ADC acquisition

1. Learning objectives

1. Understand the basic knowledge of ADC.
2. Learn the basic use of ADC and display the collected values on the serial assistant.

ADC principle

ADC (analog to digital converter) is an analog-to-digital converter used to convert analog signals (such as voltage) into digital signals. Analog signals change continuously, while digital signals are discrete binary numbers. ADC converts analog signals into digital data by sampling and quantizing them so that the processor or microcontroller can perform subsequent processing. According to its conversion principle, it is mainly divided into three types: successive approximation type, dual integration type, and voltage-frequency conversion type.

MSPM0G3507 uses a successive approximation (SAR) ADC, which is a common ADC working principle. The basic idea is to gradually approximate the digital representation of the input signal by comparing the magnitude relationship between the analog signal and the reference voltage. In a successive approximation ADC, the input signal and the reference voltage are compared through a differential amplifier to generate a differential voltage. This differential voltage is then input into a progressively approximated digital quantizer, which progressively compares it to a series of reference voltages. At each approximation stage, the quantizer compares the input signal to an intermediate voltage point and selects a higher or lower reference voltage as the reference for the next approximation stage based on the comparison result. This process continues until the quantizer finally approximates a digital output value.

ADC Basic Parameters

1. **Resolution**:

- Resolution indicates the accuracy of the ADC converter output, usually measured in bits, such as 8 bits, 10 bits, 12 bits, etc. The higher the resolution, the more discrete digital values the ADC can represent, providing higher accuracy.
- **MSPM0G3507** supports 8-bit, 10-bit, and 12-bit resolutions, and users can select the appropriate resolution for configuration as needed.

2. **Sampling Rate**:

- Sampling rate (also called conversion rate) indicates the rate at which the ADC samples the analog input signal, usually expressed in samples per second (SPS). It indicates how many times the ADC can perform analog-to-digital conversions per second.
- The sampling rate of **MSPM0G3507** is 4Msps (4 million samples per second), which is suitable for high-frequency signal acquisition and real-time data processing.

3. **Voltage reference**:

- The voltage reference of the ADC is a reference voltage, which is used to compare with the analog input signal and ultimately realize the conversion of analog signals to digital signals. The accuracy and stability of the voltage reference are crucial to the conversion accuracy of the ADC.
- MSPM0G3507

supports three voltage reference configurations:

- Internal configurable reference voltage: dedicated ADC reference voltage (VREF) of 1.4V and 2.5V.
- MCU power supply voltage (VDD) is used as the reference voltage.
- External reference voltage is provided through VREF+ and VREF- pins. If the voltage reference is not configured, the MCU power supply voltage (VDD) is used as the reference voltage by default.

4. **Sampling range**:

- The sampling range indicates the voltage range of the analog input signal that the ADC can collect, which is usually closely related to the setting of the reference voltage. The range is as follows: VREF- ≤ ADC ≤ VREF+
- **VREF-**: The negative end of the reference voltage, usually 0V.
- **VREF+**: The positive end of the reference voltage, determined by the software configuration.

These parameters together determine the performance of the ADC, including its accuracy, response speed and input voltage range.

2. Hardware Construction

MSPM0G3507 uses a successive approximation 12-bit ADC, which has 17 multiplexed channels for conversion. The 17 external channels all correspond to a pin of the microcontroller. This pin is not fixed. For details, please refer to the pin diagram or data sheet.

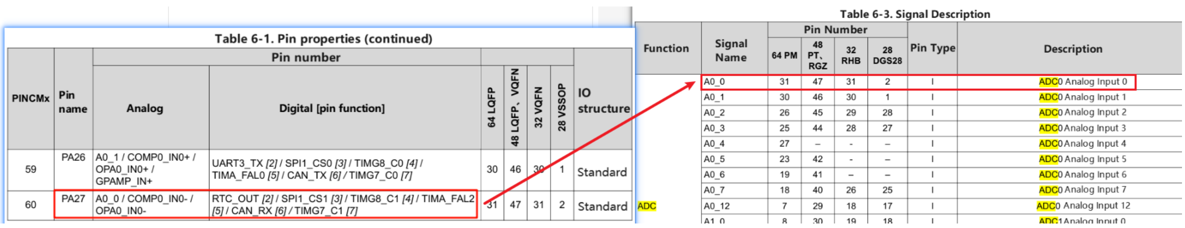


Table 6-1. Pin properties (continued)

PINCMx	Pin name	Analog	Digital [pin function]	64 LQFP	48 LQFP、VQFN	32 VQFN	28 VSSOP	IO structure
59	PA26	A0_1 / COMP0_IN0+ / OPA0_IN0+ / GPAMP_IN+	UART3_TX [2] / SPI1_CS0 [3] / TIMG8_C0 [4] / TIMA_FAL0 [5] / CAN_TX [6] / TIMG7_C0 [7]	30	46	30	1	Standard
60	PA27	A0_0 / COMP0_IN0- / OPA0_IN0-	RTC_OUT [2] / SPI1_CS1 [3] / TIMG8_C1 [4] / TIMA_FAL2 [5] / CAN_RX [6] / TIMG7_C1 [7]	31	47	31	2	Standard

Table 6-3. Signal Description

Function	Signal Name	Pin Number: 64 PM	48 PT、RGZ	32 RHB	28 DGS28	Pin Type	Description
ADC	A0_0	31	47	31	2	I	ADC0 Analog Input 0
	A0_1	30	46	30	1	I	ADC0 Analog Input 1
	A0_2	26	45	29	28	I	ADC0 Analog Input 2
	A0_3	25	44	28	27	I	ADC0 Analog Input 3
	A0_4	27	–	-	–	I	ADC0 Analog Input 4
	A0_5	23	42	-	–	I	ADC0 Analog Input 5
	A0_6	19	41	–	–	I	ADC0 Analog Input 6
	A0_7	18	40	26	25	I	ADC0 Analog Input 7
	A0_12	7	29	18	17	I	ADC0 Analog Input 12
	A1_0	8	30	19	18	I	ADC1 Analog Input 0

The A/D conversion of various channels can be configured into **single, sequence conversion** mode.

