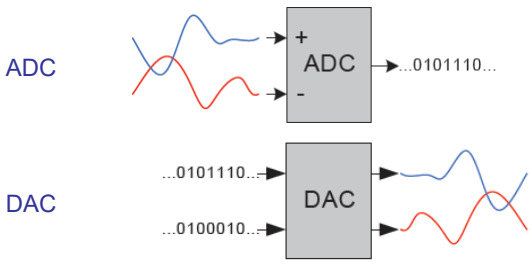
10. STM32的模拟数字转换器（ADC）

北京科技大学
计算机与通信工程学院

AD and DA



目录

CONTENTS.

STM32-10

3

电信号的形式与转换

从电信号的表现形式上，可以分为模拟信号和数字信号。

（1）模拟信号

模拟信号是指用连续变化的物理量所表达的信息，如温度、湿度、压力、长度、电流、电压等等，我们通常又把模拟信号称为连续信号，它在一定的时间范围内可以有无限多个不同的取值。

（2）数字信号

在数字电路中，由于数字信号只有0、1两个状态，它的值是通过中央值来判断的，在中央值以下规定为0，以上规定为1，所以即使混入了其他干扰信号，只要干扰信号的值不超过阀值范围，就可以再现出原来的信号。即使因干扰信号的值超过阀值范围而出现了误码，只要采用一定的编码技术，也很容易将出错的信号检测出来并加以纠正因此，与模拟信号相比，数字信号在传输过程中具有更高的抗干扰能力，更远的传输距离，且失真幅度小。

ADC的概念

模拟/数字转换（Analog to Digital Converter，简称ADC）是将输入的模拟信号转换为数字信号。

各种被测控的物理量（如：速度、压力、温度、光照强度、磁场等）是一些连续变化的物理量，传感器将这些物理量转换成与之相对应的电压和电流就是模拟信号。

单片机只能接收数字信号，要处理这些信号就必须转换成数字信号，模拟/数字转换是数字测控系统中必须的信号转换。

STM32的ADC简介

ADC：Analog to Digital，模拟数字转换器

- **1-三个独立的ADC 1 / 2 / 3**
- **2-分辨率为12位**
- **3-每个ADC具有18个通道，其中外部通道16个**

ADC功能框图讲解

1-电压输入范围

2-输入通道

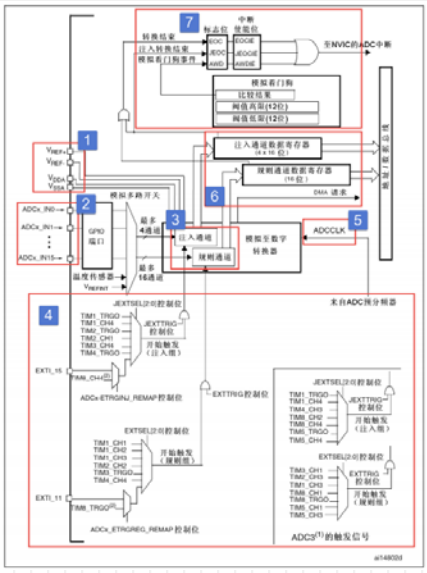
3-转换顺序

4-触发源

5-转换时间

6-数据寄存器

7-中断

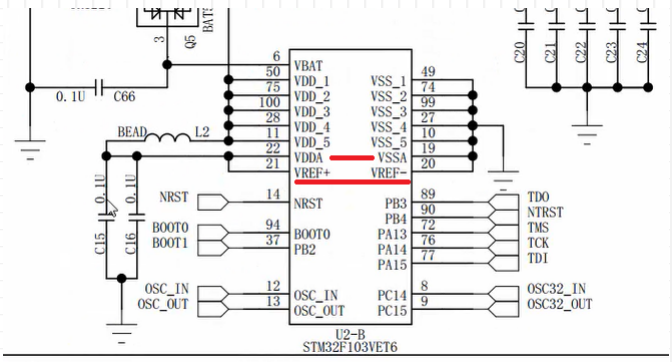


电压输入范围

输入电压：VREF- ≤ VIN ≤ VREF+

决定输入电压的引脚：VREF-、 VREF+ 、 VDDA 、 VSSA

VSSA 和 VREF-接地，把 VREF+和 VDDA 接 3V3，得到ADC 的输入电压范围为：0~3.3V。

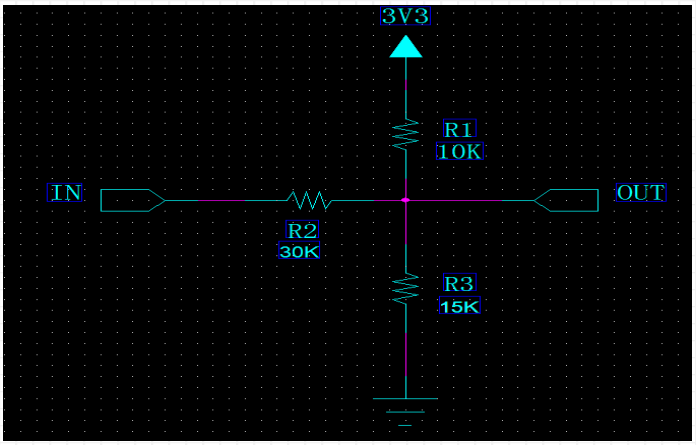


电压输入范围

超出0~3.3V的电压怎么测?

