Application Note

Document No.: AN1083

APM32F4xx_ADC Application Note

Version: V1.0

1 Introduction

This application note provides a guide to how to configure and apply ADC interface on APM32F4xx series, including interface block diagram, code implementation and application method.

APM32F4xx MCU has up to three 12-bit ADC, which share up to 21 external input channels and 3 internal channels, and provide self-calibration function. The internal channels provide the functions of measuring the built-in temperature sensor voltage, reference voltage and backup power supply voltage respectively. Each A/D conversion channel supports single, continuous, scan and intermittent conversion. ADC conversion results can be set as left-aligned or right-aligned to be stored in 16-bit data register, and support DMA access and setting analog watchdog.

Contents

1 Introduction 1

2 ADC Introduction 3

2.1 ADC classification 3

2.2 A/D conversion principle 4

2.3 A/D conversion steps 4

3 ADC in APM32 6

3.1 ADC structure 6

3.2 S/H circuit 6

3.3 DAC circuit 7

3.4 Conversion steps 7

3.5 Conversion time 9

3.6 Converted value 10

4 Configuration and Application of ADC 11

4.1 Hardware design 11

4.2 Software design 11

4.3 Hardware method of improving sampling accuracy 19

4.4 Software method of improving sampling accuracy 19

5 Version History 22

2 ADC Introduction

Analog to **D**igital **C**onverter (ADC) is a device (circuit) that converts analog signals into digital signals, for example, converting temperature, humidity, pressure, position and other information into digital signals. However, because the digital signal itself has no practical significance, it only represents a relative size. Therefore, ADC requires a reference analog quantity (REF) as the conversion standard.

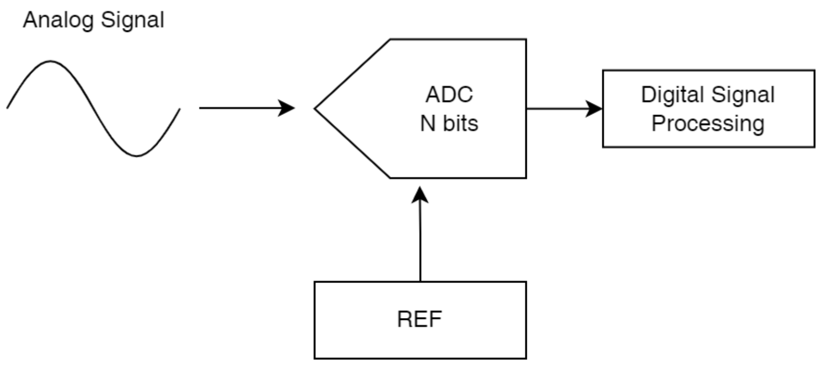

Figure 1 ADC Block Diagram

2.1 ADC classification

ADC can be divided into direct ADC and indirect ADC according to the working principle. It mainly contains the following types:

Parallel comparison ADC;

SAR ADC;

Double-integral ADC.

The SAR ADC is a direct ADC. It is widely used in integrated ADC because of its medium sampling rate, medium resolution, and use of fewer components when there are many bits (low cost).

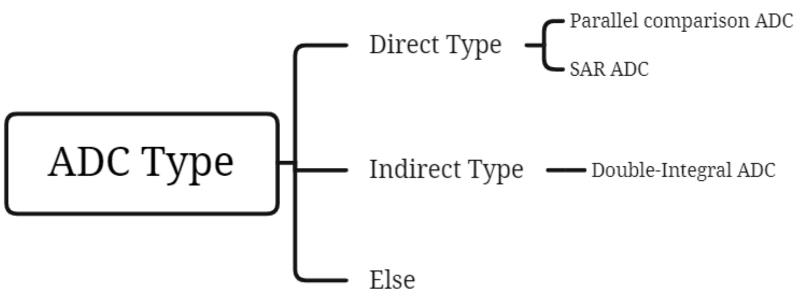

Figure 2 ADC Classification

2.2 A/D conversion principle

The function of A/D conversion is to convert **continuous** time and amplitude analog signals into **discrete** time and amplitude digital signals. Therefore, A/D conversion generally requires four processes: **sampling**, **holding**, **quantization** and **coding**.

2.3 A/D conversion steps

2.3.1 Sampling and holding

Sampling refers to discretization of analog signals in time, namely, converting continuous time signals into a series of discrete sequences of signals with equal time interval. The amplitude of discrete signal pulse depends on the input analog quantity.