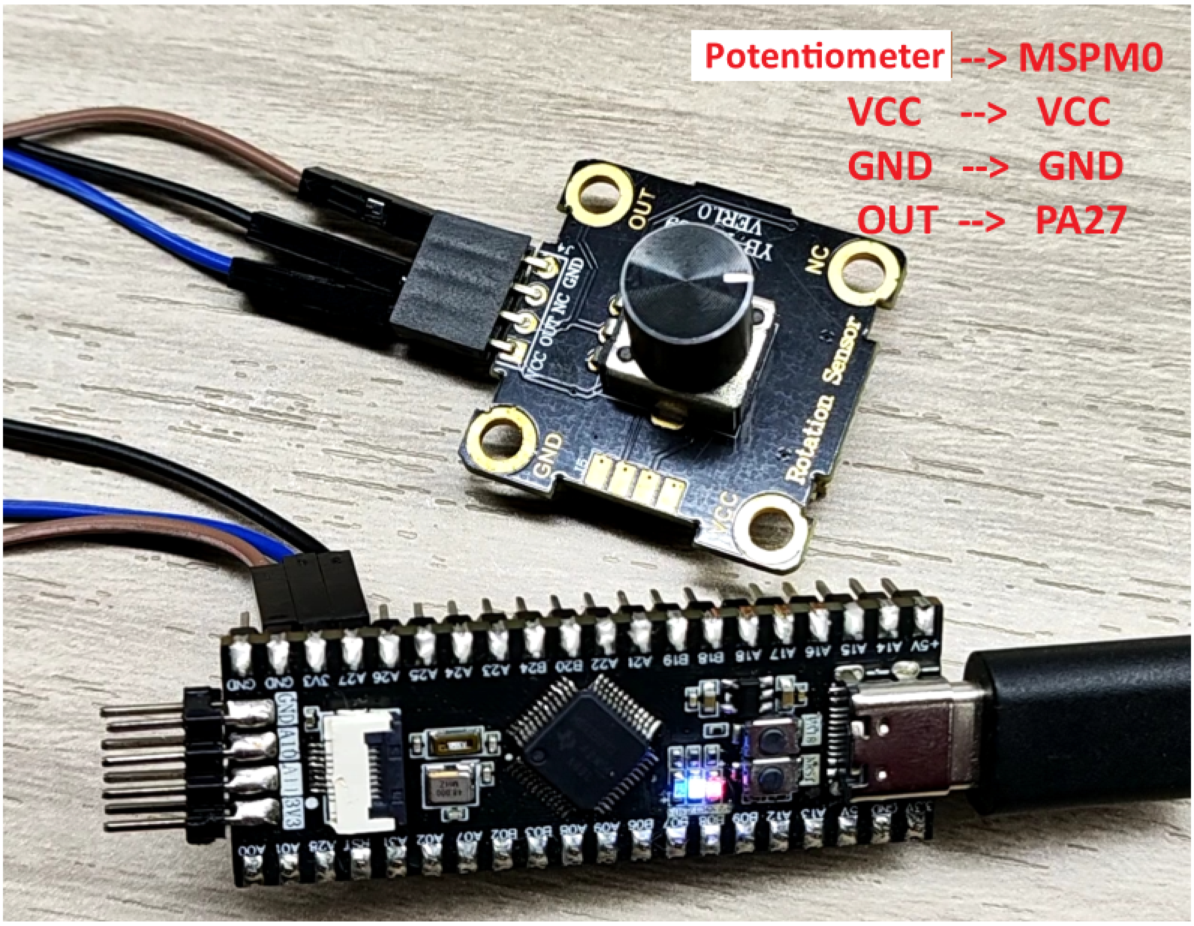
Single conversion mode: After each ADC conversion, the ADC will automatically stop and store the result in the ADC data register.

Repeated single conversion mode: When the ADC completes a conversion, it will automatically start another conversion and continue to convert until the continuous conversion is stopped by an external trigger or software trigger.

Multi-channel sequential single conversion mode: Used to convert multiple input channels in sequence. In this mode, the ADC will sample and convert multiple channels once according to the configured channel acquisition sequence.

Multi-channel sequential repeated conversion mode: Used to repeatedly convert multiple input channels in sequence. In this mode, the ADC will repeatedly sample and convert multiple channels according to the configured channel acquisition sequence.

In this case, the voltage of the PA27 pin is collected by the ADC. The potentiometer module and DuPont line used in the ADC acquisition experiment need to be purchased separately, and other input voltage methods can also be used.

