NV32F100 IIC通信接口编程

第一章 所有库函数简介

库函数列表

void I2C_Init(I2C_Type *pI2Cx,I2C_ConfigPtr pI2CConfig)
IIC 初始化函数
uint8_t I2C_Start(I2C_Type *pI2Cx)
IIC 起始位发送函数
uint8_t I2C_Stop(I2C_Type *pI2Cx)
IIC 停止位发送函数
uint8_t I2C_RepeatStart(I2C_Type *pI2Cx)
IIC 重复起始位发送
void I2C_SetSlaveAddress(I2C_Type *pI2Cx,uint16_t u16SlaveAddress)
IIC 设置从机地址
void I2C_IntDisable(I2C_Type *pI2Cx)
关闭 IIC 中断
void I2C_IntEnable(I2C_Type *pI2Cx)
打开 IIC 中断
void I2C_SetSCLLowETMeout(I2C_Type *pI2Cx, uint16_t u16ETMeout)
设置低超时 SCL 低电平的时间
void I2C_Deinit(I2C_Type *pI2Cx)
关闭 IIC 系统时钟
uint8_t I2C_WriteOneByte(I2C_Type *pI2Cx, uint8_t u8WrBuff)
让 IIC 发送单字节数据函数
uint8_t I2C_ReadOneByte(I2C_Type *pI2Cx, uint8_t *pRdBuff, uint8_t u8Ack)
让 IIC 接收单字节数据函数
uint8_t I2C_MasterSendWait(I2C_Type *pI2Cx,uint16_t u16SlaveAddress,uint8_t *pWrBuff,uint32_t u32Length)
让 IIC 发送多个字节
uint8_t I2C_MasterReadWait(I2C_Type *pI2Cx,uint16_t u16SlaveAddress,uint8_t *pRdBuff,uint32_t u32Length)
让 IIC 接收多个字节

寄存器操作的内联函数，调用内联函数和直接操作寄存器效率一样高
uint8_t I2C_IsTxMode(I2C_Type *pI2Cx);
返回 I2C 发送模式
uint8_t I2C_IsBusy(I2C_Type *pI2Cx);
返回 I2C 总线状态
uint8_t I2C_IsReceivedAck(I2C_Type *pI2Cx);
返回接收 ACK 的标志位
uint8_t I2C_IsMasterMode(I2C_Type *pI2Cx);
返回 I2C 主从机模式
void I2C_ClearSHTF2(I2C_Type *pI2Cx);
清除高超时中断标志位 2

void I2C_ClearSLTF(I2C_Type *pI2Cx);
清除低超时中断标志位
uint8_t I2C_IsSMB_SHTF2(I2C_Type *pI2Cx);
返回高超时中断标志位 2
uint8_t I2C_IsSMB_SLTF(I2C_Type *pI2Cx);
返回低超时中断标志位
void I2C_TxEnable(I2C_Type *pI2Cx);
将 I2C 配置为发送模式
void I2C_RxEnable(I2C_Type *pI2Cx);
将 I2C 配置为接收模式
void I2C_IntEnable(I2C_Type *pI2Cx);
打开 I2C 使能
void I2C_IntDisable(I2C_Type *pI2Cx);
关闭 I2C 使能
void I2C_SetBaudRate(I2C_Type *pI2Cx,uint32_t u32Bps);
设置 I2C 波特率
void I2C_SetSlaveAddress(I2C_Type *pI2Cx,uint16_t u16SlaveAddress);
设置 I2C 从机地址
void I2C_GeneralCallEnable(I2C_Type *pI2Cx);
打开地址反馈使能
void I2C_SMBusAlertEnable(I2C_Type *pI2Cx);
设置扩展地址位宽
void I2C_RangeAddressEnable(I2C_Type *pI2Cx);
打开通用地址使能
void I2C_SHTF2IntEnable(I2C_Type *pI2Cx);
打开 SHTF2 中断标志使能
void I2C_ETMeoutCounterClockSelect(I2C_Type *pI2Cx, uint8_t u8Clock);
打开快速 ACK/NACK 使能
uint8_t I2C_GetStatus(I2C_Type *pI2Cx);
返回 I2C 状态寄存器的状态
void I2C_ClearStatus(I2C_Type *pI2Cx, uint8_t u8ClearFlag);
清除目标标志位
void I2C_SendAck(I2C_Type *pI2Cx);
发送 ACK
void I2C_SendNack(I2C_Type *pI2Cx);
发送 NACK
void I2C_SecondAddressEnable(I2C_Type *pI2Cx);
打开 I2C 次地址的使能
void I2C_WriteDataReg(I2C_Type *pI2Cx, uint8_t u8DataBuff);
对数据寄存器写发送一个 8bit 数据
uint8_t I2C_ReadDataReg(I2C_Type *pI2Cx);
读取接收到的数据
void I2C0_SetCallBack(I2C_CallbackType pCallBack);
I2C0 中断回调函数

void I2C1_SetCallBack(I2C_CallbackType pCallBack);