2.3.2 Quantization and coding

Quantization is to use a limited number of amplitude values to approximate the original continuously changing amplitude values, and change the continuous amplitude of analog signals into a limited number of discrete values with certain interval. Coding is to express the quantized value with binary digits according to certain rules. The following figure lists the process from quantization to coding when the FSR of 12-bit ADC is 3.3V. Wherein:

N: Resolution, the number of bits used to quantize the input. Theoretically, the ADC with n-bit output can distinguish 2^n analog input voltages of different levels. As shown in the figure below, the minimum distinguishable input voltage step LSB = FSR / 2^n = 806uV;

FSR: Full-Scale Range;

LSB: Least Significant Bit;

MSB: Most Significant Bitt.

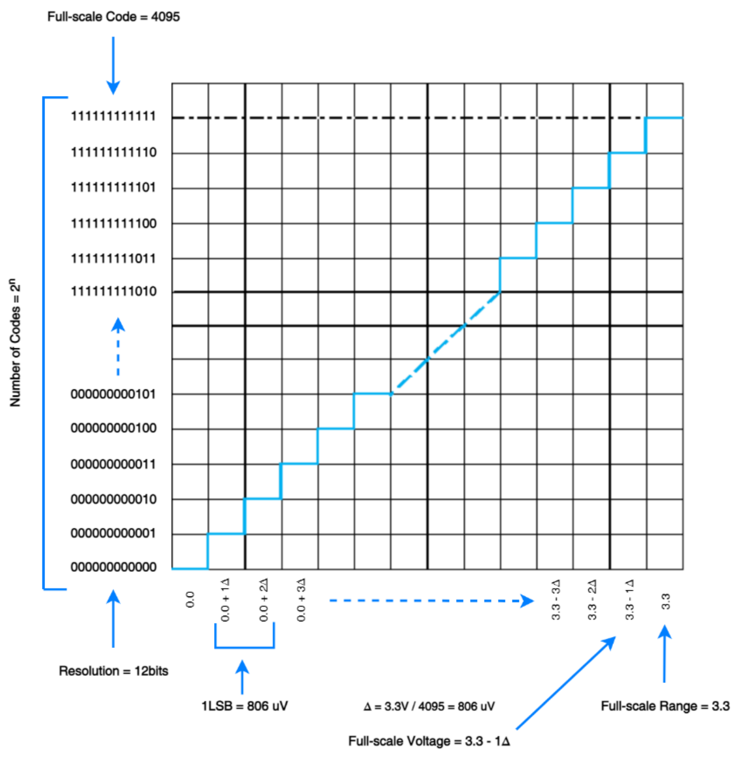

Figure 3 Quantization and Coding

2.3.3 Conversion time

The conversion time is the time from triggering the conversion control signals of ADC to obtaining the stable digital signals at the output end. This time is affected by ADC type, ADC clock and external input impedance.

3 ADC in APM32

The ADC in APM32 is a SAR ADC, which generates the comparison voltage VREF one by one, compares them with the input voltage successively, and performs A/D conversion in an increasing approximating method.

The conversion principle of SAR ADC is to sample (sampling) the input analog signal according to the specified time interval and compare it with a series of standard digital signal. The digital signal converges gradually until the two signals are equal (quantization), and finally the binary number representing this signal is output (coding).

3.1 ADC structure

The structure mainly includes sampling and holding circuit (S/H), comparator (COMP), SAR logic control circuit, clock and timing control circuit and DAC circuit.

