DMA transfer

1. Learning objectives

1. Understand the basic knowledge of DMA.
2. Learn the basic use of DMA and display the transmitted ADC data on the serial port assistant.

DMA principle

DMA (Direct Memory Access) is a technology used in computer systems that allows peripherals to directly access memory without having to go through the CPU to complete data transmission. It enables data to be exchanged efficiently between peripherals (such as ADC, DAC, memory, etc.) and memory, reducing the burden on the CPU and improving the overall performance of the system.

Principle:

- **Triggering data transfer**: DMA operations are usually triggered by external devices (such as ADC, sensors) or internal events (such as timer overflow).
- **Initialize DMA controller**: The CPU configures the DMA controller with parameters such as source address, destination address, transfer data size, and transfer direction.
- **Direct data transfer**: Once the DMA controller is activated, it can directly transfer data from the source address (the output buffer or memory of the peripheral) to the target address (memory or peripheral) without CPU intervention.
- **Notify CPU after transfer is completed**: After DMA completes data transfer, it can send an interrupt request to the CPU to tell the CPU that the data is ready or that the transfer has been successfully completed.

2. Hardware Construction

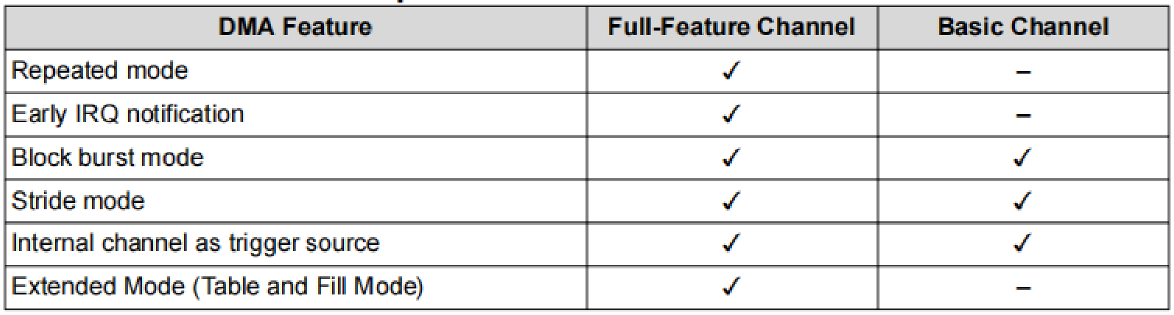
The DMA controller of MSPM0G3507 has the following features:

- 7 independent transfer channels;
- Configurable DMA channel priority;
- Support 8-bit (byte), 16-bit (short word), 32-bit (word) and 64-bit (long word) or mixed size (byte and word) transfer;
- Support data block transfer of up to 64K of any data type;

- Configurable DMA transfer trigger source;
- 6 flexible addressing modes;
- Single or block transfer mode; It has a total of 7 DMA channels, each of which can be configured independently, and a variety of data transfer modes can adapt to the data transfer needs of different application scenarios.

By looking at TI's data sheet, in addition to the common address addressing mode between memory and peripherals, the DMA function also provides two expansion modes: Fill Mode and Table Mode. DMA channels are divided into basic type and full-function type.

Table 4-1. Feature Comparison of Basic and Full-Feature DMA Channels



DMA Feature	Full-Feature Channel	Basic Channel
Repeated mode	✓	–
Early IRQ notification	✓	–
Block burst mode	✓	✓
Stride mode	✓	✓
Internal channel as trigger source	✓	✓
Extended Mode (Table and Fill Mode)	✓	–

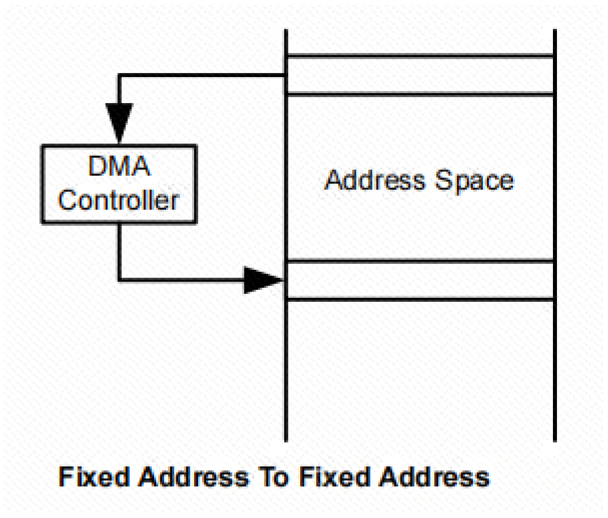
By looking at the data sheet, only the full-function type DMA channel supports repeated transfers, early interrupts, and extended modes. The basic function DMA channel supports basic data transfers and interrupts, but is sufficient to meet simple data transfer requirements.

