Voltage Detection

I. Learning Objectives

1. Understand basic ADC knowledge.
2. Learn basic ADC usage, displaying collected values on the serial debugging 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 through sampling and quantization, so that processors or microcontrollers can perform subsequent processing. According to their conversion principles, they are mainly divided into three types: successive approximation type, dual-slope integration type, and voltage-frequency conversion type.

MSPM0G3507 uses a successive approximation (SAR) ADC, which is a common ADC working principle. Its 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 successive approximation ADCs, the input signal and reference voltage are compared through a differential amplifier to produce a differential voltage. Then, this differential voltage is input to a successive approximation digital quantizer, which gradually compares it with a series of reference voltages. At each approximation stage, the quantizer compares the input signal with 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 to a digital output value.

ADC Basic Parameters

1. **Resolution**:
   - Resolution represents the precision of the ADC converter output, usually measured in bits, such as 8-bit, 10-bit, 12-bit, etc. The higher the resolution, the more discrete digital values the ADC can represent, thereby providing higher precision.
2. **Sampling Rate**:
   - Sampling rate (also called conversion rate) represents the rate at which the ADC samples the analog input signal, usually expressed in samples per second (SPS). It indicates how many analog-to-digital conversions the ADC can perform per second.
   - **MSPM0G3507** has a sampling rate of 4Msps (4 million samples per second), suitable for high-frequency signal acquisition and real-time data processing.
3. **Voltage Reference**:
   - The ADC voltage reference is a reference voltage used to compare with the analog input signal to ultimately achieve analog-to-digital signal conversion. The accuracy and stability of the voltage reference are crucial to the ADC conversion precision.

- MSPM0G3507

  supports three voltage reference configurations:

  - Internal configurable reference voltage: 1.4V and 2.5V dedicated ADC reference voltage (VREF).
  - MCU power supply voltage (VDD) as reference voltage.
  - External reference voltage provided through VREF+ and VREF- pins. If no voltage reference is configured, the MCU's power supply voltage (VDD) is used as the reference voltage by default.

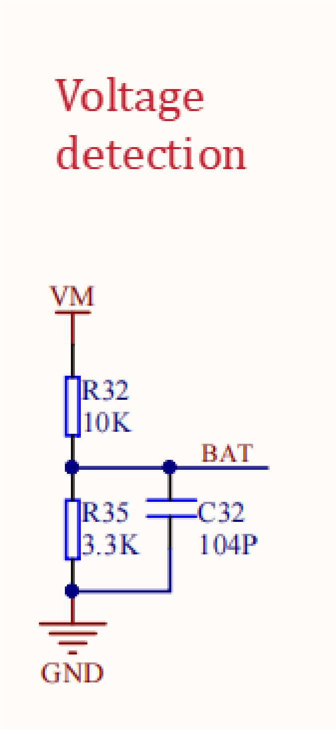
4. **Sampling Range**:
   - Sampling range represents the voltage range of analog input signals that the ADC can collect, usually closely related to the reference voltage setting. The range is as follows: VREF- ≤ ADC ≤ VREF+
     - **VREF-**: Set reference voltage negative terminal, usually 0V.
     - **VREF+**: Set reference voltage positive terminal, determined according to software configuration.

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

II. Hardware Setup

MSPM0G3507 uses a successive approximation 12-bit ADC, which has 17 multiplexed channels that can be converted. 17 external channels, all corresponding to certain pins of the microcontroller, these pins are not fixed, please refer to the pin diagram or data sheet for details.

