单端液位模组

Liquid-level-Single-ended-Pro

(LSP)

I2C 协议手册

文档版本：V1.2

2023/08/04

1 通信协议

本文介绍了敏源传感单端液位模组 I2C 接口协议及寄存器地址信息。

模组 I2C 从机地址为 0x20。

0	1	0	0	0	0	0	R/W

SLAVE ADDRESS

1.1 寄存器地址

地址	寄存器名称	寄存器说明	默认值
0X20	F1_MSB	通道 1 频率 高字节	0X00
0X21	F1_IMB	通道 1 频率 中字节	0X00
0X22	F1_LSB	通道 1 频率 低字节	0X00
0X23	F1_CRC	通道 1 频率 CRC 校验字节	0X00
0X24	C1_MSB	通道 1 电容 高字节	0X00
0X25	C1_IMB	通道 1 电容 中字节	0X00
0X26	C1_LSB	通道 1 电容 低字节	0X00
0X27	C1_CRC	通道 1 电容 CRC 校验字节	0X00
0X28	R1_MSB	双通道比值 高字节	0X00
0X29	R1_LSB	双通道比值 低字节	0X00
0X2A	R1_CRC	双通道比值 CRC 校验字节	0X00

表 1 通道 1 寄存器

地址	寄存器名称	寄存器说明	默认值
0X30	F1_MSB	通道 2 频率 高字节	0X00
0X31	F1_IMB	通道 2 频率 中字节	0X00
0X32	F1_LSB	通道 2 频率 低字节	0X00
0X33	F1_CRC	通道 2 频率 CRC 校验字节	0X00
0X34	C1_MSB	通道 2 电容 高字节	0X00
0X35	C1_IMB	通道 2 电容 中字节	0X00
0X36	C1_LSB	通道 2 电容 低字节	0X00
0X37	C1_CRC	通道 2 电容 CRC 校验字节	0X00

表 2 通道 2 寄存

地址	寄存器名称	寄存器说明	默认值
0X10	VT_MSB	VT 高字节	0X00
0X11	VT_LMB	VT 低字节	0X00
0X12	VT_CRC	VT CRC 校验字节	0X00

表 3　温度寄存器

地址	寄存器名称	寄存器说明	默认值
0X50	FZ1_MSB	通道 1 零点频率 高字节	0X00
0X51	FZ1_ISB	通道 1 零点频率 中字节	0X00
0X52	FZ1_LSB	通道 1 零点频率 低字节	0X00
0X53	FZ2_MSB	通道 2 零点频率 高字节	0X00
0X54	FZ2_ISB	通道 2 零点频率 中字节	0X00
0X55	FZ2_LSB	通道 2 零点频率 低字节	0X00
0X56	RZ1_MSB	零点比值 高字节	0X00
0X57	RZ1_LSB	零点比值 低字节	0X00
0X5C	FIN_DIV_CH1	通道 1 振荡信号分频	0X05
0X5E	FIN_DIV_CH2	通道 2 振荡信号分频	0X05

表 4　零点寄存器

地址	寄存器名称	寄存器说明	默认值
0X01	Config_REG	配置寄存器	0XF1
0X02	Parameter_REG	参数寄存器	0X00
0X03	Status_REG	状态寄存器	0X00
0X04	Control_REG	控制寄存器	0X00

表 5　通用寄存器

1.1.1 配置寄存器 Config_REG（0X01）

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	默认值
ISEL		CH_EN		CR		OS	SD	0XF1
说明:								
ISEL		驱动电流设置: 00：5mA；01：6mA；10：7mA；11：8mA；						11
CH_EN		00：关闭通道 01：开启通道 1 10：开启通道 2 11：开启双通道						11

CR	转换速率设置: 00：1s 转换；01：5s 转换；10：10s 转换；11：30s 转换；	00
OS/SD	00：连续转换(频率、电容、比值、VT) 01：停止转换，待机模式 10：连续转换 11：单次转换(频率、电容、比值、VT)	01

