IIC 驱动编写指导手册

编写：贺敏 2015 年 3 月 23 日
Review：_年_月_日
审阅:罗侍田

1. 贡献者列表

DJYOS 开发团队。

2. 概述

DJYOS 的 DjyBus 总线模型为 IIC、SPI 之类的器件提供统一的访问接口，IICBUS 模块是 DjyBus 模块的一个子模块，为 IIC 器件提供统一的编程接口，实现通信协议层与器件层的分离。也标准化了 IIC 总线和 Device 驱动接口，本手册指导驱动工程师编写 IIC 的接口程序。

IIC 总线使用手册，请参见《都江堰操作系统用户手册》。

局限性：DJYOSV1.1.1 版本的 IIC 驱动只提供主器件功能。

3. 总线资源结构

IIC 通信协议是一种总线通信方式，这意味着一条总线上可以挂多个符合总线通信协议的设备，DjyBus 资源组织结构就是符合这样一种物理的连接方式。在如图 2-1资源组织结构图，总线类型“IIC”、第 n 条总线“IICn”、第 n 条总线上面的设备“Devn”，它们都是 DjyBus 资源树上面的资源节点，每次向总线“IICn”上面增加一个设备，便向资源树上面增加了一个资源节点，它是“IICn”的子节点。

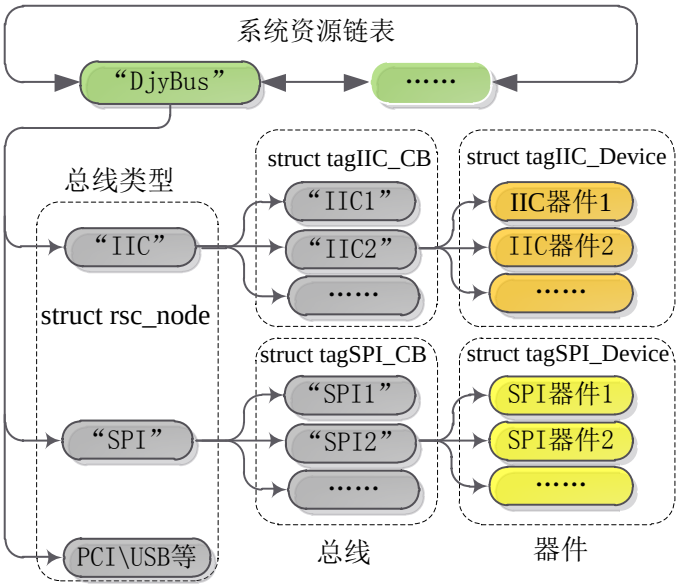

图 2-1 总线资源结构

4. 准备工作

在编写 IIC 器件驱动前，建议完成必要的准备工作，如：

1、 认真阅读器件手册，了解通信协议、参数、操作流程等内容；

2、 熟悉 iicbus.h 头文件中提供的 API，懂得参数的使用方法；

3、 阅读 IIC 总线协议文档，熟练掌握 IIC 总线。

5. IIC 总线驱动接口

5.1. 驱动架构

IICBus 是 DjyBus 模块的一个子模块，其结构如图 4-1所示，它为 IIC 器件提供标准的、一致的应用程序编程接口，并且规范了硬件驱动接口。驱动接口分为总线控制器接口和 IIC 器件接口两部分，驱动的重点是总线控制器，而器件接口实际上就是配置一下该器件的物理参数。

建议文件路径：在 eclipse 工程中的链接目录如下，如果是导入官方提供的 example 工程，那么该目录已经建立，在硬盘中添加文件后，只需要刷新工程即会自动添加进工程中。

