1. STM32F4 ADC 介绍

1.1. 简介

模数转换器，将模拟信号转换为数字信号。转换原理主要为逐次逼近型、双积分型、电压频率转换型三种。而本 ADC 呢是逐次逼近型的模拟数字转换器。

STM32F4 系列一般都有 **3 个 ADC**，这些 ADC 可以独立使用，也可以使用**双重/三重模式**（提高采样率）。STM32F4 的 ADC 是 **12 位逐次逼近型**的模拟数字转换器。

1. 多达 19 个复用通道，可以测量来自 16 个外部源、2 个内部源和 Vbat 通道的信号。

2. 这些通道的 A/D 转换可以**单次、连续、扫描或间断模式**执行。

3. 结果可以左对齐（4-15 位）或右对齐（0-11 位）的方式存储在 16 位数据寄存器中。

4. ADC 具有**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阀值上限或下限。

1.2. ADC 框图

按照顺序逐步分析 ADC 的框图，以便编程。

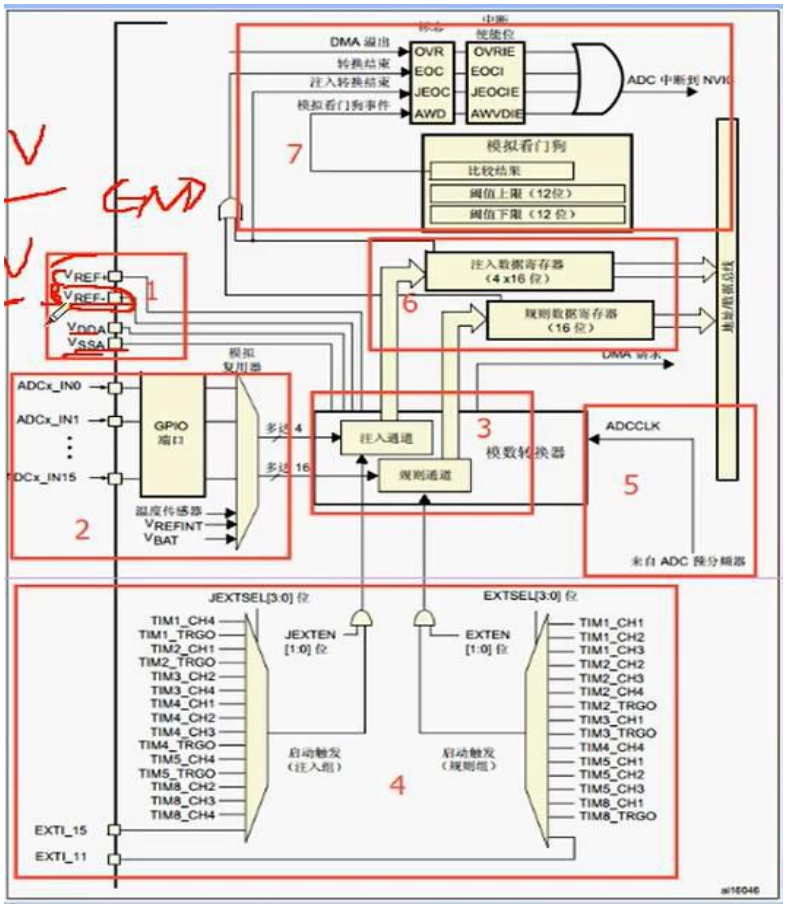

图 1 STM32F4 ADC 内部框图

1.2.1. 标号 1：参考电压

VREF+和 VDDA 连接 0~3.3V 的参考电压，是 ADC 转换的参考电压。
VREF-和 VSSA 接 GND。

1.2.2. 标号 2：输入通道

总共有 3 个 ADC 转换器以及 16 个外部输入通道，连接需要进行模/数转换的电压。另外还有 3 个内部通道，只有 ADC1。

	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	PF3
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	PF4
通道15	PC5	PC5	PF5
通道16	温度传感器		
通道17	内部参考电压VREF		
通道18	VBAT		

图 2 ADC 通道对应管脚标号

1.2.3. 标号 3：通道转换顺序

外部的 16 个通道再转换的时候可分为 2 组通道：**规则通道组**和**注入通道组**，其中规则通道组最多有 16 路，注入通道组最多有 4 路。

规则通道组：从名字来理解，规则通道就是一种规矩的通道，类似于正常执行的程序，通常我们使用的都是这个通道。

注入通道组：从名字来理解，注入即为插入，是一种不安分的通道，类似于中断。当程序正常往下执行时，中断可以打断程序的执行。同样如果再规则通道的转换过程中，有注入通道插入，那么就要先转换完注入通道，等注入通道转换完成后再回到规则通道的转换流程。

每个组包含一个转换序列，该序列可按任意顺序在任意通道上完成。

1.2.4. 标号 4：触发

分为外部触发和内部触发。

1.2.5. 标号 5：分频器

对 ADC 内部 84M 的时钟进行分频，分频有 2、4、6、8 四种分频参数。当选择 4 时，则输入 ADC 的时钟为 84/4=21M 的时钟。

1.2.6. 标号 6：数据寄存器

ADC 转换后的数据根据转换组的不同，规则组的数据放在 ACD_DR 寄存器内，注入组的数据放在 JDR×内。如果时使用双重或者三重模式，那规矩组的数据是存放在通用规矩寄存器 ADC_CDR 内的。

因为 STM32F4 的 ADC 是 12 位转换精度，而数据寄存器是 16 位，所以 ADC 在存放数据的时候就有左对齐和右对齐区分。如果是左对齐，AD 转换完成数据存放在 ADC_DR 寄存器的[4:15]位内；如果是右对齐，则存放在 ADC_DR 寄存器的[0:11]位内。具体选择何种存放方式，需通过 ADC_CR2 的 11 位 ALIGN 设置。