Hardware schematic part is as follows



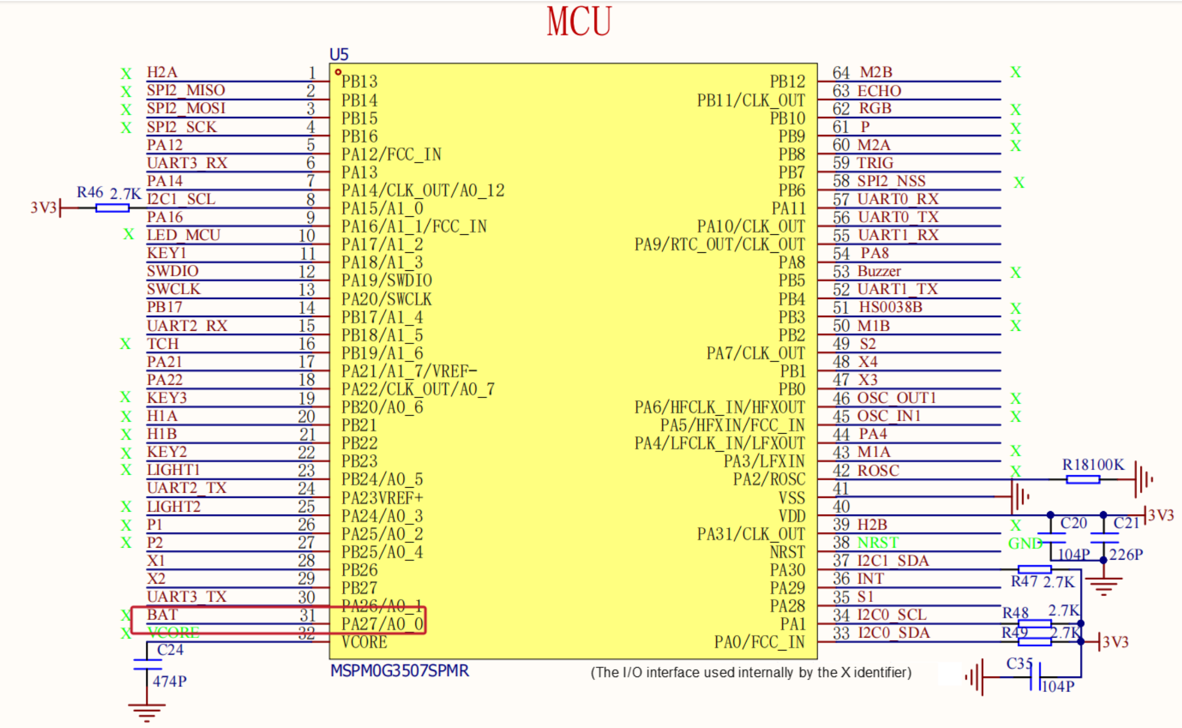


Table 6-1. Pin properties (continued)

PINCMx	Pin name	Pin number: Analog	Pin number: 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 in **single, sequence conversion** modes.

Single conversion mode: After each ADC conversion, the ADC automatically stops and stores the result in the ADC data register.

Repeated single conversion mode: When the ADC completes one conversion, it automatically starts another conversion, continuously performing conversions until external trigger or software trigger stops the continuous conversion.

Multi-channel sequential single conversion mode: Used to sequentially convert multiple input channels. In this mode, the ADC performs single sampling and conversion of multiple channels according to the configured channel acquisition sequence.

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

This case collects PA27 pin voltage through ADC. The potentiometer module and DuPont wires required for the ADC acquisition experiment need to be purchased separately, or other input voltage methods can be used.

