```
#include "sys.h"
#include "delay.h"
#include "VM_usart.h"
#include "VM_sram.h"
#include "VM_flash.h"
#include "VM_ADC.h"

int main(void){

   int i = 0;
   float ADC_T[8] = {1};



   NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
   delay_init(168);

   USART1_Init(115200);
   USART3_Init(115200);
   USART3_Enable_Init();

   FSMC_SRAM_Init();
   ADC_Init();

   USART3_Enable_Set();


   ADC_Config();


   while(1){
      ADC_Read(ADC_T);
      for(i = 0; i<8; i++){
         printf(开始测试%f\n",i,ADC_T[i]);
      }
      delay_ms(1000);
   }
}
#include "VM_ADC.h"
#include "delay.h"


void ADC_Init(void){

    GPIO_InitTypeDef GPIO_InitStructure;


    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOF|
                                                RCC_AHB1Periph_GPIOB|
                                                RCC_AHB1Periph_GPIOC|
                                                RCC_AHB1Periph_GPIOA|
```

```
                                                    RCC_AHB1Periph_GPIOE|
                                                    RCC_AHB1Periph_GPIOG
                                                    ,ENABLE);


    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;


    GPIO_InitStructure.GPIO_Pin = ADC_DB0| ADC_DB1| ADC_DB2;
    GPIO_Init(GPIOA,&GPIO_InitStructure);


    GPIO_InitStructure.GPIO_Pin = ADC_DB3| ADC_DB4| ADC_DB5| ADC_DB6;
    GPIO_Init(GPIOF,&GPIO_InitStructure);


    GPIO_InitStructure.GPIO_Pin = ADC_DB7| ADC_DB8| ADC_DB9| ADC_DB10;
    GPIO_Init(GPIOC,&GPIO_InitStructure);


    GPIO_InitStructure.GPIO_Pin  =  ADC_DB11|  ADC_DB12|  ADC_DB13|  ADC_BUSY|
ADC_FRSTDATA;
    GPIO_Init(GPIOG,&GPIO_InitStructure);


    GPIO_InitStructure.GPIO_Pin = ADC_DB14| ADC_DB15;
    GPIO_Init(GPIOB,&GPIO_InitStructure);


    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;


    GPIO_InitStructure.GPIO_Pin = ADC_CONVST| ADC_RANGE;
    GPIO_Init(GPIOB,&GPIO_InitStructure);


    GPIO_InitStructure.GPIO_Pin  =  ADC_OS0|  ADC_OS1|  ADC_OS2|  ADC_STBY|
ADC_BYTE_SEL;
    GPIO_Init(GPIOE,&GPIO_InitStructure);


    GPIO_InitStructure.GPIO_Pin = ADC_RESET| ADC_RD| ADC_CS;
    GPIO_Init(GPIOG,&GPIO_InitStructure);
}
```

```
void ADC_Config(void){
    delay_us(1);
    ADC_Reset();
    ADC_STBY_NORMAL();
    ADC_RANGE_5V();
    ADC_PRL();
    ADC_Semples(ADC_SEMPLE_200K);
    ADC_Reset();
}


void ADC_Semples(u8 ADC_Semp_Select){
    switch(ADC_Semp_Select){
        case ADC_SEMPLE_200K:
            ADC_OS2_LOW();
            ADC_OS1_LOW();
            ADC_OS0_LOW();
            break;
        case ADC_SEMPLE_100K:
            ADC_OS2_LOW();
            ADC_OS1_LOW();
            ADC_OS0_HIGH();
            break;
        case ADC_SEMPLE_50K:
            ADC_OS2_LOW();
            ADC_OS1_HIGH();
            ADC_OS0_LOW();
            break;
        case ADC_SEMPLE_25K:
            ADC_OS2_LOW();
            ADC_OS1_HIGH();
            ADC_OS0_HIGH();
            break;
        case ADC_SEMPLE_12K5:
            ADC_OS2_HIGH();
            ADC_OS1_LOW();
            ADC_OS0_LOW();
            break;
        case ADC_SEMPLE_6K25:
            ADC_OS2_HIGH();
            ADC_OS1_LOW();
            ADC_OS0_HIGH();
            break;
        case ADC_SEMPLE_3K125:
            ADC_OS2_HIGH();
            ADC_OS1_HIGH();
            ADC_OS0_LOW();
            break;
```

```
        default:
            break;
    }
}


int16_t ADC_Data_Pro(void){
    int16_t ADC_Data_Arr = 0;
    ADC_Data_Arr = GPIO_ReadOutputDataBit(GPIOA, ADC_DB0)|
