应用笔记

Application Note

文档编号：**AN1083**

APM32F4xx_ADC 应用笔记

版本：**V1.1**

© 珠海极海半导体有限公司

1 引言

本应用笔记提供如何在 APM32F4xx 系列上配置和应用 ADC 接口的指南，包括接口框图、代码实现和应用方法。

APM32F4xx 微控制器内置最多三个 12 位的逐次逼近型 ADC，ADC 最多有 16 个外部输入通道和 3 个内部通道，并提供自校准功能。内部通道分别提供测量芯片内置温度传感器电压、参考电压以及备份电源电压的功能。各个 A/D 转换通道支持单次、连续、扫描和间断的转换方式。ADC 转换的结果可以设置成左对齐或右对齐来存储在 16 位的数据寄存器中，并且支持 DMA 访问和设置模拟看门狗。

1.1 支持的型号

支持的型号	APM32F405xx(1)、APM32F407xx(1)、APM32F417xx(1)、APM32F465xx(1)、APM32F411xx(2)

注意：(1) 内置 ADC1、ADC2 和 ADC3。

(2) 内置 ADC1 和 ADC2。

目录

2 ADC 简介

模数转换器，ADC(**A**nalog to **D**igital **C**onverter)，是一个将模拟信号转换为数字信号的器件(电路)，例如将温度、湿度、压力、位置等信息转换为数字信号。但由于数字信号本身不具有实际意义，仅仅表示一个相对大小。所以 ADC 都需要一个参考模拟量（REF）作为转换的标准。

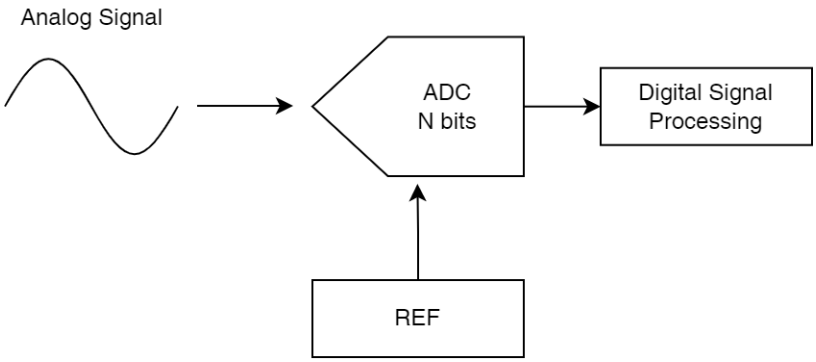

图 1 ADC 框图

2.1 ADC 分类

ADC 按工作原理可以分成直接 ADC 和间接 ADC。主要有以下几种：

并联比较型 ADC；

逐次逼近型 ADC；

双积分型 ADC。

其中逐次逼近型 ADC 是一种直接 ADC。由于其采样速率中等，分辨率中等，且位数较多时使用元器件较少等原因（成本较低），所以被广泛应用于集成 ADC 中。

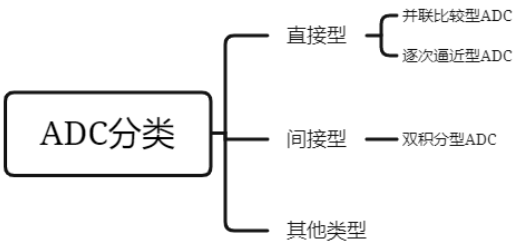

图 2 ADC 分类

2.2 A/D 转换原理

A/D 转换的作用是将时间、幅值连续的模拟信号转换为时间、幅值离散的数字信号。所以，A/D 转换一般要经过采样、保持、量化及编码四个过程。

2.3 A/D 转换步骤

2.3.1 采样和保持

采样是指在时间上将模拟信号离散化，即是将时间上连续的信号转为一系列等时间间隔的信号离散序列。其中离散信号脉冲的幅度取决于输入模拟量。

2.3.2 量化和编码

量化是用有限个幅度值近似原来连续变化的幅度值，把模拟信号的连续幅度变为有限数量的有一定间隔的离散值。而编码则是按照一定的规律，把量化后的值用二进制数字表示。下图列举了 12bits ADC 的 FSR 为 3.3V 时的量化到编码的过程。其中：

N：分辨率，用于对输入进行量化的位数。理论上，n 位输出的 ADC 能区分 2^n 个不同等级的模拟输入电压。如下图所示，能分辨的最小输入电压步长 LSB = FSR / 2^n = 806uV；

