

DHT11驱动(单总线设备)程序编写

DHT11驱动(单总线设备)程序编写

1.确定使用的IO口并准备硬件

2.修改设备树

2.1.设备树修改实操:

3.裁剪LINUX内核, 编译并烧写

3.1.裁剪内核

3.2.编译内核

3.3.烧写SD卡

4.DHT11驱动编写

4.1.DHT11基础知识

4.2.dht11_drv.c逐帧解析

4.3.dht11_drv.c(完整)

5.dht11应用程序编写

装载驱动

程序现象:

实现步骤:

1.确定使用的IO口并准备硬件

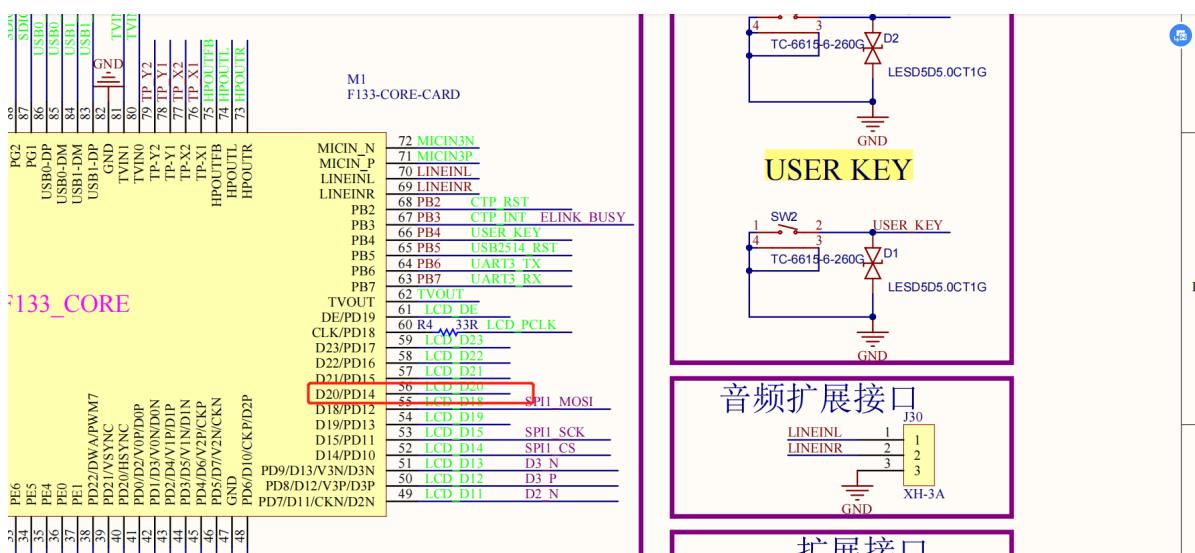
2.修改设备树

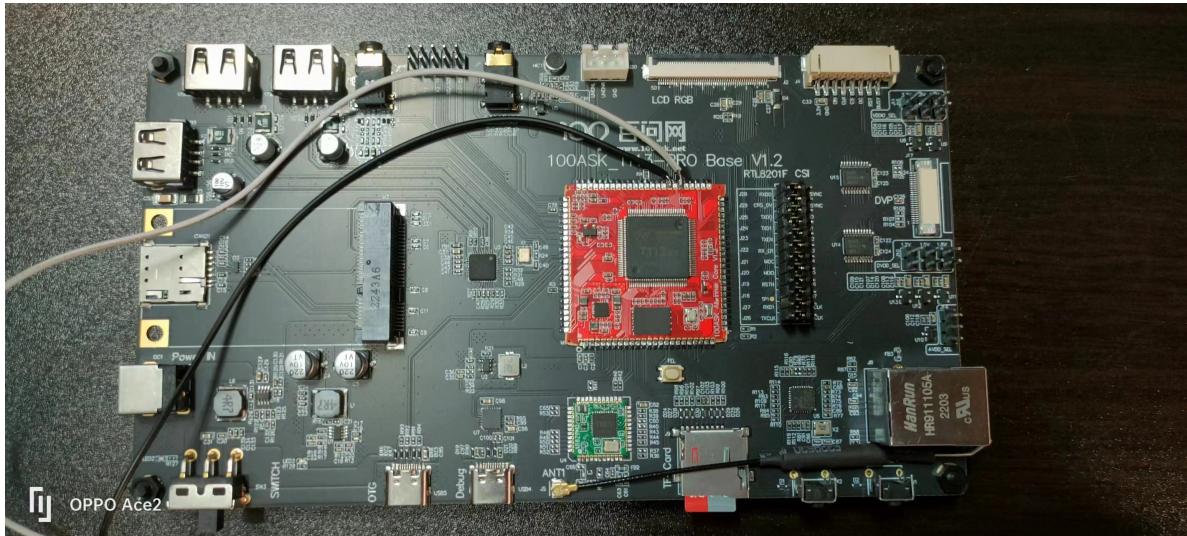
3.裁剪LINUX内核, 编译并烧写

4.DHT11驱动编写

1.确定使用的IO口并准备硬件

在之前飞线LED的IO口时, DHT11的IO (黑线) 我也一并飞了, 56号引脚, PD14。





2.修改设备树

2.1.设备树修改实操:

```
cd /buildroot-100ask_t113-pro/buildroot/output/build/linux-  
origin_master/arch/arm/boot/dts
```

```
sudo gedit sun8iw20p1-t113-100ask-t113-pro.dts
```