The DMA of the MSPM0G3507 supports four transfer modes, and each channel can configure its transfer mode separately. For example, channel 0 can be configured as a single word or single byte transfer mode, while channel 1 is configured as a block transfer mode, and channel 2 uses a repeated block transfer mode. The transfer mode is configured independently of the addressing mode. Any addressing mode can be used with any transfer mode. Three types of data can be transferred, which can be selected by the .srcWidth and .destWidth control bits. The source and destination locations can be byte, short word, or word data. It also supports transfers in byte-to-byte, short word-to-short word, word-to-word, or any combination. As follows:

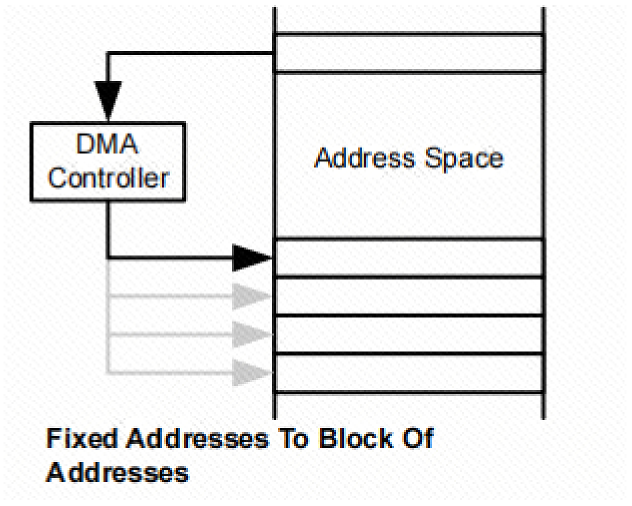
- Single word or single byte transfer: Each transfer requires a separate trigger. DMA is automatically disabled when DMAxSZ transfers have been generated.
- Block transfers: a whole block will be transferred after a trigger. DMA is automatically disabled at the end of the block transfer.
- Repeated single word or single byte transfers: a separate trigger is required for each transfer, DMA remains enabled.
- Repeated block transfers: a whole block will be transferred after a trigger, DMA remains enabled.

Based on these three modes, MSPM0G3507 provides 6 addressing modes,

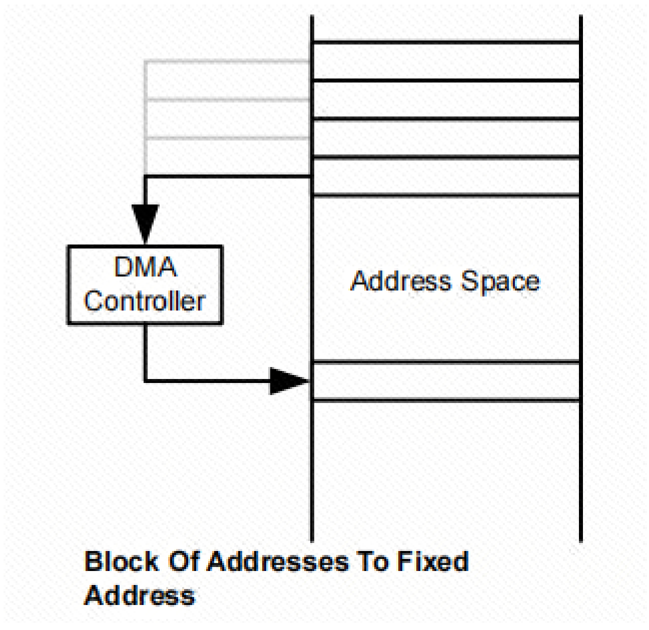
Fixed address to fixed address:



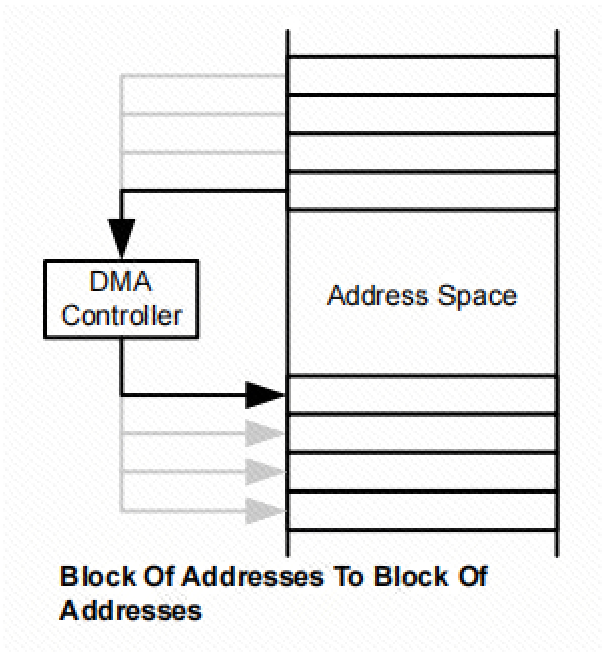
Fixed address to block of address :



Block of address to fixed address:

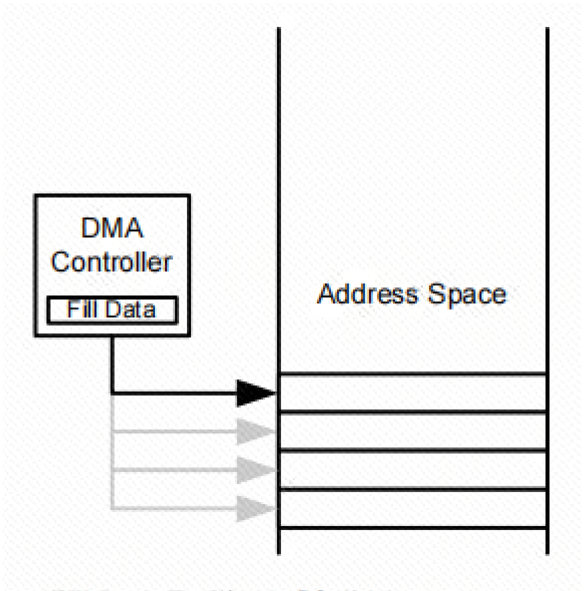


Block of address to block of address:



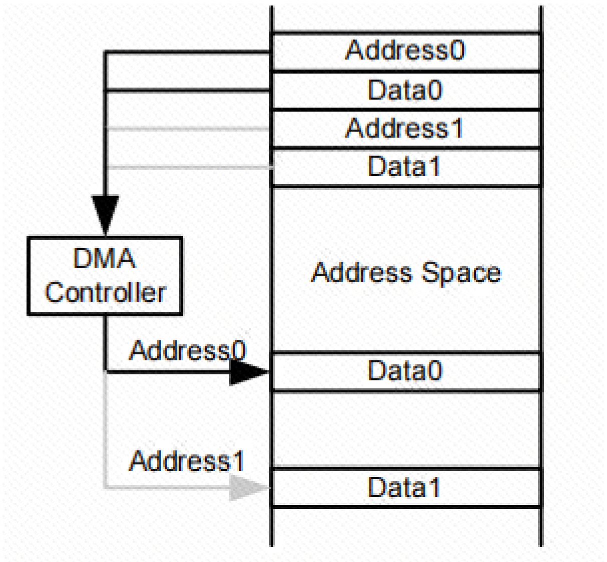
Block Of Addresses To Block Of Addresses

Fill data to block of address:



Fill Data To Block Of Addresses

Data table to specific address:



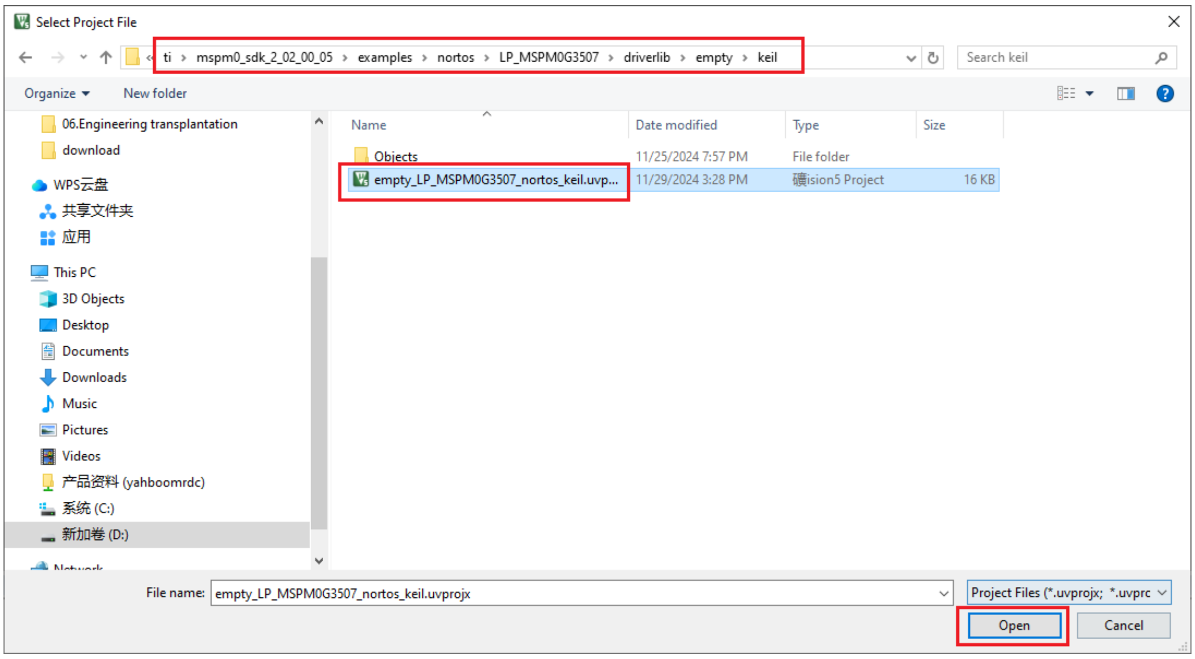
Data Table To Specific Address

3. Experimental steps

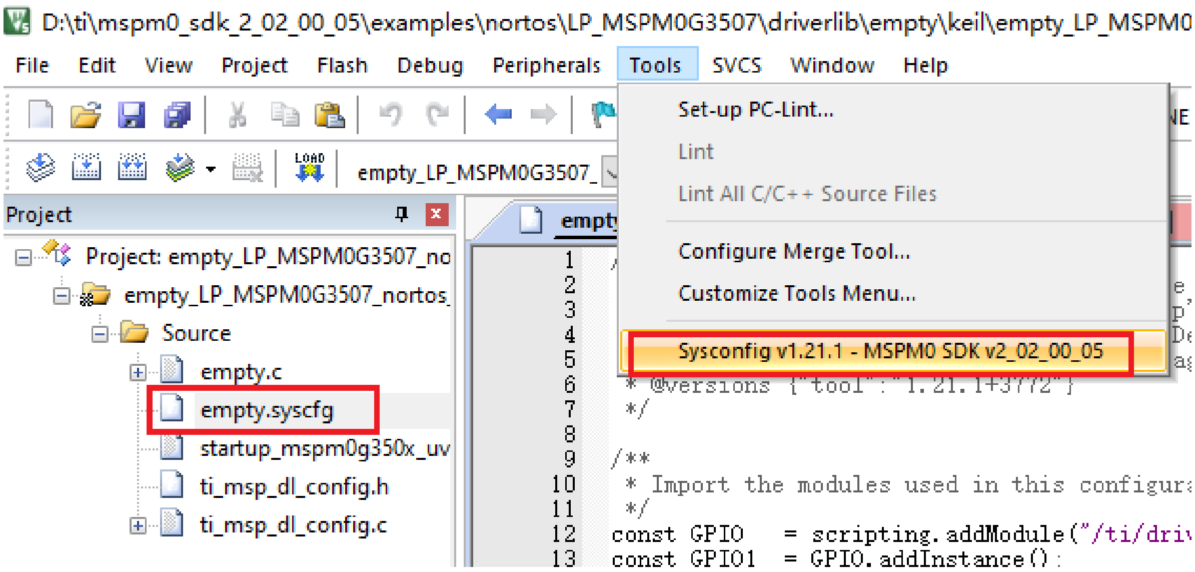
In this case, ADC will be used to collect the voltage of the PA27 pin, and the data collected by ADC will be directly moved to the specified address through DMA.

1. Open the SYSCONFIG configuration tool

Open the blank project empty in the SDK in KEIL.



Select and open, open the empty.syscfg file in the keil interface, **When the empty.syscfg file is open**, select Open the SYSCONFIG GUI interface in the Tools column.

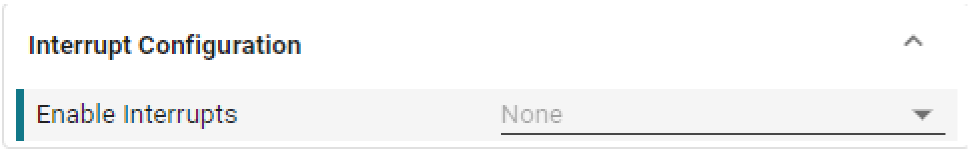


```
 * @versions {"tool":"1.21.1+3772"}
 */

/**
 * Import the modules used in this configura
 */
const GPIO   = scripting.addModule("/ti/driv
const GPIO1  = GPIO.addInstance();
```

2. ADC parameter configuration

The ADC configuration here is the same as the **ADC acquisition** case.

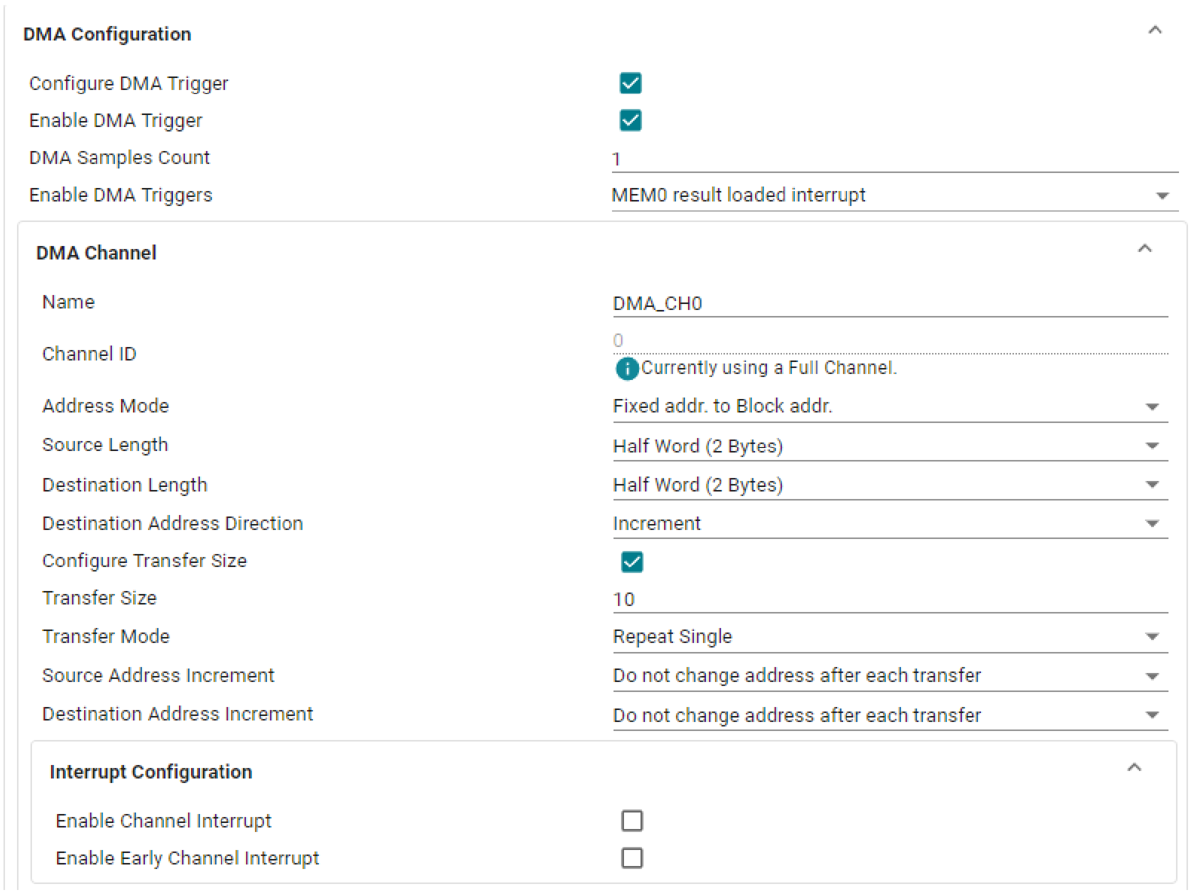
Configure the resolution and interrupt of the ADC. This case does not use interrupts, so do not select anything here.