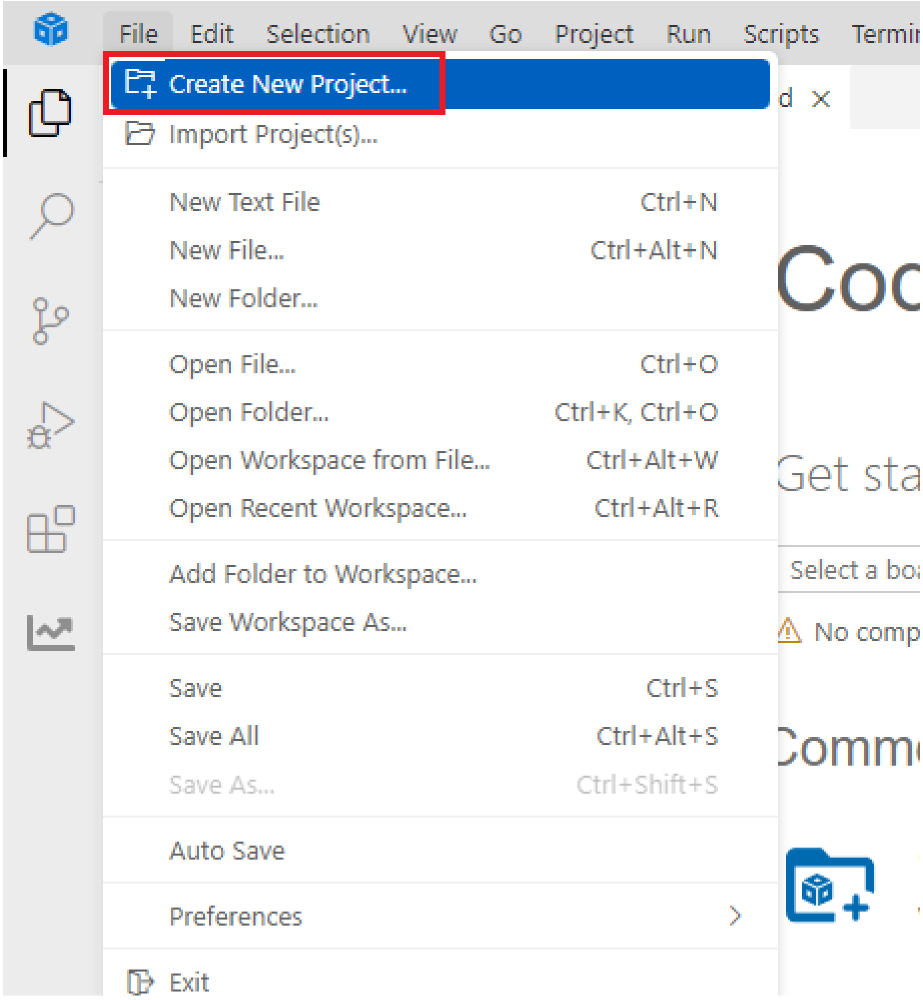


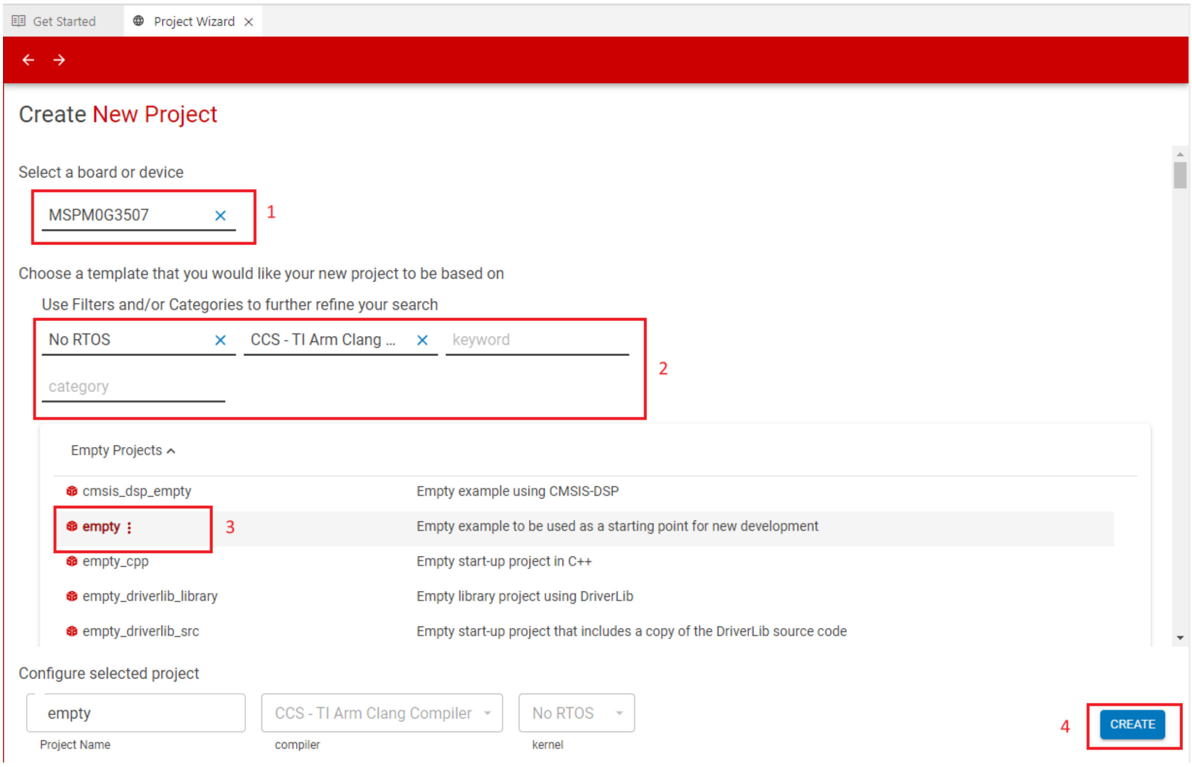
3. Experimental steps

This case will use the voltage of the PA27 pin as an experimental case. The voltage change is collected through ADC to print the voltage status on the serial port.

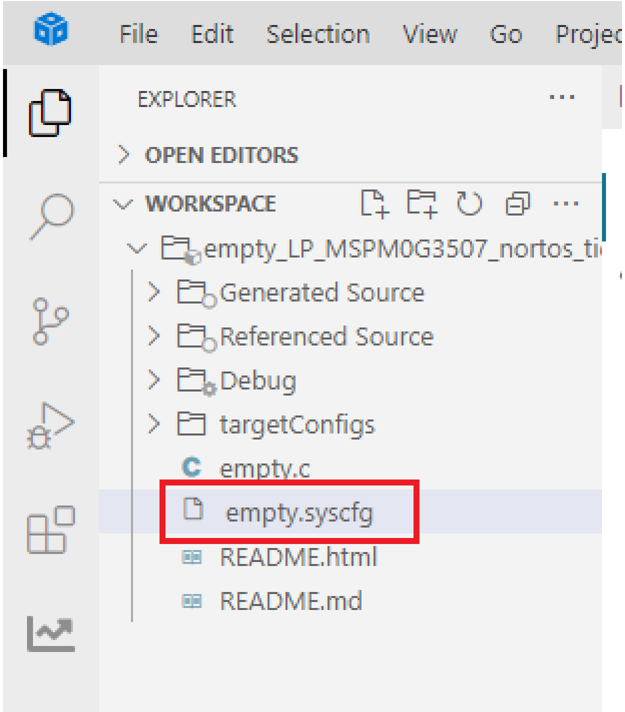
1. Open the SYSCONFIG configuration tool

Create a blank project empty in CCS.

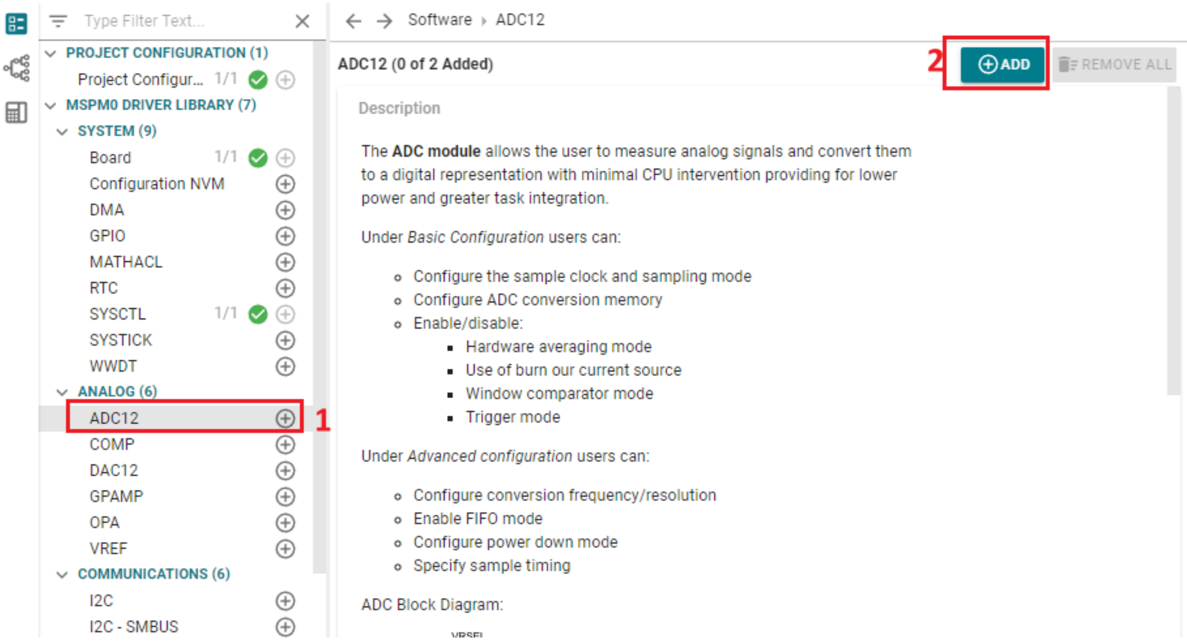




Find and open the empty.syscfg file in the left workspace of CCS.