1.1.2 参数寄存器 Parameter_REG（0X02）

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	默认值
	FIN_CH2				FIN_CH1			0X00
说明:								
FIN_CH2	通道 2 分频比设置（增加采集分辨率）；MCO=16M 000： 32 分频，全区域振荡（分辨率较低）； 001： 4 分频，振荡频率在 30MHz 以下; 010： 8 分频，振荡频率在 30-60MHz 之间; 011：16 分频，振荡频率在 60MHz 以上; 1XX:自定义分频，写入 FIN_DIV_CH2(0x5F)。							000
FIN_CH1	通道 1 分频比设置（增加采集分辨率）；MCO=16M 000： 32 分频，全区域振荡（分辨率较低）； 001： 4 分频，振荡频率在 30MHz 以下; 010： 8 分频，振荡频率在 30-60MHz 之间; 011：16 分频，振荡频率在 60MHz 以上; 1XX:自定义分频，写入 FIN_DIV_CH1(0x5C)。							000

注：设置成自定义分频之前，需要将自定义分频数据写入 FIN_DIV_CHx 寄存器。

1.1.3 状态寄存器 Status_REG（0X03）

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	默认值
ZFRY						DRDY2	DRDY1	0X00
说明:								
ZFRY	零点比值校准忙标志位: ZBR=1:正在校准（忙）；ZBR=0:校准完成;							0
DRDY1 DRDY2	通道 1/2 测量忙标志位: DRDY1/DRDY2 = 1 ： 正在测量转换（忙） DRDY1/DRDY2 = 0 ： 测量转换完成							00

1.1.4 控制寄存器 Control_REG（0X04）

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	默认值

ZFR	SC		RESET	0X00
说明:				
ZFR		双通道零点频率、比值校准; 置 1 后，开启双通道零点频率、比值校准,需等待 500ms，完成校准后清零。		0
SC		置 1 后，存储当前设置，并重启设备，需等待 20ms，完成后清零（模块再次上电后根据存储数据初始化）		0
RESET		10101：启动软恢复 模组复位 LSP，重新写入配置启动工作，需等待 20ms，完成后清零。		00000

注：校准需要采集多次数据进行平均，因此校准时间较长。

注：对寄存器进行设置修改后，均需置位 SC ，实现存储。

1.2 读写指令

支持 100-400kHz 的 I2C 总线通信速率，主机先发送从机地址和写标志位（Slave Address + W），紧跟寄存器逻辑地址（Register Address）。对于读时序，主机随后再次发送从机地址和读标志位（Slave Address + R），然后从机向主机发送数据（Data from Register）；对于写时序，主机随后直接向从机发送数据（Data to Register）。需要注意的是，从机地址宽度为 7bit，写标志位 W=0，读标志位 R=1。