                                          GPIO_ReadOutputDataBit(GPIOA,  ADC_DB1)
<< 1|
                                          GPIO_ReadOutputDataBit(GPIOA,  ADC_DB2)
<< 2|
                                          GPIO_ReadOutputDataBit(GPIOF,  ADC_DB3)
<< 3|
                                          GPIO_ReadOutputDataBit(GPIOF,  ADC_DB4)
<< 4|
                                          GPIO_ReadOutputDataBit(GPIOF,  ADC_DB5)
<< 5|
                                          GPIO_ReadOutputDataBit(GPIOF,  ADC_DB6)
<< 6|
                                          GPIO_ReadOutputDataBit(GPIOC,  ADC_DB7)
<< 7|
                                          GPIO_ReadOutputDataBit(GPIOC,  ADC_DB8)
<< 8|
                                          GPIO_ReadOutputDataBit(GPIOC,  ADC_DB9)
<< 9|
                                          GPIO_ReadOutputDataBit(GPIOC, ADC_DB10)
<< 10|
                                          GPIO_ReadOutputDataBit(GPIOG, ADC_DB11)
<< 11|
                                          GPIO_ReadOutputDataBit(GPIOG, ADC_DB12)
<< 12|
                                          GPIO_ReadOutputDataBit(GPIOG, ADC_DB13)
<< 13|
                                          GPIO_ReadOutputDataBit(GPIOB, ADC_DB14)
<< 14|
                                          GPIO_ReadOutputDataBit(GPIOB, ADC_DB15)
<< 15;

    return ADC_Data_Arr;
}


void ADC_Reset(void){

        ADC_RST_LOW();

    ADC_RST_HIGH();
        delay_us(1);
```

```
    ADC_RST_LOW();
}


void ADC_Start_Convst(void){

        ADC_CONVST_HIGH();
        delay_us(1);

        ADC_CONVST_LOW();
        delay_us(1);

    ADC_CONVST_HIGH();
}


void ADC_End_Convst(void){
        ADC_CONVST_HIGH();
        delay_us(1);

        ADC_CONVST_LOW();
        delay_us(1);

    ADC_CONVST_HIGH();
}


void ADC_Read(float *Data){
    uint8_t i = 0;


    ADC_Reset();




    ADC_Start_Convst();
    delay_us(1);

    while(ADC_BUSY_STATE){}

    delay_us(1);




    while(!ADC_FRST_STATE){}

    for(i=0; i<8; i++){
```

```
        Data[i] = ADC_Data_Pro();
        Data[i] = Data[i] *5.0f /32768.0f;




    }



    ADC_End_Convst();

}


#include "VM_sram.h"
#include "VM_usart.h"



#define Bank1_SRAM3_ADDR    ((u32)(0x68000000))

u32 Sram_Buf[50000] __attribute__((at(0X68000000)));



void FSMC_SRAM_Init(void){
   GPIO_InitTypeDef  GPIO_InitStructure;
   FSMC_NORSRAMInitTypeDef  FSMC_NORSRAMInitStructure;
  FSMC_NORSRAMTimingInitTypeDef  readWriteTiming;

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB|RCC_AHB1Periph_GPIOD|RCC_AHB1Pe
riph_GPIOE|RCC_AHB1Periph_GPIOF|RCC_AHB1Periph_GPIOG, ENABLE);
  RCC_AHB3PeriphClockCmd(RCC_AHB3Periph_FSMC,ENABLE);


   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_15;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
  GPIO_Init(GPIOB, &GPIO_InitStructure);

   GPIO_InitStructure.GPIO_Pin = (3<<0)|(3<<4)|(0XFF<<8);
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
  GPIO_Init(GPIOD, &GPIO_InitStructure);
```

```
  GPIO_InitStructure.GPIO_Pin = (3<<0)|(0X1FF<<7);
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
  GPIO_Init(GPIOE, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = (0X3F<<0)|(0XF<<12);
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
  GPIO_Init(GPIOF, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin =(0X3F<<0)| GPIO_Pin_10;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
  GPIO_Init(GPIOG, &GPIO_InitStructure);