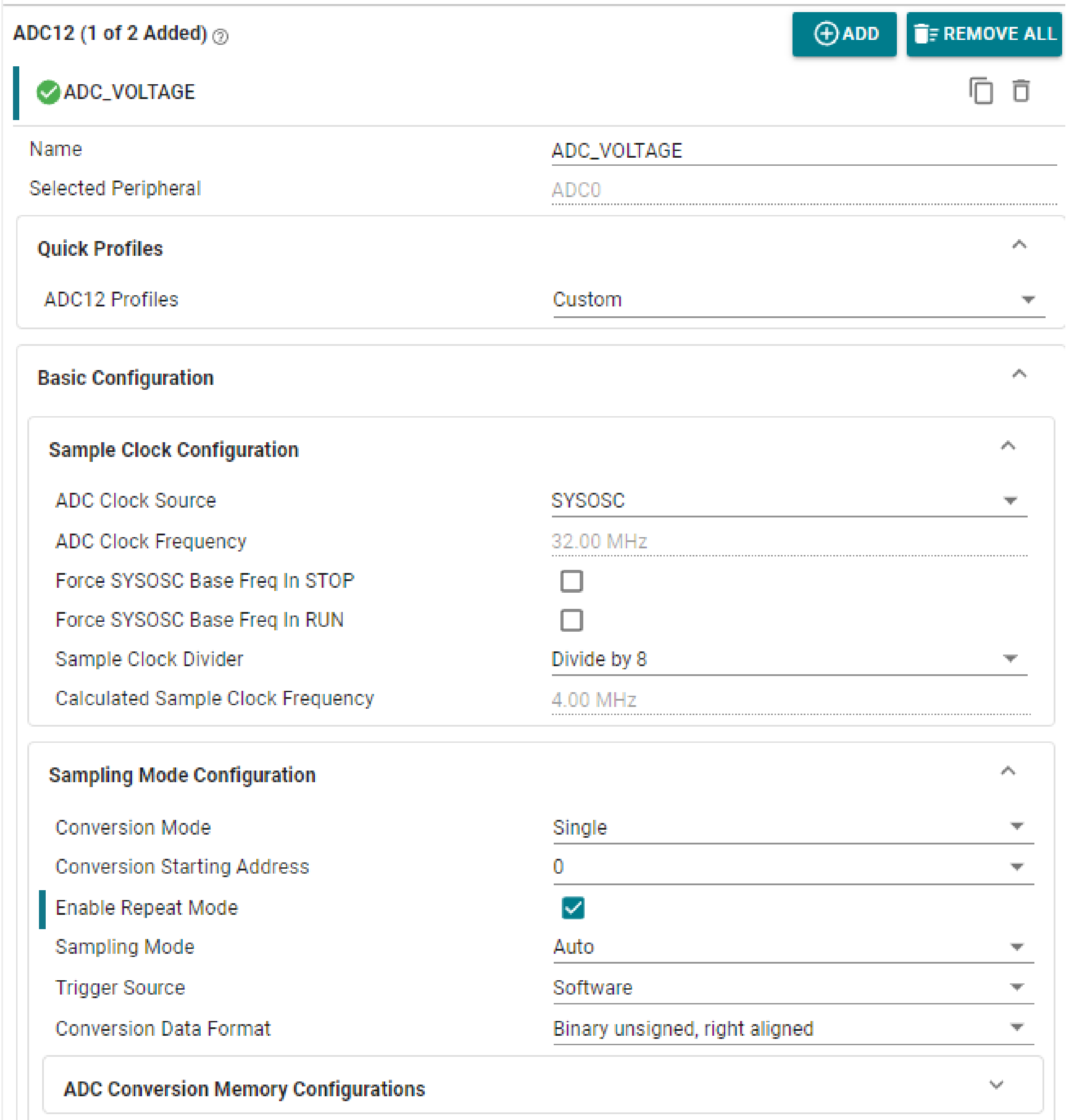


In SYSCONFIG, select MCU peripherals on the left, find the ADC12 option and click to enter. Click ADD in ADC12 to add an ADC peripheral.



2. ADC parameter configuration

The configuration selected here is to use an ADC frequency of 32Mhz, a sampling frequency of 4MHz, a single repetitive conversion mode, trigger the ADC to start sampling through software, and the data format is binary right-aligned.



Parameter Description:

ADC Clock Source: ADC clock source. Set to `SYSOSC(32MHz)`.

ADC Clock Frequency: Displays the current ADC clock frequency.

Sample Clock Divider: Sample divider. Configured as 8 dividers.

Calculated Sample clock Freguency: Displays the sampling frequency after frequency division.

Conversion Mode: Conversion mode. Configured as `Single`, single shot.

Conversion Starting Address: Conversion starting address. Configure to start sampling from the 0th (related to the storage register behind).

Enable Repeat Mode: Whether to enable repeated conversion. Check here to enable.

Sampling Mode: Sampling mode. Set to `Auto`, automatic.

Trigger Source: ADC trigger mode. Configured as `Software` software trigger mode.

Conversion Data Format: ADC data conversion format. Configured as `Binary unsigned, right aligned` binary format right alignment mode.

The ADC of MSPM0G3507 supports multiple data alignment methods to adapt to different application scenarios. Common data alignment methods include **left alignment** and **right alignment**.

- **Right alignment mode**: In right alignment mode, the ADC data is right-aligned to the lowest bit after the conversion, and the insufficient bits are filled with 0 in the high bit. Right-aligned mode allows for better dynamic range without loss of precision.

Rule group data

0	0	0	0	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

Injection group data

Sign	Sign	Sign	Sign	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

DAL=0

- **Left-aligned mode**: In left-aligned mode, the ADC data is left-aligned to the highest bit, and the insufficient bits are filled with 0 in the lower bit. Left-aligned mode can improve resolution, but will result in reduced dynamic range.