Interrupt Configuration

Enable Interrupts: None

3. DMA parameter configuration



DMA Configuration

Configure DMA Trigger	✓
Enable DMA Trigger	✓
DMA Samples Count	1
Enable DMA Triggers	MEM0 result loaded interrupt

DMA Channel

Name	DMA_CH0
Channel ID	0 Currently using a Full Channel.
Address Mode	Fixed addr. to Block addr.
Source Length	Half Word (2 Bytes)
Destination Length	Half Word (2 Bytes)
Destination Address Direction	Increment
Configure Transfer Size	✓
Transfer Size	10
Transfer Mode	Repeat Single
Source Address Increment	Do not change address after each transfer
Destination Address Increment	Do not change address after each transfer

Interrupt Configuration

Enable Channel Interrupt	☐
Enable Early Channel Interrupt	☐

Parameter description:

Configure DMA Trigger: Whether to enable the DMA configuration function, select Enable here.

Enable DMA Trigger: Enable DMA trigger.

DMA Samples Count: DMA sample count. Set to 1.

Enable DMA Triggers: DMA trigger mode. Configured as `MEM0 result loaded interrupt` ADC result register 0 is loaded interrupt.

Address Mode: DMA addressing mode. Configured as `Fixed addr. to Block addr.` Fixed address to data block address.

Source Length: Data width of source address. Configured as `Half Word (2 Bytes)` Data width 2 bytes.

Destination Length: Data width configuration of destination address. Same as above.

Destination Address Direction: Destination address direction. Select `Increment` to enable increment of data block address.

Configure Transfer size: Whether to configure the amount of data to be transferred. Select Enable here.

Transfer Size: The amount of data to be transferred. Configured as 10, the amount of data transferred in one data block is 10 numbers.

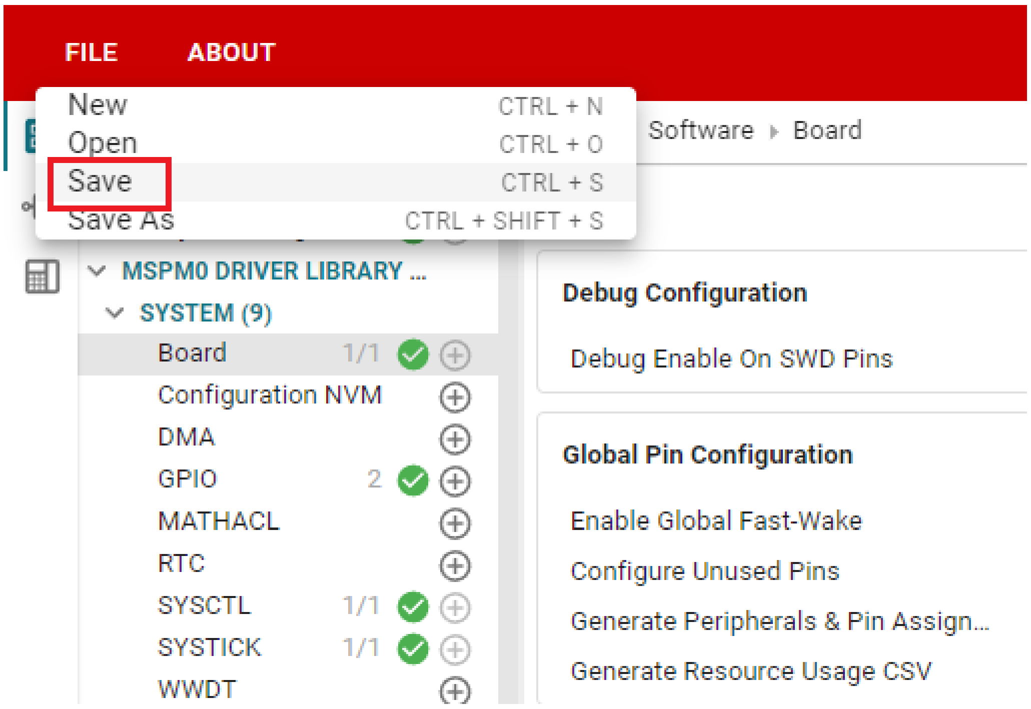
Transfer Mode: DMA transfer mode. Configure to `Repeat Single` to repeat a single transfer.