代码块1：挂载dht11设备节点到根设备节点上

```
/*添加DHT11的设备树文件*/  
dht11 { /*定义了一个名称为"**dht11**"的节点。*/  
    compatible = "dht-11"; /*属性指定了该节点与哪种设备兼容，这里的 "dht-11" 表示该节点与 DHT11 温湿度传感器设备兼容*/  
    pinctrl-names = "default";  
    pinctrl-1 = <&dht11_pin>; /*属性指向了名为 "dht11_pin" 的 pin control 节点，该节点包含了与 "default" pin control 关联的硬件引脚配置信息。即代码块2信息*/  
    dht11-gpios = <&pio PD 14 GPIO_ACTIVE_HIGH>; /*属性指定了 DHT11 节点使用的 GPIO 引脚，其中包括了引脚号 (PD 14)、设备名称 (pio) 和电平状态 (GPIO_ACTIVE_HIGH) */  
    status = "okay";  
};
```

代码块2：pio设置

```
dht11_pin: dht11_pin{  
    allwinner,pins = "PD14"; /*dht11的时钟和数据线接PD14上面*/  
};
```

在根设备节点下添加挂载dht11设备节点：

```
sun8iw20p1-t113-100ask-t113-pro.dts
-/buildroot-100ask_t113-pro/buil..origin_master/arch/arm/boot/dts

};

/*添加DHT11的设备树文件*/
dht11 {
    compatible = "dht-11";
    pinctrl-names = "default";
    pinctrl-0 = <&dht11_pin>;
    dht11-gpios = <&pio PD 14 GPIO_ACTIVE_HIGH>;
    status = "okay";
};

&cpu0 {
    cpu-supply = <&reg_vdd_cpu>;
};

&pio {
    key_pins_a: userkey {
        allwinner,pins = "PB4" ;
    };

    dht11_pin: dht11_pin{
        allwinner,pins = "PD14"; /*dht11的时钟和数据线接PD14上面*/
    };

    sdc0_pins_a: sdc0@0 {
        allwinner,pins = "PF0", "PF1", "PF2",
                           "PF3", "PF4", "PF5";
        allwinner,function = "sdc0";
        allwinner,muxsel = <2>;
        allwinner,drive = <3>;
    };
};

Plain Text ▾ Tab Width: 8 ▾ Ln 64, Col 42 ▾ INS
```

3.裁剪LINUX内核，编译并烧写

3.1.裁剪内核

为了驱动 DHT11 设备，Linux 内核需要支持 GPIO 和 I2C 子系统。因此，如果要进行内核裁剪，需要保留这两个子系统。

内核菜单路径：

- GPIO 子系统：
 - Device Drivers -> GPIO Support
- I2C 子系统：
 - Device Drivers -> I2C Support

3.2.编译内核

进入到/buildroot目录下，输入以下两条命令即可在/output/images目录下找到生成的sd卡镜像，该命令用到了buildroot框架。

```
make linux-rebuild V=1
make V=1
```

3.3.烧写SD卡

不演示

4.DHT11驱动编写

到了DHT11的驱动编写这，就涉及到驱动开发真正登堂入室的内容了

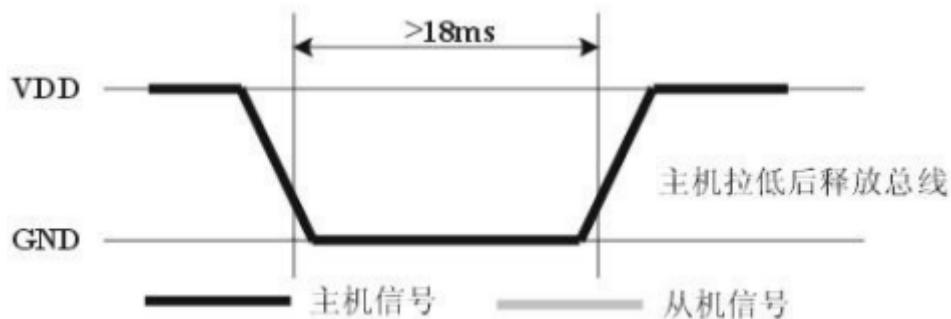
4.1.DHT11基础知识

以下的知识点都是取自DHT11手册：

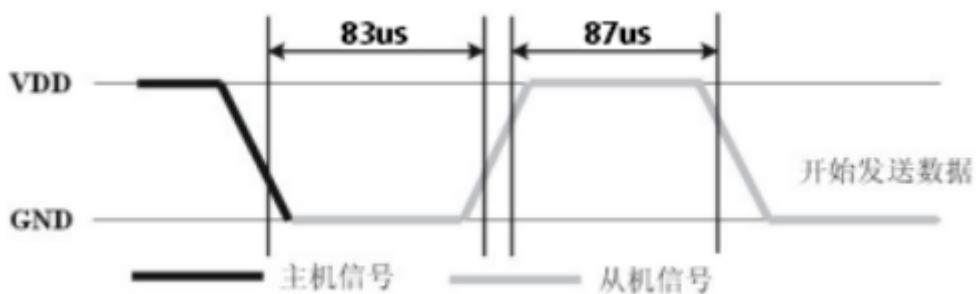
◎外设读取步骤：

步骤一:DHT11 上电后 (DHT11 上电后要等待 **1S** 以越过不稳定状态在此期间不能发送任何指令) , 测试环境 温湿度数据, 并记录数据, 同时 DHT11 的 DATA 数据线由**上拉电阻拉高一直保持高电平**; 此时 DHT11 的 DATA 引脚处于**输入状态**, 时刻检测外部信号。