I2C1 中断回调函数

I2C 特性说明

*多个主机操作
*软件可编程一个模块，其中包括 64 种不同的串行时钟频率
*软件选择应答位
*中断驱动的逐字节数据传送
*从主机到从机自动模式切换的仲裁丢失中断
*启动和停止信号的产生和检测
*重复启动信号的产生和检测
*应答位的产生和检测
*总线繁忙检测
*可识别通用广播地址
*10 位地址扩展
*支持系统管理总线（SMBus）规格，版本 2.0
*可编程毛刺输入滤波器
*在从机地址匹配下低功，耗模式可唤醒
*支持从机地址的范围选择

I2C 使用说明

配置波特率，I2C 使能，I2C 中断使能，打开系统时钟使能就可以让 I2C 工作了，但是在实际的发送和接收过程中以从机的时序为主相应的调用库函数。

1.1 IIC 模块初始化

对 I2Cx_C1 进行操作，打开 I2C 使能，中断使能，配置主从机模式

位	描述
7 IICEN	I2C 使能 使能 I2C 模块操作 0 关闭使能 1 打开使能
6 IICIE	I2C 中断使能 使能 I2C 模块操作 0 关闭使能 1 打开使能

5 MST	主机模式选择 当 MST bit 位从 0 到 1 发生变化，总线开始信号打开和主机模式被选择。当此位从 1 到 0 变化，停止信号打开并且模式从主机模式转变为从机模式。 0 从机模式 1 主机模式
4 TX	发送模式选择 选择主机和从机的传输方向。在主控模式下，该位必须根据传输类型需要设置。因此，对于地址周期，该位始终置 1。当作为从机时该位必须由软件根据状态寄存器的 SRW 位进行设置 0 接收 1 发送
3 TXAK	发送确认使能 对于主机和从机接收器，指定驱动数据会在数据确认周期内发送给到 SDA。在 FACK 位的值会影响 NACK/ ACK 产生。 注：SCL 保持低电平，直到 TXAK 被写入。 0 确认信号会被发送到总线上的即将要接收的字节（如果 FACK 清零）或当前接收的字节（如果 FACK 设置）。 1 无应答信号被发送到总线上的即将要接收的数据字节（如果 FACK 被清除）或当前接收的数据字节（如果 FACK 设置）。
2 RSTA	重复开始 写 1 到该位产生一个重复起始条件并送给当前主机。但该位值始终读为零。在错误的时间打开循环会导致仲裁丢失。
1 WUEN	唤醒使能 当 I2C 模块进行从地址匹配并且没有外设总线运行时可将 MCU 从低功耗模式下唤醒。 0 正常操作。在低功耗模式下地址匹配无唤醒中断产生。 1 打开在低功耗模式下的唤醒使能
0 保留	该位保留 这个只读域被保留，并且总是具有值 0。

对 I2Cx_F 寄存器赋值配置 I2C 模块的波特率，下面的表格有算法说明

位	描述
7-6 MULT	MULT 位定义了乘数因子 MUL。这个因子与 SCL 分频器相结合后生成 I2C 波特率 00 mul=1， 01 mul=2 ,10 mul=4 ， 11 保留

5-0 ICR	时钟速率 对 I2C 模块进行预分频，以便选择比特率。此字段和 MULT 领域决定了 I2C 波特率、SDA 保持时间、SCL 开始保持时间和 SCL 停止保持时间。对于价值观的对应列表每个 ICR 设置，请参阅 I2C 分频器和保存值。 SCL 分频数与 mul 乘数因子相乘的结果决定 I2C 波特率。 I2C baud rate = bus speed (Hz)/(mul × SCL divider) SDA 保持时间是从 SCL（I2C 时钟）的下降沿开始直至 SDA（I2C 数据）发生变化的延迟。 SDA hold time = bus period (s) × mul × SDA hold value SCL 开始保持时间是从 SCL 为高（启动状态）的 SDA（I2C 数据）的下降沿开始到 SCL（I2C 时钟）的下降沿的延迟。 SCL start hold time = bus period (s) × mul × SCL start hold value 在 SCL 停止保持时间是从 SCL（I2C 时钟）的上升沿开始到 SCL 为高（停止条件）的 SDA 的上升沿（I2C 数据）之间的延迟。 SCL stop hold time = bus period (s) × mul × SCL stop hold value