电压输入范围

ADC可以测量:-10V~10V



电压输入范围

根据基尔霍夫定律（KCL），节点流入的电流等于流出的电流

(Vint – Vout)/R2 + (3V3-Vout)/R1 = Vout / R3

Vout = (Vint + 10) /6

电压输入范围

R1 / R2 / R3 的值怎么确定?

输入通道

每个ADC具有18个通道，其中外部通道16个

STM32F103ZET6 ADC IO 分配					
ADC1	IO	ADC2	IO	ADC3	IO
通道0	PA0	通道0	PA0	通道0	PA0
通道1	PA1	通道1	PA1	通道1	PA1
通道2	PA2	通道2	PA2	通道2	PA2
通道3	PA3	通道3	PA3	通道3	PA3
通道4	PA4	通道4	PA4	通道4	PF6
通道5	PA5	通道5	PA5	通道5	PF7
通道6	PA6	通道6	PA6	通道6	PF8
通道7	PA7	通道7	PA7	通道7	PF9
通道8	PB0	通道8	PB0	通道8	PF10
通道9	PB1	通道9	PB1	通道9	连接内部VSS
通道10	PC0	通道10	PC0	通道10	PC0
通道11	PC1	通道11	PC1	通道11	PC1
通道12	PC2	通道12	PC2	通道12	PC2
通道13	PC3	通道13	PC3	通道13	PC3
通道14	PC4	通道14	PC4	通道14	连接内部VSS
通道15	PC5	通道15	PC5	通道15	连接内部VSS
通道16	连接内部温度传感器	通道16	连接内部VSS	通道16	连接内部VSS
通道17	连接内部Vrefint	通道17	连接内部VSS	通道17	连接内部VSS

A/D差分的优点

- **差分输入方式比单端输入来说，有更强的抗干扰能力:**
  - 单端输入信号时，如果一线上发生干扰变化，比如幅度增大 5mv，GND 不变，测到的数据会有偏差；
  - 而差分信号输入时，当外界存在干扰信号时，只要布线合理，大都同时被耦合到两条线上，幅度增大 5mv 会同时增大 5mv，而接收端关心的只是两个信号的差值，所以外界的这种共模噪声可以被完全抵消掉。由于两根信号的极性相反，它们对外辐射的电磁场可以相互抵消，有效的抑制释放到外界的电磁能量

输入通道分类

外部的 16 个通道在转换的时候又分为**规则通道**和**注入通道**，其中规则通道最多有 16路，注入通道最多有 4 路。那这两个通道有什么区别？在什么时候使用？

输入通道分类

规则通道：顾名思义，规则通道就是很规矩的意思，我们平时一般使用的就是这个通道。

注入通道：注入，可以理解为插入，插队的意思，是一种不安分的通道。它是一种在规则通道转换的时候强行插入要转换的一种。这点跟中断程序很像，都是不安分的主。所以，注入通道只有在规则通道存在时才会出现。

通道转换顺序

规则序列寄存器 SQRx, x（1，2，3）			
寄存器	寄存器位	功能	取值
SQR3	SQ1[4:0]	设置第1个转换的通道	通道1~16
	SQ2[4:0]	设置第2个转换的通道	通道1~16
	SQ3[4:0]	设置第3个转换的通道	通道1~16
	SQ4[4:0]	设置第4个转换的通道	通道1~16
	SQ5[4:0]	设置第5个转换的通道	通道1~16
	SQ6[4:0]	设置第6个转换的通道	通道1~16
SQR2	SQ7[4:0]	设置第7个转换的通道	通道1~16
	SQ8[4:0]	设置第8个转换的通道	通道1~16
	SQ9[4:0]	设置第9个转换的通道	通道1~16
	SQ10[4:0]	设置第10个转换的通道	通道1~16
	SQ11[4:0]	设置第11个转换的通道	通道1~16
	SQ12[4:0]	设置第12个转换的通道	通道1~16
SQR1	SQ13[4:0]	设置第13个转换的通道	通道1~16
	SQ14[4:0]	设置第14个转换的通道	通道1~16
	SQ15[4:0]	设置第15个转换的通道	通道1~16
	SQ16[4:0]	设置第16个转换的通道	通道1~16
	SQL[3:0]	需要转换多少个通道	1~16

