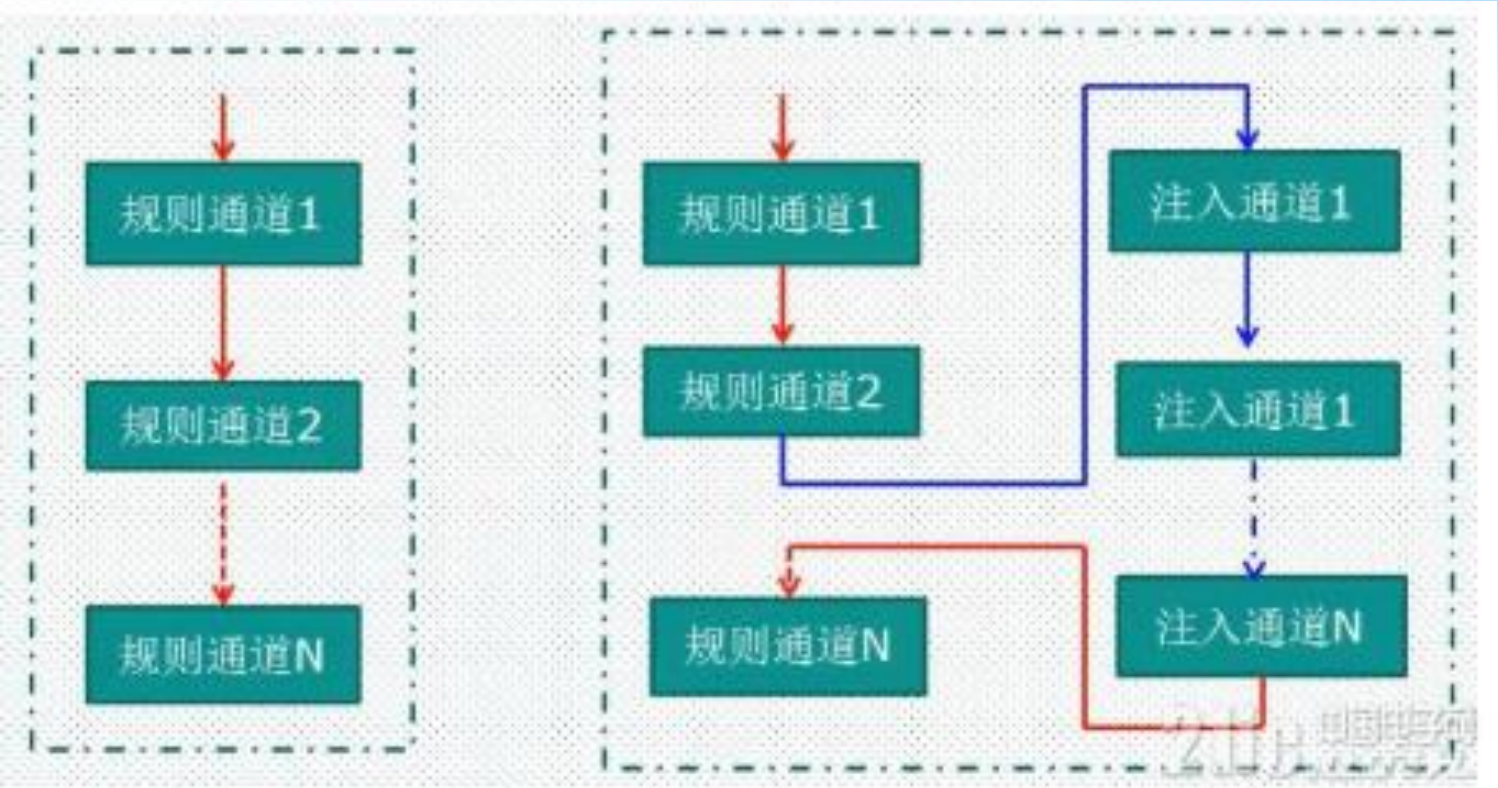


- 每个ADC具有16个模拟输入通道ADCx_IN0~ADCx_IN15
- 最多4个通道连接到注入通道，最多16个通道连接到规则通道。
- ADC1除了支持外部模拟输入，还可以支持2个内部信号源（温度传感器和内部参考电压）

规则通道组和注入通道组

- STM32的ADC通道分为规则组和注入组。因为ADC转换模块只有一个ADC功能核心，它能够支持这么多通道的数据转换，用的是分时复用的方法。分组的目的是为了赋予特定的ADC通道优先权。
- 比如，ADCx_IN2被分配到规则组，ADCx_IN3被分配到注入组，在IN2通道进行数据转换的过程中，外部信号触发了IN3通道的转换，则ADC功能核心将暂停IN2的转换，转去执行IN3的转换，完成转换后在回来执行IN2的转换。
- 由此可知，注入组的通道具有优先转换权，可以打断规则组通道正在进行的转换。
- 规则通道转换的结果保存在规则通道数据寄存器中；注入通道转换的结果保存在注入通道数据寄存器中。

规则通道组和注入通道组



- 规则通道相当于主程序，而注入通道呢，就相当于中断，中断是可以打断主程序的执行的。类似的，注入通道可以打断规则通道的转换，在注入通道被转换完成之后，规则通道才得以继续转换。

规则组和注入组的运用举例：温度监测

- **任务需求**：室外（院子里）装5个温度探头，室内装3个温度探头；需要时刻监视室外温度，但偶尔想看看室内的温度。

无分组划分的设计：
没有按键按下时，将AD循环扫描的通道配置指向室外的5个探头并显示其AD结果；按下特定按钮后，需要重新配置AD循环扫描的通道，指向室内的3个探头；按键释放后需再次配置AD循环扫描的通道，指向室外的5个探头。

使用规则组和注入组的设计：
在初始化阶段分别设置好不同的转换组（规则通道组循环扫描室外的5个探头，注入通道组循环扫描室内的3个探头）。系统运行阶段，没有按键按下时，规则通道组正常转换，显示结果为室外温度；特定按键按下时，启动注入通道，显示结果为室内温度；释放按键后，系统返回到显示规则通道AD结果的状态。

评价：前者无需变更AD循环扫描的配置，可实现两个任务互不干扰和快速切换，后者在切换时需要频繁变更AD循环扫描的配置。

- STM32的ADC的各通道可以单次、连续、扫描或者间断模式执行。
  - 单次转换模式下，ADC只执行一次转换。转换完成后不会自动转到另外一个通道的转换，需要手动设置才能启动
  - 在连续转换模式中，当前一次ADC转换结束后马上就启动另一次转换。
  - 扫描模式用来扫描一组模拟通道。ADC扫描所有被规则通道组/注入通道组选中的所有通道,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。一般通过DMA读取数据。

- 间断模式可以用来执行一个短序列的n次转换（n≤8），此转换是规则组/注入组中转换序列的一部分

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

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。

中断事件	事件标志	中断允许控制位
规则组转换结束	EOC	EOCIE
注入组转换结束	JEOC	JEOCIE
模拟看门狗状态位置位	AWD	AWDIE

ADC的DMA请求

- 规则组通道的转换结果只有一个数据寄存器
  - 多个规则通道的转换，有必要使用DMA方式处理数据，以免数据溢出
- 一个规则通道转换结束，就可产生DMA请求
  - 将规则组数据寄存器ADC_DR的数据利用DMA方式传送到用户事先选定的目的位置
- 只有ADC1（和 ADC3）能够产生DMA请求
  - ADC2可以在双ADC模式中使用ADC1的DMA请求