  GPIO_PinAFConfig(GPIOD,GPIO_PinSource0,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOD,GPIO_PinSource1,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOD,GPIO_PinSource4,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOD,GPIO_PinSource5,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOD,GPIO_PinSource8,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOD,GPIO_PinSource9,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOD,GPIO_PinSource10,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOD,GPIO_PinSource11,GPIO_AF_FSMC);
    GPIO_PinAFConfig(GPIOD,GPIO_PinSource12,GPIO_AF_FSMC);
GPIO_PinAFConfig(GPIOD,GPIO_PinSource13,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOD,GPIO_PinSource14,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOD,GPIO_PinSource15,GPIO_AF_FSMC);

  GPIO_PinAFConfig(GPIOE,GPIO_PinSource0,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOE,GPIO_PinSource1,GPIO_AF_FSMC);
    GPIO_PinAFConfig(GPIOE,GPIO_PinSource7,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOE,GPIO_PinSource8,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOE,GPIO_PinSource9,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOE,GPIO_PinSource10,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOE,GPIO_PinSource11,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOE,GPIO_PinSource12,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOE,GPIO_PinSource13,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOE,GPIO_PinSource14,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOE,GPIO_PinSource15,GPIO_AF_FSMC);

  GPIO_PinAFConfig(GPIOF,GPIO_PinSource0,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOF,GPIO_PinSource1,GPIO_AF_FSMC);
```

```
  GPIO_PinAFConfig(GPIOF,GPIO_PinSource2,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOF,GPIO_PinSource3,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOF,GPIO_PinSource4,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOF,GPIO_PinSource5,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOF,GPIO_PinSource12,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOF,GPIO_PinSource13,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOF,GPIO_PinSource14,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOF,GPIO_PinSource15,GPIO_AF_FSMC);

  GPIO_PinAFConfig(GPIOG,GPIO_PinSource0,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOG,GPIO_PinSource1,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOG,GPIO_PinSource2,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOG,GPIO_PinSource3,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOG,GPIO_PinSource4,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOG,GPIO_PinSource5,GPIO_AF_FSMC);
  GPIO_PinAFConfig(GPIOG,GPIO_PinSource10,GPIO_AF_FSMC);


    readWriteTiming.FSMC_AddressSetupTime = 0x00;
  readWriteTiming.FSMC_AddressHoldTime = 0x00;
  readWriteTiming.FSMC_DataSetupTime = 0x08;
  readWriteTiming.FSMC_BusTurnAroundDuration = 0x00;
  readWriteTiming.FSMC_CLKDivision = 0x00;
  readWriteTiming.FSMC_DataLatency = 0x00;
  readWriteTiming.FSMC_AccessMode = FSMC_AccessMode_A;



  FSMC_NORSRAMInitStructure.FSMC_Bank = FSMC_Bank1_NORSRAM3;
  FSMC_NORSRAMInitStructure.FSMC_DataAddressMux = FSMC_DataAddressMux_Disable;
  FSMC_NORSRAMInitStructure.FSMC_MemoryType =FSMC_MemoryType_SRAM;
  FSMC_NORSRAMInitStructure.FSMC_MemoryDataWidth = FSMC_MemoryDataWidth_16b;
  FSMC_NORSRAMInitStructure.FSMC_BurstAccessMode =FSMC_BurstAccessMode_Disable;
                    FSMC_NORSRAMInitStructure.FSMC_WaitSignalPolarity          =
FSMC_WaitSignalPolarity_Low;
     FSMC_NORSRAMInitStructure.FSMC_AsynchronousWait=FSMC_AsynchronousWait_Disab
le;
  FSMC_NORSRAMInitStructure.FSMC_WrapMode = FSMC_WrapMode_Disable;