FSR： Full-Scale Range，满量程；

LSB： Least Significant Bit，最低有效位；

MSB： Most Significant Bit，最高有效位。

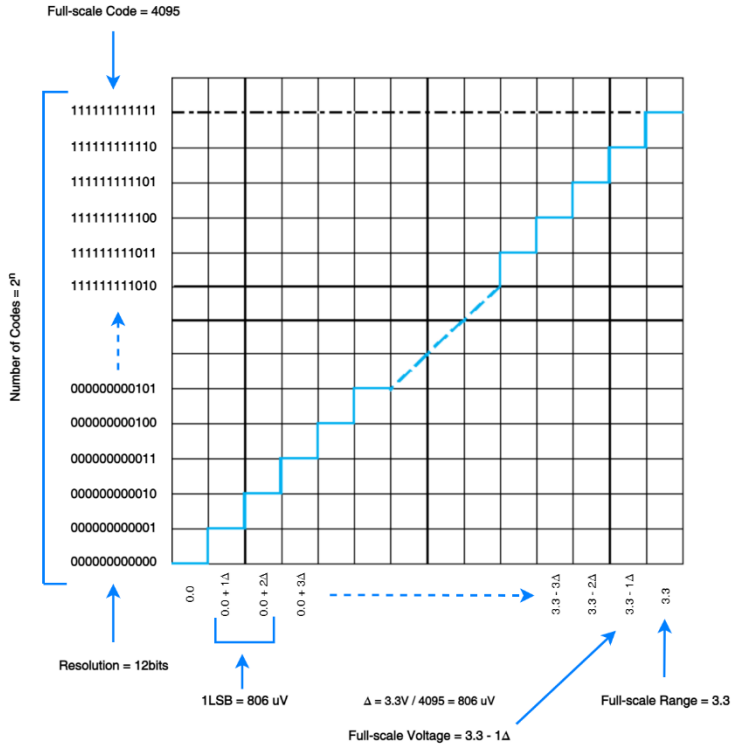

图 3 量化和编码

2.3.3 转换时间

转换时间是指 ADC 从转换控制信号触发开始，到输出端得到稳定的数字信号所经过的时间。该时间受 ADC 类型、ADC 时钟和外部输入阻抗等因素影响。

3 APM32 中的 ADC

APM32F4xx 系列产品最多有 3 个 ADC（APM32F411xx 系列最多有 2 个 ADC），精度为 12 位，每个 ADC 最多有 16 个外部通道和 3 个内部通道，各通道 A/D 转换模式有单次、连续、扫描或间断，ADC 转换结果可以左对齐或右对齐存储在 16 位数据寄存器中。

APM32 中的 ADC 是逐次逼近型 ADC(Successive Approximation ADC)，是逐个产生比较电压 VREF，并逐次与输入电压分别比较，以逐渐逼近的方式进行 A/D 转换的。

SAR ADC 的转换原理是把输入的模拟信号按规定的时间间隔采样（采样），并与一系列标准的数字信号相比较，数字信号逐次收敛，直至两种信号相等为止（量化），最后输出代表此信号的二进制数（编码）。

3.1 ADC 的特征

(1) 供电要求：全速运行为 2.4V 到 3.6V，慢速运行为 1.8V

(2) 输入范围：VREF- ≤VIN ≤VREF+

(3) 分辨率可配置 12 位、10 位、8 位或 6 位分辨率

(4) 支持规则数据转换的 DMA 请求

(5) 独立 ADC 模式 、双重/三重 ADC 模式(1)

(6) 支持转换结束中断、模拟看门狗中断和溢出中断

(7) 支持定时器或外部引脚触发

(8) 可独立设置各通道采样时间

(9) 可配置数据对齐方式为左对齐或右对齐

(10) 支持扫描模式、间断模式、注入通道管理

(11) 时钟由数字和模拟时钟两个部分组成

注意：(1) APM32F411xx 系列不支持

3.2 ADC 的结构

结构上主要包括采样保持电路(S/H)，比较器(COMPARATOR，COMP)，SAR 逻辑控制电路、时钟(CLOCK)和时序(TIMING)控制电路及 DAC 电路。

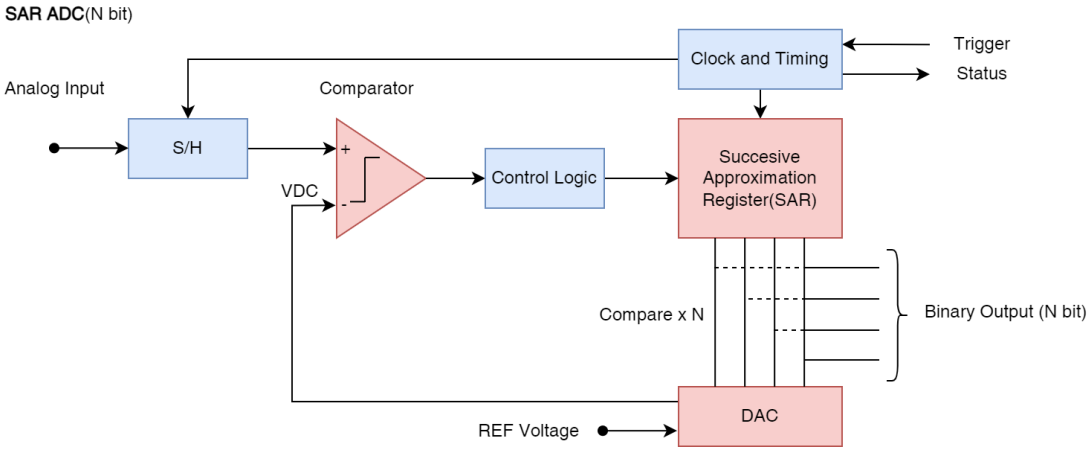

图 4 ADC 结构

3.3 S/H 电路

被采样的脉冲宽度一般是很短的，在下一个采样脉冲到来之前，要暂时保持所采得的样值脉冲幅度，以便进行后续转换。所以，在采样电路之后要加保持电路。下图是一个简单的采样保持电路配置框图。