src->OS_code->bsp->cpudrv->src->cpu_peri_iic.c。

相应的头文件目录为：

src->OS_code->bsp->cpudrv-> src->cpu_peri_iic.h。

在文件系统（硬盘）中的目录结构是：

djysrc\bsp\cpudrv\cpu_name\src\cpu_peri_iic.c。

djysrc\bsp\cpudrv\cpu_name\include\cpu_peri_iic.h。

根据以上命名，可以在 DJYOS 官方提供的代码中，找到大量范例。

IIC 驱动程序编写重点有：

1、 初始化 IIC 控制器，并且把 IIC 总线添加到 DjyBus 上。

2、 实现图 4-1中的 5 个回调函数（哪些需要实现，参考后续章节）。

3、 如果采用中断方式，须编写中断服务函数（实际上也是为 4 个回调函数服务）

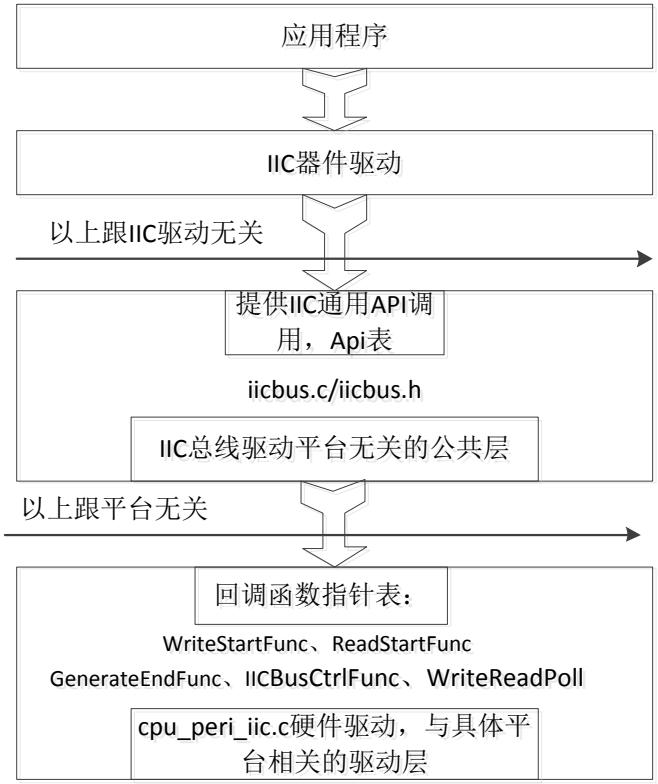

图 4-1 IICBus 总线驱动架构

5.2. 初始化函数

5.2.1. step1：初始化硬件

1、 IIC 控制器硬件的初始化，包括传输速度、IO 配置等；
2、 挂载 IIC 中断到中断系统，并配置中断类型，如配置为异步信号（若只采用轮询方式，则此功能可省略）；

5.2.2. step2：初始化参数结构体

添加 IIC 总线的参数类型为 struct tagIIC_Param，由函数 IIC_BusAdd 或 IIC_BusAdd_s 完成对 IIC 总线控制块 ICB 的初始化和添加 IICn 总线节点到 DjyBus 资源树。

代码 4-1 IIC 参数结构体

```
//IIC初始化参数结构体
struct tagIIC_Param
{
    char              *BusName;              //总线名称，如IIC1
    u8                *IICBuf;               //总线缓冲区指针
    u32               IICBufLen;             //总线缓冲区大小，字节
    ptu32_t           SpecificFlag;          //指定标记，如IIC寄存器基址
    WriteReadPoll     pWriteReadPoll;        //轮询或中断未开时使用
    WriteStartFunc    pGenerateWriteStart;   //写过程启动
    ReadStartFunc     pGenerateReadStart;    //读过程启动
    GenerateEndFunc   pGenerateEnd;          //结束通信
    IICBusCtrlFunc    pBusCtrl;              //控制函数
};
```

根据 IIC 总线通信的特点可知，无论 IIC 总线主设备有多少从设备，在同一时刻，IIC 主设备与从设备的通信只能单一单向，即单点通信，单向通信，因此，接收与发送使用同一个缓冲区。

IIC 参数结构体的回调函数参数的原型如代码 4-2所示，其中 PrivateTag 就是结构体中 IIC 的私有指定标签，即 IICn 寄存器基址。

代码 4-2 IIC 回调函数类型定义

```
typedef bool_t (*WriteStartFunc)(ptu32_t  SpecificFlag,u8 DevAddr,
                                 u32 MemAddr,u8 MenAddrLen, u32 Length,
                                 struct tagSemaphoreLCB *IIC_BusSemp);
typedef bool_t (*ReadStartFunc)(ptu32_t  SpecificFlag,u8 DevAddr,
                                u32 MemAddr,u8 MemAddrLen, u32 Length,
                                struct tagSemaphoreLCB *IIC_BusSemp);
typedef void (*GenerateEndFunc)(ptu32_t  SpecificFlag);
typedef s32 (*IICBusCtrlFunc)(ptu32_t SpecificFlag,u32 cmd,
                              ptu32_t data1,ptu32_t data2);
typedef bool_t (*WriteReadPoll)(ptu32_t  SpecificFlag,u8 DevAddr,
                                u32 MemAddr,u8 MenAddrLen,u8* Buf,
                                u32 Length,u8 WrRdFlag);
```