步骤二 (主机发送起始信号) :微处理器的 I/O 设置为输出同时输出**低电平**, 且低电平保持时间**不能小于 18ms (最大不得超过 30ms)** , 然后微处理器的 I/O 设置为输入状态, 由于上拉电阻, 微处理器的 I/O 即 DHT11 的 DATA 数据线也随之变 高, 等待 DHT11 作出回答信号, 发送信号如图所示:

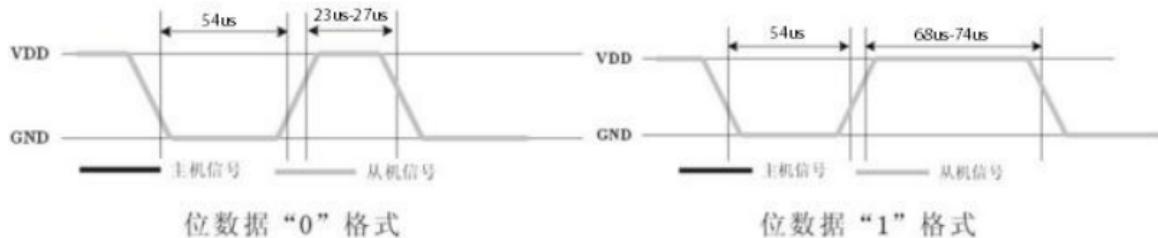


步骤三 (从机响应信号) :DHT11 的 DATA 引脚检测到外部信号有**低电平时**, 等待外部信号**低电平结束**, 延迟后 DHT11 的 DATA 引脚处于输出状态, **输出 83 微秒的低电平作为应答信号**, 紧接着**输出 87 微秒的高电平通知外设准备接收数据**, 微处理器的 I/O 此时处于输入状态, 检测到 I/O 有**低电平** (DHT11 回应信号) 后, 等待 87 微秒的高电平后的数据接收, 发送信号如图所示:



关于步骤二和步骤三, 代码主要涉及的函数为: `dht11_wait_for_ready()`和`dht11_start()`, 前者对应从机响应信号的验证, 后者则是将主机发送起始信号和检验组装一下。

步骤四:由 DHT11 的 DATA 引脚输出 40 位数据, 微处理器根据 I/O 电平的变化接收 40 位数据, **位数据“0”的格式为: 54 微秒的低电平和 23-27 微秒的高电平, 位数据“1”的格式为: 54 微秒的低电平加 68-74 微秒的高电平。**位数据“0”、“1”格式信号如图所示:



以上就是DHT11的数据传输原理, 整体还是较为简洁简单的。

还是这七步:

```
/* 1. 确定主设备号*/
/* 2. 定义file_operations结构体*/
/* 3. 实现对应的open/read/write等函数, 填入file_operations结构体 */
/* 4. 把file_operations结构体告诉内核: 注册驱动程序*/
/* 5. 谁来注册驱动程序啊? 得有一个入口函数: 安装驱动程序时就会去调用这个入口函数*/
/* 6. 有入口函数就应该有出口函数: 卸载驱动程序时, 就会去调这个出口函数*/
/* 7. 其他完善: 提供设备信息, 自动创建设备节点*/
```

4.2.dht11_drv.c逐帧解析

```
/* 1. 确定主设备号*/
```

```
struct dht11
{
    int gpio; /* gpio */
    int irq;
    dev_t dev_no; /* 设备号 */
    struct cdev chrdev; /*字符设备*/
    struct class *class; /*设备类*/
    spinlock_t lock;
};
```

```
/* 2. 定义file_operations结构体*/
```

```
static struct file_operations drv_file_ops = {
    .owner = THIS_MODULE,
    .open = _drv_open,
    .read = _drv_read,
    .release = _drv_release,
};
```

```
/* 3. 实现对应的open/read/write等函数, 填入file_operations结构体 */
```

```
static int dht11_wait_for_ready(void)
```

等待DHT11传感器响应的信号函数，其实这个函数感觉已经不能填在这里了，但实际上也确实是这部分的东西，因为前面定义的各个函数都是服务于open/read/write等函数

```
static int dht11_wait_for_ready(void)
{
    int timeout = 400;
    while (DHT11_READ() && timeout) /* 等待低电平到来///准备接收传感器的响应信号
    {
        udelay(1);
        --timeout;
    }
    if (!timeout)
    {
        printk("[failed1] timeout %d\n", __LINE__);
        return -1; /* 超时 */
    }

    timeout = 1000;
    while (!DHT11_READ() && timeout)/* 等待高电平到来///当读到低电平，说明响应信号开始了，需要持续83us
    {
        udelay(1);
        --timeout;
    }
    if (!timeout)
    {
        printk("[failed2] timeout %d\n", __LINE__);
        return -1; /* 超时 */
    }

    timeout = 1000;
    while (DHT11_READ() && timeout)/* 等待高电平结束///当读到高电平，说明响应信号进入87us阶段，跳出循环说明响应信号结束了
    {
        udelay(1);
        --timeout;
    }
    if (!timeout)
    {
        printk("[failed3] timeout %d\n", __LINE__);
        return -1; /* 超时 */
    }

    return 0;
}
```

static int dht11_start(void)