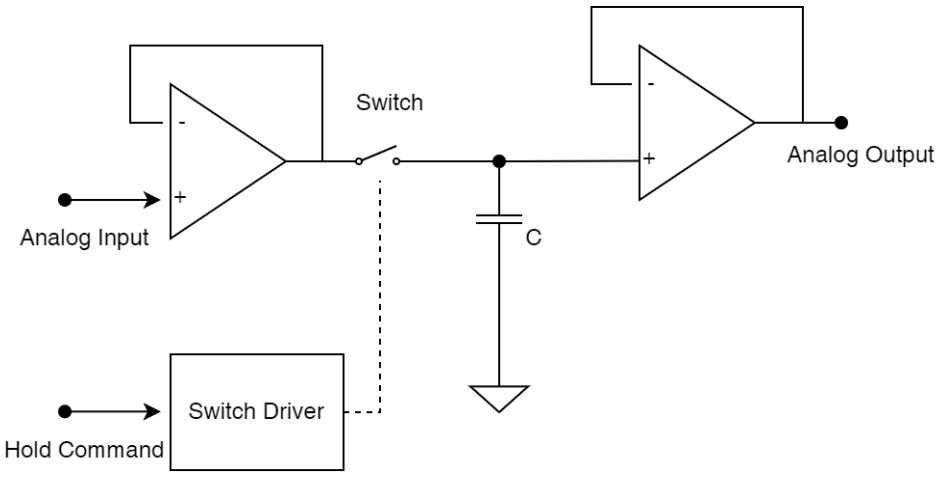

图 5 S/H 电路

3.4 DAC 电路

大多 SAR ADC 的 DAC 都使用电容式 DAC 来提供内在的跟踪/保持功能。电容式 DAC 是采用电荷再分配原理来产生模拟输出电压的。电容式 DAC 由 N 个具有二进制权重值的电容器阵列再加上一个“虚拟 LSB”电容器组成。

3.5 转换步骤

转换步骤数等于 ADC 的分辨率，比如 10bits ADC 就有 10 个转换步骤，每个 ADC 时钟产生一个数据位。以下步骤以 10bits ADC 为例。采样和保持的状态，可以参考上述的 S/H 等效电路来理解。下面着重看下量化和编码的状态。

3.5.1 量化和编码状态

该状态下，每个 ADCCLK 执行一个步骤，每一步完成后 ADC 输出一位数。采用二分法进行逐次逼近到 ADC 的精度（位数）。整个转换过程如下图所示。

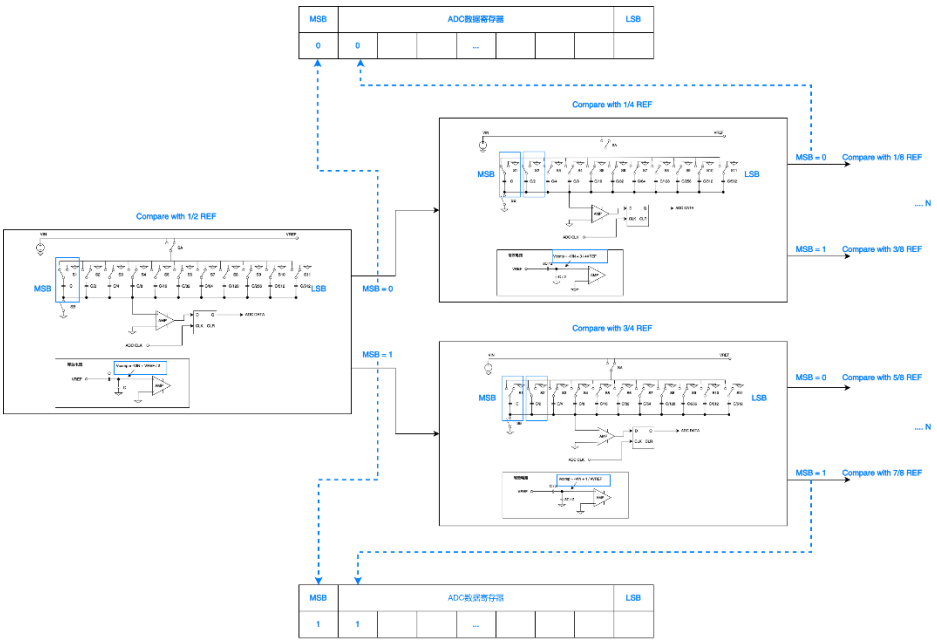

图 6 量化和编码状态

3.5.1.1 转换例子

比如 2.5V 输入到以 3.3V 为参考电压的 SAR ADC 中，则转换过程如下所示。

第一个逼近步骤时，MSB 先设置为 1。DAC 以 1/2 REF 去和 VIN 比较，若 VIN > 1/2 REF，则保持 MSB = 1（反之则 MSB = 0）。等待下一个 ADCCLK，执行下一步。