                      FSMC_NORSRAMInitStructure.FSMC_WaitSignalActive           =
FSMC_WaitSignalActive_BeforeWaitState;
  FSMC_NORSRAMInitStructure.FSMC_WriteOperation = FSMC_WriteOperation_Enable;
  FSMC_NORSRAMInitStructure.FSMC_WaitSignal = FSMC_WaitSignal_Disable;
  FSMC_NORSRAMInitStructure.FSMC_ExtendedMode = FSMC_ExtendedMode_Disable;
  FSMC_NORSRAMInitStructure.FSMC_WriteBurst = FSMC_WriteBurst_Disable;
  FSMC_NORSRAMInitStructure.FSMC_ReadWriteTimingStruct = &readWriteTiming;
  FSMC_NORSRAMInitStructure.FSMC_WriteTimingStruct = &readWriteTiming;

  FSMC_NORSRAMInit(&FSMC_NORSRAMInitStructure);

    FSMC_NORSRAMCmd(FSMC_Bank1_NORSRAM3, ENABLE);
```

```
}



void FSMC_SRAM_WriteBuffer(u8* pBuffer,u32 WriteAddr,u32 n){
   for(;n!=0;n--)
   {
      *(vu8*)(Bank1_SRAM3_ADDR+WriteAddr)=*pBuffer;
      WriteAddr++;
      pBuffer++;
   }
}



void FSMC_SRAM_ReadBuffer(u8* pBuffer,u32 ReadAddr,u32 n){
   for(;n!=0;n--)
   {
      *pBuffer++=*(vu8*)(Bank1_SRAM3_ADDR+ReadAddr);
      ReadAddr++;
   }
}


#include "sys.h"
#include "VM_usart.h"

#pragma import(__use_no_semihosting)

struct __FILE {
   int handle;
};

FILE __stdout;

void _sys_exit(int x) {
   x = x;
}


int fputc(int ch, FILE *f){
   while((USART3->SR&0X40)==0);
   USART3->DR = (u8) ch;
   return ch;
}
```

```
u8 USART_RX_BUF[USART_REC_LEN];

u16 USART_RX_STA=0;

u8 USART3_RX_BUF[USART3_REC_LEN];

u16 USART3_RX_STA=0;

void USART3_Enable_Init(void){

   GPIO_InitTypeDef GPIO_InitStructure;

   RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD,ENABLE);

   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;

   GPIO_Init(GPIOD,&GPIO_InitStructure);

   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;

   GPIO_Init(GPIOC,&GPIO_InitStructure);
}

void USART3_Enable_Set(void){
   GPIO_WriteBit(GPIOD, GPIO_Pin_3, Bit_SET);
   GPIO_WriteBit(GPIOC, GPIO_Pin_12, Bit_SET);
}

void USART3_Enable_Reset(void){
   GPIO_WriteBit(GPIOD, GPIO_Pin_3, Bit_RESET);
   GPIO_WriteBit(GPIOC, GPIO_Pin_12, Bit_RESET);
}

void USART1_Init(u32 bound){

  GPIO_InitTypeDef GPIO_InitStructure;
   USART_InitTypeDef USART_InitStructure;
   NVIC_InitTypeDef NVIC_InitStructure;

   RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB,ENABLE);
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);


   GPIO_PinAFConfig(GPIOB,GPIO_PinSource6,GPIO_AF_USART1);
```

```
    GPIO_PinAFConfig(GPIOB,GPIO_PinSource7,GPIO_AF_USART1);


  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init(GPIOB,&GPIO_InitStructure);


    USART_InitStructure.USART_BaudRate = bound;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
                            USART_InitStructure.USART_HardwareFlowControl       =
USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
  USART_Init(USART1, &USART_InitStructure);

  USART_Cmd(USART1, ENABLE);



    USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);


  NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=3;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority =3;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

}

void USART3_Init(u32 bound){
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC,ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3,ENABLE);


    GPIO_PinAFConfig(GPIOC,GPIO_PinSource10,GPIO_AF_USART3);
    GPIO_PinAFConfig(GPIOC,GPIO_PinSource11,GPIO_AF_USART3);


    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10|GPIO_Pin_11 ;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
```

```
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init(GPIOC,&GPIO_InitStructure);