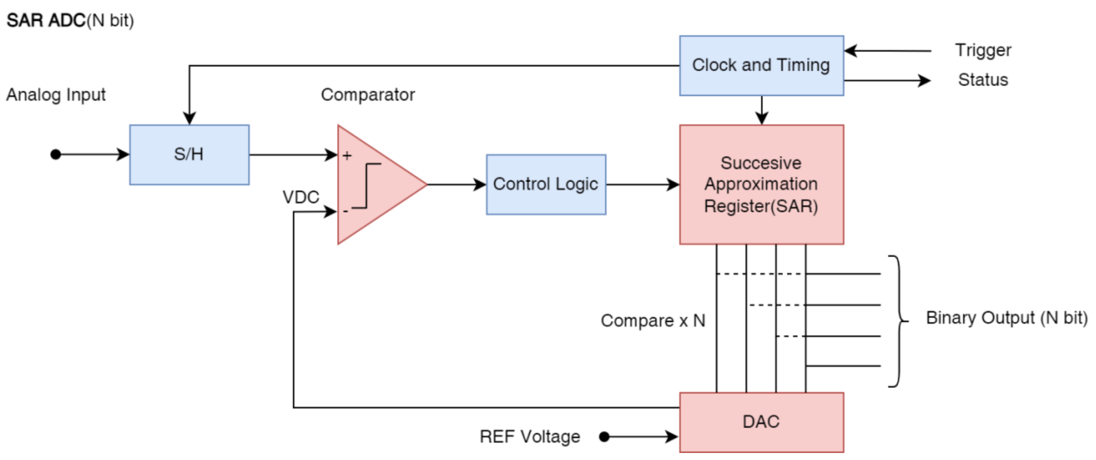

Figure 4 ADC Structure

3.2 S/H circuit

The sampled pulse width is usually very short. Before the next sampling pulse arrives, the pulse amplitude of the sampled value should be temporarily maintained for subsequent conversion. Therefore, the holding circuit shall be added after the sampling circuit. The following figure is the configuration block diagram of a simple sampling and holding circuit.

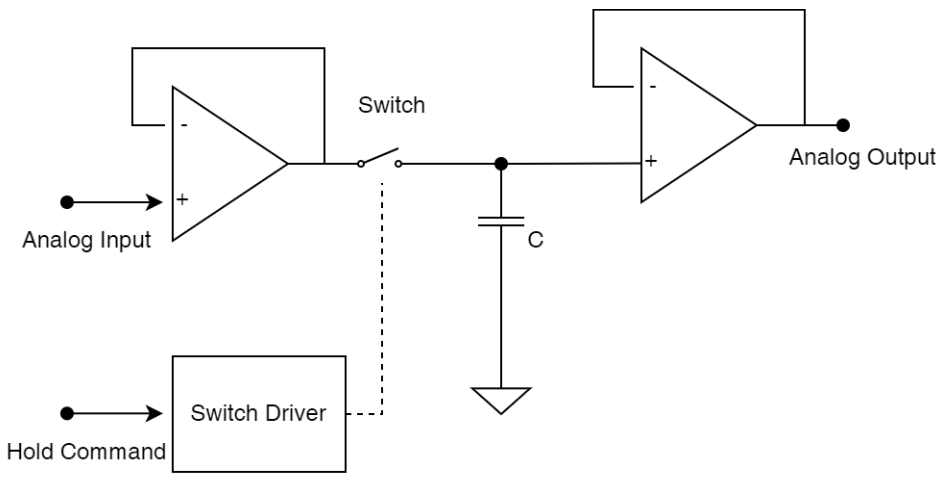

Figure 5 S/H circuit

3.3 DAC circuit

Most DAC of SAR ADC use capacitive DAC to provide intrinsic tracking / holding function. The capacitive DAC generates analog output voltage according to charge redistribution principle. The capacitive DAC consists of N capacitor arrays with binary weight values and a "virtual LSB" capacitor.

3.4 Conversion steps

The number of conversion steps is equal to the resolution of ADC. For example, 10-bit ADC has 10 conversion steps, and each ADC clock generates a data bit. The following steps take 10-bit ADC as an example. The sampling and holding state can be understood by referring to the above S/H equivalent circuit. The following focuses on the status of quantization and coding.

3.4.1 Quantization and coding status

In this status, each ADCCLK executes one step, and ADC outputs one digit after each step is completed. The binary method is used for successive approximation to the accuracy (bits) of ADC. The whole conversion process is shown in the figure below.