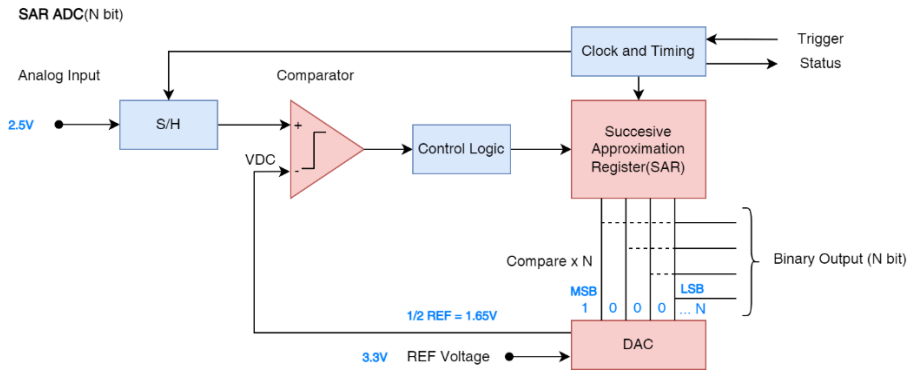

图 7 ADC 转换逼近步骤 1

第二个逼近步骤时，MSB 往后移动 1 个 bit，再以 3/4 REF 去和 VIN 进行比较，若 VIN > 3/4 REF，则保持 MSB = 1（反之则 MSB = 0）。等待下一个 ADCCLK，执行下一步，一直到所有 bit 确定，然后输出编码值。

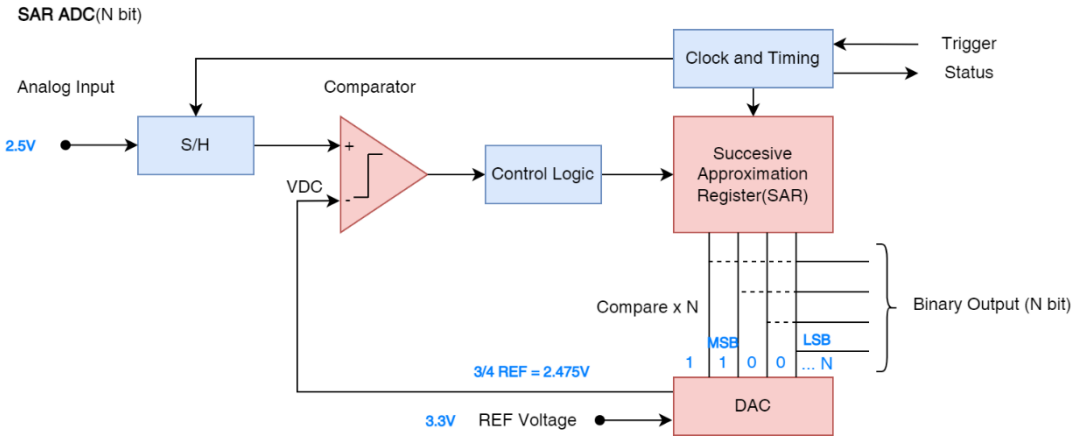

图 8 ADC 转换逼近步骤 2

3.6 转换时间

APM32F4xx 中的 ADC 转换时间 = 采样周期 + 转换周期。

3.6.1 采样周期

由**采样周期设置**决定，要注意的是，该值需要和外部电路的输入阻抗匹配。从而保证在采用阶段，采样保持电容有足够的时间充电。

3.6.2 转换周期

该值取决于 ADC 的转换精度，APM32F4xx 的 SAR ADC 默认为 12bits，可配置为 12、10、8、6bits。

表格 1 ADC 精度和转换周期关系

序号	ADC 精度	转换周期
1	12 bits	12 x ADCCLK
2	10 bits	10 x ADCCLK
3	8 bits	8 x ADCCLK
4	6 bits	6 x ADCCLK

3.7 转换数值

ADC 转换的数值 = (VIN x 2^n) / VREF，n 为 ADC 的分辨率。以上述 12bits 的 ADC 为例，则

ADC 转换的数值 = (VIN x 4096) / VREF。

4 ADC 的配置和应用

4.1 硬件设计

4.1.1 输入通道

APM32 MINI 板已经将部分 ADC 的通道通过排针接出，可以根据设计需求使用相关 IO。ADC 采样容易受外接干扰，使用时注意引脚间的抗干扰设计，以及避免 ADC 引脚和其他功能电路共用的情况。

4.1.2 电压输入范围

ADC 的电压输入范围为 V_{REF-} ~ V_{REF+}，MINI 板上的 VSSA 和 V_{REF-}接入 GND，而 VDDA 和 V_{REF+}接入 VDD，所以在 APM32 MINI 板上的 ADC 电压输入范围是 0V ~ 3.3V。

4.2 软件设计

软件设计上这里只讲解关键的配置，部分查询和逻辑代码未涉及，详细可直接参考 SDK 中相关配套的例程。(路径：APM32xxx_DAL_SDK_vx.x\Examples\xxx\ADC)

表格 2 演示板配套例程

IP / Module	Example	APM32F407_MINI	APM32F407_TINY	APM32F407_EVAL	APM32F465_MINI	APM32F411_TINY	NA	NA
ADC	ADC_AnalogWindowWatchdog	√	√	X	√	√		
	ADC_ContinuousConversion	√	√	X	√	√		
	ADC_DualInterleavedMode	√	√	X	√	X		
	ADC_DualRegulSimulMode	√	√	X	√	X		
	ADC_MultiChannelScan	√	√	X	√	√		
	ADC_TemperatureSensor	√	√	X	√	√		
	ADC_TripleInterleavedMode	√	√	X	√	X		
	ADC_DMA	√	√	X	√	√		
	ADC_VBAT	√	√	X	√	√		
	ADC_ContinuousConversionADC2	X	X	X	X	√		

