目　录

主函数

```
#include "main.h"
#include "rs485.h"
#include "PHMeter.h"
#include "DissolvedOxygenMeter.h"
#include "string.h"
/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
//static GPIO_InitTypeDef    GPIO_InitStruct;
  I2C_HandleTypeDef I2cHandle;
  __IO uint32_t UserButton1Status = 0;
/* Private function prototypes -----------------------------------------------*/
static void SystemClock_Config(void);
static void Error_Handler(void);
void CMDSwitch(uint8_t roundCnt);
void DisplaySwitch(uint8_t roundCnt);
/* Private functions ---------------------------------------------------------*/

/* Buffer used for transmission */
extern uint8_t aTxBuffer[];

/* Buffer used for reception */
extern uint8_t aRxBuffer[];
extern uint8_t aRxBufferBackUp[];


int main(void)
{
      uint8_t roundCnt=0x00;
   HAL_Init();
   /* Configure the system clock to 48 MHz */
   SystemClock_Config();
   RS485_Init(9600);
      OLED_Init();
      BSP_LED_Init(LED1);
      BSP_LED_Init(LED2);
      BSP_LED_Init(LED3);
      OLED_Clear();
      DOMeterDisplay();
      RS485_Receive_Data(aRxBuffer,RXBUFFERSIZE);
```

```
  while (1)
  {
        CMDSwitch(roundCnt);
        DisplaySwitch(roundCnt);
        RS485_Check();
        PHMeterCheck();
        DOMeterCheck();
    HAL_GPIO_TogglePin(LED2_GPIO_PORT, LED2_PIN);
    /* Insert delay 100 ms */
    HAL_Delay(100);
    roundCnt++;
  }
}
static void SystemClock_Config(void)
{
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_OscInitTypeDef RCC_OscInitStruct;

  /* No HSE Oscillator on Nucleo, Activate PLL with HSI/2 as source */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_NONE;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PREDIV = RCC_PREDIV_DIV1;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL12;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct)!= HAL_OK)
  {
    Error_Handler();
  }

  /* Select PLL as system clock source and configure the HCLK, PCLK1 clocks dividers */
  RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK |
RCC_CLOCKTYPE_PCLK1);
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1)!= HAL_OK)
  {
    Error_Handler();
  }
}
/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
```

```
  */
static void Error_Handler(void)
{
  /* User may add here some code to deal with this error */
  while(1)
  {
  }
}

void CMDSwitch(uint8_t roundCnt){
      uint8_t cntPerRound=0xff;
      if(roundCnt%cntPerRound==0){
                  DOMeterRequestData();
            HAL_Delay(8000);
      }else if(roundCnt%cntPerRound==10){
            PHMeterRequestT();
            HAL_Delay(1000);
      }else if(roundCnt%cntPerRound==20){
            PHMeterRequestORP();
            HAL_Delay(1000);
      }else if(roundCnt%cntPerRound==100){
            PHMeterRequestPH();
            HAL_Delay(1000);
      }
}

void DisplaySwitch(uint8_t roundCnt){
      if(roundCnt%100==0){
            OLED_Clear();
            PHMeterDisplay();
      }else if(roundCnt%100==50){
            OLED_Clear();
            DOMeterDisplay();
      }
}
/**
  * @brief EXTI line detection callbacks
  * @param GPIO_Pin: Specifies the pins connected EXTI line
  * @retval None
  */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  if(GPIO_Pin == USER_BUTTON1_PIN)
  {
```

```
        UserButton1Status = 1;
    }
        if(GPIO_Pin == USER_BUTTON2_PIN)
    {
//        UserButton1Status = 1;
    }
}
#ifdef    USE_FULL_ASSERT

/**
    * @brief    Reports the name of the source file and the source line number
    *                where the assert_param error has occurred.
    * @param    file: pointer to the source file name
    * @param    line: assert_param error line source number
    * @retval None
    */
void assert_failed(uint8_t *file, uint32_t line)
{
    /* User can add his own implementation to report the file name and line number,
        ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */

    /* Infinite loop */
    while (1)
    {
    }
}
#endif
```

485 驱动

```
#include "sys.h"
#include "rs485.h"
#include "delay.h"
#include "string.h"
#include "PHMeter.h"
#include "DissolvedOxygenMeter.h"
UART_HandleTypeDef UartHandle;
__IO ITStatus UartReady = RESET;
__IO uint32_t UserButtonStatus = 0;    /* set to 1 after User Button interrupt    */
static void Error_Handler(void);
/* Buffer used for transmission */
uint8_t aTxBuffer[TXBUFFERSIZE];

/* Buffer used for reception */
uint8_t aRxBuffer[RXBUFFERSIZE];
uint8_t allZero[TXBUFFERSIZE];
uint8_t aRxBufferBackUp[RXBUFFERSIZE];
uint8_t RS485Reg=0x00;
/**
   * @brief Init RS485.
   * @param    bound:BaudRate
   *      This parameter can be one of the following values:
   * @arg 9600 115200 etc.
   * @note None
   * @retval None
   */

void RS485_Init(u32 bound)
{
      GPIO_InitTypeDef    gpioinitstruct;
   __HAL_RCC_GPIOA_CLK_ENABLE();
      gpioinitstruct.Pin = RS485_2_RE_PIN;
   gpioinitstruct.Mode = GPIO_MODE_OUTPUT_PP;
   gpioinitstruct.Pull = GPIO_NOPULL;
   gpioinitstruct.Speed = GPIO_SPEED_FREQ_HIGH;
   HAL_GPIO_Init(RS485_2_RE_GPIO_PORT, &gpioinitstruct);
      UartHandle.Instance                = USART2;
   UartHandle.Init.BaudRate      = bound;
   UartHandle.Init.WordLength = UART_WORDLENGTH_8B;
   UartHandle.Init.StopBits      = UART_STOPBITS_1;
   UartHandle.Init.Parity          = UART_PARITY_NONE;
```

```
  UartHandle.Init.HwFlowCtl    = UART_HWCONTROL_NONE;
  UartHandle.Init.Mode         = UART_MODE_TX_RX;
  UartHandle.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if(HAL_UART_DeInit(&UartHandle) != HAL_OK)
  {
    Error_Handler();
  }
  if(HAL_UART_Init(&UartHandle) != HAL_OK)
  {
    Error_Handler();
  }
}
/**
  * @brief Send data with RS485.
    * @param    buf: first address of the data buffer to be sent
                  len: length of data to be sent. in Byte
  * @arg
  * @note None
  * @retval None
  */
void RS485_Send_Data(u8 *buf,u8 len)
{
    RS485_2_RE_HIGH();
     /* The board sends the message and expects to receive it back */

  /*##-2- Start the transmission process #####################################*/
  /* While the UART in reception process, user can transmit data through
      "aTxBuffer" buffer */
    if(HAL_UART_Transmit(&UartHandle,(uint8_t*)buf, len,1000) != HAL_OK)
  {
    Error_Handler();
  }
    /*##-3- Wait for the end of the transfer ###################################*/

  /* Reset transmission flag */
  UartReady = RESET;
  RS485_2_RE_LOW();
}
/**
  * @brief Get data received by RS485.