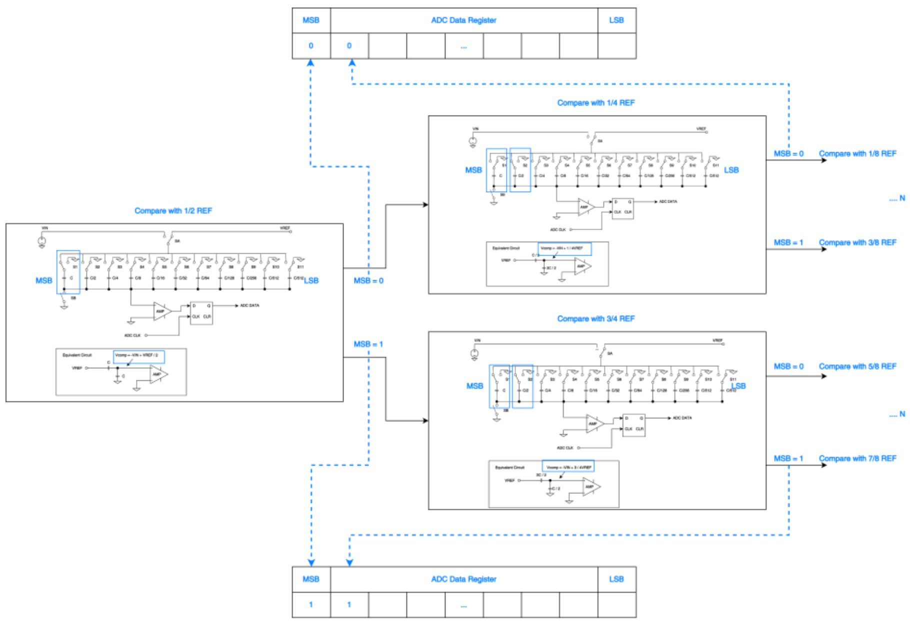

Figure 6 Quantization and Coding Status

3.4.1.1 Conversion example

For example, if 2.5V is input into the SAR ADC with reference voltage of 3.3V, the conversion process is as follows.

In the first approximation step, MSB is set to 1 first. DAC compares 1/2 REF with VIN. If VIN > 1/2 REF, hold MSB = 1 (on the contrary, MSB = 0). Wait for the next ADCCLK and execute the next step.

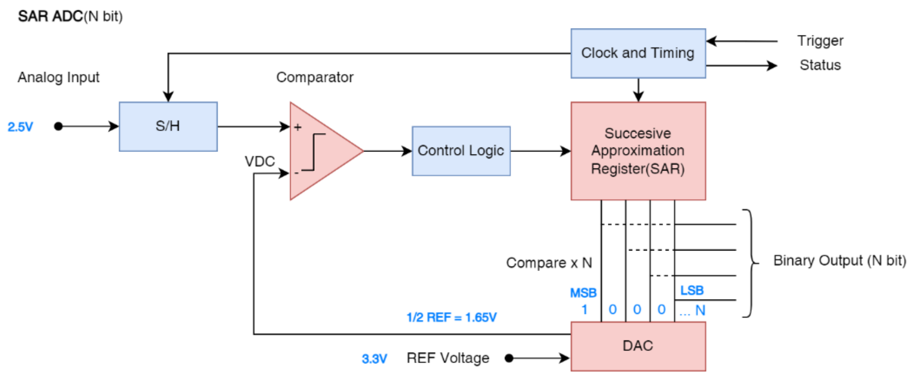

Figure 7 ADC Conversion Approximation Step 1

In the second approximation step, MSB moves back by 1 bit, and then compares 3/4 REF with VIN. If VIN > 3/4 REF, hold MSB = 1 (on the contrary, MSB = 0). Wait for the next ADCCLK, and execute the next step. When all bits are confirmed, output the coded value.

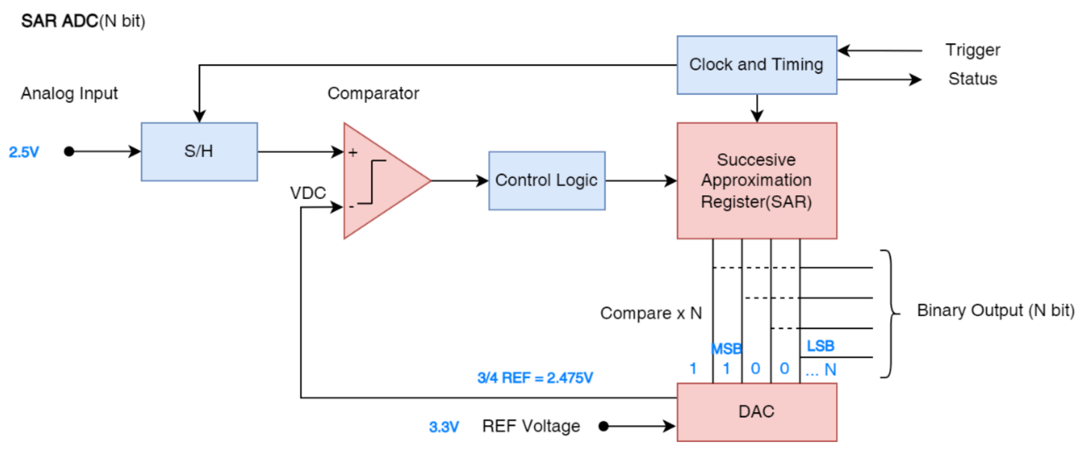

Figure 8 ADC Conversion Approximation Step 2

3.5 Conversion time

ADC conversion time in APM32F4xx = sampling cycle + conversion cycle.

3.5.1 Sampling cycle

It is determined by the **sampling cycle setting**. It should be noted that this value needs to match the input impedance of the external circuit. So that it can be ensured that the sampling and holding capacitors have enough time to charge at the adoption stage.