  NVIC_InitStructure.NVIC_IRQChannel = USART3_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=2 ;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 3;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    USART_InitStructure.USART_BaudRate = bound;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
                            USART_InitStructure.USART_HardwareFlowControl       =
USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
  USART_Init(USART3, &USART_InitStructure);

  USART_ITConfig(USART3, USART_IT_RXNE, ENABLE);
    USART_Cmd(USART3, ENABLE);

}

void USART1_IRQHandler(void){
    u8 Res;
    if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
    {
        Res =USART_ReceiveData(USART1);

        if((USART_RX_STA&0x8000)==0)
        {
            if(USART_RX_STA&0x4000)
            {
                if(Res!=0x0a)USART_RX_STA=0;
                else USART_RX_STA|=0x8000;
            }
            else
            {
                if(Res==0x0d)USART_RX_STA|=0x4000;
                else
                {
                    USART_RX_BUF[USART_RX_STA&0X3FFF]=Res ;
                    USART_RX_STA++;
                    if(USART_RX_STA>(USART_REC_LEN-1))USART_RX_STA=0;
                }
            }
        }
```

```
  }
}

void USART3_IRQHandler(void){
   u8 Res;
   if(USART_GetITStatus(USART3, USART_IT_RXNE) != RESET)
   {
      Res =USART_ReceiveData(USART3);
      if((USART3_RX_STA&0x8000)==0)
      {
         if(USART3_RX_STA&0x4000)
         {
            if(Res!=0x0a)USART3_RX_STA=0;
            else USART3_RX_STA|=0x8000;
         }
         else
         {
            if(Res==0x0d)USART3_RX_STA|=0x4000;
            else
            {
               USART3_RX_BUF[USART3_RX_STA&0X3FFF]=Res ;
               USART3_RX_STA++;
               if(USART3_RX_STA>(USART3_REC_LEN-1))USART3_RX_STA=0;
            }
         }
      }
  }
}
```

```
import usb.core
import usb.util
import numpy as np
import time

VID = 0xABAB
PID = 0xCDCD


def Find_usb():
    dev = usb.core.find(idVendor=VID, idProduct=PID)

    if not dev:
        print("Could not find USB :(")
        exit(1)
    else:
        print("Yeeha! Found a usb device!")

    dev.reset()
```

```
    if dev.is_kernel_driver_active(0):
        dev.detach_kernel_driver(0)

    dev.set_configuration()

    return dev


def Read(dev, data_size, wait_time):

    data_dict = {}


    for i in range(8):
        star = time.time()
        raw_data = dev.read(0x81, data_size, wait_time)
        end = time.time()
        index = raw_data[0]
        print(index)
        if index in data_dict:
            raw_data = raw_data[1:data_size]
            data_dict[index].append(list(raw_data))
        else:
            raw_data = raw_data[1:data_size]
            data_dict[index] = [list(raw_data)]

        print("Timer", i, ":", end - star)

    num_rows = 8
    num_cols = data_size - 1
    data_array = np.zeros((num_rows, num_cols), dtype=np.float32)

    for index in data_dict:
        data_list = data_dict[index]
        row_num = len(data_list)
        data_array[index:index + row_num] = np.array(data_list)

    return data_array


def Write(dev):
    w = 0
    while w == 0:
        w = dev.write(0x01, "hello usb")
        print(w)
```

```
from paho.mqtt import client as mqtt_client
import SQLite



mqtt_disconnect_flag = False

mqtt_connect_flag = False



def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print('rc:', rc)
        print("Connected to MQTT Broker!")