    * @param    buf: first address of the data buffer used to store data received
                  len: length of data read. in Byte
  * @arg
  * @note None
```

```
   * @retval None
   */
void RS485_Receive_Data(u8 *buf,u8 len)
{
      /*##-4- Put UART peripheral in reception process ############################*/
   while(HAL_UART_Receive_IT(&UartHandle, (uint8_t *)buf, len) != HAL_OK)
   {
      Error_Handler();
   }
}

void RS485_Check(void){
      uint8_t ReceiveDataAddr=0x00;
      if(RS485Reg){
            memcpy(aRxBufferBackUp,aRxBuffer,RXBUFFERSIZE);
            memcpy(aRxBuffer,allZero,RXBUFFERSIZE);
            if(RS485Reg&RS485_2_REC){
                  ReceiveDataAddr=aRxBufferBackUp[0];
                  switch (ReceiveDataAddr){
                        case PHMeterAddr:
                              memcpy(PHMeterDataBuf,aRxBufferBackUp,PHMETER_DATABUF_SIZE);
                              PHMeterReg|=PHMETER_RBUF_UPDATE;
                              break;
                        case DOMeterAddr:
                              memcpy(DOMeterDataBuf,aRxBufferBackUp,DOMETER_DATABUF_SIZE);
                              DOMeterReg|=DOMETER_RBUF_UPDATE;
                              break;
                        default: ;
                  }
                  memcpy(aRxBufferBackUp,allZero,RXBUFFERSIZE);
            }
      RS485Reg&=~RS485_2_REC;
      }
}
/**
   * @brief    Tx Transfer completed callback
   * @param    UartHandle: UART handle.
   * @note     This example shows a simple way to report end of IT Tx transfer, and
   *           you can add your own implementation.
   * @retval None
   */
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *UartHandle)
{
   /* Set transmission flag: trasfer complete*/
```

```
  UartReady = SET;
}

/**
  * @brief  Rx Transfer completed callback
  * @param  UartHandle: UART handle
  * @note   This example shows a simple way to report end of DMA Rx transfer, and
  *         you can add your own implementation.
  * @retval None
  */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *UartHandle)
{
  /* Set transmission flag: trasfer complete*/
  UartReady = SET;
    RS485Reg|=RS485_2_REC;
    RS485_Receive_Data(aRxBuffer,RXBUFFERSIZE);
}

/**
  * @brief  UART error callbacks
  * @param  UartHandle: UART handle
  * @note   This example shows a simple way to report transfer error, and you can
  *         add your own implementation.
  * @retval None
  */
 void HAL_UART_ErrorCallback(UART_HandleTypeDef *UartHandle)
{
    Error_Handler();
}


/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
static void Error_Handler(void)
{
  /* Turn LED2 on */
  BSP_LED_On(LED2);
    while(1)
    {
    /* Error if LED2 is slowly blinking (1 sec. period) */
    BSP_LED_Toggle(LED2);
```

```
      HAL_Delay(1000);
      }
}

#ifdef    USE_FULL_ASSERT

/**
   * @brief    Reports the name of the source file and the source line number
   *               where the assert_param error has occurred.
   * @param    file: pointer to the source file name
   * @param    line: assert_param error line source number
   * @retval None
   */
void assert_failed(uint8_t* file, uint32_t line)
{
   /* User can add his own implementation to report the file name and line number,
        ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */

   /* Infinite loop */
   while (1)
   {
   }
}
#endif
```

OLED 驱动

```
#include "oled.h"
#include "stdlib.h"
#include "oledfont.h"
#include "delay.h"

//[0]0 1 2 3 ... 127
//[1]0 1 2 3 ... 127
//[2]0 1 2 3 ... 127
//[3]0 1 2 3 ... 127
//[4]0 1 2 3 ... 127
//[5]0 1 2 3 ... 127
//[6]0 1 2 3 ... 127
//[7]0 1 2 3 ... 127
/**********************************************
//IIC Start
**********************************************/
/**********************************************
//IIC Start
**********************************************/
void IIC_Start(void)
{

	OLED_SCLK_Set() ;
	OLED_SDIN_Set();
	OLED_SDIN_Clr();
	OLED_SCLK_Clr();
}

/**********************************************
//IIC Stop
**********************************************/
void IIC_Stop(void)
{
OLED_SCLK_Set() ;
//	OLED_SCLK_Clr();
	OLED_SDIN_Clr();
	OLED_SDIN_Set();

}
```

```
void IIC_Wait_Ack(void)
{

	//GPIOB->CRH &= 0XFFF0FFFF;
	//GPIOB->CRH |= 0x00080000;
//	OLED_SDA = 1;
//	delay_us(1);
	//OLED_SCL = 1;
	//delay_us(50000);
/*	while(1)
	{
		if(!OLED_SDA)
		{
			//GPIOB->CRH &= 0XFFF0FFFF;
			//GPIOB->CRH |= 0x00030000;
			return;
		}
	}
*/
	OLED_SCLK_Set() ;
	OLED_SCLK_Clr();
}
/**********************************************
// IIC Write byte
**********************************************/

void Write_IIC_Byte(unsigned char IIC_Byte)
{
	unsigned char i;
	unsigned char m,da;
	da=IIC_Byte;
	OLED_SCLK_Clr();
	for(i=0;i<8;i++)
	{
			m=da;