5.2.3. step3：挂载总线

有多少 IIC 总线是由具体的平台决定，因此，增加 IIC 总线到 DjyBus 上是由总线驱动程序员完成，成功添加的“IICn”节点会成为“IIC”节点的子节点。

增加 IIC 总线的 API 函数可以调用 IIC_BusAdd 函数或 IIC_BusAdd_s 函数，两者的区别在于，IIC_BusAdd 只需调用者提供已初始化好的参数结构体 struct tagIIC_Param，而后者更需要提供 struct tagIIC_CB 结构体控制块（建议定义为静态变量）。

5.3. 回调函数

5.3.1. 轮询函数

如果采用轮询方式收发，5 个回调函数中只需要实现本函数和控制函数，其他指针置为 NULL 即可。

轮询函数使用场合：

1、 收发方式被设为轮询方式，则总是用轮询函数收发数据。默认值为中断方式，可调用 IIC_BusCtrl 函数设为轮询方式。

2、 在禁止调度（即禁止异步信号中断）期间，强制使用轮询方式。

3、 pGenerateReadStart==NULL，则使用轮询方式接收；pGenerateWriteStart==NULL，则使用轮询方式发送。

4、 系统初始化未完成，多事件调度尚未启动期间。

如果使用中断方式收发，且不考虑在 2~4 三种情况下收发数据，则无须实现本函数，WriteReadPoll 指针设为 NULL 即可。

回调函数说明如下：

```
typedef bool_t (*WriteReadPoll)(ptu32_t  SpecificFlag,u8 DevAddr,u32
                                MemAddr, u8 MenAddrLen,u8* Buf, u32
                                Length,u8 WrRdFlag);
```

参数：

SpecificFlag：IIC 控制器寄存器基址。

DevAddr：设备地址，低七位有效。

MemAddr：设备内部地址，若为存储设备，则为存储地址。

MemAddrLen：设备内部地址字节数。

Buf：数据缓冲区。

Length：Buf 中数据字节数。

WrRdFlag：读写标志，0 为写，1 为读。

返回：true，执行成功；false，执行失败。

说明：轮询函数在执行前必须关闭中断，否则将执行失败。轮询函数示例代码如代码 4-3所示。

代码 4-3 轮询函数示例

```
Bool_t __IIC_WriteReadPoll((tagI2CReg *reg,u8 DevAddr,u32 MemAddr,\
                        u8 MemAddrLen,u8* Buf, u32 Length,u8 WrRdFlag)
{
    __IIC_IntDisable(reg);
    if(WrRdFlag == CN_IIC_WRITE_FLAG)   //写
    {
        if(Length == __IIC_WritePoll(reg,DevAddr,MemAddr,
                                     MemAddrLen,Buf,Length))
            return true;
        else
            return false;
    }
    else                                //读
    {
        if(Length == __IIC_ReadPoll(reg,DevAddr,MemAddr,
                                    MemAddrLen,Buf,Length))
            return true;
        else
            return false;
```

```
    }
}
```

5.3.2. 启动发送

启动发送函数（WriteStartFunc）是为中断方式收发服务的，轮询方式不需要，置为 NULL 即可。

启动发送的回调函数 WriteStartFunc 完成了发送数据时 IIC 的启动时序，其执行的流程为 start---->发送器件地址---->发送内部地址，并开启中断，然后返回。对应在时序上，如图 4-2所示。

