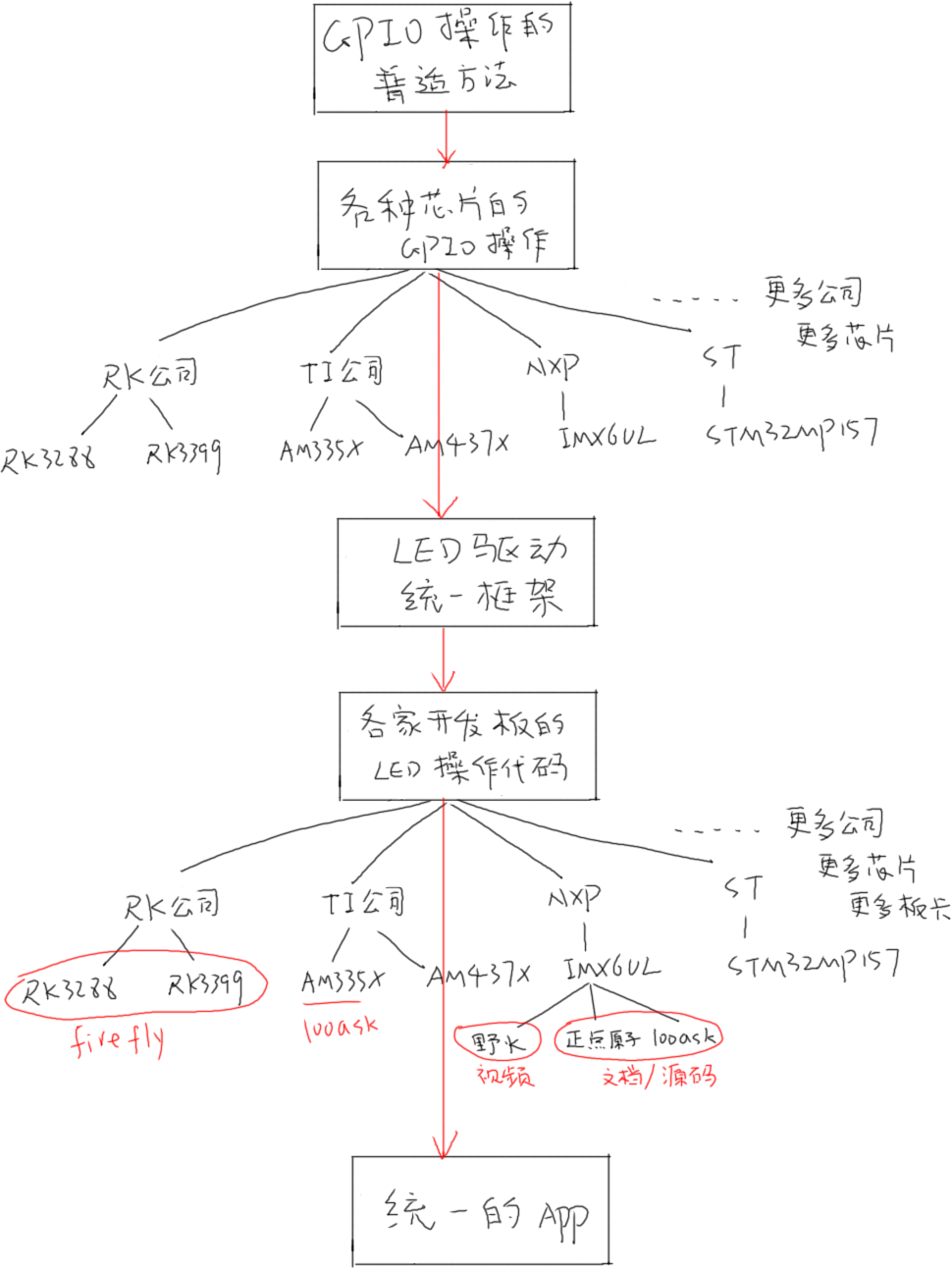
## 1. 嵌入式后Linux驱动开发基础知识的引导与说明

### 1.1 打算讲什么、怎么讲？

以几个简单的驱动程序，讲解嵌入式Linux驱动的框架，了解驱动开发的流程、方法，掌握从APP到驱动的调用流程。

会涉及很多种开发板，让你明白“Linux驱动 = 软件框架 + 硬件操作”，让你“一通百通”，掌握了普适性的原理之后，在工作中很容易在其他板子使用这些知识。

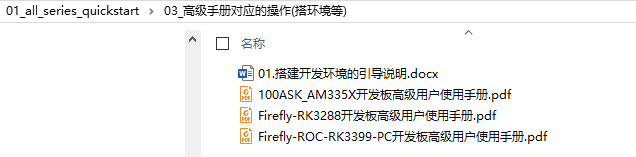
以LED驱动为例，会如下讲解：



### 1.2 需要做什么准备工作

驱动程序依赖于Linux内核，你为开发板A开发驱动，那就先在Ubuntu中得到、配置、编译开发板A所使用的Linux内核。

请使用git下载本教程的文档、源码，查看如下目录中你所用开发板的高级用户使用手册(有些开发板的手册我们还没编写完，持续更新)：



根据手册完成下面操作：

硬件部分：

① 开发板接线：串口线、电源线、网线

② 开发板烧写系统