Source Address Increment: Whether the source address is incremented. Not enabled.

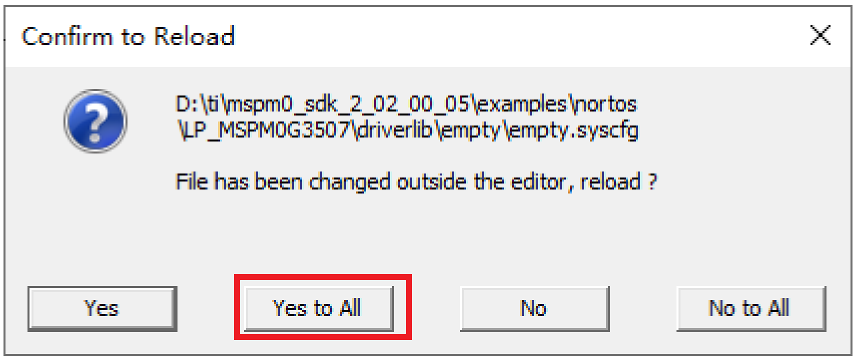
Destination Address Increment: Whether the destination address is incremented. Not enabled.

Enable Channel Interrupt: Whether to enable DMA channel interrupt. Not enabled.

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



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


Confirm to Reload

D:\ti\mspm0_sdk_2_02_00_05\examples\nortos\LP_MSPM0G3507\driverlib\empty\empty.syscfg

File has been changed outside the editor, reload ?

Similarly, we also need to confirm whether the ti_msp_dl_config.c and ti_msp_dl_config.h files are updated. Compile directly, and the compilation will automatically update to keil. If there is no update, we can also copy the file content in SYSCONFIG.

4. Write the program

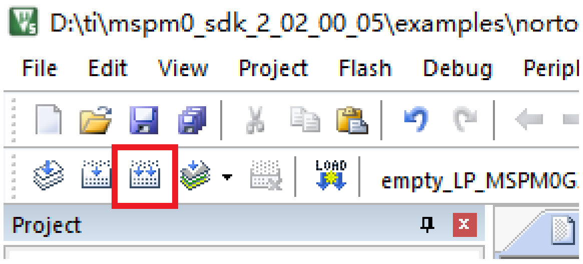
In the empty.c file, write the following code

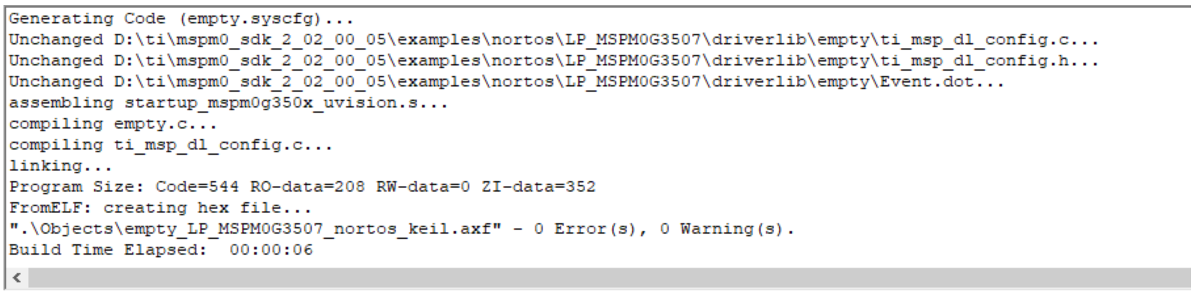
```
#include "ti_msp_dl_config.h"
#include "stdio.h"

volatile unsigned int delay_times = 0;
volatile bool gCheckADC;        //ADC采集成功标志位 ADC acquisition success flag
volatile uint16_t ADC_VALUE[20];

void delay_ms(unsigned int ms); //搭配滴答定时器的精准毫秒级延时 Precise millisecond
delay with tick timer
unsigned int adc_getValue(unsigned int number);//读取ADC的数据 Read ADC data


/******************************串口重定向 Serial port
redirection****************************************/
```

```
#if !defined(__MICROLIB)
//不使用微库的话就需要添加下面的函数
// If you don't use the micro library, you need to add the following function
#if (__ARMCLIB_VERSION <= 6000000)
//如果编译器是AC5  就定义下面这个结构体
//If the compiler is AC5, define the following structure
struct __FILE
{
        int handle;
};
#endif

FILE __stdout;

//定义_sys_exit()以避免使用半主机模式
// Define _sys_exit() to avoid using semihosting mode
void _sys_exit(int x)
{
        x = x;
}
#endif
//printf函数重定义 printf function redefinition
int fputc(int ch, FILE *stream)
{
        //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 );

        DL_UART_Main_transmitData(UART_0_INST, ch);

        return ch;
}
/***********************************************************************/

int main(void)
{
        unsigned int adc_value = 0;
        unsigned int voltage_value = 0;
        int i = 0;

        SYSCFG_DL_init();

        //清除串口中断标志 Clear the serial port interrupt flag
        NVIC_ClearPendingIRQ(UART_0_INST_INT_IRQN);
        //开启串口中断 Enable serial port interrupt
        NVIC_EnableIRQ(UART_0_INST_INT_IRQN);

        //设置DMA搬运的起始地址 Set the starting address of DMA transfer
        DL_DMA_setSrcAddr(DMA, DMA_CH0_CHAN_ID, (uint32_t) &ADC0-
>ULLMEM.MEMRES[0]);
        //设置DMA搬运的目的地址 Set the destination address of DMA transfer
        DL_DMA_setDestAddr(DMA, DMA_CH0_CHAN_ID, (uint32_t) &ADC_VALUE[0]);
        //开启DMA Enable DMA
        DL_DMA_enableChannel(DMA, DMA_CH0_CHAN_ID);
        //开启ADC转换 Start ADC conversion
        DL_ADC12_startConversion(ADC_VOLTAGE_INST);
        printf("adc_dma Demo start\r\n");

        while (1)
        {
                //获取ADC数据 Get ADC data
                adc_value = adc_getValue(10);
                printf("adc value:%d\r\n", adc_value);

                //将ADC采集的数据换算为电压 Convert the data collected by ADC into 
voltage
                voltage_value = (int)((adc_value/4095.0*3.3)*100);

                printf("voltage value:%d.%d%d\r\n",
                voltage_value/100,
                voltage_value/10%10,
                voltage_value%10 );

                delay_ms(1000);
        }
}
//延时函数 Delay function
void delay_ms(unsigned int ms)
{
        delay_times = ms;
        while( delay_times != 0 );
}

//读取ADC的数据 Read ADC data
unsigned int adc_getValue(unsigned int number)
{
        unsigned int gAdcResult = 0;
        unsigned char i = 0;

        //采集多次累加 Collect multiple times and accumulate
        for( i = 0; i < number; i++ )
        {
                gAdcResult += ADC_VALUE[i];
        }
        //均值滤波 Mean Filter
        gAdcResult /= number;

        return gAdcResult;
}

//滴答定时器的中断服务函数 Tick timer interrupt service function
void SysTick_Handler(void)
{
        if( delay_times != 0 )
        {
                delay_times--;
        }
}
```