4.2.1 ADC_HandleTypeDef 结构体

ADC_HandleTypeDef 结构体定义在 apm32f4xx_dal_adc.h 文件中，具体定义如下：

```
/**
 * @brief  ADC handle Structure definition
 */
typedef struct
{
    ADC_TypeDef                  *Instance;
    ADC_InitTypeDef              Init;
    __IO uint32_t                NbrOfCurrentConversionRank;
    DMA_HandleTypeDef            *DMA_Handle;
    DAL_LockTypeDef              Lock;
    __IO uint32_t                State;
    __IO uint32_t                ErrorCode;
} ADC_HandleTypeDef;
```

结构体中的参数含义：

表格 3 ADC_HandleTypeDef 结构体

参数	含义
*Instance	ADC 寄存器基址指针
Init	ADC 初始化结构体
NbrOfCurrentConversionRank	正在转换序列的 ADC 数量
*DMA_Handle	DMA 处理指针
Lock	ADC 锁定对象
State	ADC 转换状态
ErrorCode	ADC 错误代码

4.2.2 ADC_InitTypeDef 结构体

ADC_InitTypeDef 结构体在上述的 ADC_HandleTypeDef 中被引用。其也定义在 apm32f4xx_dal_adc.h 文件中，具体定义如下：

```
/**
 * @brief  Structure definition of ADC and regular group
initialization
  */
typedef struct
{
  uint32_t ClockPrescaler;
  uint32_t Resolution;
  uint32_t DataAlign;
  uint32_t ScanConvMode;
  uint32_t EOCSelection;
  FunctionalState ContinuousConvMode;
  uint32_t NbrOfConversion;
  FunctionalState DiscontinuousConvMode;
  uint32_t NbrOfDiscConversion;
  uint32_t ExternalTrigConv;
  uint32_t ExternalTrigConvEdge;
  FunctionalState DMAContinuousRequests;
} ADC_InitTypeDef;
```

表格 4 ADC_InitTypeDef 结构体

参数	含义
ClockPrescaler	用于配置 ADC 时钟的分频系数，ADC 时钟由 PCLK2 提供。PCLK2 除以 ClockPrescaler 就是 ADC 的时钟
Resolution	用于配置 ADC 的分辨率，可以配 ADC 的分辨率为 12bit、10bit、8bit 和 6bit。如本应用说明 SAR ADC 转换原理所述，ADC 的分辨率配置越高，相对的转换时间也越长
DataAlign	用于配置 ADC 转换结果的数据对齐方式，一般按照习惯，我们配置为右对齐模式
ScanConvMode	用于配置是否使扫描模式，一般单通道 A/D 转换应用时配置为 DISABLE，多通道 A/D 转换应用时配置为 ENABLE
EOCSelection	指定通过轮询或中断模式来使用转换结束标志进行转换
ContinuousConvMode	用于配置是单次转换，还是启用自动连续转换模式
NbrOfConversion	ADC 规则转换通道数量
DiscontinuousConvMode	配置不连续采样模式
NbrOfDiscConversion	ADC 不连续转换通道数量
ExternalTrigConv	用于配置外部触发源
ExternalTrigConvEdge	用于配置外部触发的极性
DMAContinuousRequests	配置 DMA 请求连续转换

4.2.3 ADC_ChannelConfTypeDef 结构体

ADC_ChannelConfTypeDef 结构体定义在 apm32f4xx_dal_adc.h 文件中，具体定义如下：

```
/**
  * @brief  Structure definition of ADC channel for regular group
  */
typedef struct
{
  uint32_t Channel;
  uint32_t Rank;
  uint32_t SamplingTime;
  uint32_t Offset;
} ADC_ChannelConfTypeDef;
```

结构体中的参数含义：

表格 5 ADC_ChannelConfTypeDef 结构体

参数	含义
Channel	ADC 转换通道
Rank	ADC 序列号
SamplingTime	ADC 采样时间
Offset	预留量，未使用

4.2.4 单通道转换软件设计

以“ADC_ContinuousConversion”例程为例。

配置 ADC

开启 ADC 和 GPIO 时钟后，配置 GPIO 为模拟输入模式。

```
void DAL_ADC_MspInit(ADC_HandleTypeDef* hadc)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0U};

    if(hadc->Instance == ADC1)
    {
        __DAL_RCM_ADC1_CLK_ENABLE();

        __DAL_RCM_GPIOA_CLK_ENABLE();

        /* ADC1 pin configuration */
        GPIO_InitStruct.Pin     = GPIO_PIN_0;
        GPIO_InitStruct.Mode    = GPIO_MODE_ANALOG;
        GPIO_InitStruct.Pull    = GPIO_NOPULL;
        DAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
    }
}
```

