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

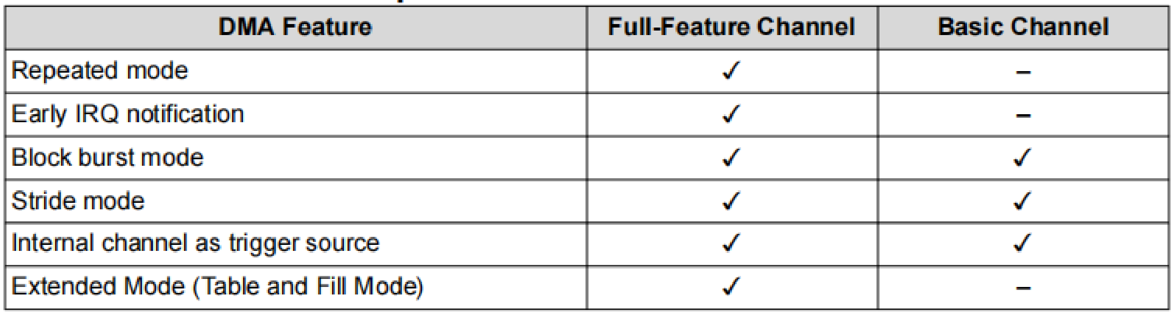
The DMA controller of MSPM0G3507 has the following features:

- 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)	✓	–

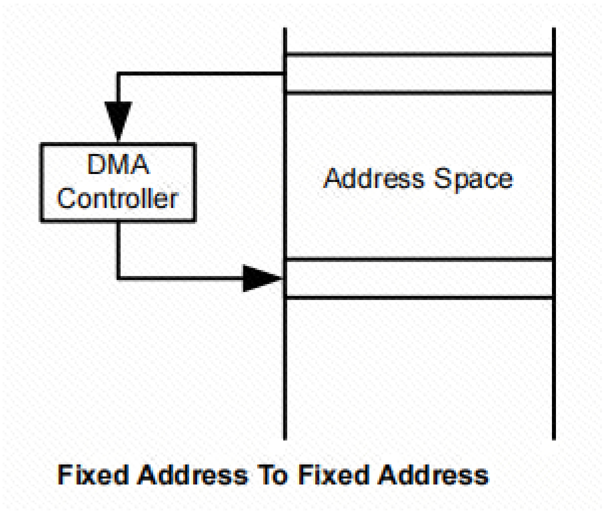
By looking at the data sheet, only the full-function type DMA channel supports repeated transfers, early interrupts, and extended modes. The basic function DMA channel supports basic data transfers and interrupts, but is sufficient to meet simple data transfer requirements.

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:

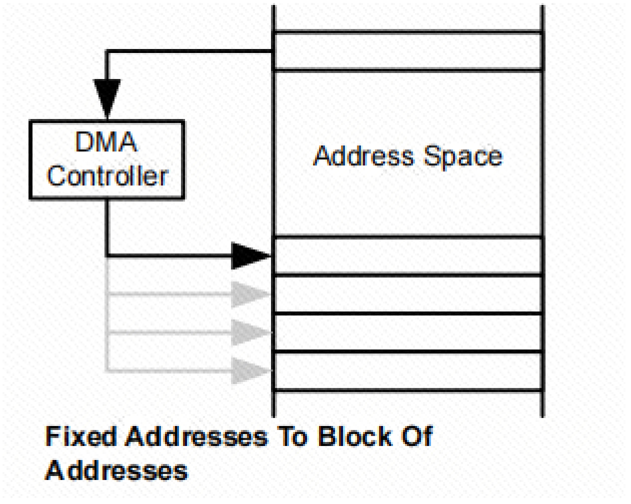
- Single word or single byte transfer: Each transfer requires a separate trigger. DMA is automatically disabled when DMAxSZ transfers have been generated.
- Block transfers: a whole block will be transferred after a trigger. DMA is automatically disabled at the end of the block transfer.
- Repeated single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

