第1章 STM32 应用笔记之 AD 篇

1.1 ADC 特色介绍

STM32F10x Cortex-M3 核 MCU 系列芯片，集成了 1~3 路 12 位逐次比较型 ADC,其时钟速率最高可达 14MHz，可编程采样时间，固定转换时间（12.5 个时钟），最快出数据速率（信号采样时间+转换时间）可达 1MHz（14 个 clk）。2 路 ADC 提供同步、交叉、交替等灵活的工作模式，为信号的同步采样、交替采样等应用提供了极大的便利。ADC 内部还集成了校准器，以减小因内部电容器组变化而造成的精度误差。据笔者实际应用，其片内 12bit ADC 比较容易做到 11bit。该 ADC 其他性能如下：

- 在选择通道上可执行连续扫描转换；
- 外部可触发转换；
- 可编程数据对齐模式；
- 可编程通道采样时间，转换时间确定（12.5clk）；
- 转换结束（仅规则组）支持 DMA 传输。

1.2 应用简介

要掌握该 ADC 的应用需要了解两部分内容：输入通道的管理、转换模式。ADC 提供 10~21 路模拟输入，其中 2 路为内部（片上温度传感器、片上参考电压）测量。其中片上温度传感器适于测量温度的变化量，而不是实际温度值，如果要应用于比较精准的温度测量，需要提供外部温度传感器。控制器提供了比较灵活的通道管理方式，下面首先介绍输入通道的管理。

1.2.1 输入通道的管理

ADC 输入通道管理模式可分为规则组、注入组。规则组指正常的转换，而注入组则是由外部触发或者软件触发，打断正常的规则组转换（本次转换被迫复位），按照预先设置的注入通道序列进行转换，等结束转换后继续被中断的规则组转换序列的一种转换方式。规则组可是某单一的通道，也可是某几个通道的组合，而这些多通道的转换次序可编程，转换次序长度最大为 16，注入通道最长为 4。

1. 规则组

规则通道的转换顺序在 ADC_SQRx 寄存器中设置，规则组中转换的总数应写入 ADC_SQR1 寄存器的 L[3:0]位中，最长可记录 16 通道的转换顺序。转换结果记录在数据寄存器(ADC_DR) 中。规则组的转换可由外部事件触发、软件触发进行转换。

应用举例：4 路 ADC 输入通道，分别为 0、2、3、9，转换顺序分别为 2、0、9、3，假设采用间隔为 71.5clk，其设置如下：

```
ADC_InitStructure.ADC_NbrOfChannel = 4;
ADC_RegularChannelConfig(ADC1, ADC_Channel_2 , 1, ADC_SampleTime_71Cycles5);
ADC_RegularChannelConfig(ADC1, ADC_Channel_0 , 2, ADC_SampleTime_71Cycles5);
ADC_RegularChannelConfig(ADC1, ADC_Channel_9 , 3, ADC_SampleTime_71Cycles5);
ADC_RegularChannelConfig(ADC1, ADC_Channel_3 , 4, ADC_SampleTime_71Cycles5);
```

特别说明：因为规则组的转换结果均保存在寄存器 ADC_DR 中，因此要完成多通道的转换需要进行特别处理。处理方式有 2 种：借助 DMA 进行传输、中断手工数据搬运方式。

a) DMA 传输方式

ADC 每次转换完成后将产生 EOC 标志（并不产生 EOC 中断，除非使能了 EOC 中断），并通知 DMA 将数据拷贝到预先配置的 SRAM 中。这个过程完全由 CPU 自行完成。仅当 DMA 配置的传输数据长度满后才产生 DMA 中断，通知该次转换完成。该方式必须配置并使能 ADC 的 DMA 功能。DMA 配置举例如下：

```
#define     buf_size    10
      uint16_t pbuf[buf_size];
       ……
      DMA_DeInit(DMA1_Channel1);
       DMA_InitStructure.DMA_PeripheralBaseAddr = ADC1_DR_Address;       // 从 ADC_DR 中拷贝数据
       DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)pbuf;             //  ADC 转换结果保存缓冲
       DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;                 //  DMA 数据拷贝方向
       DMA_InitStructure.DMA_BufferSize = buf_size;                       //  DMA 拷贝长度
       DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;   //  外设地址是否增加
       DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;            //  RAM 地址是否增加
       DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;// 外设数据类型
       DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;     // RAM 数据类型
       DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;                    // 转换模式
       DMA_InitStructure.DMA_Priority = DMA_Priority_High;                // 优先级
       DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;                       // 是否允许 MEM 到 MEM 传输
       DMA_Init(DMA1_Channel1, &DMA_InitStructure);

       DMA_ITConfig(DMA1_Channel1, DMA_IT_TC, ENABLE);                    // 允许 DMA 传输完中断
       DMA_Cmd(DMA1_Channel1, ENABLE);                                    // 使能 DMA 中断
       ADC_DMACmd(ADC1, ENABLE);                                          // 使能 ADC 的 DMA 功能
```

