30-总结

问题：
1、文档上写的步骤是怎么来的？
2、为什么按照文档做？
3、按照文档做也做不对？
4、错的不知道怎么调？
5、最终结果就是不对？
6、换一个开发板不知道改什么参数？
7、内部实现代码原理？
8、资料怎么来的？

音视频：
修改的参数和硬件是无关的。linux平台。 播放器的源码 --- 支持linux操作系统
声卡驱动 --- 友善之臂 不开源 .ko文件去使用
可执行程序、支持这个程序运行的库： lib、bin
参考资料：ARM9按键去控制音乐播放器 ---- A9做一个音乐播放器 半天
摄像头：基本的项目环境，怎么采集 usb的数据转换为RGB显示到LCD屏上。
利用内核当中V4L2的视频采集的驱动 YUV格式 ---- 转换为RGB 显示在LCD屏
3602代wifi驱动移植：
驱动是360编写好的。参考网页进行移植的。工作当中移植一个驱动：参考各种网页和资料。
结果。 ---- 成功了移植。
将360wifi当成一个无线网卡可以连接到外网。ping通百度。移植了一个wifi。所有的开发板
连接到同一个wifi上，自动分配了一个IP地址，我们可以使用这个IP地址进行套接字编程，进行通信。
iwconfig、iwlist、wpa_supplicant：linux下关于wifi的命令，配置

wifi的命令，文件系统里面没有这几个命令。(制作这些命令，和硬件无关)
iwlist：列举当前的网卡
iwconfig：配置要连接的wifi信息
wpa_supplicant：启动/关闭 网卡
作业：
1、参考音视频播放器手册 编写一个播放器，实现相关功能
2、视频监控(采集摄像头的数据、将YUV格式转换为RGB显示到LCD屏上)
3、移植wifi驱动，(俩个开发板连接同一个wifi，相互可以ping通)俩个A9开发板可以进行通信(套接字编程)

9、工作的联系、方向、内容、水平？
方向：linux
linux应用工程师：
应用层方向：应用开发(多进程、多线程、网络编程)---- 环境 、编程框架里面，框架里面的内容
取决于你们的公司。(车载、医疗、电子消费等)
C++/QT方向：做界面
linux驱动工程师：驱动的编写以及移植。(从应用层做起)
编写：几乎都是字符设备(注册、节点创建、填充相应的操作函数(根据应用层的要求))
将硬件的操作填充到相应的驱动框架里面。(硬件操作 裸机M3的一样)
移植：没有资料的参考。 (参考资料) 按照资料去找去做----结果，前期没必要搞清楚每个过程的由来。 指定内核源码、修改编译器工具。
最根本的东西：open ------ open初始化硬件
资料中：修改内核源码路径、修改指定的编译器
水平：南方：注重用吧 敢说

北方：为什么？ 去用，操作系统实现的框架去用，不用去纠结。

一大部分：学了一大半M3的水平 堆模块。 C语言： 指针、内存、数据结构、算法

翻转一个数组、strcpy 用 ---- 内部。

笔试题：c语言、数据结构和算法 (插入、翻转、删除、排序)---链表程序

指针：函数指针、指针函数、指针数组、指针和字符串

函数参数传递

重写常用的字符串操作函数

运算符优先级结合型问题

面试：M3/M4：关于硬件的知识(串口、IIC、SPI、CAN、485)(原理、怎么用、问题)

项目：功能描述、调试中遇到的难点 --- 怎么解决的

熟悉、会用 --- 熟练 精通？

10、驱动课程的意义所在？

1、了解整个嵌入式开发的流程。

c语言 处理器(M3) 51 AVR DSP M0 M8 M3 M4 //硬件操作 协议不依赖于硬件

linux：应用层 + 驱动层 2个月 做了没？

linux内核设计与实现：内核原理 第三版 2.6

linux设备驱动程序：

内核 -- 模块化编程 --- 调试技术 --- 字符设备注册 --- 函数接口 --- 定时器

中断 ---- 块设备 --- 网络设备

API：接口。实现的流程 内核如何实现 --- 内核编写

工作当前够不够用？

校招：几乎没人会

社招：很多应用层字符设备注册有几个方式？ 驱动：韦东山 2

倍语速

1-2年：

应用层：工资定死的。

嵌入式：积累。 5000-7000 8000--9000

嵌入式：工作一年 10K， 一年的积累。学习 熟练、

工作当中：一样

11、内核移植、裁剪？

1、移植工作。 嵌入式系统工程师：移植裁剪 相关uboot、内核 1-3年的工作经验

2、公司当中，有一个成熟产品线， 成熟的环境。几乎环境不用搭建，用

uboot的命令、内核裁剪(配置菜单)