软件部分：

① 下载Linux内核，Windows和Ubuntu下各放一份

② Windows下：使用Source Insight创建内核源码的工程，这是用来浏览内核、编辑驱动

③ Ubuntu下：安装工具链，配置、编译Linux内核

注意：git的使用方法请参考<http://wiki.100ask.net>中的“初学者学习路线”：



## 2. Hello驱动(不涉及硬件操作)

我们选用的内核都是4.x版本，操作都是类似的：

rk3399 linux 4.4.154

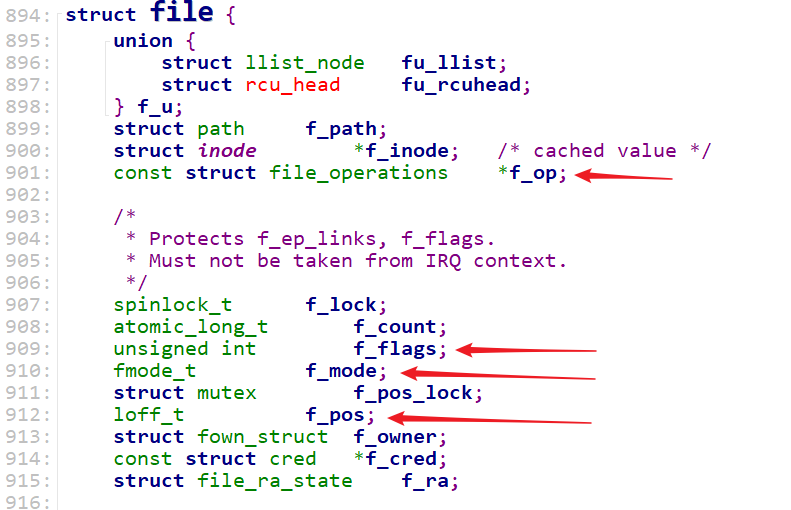
rk3288 linux 4.4.154

imx6ul linux 4.9.88

am3358 linux 4.9.168

### 2.1 APP打开的文件在内核中如何表示

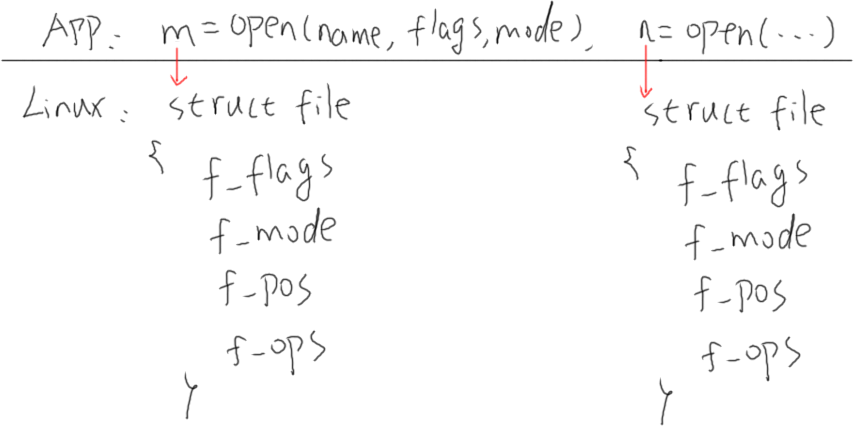
APP打开文件时，可以得到一个整数，这个整数被称为文件句柄。对于APP的每一个文件句柄，在内核里面都有一个“struct file”与之对应。