		//	OLED_SCLK_Clr();
		m=m&0x80;
		if(m==0x80)
		{OLED_SDIN_Set();}
		else OLED_SDIN_Clr();
			da=da<<1;
		OLED_SCLK_Set();
		OLED_SCLK_Clr();
		}
```

```
}
/**********************************************
// IIC Write Command
**********************************************/
void Write_IIC_Command(unsigned char IIC_Command)
{
    IIC_Start();
    Write_IIC_Byte(0x78);            //Slave address,SA0=0
     IIC_Wait_Ack();
    Write_IIC_Byte(0x00);            //write command
     IIC_Wait_Ack();
    Write_IIC_Byte(IIC_Command);
     IIC_Wait_Ack();
    IIC_Stop();
}
/**********************************************
// IIC Write Data
**********************************************/
void Write_IIC_Data(unsigned char IIC_Data)
{
    IIC_Start();
    Write_IIC_Byte(0x78);            //D/C#=0; R/W#=0
     IIC_Wait_Ack();
    Write_IIC_Byte(0x40);            //write data
     IIC_Wait_Ack();
    Write_IIC_Byte(IIC_Data);
     IIC_Wait_Ack();
    IIC_Stop();
}
void OLED_WR_Byte(unsigned dat,unsigned cmd)
{
     if(cmd)
                {

    Write_IIC_Data(dat);

          }
     else {
    Write_IIC_Command(dat);

     }
```

```
}


/*****************************************
// fill_Picture
*****************************************/
void fill_picture(unsigned char fill_Data)
{
      unsigned char m,n;
      for(m=0;m<8;m++)
      {
            OLED_WR_Byte(0xb0+m,0);         //page0-page1
            OLED_WR_Byte(0x00,0);           //low column start address
            OLED_WR_Byte(0x10,0);           //high column start address
            for(n=0;n<128;n++)
                  {
                        OLED_WR_Byte(fill_Data,1);
                  }
      }
}


/**********************Delay****************************************/
void Delay_50ms(unsigned int Del_50ms)
{
      unsigned int m;
      for(;Del_50ms>0;Del_50ms--)
            for(m=6245;m>0;m--);
}

void Delay_1ms(unsigned int Del_1ms)
{
      unsigned char j;
      while(Del_1ms--)
      {
            for(j=0;j<123;j++);
      }
}

      void OLED_Set_Pos(unsigned char x, unsigned char y)
{     OLED_WR_Byte(0xb0+y,OLED_CMD);
      OLED_WR_Byte(((x&0xf0)>>4)|0x10,OLED_CMD);
      OLED_WR_Byte((x&0x0f),OLED_CMD);
```

```
}
void OLED_Display_On(void)
{
	OLED_WR_Byte(0X8D,OLED_CMD);
	OLED_WR_Byte(0X14,OLED_CMD);    //DCDC ON
	OLED_WR_Byte(0XAF,OLED_CMD);    //DISPLAY ON
}

void OLED_Display_Off(void)
{
	OLED_WR_Byte(0X8D,OLED_CMD);
	OLED_WR_Byte(0X10,OLED_CMD);    //DCDC OFF
	OLED_WR_Byte(0XAE,OLED_CMD);    //DISPLAY OFF
}

void OLED_Clear(void)
{
	u8 i,n;
	for(i=0;i<8;i++)
	{
		OLED_WR_Byte (0xb0+i,OLED_CMD);
		OLED_WR_Byte (0x00,OLED_CMD);
		OLED_WR_Byte (0x10,OLED_CMD);
		for(n=0;n<128;n++)OLED_WR_Byte(0,OLED_DATA);
	}
}
void OLED_On(void)
{
	u8 i,n;
	for(i=0;i<8;i++)
	{
		OLED_WR_Byte (0xb0+i,OLED_CMD);
		OLED_WR_Byte (0x00,OLED_CMD);
		OLED_WR_Byte (0x10,OLED_CMD);
		for(n=0;n<128;n++)OLED_WR_Byte(1,OLED_DATA);
	}
}
//x:0~127
//y:0~63
void OLED_ShowChar(u8 x,u8 y,u8 chr,u8 Char_Size)
{
	unsigned char c=0,i=0;
		c=chr-' ';
		if(x>Max_Column-1){x=0;y=y+2;}
```

```
        if(Char_Size ==16)
            {
            OLED_Set_Pos(x,y);
            for(i=0;i<8;i++)
            OLED_WR_Byte(F8X16[c*16+i],OLED_DATA);
            OLED_Set_Pos(x,y+1);
            for(i=0;i<8;i++)
            OLED_WR_Byte(F8X16[c*16+i+8],OLED_DATA);
            }
            else {
                OLED_Set_Pos(x,y);
                for(i=0;i<6;i++)
                OLED_WR_Byte(F6x8[c][i],OLED_DATA);

            }
}
u32 oled_pow(u8 m,u8 n)
{
    u32 result=1;
    while(n--)result*=m;
    return result;
}

void OLED_ShowNum(u8 x,u8 y,u32 num,u8 len,u8 size2)
{
    u8 t,temp;
    u8 enshow=0;
    for(t=0;t<len;t++)
    {
        temp=(num/oled_pow(10,len-t-1))%10;
        if(enshow==0&&t<(len-1))
        {
            if(temp==0)
            {
                OLED_ShowChar(x+(size2/2)*t,y,' ',size2);
                continue;
            }else enshow=1;

        }
        OLED_ShowChar(x+(size2/2)*t,y,temp+'0',size2);
    }
}
void OLED_ShowString(u8 x,u8 y,u8 *chr,u8 Char_Size)
{
```

```
	unsigned char j=0;
	while (chr[j]!='\0')
	{		OLED_ShowChar(x,y,chr[j],Char_Size);
			x+=8;
		if(x>120){x=0;y+=2;}
			j++;
	}
}
void OLED_ShowCHinese(u8 x,u8 y,u8 no)
{
	u8 t,adder=0;
	OLED_Set_Pos(x,y);
	for(t=0;t<16;t++)
		{
				OLED_WR_Byte(Hzk[2*no][t],OLED_DATA);
				adder+=1;
	 }
		OLED_Set_Pos(x,y+1);
	for(t=0;t<16;t++)
			{
				OLED_WR_Byte(Hzk[2*no+1][t],OLED_DATA);
				adder+=1;
	   }
}
void OLED_DrawBMP(unsigned char x0, unsigned char y0,unsigned char x1, unsigned char
y1,unsigned char BMP[])
{
  unsigned int j=0;
  unsigned char x,y;

    if(y1%8==0) y=y1/8;
    else y=y1/8+1;
	for(y=y0;y<y1;y++)
	{
		OLED_Set_Pos(x0,y);
	for(x=x0;x<x1;x++)
		{