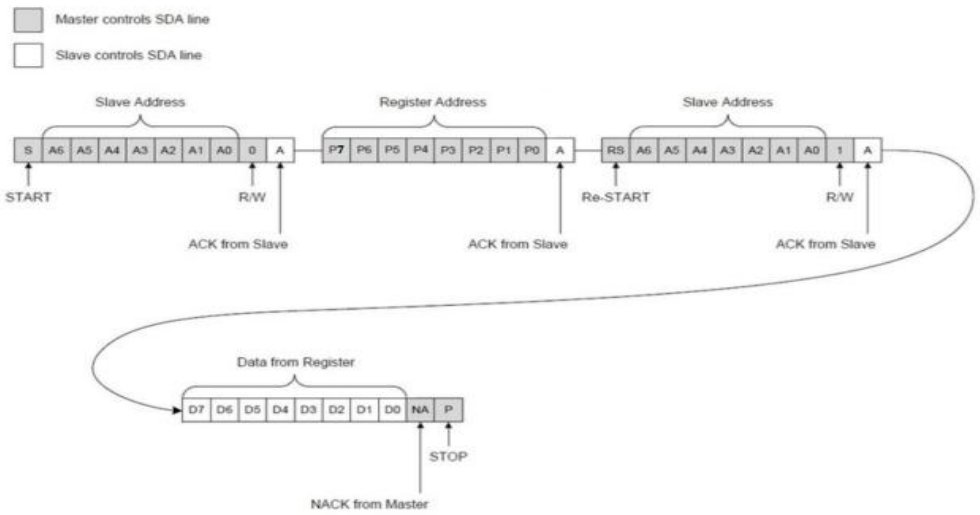

图 1　I2C 读时序

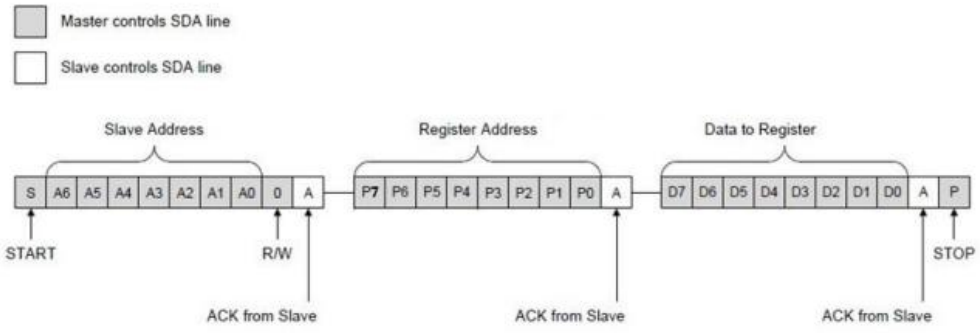

图 2　I2C 写时序

1.3 功能指令

LSP 模组提供的功能指令能快速获得传感检测数据。

命令名	功能	代码
测量通道 1 频率	Convert_F1	0xCC10
测量通道 1 电容	Convert_C1	0xCC11
测量通道 2 频率	Convert_F2	0xCC20
测量通道 2 电容	Convert_C2	0xCC21
测量比值 Ratio	Covert_R1	0xCC30
测量温度 VT	Convert_VT	0xCC70