DHT11开始函数，负责完成步骤1-3，某种程度上讲，其实DHT11上电后应该保持1s的高电平，如果驱动出现问题或者说数据有奇怪的值可以考虑将设备设置为输入模式然后延时1s

```
static int dht11_start(void)
{
    /* 主机对DHT11发送起始信号 */
```

```

DHT11_IO_OUT();
DHT11_WRITE(0);
mdelay(20);
DHT11_WRITE(1);
udelay(30);
DHT11_IO_IN(); /* 设置为输入 */
udelay(2); //等2us让高电平过渡到低电平

/*从机向主机发送响应信号*/
if (dht11_wait_for_ready())//检验DHT11传感器的响应信号
    return -1;
return 0;
}

```

static int dht11_read_byte(unsigned char *byte)

DHT11读数据/数据解析函数，负责完成步骤4

```

static int dht11_read_byte(unsigned char *byte)
{
    unsigned char i;
    unsigned char bit = 0;
    unsigned char data = 0;
    int timeout = 0;

    for (i = 0; i < 8; i++)
    {
        timeout = 1000;
        while (DHT11_READ() && timeout) /* 等待变为低电平 */
        {
            udelay(1);
            --timeout;
        }
        if (!timeout)
        {
            printk("[failed] timeout %d\n", __LINE__);
            return -1; /* 超时 */
        }

        timeout = 1000;
        while (!DHT11_READ() && timeout) /* 等待变为高电平 */
        {
            udelay(1);
            --timeout;
        }
        if (!timeout)
        {
            printk("[failed] timeout %d\n", __LINE__);
            return -1; /* 超时 */
        }
    }
    //每个二进制位的时间持续120us,高电平持续23~27us表示0,持续68~74us表示1,低电平持续
    54us
}

```

```

//若位数据为0, 结束低电平后延时40us, 信号到了一个二进制位的时间的第94us, 大于54+27,
说明此时电平必然为低
//若位数据为1, 结束低电平后延时40us, 信号到了一个二进制位的时间的第94us, 小于54+68,
说明此时电平必然为高
udelay(40);

bit = DHT11_READ();

data <= 1;
if (bit)
{
    data |= 0x01;

#if 0
    timeout = 1000;
    while (DHT11_READ() && timeout) /* 等待高电平结束 */
    {
        udelay(1);
        --timeout;
    }
    if (!timeout)
    {
        printk("timeout %d\n", __LINE__);
        return -1; /* 超时 */
    }
#endif
}
// data <= 1; /* 导致错误的原因 : 移位要放前面, 不能放在这里, 若放在后
面一旦获取最后一个位就会多移动一位导致数据不对 */
}

*byte = data;
return 0;
}

```

_drv_open(struct inode *node, struct file *file)

open函数

```

static int _drv_open(struct inode *node, struct file *file)
{
    printk("[success] dht11 open\n");
    return 0;
}

```

_drv_read(struct file *filp, char __user *buf, size_t size, loff_t *offset)

read函数, 实际上就是调用前面定义的那些函数, 分别为步骤1-4

```

static ssize_t _drv_read(struct file *filp, char __user *buf, size_t size,
loff_t *offset)
{
    int ret;
    int i;

```

```
unsigned char data[5] = {0};
unsigned long flags;

/* 缓冲区大小不为5就报错-EINVAL */
if (size != 5)
    return -EINVAL;

/* 单总线设备时序要求严格, 关闭中断, 防止时序被中断破坏 */
spin_lock_irqsave(&dht11_dev.lock, flags);

/* 启动信号 */
if (dht11_start() != 0)
{
    printk("[failed] dht11 start failed\n");
    ret = -EFAULT;
    goto failed1;
}

/* 读出5字节数据 */
for (i = 0; i < 5; i++)
{
    if (dht11_read_byte(&data[i]))
    {
        printk("[failed] data err\n");
        ret = -EAGAIN;
        goto failed1;
    }
}

/* 打开中断 */
spin_unlock_irqrestore(&dht11_dev.lock, flags);
/* 校验数据 */
if (data[4] != (data[0] + data[1] + data[2] + data[3]))
{
    printk("[failed] check data failed\n");
    ret = -EAGAIN;
    goto failed1;
}

/* 将数据拷贝回用户空间 */
if (copy_to_user(buf, data, 5))
{
    ret = -EFAULT;
    goto failed1;
}
else
{
    ret = 5;
}

return ret;

failed1:
spin_unlock_irqrestore(&dht11_dev.lock, flags);
return ret;
```

```
 }static ssize_t _drv_read(struct file *filp, char __user *buf, size_t size,
loff_t *offset)
{
    int ret;
    int i;
    unsigned char data[5] = {0};
    unsigned long flags;

    if (size != 5)
        return -EINVAL;

    /* 关闭中断,防止时序被中断破坏 */
    spin_lock_irqsave(&dht11_dev.lock, flags);

    /* 启动信号 */
    if (dht11_start() != 0)
    {
        printk("[failed] dht11 start failed\n");
        ret = -EFAULT;
        goto failed1;
    }

    /* 读出5字节数据 */
    for (i = 0; i < 5; i++)
    {
        if (dht11_read_byte(&data[i]))
        {
            printk("[failed] data err\n");
            ret = -EAGAIN;
            goto failed1;
        }
    }

    /* 打开中断 */
    spin_unlock_irqrestore(&dht11_dev.lock, flags);
    /* 校验数据 */
    if (data[4] != (data[0] + data[1] + data[2] + data[3]))
    {
        printk("[failed] check data failed\n");
        ret = -EAGAIN;
        goto failed1;
    }

    /* 将数据拷贝回用户空间 */
    if (copy_to_user(buf, data, 5))
    {
        ret = -EFAULT;
        goto failed1;
    }
    else
    {
        ret = 5;
    }

    return ret;
}
```

```
failed1:
    spin_unlock_irqrestore(&dht11_dev.lock, flags);
    return ret;
}
```