    else:
        print('rc:', rc)
        print("Failed to connect, return code %d\n", rc)



def on_disconnect(client, userdata, rc):

    from main import broker, port, database_name, excel_name, MQTTData
    print("Disconnected from MQTT Broker")
    if rc == 0:
        print("Normal disconnection")
        client.connect(broker, port)
    else:
        print("Abnormal disconnection")
        SQLite.build_database(database_name, excel_name)
        SQLite.save_database(database_name, excel_name, MQTTData)




def connect_mqtt(broker, port, client_id, username, password):
    client = mqtt_client.Client(client_id)
    client.username_pw_set(username, password)
    client.on_connect = on_connect
    client.on_disconnect = on_disconnect
    client.connect(broker, port)
    return client


def senddata(client, topic, msg):
    result = client.publish(topic, msg)

    status = result[0]
```

```
    if status == 0:
        return 1
    else:
        return 0

import sqlite3


def build_database(database_name, excel_name):

    conn = sqlite3.connect(database_name)


    conn.execute(f'''CREATE TABLE IF NOT EXISTS {excel_name} (
                 id INTEGER PRIMARY KEY,
                 myarray TEXT);''')
    print("数据库创建成功")



def save_database(database_name, excel_name, myarray):

    conn = sqlite3.connect(database_name)




    conn.execute(f"INSERT INTO {excel_name} (myarray) VALUES (?)", (myarray,))


    conn.commit()

    print("数据写入成功")


    conn.close()



def read_database(database_name, excel_name):

    conn = sqlite3.connect(database_name)


    cursor = conn.execute(f"SELECT myarray from {excel_name}")
    myarray_str = cursor.fetchone()[0]
    print(len(myarray_str))
    myarray1 = eval(myarray_str)
    print(len(myarray1))
```

```
    conn.close()
import paho.mqtt.client as mqtt
from src.config import MqttConf
from logging import Logger
from queue import Queue
from threading import Thread
from threading import Event
from src.payload import Flow
import json
import sqlite3
import random
from threading import Lock


mutex = Lock()


class Mqtt:
    def __init__(self, cfg: MqttConf, log: Logger):
        self.broker = cfg.broker
        self.port = cfg.port
        client_id = f'python-mqtt-{random.randint(0, 1000)}'
        self.client_id = client_id
        self.username = cfg.username
        self.password = cfg.password
        self.keepalive = cfg.keepalive
        self.device_id = cfg.device_id

        self.log = log

        self.connected = False
        client              =              mqtt.Client(client_id=self.client_id,
reconnect_on_failure=True)
        client.on_connect = self.on_connect
        client.on_message = self.on_message
        client.on_disconnect = self.on_disconnect
        client.on_publish = self.on_publish
        client.username_pw_set(username=self.username, password=self.password)
        client.reconnect_delay_set(min_delay=1, max_delay=120)

        self.client = client

    def Connect(self) -> int:
        err     =     self.client.connect(host=self.broker,     port=self.port,
keepalive=self.keepalive)
        match err:
            case 0:
                self.log.info("Connection broker successful!")
                self.connected = True
```

```
            case 1:
                self.log.error("Connection broker refused - incorrect protocol
version!")
                return err
            case 2:
                self.log.error("Connection broker refused - invalid client
identifier!")
                return err
            case 3:
                self.log.error("Connection broker refused - server unavailable!")
                return err
            case 4:
                self.log.error("Connection broker refused - bad username or
password!")
                return err
            case 5:
                self.log.error("Connection broker refused - not authorised 6-255:
Currently unused!")
                return err

        return err

    def Disconnect(self):
        pass


    def Start(self, queue: Queue, event: Event, cfg: MqttConf):
        t1 = Thread(target=self.worker, args=(queue, event, cfg))
        t2 = Thread(target=self.worker, args=(queue, event, cfg))
        t1.start()
        t2.start()
        t1.join()
        t2.join()


    def worker(self, queue: Queue, event: Event, cfg: MqttConf):

        with sqlite3.connect(cfg.dbpath) as conn:
            cur = conn.cursor()
        conn.execute('''CREATE TABLE IF NOT EXISTS Data (
                                                id INTEGER PRIMARY KEY,
                                                myarray TEXT);''')

        while True:

            if event.is_set():
                break


            elif not queue.empty():
```

```
                sample_data = queue.get()
                flow = Flow()

                flow.edge_compute(sample_data)

                payload = json.dumps(flow, default=lambda o: o.__dict__)
                if self.connected:
                    self.connected = False

                    device_no = 1