			OLED_WR_Byte(BMP[j++],OLED_DATA);
		}
	}
}


void OLED_Init(void)
```

```
{

      GPIO_InitTypeDef    GPIO_InitStruct;
      I2Cx_SDA_GPIO_CLK_ENABLE();
       /* -2- Configure IOs in output push-pull mode to drive external LEDs */
   GPIO_InitStruct.Pin     = I2Cx_SCL_PIN|I2Cx_SDA_PIN;
   GPIO_InitStruct.Mode    = GPIO_MODE_OUTPUT_PP;
   GPIO_InitStruct.Pull    = GPIO_PULLUP;
   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
   HAL_GPIO_Init(I2Cx_SCL_GPIO_PORT, &GPIO_InitStruct);
      OLED_SCLK_Set();
      OLED_SDIN_Set();

delay_ms(800);
OLED_WR_Byte(0xAE,OLED_CMD);//--display off
      OLED_WR_Byte(0x00,OLED_CMD);//---set low column address
      OLED_WR_Byte(0x10,OLED_CMD);//---set high column address
      OLED_WR_Byte(0x40,OLED_CMD);//--set start line address
      OLED_WR_Byte(0xB0,OLED_CMD);//--set page address
      OLED_WR_Byte(0x81,OLED_CMD); // contract control
      OLED_WR_Byte(0xFF,OLED_CMD);//--128
      OLED_WR_Byte(0xA1,OLED_CMD);//set segment remap
      OLED_WR_Byte(0xA6,OLED_CMD);//--normal / reverse
      OLED_WR_Byte(0xA8,OLED_CMD);//--set multiplex ratio(1 to 64)
      OLED_WR_Byte(0x3F,OLED_CMD);//--1/32 duty
      OLED_WR_Byte(0xC8,OLED_CMD);//Com scan direction
      OLED_WR_Byte(0xD3,OLED_CMD);//-set display offset
      OLED_WR_Byte(0x00,OLED_CMD);//

      OLED_WR_Byte(0xD5,OLED_CMD);//set osc division
      OLED_WR_Byte(0x80,OLED_CMD);//

      OLED_WR_Byte(0xD8,OLED_CMD);//set area color mode off
      OLED_WR_Byte(0x05,OLED_CMD);//

      OLED_WR_Byte(0xD9,OLED_CMD);//Set Pre-Charge Period
      OLED_WR_Byte(0xF1,OLED_CMD);//

      OLED_WR_Byte(0xDA,OLED_CMD);//set com pin configuartion
      OLED_WR_Byte(0x12,OLED_CMD);//

      OLED_WR_Byte(0xDB,OLED_CMD);//set Vcomh
      OLED_WR_Byte(0x30,OLED_CMD);//
```

```
    OLED_WR_Byte(0x8D,OLED_CMD);//set charge pump enable
    OLED_WR_Byte(0x14,OLED_CMD);//

    OLED_WR_Byte(0xAF,OLED_CMD);//--turn on oled panel
}
```

CRC16 校验

```
#include "CRC16.h"
static uint8_t auchCRCHi[] = {
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40
} ;
static int8_t auchCRCLo[] = {
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06,
0x07, 0xC7, 0x05, 0xC5, 0xC4, 0x04, 0xCC, 0x0C, 0x0D, 0xCD,
0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A,
0x1E, 0xDE, 0xDF, 0x1F, 0xDD, 0x1D, 0x1C, 0xDC, 0x14, 0xD4,
0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3,
0xF2, 0x32, 0x36, 0xF6, 0xF7, 0x37, 0xF5, 0x35, 0x34, 0xF4,
0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29,
```

```
0xEB, 0x2B, 0x2A, 0xEA, 0xEE, 0x2E, 0x2F, 0xEF, 0x2D, 0xED,
0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60,
0x61, 0xA1, 0x63, 0xA3, 0xA2, 0x62, 0x66, 0xA6, 0xA7, 0x67,
0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68,
0x78, 0xB8, 0xB9, 0x79, 0xBB, 0x7B, 0x7A, 0xBA, 0xBE, 0x7E,
0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71,
0x70, 0xB0, 0x50, 0x90, 0x91, 0x51, 0x93, 0x53, 0x52, 0x92,
0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B,
0x99, 0x59, 0x58, 0x98, 0x88, 0x48, 0x49, 0x89, 0x4B, 0x8B,
0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42,
0x43, 0x83, 0x41, 0x81, 0x80, 0x40
} ;

uint16_t CRC16(uint8_t *puchMsg, uint16_t usDataLen)
{
    uint8_t uchCRCHi = 0xFF ;
    uint8_t uchCRCLo = 0xFF ;
    unsigned uIndex ;

    while (usDataLen--)
    {
        uIndex = uchCRCHi ^ *puchMsg++ ;
        uchCRCHi = uchCRCLo ^ auchCRCHi[uIndex] ;
        uchCRCLo = auchCRCLo[uIndex] ;
    }
    return (uchCRCHi << 8 | uchCRCLo) ;
}
```

PH 表驱动

```
#include "PHMeter.h"
#include "oled.h"
uint8_t PHMeterReg=0x00;
uint8_t PHMeterCMDBuf[PHMETER_CMDBUF_SIZE];
uint8_t PHMeterDataBuf[PHMETER_DATABUF_SIZE];
uint8_t PHMeteErrFC=0x00;
uint8_t PHMeteErrCODE=0x00;
uint16_t  PHData=0x0000;
uint16_t  TData=0x0000;
uint16_t  ORPData=0x0000;
uint16_t  PHThreshold=0x0000;
uint16_t  TThreshold=0x0000;
uint16_t  ORPThreshold=0x0000;
uint16_t  PHMeterCode=0x0000;
void PHMeterErrorHandle(void){
        OLED_ShowString(PHMETER_DISP_TITAL_X,PHMETER_DISP_TITAL_Y,"PHERR",16);

}

void PHMeterTOHandle(void){
        OLED_ShowString(PHMETER_DISP_TITAL_X,PHMETER_DISP_TITAL_Y,"PHTO",16);

}

void PHMeterCRCHandle(void){
        OLED_ShowString(PHMETER_DISP_TITAL_X,PHMETER_DISP_TITAL_Y,"PHCRC",16);

}


void PHMeterDisplay(void){
        OLED_ShowString(PHMETER_DISP_TITAL_X,PHMETER_DISP_TITAL_Y,"PHMeter",16);

        OLED_ShowString(PHMETER_DISP_ITEM_X,PHMETER_DISP_ITEM_Y,"ITEM",15);
        OLED_ShowString(PHMETER_DISP_VALUE_X,PHMETER_DISP_VALUE_Y,"VAL",15);

    OLED_ShowString(PHMETER_DISP_THRESHOLD_X,PHMETER_DISP_THRESHOLD_Y,"THR",15)
;

        OLED_ShowString(PHMETER_DISP_PH_X,PHMETER_DISP_PH_Y," PH",15);
        OLED_ShowString(PHMETER_DISP_T_X,PHMETER_DISP_T_Y,"    T",15);
```

```