`_drv_release(struct inode *node, struct file *file)`

设备释放函数

```
static int _drv_release(struct inode *node, struct file *file)
{
    printk("[success] dht11 release\n");
    return 0;
}
```

宏定义

```
//前情提要
#define DEV_DTS_NODE_PATH "/dht11"      /* 设备树节点的路径，在根节点下 */
#define DEV_PIN_DTS_NAME "dht11-gpios" /* GPIO引脚的属性名 */
#define DEV_NAME "dht11"           /* 设备名 /dev/dht11 */
#define DEV_DTS_COMPATIBLE "dht-11" /* 设备匹配属性 compatible */

#define DHT11_PIN dht11_dev.gpio
#define DHT11_IO_OUT() gpio_direction_output(DHT11_PIN, 1);
#define DHT11_IO_IN() gpio_direction_input(DHT11_PIN)
#define DHT11_WRITE(bit) gpio_set_value(DHT11_PIN, bit)
#define DHT11_READ() gpio_get_value(DHT11_PIN)
```

设备树列表(platform_device注册)

```
/* 设备树的匹配列表 */
static struct of_device_id dts_match_table[] = {
{
    .compatible = DEV_DTS_COMPATIBLE,
}, /* 通过设备树来匹配 */
};
```

`_driver_probe(struct platform_device *pdev)`

```
static int _driver_probe(struct platform_device *pdev)
{
    int err;
    struct device *ds_dev;
    struct device_node *dev_node;

    struct device_node *node = pdev->dev.of_node;

    if (!node)
    {
        printk("dts node can not found!\r\n");
        return -EINVAL;
    }
```

```
    dev_node = of_find_node_by_path(DEV_DTS_NODE_PATH); /* 找到dht11的设备树节点
*/
    if (IS_ERR(dev_node))
    {
        printk("dht11 DTS Node not found!\r\n");
        return PTR_ERR(dev_node);
    }

    dht11_dev.gpio = of_get_named_gpio(dev_node, DEV_PIN_DTS_NAME, 0); /* 获取
dht11的gpio编号 */
    if (dht11_dev.gpio < 0)
    {
        printk("dht11-gpio not found!\r\n");
        return -EINVAL;
    }

    err = gpio_request(dht11_dev.gpio, DEV_PIN_DTS_NAME);
    if (err)
    {
        printk("gpio_request gpio is failed!\r\n");
        return -EINVAL;
    }

    printk("dht11 gpio %d\r\n", dht11_dev.gpio);

/* 内核自动分配设备号 */
    err = alloc_chrdev_region(&dht11_dev.dev_no, 0, 1, DEV_NAME);
    if (err < 0)
    {
        pr_err("Error: failed to register mbochs_dev, err: %d\r\n", err);
        goto failed3;
    }

    cdev_init(&dht11_dev.chrdev, &drv_file_ops);

    cdev_add(&dht11_dev.chrdev, dht11_dev.dev_no, 1);

    dht11_dev.class = class_create(THIS_MODULE, DEV_NAME);
    if (IS_ERR(dht11_dev.class))
    {
        err = PTR_ERR(dht11_dev.class);
        goto failed1;
    }

    /* 创建设备节点 */
    ds_dev = device_create(dht11_dev.class, NULL, dht11_dev.dev_no, NULL,
    DEV_NAME);
    if (IS_ERR(ds_dev))
    { /* 判断指针是否合法 */
        err = PTR_ERR(ds_dev);
        goto failed2;
    }

    spin_lock_init(&dht11_dev.lock); /* 初始化自旋锁 */
```

```

        printk("[success] dht11 probe success\r\n");
        return 0;
failed2:
    device_destroy(dht11_dev.class, dht11_dev.dev_no);
    class_destroy(dht11_dev.class);
failed1:
    unregister_chrdev_region(dht11_dev.dev_no, 1);
    cdev_del(&dht11_dev.chrdev);
failed3:
    gpio_free(dht11_dev.gpio);
    return err;
}

```

_driver_remove(struct platform_device *pdev)

```

static int _driver_remove(struct platform_device *pdev)
{
    device_destroy(dht11_dev.class, dht11_dev.dev_no);
    class_destroy(dht11_dev.class);
    unregister_chrdev_region(dht11_dev.dev_no, 1);
    cdev_del(&dht11_dev.chrdev);
    gpio_free(dht11_dev.gpio);

    printk(KERN_INFO "[success] dht11 remove success\n");

    return 0;
}

```

_platform_driver结构体

```

static struct platform_driver _platform_driver = {
    .probe = _driver_probe,
    .remove = _driver_remove,
    .driver = {
        .name = DEV_DTS_COMPATIBLE,
        .owner = THIS_MODULE,
        .of_match_table = dts_match_table, /* 通过设备树匹配 */
    },
};