图 30-3 规则序列寄存器

通道转换顺序

注入序列寄存器 JSQR 只有一个，最多支持 4 个通道，具体多少个由 JSQR 的 JL[2:0] 决定。如果 JL 的 值小于 4 的话，则 JSQR 跟 SQR 决定转换顺序的设置不一样，第一次转换的不是 JSQR1[4:0]，而是 JCQRx[4:0]，x = （4-JL），跟 SQR 刚好相反。如果 JL=00（1 个转换），那么转换的顺序是从 JSQR4[4:0]开始，而不是从 JSQR1[4:0]开始，这个要注意，编程的时候不要搞错。当 JL 等于 4 时，跟 SQR 一样。

注入序列寄存器 JSQR			
寄存器	寄存器位	功能	取值
JSQR	JSQ1[4:0]	设置第1个转换的通道	通道1~4
	JSQ2[4:0]	设置第2个转换的通道	通道1~4
	JSQ3[4:0]	设置第3个转换的通道	通道1~4
	JSQ4[4:0]	设置第4个转换的通道	通道1~4
	JL[1:0]	需要转换多少个通道	1~4

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

触发源

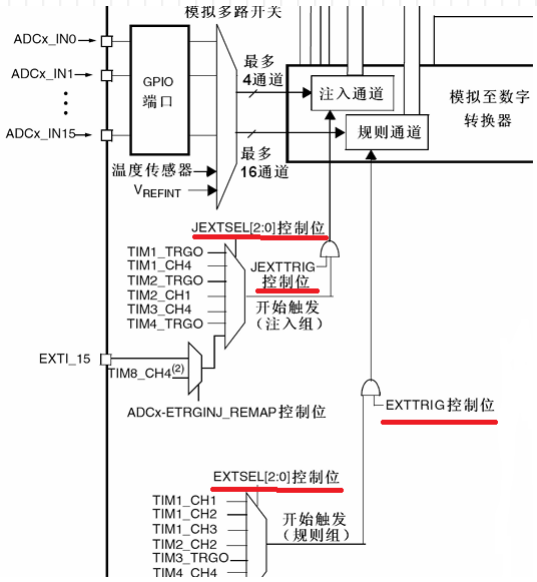
1、软件触发：

- ADC_CR2 :ADON：写1-启动转换，写0-结束转换
- ADC_CR2 : SWSTART：开始转换规则通道
- ADC_CR2 : JSWSTART：开始转换注入通道

2、外部事件触发：内部定时器/外部IO

选择：ADC_CR2 :EXTSEL[2:0]和 JEXTSEL[2:0]

激活：ADC_CR2 :EXTEN 和 JEXTEN

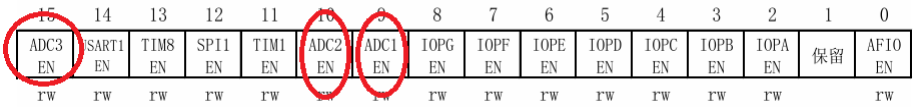


可编程的转换时间

转换时间：Tconv = 采样时间 + 12.5 个周期

ADC_CLK：ADC模拟电路时钟，最大值为14M，由PCLK2提供，还可分频，2/4/6/8，RCC_CFGR 的ADCPRE[1:0]设置。PCLK2=72M。

时钟使能：寄存器RCC_APB2ENR使能ADCx的外设时钟



转换时间

采样时间：ADC 需要若干个 ADC_CLK 周期完成对输入的模拟量进行采样，采样的周期数可通过ADC 采样时间寄存器 ADC_SMPR1 和 ADC_SMPR2 中的 SMPx[2:0]位设置，ADC_SMPR2控制的是通道 0~9， ADC_SMPR1 控制的是通道 10~17。每个通道可以分别用不同的时间采样。其中采样周期最小是 1.5 个，即如果我们要达到最快的采样，那么应该设置采样周期为 1.5个周期，这里说的周期就是 1/ADC_CLK。

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

转换时间

最短的转换时间：Tconv = 采样时间 + 12.5 个周期

PCLK2 = 72M，ADC_CLK = 72/6 = 12M

Tconv = 1.5+12.5 = 14周期 = 14/12us=1.17us

数据寄存器

一切准备就绪后， ADC 转换后的数据根据转换组的不同，规则组的数据放在**ADC_DR** 寄存器，注入组的数据放在 **JDRx**。

数据寄存器

11.12.14 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。

1-16位有效，用于存放独立模式转换完成数据

2- ADC_CR2 ： ALIGN

3-只有一个，多通道采集的是最好使用DMA

数据寄存器

11.12.13 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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JDATA[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