- ADC的时钟配置

时钟配置寄存器(RCC_CFGR)

偏移地址: 0x04

复位值: 0x0000 0000

位15:14	**ADCPRE[1:0]**：ADC预分频 (ADC prescaler) 由软件置'1'或清'0'来确定ADC时钟频率 00：PCLK2 2分频后作为ADC时钟 01：PCLK2 4分频后作为ADC时钟 10：PCLK2 6分频后作为ADC时钟 11：PCLK2 8分频后作为ADC时钟

不要让ADC时钟超过14Mhz，否则可能不准。例如：系统时钟是72M，就要选择6分频或8分频，6分频为12M。

ADC控制寄存器 1(ADC_CR1)

地址偏移：0x04

复位值：0x0000 0000

位8	**SCAN**：扫描模式 (Scan mode) 该位由软件设置和清除，用于开启或关闭扫描模式。在扫描模式中，转换由ADC_SQRx或ADC_JSQRx寄存器选中的通道。 0：关闭扫描模式； 1：使用扫描模式。 注：如果分别设置了EOCIE或JEOCIE位，只在最后一个通道转换完毕后才会产生EOC或JEOC中断。
位7	**JEOCIE**：允许产生注入通道转换结束中断 (Interrupt enable for injected channels) 该位由软件设置和清除，用于禁止或允许所有注入通道转换结束后产生中断。 0：禁止JEOC中断； 1：允许JEOC中断。当硬件设置JEOC位时产生中断。
位5	**EOCIE**：允许产生EOC中断 (Interrupt enable for EOC) 该位由软件设置和清除，用于禁止或允许转换结束后产生中断。 0：禁止EOC中断； 1：允许EOC中断。当硬件设置EOC位时产生中断。

- 在扫描模式下，由ADC_SQRx或ADC_JSQRx寄存器选中的通道被转换，如果设置了EOCIE或JEOCIE，在最后一个通道转换完毕后才会产生EOC或JEOC中断。

ADC_CR1寄存器

位19:16	**DUALMOD[3:0]**：双模式选择 (Dual mode selection) 软件使用这些位选择操作模式。 0000：独立模式 0001：混合的同步规则+注入同步模式 0010：混合的同步规则+交替触发模式 0011：混合同步注入+快速交叉模式 0100：混合同步注入+慢速交叉模式 0101：注入同步模式 0110：规则同步模式 0111：快速交叉模式 1000：慢速交叉模式 1001：交替触发模式 注： 在ADC2和ADC3中这些位为保留位 在双模式中，改变通道的配置会产生一个重新开始的条件，这将导致同步丢失。建议在进行任何配置改变前关闭双模式。

- 在有2个或以上ADC模块的产品中，可以使用双ADC模式。**一般选择独立模式，此时每个ADC接口独立工作。**

ADC_CR2寄存器

ADC控制寄存器 2(ADC_CR2)

地址偏移：0x08

复位值：0x0000 0000

- 因为STM32的ADC为12位，寄存器为16位，所以需要选择对齐方式。
- **ADC_CR2寄存器中的ALIGN位（位11）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。**

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

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

ADC_CR2寄存器

位19:17	**EXTSEL[2:0]**：选择启动规则通道组转换的外部事件 (External event select for regular group) 这些位选择用于启动规则通道组转换的外部事件 ADC1和ADC2的触发配置如下 000：定时器1的CC1事件　　100：定时器3的TRGO事件 001：定时器1的CC2事件　　101：定时器4的CC4事件 010：定时器1的CC3事件　　110：EXTI线11/ TIM8_TRGO事件，仅大容量产品具有TIM8_TRGO功能 011：定时器2的CC2事件　　111：SWSTART ADC3的触发配置如下 000：定时器3的CC1事件　　100：定时器8的TRGO事件 001：定时器2的CC3事件　　101：定时器5的CC1事件 010：定时器1的CC3事件　　110：定时器5的CC3事件 011：定时器8的CC1事件　　111：SWSTART

- 如果选择软件转换，位19：17应设置为111

ADC_CR2寄存器

位1	**CONT**：连续转换 (Continuous conversion) 该位由软件设置和清除。如果设置了此位，则转换将连续进行直到该位被清除。 0：单次转换模式； 1：连续转换模式。

位0	**ADON**：开/关A/D转换器 (A/D converter ON / OFF) 该位由软件设置和清除。当该位为'0'时，写入'1'将把ADC从断电模式下唤醒。 当该位为'1'时，写入'1'将启动转换。应用程序需注意，在转换器上电至转换开始有一个延迟t_{STAB}，参见图25。 0：关闭ADC转换/校准，并进入断电模式； 1：开启ADC并启动转换。 注：如果在这个寄存器中与ADON一起还有其他位被改变，则转换不被触发。这是为了防止触发错误的转换。

可编程的通道采样时间（转换时间）

- 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寄存器种SMPx[2:0]位更改，每个通道可以分别用不同的时间采样。

总转换时间计算如下：

Tconv=采样时间+12.5个周期

例如：

当ADC_CLK=14MHz时，采样时间为1.5个周期，则Tconv=1.5+12.5=14周期=1μs

- 即：最小的转换时间为1μs

ADC 采样时间寄存器 1(ADC_SMPR1)

地址偏移：0x0C

复位值：0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留								SMP17[2:0]			SMP16[2:0]			SMP15[2:1]	
					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMP15_0	SMP14[2:0]			SMP13[2:0]			SMP12[2:0]			SMP11[2:0]			SMP10[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位31:24	保留。必须保持为0。
位23:0	**SMPx[2:0]**：选择通道x的采样时间 (Channel x Sample time selection) 这些位用于独立地选择每个通道的采样时间。在采样周期中通道选择位必须保持不变。 000：1.5周期　　100：41.5周期 001：7.5周期　　101：55.5周期 010：13.5周期　　110：71.5周期 011：28.5周期　　111：239.5周期 注：ADC1的模拟输入通道16和通道17在芯片内部分别连到了温度传感器和V_{REFINT}。 ADC2的模拟输入通道16和通道17在芯片内部连到了Vss。 ADC3模拟输入通道14、15、16、17与Vss相连

- ADC每个通道都可以独立设置采样时间。
- ADC的时钟已经由RCC_CFGR寄存器设置。
- ADC_SMPR1寄存器用来设置通道10到通道17的采样时间。

ADC 采样时间寄存器 2(ADC_SMPR2)