5. Compile

Click the Rebuild icon. The following prompt appears, indicating that the compilation is complete and there are no errors.





```
Generating Code (empty.syscfg)...
Unchanged D:\ti\mspm0_sdk_2_02_00_05\examples\nortos\LP_MSPM0G3507\driverlib\empty\ti_msp_dl_config.c...
Unchanged D:\ti\mspm0_sdk_2_02_00_05\examples\nortos\LP_MSPM0G3507\driverlib\empty\ti_msp_dl_config.h...
Unchanged D:\ti\mspm0_sdk_2_02_00_05\examples\nortos\LP_MSPM0G3507\driverlib\empty\Event.dot...
assembling startup_mspm0g350x_uvision.s...
compiling empty.c...
compiling ti_msp_dl_config.c...
linking...
Program Size: Code=544 RO-data=208 RW-data=0 ZI-data=352
FromELF: creating hex file...
".\Objects\empty_LP_MSPM0G3507_nortos_keil.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed:  00:00:06
```

4. Program Analysis

- empty.c

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

        //采集多次累加 Collect multiple times and accumulate
        for( i = 0; i < number; i++ )
        {
                gAdcResult += ADC_VALUE[i];
        }
        //均值滤波 Mean Filter
        gAdcResult /= number;

        return gAdcResult;
}
```

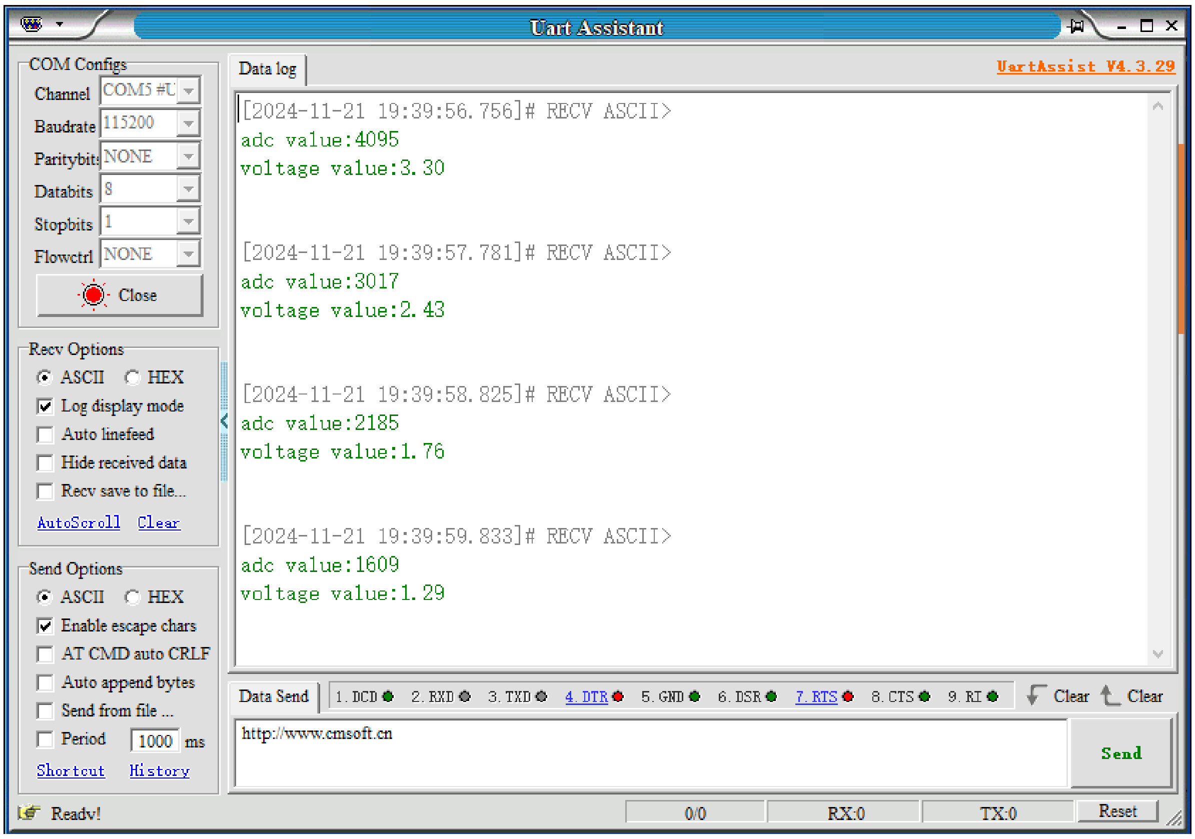
This function filters by accumulating the data in the `ADC_VALUE` array and calculating the average value. This can reduce the noise of a single ADC sampling. `number` is the number of samples, usually set to 10 or more to improve the stability of the data.