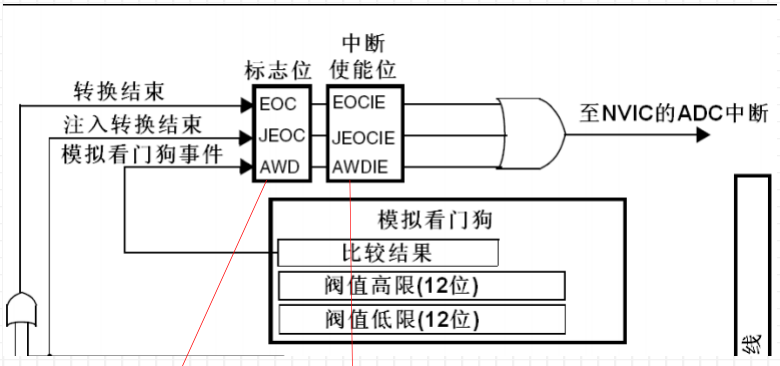
位31:16	保留。必须保持为0。
位21:20	**JDATA[15:0]**：注入转换的数据 (Injected data) 这些位为只读，包含了注入通道的转换结果。数据是左对齐或右对齐，如图29和图30所示。

1-16位有效，用于存放注入通道转换完成数据

2- ADC_CR2 ： ALIGN

3-有4个这样的寄存器

中断



1-ADC_SR，ADC_CR1

2- ADC看门狗高阈值寄存器ADC_HTR，

ADC看门狗低阈值寄存器ADC_LTR

电压转换

怎么根据数据量算出模拟量

1-电压输入范围为：0~3.3V

2-分辨率为12位

3-最小精度为：3.3/2^12

4-设数字量为X，则有模拟量 Y = (3.3 / 2^12)*X

目录

CONTENTS.

STM32-10

28

ADC外设ADC1, ADC2,ADC3

```
typedef struct
{
  __IO uint32_t SR;
  __IO uint32_t CR1;
  __IO uint32_t CR2;
  __IO uint32_t SMPR1;
  __IO uint32_t SMPR2;
  __IO uint32_t JOFR1;
  __IO uint32_t JOFR2;
  __IO uint32_t JOFR3;
  __IO uint32_t JOFR4;
  __IO uint32_t HTR;
  __IO uint32_t LTR;
  __IO uint32_t SQR1;
  __IO uint32_t SQR2;
  __IO uint32_t SQR3;
  __IO uint32_t JSQR;
  __IO uint32_t JDR1;
  __IO uint32_t JDR2;
  __IO uint32_t JDR3;
  __IO uint32_t JDR4;
  __IO uint32_t DR;
} ADC_TypeDef;
```

```
#define ADC1        ((ADC_TypeDef *) ADC1_BASE)
#define ADC2        ((ADC_TypeDef *) ADC2_BASE)
#define ADC3        ((ADC_TypeDef *) ADC3_BASE)
```

29

ADC初始化结构体

ADC_InitTypeDef

ADC_InitTypeDef 结构体定义在 stm32f10x_adc.h 文件内，具体定义如下：

```
typedef struct
{
    uint32_t ADC_Mode;                      // ADC 工作模式选择
    FunctionalState ADC_ScanConvMode;       /* ADC 扫描（多通道）
                                               或者单次（单通道）模式选择 */
    FunctionalState ADC_ContinuousConvMode; // ADC 单次转换或者连续转换选择
    uint32_t ADC_ExternalTrigConv;          // ADC 转换触发信号选择
    uint32_t ADC_DataAlign;                 // ADC 数据寄存器对齐格式
    uint8_t ADC_NbrOfChannel;               // ADC 采集通道数
} ADC_InitTypeDef;
```

ADC初始化结构体

- ADC_MODE：模式选择，ADC_CR1:DUALMOD

```
/** @defgroup ADC_mode
  * @{
  */

#define ADC_Mode_Independent                     ((uint32_t)0x00000000)
#define ADC_Mode_RegInjecSimult                  ((uint32_t)0x00010000)
#define ADC_Mode_RegSimult_AlterTrig             ((uint32_t)0x00020000)
#define ADC_Mode_InjecSimult_FastInterl          ((uint32_t)0x00030000)
#define ADC_Mode_InjecSimult_SlowInterl          ((uint32_t)0x00040000)
#define ADC_Mode_InjecSimult                     ((uint32_t)0x00050000)
#define ADC_Mode_RegSimult                       ((uint32_t)0x00060000)
#define ADC_Mode_FastInterl                      ((uint32_t)0x00070000)
#define ADC_Mode_SlowInterl                      ((uint32_t)0x00080000)
#define ADC_Mode_AlterTrig                       ((uint32_t)0x00090000)
```