Rule group data

0	0	0	0	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

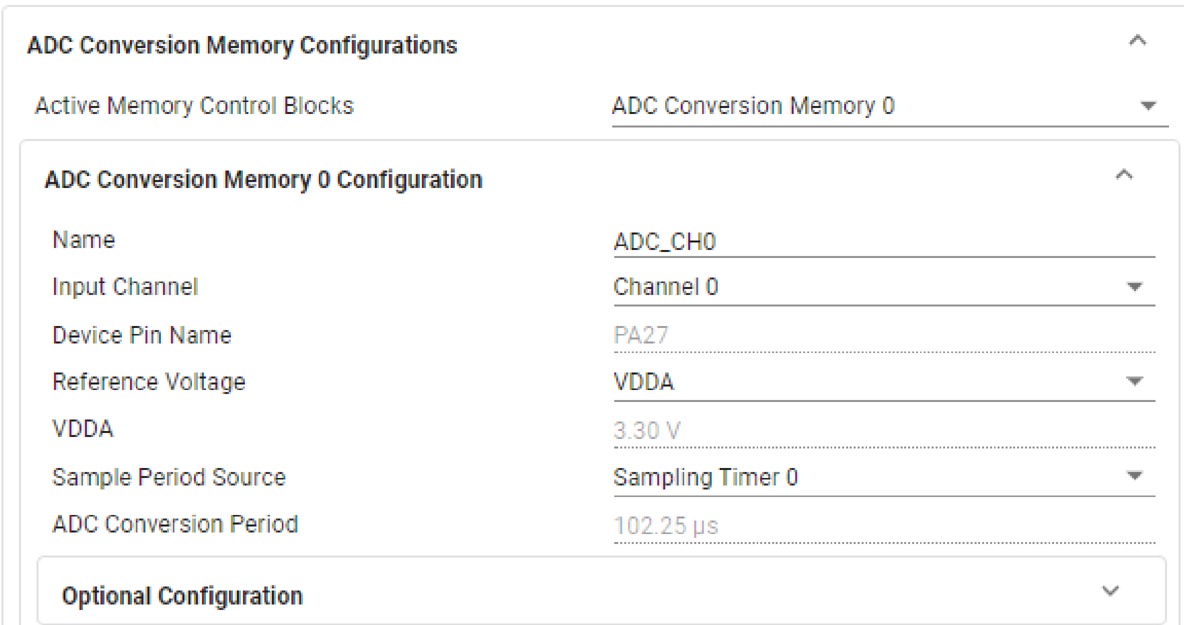
Injection group data

Sign	Sign	Sign	Sign	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

DAL=0

We configure it as right-aligned so that we can directly calculate the converted data.

3. ADC acquisition channel configuration



Parameter Description:

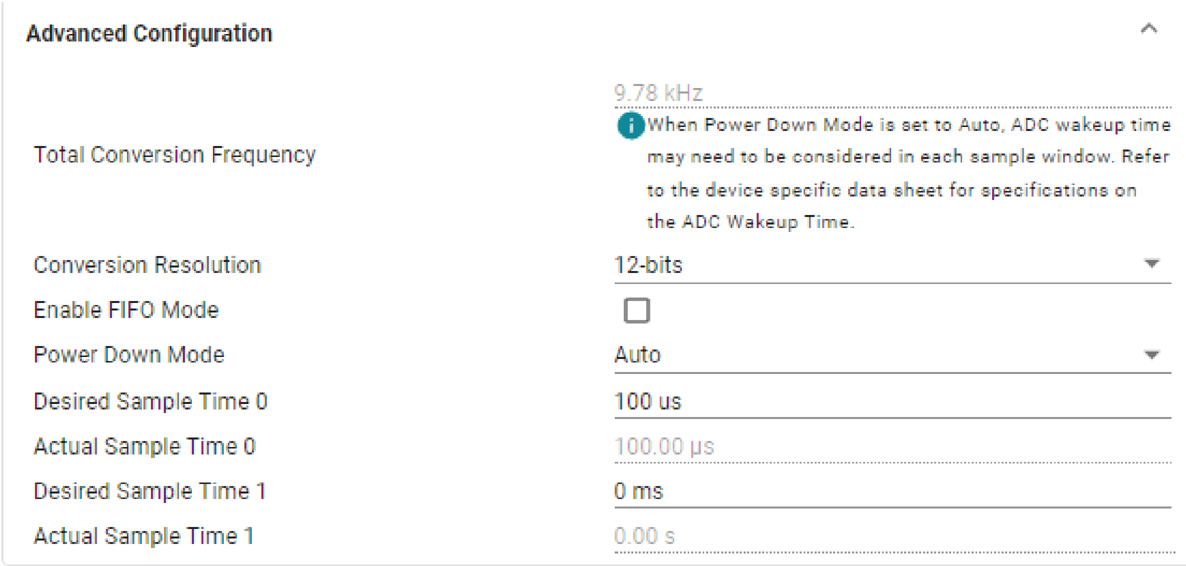
Active Memory Control Blocks: ADC data storage address. Store the ADC conversion result in register 0.

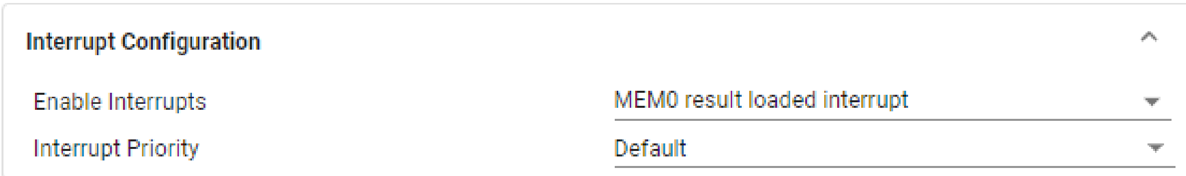
Input Channel: Input channel. Select channel 0 (each channel corresponds to a different pin).

Device Pin Name: The pin automatically selected according to the channel, the pin of channel 0 is PA27.

Reference Voltage: Voltage reference. Configured to use MCU power supply VDDA.

ADC Conversion Period: Displays the time it takes for the ADC to convert a single data.



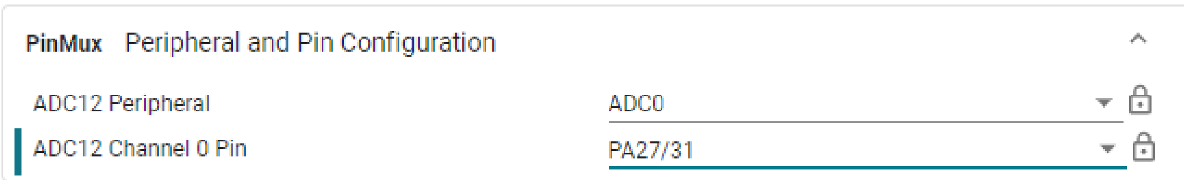


Parameter Description:

Conversion Resolution: The resolution is configured to 12-bits.