        OLED_ShowString(PHMETER_DISP_ORP_X,PHMETER_DISP_ORP_Y,"ORP",15);


    OLED_ShowNum(PHMETER_DISP_CODE_X,PHMETER_DISP_CODE_Y,PHMeterCode,2,16);

    OLED_ShowNum(PHMETER_DISP_PHVALUE_X,PHMETER_DISP_PHVALUE_Y,PHData/100,2,1
5);

    OLED_ShowString(PHMETER_DISP_PHVALUE_X+2*6+1,PHMETER_DISP_PHVALUE_Y,".",15);

    OLED_ShowNum(PHMETER_DISP_PHVALUE_X+2*6+6,PHMETER_DISP_PHVALUE_Y,PHData%
100/10,1,15);

    OLED_ShowNum(PHMETER_DISP_PHVALUE_X+3*6+6,PHMETER_DISP_PHVALUE_Y,PHData%
10,1,15);

    OLED_ShowNum(PHMETER_DISP_TVALUE_X,PHMETER_DISP_TVALUE_Y,TData/10,2,15);

    OLED_ShowString(PHMETER_DISP_PHVALUE_X+2*6+1,PHMETER_DISP_TVALUE_Y,".",15);

    OLED_ShowNum(PHMETER_DISP_TVALUE_X+2*6+6,PHMETER_DISP_TVALUE_Y,TData%10,1,
15);

    OLED_ShowNum(PHMETER_DISP_ORPVALUE_X,PHMETER_DISP_ORPVALUE_Y,ORPData,4,1
5);


    OLED_ShowNum(PHMETER_DISP_PHTHRESHOLD_X,PHMETER_DISP_PHTHRESHOLD_Y,PHTh
reshold,4,15);

    OLED_ShowNum(PHMETER_DISP_TTHRESHOLD_X,PHMETER_DISP_TTHRESHOLD_Y,TThresh
old,4,15);

    OLED_ShowNum(PHMETER_DISP_ORPTHRESHOLD_X,PHMETER_DISP_ORPTHRESHOLD_Y,O
RPThreshold,4,15);

}

void PHMeterRequestData(void){
    uint16_t CRC16DATA=0X0000;
    PHMeterCMDBuf[PHMETER_ADDR_OFF]          =PHMeterAddr;
    PHMeterCMDBuf[PHMETER_FC_OFF]            =PHMETER_FC;
    CRC16DATA=CRC16(PHMeterCMDBuf,PHMETER_CRCLEN_OFF);
    PHMeterCMDBuf[PHMETER_CRCLEN_OFF]  =CRC16DATA>>8;
```

```
    PHMeterCMDBuf[PHMETER_CRCLEN_OFF+1]     =CRC16DATA&0xff;
    RS485_Send_Data(PHMeterCMDBuf,PHMETER_CMDBUF_SIZE);
}

void PHMeterRequestPH(void){
/*  uint8_t Temp=0x00;
    while(PHMeterReg){
        Temp++;
        if(Temp>>PHMeterTO){
            PHMeterTOHandle();
            return ;
        }
    }*/
    PHMeterCMDBuf[PHMETER_OFFSET_OFF+1]     =PHMETER_PH_OFFSET&0xff;
    PHMeterCMDBuf[PHMETER_OFFSET_OFF]   =PHMETER_PH_OFFSET>>8;
    PHMeterCMDBuf[PHMETER_DATACNT_OFF+1]=PHMETER_PH_DATACNT&0xff;
    PHMeterCMDBuf[PHMETER_DATACNT_OFF]=PHMETER_PH_DATACNT>>8;
    PHMeterReg|=PHMETER_PH_PEND;
    PHMeterRequestData();
}

void PHMeterRequestT(void){
/*  uint8_t Temp=0x00;
    while(PHMeterReg){
        Temp++;
        if(Temp>>PHMeterTO){
            PHMeterTOHandle();
            return ;
        }
    }*/
    PHMeterCMDBuf[PHMETER_OFFSET_OFF+1]     =PHMETER_T_OFFSET&0xff;
    PHMeterCMDBuf[PHMETER_OFFSET_OFF]   =PHMETER_T_OFFSET>>8;
    PHMeterCMDBuf[PHMETER_DATACNT_OFF+1]    =PHMETER_T_DATACNT&0xff;
    PHMeterCMDBuf[PHMETER_DATACNT_OFF]=PHMETER_T_DATACNT>>8;
    PHMeterReg|=PHMETER_T_PEND;
    PHMeterRequestData();
}

void PHMeterRequestPHT(void){
/*  uint8_t Temp=0x00;
    while(PHMeterReg){
        Temp++;
        if(Temp>>PHMeterTO){
            PHMeterTOHandle();
```

```
                return ;
            }
        }*/
        PHMeterCMDBuf[PHMETER_OFFSET_OFF+1]      =PHMETER_PHT_OFFSET&0xff;
        PHMeterCMDBuf[PHMETER_OFFSET_OFF]   =PHMETER_PHT_OFFSET>>8;
        PHMeterCMDBuf[PHMETER_DATACNT_OFF+1]    =PHMETER_PHT_DATACNT&0xff;
        PHMeterCMDBuf[PHMETER_DATACNT_OFF]=(PHMETER_PHT_DATACNT)>>8;
        PHMeterReg|=PHMETER_PHT_PEND;
        PHMeterRequestData();
}

void PHMeterRequestORP(void){
/*    uint8_t Temp=0x00;
        while(PHMeterReg){
            Temp++;
            if(Temp>>PHMeterTO){
                PHMeterTOHandle();
                return ;
            }
        }*/
        PHMeterCMDBuf[PHMETER_OFFSET_OFF+1]      =PHMETER_ORP_OFFSET&0xff;
        PHMeterCMDBuf[PHMETER_OFFSET_OFF]   =PHMETER_ORP_OFFSET>>8;
        PHMeterCMDBuf[PHMETER_DATACNT_OFF+1]    =PHMETER_ORP_DATACNT&0xff;
        PHMeterCMDBuf[PHMETER_DATACNT_OFF]=PHMETER_ORP_DATACNT>>8;
        PHMeterReg|=PHMETER_ORP_PEND;
        PHMeterRequestData();
}

void PHMeterRequestORPT(void){
/*    uint8_t Temp=0x00;
        while(PHMeterReg){
            Temp++;
            if(Temp>>PHMeterTO){
                PHMeterTOHandle();
                return ;
            }
        }*/
        PHMeterCMDBuf[PHMETER_OFFSET_OFF+1]      =PHMETER_TORP_OFFSET&0xff;
        PHMeterCMDBuf[PHMETER_OFFSET_OFF]   =PHMETER_TORP_OFFSET>>8;
        PHMeterCMDBuf[PHMETER_DATACNT_OFF+1]    =PHMETER_TORP_DATACNT&0xff;
        PHMeterCMDBuf[PHMETER_DATACNT_OFF]=(PHMETER_TORP_DATACNT)>>8;
        PHMeterReg|=PHMETER_ORPT_PEND;
        PHMeterRequestData();
}
```

```
void PHMeterErrReceiveHandle(void){
    uint8_t Temp=0x00;
    uint16_t rCRC=0x0000;          //readCRC
    uint16_t cCRC=0x0000;          //calculateCRC
    Temp=PHMeterReg&(~PHMETER_RBUF_UPDATE);
    rCRC=PHMeterDataBuf[PHMETER_ERRCRC_OFF]<<8|PHMeterDataBuf[PHMETER_ERRCRC_
OFF+1];
    cCRC=CRC16(PHMeterDataBuf,PHMETER_ERRCRC_OFF);
    if(rCRC!=cCRC){
        PHMeterCRCHandle();//CRCERR
        return ;
    }
    if(PHMeterDataBuf[PHMETER_ADDR_OFF]!=PHMeterAddr){
        PHMeterErrorHandle();