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

ADC初始化结构体

- ADC_ScanConvMode：扫描模式
  - 寄存器ADC_CR1:SCAN
  - 只有两种选择：ENABLE or DISABLE
    - 扫描模式：即多通道模式
    - 非扫描模式:即单通道模式
- ADC_ContinuousConvMode：连续模式
  - 寄存器ADC_CR2:CONT
  - 只有两种选择：ENABLE or DISABLE
    - Enable：连续转换
    - Disable：单次转换

ADC初始化结构体

- ADC_ExternalTrigConv：转换触发信号配置

```
/** @defgroup ADC_external_trigger_sources_for_regular_channels_conversion
  * @{
  */

#define ADC_ExternalTrigConv_T1_CC1             ((uint32_t)0x00000000) /*!< For ADC1 and ADC2 */
#define ADC_ExternalTrigConv_T1_CC2             ((uint32_t)0x00020000) /*!< For ADC1 and ADC2 */
#define ADC_ExternalTrigConv_T2_CC2             ((uint32_t)0x00060000) /*!< For ADC1 and ADC2 */
#define ADC_ExternalTrigConv_T3_TRGO            ((uint32_t)0x00080000) /*!< For ADC1 and ADC2 */
#define ADC_ExternalTrigConv_T4_CC4             ((uint32_t)0x000A0000) /*!< For ADC1 and ADC2 */
#define ADC_ExternalTrigConv_Ext_IT11_TIM8_TRGO ((uint32_t)0x000C0000) /*!< For ADC1 and ADC2 */

#define ADC_ExternalTrigConv_T1_CC3             ((uint32_t)0x00040000) /*!< For ADC1, ADC2 and ADC3 */
#define ADC_ExternalTrigConv_None               ((uint32_t)0x000E0000) /*!< For ADC1, ADC2 and ADC3 */

#define ADC_ExternalTrigConv_T3_CC1             ((uint32_t)0x00000000) /*!< For ADC3 only */
#define ADC_ExternalTrigConv_T2_CC3             ((uint32_t)0x00020000) /*!< For ADC3 only */
#define ADC_ExternalTrigConv_T8_CC1             ((uint32_t)0x00060000) /*!< For ADC3 only */
#define ADC_ExternalTrigConv_T8_TRGO            ((uint32_t)0x00080000) /*!< For ADC3 only */
#define ADC_ExternalTrigConv_T5_CC1             ((uint32_t)0x000A0000) /*!< For ADC3 only */
#define ADC_ExternalTrigConv_T5_CC3             ((uint32_t)0x000C0000) /*!< For ADC3 only */
```

ADC初始化结构体

- ADC_DataAlign：数据对齐格式，ADC_CR2:ALIGN

```
/** @defgroup ADC_data_align
  * @{
  */

#define ADC_DataAlign_Right                     ((uint32_t)0x00000000)
#define ADC_DataAlign_Left                      ((uint32_t)0x00000800)
```

- ADC_NbrOfChannel:转换的通道数，配置规则序列寄存器和注入序列寄存器
  - 1 to 16

几个常用的固件库函数讲解

- ADC_Init(); 218
- RCC_ADCCLKConfig(); 766
- ADC_RegularChannelConfig(); 590
- ADC_GetConversionValue(); 708
- ADC_Cmd(); 299
- ADC_SoftwareStartConvCmd(); 457
- ADC_ExternalTrigConvCmd(); 686
- ADC_DMACmd(); 324

目录

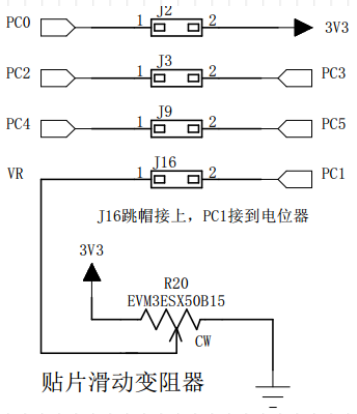
CONTENTS.