b) 中断方式

该种方式，必须使能 EOC 中断。并且每次转换结束后，需要软件把转换结果拷贝到对应通道的 SRAM 中。该种方式如果软件处理不当，容易造成通道间的转换数据混乱、通道数据整体移位现象，如 1 通道的转换结果变成了 2 通道的转换数据，2 通道的转换数据变成 6 通道的转换结果。

```
       ADC_ITConfig(ADC1, ADC_IT_EOC, ENABLE);
```

2. 注入组

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应写入 ADC_JSQR 寄存器的 L[1:0]位中，最大可记录 4 通道的转换顺序。注入组中的每个通道的转换结果都保存在独自的数据寄存器中。其中注入组的转换可由外部事件触发、软件触发进行转换。

注意：注入组需启动规则组的转换。

应用举例：ADC 通道 1、3、6、8 被设置为注入组，转换顺序为 6、1、8、3，转换完成后产生中断，数据按照转换顺序进行存储，采集时间为 71.5clk。

```
       uint16_t dat[4] = {0};
       ……
       ADC_InitStructure.ADC_NbrOfChannel = 1;
```

```
    ADC_InjectedSequencerLengthConfig(ADC1, 4);                    // 配置注入组总长度
    ADC_InjectedChannelConfig(ADC1, ADC_Channel_6, 1, ADC_SampleTime_71Cycles5);
    ADC_InjectedChannelConfig(ADC1, ADC_Channel_1, 2, ADC_SampleTime_71Cycles5);
    ADC_InjectedChannelConfig(ADC1, ADC_Channel_8, 3, ADC_SampleTime_71Cycles5);
    ADC_InjectedChannelConfig(ADC1, ADC_Channel_3, 4, ADC_SampleTime_71Cycles5);
    ADC_ExternalTrigInjectedConvConfig(ADC1, ADC_ExternalTrigInjecConv_None);
    ADC_ExternalTrigInjectedConvCmd(ADC1, DISABLE);                // 禁止外部触发
    ADC_AutoInjectedConvCmd(ADC1,ENABLE);
    ADC_ITConfig(ADC1, ADC_IT_JEOC, ENABLE);
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);                        // 触发规则组转换
    ADC_SoftwareStartInjectedConvCmd(ADC1, ENABLE);                // 软件触发注入组的转换
    ……
void   ADC_ISR(void)
{
    For (int i = 0;i < 4;i++) {
        Dat[i]ADC_GetInjectedConversionValue(ADC1, ADC_InjectedChannel_1 + 4 * i);
    }
    ADC_ClearITPendingBit(ADC1, ADC_IT_JEOC);
}
```

1.2.2 转换模式

该 ADC 的工作模式主要有：单次（single conversion mode）、连续（continous conversion mode）、间断模式（discontinuous conversion mode）。以下进行详细介绍。

1. 单通道单次、连续转换

单次转换模式，只转换一次，无论规则组还是注入组里所选择的通道转换完成，ADC 置位 EOC 或者 JEOC 后，如果使能了相应中断则产生对应中断，ADC 就停止。软件、外部事件再触发，ADC 才进行下一次转换。而连续模式则不需要软件、外部事件的触发自动进行下一次转换。因此单通道的单次、连续转换模式主要针对某单一输入通道进行转换。