    }
    PHMeteErrFC=PHMeterDataBuf[PHMETER_FC_OFF]&~0x80;
    PHMeteErrCODE=PHMeterDataBuf[PHMETER_ERRCRC_OFF];
    OLED_Clear();
    OLED_ShowString(PHMETER_DISP_TITAL_X,PHMETER_DISP_TITAL_Y,"PHMeter",16);    //L1
    OLED_ShowString(PHMETER_DISP_CODE_X,PHMETER_DISP_CODE_Y,"Error",16);    //error

    OLED_ShowString(PHMETER_DISP_PH_X,PHMETER_DISP_PH_Y,"    ErrFC",15);
    OLED_ShowString(PHMETER_DISP_T_X,PHMETER_DISP_T_Y,"ErrCODE",15);
    OLED_ShowNum(PHMETER_DISP_PHVALUE_X,PHMETER_DISP_PHVALUE_Y,PHMeteErrFC,4,
15);
    OLED_ShowNum(PHMETER_DISP_TVALUE_X,PHMETER_DISP_TVALUE_Y,PHMeteErrCODE,4,
15);
    switch (Temp){
        case PHMETER_PH_PEND:
            PHMeterReg&=(~PHMETER_RBUF_UPDATE);
            PHMeterReg&=(~PHMETER_PH_PEND);
            PHMeterRequestPH();
            break;
        case PHMETER_T_PEND:
            PHMeterReg&=(~PHMETER_RBUF_UPDATE);
            PHMeterReg&=(~PHMETER_T_PEND);
            PHMeterRequestT();
            break;
        case PHMETER_PHT_PEND:
            PHMeterReg&=(~PHMETER_RBUF_UPDATE);
            PHMeterReg&=(~PHMETER_PH_PEND);
            PHMeterReg&=(~PHMETER_T_PEND);
            PHMeterRequestPHT();
```

```
                break;
            case PHMETER_ORP_PEND:
                PHMeterReg&=(~PHMETER_RBUF_UPDATE);
                PHMeterReg&=(~PHMETER_ORP_PEND);
                PHMeterRequestORP();
                break;
            case PHMETER_ORPT_PEND:
                PHMeterReg&=(~PHMETER_RBUF_UPDATE);
                PHMeterReg&=(~PHMETER_ORP_PEND);
                PHMeterReg&=(~PHMETER_T_PEND);
                PHMeterRequestORPT();
                break;
            default:   ;
        }
        PHMeterReg=0x00;
}

void PHMeterDataReceiveHandle(void){
        uint8_t Temp=0x00;
        uint16_t CRCData=0x0000;
        Temp=PHMeterReg&(~PHMETER_RBUF_UPDATE);
        if(PHMeterDataBuf[PHMETER_ADDR_OFF]!=PHMeterAddr){
            PHMeterErrorHandle();
        }
        if(PHMeterDataBuf[PHMETER_FC_OFF]!=PHMETER_FC){
            PHMeterErrorHandle();
        }
        switch (Temp){
            case PHMETER_PH_PEND:

        CRCData=PHMeterDataBuf[PHMETER_SCRCLEN_OFF]<<8|PHMeterDataBuf[PHMETER_SCRC
LEN_OFF+1];
                if(PHMeterDataBuf[PHMETER_DATALEN_OFF]!=PHMETER_PH_DATALEN){
                    PHMeterErrorHandle();
                }
                if(CRCData!=CRC16(PHMeterDataBuf,PHMETER_SCRCLEN_OFF)){
                    PHMeterCRCHandle();
                }

        PHData=PHMeterDataBuf[PHMETER_DATA_OFF]<<8|PHMeterDataBuf[PHMETER_DATA_OF
F+1];
                PHMeterReg&=(~PHMETER_RBUF_UPDATE);
                PHMeterReg&=(~PHMETER_PH_PEND);
                break;
```

```
        case PHMETER_T_PEND:

    CRCData=PHMeterDataBuf[PHMETER_SCRCLEN_OFF]<<8|PHMeterDataBuf[PHMETER_SCRC
LEN_OFF+1];
                if(PHMeterDataBuf[PHMETER_DATALEN_OFF]!=PHMETER_T_DATALEN){
                    PHMeterErrorHandle();
                }
                if(CRCData!=CRC16(PHMeterDataBuf,PHMETER_SCRCLEN_OFF)){
                    PHMeterCRCHandle();
                }

    TData=PHMeterDataBuf[PHMETER_DATA_OFF]<<8|PHMeterDataBuf[PHMETER_DATA_OFF+
1];
                PHMeterReg&=(~PHMETER_RBUF_UPDATE);
                PHMeterReg&=(~PHMETER_T_PEND);
                break;
            case PHMETER_PHT_PEND:
    CRCData=PHMeterDataBuf[PHMETER_DCRCLEN_OFF]<<8|PHMeterDataBuf[PHMETER_DCR
CLEN_OFF+1];
                if(PHMeterDataBuf[PHMETER_DATALEN_OFF]!=PHMETER_PHT_DATALEN){
                    PHMeterErrorHandle();
                }
                if(CRCData!=CRC16(PHMeterDataBuf,PHMETER_DCRCLEN_OFF)){
                    PHMeterCRCHandle();
                }
    PHData=PHMeterDataBuf[PHMETER_DATA_OFF]<<8|PHMeterDataBuf[PHMETER_DATA_OF
F+1];
    TData=PHMeterDataBuf[PHMETER_DATA_OFF+2]<<8|PHMeterDataBuf[PHMETER_DATA_OF
F+3];
                PHMeterReg&=(~PHMETER_RBUF_UPDATE);
                PHMeterReg&=(~PHMETER_PH_PEND);
                PHMeterReg&=(~PHMETER_T_PEND);
                break;
            case PHMETER_ORP_PEND:
    CRCData=PHMeterDataBuf[PHMETER_SCRCLEN_OFF]<<8|PHMeterDataBuf[PHMETER_SCRC
LEN_OFF+1];
                if(PHMeterDataBuf[PHMETER_DATALEN_OFF]!=PHMETER_ORP_DATALEN){
                    PHMeterErrorHandle();
                }
                if(CRCData!=CRC16(PHMeterDataBuf,PHMETER_SCRCLEN_OFF)){
                    PHMeterCRCHandle();
                }

    ORPData=PHMeterDataBuf[PHMETER_DATA_OFF]<<8|PHMeterDataBuf[PHMETER_DATA_O
```

```
FF+1];
                PHMeterReg&=(~PHMETER_RBUF_UPDATE);
                PHMeterReg&=(~PHMETER_ORP_PEND);
                break;
            case PHMETER_ORPT_PEND:

    CRCData=PHMeterDataBuf[PHMETER_DCRCLEN_OFF]<<8|PHMeterDataBuf[PHMETER_DCR
CLEN_OFF+1];