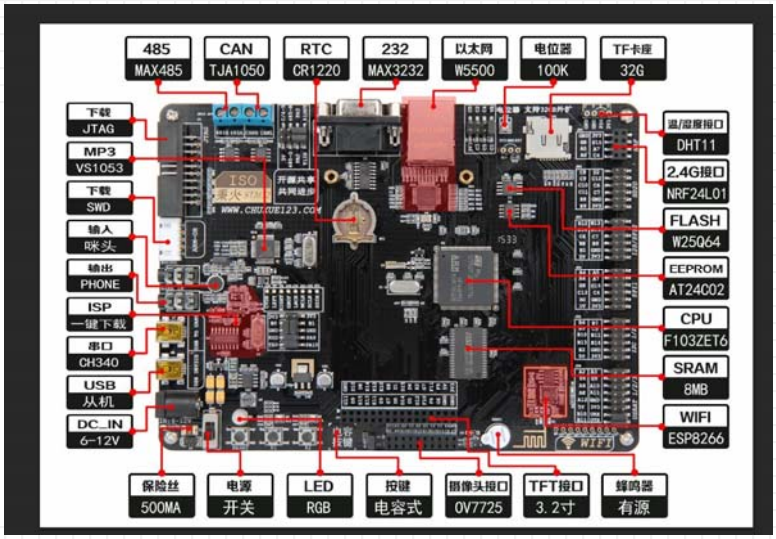
STM32-10

36

ADC硬件设计

ADC接口





实验设计

- Ex1-独立模式-单通道-中断读取
- Ex2-独立模式-多通道-DMA读取

编程要点

1-独立模式-单通道-中断读取

1-初始化ADC用到的GPIO
2-初始化ADC初始化结构体
3-配置ADC时钟，配置通道的转换顺序和采样时间
4-使能ADC转换完成中断，配置中断优先级
5-使能ADC，准备开始转换
6-校准ADC
7-软件触发ADC，真正开始转换
8-编写中断服务函数，读取ADC转换数据
9-编写main函数，把转换的数据打印出来

EX1-独立模式-单通道-中断读取

代码清单 31-1 ADC 宏定义

```
1  // ADC 编号选择
2  // 可以是 ADC1/2，如果使用 ADC3，中断相关的宏要改成 ADC3 的
3  #define     ADC_APBxClock_FUN              RCC_APB2PeriphClockCmd
4  #define     ADCx                           ADC2
5  #define     ADC_CLK                        RCC_APB2Periph_ADC2
6
7  // ADC GPIO 宏定义
8  // 注意：用作 ADC 采集的 IO 必须没有复用，否则采集电压会有影响
9  #define     ADC_GPIO_APBxClock_FUN         RCC_APB2PeriphClockCmd
10 #define     ADC_GPIO_CLK                   RCC_APB2Periph_GPIOC
11 #define     ADC_PORT                       GPIOC
12 #define     ADC_PIN                        GPIO_Pin_1
13 // ADC 通道宏定义
14 #define     ADC_CHANNEL                    ADC_Channel_11
15
16 // ADC 中断相关宏定义
17 #define     ADC_IRQ                        ADC1_2_IRQn
18 #define     ADC_IRQHandler                 ADC1_2_IRQHandler
```

EX1-独立模式-单通道-中断读取

代码清单 31-2 ADC GPIO 初始化

```c
static void ADCx_GPIO_Config(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    // 打开 ADC IO 端口时钟
    ADC_GPIO_APBxClock_FUN ( ADC_GPIO_CLK, ENABLE );

    // 配置 ADC IO 引脚模式
    // 必须为模拟输入
    GPIO_InitStructure.GPIO_Pin = ADC_PIN;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;

    // 初始化 ADC IO
    GPIO_Init(ADC_PORT, &GPIO_InitStructure);
}
```

代码清单 31-3 ADC 工作模式配置

```c
static void ADCx_Mode_Config(void)
{
    ADC_InitTypeDef ADC_InitStructure;

    // 打开 ADC 时钟
    ADC_APBxClock_FUN ( ADC_CLK, ENABLE );

    // ADC 模式配置
    // 只使用一个 ADC，属于独立模式
    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;

    // 禁止扫描模式，多通道才要，单通道不需要
    ADC_InitStructure.ADC_ScanConvMode = DISABLE ;

    // 连续转换模式
    ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;

    // 不用外部触发转换，软件开启即可
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;

    // 转换结果右对齐
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;

    // 转换通道 1 个
    ADC_InitStructure.ADC_NbrOfChannel = 1;

    // 初始化 ADC
    ADC_Init(ADCx, &ADC_InitStructure);

    // 配置 ADC 时钟为 PCLK2 的 8 分频，即 9MHz
    RCC_ADCCLKConfig(RCC_PCLK2_Div8);

    // 配置 ADC 通道转换顺序和采样时间
    ADC_RegularChannelConfig(ADCx, ADC_CHANNEL, 1,
                             ADC_SampleTime_55Cycles5);

    // ADC 转换结束产生中断，在中断服务程序中读取转换值
    ADC_ITConfig(ADCx, ADC_IT_EOC, ENABLE);

    // 开启 ADC ，并开始转换
    ADC_Cmd(ADCx, ENABLE);

    // 初始化 ADC 校准寄存器
    ADC_ResetCalibration(ADCx);
    // 等待校准寄存器初始化完成
    while (ADC_GetResetCalibrationStatus(ADCx));

    // ADC 开始校准
    ADC_StartCalibration(ADCx);
    // 等待校准完成
    while (ADC_GetCalibrationStatus(ADCx));

    // 由于没有采用外部触发，所以使用软件触发 ADC 转换
    ADC_SoftwareStartConvCmd(ADCx, ENABLE);
}
```