1.3.1 测量通道 1 频率（0xCC10）

发出测量命令 Convert_F1 (0xCC10）触发一次通道 1 频率的测量。

建议等待 20ms 后读取返回值。

Freq=（频率高字节*65536+频率中字节*256+频率低字节）/1000，单位 MHz。

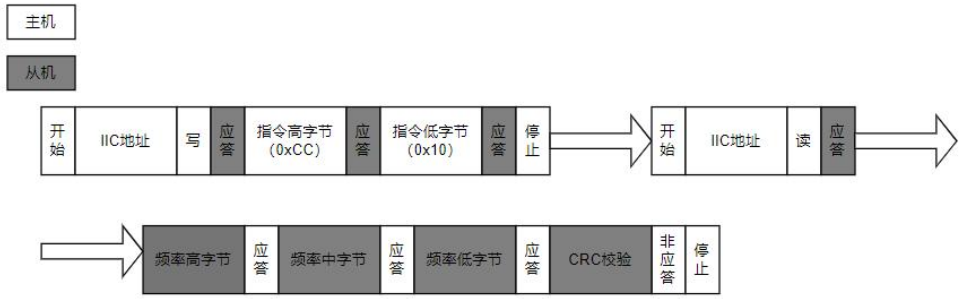

图 3　测量通道 1 频率流程图

1.3.2 测量通道 1 电容（0xCC11）

发出测量命令 Convert_C1 (0xCC11）触发一次通道 1 电容的测量计算。

建议等待 20ms 后读取返回值。

Cap=（电容高字节*65536+电容中字节*256+电容低字节）/1000，单位 pF。

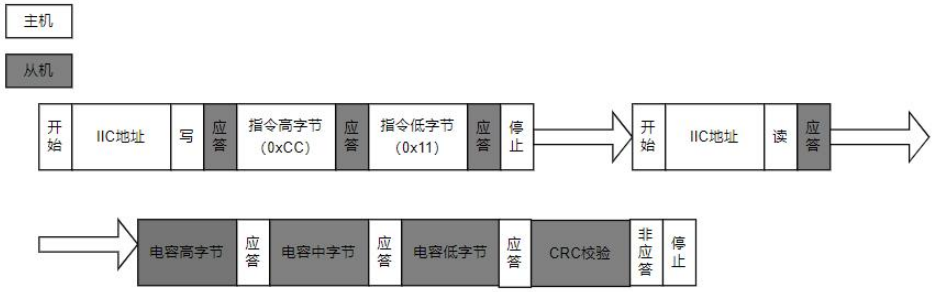

图 4　测量通道 1 电容值流程图

1.3.3 测量通道 2 频率（0xCC20）

发出测量命令 Convert_F2 (0xCC20）触发一次通道 2 频率的测量。

建议等待 20ms 后读取返回值。

Freq=（频率高字节*65536+频率中字节*256+频率低字节）/1000，单位 MHz。

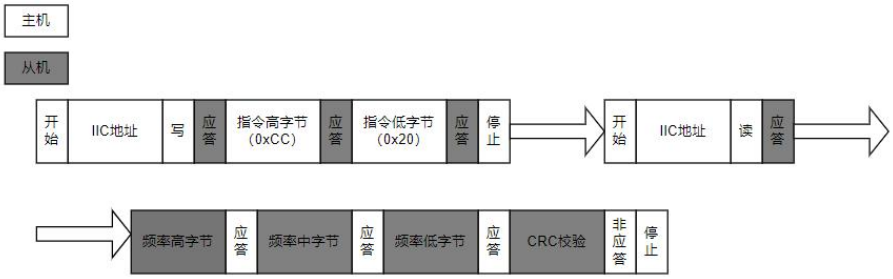

图 5　测量通道 2 频率流程图

1.3.4 测量通道 2 电容（0xCC21）

发出测量命令 Convert_C2 (0xCC21）触发一次通道 2 电容的测量计算。

建议等待 20ms 后读取返回值。

Cap=（电容高字节*65536+电容中字节*256+电容低字节）/1000，单位 pF。

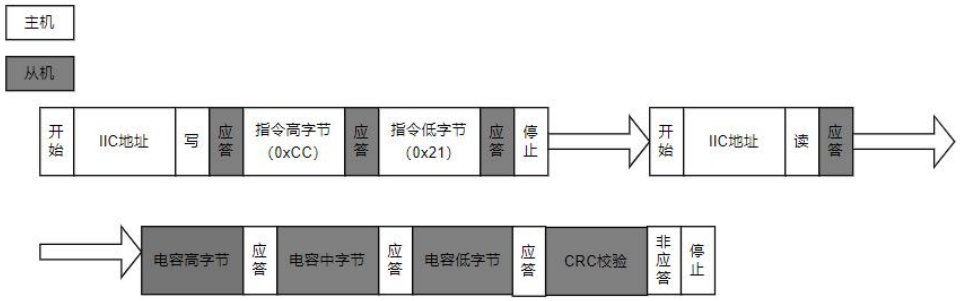

图 6　测量通道 2 电容值流程图

1.3.5 测量比值（0xCC30）

发出测量命令 Convert_R1 (0XCC30）触发一次双通道频率比值的测量计算。建议等待 50ms 后读取返回值。

Ratio=（R1 比值高字节*256+R1 比值低字节）/10000。

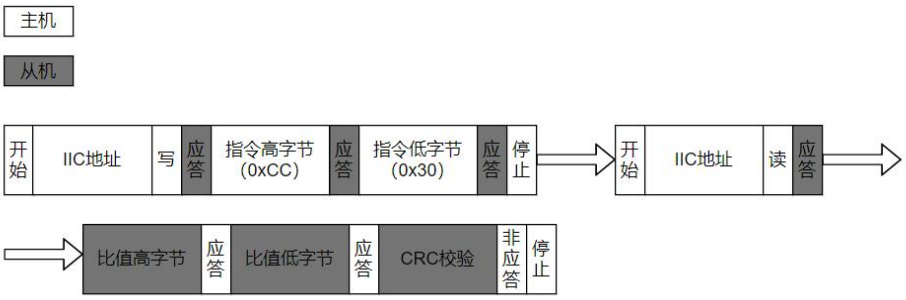

图 7　测量比值流程图

1.3.6 测量温度（0xCC70）

发出测量命令 Convert_VT (0xCC70）触发一次芯片环境温度的测量。VT 电压与温度成反比。建议等待 20ms 后读取返回值。

VT=（VT 电压高字节*256+VT 电压低字节）/10，单位 mV。

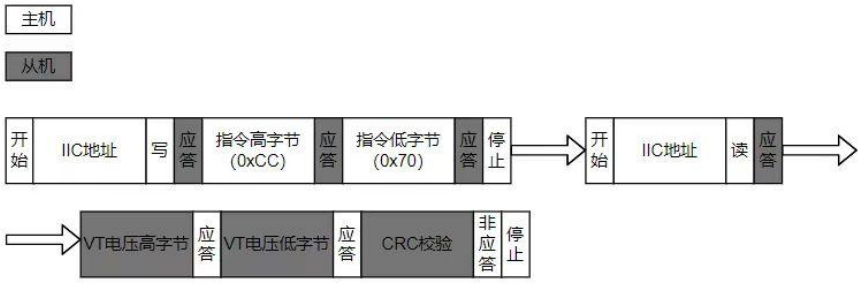

图 8　测量 VT 流程图

2 通讯协议示例

```
//硬件 IIC 初始化;
void I2C_HardInit(void)
    {
        //IIC GPIO 初始化;
        I2C1_GPIO_Config();
        //初始化 IIC 主机模式;
        I2C_MasterModeInit(I2C1, 400000);
        //配置 IIC 从机地址;
        I2C_SetDeviceAddr(I2C1, IIC_ADDR);
}
//IIC GPIO 配置;
 void I2C1_GPIO_Config(void)
    {
```

```
    GPIO_InitTypeDef GPIO_InitStruct;
    //使能 GPIO 时钟;
    RCC_AHBPeriphClockCmd(I2C1_SDA_BUSCLK, ENABLE);
    RCC_AHBPeriphClockCmd(I2C1_SCL_BUSCLK, ENABLE);
    //复用 GPIO 功能;
    GPIO_PinAFConfig(I2C1_SCL_PORT, GPIO_PinSource5, GPIO_AF_3);
    GPIO_PinAFConfig(I2C1_SDA_PORT, GPIO_PinSource4, GPIO_AF_3);
    GPIO_StructInit(&GPIO_InitStruct);
    GPIO_InitStruct.GPIO_Pin   = I2C1_SCL_PIN;
    //设置 GPIO 速度;
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
    //保持 SCL&SDA 总线初始因为上拉电阻为高;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF_OD;
    GPIO_Init(I2C1_SCL_PORT, &GPIO_InitStruct);
    GPIO_InitStruct.GPIO_Pin   = I2C1_SDA_PIN;
    GPIO_Init(I2C1_SDA_PORT, &GPIO_InitStruct);
}
//IIC 主机模式初始化;
 void I2C_MasterModeInit(I2C_TypeDef* I2Cx, u32 uiI2C_speed)
     {
        I2C_InitTypeDef I2C_InitStruct;
        //使能 IIC 时钟;
        RCC_APB1PeriphClockCmd(RCC_APB1ENR_I2C1, ENABLE);
        I2C_StructInit(&I2C_InitStruct);
        //配置 IIC 作为主机模式;
        I2C_InitStruct.Mode = I2C_CR_MASTER;
        I2C_InitStruct.OwnAddress = 0;
        //配置 IIC 为快速模式 100k-400khz
        I2C_InitStruct.Speed = I2C_CR_FAST;