函数名	I2C_Init
函数原形	I2C_Init(I2C_Type *pI2Cx, I2C_ConfigPtr pI2CConfig)
功能描述	以配置结构体 pConfig 来初始化 I2C
输入参数	配置结构体 I2C_ConfigType，模块结构体 I2C_Type
输出参数	无
返回值	无
先决条件	无
函数使用实例	先设置配置结构体，I2C_Init(I2C0, &I2C_Config);

```
/*****************************************************************************//
*
* 打开 IIC 系统时钟 设置 IIC 波特率 设置主机模式
*
* @无返回
*
*****************************************************************************/

void I2C_Init(I2C_Type *pI2Cx,I2C_ConfigPtr pI2CConfig)
{
    uint8_t u8Temp;

#if defined(CPU_NV32)
    SIM->SCGC |= SIM_SCGC_IIC_MASK;      //打开 IIC 系统时钟
#endif

    I2C_SetBaudRate(pI2Cx,pI2CConfig->u16F);          //给分频寄存器赋值，设置波特率
    I2C_SetSlaveAddress(pI2Cx,pI2CConfig->u16OwnA1);   //设置从机地址
    pI2Cx->FLT = (uint8_t)pI2CConfig->u16Filt;              //配置是否打开杂波过滤器
```

```
    pI2Cx->RA = (uint8_t)pI2CConfig->u16RangeA & 0xfe;   //配置 I2C 从机地址
    I2C_SetSCLLowETMeout(pI2Cx,pI2CConfig->u16Slt);//设置低超时时钟低电平时间

/* 对 C2 控制寄存器赋值*/
/*对 I2C 各个参数进行使能操作*/
    u8Temp = 0;
    if( pI2CConfig->sSetting.bGCAEn )
    {
        u8Temp |= I2C_C2_GCAEN_MASK;        //打开通用调用地址使能
    }
    if( pI2CConfig->sSetting.bAddressExt )
    {
        u8Temp |= I2C_C2_ADEXT_MASK;        //配置扩展地址位宽
    }
    if( pI2CConfig->sSetting.bRangeAddEn )
    {
        u8Temp |= I2C_C2_RMEN_MASK;         //打开范围地址匹配使能
    }
    pI2Cx->C2 |= u8Temp;

    /* 初始化 SMB 寄存器 */
    u8Temp = 0;
    if( pI2CConfig->sSetting.bFackEn )
    {
        u8Temp |= I2C_SMB_FACK_MASK;      //打开快速 NACK/ACK 使能
    }
    if( pI2CConfig->sSetting.bSMB_AlertEn )
    {
        u8Temp |= I2C_SMB_ALERTEN_MASK;   //打开 SMBus 提醒响应地址使能
    }
    if( pI2CConfig->sSetting.bSecondAddressEn )
    {
        u8Temp |= I2C_SMB_SIICAEN_MASK;    //打开次地址使能
    }
    if( pI2CConfig->sSetting.bSHTF2IntEn )
    {
        u8Temp |= I2C_SMB_SHTF2IE_MASK;    //打开 SHTF2 中断使能
    }
    pI2Cx->SMB = u8Temp;

    /* 配置 C1 寄存器 */
    u8Temp = 0;
    if( pI2CConfig->sSetting.bIntEn )    //打开 I2C 中断使能
    {
```

```
        u8Temp |= I2C_C1_IICIE_MASK;
        if(pI2Cx == I2C0)
        {
            NVIC_EnableIRQ(I2C0_IRQn);
        }
    #if defined(CPU_NV32M4)
        else if(pI2Cx == I2C1)
        {
            NVIC_EnableIRQ(I2C1_IRQn);
        }
    #endif
        else
        {
            //
        }
    }
    if( pI2CConfig->sSetting.bWakeUpEn )   //打开唤醒使能
    {
        u8Temp |= I2C_C1_WUEN_MASK;
    }
    if( pI2CConfig->sSetting.bI2CEn )       //打开 I2C 使能
    {
        u8Temp |= I2C_C1_IICEN_MASK;
    }
    pI2Cx->C1 = u8Temp;


}
```

1.2 IIC 起始位发送

函数名	I2C_Start
函数原形	I2C_Start(I2C_Type *pI2Cx)
功能描述	配置寄存器发送 I2C 起始位
输入参数	模块结构体 I2C_Type
输出参数	无
返回值	返回发送成功与否值
先决条件	无
函数使用实例	指向对应结构体，I2C_Start(I2C0)；

```
/*****************************************************************************//*!
*
* @简介  发送起始位
*
* @起始位未发送成功返回错误值
```

```
*****************************************************************************/

uint8_t I2C_Start(I2C_Type *pI2Cx)
{
    uint32_t u32ETMeout;
    uint8_t u8ErrorStatus;

    u32ETMeout = 0;
    u8ErrorStatus = 0x00;

    I2C_TxEnable(pI2Cx);      //将 I2C 配置成 TX 发送模式
    pI2Cx->C1 |= I2C_C1_MST_MASK;     //将 I2C 配置成主机模式

    //持续监测起始位有没有发送成功
    while( (!I2C_IsBusy(pI2Cx)) && ( u32ETMeout < I2C_WAIT_STATUS_ETMEOUT))
     {
        u32ETMeout ++;
     }

     if( u32ETMeout == I2C_WAIT_STATUS_ETMEOUT )
     {
        u8ErrorStatus |= I2C_ERROR_START_NO_BUSY_FLAG;
     }

     return u8ErrorStatus;
}
```