配置 ADC 工作方式，开启连续转换模式。

```
void DAL_ADC1_Config(void)
{
    ADC_ChannelConfTypeDef Channel_ConfigStruct = {0};

    /* Configure the ADC1 */
    hadc1.Instance                   = ADC1;
    hadc1.Init.ClockPrescaler        = ADC_CLOCK_SYNC_PCLK_DIV4;
    hadc1.Init.Resolution            = ADC_RESOLUTION_12B;
    hadc1.Init.ScanConvMode          = DISABLE;
    hadc1.Init.ContinuousConvMode    = ENABLE;
    hadc1.Init.DiscontinuousConvMode = DISABLE;
    hadc1.Init.ExternalTrigConvEdge  = ADC_EXTERNALTRIGCONVEDGE_NONE;
    hadc1.Init.ExternalTrigConv      = ADC_SOFTWARE_START;
    hadc1.Init.DataAlign             = ADC_DATAALIGN_RIGHT;
    hadc1.Init.NbrOfConversion       = 1U;
    hadc1.Init.DMAContinuousRequests = DISABLE;
    hadc1.Init.EOCSelection          = ADC_EOC_SINGLE_CONV;
    if (DAL_ADC_Init(&hadc1) != DAL_OK)
    {
        DAL_ErrorHandler();
    }
```

配置开启 ADC 中断。

```
void DAL_NVIC_Config(void)
{
    /* Set interrupt group priority */
    DAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_4);

    /* ADC interrupt */
    DAL_NVIC_SetPriority(ADC_IRQn, 0U, 0U);
    DAL_NVIC_EnableIRQ(ADC_IRQn);
}
```

配置 ADC 中断服务函数

检测到转换完成后，回调该中断服务函数。并读取此时的 ADC 转换值，然后将其转换成对应电压值，该值受量化误差和硬件抗干扰能力所影响。这里转换出来的电压单位是 mV。

```
void ADC_IRQHandler(void)
{
    DAL_ADC_IRQHandler(&hadc1);
}
```

启动 ADC 转换

完成 ADC 配置和中断服务请求配置后，在 main 函数中启动 ADC 中断方式的转换。

```
int main(void)
{
    /* Device configuration */
    DAL_DeviceConfig();

    /* Strat ADC conversation */
    if (DAL_ADC_Start_IT(&hadc1) != DAL_OK)
    {
        DAL_ErrorHandler();
    }
}
```

处理转换数据

当 ADC 转换完成后，会调用默认的 DAL_ADC_ConvCpltCallback()转换完成函数。用户可以在该函数中获取转换的数据并处理。

```
void DAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
    uint16_t voltage;

    /* Get the converted value */
    adc1ConvValue = DAL_ADC_GetValue(hadc);

    voltage = (adc1ConvValue * 3300U) / 4095U;

    DAL_LOGI(tag, "ADC1 CH0 Value: %d mV\r\n",voltage);

    /* Strat ADC Conversation */
    if (DAL_ADC_Start_IT(&hadc1) != DAL_OK)
    {
        DAL_ErrorHandler();
    }
}
```

4.2.5 多通道扫描软件设计

以“ADC_MultiChannelScan”例程为例。

定义公共信息

这里定义了本样例采样的通道数为 3 个通道，并定义了将用于 DMA 存储 ADC 各通道扫描数据的数组 adc1ConvValue [3]。以上这些信息都会用于后续配置和应用中。

```
__IO uint16_t adc1ConvValue[3] = {0U};
```

配置 ADC

和单通道转换配置顺序一样，首先将要扫描的三个通道开启相应时钟和配置为模拟输入模式。DMA 的不同通道和数据流规定了其所属的外设，使用时要注意。这里配置为数据流

DMA2_Stream0 的 DMA_CHANNEL_0 通道。内存和外设模式均为半字，方向为外设到内存，并开启循环模式。

```
void DAL_ADC_MspInit(ADC_HandleTypeDef* hadc)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0U};

    if(hadc->Instance == ADC1)
    {
        __DAL_RCM_ADC1_CLK_ENABLE();

        __DAL_RCM_GPIOA_CLK_ENABLE();

        /* ADC1 pin configuration */
        GPIO_InitStruct.Pin     = GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2;
        GPIO_InitStruct.Mode    = GPIO_MODE_ANALOG;
        GPIO_InitStruct.Pull    = GPIO_NOPULL;
        DAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

        /* ADC1 DMA Init */
        hdma_adc1.Instance                 = DMA2_Stream0;
        hdma_adc1.Init.Channel             = DMA_CHANNEL_0;
        hdma_adc1.Init.Direction           = DMA_PERIPH_TO_MEMORY;
        hdma_adc1.Init.PeriphInc           = DMA_PINC_DISABLE;
        hdma_adc1.Init.MemInc              = DMA_MINC_ENABLE;
…
        if (DAL_DMA_Init(&hdma_adc1) != DAL_OK)
        {
            DAL_ErrorHandler();
        }
```