        //设置 IIC 时钟速度;
        I2C_InitStruct.ClockSpeed = uiI2C_speed;
        //初始化 IIC 设置;
        I2C_Init(I2Cx, &I2C_InitStruct);
       //使能 IIC;
        I2C_Cmd(I2Cx, ENABLE);
```

```
    }
    //设置 IIC 从机地址；
     void I2C_SetDeviceAddr(I2C_TypeDef* I2Cx, u8 deviceaddr)
        {
            //失能 IIC;
            I2C_Cmd(I2Cx, DISABLE);
            //设置从机设备地址；
            I2C_Send7bitAddress(I2Cx, deviceaddr, I2C_Direction_Transmitter);
            //使能 IIC;
            I2C_Cmd(I2Cx, ENABLE);
    }

    unsigned char MY_I2C_CRC8(unsigned char Data[], unsigned char nbrOfBytes)
    {
      unsigned char BIT;            // bit mask
      unsigned char crc = 0xFF; // calculated checksum
      unsigned char byteCtr;       // byte counter

      // calculates 8-Bit checksum with given polynomial
      for(byteCtr = 0; byteCtr < nbrOfBytes; byteCtr++)
      {
        crc ^= (Data[byteCtr]);
        for(BIT = 8; BIT > 0; --BIT)
        {
          if(crc & 0x80) crc = (crc << 1) ^ 0x131;
          else    crc = (crc << 1);
        }
      }

      return crc;
}
    //读任意寄存器；
     int I2C_ReadANYREG(char *argstr)
        {
          u8 args,len,i,addr;
```

```
        unsigned int temp;
        //获取输入键值;
        args = sscanf(argstr+1, "%x", &temp);
        //低 8 位为读取长度;
        len = temp&0xff;
        //高 8 位为起始地址;
        addr = temp>>8;
        if(args)
        {
            u8 tempdata[128]={0x00};
             //读取从机寄存器;
            iic_read_anyreg(tempdata,temp);
            //打印寄存器地址对应的数值;
            for(i=0;i<len;i++)
            {
                printf("\r\n ADDR:%02x VALUE:%02X",addr+i,tempdata[i]);
            }
        }
        return 1;

}
 void iic_read_anyreg(uint8_t *data,uint16_t temp)
    {
        u32 timeout=100000;
        uint8_t addr,len,flag=0,i,ERROR_IIC=0;;
        addr = temp>>8;
        len = temp&0xff;
        for(i=0;i<len;i++)
        {
            //IIC 清除 STOP 标志位;
            I2C_ClearFlag(I2C1,I2C_FLAG_STOP_DET);
            //主机 start 并且发送 IIC 地址+写位，从机应答后继续发送初始地址;
            I2C_SendData(I2C1, addr);
            //地址递增;
            addr+=1;
```

```
//检测发送缓冲区是否为空，同时 timeout 开始递减，保持程序运行；
while((I2C_GetFlagStatus(I2C1, I2C_STATUS_FLAG_TFE) == 0)&&timeout>0)
{
    timeout--;
}
//等待 100us 检测从机是否应答，无应答则产生 STOP；
Delay_us(100);
//检测是否产生了 STOP 信号；
if(I2C_GetITStatus(I2C1, I2C_IT_STOP_DET)==1)
{
    //停止 IIC，清除所有缓存区；
    I2C_GenerateSTOP(I2C1, ENABLE);
    //检测 IIC 是否已经停止，同时 timeout 开始递减，保持程序运行；
    while(((I2C_GetITStatus(I2C1, I2C_IT_STOP_DET)) == 0)&&timeout>0)
