i2c总线

概念：

I2C总线是一种由 PHILIPS 公司开发的两线式串行总线，一根是双向的数据线SDA，另一根是时钟线SCL，用于连接微控制器及其外围设备。

三种信号

```
起始信号：当scl为高电平的时候，sda从高到低的跳变
停止信号：当scl为高电平的时候，sda从低到高的跳变
应答信号：在第九个时钟周期的时候，将sda拉为低电平即可
```

写时序

```
start+(slave 7bit  w 1bit)+ack+reg+ack+data+ack+stop
```

读时序

```
start+(slave 7bit  w 1bit)+ack+reg+ack+
    start+(slave 7bit  r 1bit)+ack+data+NO ack+stop
```

i2c特点：

```
i2c是一个半双工的、同步的、串行的总线协议
```

6.i2c的速率

```
100Kbps   400Kbps   3.4Mbps
    低速       全速       高速
```

I2C设备节点配置介绍

i2c0:

```
i2c0: i2c@11007000 {
        compatible = "mediatek,i2c";//设置属性
        id = <0>;                   //设置i2c标志（0-6）
        reg = <0 0x11007000 0 0x1000//设置寄存器大小
            <0 0x11000100 0 0x80>;
        interrupts = <GIC_SPI 84 IRQ_TYPE_LEVEL_LOW>;//中断，管脚84
        clocks = <&infracfg_ao CLK_INFRA_I2C0>, <&infracfg_ao
CLK_INFRA_AP_DMA>;//使能
        clock-names = "main", "dma";//引用名字，可以随意
        clock-div = <5>;
        aed = <0x1f>;
    };
```

i2c1:

```
i2c1: i2c@11008000 {
        compatible = "mediatek,i2c";
        id = <1>;
        reg = <0 0x11008000 0 0x1000>,
            <0 0x11000180 0 0x80>;
        interrupts = <GIC_SPI 85 IRQ_TYPE_LEVEL_LOW>;
        clocks = <&infracfg_ao CLK_INFRA_I2C1>, <&infracfg_ao CLK_INFRA_AP_DMA>;
        clock-names = "main", "dma";
        clock-div = <5>;
        aed = <0x1f>;
    };
```

i2c2:

```
i2c2: i2c@11009000 {
        compatible = "mediatek,i2c";
        id = <2>;
        reg = <0 0x11009000 0 0x1000>,
            <0 0x11000200 0 0x80>;
        interrupts = <GIC_SPI 86 IRQ_TYPE_LEVEL_LOW>;
        clocks = <&infracfg_ao CLK_INFRA_I2C2>, <&infracfg_ao CLK_INFRA_AP_DMA>;
        clock-names = "main", "dma";
        clock-div = <5>;
        aed = <0x1f>;
    };
```

i2c3:

```
i2c3: i2c@1100f000 {
        compatible = "mediatek,i2c";
        id = <3>;
        reg = <0 0x1100f000 0 0x1000>,
            <0 0x11000280 0 0x80>;
        interrupts = <GIC_SPI 87 IRQ_TYPE_LEVEL_LOW>;
        clocks = <&infracfg_ao CLK_INFRA_I2C3>, <&infracfg_ao CLK_INFRA_AP_DMA>;
        clock-names = "main", "dma";
        clock-div = <5>;
        aed = <0x1f>;
    };
```

i2c4:

```
i2c4: i2c@11011000 {
        compatible = "mediatek,i2c";
        id = <4>;
        reg = <0 0x11011000 0 0x1000>,
            <0 0x11000300 0 0x80>;
        interrupts = <GIC_SPI 88 IRQ_TYPE_LEVEL_LOW>;
        clocks = <&infracfg_ao CLK_INFRA_I2C4>, <&infracfg_ao CLK_INFRA_AP_DMA>;
        clock-names = "main", "dma";
        clock-div = <5>;
        aed = <0x1f>;
    };
```

i2c5:

```
  i2c5: i2c@11016000 {
        compatible = "mediatek,i2c";
        id = <5>;
        reg = <0 0x11016000 0 0x1000>,
            <0 0x11000380 0 0x80>;
        interrupts = <GIC_SPI 130 IRQ_TYPE_LEVEL_LOW>;
        clocks = <&infracfg_ao CLK_INFRA_I2C5>, <&infracfg_ao CLK_INFRA_AP_DMA>;
        clock-names = "main", "dma";
        clock-div = <5>;
        aed = <0x1f>;
    };
```