1.3 IIC 停止位发送

函数名	I2C_Stop
函数原形	I2C_Stop(I2C_Type *pI2Cx)
功能描述	配置寄存器发送 I2C 停止位
输入参数	模块结构体 I2C_Type
输出参数	无
返回值	返回发送成功与否值
先决条件	无
函数使用实例	指向对应结构体，I2C_Stop(I2C0);

```
/*****************************************************************************//*!
*
* @简介  发送 IIC 停止位
*
* @如果停止位没有发送成功返回错误值
*
*****************************************************************************/
```

```
uint8_t I2C_Stop(I2C_Type *pI2Cx)
{
    uint32_t u32ETMeout;
    uint8_t u8ErrorStatus;

    u32ETMeout = 0;
    u8ErrorStatus = 0x00;

    pI2Cx->C1 &= ~I2C_C1_MST_MASK;

  //持续监测 I2C 停止位是否发送成功
    while( (I2C_IsBusy(pI2Cx) ) && ( u32ETMeout < I2C_WAIT_STATUS_ETMEOUT))
     {
        u32ETMeout ++;
     }

     if( u32ETMeout == I2C_WAIT_STATUS_ETMEOUT )
     {
        u8ErrorStatus |= I2C_ERROR_STOP_BUSY_FLAG;
     }

     return u8ErrorStatus;
}
```

1.4 IIC 重复起始位发送

函数名	I2C_RepeatStart
函数原形	I2C_RepeatStart(I2C_Type *pI2Cx)
功能描述	配置寄存器发送 I2C 重复起始位
输入参数	模块结构体 I2C_Type
输出参数	无
返回值	返回发送成功与否值
先决条件	无
函数使用实例	指向对应结构体，I2C_RepeatStart(I2C0);

```
/*****************************************************************************//
*
* @简介  在发送完从机地址后的起始信号的发送
*
* @返回  如果起始位没有发送成功则返回错误值
*
*****************************************************************************/

uint8_t I2C_RepeatStart(I2C_Type *pI2Cx)
{
```

```
    uint32_t u32ETMeout;
    uint8_t u8ErrorStatus;

    u32ETMeout = 0;
    u8ErrorStatus = 0x00;

    pI2Cx->C1 |= I2C_C1_RSTA_MASK;

    while( (!I2C_IsBusy(I2C0) ) && ( u32ETMeout < I2C_WAIT_STATUS_ETMEOUT))
     {
        u32ETMeout ++;
     }

   //持续监测起始位是否发送成功
     if( u32ETMeout == I2C_WAIT_STATUS_ETMEOUT )
     {
        u8ErrorStatus |= I2C_ERROR_START_NO_BUSY_FLAG;
     }

     return u8ErrorStatus;
}
```

1.5 设置从机地址

对寄存器 I2Cx_A1 进行操作

字段	描述
7-1 RAD	范围从机地址 此字段包含 I2C 模块使用的从机地址。该字段被用在 7 位地址方案。任何非零值写给该位都能打开该寄存器。该寄存器的使用类似于在 A1 寄存器，但此外，该寄存器可以考虑范围匹配模式中的最大边界。
0 保留	该位保留 该位只读并有且仅有值 0

函数名	I2C_SetSlaveAddress
函数原形	I2C_SetSlaveAddress(I2C_Type*pI2Cx,uint16_t u16SlaveAddress)
功能描述	配置寄存器发送 I2C 从机地址
输入参数	模块结构体 I2C_Type，从机地址
输出参数	无
返回值	无
先决条件	无

函数使用实例	指向对应结构体， I2C_SetSlaveAddress(I2C0,SlaveAddress);

```
/*****************************************************************************//*!
*
* @简介 设置从机地址
*
* @无返回
*****************************************************************************/

void I2C_SetSlaveAddress(I2C_Type *pI2Cx,uint16_t u16SlaveAddress)
{
    /* 写入低八位地址 */
    pI2Cx->A1 = (uint8_t)u16SlaveAddress;

    /* 如果支持十位从机地址，写入高三位地址 */
    pI2Cx->C2 &= ~I2C_C2_AD_MASK;
    pI2Cx->C2 |= (uint8_t)(u16SlaveAddress>>8)&0x03;
}
```

1.6 关闭中断

函数名	I2C_IntDisable
函数原形	I2C_IntDisable(I2C_Type *pI2Cx)
功能描述	关闭 I2C 模块使能
输入参数	模块结构体 I2C_Type
输出参数	无
返回值	无
先决条件	无
函数使用实例	指向对应结构体，I2C_IntDisable(I2C0);

```
/*****************************************************************************//*!
*
* @简介 关闭 IIC 中断
*
* @无返回
*****************************************************************************/

void I2C_IntDisable(I2C_Type *pI2Cx)
{
    pI2Cx->C1 &= ~I2C_C1_IICIE_MASK;
    if(pI2Cx == I2C0)
    {
        NVIC_DisableIRQ(I2C0_IRQn);
    }
    #if defined(CPU_NV32M4)
    else if(pI2Cx == I2C1)
```

```
    {
        NVIC_DisableIRQ(I2C1_IRQn);
    }
    #endif
    else
    {

    }
}
```