地址偏移：0x10

复位值：0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留		SMP9[2:0]			SMP8[2:0]			SMP7[2:0]			SMP6[2:0]			SMP5[2:1]	
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMP5_0		SMP4[2:0]			SMP3[2:0]			SMP2[2:0]			SMP1[2:0]			SMP0[2:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位31:30	保留。必须保持为0。
位29:0	**SMPx[2:0]**：选择通道x的采样时间 (Channel x Sample time selection) 这些位用于独立地选择每个通道的采样时间。在采样周期中通道选择位必须保持不变。 000：1.5周期　　100：41.5周期 001：7.5周期　　101：55.5周期 010：13.5周期　　110：71.5周期 011：28.5周期　　111：239.5周期 注：ADC3模拟输入通道9与Vss相连

- ADC_SMPR2寄存器用来设置通道0到通道9的采样时间。

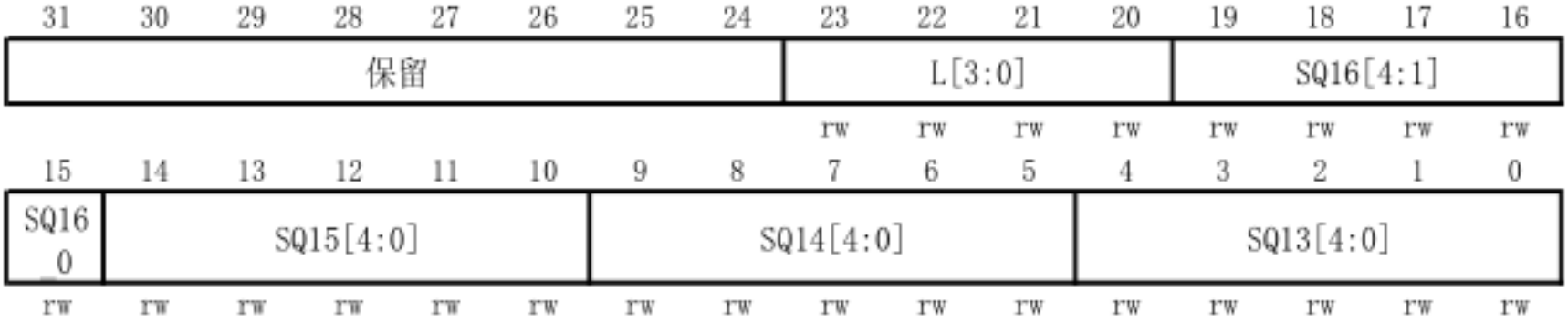
ADC 规则序列寄存器

- ADC 规则序列寄存器ADC_SQR1,ADC_SQR2,ADC_SQR3设置规则通道序列的长度，第一个转换对应哪一个通道，第二个转换对应哪个通道,…,第16个转换对应哪个通道。

ADC 规则序列寄存器 1(ADC_SQR1)

地址偏移：0x2C

复位值：0x0000 0000

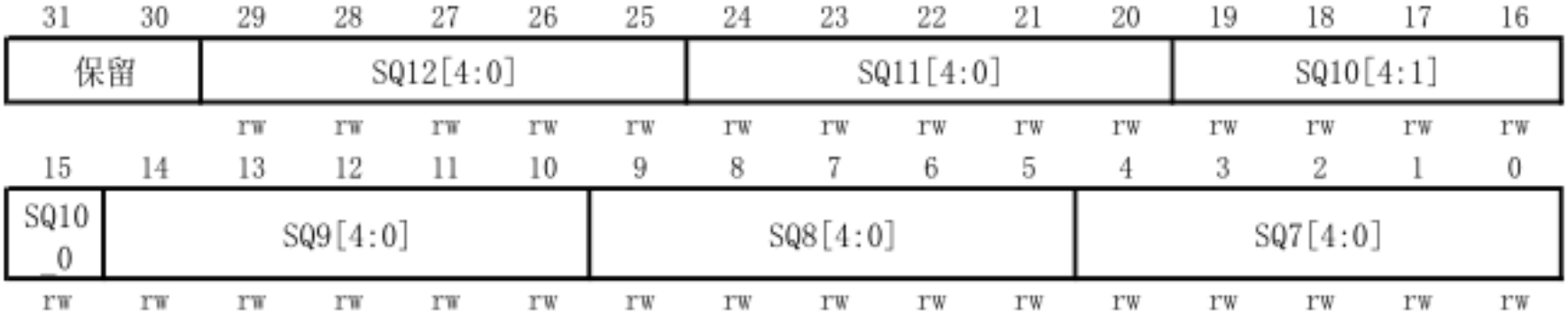


位31:24	保留。必须保持为0。
位23:20	**L[3:0]**：规则通道序列长度 (Regular channel sequence length) 这些位由软件定义在规则通道转换序列中的通道数目。 0000：1个转换 0001：2个转换 ..... 1111：16个转换
位19:15	**SQ16[4:0]**：规则序列中的第16个转换 (16th conversion in regular sequence) 这些位由软件定义转换序列中的第16个转换通道的编号(0~17)。
位14:10	**SQ15[4:0]**：规则序列中的第15个转换 (15th conversion in regular sequence)
位9:5	**SQ14[4:0]**：规则序列中的第14个转换 (14th conversion in regular sequence)
位4:0	**SQ13[4:0]**：规则序列中的第13个转换 (13th conversion in regular sequence)

ADC 规则序列寄存器 2(ADC_SQR2)

地址偏移：0x30

复位值：0x0000 0000

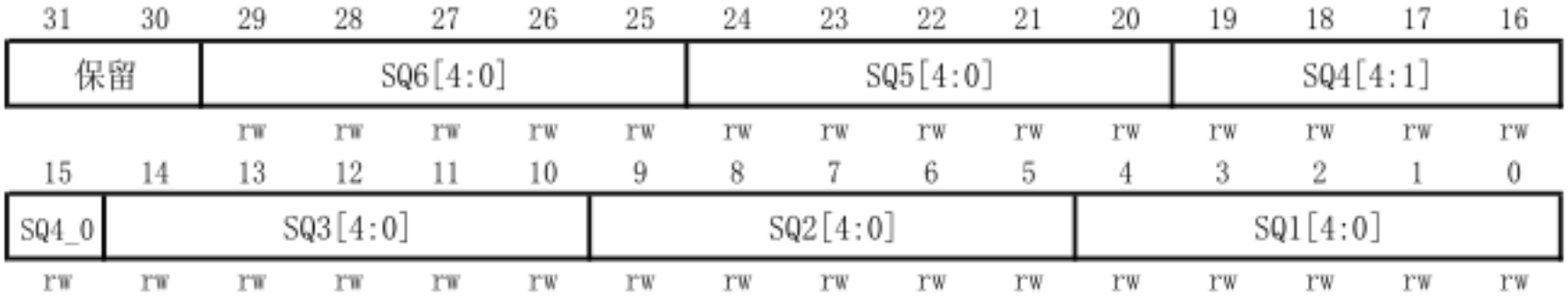


位31:30	保留。必须保持为0。
位29:25	**SQ12[4:0]**：规则序列中的第12个转换 (12th conversion in regular sequence) 这些位由软件定义转换序列中的第12个转换通道的编号(0~17)。
位24:20	**SQ11[4:0]**：规则序列中的第11个转换 (11th conversion in regular sequence)
位19:15	**SQ10[4:0]**：规则序列中的第10个转换 (10th conversion in regular sequence)
位14:10	**SQ9[4:0]**：规则序列中的第9个转换 (9th conversion in regular sequence)
位9:5	**SQ8[4:0]**：规则序列中的第8个转换 (82th conversion in regular sequence)
位4:0	**SQ7[4:0]**：规则序列中的第7个转换 (7th conversion in regular sequence)