3.5.2 Conversion cycle

This value depends on the conversion accuracy of the ADC. The SAR ADC of APM32F4xx is 12 bits by default and can be configured as 12, 10, 8, and 6 bits.

Table 1 Relationship between ADC Accuracy and Conversion Cycle

No.	ADC accuracy	Conversion cycle
1	12 bits	12 x ADCCLK
2	10 bits	10 x ADCCLK
3	8 bits	8 x ADCCLK
4	6 bits	6 x ADCCLK

3.6 Converted value

ADC converted value = (VIN x 2^n) / VREF, wherein n is the resolution of ADC. Take the above 12-bit ADC as an example, then

ADC converted value = (VIN x 4096) / VREF.

4 Configuration and Application of ADC

4.1 Hardware design

4.1.1 Input channel

The MINI Board has connected the channels of some ADC through the pin header, and relevant IO can be used according to design requirements. ADC sampling is prone to external interference. When using it, pay attention to the anti-interference design between pins and avoid the sharing of ADC pins and other functional circuits.

4.1.2 Voltage input range

The voltage input range of ADC is V_{REF-} ~ V_{REF+}, VSSA and V_{REF-} on the MINI board are connected to GND, while VDDA and V_{REF+} are connected to VDD, so the voltage input range of ADC on the MINI board is 0V ~ 3.3V.

4.2 Software design

The software design only explains key configurations, and some queries and logic codes are not designed. For details, you can directly refer to supporting routines of this application description.

4.2.1 ADC initialization structure

ADC_Config_T structure is defined in the document of APM32F4xx_adc.h. The specific definition is as follows:

```
/**
 * @brief ADC configuration Mode
 */
typedef struct
{
    ADC_RESOLUTION_T        resolution;
    uint8_t                 scanConvMode;
    uint8_t                 continuousConvMode;
    ADC_EXT_TRIG_EDGE_T     extTrigEdge;
    ADC_EXT_TRIG_CONV_T     extTrigConv;
    ADC_DATA_ALIGN_T        dataAlign;
    uint8_t                 nbrOfChannel;
} ADC_Config_T;
```

Meaning of parameters in the structure:

resolution: used to configure the resolution of ADC. The resolution of ADC can be configured to 12 bits, 10 bits, 8 bits and 6 bits. As described in the SAR ADC conversion principle of this application description, the higher the resolution of ADC is, the longer the relative conversion time is;

scanConvMode: Used to configure whether to enable the scan mode. Generally, it is configured as DISABLE when single-channel A/D conversion is applied, and configured as ENABLE when multi-channel A/D conversion is applied;

continuousConvMode: Used to configure single conversion or enable automatic continuous conversion mode;

extTrigEdge: Used to configure the polarity of external trigger. If external trigger is not enabled, it can be configured as ADC_EXT_TRIG_EDGE_NONE;

extTrigConv: Used to configure external trigger source;

dataAlign: Used to configure the data alignment method of ADC conversion results. Generally, we configure it as the right-aligned mode according to the custom;

nbrOfChannel: Used to configure the number of A/D conversion channels.

4.2.2 ADC general initialization structure

```
/**
  * @brief ADC Common Init structure definition
  */
typedef struct
{
    ADC_MODE_T              mode;
    ADC_PRESCALER_T         prescaler;
    ADC_ACCESS_MODE_T       accessMode;
    ADC_TWO_SAMPLING_T      twoSampling;
} ADC_CommonConfig_T;
```

ADC_ CommonConfig _T structure is defined in the document of APM32F4xx_adc.h. The specific definition is as follows:

Meaning of parameters in the structure:

mode: Used to configure the working mode of ADC. There are three configuration items, namely, independent mode, dual mode and triple mode;

prescaler: Used to configure the frequency division coefficient of ADC clock, which is provided by PCLK2. PCLK2 divided by prescaler is the ADC clock;

accessMode: Used to configure DMA mode;

twoSampling: Used to configure the delay between two sampling stages.

4.2.3 Design of single-channel conversion software

Take "ADC_ContinuousConversion" as an example.