1.7 打开中断

函数名	I2C_IntEnable
函数原形	I2C_IntEnable(I2C_Type *pI2Cx)
功能描述	打开 I2C 模块使能
输入参数	模块结构体 I2C_Type
输出参数	无
返回值	无
先决条件	无
函数使用实例	指向对应结构体， I2C_IntEnable(I2C0);

```
/*****************************************************************************//*!
*
* @简介 打开 IIC 中断
*
*
* @无返回
*****************************************************************************/

void I2C_IntEnable(I2C_Type *pI2Cx)
{
    pI2Cx->C1 |= I2C_C1_IICIE_MASK;
    if(pI2Cx == I2C0)
    {
        NVIC_EnableIRQ(I2C0_IRQn);
    }
    #if defined(CPU_NV32M4)
    else if(pI2Cx == I2C1)
    {
        NVIC_EnableIRQ(I2C1_IRQn);
    }
    #endif
    else
    {
```

```
    }
}
```

1.8 IIC 低超时时间设置

函数名	I2C_SetSCLLowETMeout
函数原形	I2C_SetSCLLowETMeout(I2C_Type *pI2Cx, uint16_t u16ETMeout)
功能描述	设置 I2C 低超时时间
输入参数	模块结构体 I2C_Type，低超时时间值
输出参数	无
返回值	无
先决条件	无
函数使用实例	指向对应结构体，I2C_SetSCLLowETMeout(I2C0, ETMeout);

```
/*****************************************************************************//*!
*
* @简介 给 ETMeout 寄存器赋值来改变低超时的时间
*
*
* @无返回
*****************************************************************************/

void I2C_SetSCLLowETMeout(I2C_Type *pI2Cx, uint16_t u16ETMeout)
{
    pI2Cx->SLTL = (uint8_t)u16ETMeout;
    pI2Cx->SLTH = (uint8_t)(u16ETMeout>>8);
}
```

1.9 关闭 IIC 系统时钟

函数名	I2C_Deinit
函数原形	I2C_Deinit(I2C_Type *pI2Cx)
功能描述	关闭 I2C 系统时钟
输入参数	模块结构体 I2C_Type
输出参数	无
返回值	无
先决条件	无
函数使用实例	指向对应结构体，I2C_Deinit(I2C0);

```
/*****************************************************************************//*!
*
* @简介 关闭 IIC 系统时钟
*
* @无返回
*****************************************************************************/
```

```
void I2C_Deinit(I2C_Type *pI2Cx)
{
    pI2Cx->C1 &= ~I2C_C1_IICEN_MASK;
#if defined(CPU_NV32)
    SIM->SCGC &= ~SIM_SCGC_IIC_MASK;
#elif defined(CPU_NV32M3)
    SIM->SCGC &= ~SIM_SCGC_IIC_MASK;
#elif defined(CPU_NV32M4)
    if(pI2Cx == I2C0)
    {
        SIM->SCGC &= ~SIM_SCGC_I2C0_MASK;
    }
    else
    {
        SIM->SCGC &= ~SIM_SCGC_I2C1_MASK;
    }
#endif
}
```

1.10 IIC 发送单字节数据

字段	描述
7—0 DATA	数据 在主机发送模式下，当数据被写入这个寄存器，数据传输开始。最高有效位首先发送。在主接收模式下，读这个寄存器会启动接收下一个字节数据。 注意：当要转出的主接收模式，读取数据寄存器之前要切换 I2C 模式，以防止主意外启动接收数据传输。 当你对此该寄存器写的时候 IIC 便会发送一个 byte 数据，读取该寄存器就是取出 IIC 接收的数据，IIC 模块在硬件内部分成接收模块和发送模块，但是它们公用一个寄存器名称

函数名	I2C_WriteOneByte
函数原形	I2C_WriteOneByte(I2C_Type *pI2Cx, uint8_t u8WrBuff)
功能描述	发送 I2C 数据
输入参数	模块结构体 I2C_Type，I2C 发送变量
输出参数	无
返回值	返回发送成功与否值
先决条件	无
函数使用实例	指向对应结构体，I2C_WriteOneByte(I2C0, WrBuff);

```
/*****************************************************************************//*!
*
* @简介 IIC 发送一个字节
*
* @字节未成功发送 返回错误值
```

```
    *
    ******************************************************************************/

uint8_t I2C_WriteOneByte(I2C_Type *pI2Cx, uint8_t u8WrBuff)
{
    uint32_t u32ETMeout;
    uint8_t u8ErrorStatus;

    u32ETMeout = 0;
    u8ErrorStatus = 0x00;

    while (((I2C_GetStatus(pI2Cx)&I2C_S_TCF_MASK) !=  I2C_S_TCF_MASK)
            && (u32ETMeout<I2C_WAIT_STATUS_ETMEOUT))
    {
        u32ETMeout ++;
    }
    if (u32ETMeout >= I2C_WAIT_STATUS_ETMEOUT)
    {
        u8ErrorStatus |= I2C_ERROR_NO_WAIT_TCF_FLAG;
        return u8ErrorStatus;
    }

    I2C_TxEnable(pI2Cx);                    //将 I2C 配置成 TX 输出模式
    I2C_WriteDataReg(pI2Cx,u8WrBuff);    //写数据寄存器发送数据

    u32ETMeout = 0;
    while (((I2C_GetStatus(pI2Cx)&I2C_S_IICIF_MASK) !=  I2C_S_IICIF_MASK)
            && (u32ETMeout<I2C_WAIT_STATUS_ETMEOUT))
    {
        u32ETMeout ++;
    }
    if (u32ETMeout >= I2C_WAIT_STATUS_ETMEOUT)
    {
        u8ErrorStatus |= I2C_ERROR_NO_WAIT_IICIF_FLAG;
        return u8ErrorStatus;
    }

    /* 清除中断标志位 */
    I2C_ClearStatus(pI2Cx,I2C_S_IICIF_MASK);
    if (I2C_GetStatus(pI2Cx) & I2C_S_RXAK_MASK)
    {
        u8ErrorStatus |= I2C_ERROR_NO_GET_ACK;
    }
    return u8ErrorStatus;
```