```

/* 5. 谁来注册驱动程序啊？得有一个入口函数：安装驱动程序时就会去调用这个入口函数*/

/* 6. 有入口函数就应该有出口函数：卸载驱动程序时，就会去调这个出口函数*/

```

/* 入口函数 */
static int __init _driver_init(void)
{
    int ret;
    printk("dht11 %s\n", __FUNCTION__);

    ret = platform_driver_register(&_platform_driver); //注册platform驱动

    return ret;
}

```

```

}

/* 出口函数 */
static void __exit __driver_exit(void)
{
    printk("dht11 %s\n", __FUNCTION__);
    platform_driver_unregister(&platform_driver);
}

```

/* 7. 其他完善：提供设备信息，自动创建设备节点*/

```

module_init(__driver_init);
module_exit(__driver_exit);

MODULE_AUTHOR("Fourth");
MODULE_LICENSE("GPL");

```

4.3.dht11_drv.c(完整)

```

#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/input.h>
#include <linux/delay.h>
#include <linux/slab.h>
#include <linux/interrupt.h>
#include <linux/irq.h>
#include <linux/gpio.h>
#include <linux/cdev.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/uaccess.h>

#include <linux/of_gpio.h>

typedef unsigned char uint8_t;

#define DEV_DTS_NODE_PATH "/dht11"      /* 设备树节点的路径，在根节点下 */
#define DEV_PIN_DTS_NAME "dht11-gpios" /* GPIO引脚的属性名 */
#define DEV_NAME "dht11"                /* 设备名 /dev/dht11 */
#define DEV_DTS_COMPATIBLE "dht-11"    /* 设备匹配属性 compatible */

#define DHT11_PIN dht11_dev.gpio
#define DHT11_IO_OUT() gpio_direction_output(DHT11_PIN, 1);
#define DHT11_IO_IN() gpio_direction_input(DHT11_PIN)
#define DHT11_WRITE(bit) gpio_set_value(DHT11_PIN, bit)
#define DHT11_READ() gpio_get_value(DHT11_PIN)

struct dht11
{
    int gpio; /* gpio */
    int irq;
    dev_t dev_no; /* 设备号 */
    struct cdev chrdev; /*字符设备*/
}