4.2.3.1 Configure ADC

After the GPIO clock is turned on, configure GPIO to analog input mode.

```
/*!
 * @brief          ADC Init
 *
 * @param           None
 *
 * @retval         None
 */
void ADC_Init(void)
{
    GPIO_Config_T     gpioConfig;
    ADC_Config_T      adcConfig;

    /** Enable GPIOA clock */
    RCM_EnableAHB1PeriphClock(RCM_AHB1_PERIPH_GPIOA);

    /** ADC channel 0 configuration */
    GPIO_ConfigStructInit(&gpioConfig);
    gpioConfig.mode        = GPIO_MODE_AN;
    gpioConfig.pupd        = GPIO_PUPD_NOPULL;
    gpioConfig.pin         = GPIO_PIN_0;
    GPIO_Config(GPIOA, &gpioConfig);
```

Configure ADC working mode and turn on the continuous conversion mode.

```
/** Enable ADC clock */
    RCM_EnableAPB2PeriphClock(RCM_APB2_PERIPH_ADC1);

    /** ADC configuration */
    ADC_Reset();
    ADC_ConfigStructInit(&adcConfig);
    adcConfig.resolution              = ADC_RESOLUTION_12BIT;
    adcConfig.continuousConvMode      = ENABLE;
    adcConfig.dataAlign               = ADC_DATA_ALIGN_RIGHT;
    adcConfig.extTrigEdge             = ADC_EXT_TRIG_EDGE_NONE;
    adcConfig.scanConvMode            = DISABLE;
    ADC_Config(ADC1, &adcConfig);
```

Configure to turn on ADC interrupt, enable ADC and trigger the conversion by software.

```
    /** ADC channel 0 Convert configuration */
    ADC_ConfigRegularChannel(ADC1, ADC_CHANNEL_0, 1,
    ADC_SAMPLETIME_112CYCLES);

    /** Enable complete conversion interupt */
    ADC_EnableInterrupt(ADC1, ADC_INT_EOC);

    /** NVIC configuration */
    NVIC_EnableIRQRequest(ADC_IRQn, 1, 1);

    /** Enable ADC */
    ADC_Enable(ADC1);

    /** ADC start conversion */
    ADC_SoftwareStartConv(ADC1);
}
```

4.2.3.2 ADC interrupt service function

After the completion of the conversion is detected, call back the interrupt service function. Read the ADC converted value at this time, and then convert it into the corresponding voltage value, which is affected by the quantization error and the anti-interference ability of the hardware. The unit of the voltage converted here is mV.

```
/*!
 * @brief              ADC interrupt service routine
 *
 * @param              None
 *
 * @retval             None
 */
void ADC_Isr(void)
{
    uint16_t adcData = 0;
    uint16_t voltage = 0;

    if (ADC_ReadStatusFlag(ADC1, ADC_FLAG_EOC))
    {
        ADC_ClearStatusFlag(ADC1, ADC_FLAG_EOC);
        adcData = ADC_ReadConversionValue(ADC1);
        voltage = (adcData * 3300) / 4095;
        printf("\r\n voltage : %d mV\r\n", voltage);
    }
}
```

4.2.4 Design of multi-channel scanning software

Take "ADC_MultiChannelScan" as an example.

4.2.4.1 Define common information

Here, the number of sampling channels of this sample is defined as 3, and the array adcData[ADC_CH_SIZE] that will be used for DMA to store the scanning data of each channel of ADC is defined. In addition, the address of ADC rule data register is macro-defined as ADC_DR_ADDR. All the above information will be used in subsequent configuration and application.

```
        /** save adc data*/
        #define ADC_CH_SIZE              3
        #define ADC_DR_ADDR              ((uint32_t)ADC1_BASE + 0x4C)


        uint16_t adcData[ADC_CH_SIZE];
```

4.2.4.2 Configure DMA

Set the rule register address of ADC to the register address accessed by DMA, set the register to increase and specify the buffer size as 3. Finally turn on the cycle mode.

Different channels and data streams of DMA specify the peripherals to which they belong. You should pay attention to it when using.

```
/*!
 * @brief        DMA Init
 *
 * @param        None
 *
 * @retval       None
 */
void DMA_Init(void)
{
    DMA_Config_T dmaConfig;

    RCM_EnableAHB1PeriphClock(RCM_AHB1_PERIPH_DMA2);

    dmaConfig.peripheralBaseAddr = ADC_DR_ADDR;
    dmaConfig.memoryBaseAddr = (uint32_t)&adcData;
    dmaConfig.dir = DMA_DIR_PERIPHERALTOMEMORY;
    dmaConfig.bufferSize = ADC_CH_SIZE;
    dmaConfig.peripheralInc = DMA_PERIPHERAL_INC_DISABLE;
    dmaConfig.memoryInc = DMA_MEMORY_INC_ENABLE;
    dmaConfig.peripheralDataSize = DMA_PERIPHERAL_DATA_SIZE_HALFWORD;
    dmaConfig.memoryDataSize = DMA_MEMORY_DATA_SIZE_HALFWORD;
    dmaConfig.loopMode = DMA_MODE_CIRCULAR;
    dmaConfig.priority = DMA_PRIORITY_HIGH;
    dmaConfig.fifoMode = DMA_FIFOMODE_DISABLE;
    dmaConfig.fifoThreshold = DMA_FIFOTHRESHOLD_HALFFULL;
    dmaConfig.memoryBurst = DMA_MEMORYBURST_SINGLE;
    dmaConfig.peripheralBurst = DMA_PERIPHERALBURST_SINGLE;
    dmaConfig.channel = DMA_CHANNEL_0;
    DMA_Config(DMA2_Stream0,&dmaConfig);

    DMA_Enable(DMA2_Stream0);
}
```