编程要注意这些寄存器的设置，如果不是特别清楚最好参考一下手册上的资料。

2. STM32F4 ADC 配置步骤

2.1.步骤

2.1.1. 使能端口时钟和 ADC 时钟，设置引脚模式为模拟输入

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
GPIO_InitStructure, GPIO_Mode = GPIO_Mode_AN;//模拟输入模式
```

2.1.2. 设置 ADC 通用控制寄存器 CCR，包括 ADC 模式、ADC 输入时钟分频等

```
Void ADC_CommonInit(ADC_CommonInitTypeDef* ADC_CommonInitStruct);
typedef struct
{
    unit32_t ADC_Mode; //ADC 模式选择
    unit32_t ADC_Prescaler; //ADC 分频系数
    unit32_t ADC_DMAAccessMode;//ADC DMA 模式配置
    unit32_t ADC_TwoSamplingDelay; //ADC 采样延迟
}ADC_CommonInitTypedef;
```

3. 初始化 ADC，包括 ADC 分辨率、转换模式、数据对齐方式等

```
void ADC_Init(ADC_TypeDef* ADC×, ADC_InitTypeDef* ADC_InitStruct);
```

```
typedef struct
{
    unit32_t ADC_Resolution; //ADC 分辨率选择
    FunctionalState ADC_ScanConvMode;//ADC 扫描模式选择
    FunctionalState ADC_ContinuousConvMode;//ADC 连续转换模式选择
    unit32_t ADC_ExternalTriConvEdge;//外部触发极性
    unit32_t ADC_ExternalTrigConv;//ADC 外部触发选择
    unit32_t ADC_DataAlign;//ADC 数据对齐方式
        unit8_t ADC_NbrOfConversion;//ADC 规则序列长度
}ADC_InitTypeDef;
```

2.1.3. 开启 ADC

```
void ADC_Cmd(ADC_TypeDef* ADCx, FunctionalState NewState);
ADC_Cmd(ADC1_ENABLE);//开启 AD 转换器
```

2.1.4. 读取 ADC 转换值

设置规则序列通道以及采样周期的库函数是：

```
void ADC_RegularChannelConfig(ADC_TypeDef* ADCx, unit8_t ADC_Channel, unit8_t Rank, unit8_t ADC_SampleTime);

ADC_RegularChannelConfig(ADC1, ADC_Channel_5, 1, ADC_SampleTime_480Cycles);
```

设置好规则序列通道及采样周期，接下来就要开启转换，由于我们采用的是软件触发，库函数

```
void ADC_SoftwareStartConv(ADC_TypeDef* ADCx);
```

开启转换之后，就可以获取 ADC 转换结果数据，调用的库函数是：

```
unit16_t ADC_GetConversionValue(ADC_TypeDef* ADCx);
```

例如我们要判断 ADC1 的转换是否结束，方法是：

```
while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC));//等待转换结束
```

3. 编写 ADC 控制程序

本章所要实现的功能是：通过 ADC1 通道 5 采样外部电压值，将采样的 AD 值和转换后的电压值通过串口打印出来，同时 D1 指示灯闪烁，提示系统正常运行。程序框架如下：

1. 初始化 ADC1_IN5 相关参数，开启 ADC1

2. 编写获取 ADC1_IN5 的 AD 转换值函数
3. 编写主函数

3.1.初始化 ADC1_IN5 相关参数，开启 ADC1

```
1. void ADCx_Init()
2. {
3.     GPIO_InitTypeDef GPIO_InitStructure;//定义结构体变量
4.     ADC_CommonInitTypeDef ADC_CommonInitSructure;//定义 ADC 配置结构体变量
5.     ADC_InitTypeDef ADC_InitStructure;
6.
7.     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
8.     RCC_AHB2PeriphClockCmd(RCC_AHB2Periph_ADC1, ENABLE);
9.
10.        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;//模拟输入模式
11.        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;//管脚设置 PA5
12.        GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;//浮空
13.        GPIO_Init(GPIOA, &GPIO_InitStructure;//初始化结构体
14.
15.        ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;
16.          ADC_CommonInitStructure.ADC_TwoSamplingDelay=ADC_TwoSamplingDelay_5Cycles;
17.          ADC_CommonInitStructure.ADC_DMAAccessMode=ADC_DMAAccessMode_Disabled;
18.        ADC_CommonInitStructure.ADC_Prescaler=ADC_Prescaler_Div4;
19.        ADC_CommonInit(&ADC_CommonInitStructure);
20.
21.        ADC_InitStructure.ADC_Resolution =  ADC_Resolution_12b;//12 位模式
22.        ADC_InitStructure.ADC_ScanConvMode = DISABLE;//非扫描模式
23.        ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;//关闭连续转换
24.        ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;//禁止触发检测
25.        ADC_InitStructure.ADC_DataAlign =   ADC_DataAlign_Right;//右对齐
26.        ADC_InitStructure.ADC_NbrOfConversion = 1;//1 个转换在规则序列中，也就是只转换规则序列 1
27.        ADC_Init(ADC1, &ADC_InitStructure);//ADC 初始化
28.
29.  }
```

3.2.编写获取 ADC1_IN5 的 AD 转换值函数

```
u16 Get_ADC_Value(u8 ch, u8 times)
{
    u32 temp_val = 0;
    u8 t;

    ADC_RegularChannelConfig(ADC1, ch, 1, 480Cycles);
    ADC_SoftwareStartConv(ADC1);

    fot(t = 0; t < times; t++)
        {
             while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC));//等待转换结束

            temp_val += ADC_GetConversionValue(ADC1);
            }

        return temp_val/times;
    }
```