    {
        timeout--;
    }
    timeout=10000;
    //IIC_Error 标志位置 1，本次通信错误；
    ERROR_IIC=1;
    printf("\r\n NO ADDR ACK");
    break;
}
timeout=100000;
//主机发送 restart 信号；
I2C_GenerateSTART(I2C1, ENABLE);
//检测 IIC 是否已经 start，同时 timeout 开始递减，保持程序运行；
while(((I2C_GetITStatus(I2C1, I2C_IT_START_DET)) == 0)&&timeout>0)
{
    timeout--;
}
timeout=100000;
while(1)
{
        //检测 IIC 发送缓冲区是否未满，主机需要读取一次从机寄存器；
```

```
                if ((I2C_GetFlagStatus(I2C1, I2C_STATUS_FLAG_TFNF)) && (flag == 0))
                {
                    //IIC 清除 STOP 标志位；
                    I2C_ClearFlag(I2C1,I2C_FLAG_STOP_DET);
                    //发送地址+读位；
                    I2C_ReadCmd(I2C1);
                    flag = 1;
                }

                //检测 IIC 接收缓存非空；
                if (I2C_GetFlagStatus(I2C1, I2C_STATUS_FLAG_RFNE)) {
                        //从 IIC 接收缓存区获取数据到 data 数组；
                        data[i] = I2C_ReceiveData(I2C1);
                        break;
                }
                else
                {
                    timeout--;
                    if(timeout<=0)break;
                }
                //检测是否产生了 STOP 信号；
                if((I2C_GetITStatus(I2C1, I2C_IT_STOP_DET) == 1) )
                {
                    //停止 IIC，清除所有缓存区；
                    I2C_GenerateSTOP(I2C1, ENABLE);
                    //检测 IIC 是否已经停止，同时 timeout 开始递减，保持程序运行；.
                    while(((I2C_GetITStatus(I2C1,I2C_IT_STOP_DET))==
0)&&timeout>0)
                    {
                        timeout--;
                    }
                    timeout=10000;
                    //IIC_Error 标志位置 1，本次通信错误；
                    ERROR_IIC=1;
                    printf("\r\n RECEVICE_FAIL");
```

```
                                break;
                            }
                    }
                    timeout=10000;
                    //停止 IIC，清除所有缓存区；
                    I2C_GenerateSTOP(I2C1, ENABLE);
                    //检测 IIC 是否已经停止，同时 timeout 开始递减，保持程序运行；
                    while(((I2C_GetITStatus(I2C1, I2C_IT_STOP_DET)) == 0)&&timeout>0)
                    {
                        timeout--;
                    }
                    timeout=10000;
                    flag=0;
                }
            }
            //测量频率例程；
            void IIC_Measure_F(void)
            {
            u32 i = 0,cnt=0,flag=0,timeout=100000;;
             u8 temp[4]={0x00};
             float fre=0;
            //主机 start 并且发送 IIC 地址+写位，从机应答后继续发送指令高 8 位 0xCC；
             I2C_SendData(I2C1, 0xcc);