ADC 规则序列寄存器 3(ADC_SQR3)

地址偏移：0x34

复位值：0x0000 0000



位31:30	保留。必须保持为0。
位29:25	**SQ6[4:0]**：规则序列中的第6个转换 (6th conversion in regular sequence) 这些位由软件定义转换序列中的第6个转换通道的编号(0~17)。
位24:20	**SQ5[4:0]**：规则序列中的第5个转换 (5th conversion in regular sequence)
位19:15	**SQ4[4:0]**：规则序列中的第4个转换 (4th conversion in regular sequence)
位14:10	**SQ3[4:0]**：规则序列中的第3个转换 (3rd conversion in regular sequence)
位9:5	**SQ2[4:0]**：规则序列中的第2个转换 (2nd conversion in regular sequence)
位4:0	**SQ1[4:0]**：规则序列中的第1个转换 (1st conversion in regular sequence)

ADC注入系列寄存器ADC_JSQR

地址偏移：0x38

复位值：0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留										JL[3:0]		JSQ4[4:1]			
										rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JSQ4_0	JSQ3[4:0]					JSQ2[4:0]				JSQ1[4:0]					
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位31:22	保留。必须保持为0。
位21:20	**JL[1:0]**：注入通道序列长度 (Injected sequence length) 这些位由软件定义在规则通道转换序列中的通道数目。 00：1个转换 01：2个转换 10：3个转换 11：4个转换
位19:15	**JSQ4[4:0]**：注入序列中的第4个转换 (4th conversion in injected sequence) 这些位由软件定义转换序列中的第4个转换通道的编号(0~17)。 注：不同于规则转换序列，如果JL[1:0]的长度小于4，则转换的序列顺序是从(4-JL)开始。例如：ADC_JSQR[21:0] = 10 00011 00011 00111 00010，意味着扫描转换将按下列通道顺序转换：7、3、3，而不是2、7、3。
位14:10	**JSQ3[4:0]**：注入序列中的第3个转换 (3rd conversion in injected sequence)
位9:5	**JSQ2[4:0]**：注入序列中的第2个转换 (2nd conversion in injected sequence)
位4:0	**JSQ1[4:0]**：注入序列中的第1个转换 (1st conversion in injected sequence)

- ADC注入系列寄存器ADC_JSQR设置注入通道序列的长度，第一个转换对应哪一个通道，第二个转换对应哪个通道,…,第4个转换对应哪个通道。

ADC 规则数据寄存器(ADC_DR)

地址偏移：0x4C

复位值：0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ADC2DATA[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

位31:16	**ADC2DATA[15:0]**：ADC2转换的数据 (ADC2 data) - 在ADC1中：双模式下，这些位包含了ADC2转换的规则通道数据。见11.9：双ADC模式 - 在ADC2和ADC3中：不使用这些位。
位15:0	**DATA[15:0]**：规则转换的数据 (Regular data) 这些位为只读，包含了规则通道的转换结果。数据是左对齐或右对齐，如图29和图30所示。

- 保存规则通道的转换结果，一般仅低16位包含有效结果

ADC 注入数据寄存器x（ADC_JDRx）（x= 1..4）

地址偏移：0x3C – 0x48

复位值：0x0000 0000

位31:16	保留。必须保持为0。
位21:20	**JDATA[15:0]**：注入转换的数据 (Injected data) 这些位为只读，包含了注入通道的转换结果。数据是左对齐或右对齐，如图29和图30所示。

- 保存注入通道的转换结果

ADC状态寄存器ADC_SR

地址偏移：0x00

复位值：0x0000 0000

位31:15	保留。必须保持为0。
位4	**STRT**：规则通道开始位 (Regular channel Start flag) 该位由硬件在规则通道转换开始时设置，由软件清除。 0：规则通道转换未开始； 1：规则通道转换已开始。
位3	**JSTRT**：注入通道开始位 (Injected channel Start flag) 该位由硬件在注入通道组转换开始时设置，由软件清除。 0：注入通道组转换未开始； 1：注入通道组转换已开始。
位2	**JEOC**：注入通道转换结束位 (Injected channel end of conversion) 该位由硬件在所有注入通道组转换结束时设置，由软件清除 0：转换未完成； 1：转换完成。
位1	**EOC**：转换结束位 (End of conversion) 该位由硬件在(规则或注入)通道组转换结束时设置，由软件清除或由读取ADC_DR时清除 0：转换未完成； 1：转换完成。
位0	**AWD**：模拟看门狗标志位 (Analog watchdog flag) 该位由硬件在转换的电压值超出了ADC_LTR和ADC_HTR寄存器定义的范围时设置，由软件清除 0：没有发生模拟看门狗事件； 1：发生模拟看门狗事件。

ADC触发选择

- ADC部件需要收到触发信号才开始进行转换
- 对于ADC1和ADC2，触发信号可以
  - 来自外部（规则通道EXTI11，注入通道EXTI15）
  - 来自内部定时器（TIM1～TIM4的有关事件）
  - 使用软件触发转换
- ADC3触发信号
  - 来自内部定时器（TIM1～TIM4、TIM5和TIM8的有关事件）
  - 也可以是软件触发

Table 5. ADC 固件库函数

函数名	描述
ADC_DeInit	将外设 ADCx 的全部寄存器重设为缺省值
ADC_Init	根据 ADC_InitStruct 中指定的参数初始化外设 ADCx 的寄存器
ADC_StructInit	把 ADC_InitStruct 中的每一个参数按缺省值填入
ADC_Cmd	使能或者失能指定的 ADC
ADC_DMACmd	使能或者失能指定的 ADC 的 DMA 请求
ADC_ITConfig	使能或者失能指定的 ADC 的中断
ADC_ResetCalibration	重置指定的 ADC 的校准寄存器
ADC_GetResetCalibrationStatus	获取 ADC 重置校准寄存器的状态
ADC_StartCalibration	开始指定 ADC 的校准程序
ADC_GetCalibrationStatus	获取指定 ADC 的校准状态
ADC_SoftwareStartConvCmd	使能或者失能指定的 ADC 的软件转换启动功能
ADC_GetSoftwareStartConvStatus	获取 ADC 软件转换启动状态
ADC_DiscModeChannelCountConfig	对 ADC 规则组通道配置间断模式
ADC_DiscModeCmd	使能或者失能指定的 ADC 规则组通道的间断模式
ADC_RegularChannelConfig	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
ADC_ExternalTrigConvConfig	使能或者失能 ADCx 的经外部触发启动转换功能
ADC_GetConversionValue	返回最近一次 ADCx 规则组的转换结果
ADC_GetDuelModeConversionValue	返回最近一次双 ADC 模式下的转换结果
ADC_AutoInjectedConvCmd	使能或者失能指定 ADC 在规则组转化后自动开始注入组转换
ADC_InjectedDiscModeCmd	使能或者失能指定 ADC 的注入组间断模式
ADC_ExternalTrigInjectedConvConfig	配置 ADCx 的外部触发启动注入组转换功能
ADC_ExternalTrigInjectedConvCmd	使能或者失能 ADCx 的经外部触发启动注入组转换功能
ADC_SoftwareStartinjectedConvCmd	使能或者失能 ADCx 软件启动注入组转换功能
ADC_GetsoftwareStartinjectedConvStatus	获取指定 ADC 的软件启动注入组转换状态
ADC_InjectedChannleConfig	设置指定 ADC 的注入组通道，设置它们的转化顺序和采样时间
ADC_InjectedSequencerLengthConfig	设置注入组通道的转换序列长度
ADC_SetinjectedOffset	设置注入组通道的转换偏移值
ADC_GetInjectedConversionValue	返回 ADC 指定注入通道的转换结果
ADC_AnalogWatchdogCmd	使能或者失能指定单个/全体，规则/注入组通道上的模拟看门狗
ADC_AnalogWatchdongThresholdsConfig	设置模拟看门狗的高/低阈值
ADC_AnalogWatchdongSingleChannelConfig	对单个 ADC 通道设置模拟看门狗
ADC_TampSensorVrefintCmd	使能或者失能温度传感器和内部参考电压通道
ADC_GetFlagStatus	检查制定 ADC 标志位置 1 与否
ADC_ClearFlag	清除 ADCx 的待处理标志位
ADC_GetITStatus	检查指定的 ADC 中断是否发生
ADC_ClearITPendingBit	清除 ADCx 的中断待处理位