}

1.11 IIC 读取单字节数据

函数名	I2C_ReadOneByte
函数原形	I2C_ReadOneByte(I2C0, RdBuff,Ack)
功能描述	读取 I2C 数据
输入参数	模块结构体 I2C_Type，I2C 读取变量，是否发送 ACK 使能
输出参数	无
返回值	返回读取成功与否值
先决条件	无
函数使用实例	指向对应结构体，I2C_ReadOneByte(I2C0,RdBuff, Ack);

```
/*****************************************************************************//*!
*
* @简介  从 IIC 从机接收数据
*
*
* @如果收不到数据反馈错误值
*****************************************************************************/

uint8_t I2C_ReadOneByte(I2C_Type *pI2Cx, uint8_t *pRdBuff, uint8_t u8Ack)
{
    uint32_t u32ETMeout;
    uint8_t u8ErrorStatus;

    u32ETMeout = 0;
    u8ErrorStatus = 0x00;
    while (((I2C_GetStatus(pI2Cx)&I2C_S_TCF_MASK) !=  I2C_S_TCF_MASK)
            && (u32ETMeout<I2C_WAIT_STATUS_ETMEOUT))
    {
        u32ETMeout ++;
    }
    if (u32ETMeout >= I2C_WAIT_STATUS_ETMEOUT)
    {
        u8ErrorStatus |= I2C_ERROR_NO_WAIT_TCF_FLAG;
        return u8ErrorStatus;
    }

    I2C_RxEnable(pI2Cx);          //将 I2C 配置为输入模式

    if( u8Ack )
    {
        /* send out nack */
        I2C_SendNack(pI2Cx);   //发送 noack
```

```
    }
    else
    {
        /* send out ack */
        I2C_SendAck(pI2Cx);   //发送 ack
    }
    *pRdBuff = I2C_ReadDataReg(pI2Cx);       //将读到的数据存放到参数里头

    u32ETMeout = 0;
    //持续监测中断标志
    while (((I2C_GetStatus(pI2Cx)&I2C_S_IICIF_MASK) !=   I2C_S_IICIF_MASK)
            && (u32ETMeout<I2C_WAIT_STATUS_ETMEOUT))
    {
        u32ETMeout ++;
    }
    if (u32ETMeout >= I2C_WAIT_STATUS_ETMEOUT)
    {
        u8ErrorStatus |= I2C_ERROR_NO_WAIT_IICIF_FLAG;
        return u8ErrorStatus;
    }

    /* 清除 IIC 中断标志位 */
    I2C_ClearStatus(pI2Cx,I2C_S_IICIF_MASK);

    return u8ErrorStatus;
}
```

1.12 IIC 发送多个字节

函数名	I2C_MasterSendWait
函数原形	I2C_MasterSendWait(I2C_Type*pI2Cx,uint16_t u16SlaveAddress,uint8_t *pWrBuff,uint32_t u32Length)
功能描述	发送 I2C 数据
输入参数	模块结构体 I2C_Type，I2C 发送变量，发送个数，从机地址
输出参数	无
返回值	返回读取成功与否值
先决条件	无
函数使用实例	指向对应结构体， I2C_MasterSendWait(I2C0,SlaveAddress,WrBuff[0],Length);

```
/*****************************************************************************//*!
*
```

```
    * @简介  发送 IIC 数据并且等待发送完成.
    *
    * @参数 1   16 位从机地址
    * @参数 2   需要发送的缓存数组
    * @参数 3   发送字节的数目.
    *
    * @发送错误返回错误值
    *******************************************************************************/

uint8_t I2C_MasterSendWait(I2C_Type *pI2Cx,uint16_t u16SlaveAddress,uint8_t *pWrBuff,uint32_t u32Length)
{
    uint32_t i;
    uint8_t u8ErrorStatus;

     /* 发送起始信号 */
    u8ErrorStatus = I2C_Start(pI2Cx);

    /* 给从机发送从机地址 */
    u8ErrorStatus = I2C_WriteOneByte(pI2Cx,((uint8_t)u16SlaveAddress<<1) | I2C_WRITE);

    /* 如果没有错误发生，则继续发送字节*/
    if( u8ErrorStatus == I2C_ERROR_NULL )
    {
        for(i=0;i<u32Length;i++)
        {
            u8ErrorStatus = I2C_WriteOneByte(pI2Cx,pWrBuff[i]);
            if( u8ErrorStatus != I2C_ERROR_NULL )
            {
                return u8ErrorStatus;
            }
        }
     }

     /* 发送 I2C 停止位 */
     u8ErrorStatus = I2C_Stop(pI2Cx);

     return u8ErrorStatus;

}
```