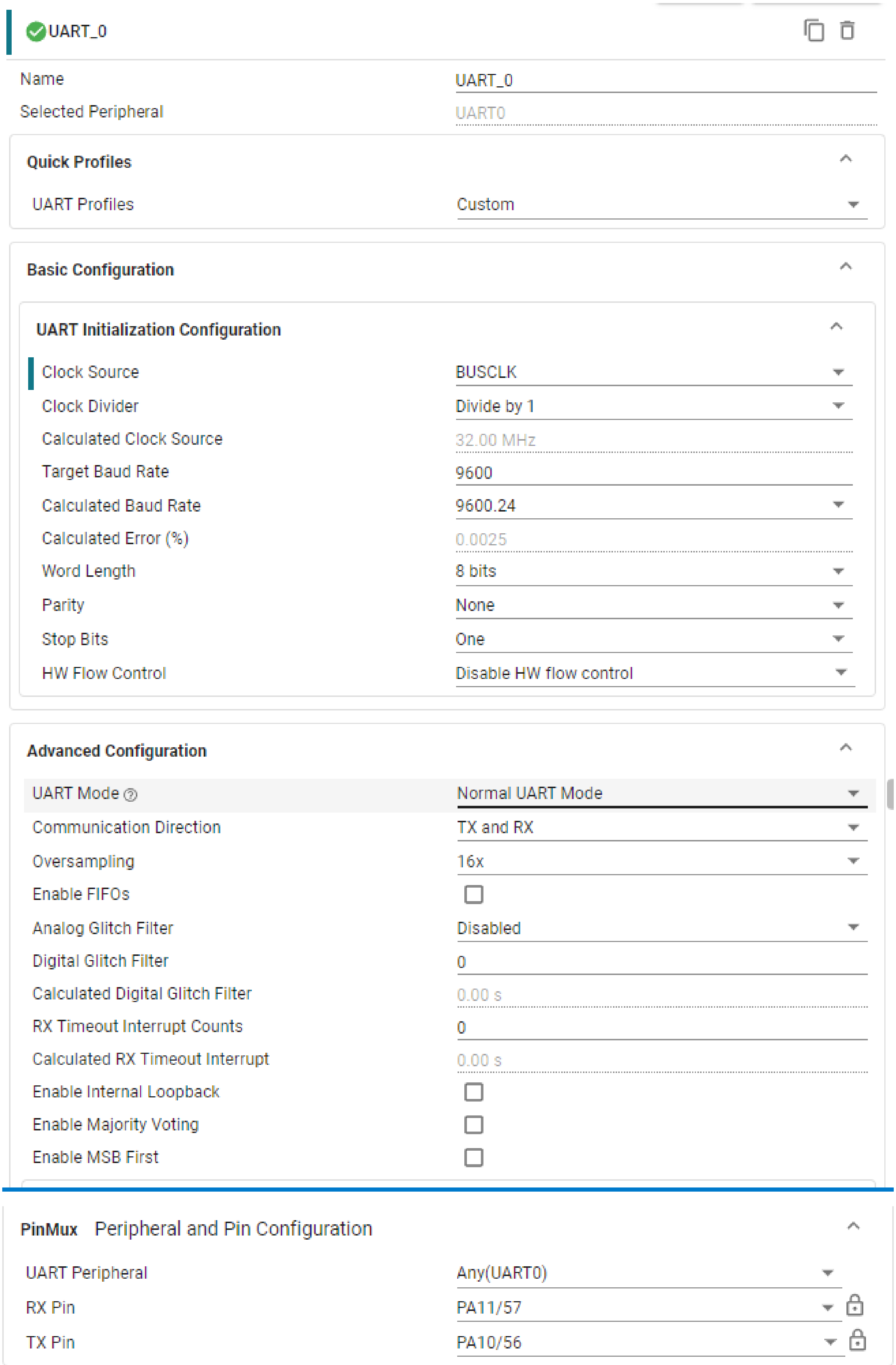
Desired Sample Time 0: The sampling time is set to 10ms.

Enable interrupts: Enable the interrupt for ADC acquisition completion. When the ADC acquisition is completed, the interrupt is triggered and we process the data.



4. Serial port parameter configuration

This case uses the serial port output method to output the values collected by the ADC to the host computer (computer) through the serial port, so the serial port sending function also needs to be configured. The undisplayed part is the default option.



Use the shortcut key Ctrl+S to save the configuration in the .syscfg file.