III. Experiment Steps

Import Project

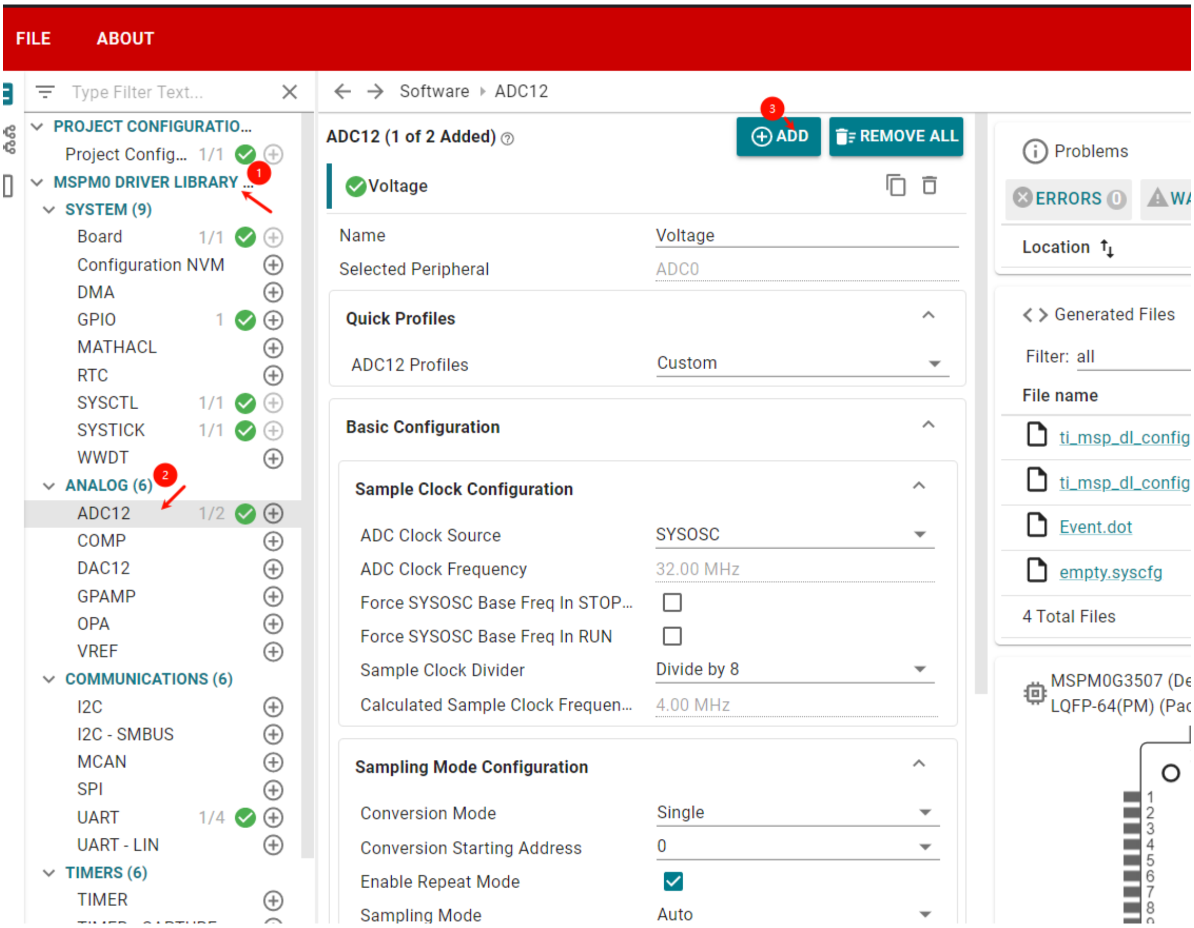
In KEIL, open our serial communication project from the previous section, or you can use the project template from the 80MHz clock frequency configuration chapter or the empty project from the SDK.

After selecting and opening, in the KEIL interface, open the empty.syscfg file. **With the empty.syscfg file open**, select Open SYSCONFIG GUI interface from the Tools menu.