ADC常用库函数

```
//ADC初始化
void ADC_Init  ( ADC_TypeDef *  ADCx, ADC_InitTypeDef *  ADC_InitStruct   ) ;
void ADC_DeInit  ( ADC_TypeDef *  ADCx );
//ADC使能
void ADC_Cmd  ( ADC_TypeDef *  ADCx,    FunctionalState  NewState   ) ;
void ADC_ITConfig  ( ADC_TypeDef *  ADCx,    uint16_t  ADC_IT,    FunctionalState
NewState   ) ;
//ADC使能软件转换
void  ADC_SoftwareStartConvCmd    (  ADC_TypeDef  *    ADCx,        FunctionalState
NewState   ) ;
//ADC规则通道配置
void ADC_RegularChannelConfig   ( ADC_TypeDef *  ADCx,     uint8_t  ADC_Channel,
uint8_t  Rank,
uint8_t  ADC_SampleTime  ) ;
//ADC获取结果
uint16_t ADC_GetConversionValue  ( ADC_TypeDef *  ADCx );
void ADC_ResetCalibration  ( ADC_TypeDef *  ADCx )  ;
FlagStatus ADC_GetResetCalibrationStatus  ( ADC_TypeDef *  ADCx ) ;
void ADC_StartCalibration  ( ADC_TypeDef *  ADCx ) ;
FlagStatus ADC_GetCalibrationStatus  ( ADC_TypeDef *  ADCx ) ;
```

- 用库函数来设定使用 ADC1 的通道 1 进行 AD 转换，详细步骤如下：

1）开启 PA 口时钟和 和 ADC1 时钟，设置 PA1 为模拟输入。

ADC 通道 1 为 PA1 的复用功能，故使用库函数：

RCC_APB2PeriphClockCmd ();

设置 PA1 的工作模式为模拟输入，使用库函数:

GPIO_Init();

2 ） 复位 ADC1 ，同时 设置 ADC1 分频因子。

通过复位 ADC1，将 ADC1 的全部寄存器重设为缺省值:

```
ADC_DeInit(ADC1);
```

通过 RCC_CFGR 寄存器设置 ADC1 的分频因子，分频因子要确保 ADC1 的时钟（ADCCLK）不要超过 14Mhz，若设置分频因子为6，则时钟为 72/6=12MHz

```
RCC_ADCCLKConfig(RCC_PCLK2_Div6);
```

3 ） 初始化 ADC1 参数， 设置 ADC1 的工作模式 以及规则序列的相关信息 。

void ADC_Init(ADC_TypeDef* ADCx, ADC_InitTypeDef* ADC_InitStruct);

设置单次转换模式、触发方式选择、数据对齐方式等

ADC_InitTypeDef类型

```
typedef struct
{
uint32_t ADC_Mode;
FunctionalState ADC_ScanConvMode;
FunctionalState ADC_ContinuousConvMode;
uint32_t ADC_ExternalTrigConv;
uint32_t ADC_DataAlign;
uint8_t ADC_NbrOfChannel;
}ADC_InitTypeDef;
```

实际设置

```
ADC_InitTypeDef ADC_InitStructure;
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;  //ADC 工作模式:独立模式
ADC_InitStructure.ADC_ScanConvMode = DISABLE;  //AD 单通道模式
ADC_InitStructure.ADC_ContinuousConvMode = DISABLE; //AD 单次转换模式
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;  //顺序进行规则转换的 ADC 通道的数目 1
ADC_Init(ADC1, &ADC_InitStructure);  //根据指定的参数初始化外设 ADCx
```

5 ） 使能 ADC 并校准（复位校准和ADC校准）。

注意：这两步是必须的！不校准将导致结果很不准确。

使能指定的 ADC ：

```
ADC_Cmd(ADC1, ENABLE);  //使能指定的 ADC1
```

执行复位校准并等待复位校准结束：

```
ADC_ResetCalibration(ADC1);
while(ADC_GetResetCalibrationStatus(ADC1)); //等待复位校准结束
```

执行 ADC 校准并等待ADC校准结束：

```
ADC_StartCalibration(ADC1);  //开始指定 ADC1 的校准状态
while(ADC_GetCalibrationStatus(ADC1)); //等待 AD 校准结束
```

记住：每次进行校准之后要等待校准结束。这里是通过获取校准状态来判断是否校准是否结束。

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，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，校准码储存在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。*

摘自《参考手册》

6 ） 设置通道、启动转换并读取 ADC结果。

- 设置规则序列通道以及采样周期的函数：

```
void ADC_RegularChannelConfig(ADC_TypeDef* ADCx, uint8_t ADC_Channel,
uint8_t Rank, uint8_t ADC_SampleTime);
```

```
例：
/* Configures ADC1 Channel12 as: first converted channel with an 7.5
cycles sample time */
ADC_RegularChannelConfig(ADC1, ADC_Channel_12, 1,
ADC_SampleTime_7Cycles5);
/* Configures ADC1 Channel18 as: second converted channel with an 1.5
cycles sample time */
ADC_RegularChannelConfig(ADC1, ADC_Channel_18, 2,
ADC_SampleTime_1Cycles5);
```

- 软件开启 ADC 转换：

```
ADC_SoftwareStartConvCmd(ADC1, ENABLE);//使能指定的 ADC1 的软件转换启动功能
```

- 开启转换之后，就可以获取转换 ADC 转换结果数据：

```
while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC ));//等待转换结束，标志置1表示EOC
ADC_GetConversionValue(ADC1);
```

注：在 AD 转换中，要获取 AD 转换的各个状态信息，可以使用库函数：

```
FlagStatus ADC_GetFlagStatus(ADC_TypeDef* ADCx, uint8_t ADC_FLAG);
```