```c
int main(void)
{
        unsigned int adc_value = 0;
        unsigned int voltage_value = 0;
        int i = 0;

        SYSCFG_DL_init();

        //清除串口中断标志 Clear the serial port interrupt flag
        NVIC_ClearPendingIRQ(UART_0_INST_INT_IRQN);
        //开启串口中断 Enable serial port interrupt
        NVIC_EnableIRQ(UART_0_INST_INT_IRQN);

        //设置DMA搬运的起始地址 Set the starting address of DMA transfer
        DL_DMA_setSrcAddr(DMA, DMA_CH0_CHAN_ID, (uint32_t) &ADC0-
>ULLMEM.MEMRES[0]);
        //设置DMA搬运的目的地址 Set the destination address of DMA transfer
        DL_DMA_setDestAddr(DMA, DMA_CH0_CHAN_ID, (uint32_t) &ADC_VALUE[0]);
        //开启DMA Enable DMA
        DL_DMA_enableChannel(DMA, DMA_CH0_CHAN_ID);
        //开启ADC转换 Start ADC conversion
        DL_ADC12_startConversion(ADC_VOLTAGE_INST);

        printf("adc_dma Demo start\r\n");

        while (1)
        {
                //获取ADC数据 Get ADC data
                adc_value = adc_getValue(10);
                printf("adc value:%d\r\n", adc_value);

                //将ADC采集的数据换算为电压 Convert the data collected by ADC into
voltage
                voltage_value = (int)((adc_value/4095.0*3.3)*100);

                printf("voltage value:%d.%d%d\r\n",
                voltage_value/100,
                voltage_value/10%10,
                voltage_value%10 );

                delay_ms(1000);
        }
}
```

Call `SYSCFG_DL_init()` function to initialize the system, clear and enable serial port interrupt for serial port data transmission. Set DMA source address to `ADC0->ULLMEM.MEMRES[0]`, which stores ADC conversion results. Set DMA destination address to `ADC_VALUE[0]`, and store DMA results in this array. Enable DMA channel and start ADC conversion. Get 10 ADC sampling results through `adc_getValue(10)` and perform mean filtering to return a stable ADC value. Convert ADC value to voltage value, assuming ADC has 12-bit resolution (4095 is the maximum value), and convert it to voltage value in the range of 0-3.3V.

5. Experimental phenomenon

After the program is downloaded, configure the serial port assistant as shown below, adjust the potentiometer knob, output voltage to PA27 pin, and after ADC loads data acquisition into ADC result register 0, DMA is triggered, and DMA will move data to memory variable ADC_VALUE. Data can be obtained by directly calling ADC_VALUE.



Channel: COM5 #U
Baudrate: 115200
Paritybits: NONE
Databits: 8
Stopbits: 1
Flowctrl: NONE

Recv Options: [✓ ASCII] HEX
[✓] Log display mode
[ ] Auto linefeed
[ ] Hide received data
[ ] Recv save to file...

Send Options: [✓ ASCII] HEX
[✓] Enable escape chars
[ ] AT CMD auto CRLF
[ ] Auto append bytes
[ ] Send from file ...
[ ] Period 1000 ms

```
[2024-11-21 19:39:56.756]# RECV ASCII>
adc value:4095
voltage value:3.30


[2024-11-21 19:39:57.781]# RECV ASCII>
adc value:3017
voltage value:2.43


[2024-11-21 19:39:58.825]# RECV ASCII>
adc value:2185
voltage value:1.76


[2024-11-21 19:39:59.833]# RECV ASCII>
adc value:1609
voltage value:1.29
```

1.DCD ● 2.RXD ● 3.TXD ● 4.DTR ● 5.GND ● 6.DSR ● 7.RTS ● 8.CTS ● 9.RI ●

http://www.cmsoft.cn

0/0 RX:0 TX:0