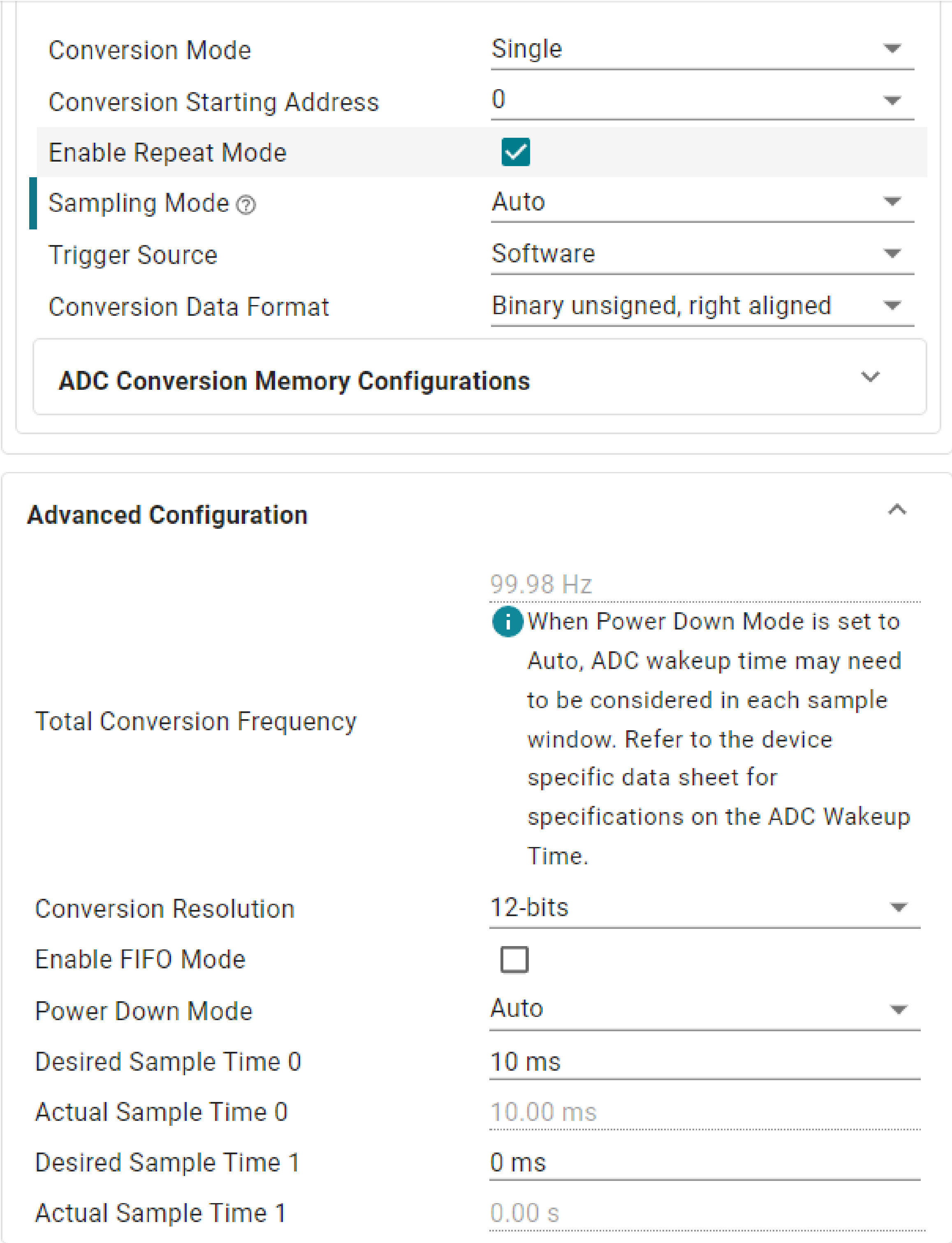
Add ADC Peripheral Configuration

Since we are porting the serial communication project from the previous section, serial configuration details can refer to the previous section.