1.13 IIC 接收多个字节

函数名	I2C_MasterReadWait
函数原形	I2C_MasterReadWait(I2C_Type*pI2Cx,uint16_t

	u16SlaveAddress,uint8_t *pRdBuff,uint32_t u32Length)
功能描述	接收 I2C 数据
输入参数	模块结构体 I2C_Type，接收个数，从机地址
输出参数	I2C 接收变量
返回值	返回读取不到数据的个数
先决条件	无
函数使用实例	指向对应结构体 I2C_MasterReadWait(I2C0,SlaveAddress,RdBuff,Length);

```
/*****************************************************************************//*!
*
* @简介 从 IIC 从机读取数据并且等待全部读取完毕
*
* @参数 1  从机地址
* @参数 2  接收数据的指针*pRdBuff
* @参数 3  发送数据的字节数.
*
* @如果接收不到数据 返回读取错误的个数
*****************************************************************************/

uint8_t I2C_MasterReadWait(I2C_Type *pI2Cx,uint16_t u16SlaveAddress,uint8_t *pRdBuff,uint32_t u32Length)
{
    uint32_t i;
    uint8_t u8ErrorStatus;

    /* 发送起始信号 */
    u8ErrorStatus = I2C_Start(pI2Cx);

    /* 发送器件地址给从机 */
    u8ErrorStatus = I2C_WriteOneByte(pI2Cx,((uint8_t)u16SlaveAddress<<1) | I2C_READ);

    /* 如果没有发生错误将持续接收数据 */
        I2C_ReadOneByte(pI2Cx,&pRdBuff[0],I2C_SEND_ACK);

    if( u8ErrorStatus == I2C_ERROR_NULL )
    {
        for(i=0;i<u32Length-1;i++)
        {
            u8ErrorStatus = I2C_ReadOneByte(pI2Cx,&pRdBuff[i],I2C_SEND_ACK);
            if( u8ErrorStatus != I2C_ERROR_NULL )
            {
                return u8ErrorStatus;
            }
        }
        u8ErrorStatus = I2C_ReadOneByte(pI2Cx,&pRdBuff[i],I2C_SEND_NACK);
```

```
    }
    /* 发送停止信号 */
    u8ErrorStatus = I2C_Stop(pI2Cx);

    return u8ErrorStatus;

}
```

第二章 样例程序

2.1 IIC 主机模式发送和接收数据

```
/*****************************************************************************************
*
* @简介   初始化一个 64 位的数组并存入数据，将存入的数据通过 IIC 发送给 IIC 从机设备，然后从从机
*         设备里再读取刚刚发过去的数据，检查反馈回来的数据并通过 UART1 打印出来。
*
*****************************************************************************************/

#include "common.h"
#include "ics.h"
#include "rtc.h"
#include "uart.h"
#include "i2c.h"
#include "i2C_app.h"
#include "sysinit.h"

#define I2C_READ_DATA_LENGTH      64
#define I2C_SLAVE_ADDRESS1        0x50

uint8_t u8I2C_SendBuff[64];
uint8_t u8I2C_ReceiveBuff[64];
uint32_t u32I2C_ReceiveLength;

int main (void)
{
    uint8_t             u8I2C_ErrorStatus;
    I2C_ConfigType    sI2C_Config = {0};
    volatile uint32_t i;

    sysinit();
```

```
    printf("\nRunning the I2C_MasterInt_demo project.\r\n");
    LED0_Init();
    LED2_Init();

    UART_WaitTxComplete(TERM_PORT);

    /* initialize I2C global variable and call back function*/
    I2C_InitGlobalVariable( );

    /* 对 IIC 结构体的各个成员参数进行配置 */
    sI2C_Config.u16Slt = 0;                 //设置低超时 scl 低电平保持时间为 0
    sI2C_Config.u16F = 0x1F;                //设置分频系数
    sI2C_Config.sSetting.bIntEn = 1;        //打开中断使能
    sI2C_Config.sSetting.bI2CEn = 1;        //打开 IIC 使能

    I2C_Init(I2C0,&sI2C_Config );           //初始化 IIC 模块


    for(i=0;i<64;i++)
    {
        u8I2C_SendBuff[i] = i;      //给发送数据的数组里存放数据
}

    while(1)
    {
        /*反馈接收字节的个数*/
        u32I2C_ReceiveLength = I2C_MasterCheckRead(&u8I2C_ReceiveBuff[0]);
        if( !I2C_IsBusy(I2C0) )
        {
            /*从从机接收数据 */
            printf("start to read data from slave!\r\n");
            I2C_MasterRead(I2C_SLAVE_ADDRESS1,I2C_READ_DATA_LENGTH);
        }
        if( u32I2C_ReceiveLength == 0 )
        {
            printf("don't receive any data from slave!\n");
        }
        else
        {
            for(i=0;i<u32I2C_ReceiveLength;i++)
            {
                if( (i%8) == 0 )
                {
                    printf("\r\n");
```