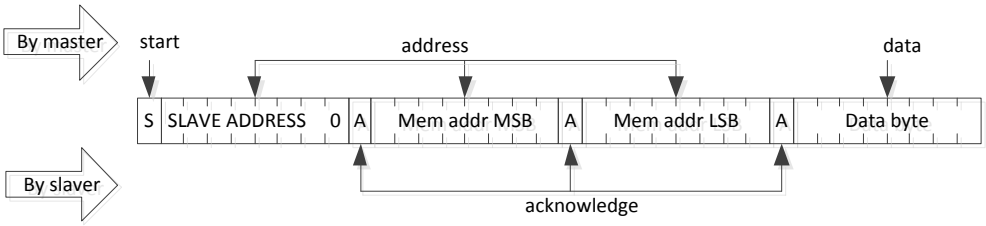

图 4-2 启动发送

如图中所示，写时序是以主控制器发送 start 信号为起始条件，紧接最低位为 0 从器件地址表示写操作，当收到 ACK 信号后会将从器件的存储地址（图中地址为 2 字节，具体多少字节视情况而定）发送到总线，最后发送正式的正文。

回调函数说明如下：

```
typedef bool_t (*WriteStartFunc)(ptu32_t  SpecificFlag,u8 DevAddr,\
                                 u32 MemAddr,u8 MenAddrLen, u32 Length,\
                                 struct tagSemaphoreLCB *IIC_BusSemp);
```

功能：产生 IIC 写数据时序，并发送内部地址

参数：

SpecificFlag，寄存器基址

DevAddr，器件地址的前 7 比特，已将内部地址所占的 bit 位更新，该函数需将该地址左移一位增加最后一位读/写比特；

MemAddr，存储器内部地址，即发送到总线上的地址，该地址未包含放在设备地址上的比特位；

MenAddrLen，存储器内部地址的长度，字节单位，未包含在设备地址里面的比特位；

Length，发送的数据总量，最后一个字节发送完时，需产生停止时序，并释放信号量；

IIC_BusSemp，总线控制信号量，发送完数据后需底层驱动释放；

返回：true，启动发送过程正确，false，发生错误

5.3.3. 启动接收

启动接收函数（ReadStartFunc）是为中断方式收发服务的，轮询方式不需要，置为 NULL 即可。

启动发送函数（ReadStartFunc）是为中断方式收发服务的，轮询方式不需要，置为 NULL 即可。

启动接收的回调函数 ReadStartFunc 主要完成了 IIC 时序上面读数据时的总线控制，读时序的时序控制过程如图 4-3所示。该函数依次实现了写 start---->器件地址（写）---->写存储地址---->start（或者 restart）---->器件地址（读）的时序过程。在启动接收时序正确完成后，需使能中断（若不使用中断，则需配置接收到数据 pop 的事件），并配置回复 ACK，在中断中接收从器件发送的数据。

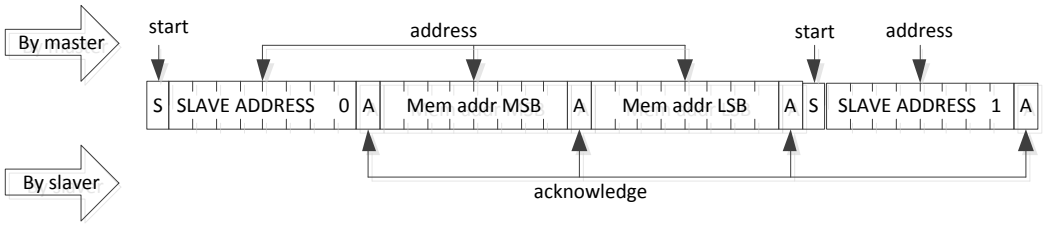

图 4-3 启动接收

如图所示的时序图中，有两个 start 时序，可以通过配置 repeated 来重新启动一次新的时序，而不产生停止位。

回调函数说明如下：

```
typedef bool_t (*ReadStartFunc)(ptu32_t  SpecificFlag,u8 DevAddr,\
                                u32 MemAddr,u8 MemAddrLen, u32 Length,\
                                struct tagSemaphoreLCB *IIC_BusSemp);
```

功能：完成读时序的启动，并使能中断

参数：

SpecificFlag，寄存器基址

DevAddr，从器件地址的高七比特（同__IIC_GenerateWriteStart 参数说明）

MemAddr，存储器件的内部地址（同__IIC_GenerateWriteStart 参数说明）

MemAddrLen，存储器件地址长度，字节单位（同__IIC_GenerateWriteStart 参数说明）

Length，接收的数据总量，接收数据的倒数第一字节，即 count-1，停止产生 ACK 信号，当接收的字节数为 count 时，产生停止时序，并释放信号量 iic_buf_semp;