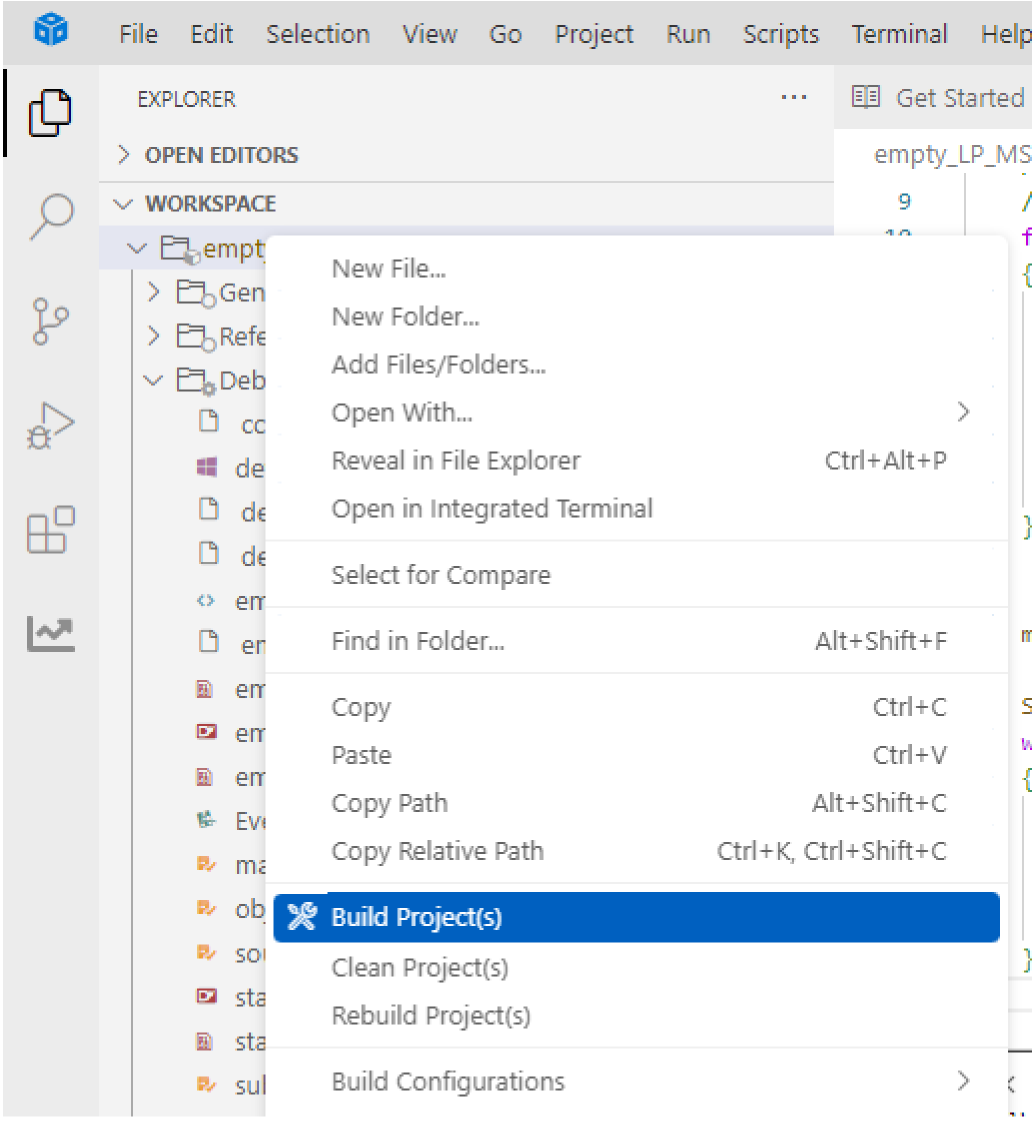
5. Write the program

In the empty.c file, write the following code

```
#include "ti_msp_dl_config.h"
#include "stdio.h"

volatile bool gCheckADC;        //ADC采集成功标志位 ADC acquisition success flag
unsigned int adc_getValue(void);//读取ADC的数据 Read ADC data

//串口发送字符串 Send string via serial port
void uart0_send_string(char* str)
{
    //当前字符串地址不在结尾 并且 字符串首地址不为空
    // The current string address is not at the end and the string's first
address is not empty
    while(*str!=0&&str!=0)
    {
        //当串口0忙的时候等待，不忙的时候再发送传进来的字符
        // Wait when serial port 0 is busy, and send the incoming characters
when it is not busy
        while( DL_UART_isBusy(UART_0_INST) == true );
        //发送字符串首地址中的字符，并且在发送完成之后首地址自增
        // Send the characters in the first address of the string, and increment
the first address after sending.
        DL_UART_Main_transmitData(UART_0_INST, *str++);
    }
}

int main(void)
{
    char output_buff[50] = {0};
    unsigned int adc_value = 0;
    float voltage_value = 0;

    SYSCFG_DL_init();

    //开启ADC中断 Enable ADC interrupt
    NVIC_EnableIRQ(ADC_VOLTAGE_INST_INT_IRQN);

    uart0_send_string("adc Demo start\r\n");
    while (1)
    {
        //获取ADC数据 Get ADC data
        adc_value = adc_getValue();
        sprintf(output_buff, "adc value:%d\r\n", adc_value);
        uart0_send_string(output_buff);

        //将ADC采集的数据换算为电压 Convert the data collected by ADC into voltage
        voltage_value = adc_value/4095.0*3.3;
        sprintf(output_buff, "voltage value:%.2f\r\n", voltage_value);
        uart0_send_string(output_buff);

        delay_cycles(32000000);
    }
}
```

```
//读取ADC的数据 Read ADC data
unsigned int adc_getValue(void)
{
        unsigned int gAdcResult = 0;

        //软件触发ADC开始转换 Software triggers ADC to start conversion
        DL_ADC12_startConversion(ADC_VOLTAGE_INST);
        //如果当前状态为正在转换中则等待转换结束 If the current state is in transition,
wait for the transition to end.
        while (false == gCheckADC) {
            __WFE();
        }
        //获取数据 Get data
        gAdcResult = DL_ADC12_getMemResult(ADC_VOLTAGE_INST,
ADC_VOLTAGE_ADCMEM_ADC_CH0);

        //清除标志位 Clear flag
        gCheckADC = false;

        return gAdcResult;
}

//ADC中断服务函数 ADC interrupt service function
void ADC_VOLTAGE_INST_IRQHandler(void)
{
    //查询并清除ADC中断 Query and clear ADC interrupt
    switch (DL_ADC12_getPendingInterrupt(ADC_VOLTAGE_INST))
    {
        //检查是否完成数据采集 Check whether data collection is completed
        case DL_ADC12_IIDX_MEM0_RESULT_LOADED:
            gCheckADC = true;//将标志位置1 Set the flag position to 1
            break;
        default:
            break;
    }
}
```

6. Compile



If the compilation is successful, you can download the program to the development board.

4. Program Analysis

- empty.c

```
//串口发送字符串 Send string via serial port
void uart0_send_string(char* str)
{
    //当前字符串地址不在结尾 并且 字符串首地址不为空
    // The current string address is not at the end and the string's first address is not empty
    while(*str!=0&&str!=0)
    {
        //当串口0忙的时候等待，不忙的时候再发送传进来的字符
        // Wait when serial port 0 is busy, and send the incoming characters when it is not busy
        while( DL_UART_isBusy(UART_0_INST) == true );
        //发送字符串首地址中的字符，并且在发送完成之后首地址自增
        // Send the characters in the first address of the string, and increment the first address after sending.
        DL_UART_Main_transmitData(UART_0_INST, *str++);
    }
}
```

The `uart0_send_string` function sends a string through the UART serial port. The `while (*str != 0)` loop iterates through each character until the end of the string (`*str == 0`). Before sending each character, `DL_UART_isBusy(UART_0_INST)` checks whether the UART is busy. Data can only be sent when the UART is idle. The current character is sent through `DL_UART_Main_transmitData(UART_0_INST, *str++)`, and `str` points to the next character.

```c
//读取ADC的数据 Read ADC data
unsigned int adc_getValue(void)
{
    unsigned int gAdcResult = 0;

    //软件触发ADC开始转换 Software triggers ADC to start conversion
    DL_ADC12_startConversion(ADC_VOLTAGE_INST);
    //如果当前状态为正在转换中则等待转换结束
    // If the current state is converting, wait for the conversion to end
    while (false == gCheckADC) {
        __WFE();
    }
    //获取数据 Get data
    gAdcResult = DL_ADC12_getMemResult(ADC_VOLTAGE_INST, ADC_VOLTAGE_ADCMEM_ADC_CH0);

    //清除标志位 Clear flag
    gCheckADC = false;

    return gAdcResult;
}
```