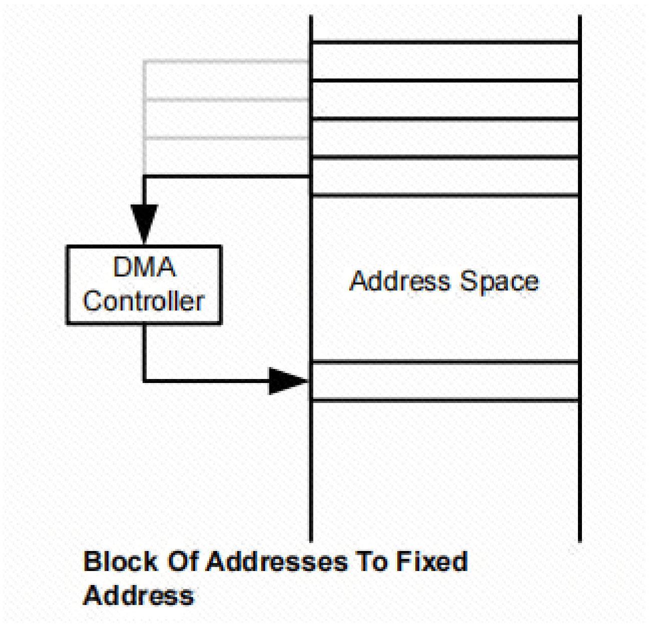
Fixed address to fixed address:



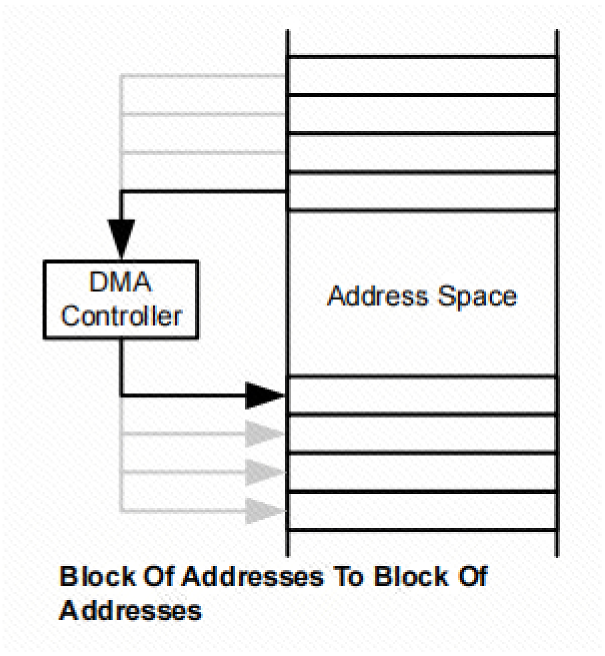
Fixed address to block of address :



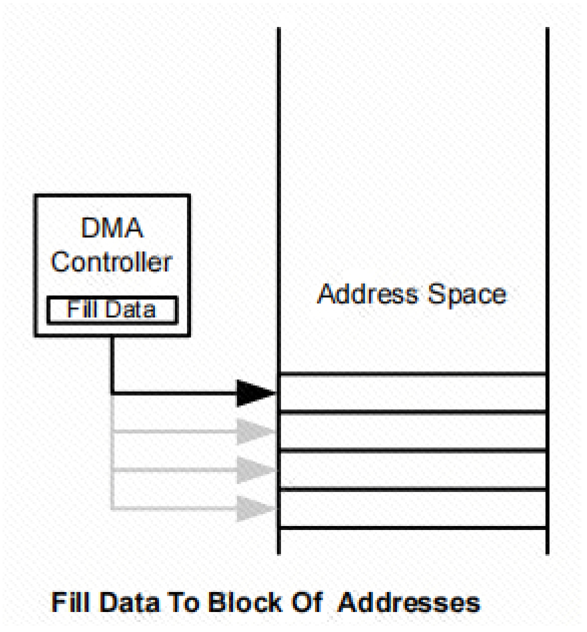
Block of address to fixed address:



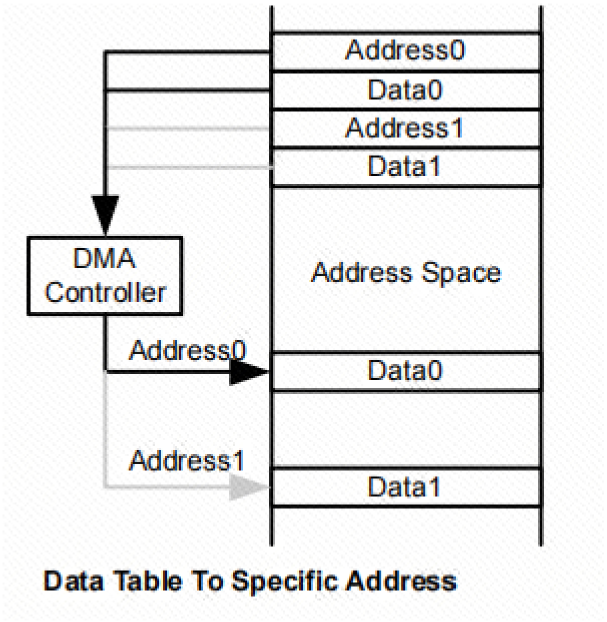
Block of address to block of address:



Fill data to block of address:



Data table to specific address:

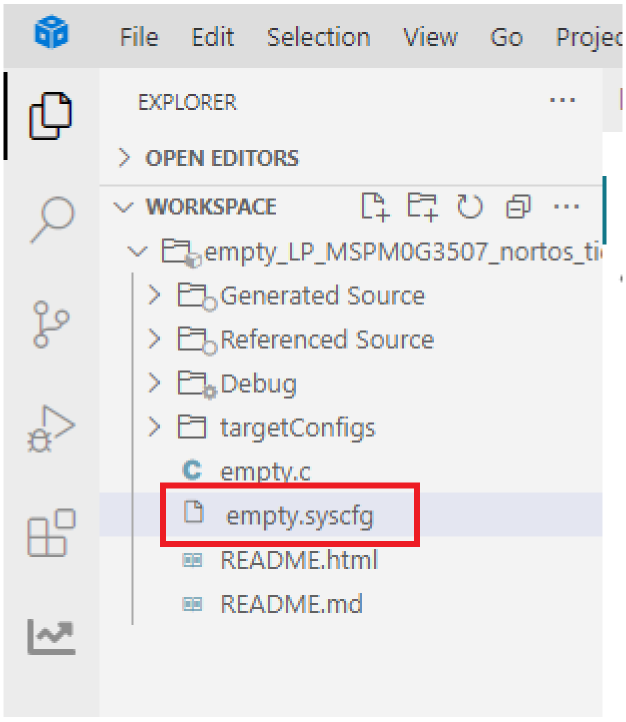


3. Experimental steps

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

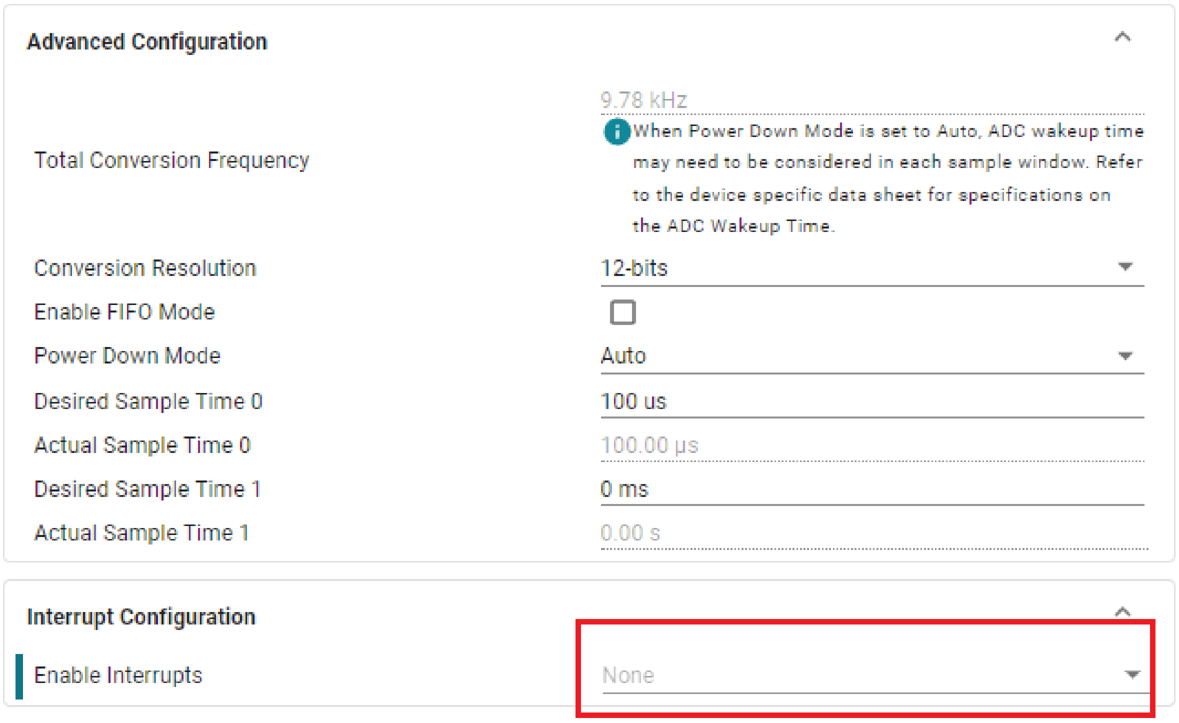
Here, the **ADC acquisition** course is used as a template for introduction. Open the ADC project and open the .syscfg file.



2. ADC parameter configuration

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

Configure the resolution and interrupt of ADC. This case does not use interrupts, so nothing is selected here.