然后是 ADC 工作模式的配置，这里配置为连续扫描模式，并设置转换通道数为 3 个。

```
   void DAL_ADC1_Config(void)
{
    ADC_ChannelConfTypeDef Channel_ConfigStruct = {0};

    /* Configure the ADC1 */
    hadc1.Instance                      = ADC1;
    hadc1.Init.ClockPrescaler           = ADC_CLOCK_SYNC_PCLK_DIV4;
    hadc1.Init.Resolution               = ADC_RESOLUTION_12B;
    hadc1.Init.ScanConvMode             = ENABLE;
    hadc1.Init.ContinuousConvMode       = ENABLE;
    hadc1.Init.DiscontinuousConvMode    = DISABLE;
    hadc1.Init.ExternalTrigConvEdge     = ADC_EXTERNALTRIGCONVEDGE_NONE;
    hadc1.Init.ExternalTrigConv         = ADC_SOFTWARE_START;
    hadc1.Init.DataAlign                = ADC_DATAALIGN_RIGHT;
    hadc1.Init.NbrOfConversion          = 3U;
    hadc1.Init.DMAContinuousRequests    = ENABLE;
    hadc1.Init.EOCSelection             = ADC_EOC_SINGLE_CONV;
    if (DAL_ADC_Init(&hadc1) != DAL_OK)
    {
        DAL_ErrorHandler();
    }
    …
}
```

配置各通道的转换顺序及采样周期。

ADC 通道 0 配置在转换序列的 1 号位，采样时间为 480 Cycles。

ADC 通道 1 配置在转换序列的 2 号位，采样时间为 480 Cycles。

ADC 通道 2 配置在转换序列的 3 号位，采样时间为 480 Cycles。

```
    /* Configure for the selected ADC regular channel */
    Channel_ConfigStruct.Channel       = ADC_CHANNEL_0;
    Channel_ConfigStruct.Rank          = 1U;
    Channel_ConfigStruct.SamplingTime   = ADC_SAMPLETIME_480CYCLES;
    if (DAL_ADC_ConfigChannel(&hadc1, &Channel_ConfigStruct) != DAL_OK)
    {
        DAL_ErrorHandler();
    }

    Channel_ConfigStruct.Channel       = ADC_CHANNEL_1;
    Channel_ConfigStruct.Rank          = 2U;
    Channel_ConfigStruct.SamplingTime   = ADC_SAMPLETIME_480CYCLES;
    if (DAL_ADC_ConfigChannel(&hadc1, &Channel_ConfigStruct) != DAL_OK)
    {
        DAL_ErrorHandler();
    }

    Channel_ConfigStruct.Channel       = ADC_CHANNEL_2;
    Channel_ConfigStruct.Rank          = 3U;
    Channel_ConfigStruct.SamplingTime   = ADC_SAMPLETIME_480CYCLES;
    if (DAL_ADC_ConfigChannel(&hadc1, &Channel_ConfigStruct) != DAL_OK)
    {
        DAL_ErrorHandler();
    }
}
```

开启 DMA 时钟和中断。

```
void DAL_DMA_Config(void)
{
    __DAL_RCM_DMA2_CLK_ENABLE();
}

void DAL_NVIC_Config(void)
{
    /* Set interrupt group priority */
    DAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_4);

    /* DMA2 stream 0 interrupt */
    DAL_NVIC_SetPriority(DMA2_Stream0_IRQn, 0U, 0U);
    DAL_NVIC_EnableIRQ(DMA2_Stream0_IRQn);
}
```

配置中断服务函数

```
void DMA2_STR0_IRQHandler(void)
{
    DAL_DMA_IRQHandler(&hdma_adc1);
}
```

启动 ADC 以 DMA 方式转换

```
int main(void)
{
    /* Device configuration */
    DAL_DeviceConfig();

    /* Strat ADC conversation with DMA */
    if (DAL_ADC_Start_DMA(&hadc1, (uint32_t*)&adc1ConvValue, 3U) != DAL_OK)
    {
        DAL_ErrorHandler();
    }

    /* Infinite loop */
    while (1)
    {
    }
}
```

处理转换的数据

当 ADC 转换完成后，会调用默认的 DAL_ADC_ConvCpltCallback()转换完成函数。用户可以在该函数中获取转换的数据并处理。

```
void DAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
    DAL_LOGI(tag, "ADC1 CH0 Value: %d mV\r\n",(adc1ConvValue[0] * 3300U) / 4095U);
    DAL_LOGI(tag, "ADC1 CH1 Value: %d mV\r\n",(adc1ConvValue[1] * 3300U) / 4095U);
    DAL_LOGI(tag, "ADC1 CH2 Value: %d mV\r\n\r\n",(adc1ConvValue[2] * 3300U) / 4095U);
```

4.3 提高采样精度的硬件方法

(1) 保证参考电压噪声最小化；

(2) IO 引脚串扰最小化；

(3) 加入屏蔽减少 EMI；

(4) PCB 将模拟和数字分开布局和铺设。

4.4 提高采样精度的软件方法

4.4.1 采样平均

在硬件抗干扰能力不足的情况下，我们可以牺牲采用速率，采用软件方法进行滤波，从而得到更加稳定的采样值。

4.4.2 数字信号滤波

可通过数字低通、高通等滤波器进行软件滤波。比如知道被测信号中的噪声来自 50 Hz 供电线，通过适当的数字滤波，可以只抑制 50 Hz 频率并传输无此噪声的数据信号。