IIC_BusSemp，发送完成的缓冲区信号量，告知上层，本次发送已经完成。

返回：TRUE，启动读时序成功，FALSE 失败

5.3.4. 结束传输

结束传输函数（GenerateEndFunc）是为中断方式收发服务的，轮询方式不需要，置为 NULL 即可。

结束传输的回调函数__IIC_GenerateEnd 主要用于停止当前正在进行的传输，特别是在发生超时传输时，用于停止本次发送或接收，实际上，该函数调用了产生停止时序的函数，使 IIC 主器件停止本帧数据的传输。启动和停止时序如图 4-4所示。

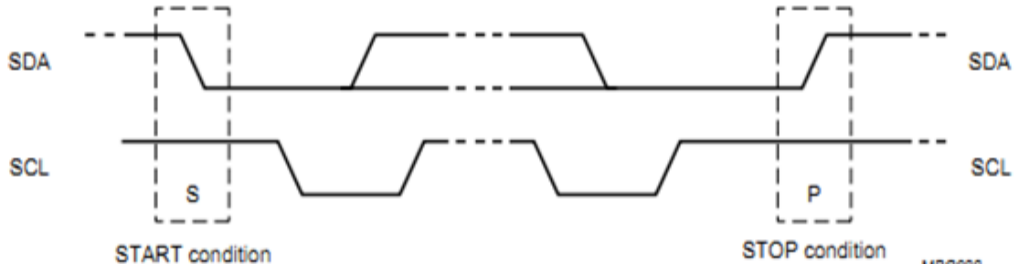

图 4-4 IIC 启动和停止时序

5.3.5. 控制函数

目前，控制 IIC 的底层驱动只需要实现对 IIC 总线传输时钟的控制即可，相对较为简单，此处不作详细说明，请参看源码 cpu_peri_iic.c。

5.4. 中断服务函数

5.4.1. 中断实现过程

如果使用轮询方式实现驱动，则无须编写中断服务函数。

相比轮询通信方式，中断方式的执行效率更高，对 CPU 的消耗更少。由于各种控制器五花八门，因此，中断的具体实现方式也不同。但是基于 DjyBus 设计的 IIC 中断方式接收与发送数据大体的框架和流程基本相似。

IIC 模块对 IIC 总线驱动程序在中断中需要完成的功能作如下要求：

第一， 根据中断线或中断标志判断使用的 IIC 控制块和静态变量参数；

第二， 发送数据中断时，调用 API 函数 IIC_ReadPort 读取需要发送的参数，并将静态变量计数器 IntParam->TransCount 递增；

第三， 若发送结束，即 IIC_ReadPort 读不到数据，且 IntParam->TransCount = IntParam->TransTotalLen，则需要产生停止时序和释放信号量；

第四， 若为接收数据中断，则需调用 IIC_WritePort 将接收到的数据写入缓冲区，并将计数器 IntParam->TransCount 递增，接收到倒数第二个数据时，还需配置寄存器不发送 ACK 信号；

第五， 若接收到所有数据，则需产生停止时序和释放信号量。

下面以 p1020 的 IIC 控制器连接铁电为例，简要讲解一下中断服务函数中的流程。

在中断服务函数内部，通过寄存器判断是发送中断还是接收中断，如图 4-5所示。发送中断时，需要判断是否收到 IIC 从器件 ACK 信号，然后读简易缓冲区中的数据，并发送之；若缓冲区中为空，判断发送的总量 count 是否为零，若是，则表示该帧数据已经全部发送完毕，需产生停止时序，释放信号量 IntParam->pDrvPostSemp，该信号量是__IIC_GenerateWriteStart 的参数。

在接收中断中，需要判断是否为倒数第二个接收的字节，若是，需要配置控制寄存器不发送 ACK 信号，使控制器接收到倒数第二个字节时不发送 ACK 信号，用于通知从设备接收的数据足够。接收到数据后调用 IIC_PortWrite，该函数将接收到的数据保存到用户缓冲区。若接收到最后一个字节数据，则产生停止时序，并释放信号量 IntParam->pDrvPostSemp，本次接收完成。