[例]ADC用法举例

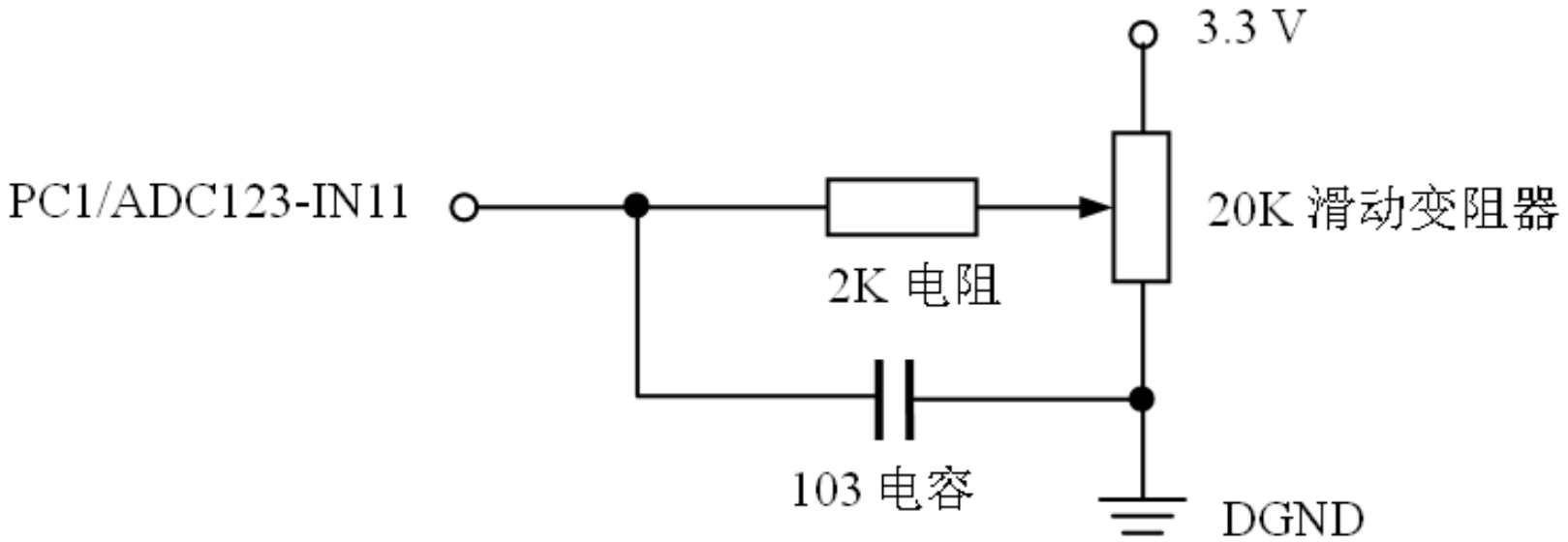
- 硬件连接：

Pc1—adc1连接外部电压

Usb转串口（ch340）连接至pc串口

- 程序功能：

通过usart以1s间隔向pc发送ADC1转换的结果



版本1：单次方式，软件触发转换，查询方式读取结果

```
int main(void)
{
  int ADC_ConvertedValue;
  float voltage;
  /* USART1 config */
  USART1_Config();
  /* enable adc1 and config adc1 to dma mode */
  ADC1_Init();
  printf("ADC1转换结果\r\n");
  while (1)
  {
    /* 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);
    /* 等待转换结束 */
    while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC));
    ADC_ConvertedValue=ADC_GetConversionValue(ADC1);
    voltage =(float) ADC_ConvertedValue /4096*3.3; // 读取转换的AD值
    printf("\r\n当前电压数字量: 0x%04X \r\n", ADC_ConvertedValue);
    printf("\r\n当前电压模拟值: %f V \r\n",voltage);
    //Delay(0xffffee);                                    // 延时
  }
}
```

```c
void USART1_Config(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;
  USART_InitTypeDef USART_InitStructure;
  /* config USART1 clock */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1 | RCC_APB2Periph_GPIOA, ENABLE);
  /* USART1 GPIO config */
  /* 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;
  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);
  /* USART1 mode config */
  USART_InitStructure.USART_BaudRate = 115200;
  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_HardwareFlowControl_None;
  USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
  USART_Init(USART1, &USART_InitStructure);
  USART_Cmd(USART1, ENABLE);
}
```

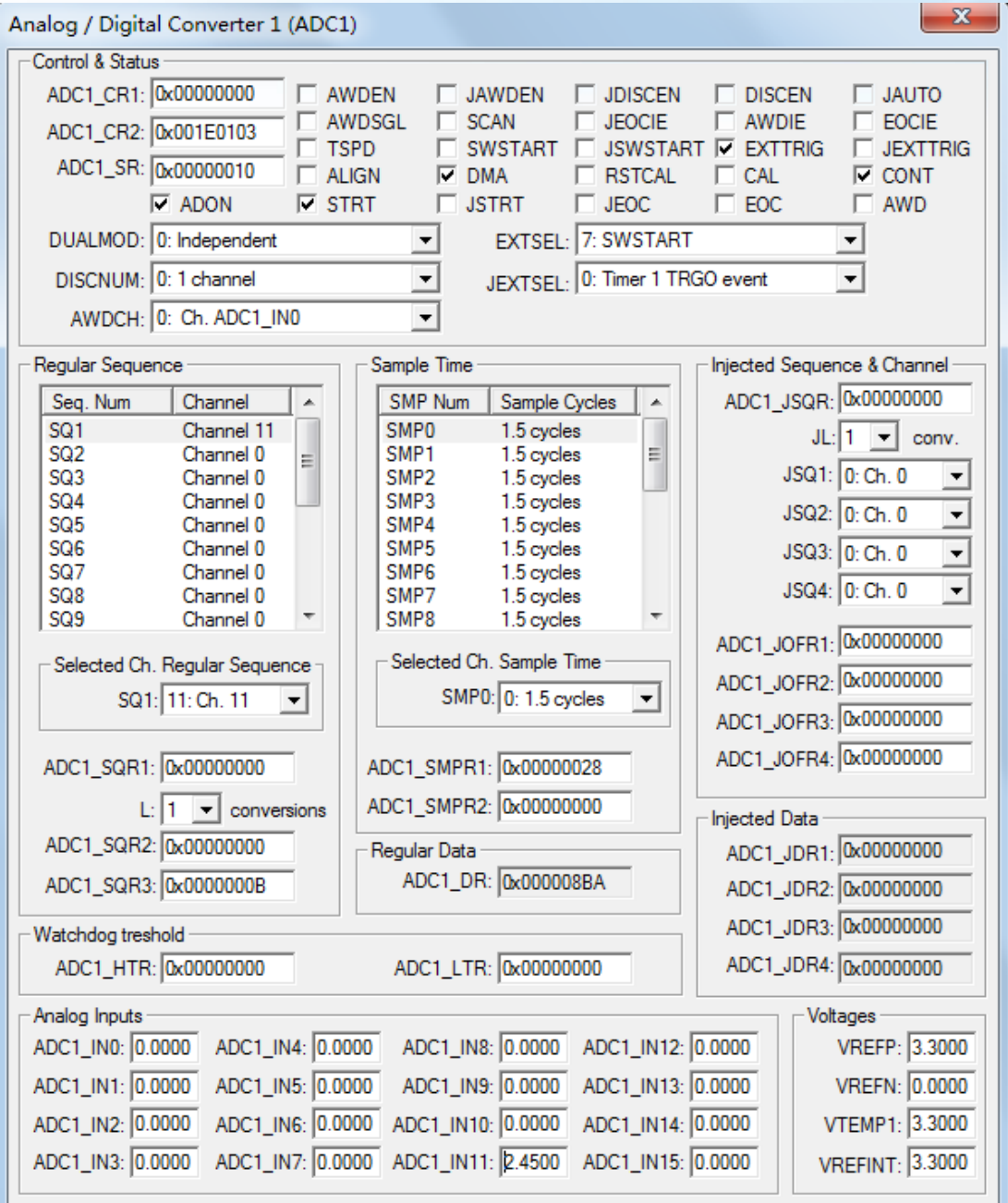
```
void ADC1_Init(void)
{
    ADC1_GPIO_Config();
    ADC1_Mode_Config();
}
```