EX1-独立模式-单通道-中断读取

EX1-独立模式-单通道-中断读取

代码清单 31-4 ADC 中断配置

```
static void ADC_NVIC_Config(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;

    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);

    NVIC_InitStructure.NVIC_IRQChannel = ADC_IRQ;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;

    NVIC_Init(&NVIC_InitStructure);
}
```

__IO uint16_t ADC_ConvertedValue; //定义在stm32f10x_it.c中的全局变量

代码清单 31-5 ADC 中断服务函数

```
void ADC_IRQHandler(void)
{
    if (ADC_GetITStatus(ADCx,ADC_IT_EOC)==SET) {
        // 读取 ADC 的转换值
        ADC_ConvertedValue = ADC_GetConversionValue(ADCx);

    }
    ADC_ClearITPendingBit(ADCx,ADC_IT_EOC);

}
```

EX1-独立模式-单通道-中断读取

extern __IO uint16_t ADC_ConvertedValue;

代码清单 31-6 主函数

```
int main(void)
{
    // 配置串口
    USART_Config();

    // ADC 初始化
    ADCx_Init();

    printf("\r\n ----这是一个 ADC 单通道中断读取实验----\r\n");

    while (1)
    {
        ADC_ConvertedValueLocal =(float) ADC_ConvertedValue/4096*3.3;

        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);
        printf("\r\n\r\n");

        Delay(0xffffee);
    }
}
```

编程要点

2-独立模式-多通道-DMA读取

1) 初始化ADC GPIO;
2) 初始化ADC 工作参数;
3) 配置DMA 工作参数;
4) 读取ADC 采集的数据;

EX2-2-独立模式-多通道-DMA读取

代码清单 31-7 多通道 ADC 相关宏定义

```
1  // ADC 宏定义
2  #define    ADCx                        ADC1
3  #define    ADC_APBxClock_FUN           RCC_APB2PeriphClockCmd
4  #define    ADC_CLK                     RCC_APB2Periph_ADC1
5
6  #define    ADC_GPIO_APBxClock_FUN      RCC_APB2PeriphClockCmd
7  #define    ADC_GPIO_CLK                RCC_APB2Periph_GPIOC
8  #define    ADC_PORT                    GPIOC
9
10 // 转换通道个数
11 #define    NOFCHANEL                   5
12
13 #define    ADC_PIN1                    GPIO_Pin_0
14 #define    ADC_CHANNEL1                ADC_Channel_10
15
16 #define    ADC_PIN2                    GPIO_Pin_1
17 #define    ADC_CHANNEL2                ADC_Channel_11
18
19 #define    ADC_PIN3                    GPIO_Pin_3
20 #define    ADC_CHANNEL3                ADC_Channel_13
21
22 #define    ADC_PIN4                    GPIO_Pin_4
23 #define    ADC_CHANNEL4                ADC_Channel_14
24
25 #define    ADC_PIN5                    GPIO_Pin_5
26 #define    ADC_CHANNEL5                ADC_Channel_15
27
28 // ADC1 对应 DMA1 通道 1，ADC3 对应 DMA2 通道 5，ADC2 没有 DMA 功能
29 #define    ADC_DMA_CHANNEL             DMA1_Channel1
```

EX2-2-独立模式-多通道-DMA读取

__IO uint16_t ADC_ConvertedValue[NOFCHANEL]={0,0,0,0,0,0};

代码清单 31-8 ADC GPIO 初始化

```c
static void ADCx_GPIO_Config(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    // 打开 ADC IO 端口时钟
    ADC_GPIO_APBxClock_FUN ( ADC_GPIO_CLK, ENABLE );

    // 配置 ADC IO 引脚模式
    GPIO_InitStructure.GPIO_Pin = ADC_PIN1
                                |ADC_PIN2
                                |ADC_PIN3
                                |ADC_PIN4
                                |ADC_PIN5;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    // 初始化 ADC IO
    GPIO_Init(ADC_PORT, &GPIO_InitStructure);
}
```