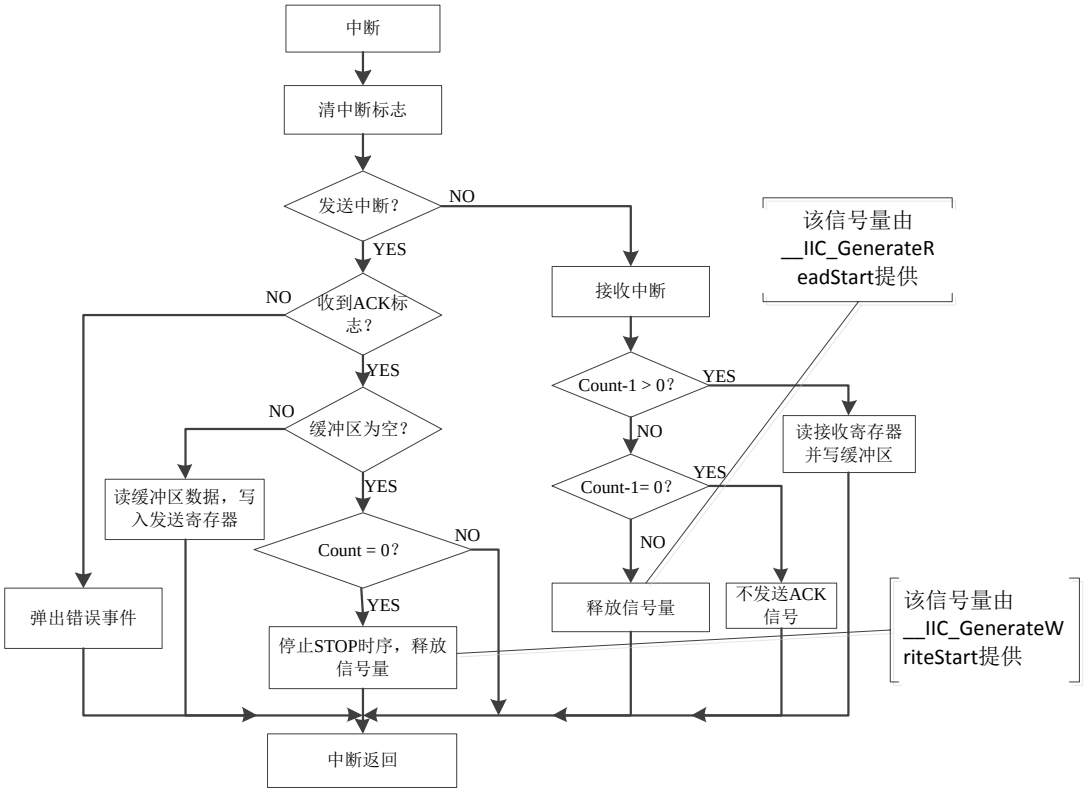

图 4-5 中断方式接收发送流程图

5.4.2. 注意事项

使用中断方式实现 IIC 主设备与从设备通信，需要注意以下几点：

1、 发送中断不仅要判断中断标志位，清标志位，同时还需判断是否接收到 ACK 信号；

2、 正常的发送结束时，IIC_PortRead 读到数据为 0，计数值 IntParam->TransCount 与 IntParam->TransTotalLen 应该相等，若不等，则可能出现逻辑错误；

3、 读数据的倒数第二个字节时，需停止时序，因为，此时数据已经发送到总线上面；
4、 通信结束后，需释放信号量和停止时序。

5.5. 移植建议

由于大部分的 IIC 控制器的设计基本相似，因此，BSP 程序人员可采取下面的步骤快速的完成 DJYOS 驱动架构下 IIC 底层驱动的开发。

1、 拷贝其他工程已测试通过的 IIC 驱动文件 cpu_peri_iic.c/cpu_peri_iic.h；
2、 添加 IIC 的中断号到 critical.c 文件下面 tg_IntUsed 数组；
3、 修改 `cpu_peri_iic.c/cpu_peri_iic.h` 中与具体 `IIC` 寄存器相关的部分；
4、 回调函数的具体实现和中断收发数据。