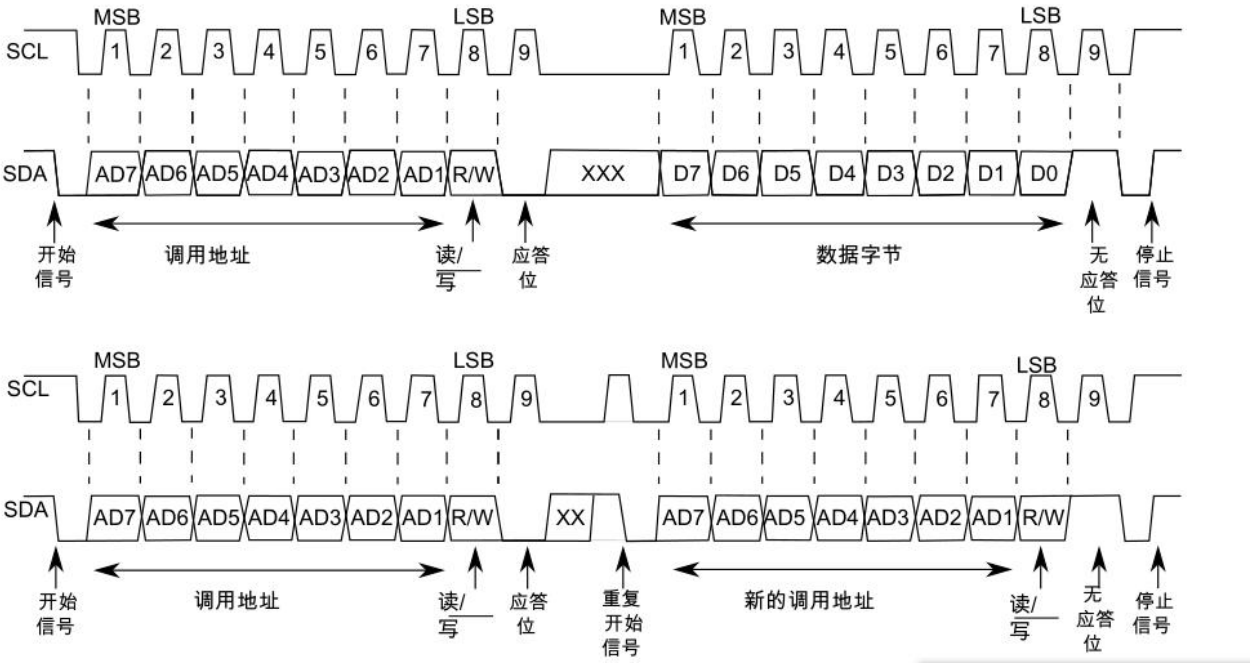
```
                }
                printf("0x%x,",u8I2C_ReceiveBuff[i]);

            }
            if( u32I2C_ReceiveLength >= I2C_READ_DATA_LENGTH )
            {
                printf("\r\nreceived all required data!\r\n");
                printf("start to send data to slave!\r\n");
             u8I2C_SendBuff[0]++;

                for(i=1;i<64;i++)
                {
                    u8I2C_SendBuff[i] = i+u8I2C_SendBuff[0];
                }

                /* 反馈发送失败的数目 */
                u8I2C_ErrorStatus = I2C_MasterSend(I2C_SLAVE_ADDRESS1,&u8I2C_SendBuff[0],64);

                if(   u8I2C_ErrorStatus != I2C_ERROR_NULL )
                {
                    printf("I2C transfer status:0x%x\r\n",u8I2C_ErrorStatus);
                }
            }
        }
        for(i=0;i<0xfffff;i++);
    }

}
```

第三章 I2C 时序分析图

I2C 总线系统采用串行数据线路(SDA)和串行时钟线路(SCL)进行数据传输。连接至 I2C 总线系统的所有器件都必须具有开漏或集电极开路输出。使用外部上拉电阻在这两条线路上执行逻辑和(AND)功能。电阻值取决于系统。正常情况下，一个标准的通信实例由四部分组成：

1. 开始信号
2. 从机地址发送
3. 数据传输
4. 停止信号

不得将停止信号与 CPU 停止指令混淆。下图对 I2C 总线系统通信进行了说明。


MSB
LSB
SCL
SDA
AD7
AD6
AD5
AD4
AD3
AD2
AD1
R/W
XXX
D7
D6
D5
D4
D3
D2
D1
D0
开始信号
调用地址
读/写
应答位
数据字节
无应答位
停止信号
XX
重复开始信号
新的调用地址