最重要的：driver(驱动) linux：linux内核框架性的函数接口的头文件所在

arch/arm/mach--xxxx//开发板相关参考文件

3、搭建任何嵌入式linux开发环境。(团队 --- 参考资料)

uboot:

常见的开源的bootloader(初始化硬件(flash、串口、时钟))，引导加操作系统(搬运操作系统)

驱动接口函数：

注册字符设备：

int misc_register(struct miscdevice * misc);

static inline int register_chrdev(unsigned int major, const char *name, const struct file_operations *fops)

int alloc_chrdev_region(dev_t *dev, unsigned baseminor, unsigned count,const char *name)

int register_chrdev_region(dev_t from, unsigned count, const char *name)

struct cdev *cdev_alloc(void)

cdev_init(struct cdev * cdev,const struct file_operations * fops)

int cdev_add(struct cdev *p, dev_t dev, unsigned count)

class_create(owner,name)

device_create(struct class * class,struct device * parent,dev_t devt,void * drvdata,const char * fmt,...)

ioremap(cookie,size)
iounmap(char *p)
函数接口：
int (*open) (struct inode *, struct file *);
int (*release) (struct inode *, struct file *);
可以分离设备号iminor(const struct inode * inode) imajor(const struct inode * inode)
MKDEV(ma,mi)
ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
copy_to_user(to,from,n)
ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
copy_from_user(to,from,n)
long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
作用比杂，自己去编写的控制命令
loff_t (*llseek) (struct file *fp, loff_t, int);
定位 offset = fp->f_ops；根据文件的大小进行定位
unsigned int (*poll) (struct file *, struct poll_table_struct *);
轮询 pollwait填充，轮询等待队列的事件 返回相应的值
int (*fsync) (struct file *, loff_t, loff_t, int datasync);
异步通知 fasync_helper填充 指定的位置用kill_fasync发送指定的信号
int (*mmap)(struct file *, struct vm_area_struct *);
映射地址
中断：
gpio_to_irq(unsigned gpio)
request_irq(unsigned int irq,irq_handler_t handler,unsigned long flags,const char * name,void * dev)
free_irq(unsigned int irq,void * dev_id)
判断是哪一个中断引发的：1、中断编号 2、传递给中断服务函数的参数
void disable_irq(unsigned int irq);
void enable_irq(unsigned int irq);
DECLARE_WAIT_QUEUE_HEAD(name)
wait_event_interruptible(wq,condition,ret)
wake_up_interruptible(x);

内核定时器：
init_timer(timer)
add_timer(struct timer_list * timer)

del_timer(struct timer_list * timer)
mod_timer(struct timer_list * timer,unsigned long expires)
unsigned long msecs_to_jiffies(const unsigned int u)
unsigned long usecs_to_jiffies(const unsigned int u)

内核tasklet机制
tasklet_init(struct tasklet_struct * t,void(* func)(unsigned long),unsigned long data)
tasklet_schedule(struct tasklet_struct * t)
tasklet_kill(tasklet)
INIT_WORK(_work,_func)
schedule_work(_work)
container_of(ptr,type,member)

内核同步机制
sema_init(struct semaphore * sem,int val)
up(struct semaphore * sem)
down(struct semaphore * sem)

spin_lock_init(_lock)
spin_lock(lock,flags)
spin_unlock(lock,flags)
平台设备总线：
platform_device_register(struct platform_device * pdev)
platform_device_unregister(struct platform_device * pdev)
platform_driver_register(struct platform_driver * drv)
platform_driver_unregister(struct platform_driver * drv)
platform_get_resource(struct platform_device * dev,unsigned int type,unsigned int num)

输入子系统：
struct input_dev *input_allocate_device(void)
input_register_device(struct input_dev * dev)