可以猜测，我们使用open打开文件时，传入的flags、mode等参数会被记录在内核中对应的struct file结构体里(f\_flags、f\_mode)：

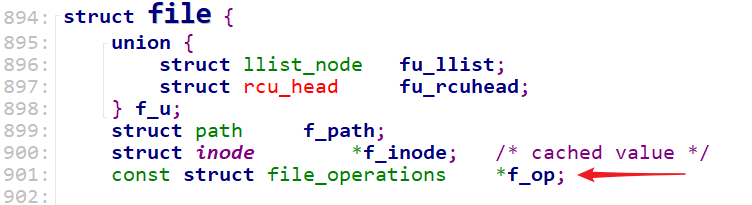
int open(const char \*pathname, int flags, mode\_t mode);

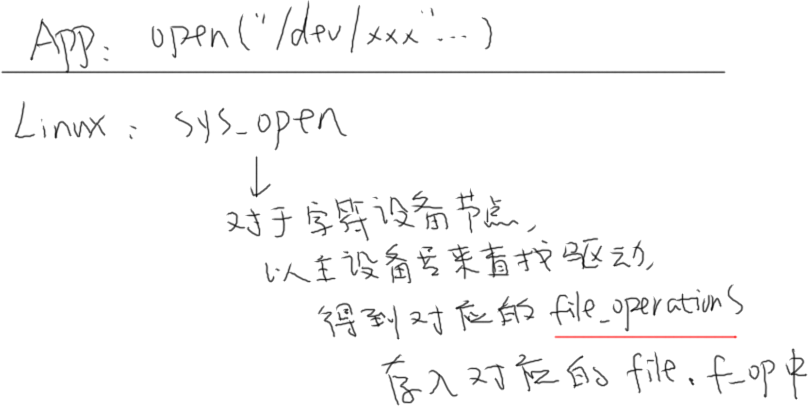
去读写文件时，文件的当前偏移地址也会保存在struct file结构体的f\_pos成员里。