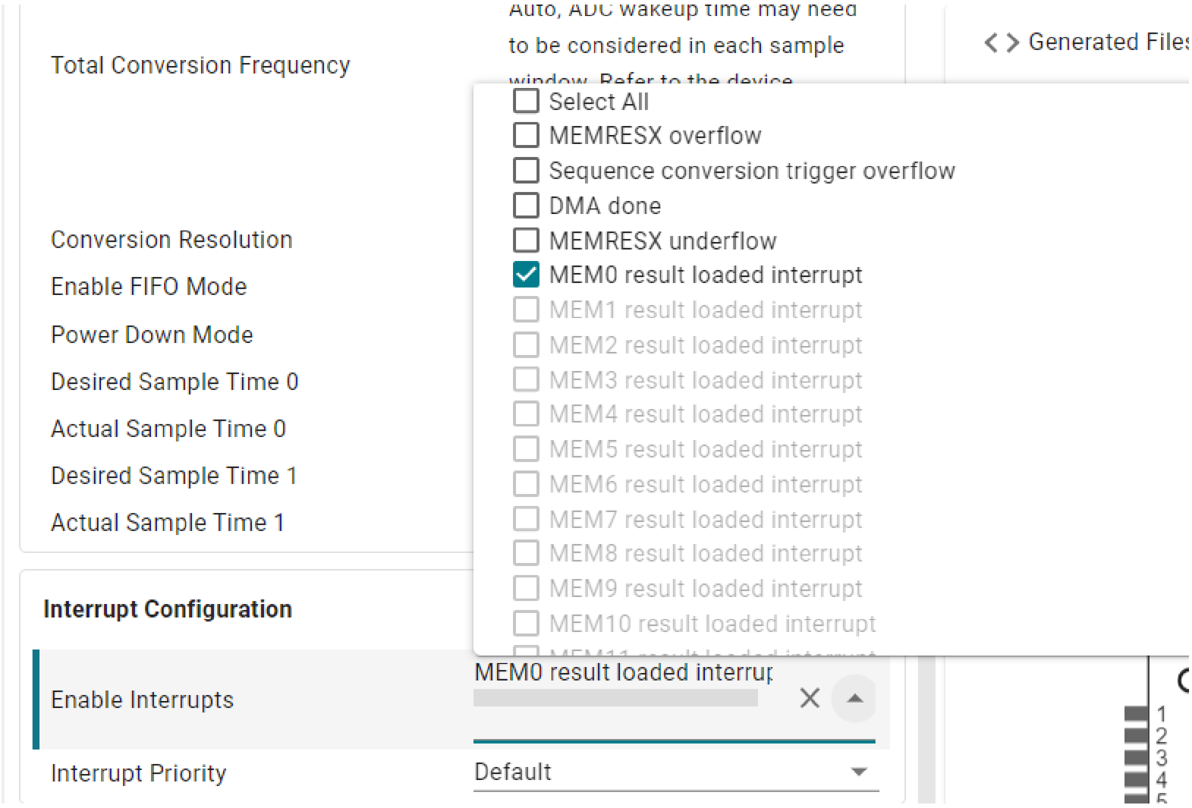
In SYSCONFIG, select MCU peripherals on the left, find ADC12 option and click to enter. In ADC12, click ADD to add ADC peripheral.



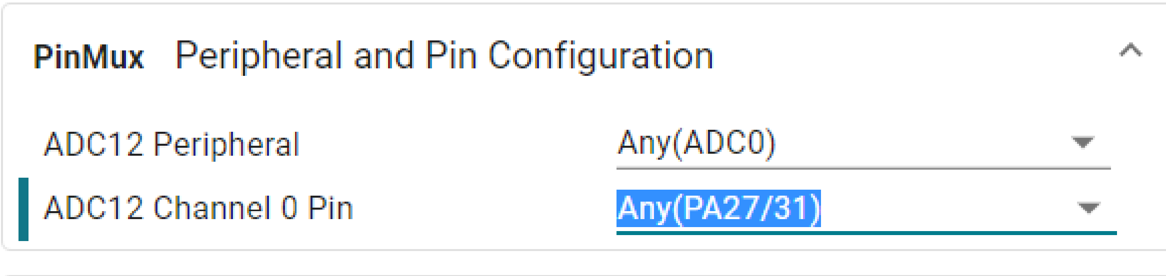
The configuration selected here is to use 32MHz ADC frequency, sampling frequency of 4MHz, single repeat conversion mode, ADC sampling is triggered by software, data format is binary right-aligned. We set the sampling time to 10ms



Enable memory 0 result load interrupt



Configure ADC0 channel 0 pin



Parameter description:

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

ADC Clock Frequency: Shows the current ADC clock frequency.