ADC1_GPIO_Config()

```
static void ADC1_GPIO_Config(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;
  /* Enable ADC1 and GPIOC clock */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 | RCC_APB2Periph_GPIOC, ENABLE);
  /* Configure PC.01  as analog input */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
  GPIO_Init(GPIOC, &GPIO_InitStructure);          // PC1,输入时不用设置速率
}
```

```
static void ADC1_Mode_Config(void)
{
  ADC_InitTypeDef ADC_InitStructure;
  /* ADC1 configuration */
  ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;  //独立ADC模式
  ADC_InitStructure.ADC_ScanConvMode = DISABLE ;   //禁止扫描模式，扫描模式用于多通道采集
  ADC_InitStructure.ADC_ContinuousConvMode = DISABLE; //单次转换模式
  ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None; //不使用外部触发转换
  ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;  //采集数据右对齐
  ADC_InitStructure.ADC_NbrOfChannel = 1;   //要转换的通道数目1
  ADC_Init(ADC1, &ADC_InitStructure);
  /*配置ADC时钟，为PCLK2的8分频，即9Hz*/
  RCC_ADCCLKConfig(RCC_PCLK2_Div8);
  /*配置ADC1的通道11为55.5个采样周期，序列为1 */
  ADC_RegularChannelConfig(ADC1, ADC_Channel_11, 1, ADC_SampleTime_55Cycles5);
  /* 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));
}
```

```
while (1)
{
  /* 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);
  /* 等待转换结束 */
  while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC));
  ADC_ConvertedValue=ADC_GetConversionValue(ADC1);
  voltage =(float) ADC_ConvertedValue /4096*3.3; // 读取转换的AD值
  printf("\r\n当前电压数字量: 0x%04X \r\n", ADC_ConvertedValue);
  printf("\r\n当前电压模拟值: %f V \r\n",voltage);
  //Delay(0xffffee);                                    // 延时
}
```


Analog / Digital Converter 1 (ADC1)
Control & Status
ADC1_CR1: 0x00000000
ADC1_CR2: 0x001E0103
ADC1_SR: 0x00000010
AWDEN JAWDEN JDISCEN DISCEN JAUTO
AWDSGL SCAN JEOCIE AWDIE EOCIE
TSPD SWSTART JSWSTART EXTTRIG JEXTTRIG
ALIGN DMA RSTCAL CAL CONT
ADON STRT JSTRT JEOC EOC AWD
DUALMOD: 0: Independent
EXTSEL: 7: SWSTART
DISCNUM: 0: 1 channel
JEXTSEL: 0: Timer 1 TRGO event
AWDCH: 0: Ch. ADC1_IN0
Regular Sequence
Seq. Num Channel
SQ1 Channel 11
SQ2 Channel 0
SQ3 Channel 0
SQ4 Channel 0
SQ5 Channel 0
SQ6 Channel 0
SQ7 Channel 0
SQ8 Channel 0
SQ9 Channel 0
Selected Ch. Regular Sequence
SQ1: 11: Ch. 11
ADC1_SQR1: 0x00000000
L: 1 conversions
ADC1_SQR2: 0x00000000
ADC1_SQR3: 0x0000000B
Sample Time
SMP Num Sample Cycles
SMP0 1.5 cycles
SMP1 1.5 cycles
SMP2 1.5 cycles
SMP3 1.5 cycles
SMP4 1.5 cycles
SMP5 1.5 cycles
SMP6 1.5 cycles
SMP7 1.5 cycles
SMP8 1.5 cycles
Selected Ch. Sample Time
SMP0: 0: 1.5 cycles
ADC1_SMPR1: 0x00000028
ADC1_SMPR2: 0x00000000
Regular Data
ADC1_DR: 0x000008BA
Injected Sequence & Channel
ADC1_JSQR: 0x00000000
JL: 1 conv.
JSQ1: 0: Ch. 0
JSQ2: 0: Ch. 0
JSQ3: 0: Ch. 0
JSQ4: 0: Ch. 0
ADC1_JOFR1: 0x00000000
ADC1_JOFR2: 0x00000000
ADC1_JOFR3: 0x00000000
ADC1_JOFR4: 0x00000000
Injected Data
ADC1_JDR1: 0x00000000
ADC1_JDR2: 0x00000000
ADC1_JDR3: 0x00000000
ADC1_JDR4: 0x00000000
Watchdog treshold
ADC1_HTR: 0x00000000
ADC1_LTR: 0x00000000
Analog Inputs
ADC1_IN0: 0.0000 ADC1_IN4: 0.0000 ADC1_IN8: 0.0000 ADC1_IN12: 0.0000
ADC1_IN1: 0.0000 ADC1_IN5: 0.0000 ADC1_IN9: 0.0000 ADC1_IN13: 0.0000
ADC1_IN2: 0.0000 ADC1_IN6: 0.0000 ADC1_IN10: 0.0000 ADC1_IN14: 0.0000
ADC1_IN3: 0.0000 ADC1_IN7: 0.0000 ADC1_IN11: 2.4500 ADC1_IN15: 0.0000
Voltages
VREFP: 3.3000
VREFN: 0.0000
VTEMP1: 3.3000
VREFINT: 3.3000


UART #1
ADC1转换结果：
当前电压数字量：0x0000
当前电压模拟值：0.000000 V
当前电压数字量：0x0818
当前电压模拟值：1.669336 V
当前电压数字量：0x08BA
当前电压模拟值：1.799854 V
当前电压数字量：0x0BE0
当前电压模拟值：2.449219 V
当前电压数字量：0x0BE0
当前电压模拟值：2.449219 V

ADC1_Mode_Config()函数part1

```
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;  //ADC地址
  DMA_InitStructure.DMA_MemoryBaseAddr = (u32)&ADC_ConvertedValue;//内存地址
  DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
  DMA_InitStructure.DMA_BufferSize = 1;
  DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;//外设地址固定
  DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Disable;  //内存地址固定
  DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord; //半字
  DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
  DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;    //循环传输
  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);
```