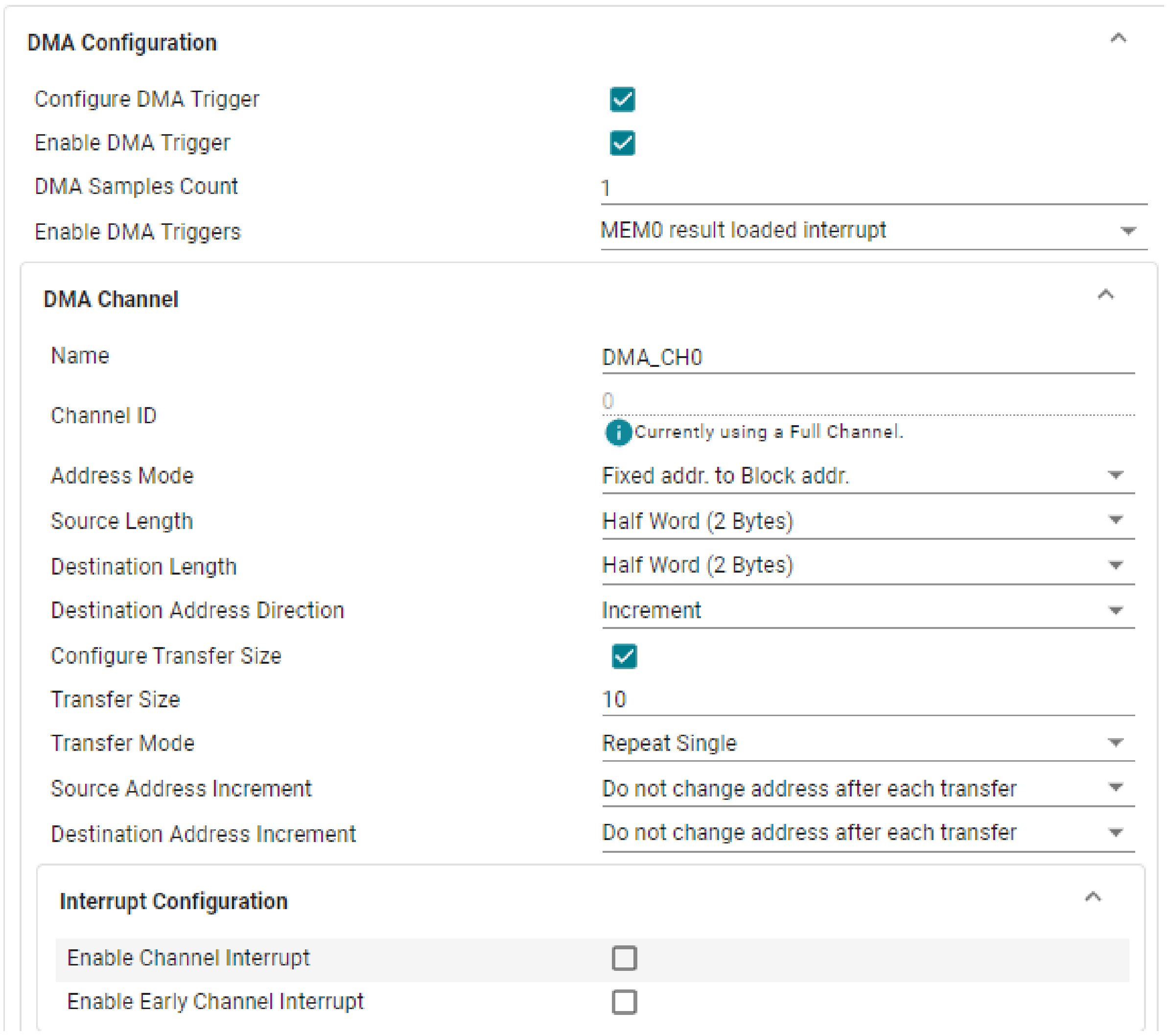
Advanced Configuration

Total Conversion Frequency	9.78 kHz When Power Down Mode is set to Auto, ADC wakeup time may need to be considered in each sample window. Refer to the device specific data sheet for specifications on the ADC Wakeup Time.
Conversion Resolution	12-bits
Enable FIFO Mode	☐
Power Down Mode	Auto
Desired Sample Time 0	100 us
Actual Sample Time 0	100.00 µs
Desired Sample Time 1	0 ms
Actual Sample Time 1	0.00 s