Sample Clock Divider: Sampling divider. Configured as divide by 8.

Calculated Sample clock Frequency: Shows the sampling frequency after division.

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

Conversion Starting Address: Conversion start address. Configured to start sampling from the 0th (related to the storage register below).

Enable Repeat Mode: Whether to enable repeat conversion. Checked here, enabled.

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

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

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

MSPM0G3507's ADC supports multiple data alignment methods to adapt to different application scenarios. Common data alignment methods include **left-aligned** and **right-aligned**.

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

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 most significant bits, with insufficient bits filled with 0 in the low bits. Left-aligned mode can improve resolution but results in reduced dynamic range.

Rule group data

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

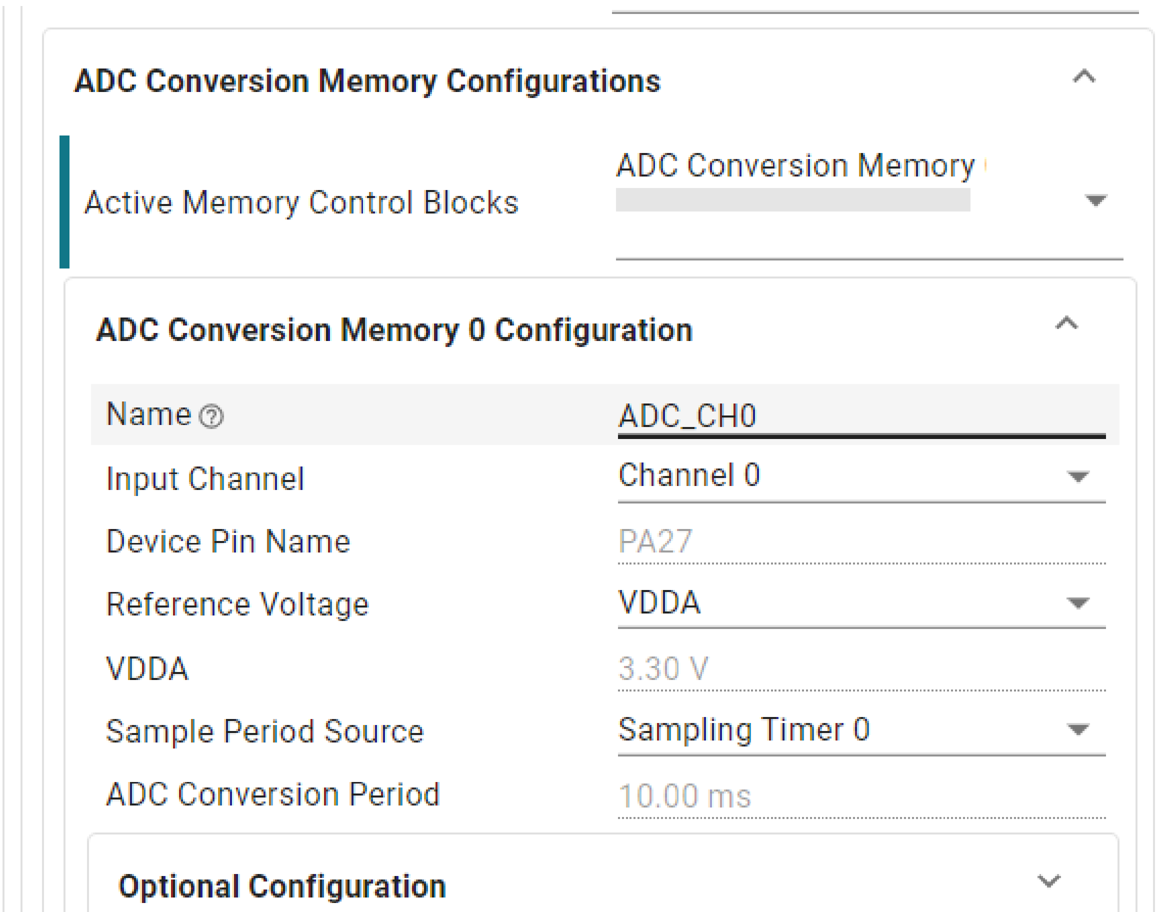
Injection group data

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

DAL=1

We configure as right-aligned method, so we can directly perform calculations on the converted data.