             //检测发送缓冲区是否为空，同时 timeout 开始递减，保持程序运行；
             while((I2C_GetFlagStatus(I2C1,I2C_STATUS_FLAG_TFE)==0)&&timeout>0)
             {
                timeout--;
             }
             timeout=100000;
            //发送指令低 8 位 0x10；
             I2C_SendData(I2C1, 0x10);
             //检测发送缓冲区是否为空，同时 timeout 开始递减，保持程序运行；

             while((I2C_GetFlagStatus(I2C1,I2C_STATUS_FLAG_TFE)==0)&&timeout>0)
             {
```

```
        timeout--;
    }
    timeout=100000;
    //停止 IIC，清除所有缓存区；
    I2C_GenerateSTOP(I2C1, ENABLE);
    //检测 IIC 是否已经停止，同时 timeout 开始递减，保持程序运行；
    while(((I2C_GetITStatus(I2C1, I2C_IT_STOP_DET)) == 0)&&timeout>0)
    {
        timeout--;
    }
    timeout=100000;
   //等待 10ms，从机开始测量；
    Delay_ms(10);
   //连续读取 4 个寄存器；
   for (i = 0; i < 4; i++)
    {
     while(1)
        {
                //检测 IIC 发送缓冲区是否未满，主机需要读取两次从机寄存器；
                if ((I2C_GetFlagStatus(I2C1, I2C_STATUS_FLAG_TFNF)) && (flag == 0)) {
                        //主机 start，发送地址+读位；
                        I2C_ReadCmd(I2C1);
                        //累计主机读取次数；
                        cnt++;
                        //当
                        if (cnt == 2) flag = 1;
                }
                //检测 IIC 接收缓存非空；
                if (I2C_GetFlagStatus(I2C1, I2C_STATUS_FLAG_RFNE)) {
                        //从 IIC 接收缓存区获取数据到 temp 数组；
                        temp[i] = I2C_ReceiveData(I2C1);
                        break;
                }
                else
```

```
                {
                    timeout--;
                    if(timeout<=0)break;
                }
            }
            timeout=100000;
        }
            //停止 IIC，清除所有缓存区；
            I2C_GenerateSTOP(I2C1, ENABLE);
    //检测 IIC 是否已经停止，同时 timeout 开始递减，保持程序运行；
    while(((I2C_GetITStatus(I2C1,I2C_IT_STOP_DET))== 0)&&timeout>0)
            {
                timeout--;
            }
    //校验数据，计算频率；
        if(temp[3]==MY_I2C_CRC8(temp,3))
        {
                fre = (temp[0]<<16|temp[1]<<8|temp[2])/1000.0f;
                printf(" FA= %.3f Mhz ",fre);
            }
    else printf("\r\nCRC FAULSE %02X %02X %02X %02X
    CRC %02X",temp[0],temp[1],temp[2],temp[3],MY_I2C_CRC8(temp,2));
    }
```

REVISION HISTORY

Rev #	Date	Author	Changes	Comments
0.9	2023/02/02	Minyuan Team	Draft	For evaluation only

1.0	2023/02/07	Minyuan Team	Update Register Map ADD CRC ADD multibyte read	
1.1	2023/04/19	Minyuan Team	Update Register Map ADDFunctionalinstructions	
1.2	2023/06/28	Minyuan Team	Update Register Map ADDFunctionalinstructions ADDGeneralregister	