4.2.4.3 Configure ADC

The same as the configuration sequence of single-channel conversion, first turn on the corresponding clock of the three channels to be scanned and configure them to analog input mode.

```
/*!
 * @brief       ADC Init
 *
 * @param       None
 *
 * @retval      None
 */
void ADC_Init(void)
{
    GPIO_Config_T               gpioConfig;
    ADC_Config_T                adcConfig;
    ADC_CommonConfig_T          adcCommonConfig;

    /** Enable GPIOA clock */
    RCM_EnableAHB1PeriphClock(RCM_AHB1_PERIPH_GPIOA);

    /** ADC channel 0 configuration */
    GPIO_ConfigStructInit(&gpioConfig);
    gpioConfig.mode     = GPIO_MODE_AN;
    gpioConfig.pupd     = GPIO_PUPD_NOPULL;
    gpioConfig.pin      = GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2;
    GPIO_Config(GPIOA, &gpioConfig);
```

Next is the general configuration of ADC.

```
/** Enable ADC clock */
RCM_EnableAPB2PeriphClock(RCM_APB2_PERIPH_ADC1);

/** ADC configuration */
ADC_Reset();

adcCommonConfig.mode               = ADC_MODE_INDEPENDENT;
adcCommonConfig.prescaler          = ADC_PRESCALER_DIV2;
adcCommonConfig.accessMode         = ADC_ACCESS_MODE_DISABLED;
adcCommonConfig.twoSampling        = ADC_TWO_SAMPLING_20CYCLES;
ADC_CommonConfig(&adcCommonConfig);
```

Next is the configuration of ADC working mode. It is configured as continuous scan mode, and the number of conversion channels is set to 3.

```
ADC_ConfigStructInit(&adcConfig);
adcConfig.resolution               = ADC_RESOLUTION_12BIT;
adcConfig.scanConvMode             = ENABLE;
adcConfig.continuousConvMode       = ENABLE;
adcConfig.dataAlign                = ADC_DATA_ALIGN_RIGHT;
adcConfig.extTrigEdge              = ADC_EXT_TRIG_EDGE_NONE;
adcConfig.extTrigConv              = ADC_EXT_TRIG_CONV_TMR1_CC1;
adcConfig.nbrOfChannel             = ADC_CH_SIZE;
ADC_Config(ADC1, &adcConfig);
```

Configure the conversion sequence and sampling cycle of each channel.

```
/** ADC channel Convert configuration */
    ADC_ConfigRegularChannel(ADC1, ADC_CHANNEL_0,
    ADC_SAMPLETIME_480CYCLES);
    ADC_ConfigRegularChannel(ADC1, ADC_CHANNEL_1, 2,
                                     ADC_SAMPLETIME_480CYCLES);
    ADC_ConfigRegularChannel(ADC1, ADC_CHANNEL_2, 3,
    ADC_SAMPLETIME_480CYCLES);
```

Finally, turn on DMA, enable ADC and trigger the conversion. Then the configuration of multi-channel scanning is completed. Then directly poll the stored array of ADC converted value to obtain the scanning value of each channel.

```
    /** Config DMA*/
    DMA_Init();

    /** Enable ADC DMA Request*/
    ADC_EnableDMARequest(ADC1);

    /** Enable ADC DMA*/
    ADC_EnableDMA(ADC1);

    /** Enable ADC */
    ADC_Enable(ADC1);

    /** ADC start conversion */
    ADC_SoftwareStartConv(ADC1);
}
```

4.3 Hardware method of improving sampling accuracy

1. Ensure that the reference voltage noise is minimized;

2. Minimize the crosstalk interference of IO pins;

3. Add masks to reduce EMI;

4. Arrange and lay analog and digital signals on the PCB separately.

4.4 Software method of improving sampling accuracy

4.4.1 Sampling average

When the anti-interference ability of the hardware is insufficient, we can sacrifice the sampling rate and use software method to filter, so as to obtain more stable sampling values.