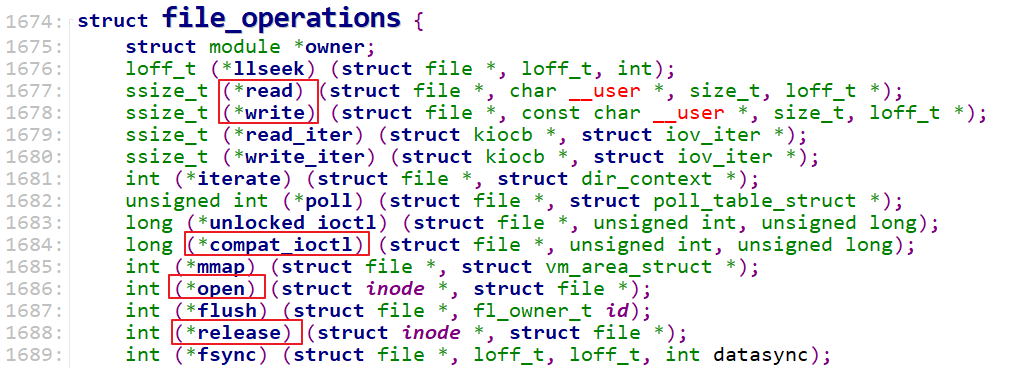
### 2.2 打开字符设备节点时，内核中也有对应的struct file

注意这个结构体中的结构体：struct file\_operations \*f\_op，这是由驱动程序提供的。





结构体struct file\_operations的定义如下：



### 2.3 请猜猜：怎么编写驱动程序？

#### ① 确定主设备号

#### ② 定义自己的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，就会去调用这个出口函数

#### ⑦ 其他完善：提供设备信息，自动创建设备节点

### 2.4 请不要啰嗦，表演你的代码吧

#### ① 写驱动程序

参考driver/char中的程序，包含头文件，写框架，传输数据：

A. 驱动中实现open, read, write, release，APP调用这些函数时，都打印内核信息

B. APP调用write函数时，传入的数据保存在驱动中

C. APP调用read函数时，把驱动中保存的数据返回给APP

#### ② 写测试程序

测试程序要实现写、读功能：

A. ./hello\_drv\_test -w wiki.100ask.net // 把字符串“wiki.100ask.net”发给驱动程序

B. ./hello\_drv\_test -r // 把驱动中保存的字符串读回来

#### ③ 测试

A. 编写驱动程序的Makefile

B. 上机实验

### 2.5 Hello驱动中的一些补充知识

#### ① module\_init/module\_exit的实现

#### ② register\_chrdev的内部实现

#### ③ class\_destroy/device\_create浅析

## 3. 硬件知识\_LED原理图

当我们学习C语言的时候，我们会写个Hello程序。

那当我们写ARM程序，也该有一个简单的程序引领我们入门，这个程序就是点亮LED。

我们怎样去点亮一个LED呢？

分为三步：

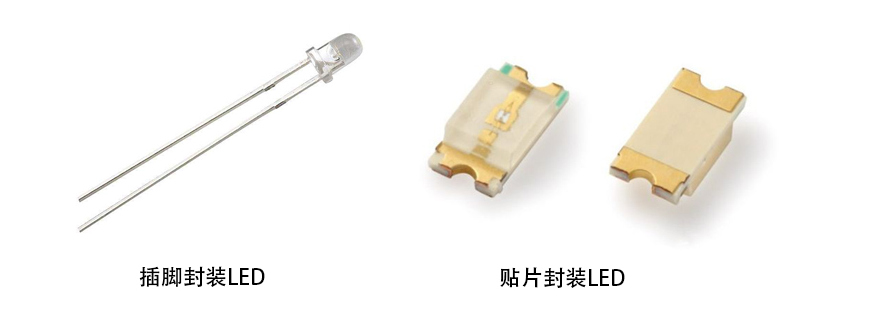
1.看原理图，确定控制LED的引脚;

2.看主芯片的芯片手册，确定如何设置控制这个引脚;

3.写程序;