应用举例：ADC 输入通道 2 输入，通道采集时间 71.5clk 软件触发，触发间隔 0.5ms，产生中断。

```
    ……
    Global uint16_t GuiAdcDat = 0;
    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;   // 独立 ADC 模式，区别于双 ADC 模式
    ADC_InitStructure.ADC_ScanConvMode = DISABLE;        // 扫描模式
    ADC_InitStructure.ADC_ContinuousConvMode = DISABLE; // 是否使能连续转换
#if   通道配置为规则组
ADC_InitStructure.ADC_NbrOfChannel = 1;
    ADC_RegularChannelConfig(ADC1, ADC_Channel_2 , 1, ADC_SampleTime_71Cycles5);
     ADC_ITConfig(ADC1, ADC_IT_EOC, ENABLE);
#else
    ADC_InjectedSequencerLengthConfig(ADC1, 1);                    // 配置注入组总长度
    ADC_InjectedChannelConfig(ADC1, ADC_Channel_2, 1, ADC_SampleTime_71Cycles5);
    ADC_ExternalTrigInjectedConvConfig(ADC1, ADC_ExternalTrigInjecConv_None);
```

```
    ADC_ExternalTrigInjectedConvCmd(ADC1, DISABLE);                    // 禁止外部触发
    ADC_AutoInjectedConvCmd(ADC1,ENABLE);                              // 禁止自动转换
     ADC_ITConfig(ADC1, ADC_IT_JEOC, ENABLE);
#endif
    ……

void ADC1_ISR
{
#if   通道配置为规则组
    GuiAdcDat = ADC_GetConversionValue(ADC1);
    ADC_ClearITPendingBit(ADC1, ADC_IT_EOC);
#else
    GuiAdcDat = ADC_GetInjectedConversionValue(ADC1, ADC_InjectedChannel_1);
    ADC_ClearITPendingBit(ADC1, ADC_IT_JEOC);
#endif
}
    ……
static   void   task_start (void *p_arg)
{
    while(1) {
     #if    通道配置为规则组
          ADC_SoftwareStartConvCmd(ADC1, ENABLE);
    #else
          ADC_SoftwareStartInjectedConvCmd(ADC1, ENABLE);
    #endif
         OSTimeDlyHMSM(0, 0, 0, 100); // 本举例中仅进行粗略定时，如果要精准定时，请采用定时器
    }
}
```

2. 多通道的单次、连续转换

上面谈到某单一通道可进行单次、连续转换，如果规则组或者注入组里的转换序列长度大于 1，ADC 又是如何进行转换的呢？ADC 提供了通道扫描功能。当使能了该功能后，同一组上一通道转换结束后，将自动启动同组的下一通道进行转换。单次转换模式中，同一组的最后一通道转换结束后，ADC 将停止，直到下次触发；连续转换模式中，ADC 将从该组的第一转换序列开始，周而复始。因此该方式下，如果通道配置成规则组，建议采用 DMA 方式，让 CPU 自动在每次数据转换结束后，通知 DMA 进行数据拷贝，减少软件的复杂度。

应用举例：ADC 输入通道 0、2、3、4，按照 0、2、3、4 进行转换，采样间隔 71.5clk，采用 DMA 传输方式，单通道采样次数 10 次。

```
#define    ADC_VONVERTOR_CNT        10
#define    ADC_INPUT_CHN_NUM        4
    ……
    Global   uint16_t   GuiAdcValue[ADC_VONVERTOR_CNT * ADC_INPUT_CHN_NUM]


    DMA_DeInit(DMA1_Channel1);
```

```
DMA_InitStructure.DMA_PeripheralBaseAddr = ADC1_DR_Address;      // 从 ADC_DR 中拷贝数据
DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t) GuiAdcValue; //   ADC 转换结果保存缓冲
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;                //   DMA 数据拷贝方向
DMA_InitStructure.DMA_BufferSize = ADC_VONVERTOR_CNT * ADC_INPUT_CHN_NUM;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;   //   外设地址是否增加
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;        //   RAM 地址是否增加
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;// 外设数据类型
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;     // RAM 数据类型
DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;                  // 转换模式
DMA_InitStructure.DMA_Priority = DMA_Priority_High;              // 优先级
DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;                     // 是否允许 MEM 到 MEM 传输
DMA_Init(DMA1_Channel1, &DMA_InitStructure);

DMA_ITConfig(DMA1_Channel1, DMA_IT_TC, ENABLE);                  // 允许 DMA 传输完中断
DMA_Cmd(DMA1_Channel1, ENABLE);                                  // 使能 DMA 中断
ADC_DMACmd(ADC1, ENABLE);                                        // 使能 ADC 的 DMA 功能
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;   // 独立 ADC 模式，区别于双 ADC 模式
ADC_InitStructure.ADC_ScanConvMode = ENABLE;          // 扫描模式
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE; // 是否使能连续转换
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfChannel = 4;
ADC_Init(ADC1, &ADC_InitStructure);
ADC_RegularChannelConfig(ADC1, ADC_Channel_0 , 1, ADC_SampleTime_71Cycles5);
ADC_RegularChannelConfig(ADC1, ADC_Channel_2 , 2, ADC_SampleTime_71Cycles5);
ADC_RegularChannelConfig(ADC1, ADC_Channel_3 , 3, ADC_SampleTime_71Cycles5);
ADC_RegularChannelConfig(ADC1, ADC_Channel_4 , 4, ADC_SampleTime_71Cycles5);
ADC_Cmd(ADC1, ENABLE);                                           // 启动 ADC 的转换
```

特别说明：在多通道连续扫描模式下，请不要再次调用软件触发 ADC 的转换，否则会出现 ADC 转换结果整体偏移的现象。错误的函数调用如下：

```
ADC_SoftwareStartConvCmd(ADC1, ENABLE);          // 启动规则组的转换
ADC_SoftwareStartConvCmd(ADC1, DISABLE);         // 禁止规则组的转换
```