                    data = json.loads(payload)
                    ch_id = data["ch"]
                    topic = f'{self.device_id}/{device_no}/{ch_id}/multivalue'
                    self.client.publish(topic, payload=payload)

                else:
                    self.log.info("Save data to database")
                    mutex.acquire()

                    cur.execute("INSERT INTO Data (myarray) VALUES (?)",
(payload,))
                    conn.commit()

                    mutex.release()


            elif self.connected:
                cur.execute("SELECT * FROM Data")
                rows = cur.fetchall()
                if len(rows) > 0:
                    for row in rows:

                        payload = row[1]
                        self.log.info("Transmit data to mqtt")

                        device_no = 1
                        data = json.loads(payload)
                        ch_id = data["ch"]
                        topic                                                =
f'{self.device_id}/{device_no}/{ch_id}/multivalue'
                        self.client.publish(topic, payload=payload)

                        cur.execute("DELETE FROM Data WHERE id=?", (row[0],))
                        conn.commit()
            if not self.connected:
                try:
                    self.Connect()
                except Exception as e:
                    self.log.error(f"连接失败: {e}")
                    continue
```

```
        cur.close()

        conn.close()


    def on_connect(self, client, userdata, flags, rc):
        if rc == 0:
            self.connected = True
        self.log.info("Connected with result code " + str(rc))

        self.client.subscribe(topic="/device/info", qos=0)
        self.client.subscribe(topic="/device/config", qos=0)
        self.client.subscribe(topic="/device/sysReset", qos=0)

        self.client.subscribe(topic=f'{self.device_id}/+/+/trigger', qos=0)




    def on_disconnect(self, client, userdata, rc):
        if rc == 0:
            self.log.info('MQTT 服务连接已关闭')
            return
        elif 1 <= rc <= 5:
            self.log.error(f'意外断开：{mqtt.connack_string(rc)}，正在重连 ……
')
            self.client.reconnect()
        else:
            self.log.error(f'未知错误码：{rc}，MQTT 服务连接关闭')
        self.connected = False


    def on_message(self, client, userdata, msg: mqtt.MQTTMessage):
        self.log.info(msg.topic + " " + str(msg.payload))
        match msg.topic:
            case "/device/info":
                self.deviceInfo(msg.payload)
            case "/device/config":
                self.deviceConfig(msg.payload)
            case "/device/sysReset":
                self.sysReset(msg.payload)


    def on_publish(self, client, userdata, flags):
        self.connected = True



    def deviceInfo(self, payload: bytes | bytearray):
```

```
        pass


    def deviceConfig(self, payload: bytes | bytearray):
        pass


    def sysReset(self, payload: bytes | bytearray):
        pass


    def publish(self, topic: str, payload: any):
        self.client.publish(topic=topic, payload=payload)

import daemon
import logging
import logging.handlers
from threading import Thread
from threading import Event
from queue import Queue
from src.usb import Usb
from src.mqtt import Mqtt
from src.config import Config
import signal
from daemon.pidfile import PIDLockFile


class App():
    def __init__(self):

        self.cfg = Config()

    def run(self):

        signal.signal(signal.SIGINT, self.exit)
        signal.signal(signal.SIGTERM, self.exit)

        signal.signal(signal.SIGHUP, self.exit)


        log = logging.getLogger('gateway')
        log.setLevel(logging.DEBUG)


        console = logging.StreamHandler()
        console.setLevel(logging.DEBUG)


        fh     =     logging.handlers.RotatingFileHandler(self.cfg.Sys.logpath,
maxBytes=10000000,
```

```
                                                    backupCount=5)
        fh.setLevel(logging.DEBUG)


        formatter = logging.Formatter(u'%(asctime)s [%(levelname)s] %(message)s')
        console.setFormatter(formatter)
        fh.setFormatter(formatter)


        log.addHandler(console)
        log.addHandler(fh)


        q = Queue()

        self.event = Event()


        usb = Usb(self.cfg.Usb, log)
        err = usb.Connect()
        if err != 0:
            exit(err)

        usb_thread = Thread(target=usb.Start, args=(q, self.event))


        mqtt = Mqtt(self.cfg.Mqtt, log)
        err = mqtt.Connect()


        if err != 0:
            exit(err)

        mqtt_thread    =    Thread(target=mqtt.Start,    args=(q,    self.event,
self.cfg.Mqtt))

        usb_thread.start()
        mqtt_thread.start()

        usb_thread.join()
        mqtt_thread.join()

        log.info("program exit!")
        usb.Disconnect()
        mqtt.Disconnect()

    def exit(self, signum, frame):
        self.event.set()
```

```
app = App()

pidfile = PIDLockFile(app.cfg.Sys.pidpath)


if pidfile.is_locked():
    pidfile.break_lock()

with daemon.DaemonContext(pidfile=pidfile):
    app.run()
```