EX2-2-独立模式-多通道-DMA读取

代码清单 31-9 ADC 工作模式配置

```c
static void ADCx_Mode_Config(void)
{
    DMA_InitTypeDef DMA_InitStructure;
    ADC_InitTypeDef ADC_InitStructure;

    // 打开 DMA 时钟
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
    // 打开 ADC 时钟
    ADC_APBxClock_FUN ( ADC_CLK, ENABLE );

    /* -------------------DMA 模式配置---------------- */
    // 复位 DMA 控制器
    DMA_DeInit(ADC_DMA_CHANNEL);
    // 配置 DMA 初始化结构体
    // 外设基址为：ADC 数据寄存器地址
    DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)(&( ADCx->DR ));
    // 存储器地址
    DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)ADC_ConvertedValue;
    // 数据源来自外设
    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
    // 缓冲区大小，应该等于数据目的地的大小
    DMA_InitStructure.DMA_BufferSize = NOFCHANEL;
    // 外设寄存器只有一个，地址不用递增
    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
    // 存储器地址递增
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
    // 外设数据大小为半字，即两个字节
    DMA_InitStructure.DMA_PeripheralDataSize =
        DMA_PeripheralDataSize_HalfWord;
    // 内存数据大小也为半字，跟外设数据大小相同
    DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
    // 循环传输模式
    DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
    // DMA 传输通道优先级为高，当使用一个 DMA 通道时，优先级设置不影响
    DMA_InitStructure.DMA_Priority = DMA_Priority_High;
    // 禁止存储器到存储器模式，因为是从外设到存储器
    DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
    // 初始化 DMA
    DMA_Init(ADC_DMA_CHANNEL, &DMA_InitStructure);
    // 使能 DMA 通道
    DMA_Cmd(ADC_DMA_CHANNEL , ENABLE);

}
```

EX2-2-独立模式-多通道-DMA读取

```
/* ----------------ADC 模式配置---------------------- */
// 只使用一个 ADC，属于单模式
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
// 扫描模式
ADC_InitStructure.ADC_ScanConvMode = ENABLE ;
// 连续转换模式
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
// 不用外部触发转换，软件开启即可
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
// 转换结果右对齐
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
// 转换通道个数
ADC_InitStructure.ADC_NbrOfChannel = NOFCHANEL;
// 初始化 ADC
ADC_Init(ADCx, &ADC_InitStructure);
// 配置 ADC 时钟为 PCLK2 的 8 分频，即 9MHz
RCC_ADCCLKConfig(RCC_PCLK2_Div8);
// 配置 ADC 通道的转换顺序和采样时间
ADC_RegularChannelConfig(ADCx, ADC_CHANNEL1, 1,
                         ADC_SampleTime_55Cycles5);
ADC_RegularChannelConfig(ADCx, ADC_CHANNEL2, 2,
                         ADC_SampleTime_55Cycles5);
ADC_RegularChannelConfig(ADCx, ADC_CHANNEL3, 3,
                         ADC_SampleTime_55Cycles5);
ADC_RegularChannelConfig(ADCx, ADC_CHANNEL4, 4,
                         ADC_SampleTime_55Cycles5);
ADC_RegularChannelConfig(ADCx, ADC_CHANNEL5, 5,
                         ADC_SampleTime_55Cycles5);
```

```
    // 使能 ADC DMA 请求
    ADC_DMACmd(ADCx, ENABLE);
    // 开启 ADC ，并开始转换
    ADC_Cmd(ADCx, ENABLE);
    // 初始化 ADC 校准寄存器
    ADC_ResetCalibration(ADCx);
    // 等待校准寄存器初始化完成
    while (ADC_GetResetCalibrationStatus(ADCx));
    // ADC 开始校准
    ADC_StartCalibration(ADCx);
    // 等待校准完成
    while (ADC_GetCalibrationStatus(ADCx));
    // 由于没有采用外部触发，所以使用软件触发 ADC 转换
    ADC_SoftwareStartConvCmd(ADCx, ENABLE);
}
```

EX2-2-独立模式-多通道-DMA读取

代码清单 31-10 主函数

```
int main(void)
{
    // 配置串口
    USART_Config();

    // ADC 初始化
    ADCx_Init();

    printf("\r\n ----这是一个 ADC 多通道采集 DMA 读取实验----\r\n");

    while (1)
    {

        ADC_ConvertedValueLocal[0] =(float)
                                    ADC_ConvertedValue[0]/4096*3.3;
        ADC_ConvertedValueLocal[1] =(float)
                                    ADC_ConvertedValue[1]/4096*3.3;
        ADC_ConvertedValueLocal[2] =(float)
                                    ADC_ConvertedValue[2]/4096*3.3;
        ADC_ConvertedValueLocal[3] =(float)
                                    ADC_ConvertedValue[3]/4096*3.3;
        ADC_ConvertedValueLocal[4] =(float)
                                    ADC_ConvertedValue[4]/4096*3.3;

        printf("\r\n CH1 value = %f V \r\n",ADC_ConvertedValueLocal[0]);
        printf("\r\n CH2 value = %f V \r\n",ADC_ConvertedValueLocal[1]);
        printf("\r\n CH3 value = %f V \r\n",ADC_ConvertedValueLocal[2]);
        printf("\r\n CH2 value = %f V \r\n",ADC_ConvertedValueLocal[3]);
        printf("\r\n CH3 value = %f V \r\n",ADC_ConvertedValueLocal[4]);

        printf("\r\n\r\n");
        Delay(0xffffee);

    }
}
```