```

```
    struct class *class; /*设备类*/
    spinlock_t lock;
};

struct dht11 dht11_dev;

static int dht11_wait_for_ready(void)
{
    int timeout = 400;
    while (DHT11_READ() && timeout) /* 等待低电平到来 */
    {
        udelay(1);
        --timeout;
    }
    if (!timeout)
    {
        printk("[failed1] timeout %d\n", __LINE__);
        return -1; /* 超时 */
    }

    timeout = 1000;
    while (!DHT11_READ() && timeout) /* 等待高电平到来 */
    {
        udelay(1);
        --timeout;
    }
    if (!timeout)
    {
        printk("[failed2] timeout %d\n", __LINE__);
        return -1; /* 超时 */
    }

    timeout = 1000;
    while (DHT11_READ() && timeout) /* 等待高电平结束 */
    {
        udelay(1);
        --timeout;
    }
    if (!timeout)
    {
        printk("[failed3] timeout %d\n", __LINE__);
        return -1; /* 超时 */
    }

    return 0;
}

static int dht11_start(void)
{
    DHT11_IO_OUT();
    DHT11_WRITE(0);
    mdelay(20);
    DHT11_WRITE(1);
    udelay(30);
}
```

```
DHT11_IO_IN(); /* 设置为输入 */
udelay(2);

if (dht11_wait_for_ready())
    return -1;
return 0;

}

static int dht11_read_byte(unsigned char *byte)
{
    unsigned char i;
    unsigned char bit = 0;
    unsigned char data = 0;
    int timeout = 0;

    for (i = 0; i < 8; i++)
    {
        timeout = 1000;
        while (DHT11_READ() && timeout) /* 等待变为低电平 */
        {
            udelay(1);
            --timeout;
        }
        if (!timeout)
        {
            printk("[failed] timeout %d\n", __LINE__);
            return -1; /* 超时 */
        }

        timeout = 1000;
        while (!DHT11_READ() && timeout) /* 等待变为高电平 */
        {
            udelay(1);
            --timeout;
        }
        if (!timeout)
        {
            printk("[failed] timeout %d\n", __LINE__);
            return -1; /* 超时 */
        }
        udelay(40);

        bit = DHT11_READ();

        data <= 1;
        if (bit)
        {
            data |= 0x01;

#if 0
            timeout = 1000;
            while (DHT11_READ() && timeout) /* 等待高电平结束 */
            {
                udelay(1);
            }
#endif
        }
    }
}
```

```
    --timeout;
}
if (!timeout)
{
    printk("timeout %d\n", __LINE__);
    return -1; /* 超时 */
}
#endif
}

// data <= 1; /* 导致错误的原因：移位要放前面，不能放在这里，若放在后面一旦获取最后一个位就会多移动一位导致数据不对 */
}

*byte = data;
return 0;
}

/* 使设备只能被一个进程打开 */
static int _drv_open(struct inode *node, struct file *file)
{
    printk("[success] dht11 open\n");
    return 0;
}

static ssize_t _drv_read(struct file *filp, char __user *buf, size_t size,
loff_t *offset)
{
    int ret;
    int i;
    unsigned char data[5] = {0};
    unsigned long flags;

    if (size != 5)
        return -EINVAL;

    /* 关闭中断，防止时序被中断破坏 */
    spin_lock_irqsave(&dht11_dev.lock, flags);

    /* 启动信号 */
    if (dht11_start() != 0)
    {
        printk("[failed] dht11 start failed\n");
        ret = -EFAULT;
        goto failed1;
    }

    /* 读出5字节数据 */
    for (i = 0; i < 5; i++)
    {
        if (dht11_read_byte(&data[i]))
        {
            printk("[failed] data err\n");
            ret = -EAGAIN;
            goto failed1;
        }
    }
}

static void _drv_release(struct inode *inode, struct file *file)
{
    spin_unlock_irqrestore(&dht11_dev.lock, flags);
}
```

```
        }

/* 打开中断 */
spin_unlock_irqrestore(&dht11_dev.lock, flags);
/* 校验数据 */
if (data[4] != (data[0] + data[1] + data[2] + data[3]))
{
    printk("[failed] check data failed\n");
    ret = -EAGAIN;
    goto failed1;
}

/* 将数据拷贝回用户空间 */
if (copy_to_user(buf, data, 5))
{
    ret = -EFAULT;
    goto failed1;
}
else
{
    ret = 5;
}

return ret;

failed1:
spin_unlock_irqrestore(&dht11_dev.lock, flags);
return ret;
}

static int _drv_release(struct inode *node, struct file *file)
{
    printk("[success] dht11 release\n");
    return 0;
}

static struct file_operations drv_file_ops = {
    .owner = THIS_MODULE,
    .open = _drv_open,
    .read = _drv_read,
    .release = _drv_release,
};

/* 设备树的匹配列表 */
static struct of_device_id dts_match_table[] = {
{
    .compatible = DEV_DTS_COMPATIBLE,
}, /* 通过设备树来匹配 */
};

static int _driver_probe(struct platform_device *pdev)
{
    int err;
    struct device *ds_dev;
```

```

    struct device_node *dev_node;

    struct device_node *node = pdev->dev.of_node;

    if (!node)
    {
        printk("dts node can not found!\r\n");
        return -EINVAL;
    }

    dev_node = of_find_node_by_path(DEV_DTS_NODE_PATH); /* 找到dht11的设备树节点
*/
    if (IS_ERR(dev_node))
    {
        printk("dht11 DTS Node not found!\r\n");
        return PTR_ERR(dev_node);
    }

    dht11_dev.gpio = of_get_named_gpio(dev_node, DEV_PIN_DTS_NAME, 0); /* 获取
dht11的gpio编号 */
    if (dht11_dev.gpio < 0)
    {
        printk("dht11-gpio not found!\r\n");
        return -EINVAL;
    }

    err = gpio_request(dht11_dev.gpio, DEV_PIN_DTS_NAME);
    if (err)
    {
        printk("gpio_request gpio is failed!\n");
        return -EINVAL;
    }

    printk("dht11 gpio %d\n", dht11_dev.gpio);

/* 内核自动分配设备号 */
    err = alloc_chrdev_region(&dht11_dev.dev_no, 0, 1, DEV_NAME);
    if (err < 0)
    {
        pr_err("Error: failed to register mbochs_dev, err: %d\n", err);
        goto failed3;
    }

    cdev_init(&dht11_dev.chrdev, &drv_file_ops);

    cdev_add(&dht11_dev.chrdev, dht11_dev.dev_no, 1);

    dht11_dev.class = class_create(THIS_MODULE, DEV_NAME);
    if (IS_ERR(dht11_dev.class))
    {
        err = PTR_ERR(dht11_dev.class);
        goto failed1;
    }

/* 创建设备节点 */

```

```
    ds_dev = device_create(dht11_dev.class, NULL, dht11_dev.dev_no, NULL,
DEV_NAME);
    if (IS_ERR(ds_dev))
    { /* 判断指针是否合法 */
        err = PTR_ERR(ds_dev);
        goto failed2;
    }

    spin_lock_init(&dht11_dev.lock); /* 初始化自旋锁 */
    printk("[success] dht11 probe success\r\n");
    return 0;

failed2:
    device_destroy(dht11_dev.class, dht11_dev.dev_no);
    class_destroy(dht11_dev.class);
failed1:
    unregister_chrdev_region(dht11_dev.dev_no, 1);
    cdev_del(&dht11_dev.chrdev);
failed3:
    gpio_free(dht11_dev.gpio);
    return err;
}

static int _driver_remove(struct platform_device *pdev)
{
    device_destroy(dht11_dev.class, dht11_dev.dev_no);
    class_destroy(dht11_dev.class);
    unregister_chrdev_region(dht11_dev.dev_no, 1);
    cdev_del(&dht11_dev.chrdev);
    gpio_free(dht11_dev.gpio);

    printk(KERN_INFO "[success] dht11 remove success\n");

    return 0;
}

static struct platform_driver _platform_driver = {
    .probe = _driver_probe,
    .remove = _driver_remove,
    .driver = {
        .name = DEV_DTS_COMPATIBLE,
        .owner = THIS_MODULE,
        .of_match_table = dts_match_table, /* 通过设备树匹配 */
    },
};

/* 入口函数 */
static int __init _driver_init(void)
{
    int ret;
    printk("dht11 %s\n", __FUNCTION__);

    ret = platform_driver_register(&_platform_driver); //注册platform驱动
```

```

    return ret;
}

/* 出口函数 */
static void __exit __driver_exit(void)
{
    printk("dht11 %s\n", __FUNCTION__);
    platform_driver_unregister(&_platform_driver);
}

module_init(__driver_init);
module_exit(__driver_exit);

MODULE_AUTHOR("Fourth");
MODULE_LICENSE("GPL");