设备驱动常用API

```
1.#define i2c_add_driver(driver)  i2c_register_driver(THIS_MODULE, driver)
int i2c_register_driver(struct module *, struct i2c_driver *);
参数：driver（已经初始化的i2c_driver结构体指针）
功能：注册I2C设备驱动
返回：返回0则注册成功

2.void i2c_del_driver(struct i2c_driver *driver)
参数：driver（已经初始化的i2c_driver结构体指针）
功能：注销I2C设备驱动

3.int i2c_master_send(const struct i2c_client *client, const char *buf,
                int count)
参数：client（内核创建的i2c_client结构体指针），buf（要发送给从设备的数据缓冲区指针），count（发送的字节数量）
功能：主设备向从设备发送数据
返回:返回成功发送的字节数，失败则返回值小于0
注：该API实现中，已用i2c_msg结构体获取了i2c_client结构体中从设备地址和写方        向，使用时无需考虑从设备地址和读写方向

4.int i2c_master_recv(const struct i2c_client *client, char *buf,int count)
参数：client（内核创建的i2c_client结构体指针），buf（存放接收数据的缓冲区指针），count（要接收的字节数量）
功能：主设备接收从设备发送的数据
返回：返回成功接收的字节数，失败则返回值小于0

5.int i2c_transfer(struct i2c_adapter *adap, struct i2c_msg *msgs, int num)
```

参数：adap(所使用的I2C适配器指针)，msgs（i2c_msg消息结构体指针），num（传输　　的消息数量）
功能：通过找到i2c_adapter的i2c_algorithm，然后使用其master_xfer函数去驱动硬　　件，一次可以传输多条i2c_msg
返回值：成功返回0
注：使用i2c_master_send()，i2c_master_recv()时不用考虑读写方向，API的实现中已经通过i2c_msg结构体的flags成员确定了读写方向。

I2C用户空间文件系统接口

```
#include <linux/i2c-dev.h>
static const struct file_operations i2cdev_fops = {
    .owner      = THIS_MODULE,
    .llseek     = no_llseek,
    .read       = i2cdev_read,//读
    .write      = i2cdev_write,//写
    .unlocked_ioctl = i2cdev_ioctl,
    .open       = i2cdev_open,//打开
    .release    = i2cdev_release,
};
```

实例：

本例中实现注册一个从地址为0x40的I2C设备，并将其挂载在I2C1总线上

```
&i2c1 {
    agenew_i2c@40 {
        compatible = "agenew,agenew_i2c";
        reg = <0x40>;
        status = "okay";//可用 ；disable 代表不可用
    };
};
```

驱动代码：

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/i2c.h>

#define I2C_TEST_NAME "agenew_i2c"

static int i2c_test_probe(struct i2c_client *client, const struct i2c_device_id
*id);
static int i2c_test_remove(struct i2c_client *client);

static struct i2c_device_id i2c_dev[] = {
    {I2C_TEST_NAME},
};

static const struct of_device_id test_driver_of_match[] = {
    {.compatible = "agenew,agenew_i2c",},
    {},
};
```

```c
static struct i2c_driver test_driver = {

    .probe = i2c_test_probe,
    .remove = i2c_test_remove,
    .id_table = i2c_dev,
    .driver = {
        .owner = THIS_MODULE,
        .name = "agenew_test"
        .of_match_table = test_driver_of_match,
    }
};

static int i2c_test_probe(struct i2c_client *client, const struct i2c_device_id
*id)
{
    printk("client->addr = %d   id->name = %s\n", client->addr, id->name);
    printk("%s  called\n", __func__);
    /*You can do what you want here*/
    return 0;
}
static int i2c_test_remove(struct i2c_client *client)
{
    printk("%s  called\n", __func__);
    return 0;
}

static int __init i2c_test_init(void)
{
    printk("%s start\n",__func__);
    i2c_add_driver(&test_driver);
    printk("%s end\n",__func__);
    return 0;
}

static void __exit i2c_test_exit(void)
{
    i2c_del_driver(&test_driver);
}

module_init(i2c_test_init);
module_exit(i2c_test_exit);
MODULE_AUTHOR("chenzelu@agenewtech.com");
MODULE_DESCRIPTION("agenew_i2c driver");
MODULE_LICENSE("GPL");
```