This code reads ADC (analog-to-digital converter) data. By triggering ADC conversion and waiting for the conversion to complete, finally get the ADC sampling result and return.

```c
//ADC中断服务函数 ADC interrupt service function
void ADC_VOLTAGE_INST_IRQHandler(void)
{
    //查询并清除ADC中断 Query and clear ADC interrupt
    switch (DL_ADC12_getPendingInterrupt(ADC_VOLTAGE_INST))
    {
        // 检查是否完成数据采集 Check whether data collection is completed
        case DL_ADC12_IIDX_MEM0_RESULT_LOADED:
            gCheckADC = true;//将标志位置1 Set the flag position to 1
            break;
        default:
            break;
    }
}
```

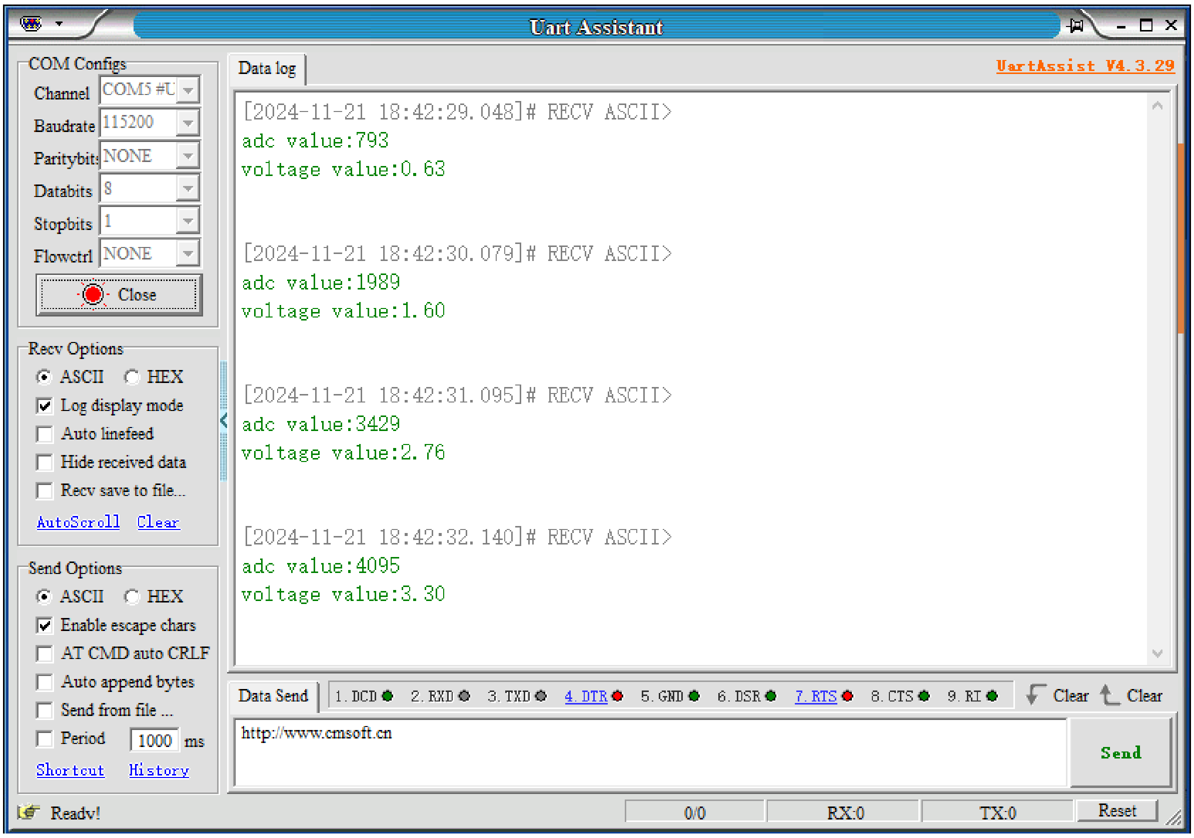
ADC's interrupt service function `ADC_VOLTAGE_INST_IRQHandler` is used to handle interrupt events and set the flag bit after ADC data acquisition is completed.

```c
int main(void)
{
    char output_buff[50] = {0};
    unsigned int adc_value = 0;
    float voltage_value = 0;

    SYSCFG_DL_init();

    //开启ADC中断 Enable ADC interrupt
    NVIC_EnableIRQ(ADC_VOLTAGE_INST_INT_IRQN);

    uart0_send_string("adc Demo start\r\n");
    while (1)
    {
        //获取ADC数据 Get ADC data
        adc_value = adc_getValue();
        sprintf(output_buff, "adc value:%d\r\n", adc_value);
        uart0_send_string(output_buff);

        //将ADC采集的数据换算为电压 Convert the data collected by ADC into voltage
        voltage_value = adc_value/4095.0*3.3;
        sprintf(output_buff, "voltage value:%.2f\r\n", voltage_value);
        uart0_send_string(output_buff);

        delay_cycles(32000000);
    }
}
```

This code implements reading ADC (Analog to Digital Converter) data, calculating voltage value, and outputting the result through UART serial port. Initialize system configuration and ADC interrupt. Enter the main loop, continuously read ADC data and calculate voltage. Send ADC value and calculated voltage to the terminal through UART serial port.

5. Experimental phenomenon

After the program is downloaded, configure the serial port assistant as shown below, adjust the potentiometer knob, and you can see the effect of real-time voltage change between 0-3.3v in the serial port assistant receiving window.


Uart Assistant
COM Configs
Channel COM5
Baudrate 115200
Paritybits NONE
Databits 8
Stopbits 1
Flowctrl NONE
Close
Data log
UartAssist V4.3.29
[2024-11-21 18:42:29.048]# RECV ASCII>
adc value:793
voltage value:0.63
[2024-11-21 18:42:30.079]# RECV ASCII>
adc value:1989
voltage value:1.60
[2024-11-21 18:42:31.095]# RECV ASCII>
adc value:3429
voltage value:2.76
[2024-11-21 18:42:32.140]# RECV ASCII>
adc value:4095
voltage value:3.30
Recv Options
ASCII HEX
Log display mode
Auto linefeed
Hide received data
Recv save to file...
AutoScroll Clear
Send Options
ASCII HEX
Enable escape chars
AT CMD auto CRLF
Auto append bytes
Send from file ...
Period 1000 ms
Shortcut History
Data Send
1. DCD 2. RXD 3. TXD 4. DTR 5. GND 6. DSR 7. RTS 8. CTS 9. RI
Clear Clear
http://www.cmsoft.cn
Send
Ready!
0/0
RX:0
TX:0
Reset