ADC conversion channel configuration



Parameter description:

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

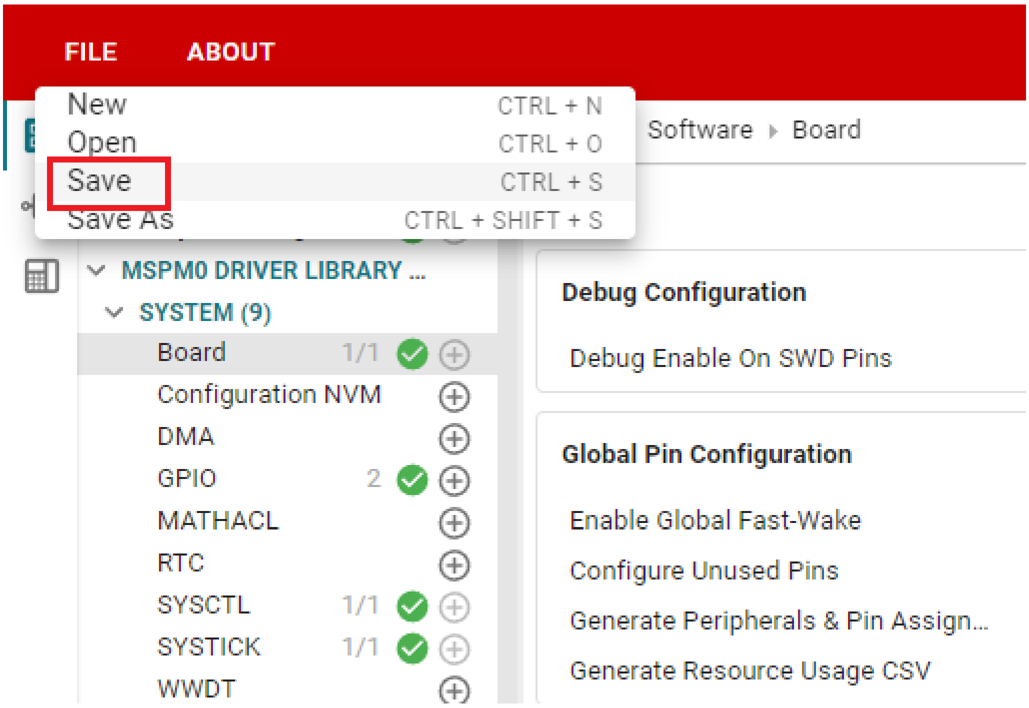
Input Channel: Input channel. Select channel 0 (each channel corresponds to different pins).

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

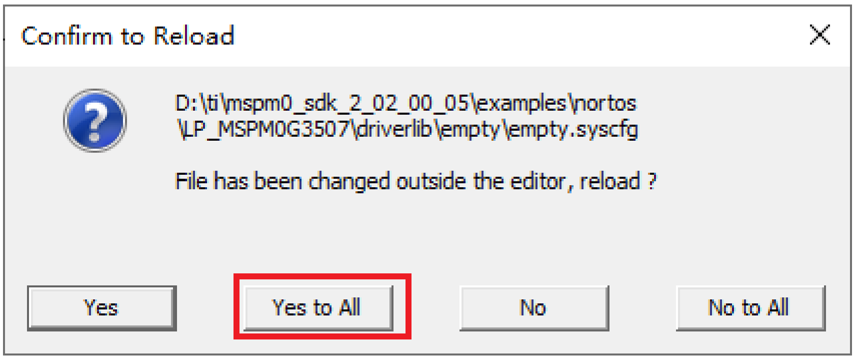
Reference Voltage: Voltage reference. Configured to use MCU power supply VDDA.