4.4.2 Digital signal filtering

Software can be filtered through digital low-pass, high-pass and other filters. For example, we know that the noise in the measured signal comes from the 50 Hz power supply line, and through appropriate digital filtering, only the 50 Hz frequency can be suppressed and the data signal without this noise can be transmitted.

4.4.3 ADC software calibration

If the sampling value is relatively fixed, but still has a large difference from the target value, linear fitting (linear calibration curve) can also be used to make the sampling value closer to the target value.

4.4.3.1 Sample

First, take samples of enough points based on the standard source table, as shown in the following table (the more the points, the more accurate the fitting).

Table 2 Sampling Table

No.	Sampling value (mV)	Target value (mV)
1	96.7	100
2	194.5	200
3	498.8	500
4	796	800
5	1495	1500

4.4.3.2 Fitting

Do linear fitting (linear, polynomial or exponential) in mathematical tools (Matlab or Excel, etc.) to obtain the calibration formula and the value of the correlation coefficient R.

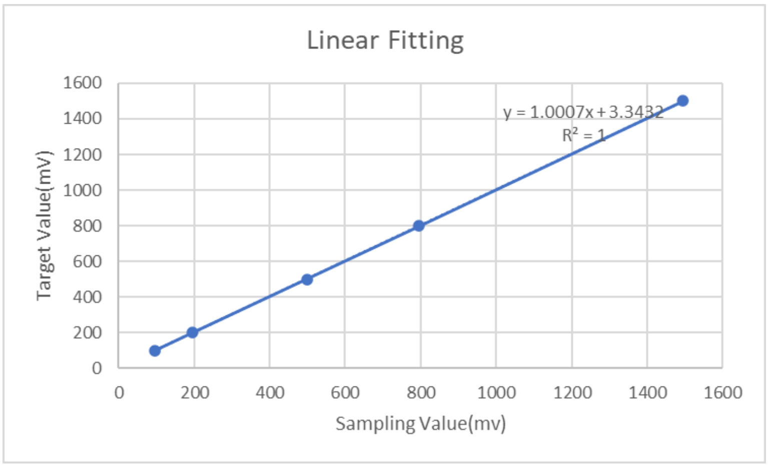

Figure 9 Fitting

4.4.3.3 Calibration

Calibrate the sampling value with the calibration formula obtained in the fitting step.

Table 3 Sampling Table

No.	Sampling value (mV)	Target value (mV)	Calibration Value (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 Version History

Table 4 Document Version History

Date	Version	Change History
May 31, 2022	1.0	New

Statement

standards, safety or other reliability requirements. If loses are caused to users due to the user’s failure to fully verify and test Geehy products, Geehy will not bear any responsibility.

5. Legality

USERS SHALL ABIDE BY ALL APPLICABLE LOCAL LAWS AND REGULATIONS WHEN USING THIS DOCUMENT AND THE MATCHING GEEHY PRODUCTS. USERS SHALL UNDERSTAND THAT THE PRODUCTS MAY BE RESTRICTED BY THE EXPORT, RE-EXPORT OR OTHER LAWS OF THE COUNTIRIES OF THE PRODUCTS SUPPLIERS, GEEHY, GEEHY DISTRIBUTORS AND USERS. USERS (ON BEHALF OR ITSELF, SUBSIDIARIES AND AFFILIATED ENTERPRISES) SHALL AGREE AND PROMISE TO ABIDE BY ALL APPLICABLE LAWS AND REGULATIONS ON THE EXPORT AND RE-EXPORT OF GEEHY PRODUCTS AND/OR TECHNOLOGIES AND DIRECT PRODUCTS.

6. Disclaimer of Warranty

THIS DOCUMENT IS PROVIDED BY GEEHY "AS IS" AND THERE IS NO WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, TO THE EXTENT PERMITTED BY APPLICABLE LAW.

GEEHY WILL BEAR NO RESPONSIBILITY FOR ANY DISPUTES ARISING FROM THE SUBSEQUENT DESIGN OR USE BY USERS.

7. Limitation of Liability

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL GEEHY OR ANY OTHER PARTY WHO PROVIDE THE DOCUMENT "AS IS", BE LIABLE FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, DIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE DOCUMENT (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY USERS OR THIRD PARTIES).

8. Scope of Application

The information in this document replaces the information provided in all previous versions of the document.