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	☐

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

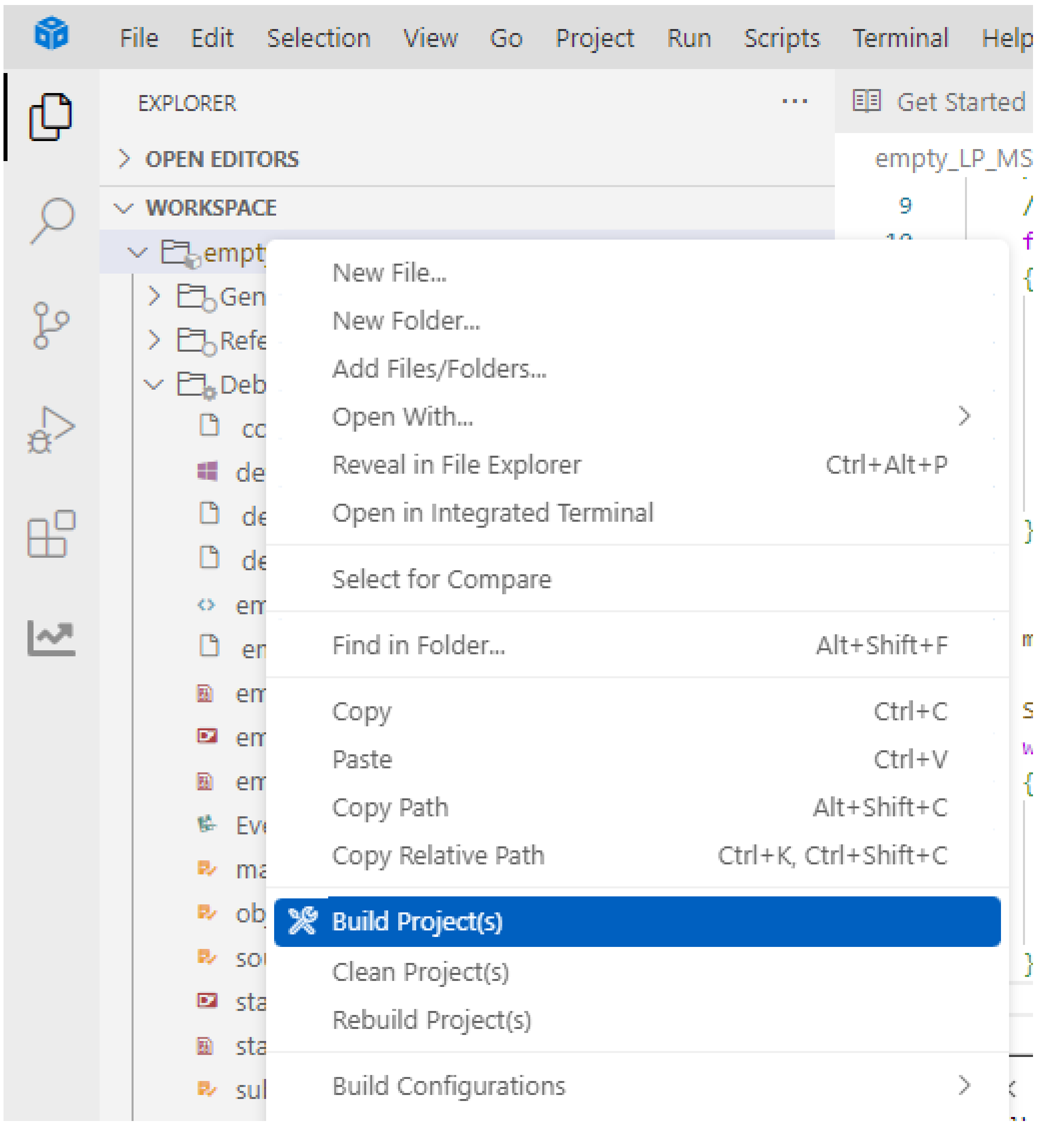
4. Write the program

In the empty.c file, write the following code

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

volatile uint16_t ADC_VALUE[20];//ADC采集的数据保存地址 The data collected by ADC is
saved in the address

unsigned int adc_getValue(unsigned int number); //读取ADC的数据 Read ADC data
void uart0_send_string(char* str); //串口发送字符串 Send string via serial port

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

    SYSCFG_DL_init();

    //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);

    uart0_send_string("adc+dma Demo start\r\n");
    while (1)
    {
        //获取ADC数据 Get ADC data
        adc_value = adc_getValue(10);
        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(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;
}

//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 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++);
    }
```

```
}
```

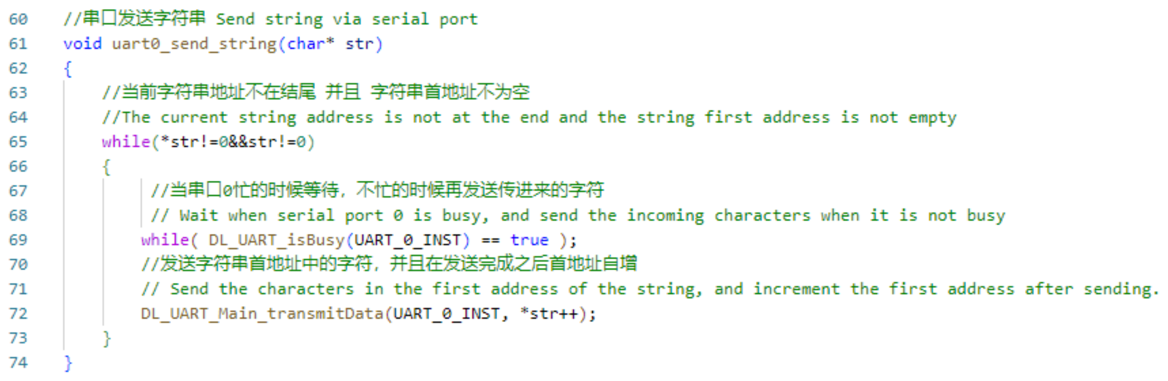
5. Compile



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 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 string ends (`*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. Send the current character through

`DL_UART_Main_transmitData(UART_0_INST, *str++)`, and let `str` point to the next character.