以上调用函数仅用于软件触发模式，进行单次转换。而不应出现在连续转换模式下。联系模式中正确启动、停止 ADC 转换函数如下：

```
ADC_Cmd(ADC1, ENABLE);                           // 启动 ADC 的转换
ADC_Cmd(ADC1, DISABLE);                          // 禁止 ADC 的转换
```

3. 间断模式

间断采样功能，主要是对多通道转换序列组合的补充。间断长度为 n（n < 8），数值 n 由 ADC_CR1 寄存器的 DISCNUM[2:0]设置。它由外部事件触发转换。例如某规则通道序列为 0、2、3、5、7、8、9，其中 n 为 3,，那么转换顺序如下：

第一次触发：通道 0、2、3 被转换；

第二次触发：通道 5、7、8 被转换；

第三次触发：通道 9 被转换；

第四次触发：0、2、3 被转换。

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、2、3。 而注入组类似。但是不能同时配置成自动注入转换和间断注入转换。

4. 双 ADC 模式

双 ADC 转换设置类单 ADC 的设置，而 2 路 ADC 之间的同步方式。其同步方式主要有同步注入模式、同步规则模式、快速交叉模式、慢速交叉模式、交替触发模式、独立模式。多样的同步模式可以相互组合配置成不同方式，以应用于不同场合的需要。

备注：本文采用的函数来自 STM32F10x 官方提供的软件库。笔者对 STM32 官方提供的英文用户手册该章节内容安排表示遗憾，该章节内容安排比较混乱，功能分节不清晰，模式介绍突然插入功能介绍内容，以致各模式没有明显的对比，容易让人晕头转向，浪费开发者的时间。性能如此优秀的片内 ADC 怎能配这样的内容安排呢？

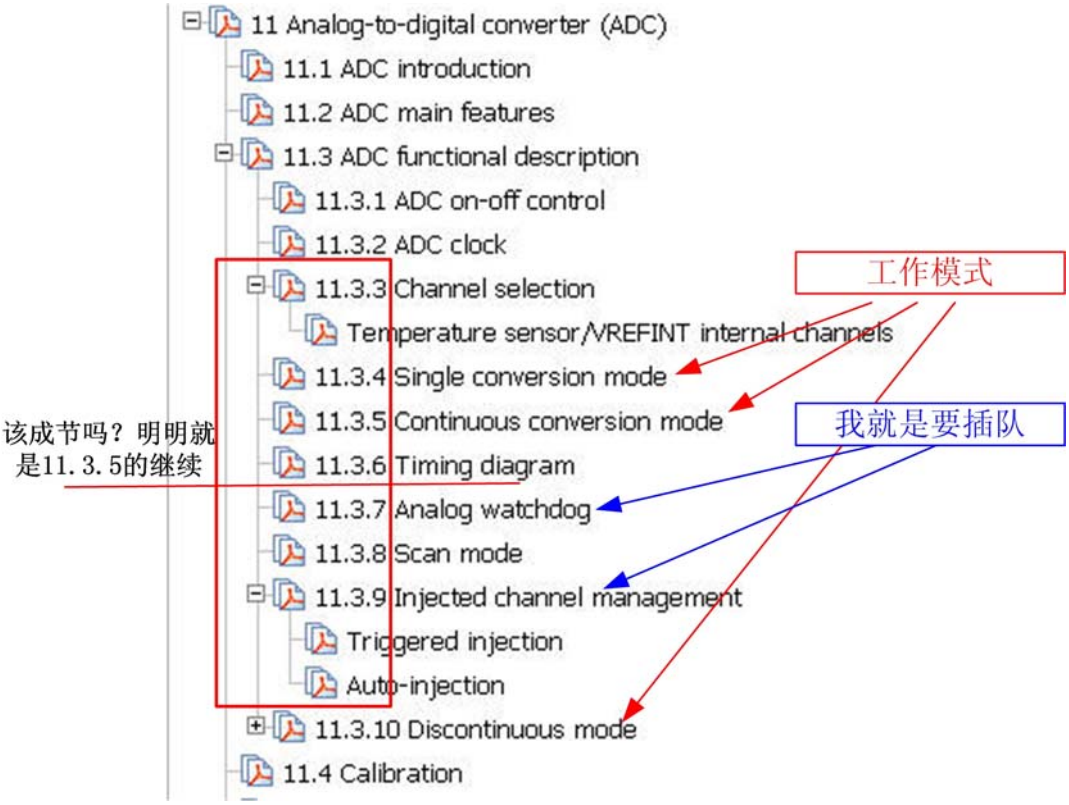

图 1-1 篇章内容编辑图