测试驱动前，确保已经调用初始化函数 `ModuleInit_DjyBus(0)`和 `ModuleInit_IICBus(0)`。

6. IIC 器件驱动接口

建议将器件驱动的存放目录为 djysrc\bsp\chip\xxx，其中，xxx 是具体芯片的文件夹名称。

IIC 总线初始化完成后，添加一个器件到总线上的过程，非常简单，就是初始化一下该器件的寻址特性参数，然后调用 IIC_DevAdd_s 或 IIC_DevAdd 函数把器件添加到总线上即可。需配置的参数，都在 iicbus.h 文件中定义的 struct tagIIC_Device 中描述。struct tagIIC_Device 结构定义如下：

代码 5-1 IIC 器件结构体

```
//IIC总线器件结构体
struct tagIIC_Device
{
    struct tagRscNode DevNode;
    u8 DevAddr;                 //七位的器件地址
    u8 BitOfMemAddrInDevAddr;//器件地址中内部地址所占比特位数
    u8 BitOfMemAddr;  //器件内部地址寻址总bit数，包含了BitOfMemAddrInDevAddr
};
```

对 tagIIC_Device 结构体作如下详细说明：

- 器件地址 DevAddr 是七个比特地址，如 0x50，在总线上体现的地址为 0x80/0x81；
- BitOfMemAddrInDevAddr 是指 dev_addr 的低三个比特中，有多少个 bit 用于器件内部存储空间寻址，取值范围：0~3。
- BitOfMemAddr 表示被操作的器件内部地址的总位数，包含器件地址上的比特位；

举例说明：存储大小为 128K 的铁电，器件地址为 0x50，页大小为 64K，则地址范围为 0x00000 ~ 0x1FFFF，若存储地址占用器件地址的 1 个比特，则 dev_addr 为 0x50，`BitOfMemAddrInDevAddr` 为 1，BitOfMemAddr 为 17，构成了 128K 的寻址空间。

6.1. 初始化过程

添加器件到总线的过程就是将器件节点挂到相应的“IICn”总线节点的过程，同时，配置好相应的总线通信参数。

1、 若使用 IIC_DevAdd_s 挂载器件，定义 static struct tagIIC_Device 类型的静态变量；
2、 若使用 IIC_DevAdd_s 挂载器件，初始化该变量的各成员；
3、 调用 IIC_DevAdd_s 或 IIC_DevAdd 添加设备到总线节点。
4、 调用 IIC_BusCtrl 设置总线参数

IIC_DevAdd_s 或 IIC_DevAdd 都可以把器件添加到总线上，但两者是有区别的：

1、 使用 IIC_DevAdd_s 的话，你需要自己准备 struct tagIIC_Device 结构，并且自行初始化，特

别是，当操作系统的 spibus 模块被修改导致该结构的定义发生变化时，器件驱动程序也需要修改。

2、 使用 IIC_DevAdd_s 的好处是，该结构无须动态分配，符合像 OSEK 之类的严谨规范。

3、 使用 IIC_DevAdd 的好处是，驱动程序非常简单。

下面用 FreeScale 公司的 CRTOUCH 触摸芯片为例说明添加设备过程。如代码 5-2所示，将 CRTOUCH 芯片添加到总线“IIC0”，并命名为“IIC_Dev_CRTCH”，并配置了总线速度和采用轮询通信方式。

代码 5-2 添加 IIC 设备实例

```
ptu32_t CRT_Init(ptu32_t para)
{
    bool_t result = false;
    static struct tagIIC_Device s_CRT_Dev;

    //初始化IIC设备结构体
    s_CRT_Dev.DevAddr                   = CRT_ADDRESS;
    s_CRT_Dev.BitOfMemAddr              = 8;
    s_CRT_Dev.BitOfMemAddrInDevAddr     = 0;

    //添加CRTCH到IIC0总线
    if(NULL != IIC_DevAdd_s("IIC0","IIC_Dev_CRTCH",&s_CRT_Dev))
    {
        ps_CRT_Dev = &s_CRT_Dev;
        IIC_BusCtrl(ps_CRT_Dev,CN_IIC_SET_CLK,CRT_CLK_FRE,0);
        IIC_BusCtrl(ps_CRT_Dev,CN_IIC_SET_POLL,0,0);
        result = true;
    }

    return result;
}
```

7. 访问器件

器件装载到 IIC 总线之后，可以通过访问 IIC 总线实现访问器件，具体就是调用 iicbus.h 提供的 API 函数 IIC_Write（）和 IIC_Read（）。