```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;
}
```

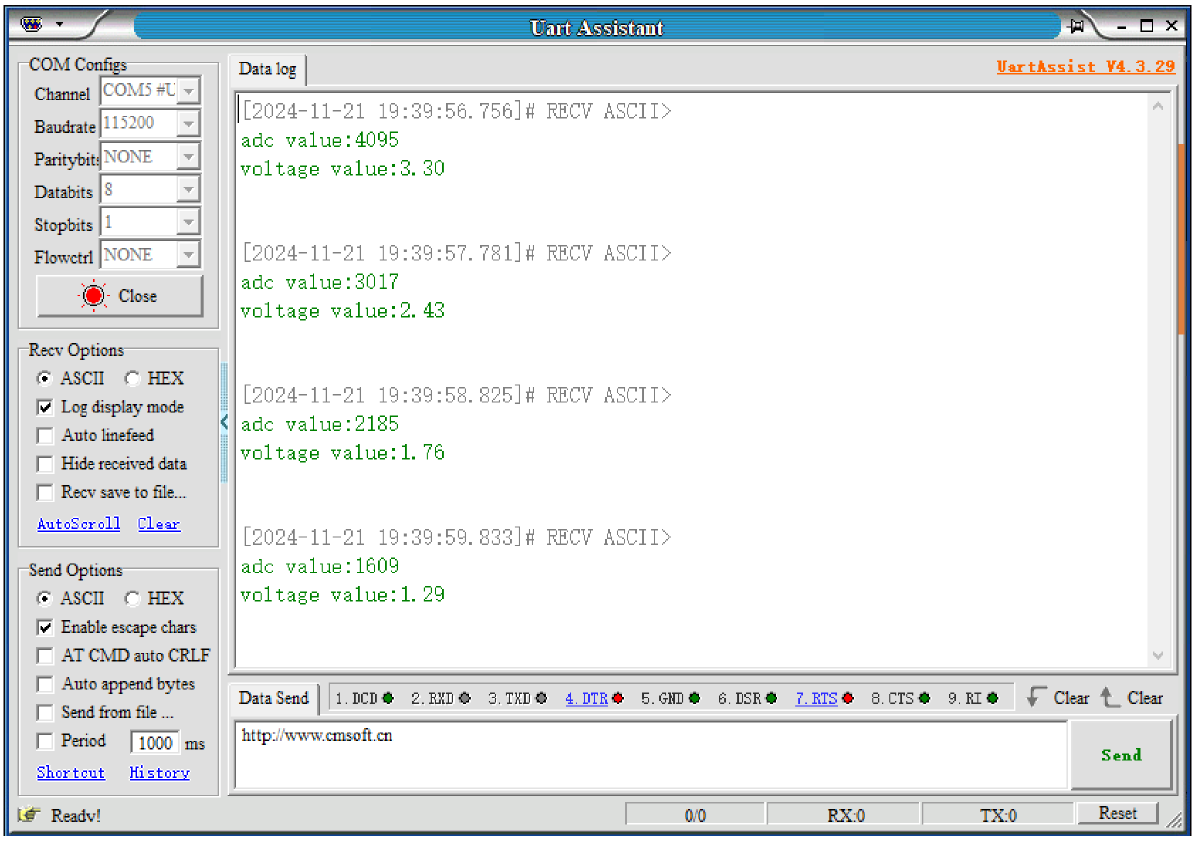
This code reads ADC (analog-to-digital converter) data. By triggering ADC conversion and waiting for the conversion to complete, the ADC sampling result is finally obtained and returned.

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

    SYSCFG_DL_init();

    //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);

    uart0_send_string("adc+dma Demo start\r\n");
    while (1)
    {
        //获取ADC数据 Get ADC data
        adc_value = adc_getValue(10);
        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);
    }
}
```

The function of this code is to collect analog signals and output the results through the serial port by combining **ADC** (analog-to-digital converter) and **DMA** (direct memory access). ADC data is directly moved to memory through DMA, without manual reading every time, and ADC sampling values and calculated voltage values are regularly output through the serial port.

5. Experimental phenomenon

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

http://www.cmsoft.cn