ADC Conversion Period: Shows the time for ADC to convert a single data

Click SAVE to save the configuration in SYSCONFIG, then close SYSCONFIG and return to keil.



Select `Yes to All` in the pop-up window



IV. Program Analysis

Like the previous section, we create new bsp_vol_adc.c, bsp_vol_adc.h under the BSP file, configured as follows

bsp_vol_adc.c

```
#include "bsp_vol_adc.h"

volatile bool gCheckADC;        // ADC acquisition success flag

volatile static int adc_value = 0;      //ADC raw value
volatile static int voltage_value = 0; // Converted voltage value
```

```
/**
 * @brief Voltage ADC module initialization
 */
void Vol_ADC_Init(void)
{
    //Enable voltage acquisition ADC interrupt
    NVIC_EnableIRQ(Voltage_INST_INT_IRQN);
}

/**
 * @brief Get voltage value from ADC
 */
void Get_Vol(void)
{
    //Get ADC data
    adc_value = adc_getValue();
}

/**
 * @brief Read ADC value and convert to voltage
 * @return ADC raw value
 */
unsigned int adc_getValue(void)
{
    volatile uint16_t gAdcResult = 0;  //ADC conversion result / ADC转换结果

    //Software trigger ADC to start conversion
    DL_ADC12_startConversion(Voltage_INST);

    //If current state is converting, wait for conversion to complete
    while (0 == gCheckADC) {
        __WFE(); // enter low-power mode 进入低功耗模式
    }

    //Get data
    gAdcResult = DL_ADC12_getMemResult(Voltage_INST, Voltage_ADCMEM_ADC_CH0);

    //Clear flag
    gCheckADC = false;

    //Enable ADC for next conversion
    DL_ADC12_enableConversions(Voltage_INST);


    //Scale factor 403 for final output
    //Convert ADC value to voltage (3.3V reference voltage, 12-bit ADC)
    voltage_value = (int)((gAdcResult*3.3/4095.0)*403);

    //Print voltage value, format: X.XXV
    printf("voltage 
value:%d.%d%dV\r\n",voltage_value/100,voltage_value/10%10,voltage_value%10);

    return gAdcResult;
}

/**
 * @brief ADC interrupt service function
 */
```

```c
void Voltage_INST_IRQHandler(void)
{
    //Query and clear ADC interrupt
    switch (DL_ADC12_getPendingInterrupt(Voltage_INST))
    {
        //Check if data acquisition is complete
        case DL_ADC12_IIDX_MEM0_RESULT_LOADED:
            gCheckADC = true; //Set flag
            break;
        default:
            break;
    }
}
```

bsp_vol_adc.h

```c
#ifndef __BSP_VOL_ADC_H_
#define __BSP_VOL_ADC_H_
#include "stdio.h"
#include "stdint.h"
#include "ti_msp_dl_config.h"

extern volatile bool gCheckADC;

void Get_Vol(void);
unsigned int adc_getValue(void);

#endif
```

empty.c

```c
#include "ti_msp_dl_config.h"
#include "uart0.h"
#include "bsp_vol_adc.h"

int main(void)
{
        //Serial port interrupt enable
        USART_Init();
        //ADC conversion interrupt enable
        NVIC_EnableIRQ(Voltage_INST_INT_IRQN);

    while (1)
    {
            Get_Vol();
    }

}
```

V. Experiment Phenomenon

We connect typeC to the computer and the car, then open the serial debugging assistant, baud rate 9600, eight data bits, one stop bit, no parity. **Note that when using typeC as debugging serial port output, it is not recommended to connect other sensors to serial port 0!**

We can toggle the car's switch and observe the voltage changes (battery needs to be connected)