- 使用dma1的通道1，数据从ADC外设的数据寄存器(ADC1_DR_Address)转移到内存（ADC_ConvertedValue变量），内存、外设地址都固定，每次传输的数据大小为半字（16位），使用DMA循环传输模式

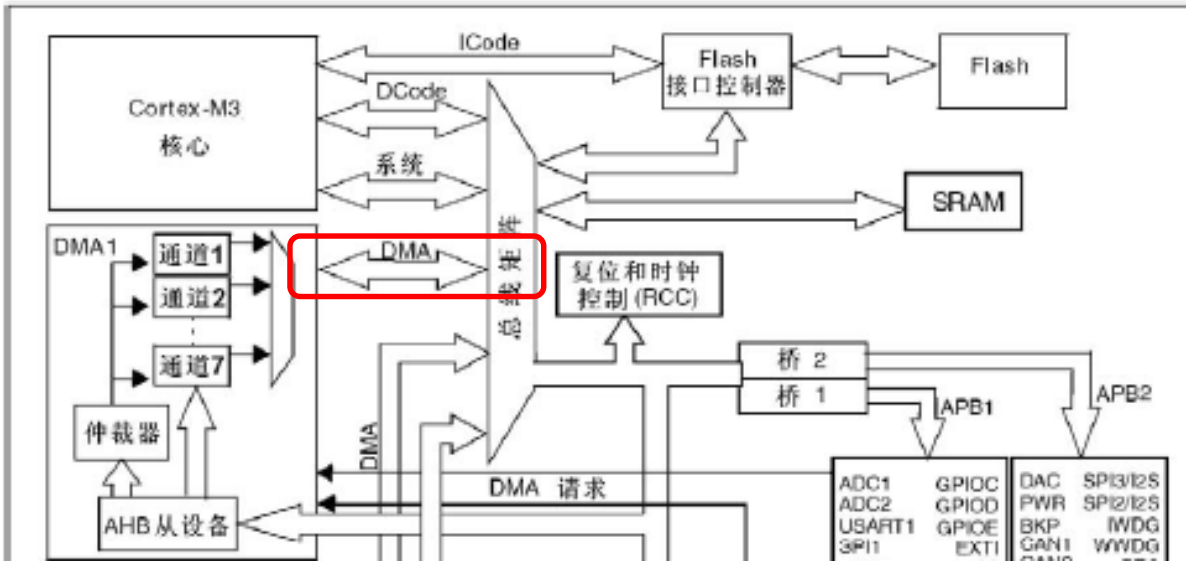
版本2：连续方式，软件触发，DMA传送结果

```
int main(void)
{
  /* USART1 config */
  USART1_Config();
  /* enable adc1 and config adc1 to dma mode */
  ADC1_Init();
  printf("\r\n -------这是一个ADC实验------\r\n");
  while (1)
  {
    ADC_ConvertedValueLocal =(float) ADC_ConvertedValue/4096*3.3; // 读取转换的AD值
    printf("\r\n The current AD value = 0x%04X \r\n", ADC_ConvertedValue);
    printf("\r\n The current AD value = %f V \r\n",ADC_ConvertedValueLocal);
    Delay(0xffffee);                                    // 延时
  }
}
```

ADC1_Mode_Config()函数part2

```
/* ADC1 configuration */
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;  //独立ADC模式
ADC_InitStructure.ADC_ScanConvMode = DISABLE ;   //禁止扫描模式，扫描模式用于多通道采集
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;  //开启连续转换模式，即不停地进行ADC转换
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None; //不使用外部触发转换
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;  //采集数据右对齐
ADC_InitStructure.ADC_NbrOfChannel = 1;   //要转换的通道数目1
ADC_Init(ADC1, &ADC_InitStructure);
/*配置ADC时钟，为PCLK2的8分频，即9Hz*/
RCC_ADCCLKConfig(RCC_PCLK2_Div8);
/*配置ADC1的通道11为55．5个采样周期，序列为1 */
ADC_RegularChannelConfig(ADC1, ADC_Channel_11, 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);
```

```
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);
  /* Configure PC.01  as analog input */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
  GPIO_Init(GPIOC, &GPIO_InitStructure);          // PC1,输入时不用设置速率
}
```



DMAC1挂在AHB总线上

ADC1_Mode_Config()part1:dma配置

```
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;  //ADC地址
  DMA_InitStructure.DMA_MemoryBaseAddr = (u32)&ADC_ConvertedValue;//内存地址
  DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
  DMA_InitStructure.DMA_BufferSize = 1;
  DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;//外设地址固定
  DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Disable;  //内存地址固定
  DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord; //半字
  DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
  DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;   //循环传输
  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);
```

外设	通道1	通道2	通道3	通道4	通道5	通道6	通道7
ADC1	ADC1						
SPI/I^2S		SPI1_RX	SPI1_TX	SPI/I2S2_RX	SPI/I2S2_TX		
USART		USART3_TX	USART3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
I^2C				I2C2_TX	I2C2_RX	I2C1_TX	I2C1_RX
TIM1		TIM1_CH1	TIM1_CH2	TIM1_TX4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
TIM2	TIM2_CH3	TIM2_UP			TIM2_CH1		TIM2_CH2 TIM2_CH4
TIM3		TIM3_CH3	TIM3_CH4 TIM3_UP			TIM3_CH1 TIM3_TRIG	
TIM4	TIM4_CH1			TIM4_CH2	TIM4_CH3		TIM4_UP

片上外设ADC1使用DMAC1的通道1

ADC1_Mode_Config()part2:ADC配置

```
/* ADC1 configuration */
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;  //独立ADC模式
ADC_InitStructure.ADC_ScanConvMode = DISABLE ;    //禁止扫描模式，扫描模式用于多通道采集
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;  //开启连续转换模式，即不停地进行ADC转换
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None; //不使用外部触发转换
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;  //采集数据右对齐
ADC_InitStructure.ADC_NbrOfChannel = 1;    //要转换的通道数目1
ADC_Init(ADC1, &ADC_InitStructure);
/*配置ADC时钟，为PCLK2的8分频，即9Hz*/
RCC_ADCCLKConfig(RCC_PCLK2_Div8);
/*配置ADC1的通道11为55.5个采样周期，序列为1 */
ADC_RegularChannelConfig(ADC1, ADC_Channel_11, 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);
}
```

与查询方式的两处变化：连续转换模式；软件触发只需在初始化部分写一次

```
#define ADC1_DR_Address    ((u32)0x40012400+0x4c)

__IO uint16_t ADC_ConvertedValue;
```

```
while (1)
{
  ADC_ConvertedValueLocal =(float) ADC_ConvertedValue/4096*3.3; // 读取转换的AD值
  printf("\r\n The current AD value = 0x%04X \r\n", ADC_ConvertedValue);
  printf("\r\n The current voltage value = %f V \r\n",ADC_ConvertedValueLocal);
  Delay(0xffffee);                                    // 延时
}
```

因为是连续转换方式，在初始化部分软件触发转换后，工作循环部分无需反复启动转换；转换结果采用DMA方式传输到内存全局变量