### 3.1 先来讲讲怎么看原理图：

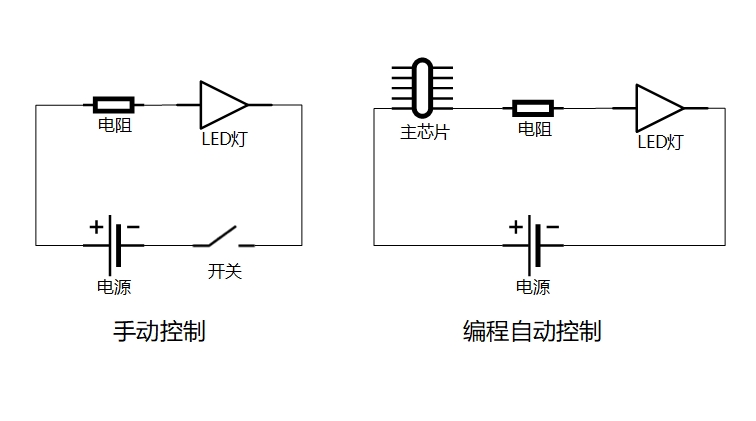
LED样子有很多种，像插脚的，贴片的。



它们长得完全不一样，因此我们在原理图中将它抽象出来。

点亮LED需要通电源，同时为了保护LED，加个电阻减小电流。

控制LED灯的亮灭，可以手动开关LED，但在电子系统中，不可能让人来控制开关，通过编程，利用芯片的引脚去控制开关。



LED的驱动方式，常见的有四种。

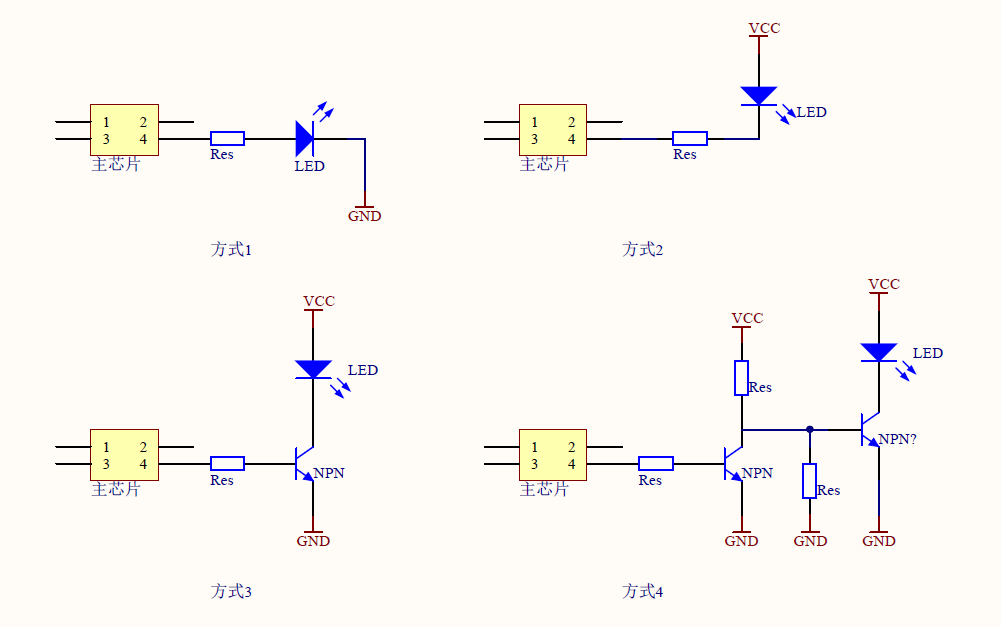
方式1：使用引脚输出3.3V点亮LED，输出0V熄灭LED。

方式2：使用引脚拉低到0V点亮LED，输出3.3V熄灭LED。

有的芯片为了省电等原因，其引脚驱动能力不足，这时可以使用三极管驱动。

方式3：使用引脚输出1.2V点亮LED，输出0V熄灭LED。

方式4：使用引脚输出0V点亮LED，输出1.2V熄灭LED。



由此，主芯片引脚输出高电平/低电平，即可改变LED状态，而无需关注GPIO引脚输出的是3.3V还是1.2V。

所以简称输出1或0：

逻辑1-->高电平

逻辑0-->低电平

## 4. 普适的GPIO引脚操作方法

GPIO: General-purpose input/output，通用的输入输出口

### 4.1 GPIO模块一般结构：

1. 有多组GPIO，每组有多个GPIO
2. 使能：电源/时钟
3. 模式(Mode)：引脚可用于GPIO或其他功能
4. 方向：引脚Mode设置为GPIO时，可以继续设置它是输出引脚，还是输入引脚
5. 数值：对于输出引脚，可以设置寄存器让它输出高、低电平

对于输入引脚，可以读取寄存器得到引脚的当前电平

### 4.2 GPIO寄存器操作：

1. 芯片手册一般有相关章节，用来介绍：power/clock

可以设置对应寄存器使能某个GPIO模块(Module)

有些芯片的GPIO是没有使能开头的，即它总是使能的

1. 一个引脚可以用于GPIO、串口、USB或其他功能，

有对应的寄存器来选择引脚的功能

1. 对于已经设置为GPIO功能的引脚，有方向寄存器用来设置它的方向：输出、输入
2. 对于已经设置为GPIO功能的引脚，有数据寄存器用来写、读引脚电平状态

GPIO寄存器的2种操作方法：

原则：不能影响到其他位

1. 直接读写：读出、修改对应位、写入

要设置bit n：

val = data\_reg;

val = val | (1<<n);

data\_reg = val;

要清除bit n：

val = data\_reg;

val = val & ~(1<<n);

data\_reg = val;

1. set-and-clear protocol：

set\_reg, clr\_reg, data\_reg 三个寄存器对应的是同一个物理寄存器,

要设置bit n：set\_reg = (1<<n);

要清除bit n：clr\_reg = (1<<n);

### 4.3 GPIO的其他功能：防抖动、中断、唤醒：

后续章节再介绍

## 5. 具体单板的GPIO操作方法

请使用GIT下载文档后，看下图红框所示目录中各板子对应的文档。