                if(PHMeterDataBuf[PHMETER_DATALEN_OFF]!=PHMETER_TORP_DATALEN){
                    PHMeterErrorHandle();
                }
                if(CRCData!=CRC16(PHMeterDataBuf,PHMETER_DCRCLEN_OFF)){
                    PHMeterCRCHandle();
                }

    TData=PHMeterDataBuf[PHMETER_DATA_OFF]<<8|PHMeterDataBuf[PHMETER_DATA_OFF+
1];

    ORPData=PHMeterDataBuf[PHMETER_DATA_OFF+2]<<8|PHMeterDataBuf[PHMETER_DATA
_OFF+3];
                PHMeterReg&=(~PHMETER_RBUF_UPDATE);
                PHMeterReg&=(~PHMETER_ORP_PEND);
                PHMeterReg&=(~PHMETER_T_PEND);
                break;
            default:   ;
    }
    PHMeterReg=0x00;
}

void PHMeterReceiveHandle(void){
    if(PHMeterDataBuf[PHMETER_FC_OFF]&0x80){
        PHMeterErrReceiveHandle();
    }else{
        PHMeterDataReceiveHandle();
    }
}
void PHMeterCheck(void){
    if(PHMeterReg&PHMETER_RBUF_UPDATE){
        PHMeterReceiveHandle();
        PHMeterReg&=~PHMETER_RBUF_UPDATE;
    }

}
```

溶氧仪驱动

```
#include "DissolvedOxygenMeter.h"
#include "oled.h"
uint8_t DOMeterReg=0x00;
uint8_t DOMeterCMDBuf[DOMETER_CMDBUF_SIZE];
uint8_t DOMeterDataBuf[DOMETER_DATABUF_SIZE];
uint8_t DOMeterErrFC=0x00;
uint8_t DOMeterErrCODE=0x00;
uint16_t  DOData=0x0000;
uint16_t  DOTData=0x0000;
uint16_t  HALMData=0x0000;
uint16_t  LALMData=0x0000;
uint16_t  LTCData=0x0000;
uint16_t  STAData=0x0000;
uint16_t  setHALM=0xffff;
uint16_t  setLALM=0x0000;
uint16_t  setLTC=0x00ff;
void DOMeterErrorHandle(void){
    OLED_ShowString(DOMETER_DISP_TITAL_X,DOMETER_DISP_TITAL_Y,"DOERR",16);    //L1
}

void DOMeterCRCHandle(void){
    OLED_ShowString(DOMETER_DISP_TITAL_X,DOMETER_DISP_TITAL_Y,"DOCRC",16);    //L1
}

void DOMeterTOHandle(void){
    OLED_ShowString(DOMETER_DISP_TITAL_X,DOMETER_DISP_TITAL_Y,"DOTO",16);    //L1
}


void DOMeterDisplay(void){
        OLED_ShowString(DOMETER_DISP_TITAL_X,DOMETER_DISP_TITAL_Y,"DOMeter",16);
//L1
        OLED_ShowNum(DOMETER_DISP_CODE_X,DOMETER_DISP_CODE_Y,STAData,2,16);
    //STA

        OLED_ShowString(DOMETER_DISP_ITEM_X,DOMETER_DISP_ITEM_Y,"ITEM",15);
    //L2
        OLED_ShowString(DOMETER_DISP_MIN_X,DOMETER_DISP_MIN_Y,"MIN",15);
        OLED_ShowString(DOMETER_DISP_VALUE_X,DOMETER_DISP_VALUE_Y,"VAL",15);
        OLED_ShowString(DOMETER_DISP_MAX_X,DOMETER_DISP_MAX_Y,"MAX",15);
```

```
        OLED_ShowString(DOMETER_DISP_DO_X,DOMETER_DISP_DO_Y," DO",15);     //ITEM

        OLED_ShowString(DOMETER_DISP_T_X,DOMETER_DISP_T_Y,"    T",15);
        OLED_ShowString(DOMETER_DISP_LTC_X,DOMETER_DISP_LTC_Y,"LTC",15);


    OLED_ShowNum(DOMETER_DISP_DOMIN_X,DOMETER_DISP_DOMIN_Y,LALMData/10,2,15);
    //MIN

    OLED_ShowString(DOMETER_DISP_DOMIN_X+2*6+1,DOMETER_DISP_DOMIN_Y,".",15);

    OLED_ShowNum(DOMETER_DISP_DOMIN_X+3*6,DOMETER_DISP_DOMIN_Y,LALMData%10,
1,15);



    OLED_ShowNum(DOMETER_DISP_DOVALUE_X,DOMETER_DISP_DOVALUE_Y,DOData/1000,
1,15);      //VALUE

    OLED_ShowString(DOMETER_DISP_DOVALUE_X+6+1,DOMETER_DISP_DOVALUE_Y,".",15);

    OLED_ShowNum(DOMETER_DISP_DOVALUE_X+2*6,DOMETER_DISP_DOVALUE_Y,DOData%
1000/100,1,15);

    OLED_ShowNum(DOMETER_DISP_DOVALUE_X+3*6,DOMETER_DISP_DOVALUE_Y,DOData%
100/10,1,15);

    OLED_ShowNum(DOMETER_DISP_DOVALUE_X+4*6,DOMETER_DISP_DOVALUE_Y,DOData%
10,1,15);

    OLED_ShowNum(DOMETER_DISP_TVALUE_X,DOMETER_DISP_TVALUE_Y,DOTData/100,2,15)
;

    OLED_ShowString(DOMETER_DISP_TVALUE_X+2*6+1,DOMETER_DISP_TVALUE_Y,".",15);

    OLED_ShowNum(DOMETER_DISP_TVALUE_X+3*6,DOMETER_DISP_TVALUE_Y,DOTData%10
0/10,1,15);

    OLED_ShowNum(DOMETER_DISP_TVALUE_X+4*6,DOMETER_DISP_TVALUE_Y,DOTData/10,1,
15);

    OLED_ShowNum(DOMETER_DISP_LTCVALUE_X,DOMETER_DISP_LTCVALUE_Y,LTCData/10,2,
15);

    OLED_ShowString(DOMETER_DISP_LTCVALUE_X+2*6+1,DOMETER_DISP_LTCVALUE_Y,".",15)
```

```
;

    OLED_ShowNum(DOMETER_DISP_LTCVALUE_X+3*6,DOMETER_DISP_LTCVALUE_Y,LTCData%
10,1,15);


    OLED_ShowNum(DOMETER_DISP_DOMAX_X,DOMETER_DISP_DOMAX_Y,HALMData/10,2,1
5);  //MAX

    OLED_ShowString(DOMETER_DISP_DOMAX_X+2*6+1,DOMETER_DISP_DOMAX_Y,".",15);


    OLED_ShowNum(DOMETER_DISP_DOMAX_X+3*6,DOMETER_DISP_DOMAX_Y,HALMData%
10,1,15);
}

void DOMeterRequestData(void){
    uint16_t CRCDATA=0X0000;
    DOMeterCMDBuf[DOMETER_ADDR_OFF]          =DOMeterAddr;
    DOMeterCMDBuf[DOMETER_FC_OFF]        =DOMETER_REQ_FC;
    DOMeterCMDBuf[DOMETER_START_OFF]   =0X00;
    DOMeterCMDBuf[DOMETER_START_OFF+1]=0X00;
    DOMeterCMDBuf[DOMETER_REGCNT_OFF] =0X00;
    DOMeterCMDBuf[DOMETER_REGCNT_OFF+1]   =0X06;

    CRCDATA=CRC16(DOMeterCMDBuf,DOMETER_REQCRC_OFF);
    DOMeterCMDBuf[DOMETER_REQCRC_OFF+1]   =CRCDATA&0XFF;
    DOMeterCMDBuf[DOMETER_REQCRC_OFF] =CRCDATA>>8;
    RS485_Send_Data(DOMeterCMDBuf,DOMETER_REQCRC_OFF+2);
    DOMeterReg|=DOMETER_REQ_PEND;
}

void DOMeterWriteReg(uint16_t HALM,uint16_t LALM,uint16_t LTC){