```

5.dht11应用程序编写

```

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <sys/ioctl.h>
#include <poll.h>
#include <stdint.h>

#define DEV_NAME "/dev/dht11"

typedef struct dht11
{
    float temperature;
    float humidity;
} DHT11;

void sleep_ms(unsigned int ms)
{
    struct timeval delay;
    delay.tv_sec = 0;
    delay.tv_usec = ms * 1000;
    select(0, NULL, NULL, NULL, &delay);
}

int getDht11(DHT11 *dht11_data)
{
    int fd;
    int ret;

    /* 2. 打开文件 */
    fd = open(DEV_NAME, O_RDONLY); // | O_NONBLOCK

    if (fd < 0)

```

```

    {
        printf("[failed] can not open file %s, %d\n", DEV_NAME, fd);
        return -1;
    }

    uint8_t dht11_temp_data[5];
    int timeout = 5;
    while (timeout)
    {
        ret = read(fd, dht11_temp_data, sizeof(dht11_temp_data)) ==
        sizeof(dht11_temp_data);
        if (ret)
        {
            sleep_ms(500);
            ret = read(fd, dht11_temp_data, sizeof(dht11_temp_data)) ==
            sizeof(dht11_temp_data);
            if (ret)
            {
                dht11_data->temperature = dht11_temp_data[2] +
                (float)dht11_temp_data[3] / 10.00;
                dht11_data->humidity = dht11_temp_data[0] + dht11_temp_data[1] /
                10.00;
                printf("[success] temperture %d.%d  humi %d.%d\r\n",
                dht11_temp_data[2],
                dht11_temp_data[3], dht11_temp_data[0],
                dht11_temp_data[1]);
                close(fd);
                return 0;
            }
            else
                continue;
        }
        else
        {
            printf("[Failed] tempget temp err %d\n", ret);
            timeout--;
        }
        sleep_ms(500);
    }
    close(fd);
    return -1;
}

int main(int argc, char **argv)
{
    DHT11 dht11_data;
    if (!getDht11(&dht11_data))
    {
        printf("!!! temp %.1f  humi %.1f\n", dht11_data.temperature,
        dht11_data.humidity);
    }
    else
        printf("read dht11 error\n");
    return 0;
}

```

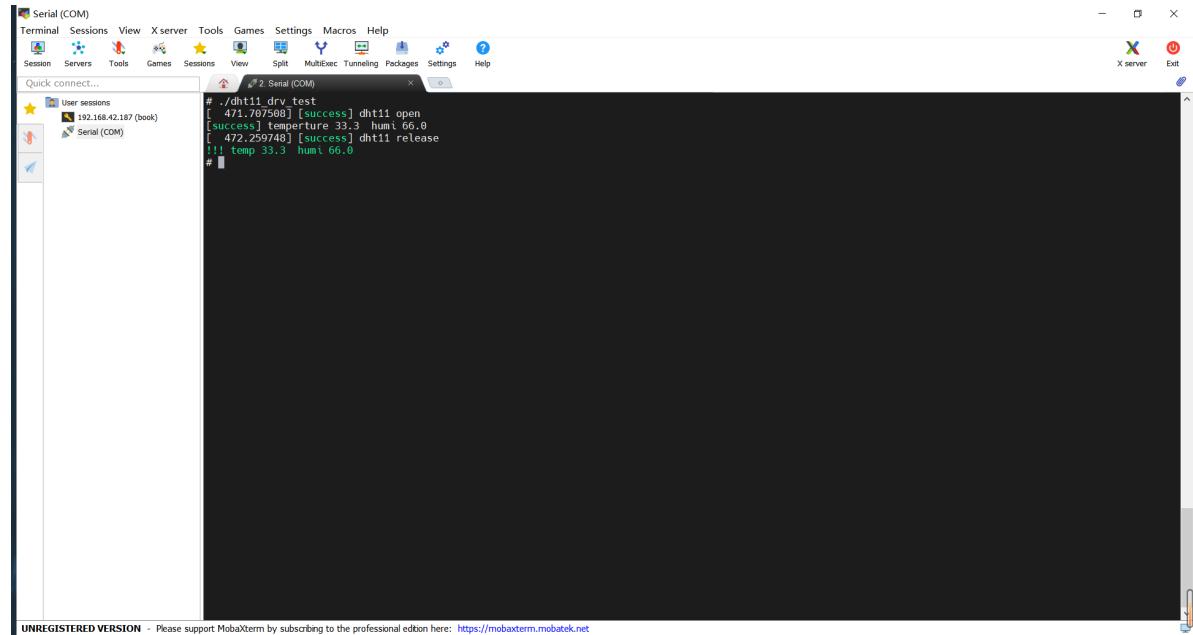
}

装载驱动

```
insmod dht11_drv.ko
```

1smod

程序现象：



每次运行只读一次是合理的，后面还要结合onenet的代码