4.4.3 ADC 软件校准

如果采样值较为固定，但离目标值相差大，还可以利用线性拟合（线性校准曲线）的方式让采样值更接近目标值。

4.4.3.1 采样

首先基于标准源表采样足够多的点，如下表（点数越多，拟合越准确）。

表格 6 采样表

序号	采样值(mV)	目标值(mV)
1	96.7	100
2	194.5	200
3	498.8	500
4	796	800
5	1495	1500

4.4.3.2 拟合

在数学工具（Matlab 或 Excel 等）中做线性拟合（线性、多项式、指数皆可），得到校准公式和相关系数 R 值。

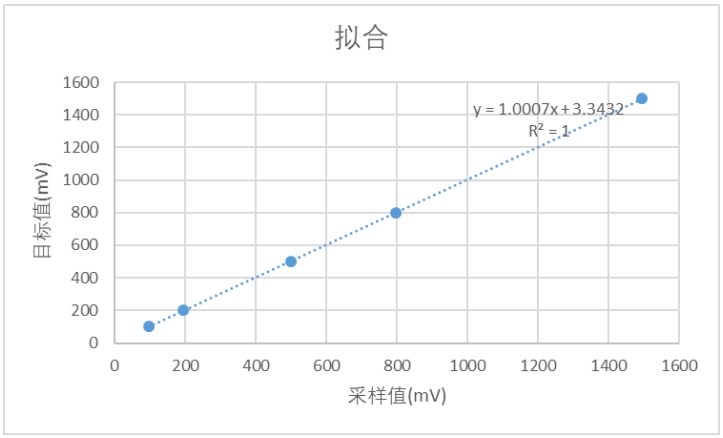

图 9 拟合

4.4.3.3 校准

用拟合步骤中得到的校准公式对采样值进行校准。

表格 7 采样表

序号	采样值(mV)	目标值(mV)	校准值(mV)
1	96.7	100	100.1
2	194.5	200	197.9
3	498.8	500	502.5
4	796	800	799.9
5	1495	1500	1499.4

5 版本历史

表格 8 文件版本历史

日期	版本	变更历史
2022.05.31	1.0	新建
2024.01.17	1.1	新增 APM32F411xx 的 ADC 描述，修改软件设计的代码为 DAL 库的代码

声明

本手册由珠海极海半导体有限公司（以下简称“极海”）制订并发布，所列内容均受商标、著作权、软件著作权相关法律法规保护，极海保留随时更正、修改本手册的权利。使用极海产品前请仔细阅读本手册，一旦使用产品则表明您（以下称“用户”）已知悉并接受本手册的所有内容。用户必须按照相关法律法规和本手册的要求使用极海产品。

1、权利所有

本手册仅应当被用于与极海所提供的对应型号的芯片产品、软件产品搭配使用，未经极海许可，任何单位或个人均不得以任何理由或方式对本手册的全部或部分内容进行复制、抄录、修改、编辑或传播。

本手册中所列带有“®”或“TM”的“极海”或“Geehy”字样或图形均为极海的商标，其他在极海产品上显示的产品或服务名称均为其各自所有者的财产。

2、无知识产权许可

极海拥有本手册所涉及的全部权利、所有权及知识产权。

极海不应因销售、分发极海产品及本手册而被视为将任何知识产权的许可或权利明示或默示地授予用户。

如果本手册中涉及任何第三方的产品、服务或知识产权，不应被视为极海授权用户使用前述第三方产品、服务或知识产权，除非在极海销售订单或销售合同中另有约定。

3、版本更新

用户在下单购买极海产品时可获取相应产品的最新版的手册。

如果本手册中所述的内容与极海产品不一致的，应以极海销售订单或销售合同中的约定为准。

4、信息可靠性

本手册相关数据经极海实验室或合作的第三方测试机构批量测试获得，但本手册相关数据难免会出现校正笔误或因测试环境差异所导致的误差，因此用户应当理解，极海对本手册中可能出现的该等错误无需承担任何责任。本手册相关数据仅用于指导用户作为性能参数参照，不构成极海对任何产品性能方面的保证。

用户应根据自身需求选择合适的极海产品，并对极海产品的应用适用性进行有效验证和测试，以确认极海产品满足用户自身的需求、相应标准、安全或其它可靠性要求；若因用户未充分对极海产品进行有效验证和测试而致使用户损失的，极海不承担任何责任。

5、合规要求

用户在使用本手册及所搭配的极海产品时，应遵守当地所适用的所有法律法规。用户应了解产品可能受到产品供应商、极海、极海经销商及用户所在地等各国有关出口、再出口或其它法律的限制，用户（代表其本身、子公司及关联企业）应同意并保证遵守所有关于取得极海产品及／或技术与直接产品的出口和再出口适用法律与法规。

6、免责声明

本手册由极海“按原样”（as is）提供，在适用法律所允许的范围内，极海不提供任何形式的明示或暗示担保，包括但不限于对产品适销性和特定用途适用性的担保。

对于用户后续在针对极海产品进行设计、使用的过程中所引起的任何纠纷，极海概不承担责任。

7、责任限制

在任何情况下，除非适用法律要求或书面同意，否则极海和/或以“按原样”形式提供本手册的任何第三方均不承担损害赔偿责任，包括任何一般、特殊因使用或无法使用本手册相关信息而产生的直接、间接或附带损害（包括但不限于数据丢失或数据不准确，或用户或第三方遭受的损失）。

8、适用范围

本手册的信息用以取代本手册所有早期版本所提供的信息。

©2022-2024　珠海极海半导体有限公司　－　保留所有权利