    uint16_t CRCDATA=0X0000;
    DOMeterCMDBuf[DOMETER_ADDR_OFF]              =DOMeterAddr;
    DOMeterCMDBuf[DOMETER_FC_OFF]                =DOMETER_WREG_FC;
    DOMeterCMDBuf[DOMETER_START_OFF]         =0X00;
    DOMeterCMDBuf[DOMETER_START_OFF+1]=0X00;
    DOMeterCMDBuf[DOMETER_REGCNT_OFF]        =0X00;
    DOMeterCMDBuf[DOMETER_REGCNT_OFF+1]   =0X03;
    DOMeterCMDBuf[DOMETER_DATANUM_OFF]    =0X06;
    DOMeterCMDBuf[DOMETER_DATANUM_OFF+1]=HALM>>8;
    DOMeterCMDBuf[DOMETER_DATANUM_OFF+2]=HALM&0XFF;
    DOMeterCMDBuf[DOMETER_DATANUM_OFF+3]=LALM>>8;
```

```
    DOMeterCMDBuf[DOMETER_DATANUM_OFF+4]=LALM&0XFF;
    DOMeterCMDBuf[DOMETER_DATANUM_OFF+5]=LTC>>8;
    DOMeterCMDBuf[DOMETER_DATANUM_OFF+6]=LTC&0XFF;

    CRCDATA=CRC16(DOMeterCMDBuf,DOMETER_DATANUM_OFF+7);
    DOMeterCMDBuf[DOMETER_DATANUM_OFF+7] =CRCDATA>>8;
    DOMeterCMDBuf[DOMETER_DATANUM_OFF+8] =CRCDATA&0XFF;
    RS485_Send_Data(DOMeterCMDBuf,DOMETER_CMDBUF_SIZE);
    DOMeterReg|=DOMETER_WRITE_PEND;
}

void DOMeterErrReceiveHandle(void){
    uint8_t Temp=0x00;
    uint16_t rCRC=0x0000;          //readCRC
    uint16_t cCRC=0x0000;          //calculateCRC
    Temp=DOMeterReg&(~DOMETER_RBUF_UPDATE);
    rCRC=DOMeterDataBuf[DOMETER_ERRCRC_OFF]<<8|DOMeterDataBuf[DOMETER_ERRCRC
_OFF+1];
    cCRC=CRC16(DOMeterDataBuf,DOMETER_ERRCRC_OFF);
    if(rCRC!=cCRC){
        DOMeterCRCHandle();//CRCERR
        return ;
    }
    if(DOMeterDataBuf[DOMETER_ADDR_OFF]!=DOMeterAddr){
        DOMeterErrorHandle();  //Meter addr err
    }
    DOMeterErrFC=DOMeterDataBuf[DOMETER_FC_OFF]&~0x80;
    DOMeterErrCODE=DOMeterDataBuf[DOMETER_CODE_OFF];
    OLED_Clear();
    OLED_ShowString(DOMETER_DISP_TITAL_X,DOMETER_DISP_TITAL_Y,"DOMeter",16);    //L1
    OLED_ShowString(DOMETER_DISP_CODE_X,DOMETER_DISP_CODE_Y,"Error",16);
    //error
    OLED_ShowString(DOMETER_DISP_DO_X,DOMETER_DISP_DO_Y,"    ErrFC",15);      //ITEM

    OLED_ShowString(DOMETER_DISP_T_X,DOMETER_DISP_T_Y,"ErrCODE",15);
    OLED_ShowNum(DOMETER_DISP_DOVALUE_X,DOMETER_DISP_DOVALUE_Y,DOMeterErrFC,
4,15);      //VALUE
    OLED_ShowNum(DOMETER_DISP_TVALUE_X,DOMETER_DISP_TVALUE_Y,DOMeterErrCODE,
4,15);
    switch (Temp){
        case DOMETER_REQ_PEND:
            DOMeterReg&=(~DOMETER_RBUF_UPDATE);
            DOMeterReg&=(~DOMETER_REQ_PEND);
            DOMeterRequestData();
```

```
                break;
            case DOMETER_WRITE_PEND:
                DOMeterReg&=(~DOMETER_RBUF_UPDATE);
                DOMeterReg&=(~DOMETER_WRITE_PEND);
                DOMeterWriteReg(setHALM,setLALM,setLTC);
                break;

            default:   ;
        }
}

void DOMeterDataReceiveHandle(void){
        uint16_t rCRC=0x0000;               //readCRC
        uint16_t cCRC=0x0000;               //calculateCRC
        if(DOMeterDataBuf[DOMETER_ADDR_OFF]!=DOMeterAddr){
            DOMeterErrorHandle();
        }
        switch (DOMeterDataBuf[DOMETER_FC_OFF]){
            case DOMETER_REQ_FC:

        rCRC=DOMeterDataBuf[DOMETER_DATACRC_OFF]<<8|DOMeterDataBuf[DOMETER_DATAC
RC_OFF+1];
                cCRC=CRC16(DOMeterDataBuf,DOMETER_DATACRC_OFF);
                if(rCRC!=cCRC){
                    DOMeterCRCHandle();//CRCERR
                    return ;
                }
                DOData
        =DOMeterDataBuf[DOMETER_DOH_OFFSET]<<8|DOMeterDataBuf[DOMETER_DOL_OFFSET]
;
                DOTData
        =DOMeterDataBuf[DOMETER_TH_OFFSET]<<8|DOMeterDataBuf[DOMETER_TL_OFFSET];

        HALMData=DOMeterDataBuf[DOMETER_HALMH_OFFSET]<<8|DOMeterDataBuf[DOMETER
_HALML_OFFSET];

        LALMData=DOMeterDataBuf[DOMETER_LALMH_OFFSET]<<8|DOMeterDataBuf[DOMETER_
LALML_OFFSET];
                LTCData
        =DOMeterDataBuf[DOMETER_LTCH_OFFSET]<<8|DOMeterDataBuf[DOMETER_LTCL_OFFSET]
;
                STAData   =DOMeterDataBuf[DOMETER_ALMSTA_OFFSET];
                DOMeterReg&=(~DOMETER_RBUF_UPDATE);
                DOMeterReg&=(~DOMETER_REQ_PEND);
```

```
                break;
            case DOMETER_WREG_FC:
                rCRC=DOMeterDataBuf[6]<<8|DOMeterDataBuf[6+1];
                cCRC=CRC16(DOMeterDataBuf,6);
                if(rCRC!=cCRC){
                    DOMeterCRCHandle();//CRCERR
                    return ;
                }
                DOMeterReg&=(~DOMETER_RBUF_UPDATE);
                DOMeterReg&=(~DOMETER_WRITE_PEND);
                break;

            default:   ;
        }
}

void DOMeterReceiveHandle(void){
        if(DOMeterDataBuf[DOMETER_FC_OFF]&0x80){
            DOMeterErrReceiveHandle();
        }else{
            DOMeterDataReceiveHandle();
        }
}

void DOMeterCheck(void){
        if(DOMeterReg&DOMETER_RBUF_UPDATE){
            DOMeterReceiveHandle();
            DOMeterReg&=~DOMETER_RBUF_UPDATE;
        }


}
```