如何开发搭建环境

2017年4月21日 6:53

第三期视频所用的 u-boot kernel 和 nfs文件系统

详细视频在第二期

毕业班第3课第4节_移植3.4.2内核之裁剪及ECC简介及制作补丁.WMV 的42分钟左右

所需文件:fs_mini_mdev_new.yaffs2 u-boot_new.bin uImage_new

初始条件：

板子上什么都没有，没有uboot和其它一切东西。

1.烧写u-boot

所需硬件：openjtag 烧写软件：oflash 烧写内容：u-boot_new.bin

a).在运行cmd指令，开启dos界面

b).进入u-boot.bin所在的目录

例

g:

cd G:\All_workplace\jz2440_lessen\第三期\第三期学前班\原始bin

c).烧写

oflash u-boot_new.bin

烧写到nand flash的第0块

2.烧写内核

所需硬件：网线连接 烧写软件：nfs 烧写内容： uImage_new

a).在u-boot里设置相关的参数

set ipaddr 192.168.1.250

set serverop 192.168.1.254

b).设置和测试虚拟机上的nfs服务器

察看虚拟机上/etc/exports文件 cat /etc/exports，以确保所要连接的文件处于可以访问状态

复位nfs 服务器

sudo service portmap restart

sudo /etc/init.d/nfs-kernel-server restart

检查文件是否能挂载

sudo mount -t nfs localhost:/work/nfs-root /mnt

sudo umount /mnt

c).把window上的相关文件拷贝到虚拟机上的/work/nfs_root文件中

d).在jz2440上使用nfs命令烧写内核

检查连接 ping 192.168.1.254 若出现host 192.168.1.254 is alive便是可以ping通

下载内核uImage_new到uboot

```
nfs 30000000 192.168.1.254:/work/nfs_root/uImage_new
```

擦除内核分区(注意不同的uboot的命令可能不通所以要help nand看下)

```
nand erase.part kernel
nand erase.part kernel
```

烧写内核(注意不同的uboot的命令可能不通所以要help nand看下)

```
nand write.jffs2 30000000 kernel
```

3.烧写文件系统

e).在jz2440上使用nfs命令烧写文件系统

下载文件系统fs_mini_mdev_new.yaffs2到uboot

```
nfs 30000000 192.168.1.254:/work/nfs_root/fs_mini_mdev_new.yaffs2
```

擦除根文件系统

```
nand erase.part rootfs
```

烧写文件系统

```
nand write.yaffs 30000000 260000 $filesize
```

f).在uboot里设置启动参数

使用print来察看bootcmd和bootargs，使用内核启动的话要满足如下的设置

```
bootcmd=nand read 30000000 kernel; bootm 30000000
bootargs=console=ttySAC0,115200 root=/dev/mtdblock3
```

输出设备　　波特率　　从第三块启动

如何设置

```
set 'nand read 30000000 kernel;bootm 30000000'
set bootargs console=ttySAC0,115200 root=/dev/mtdblock3
```

第一课第1节数码相框之系统框架

2017年8月22日 9:04

1.开发流程

弄清需求、设计框架、编写代码、测试。

对于“弄清需求”这个问题，可以参考实际的例子。最终确定需求如图1所示。

对于“设计框架”这个问题关乎性能问题。应该有分层的概念。最终确定如图2所示。

总结如图3所示。

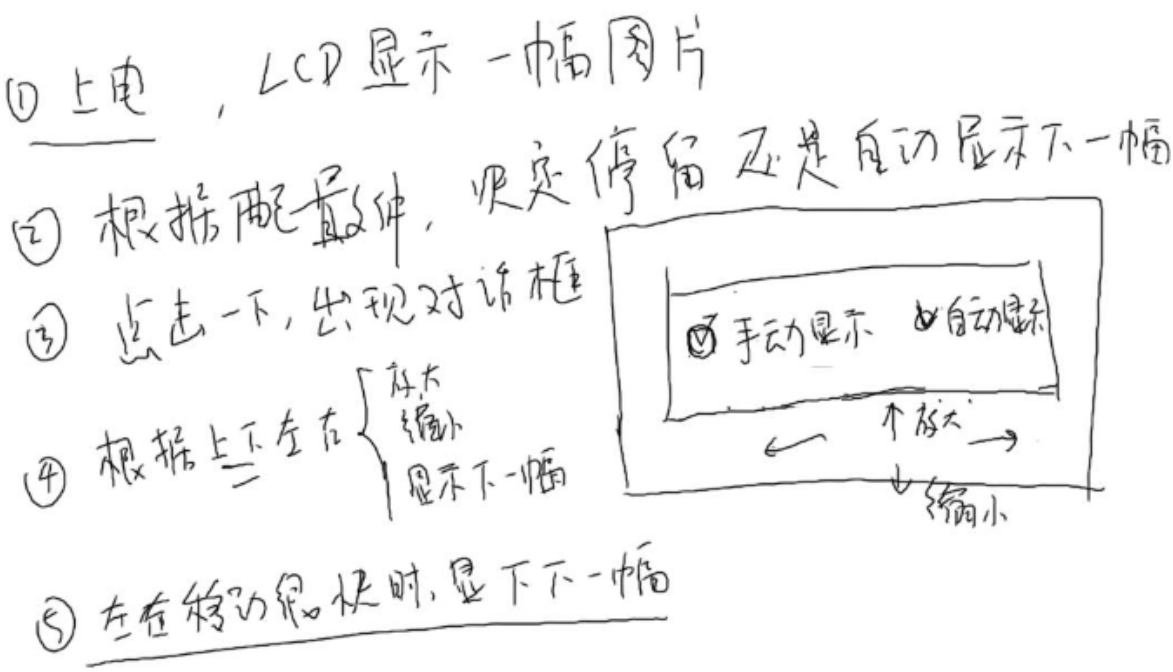


图1

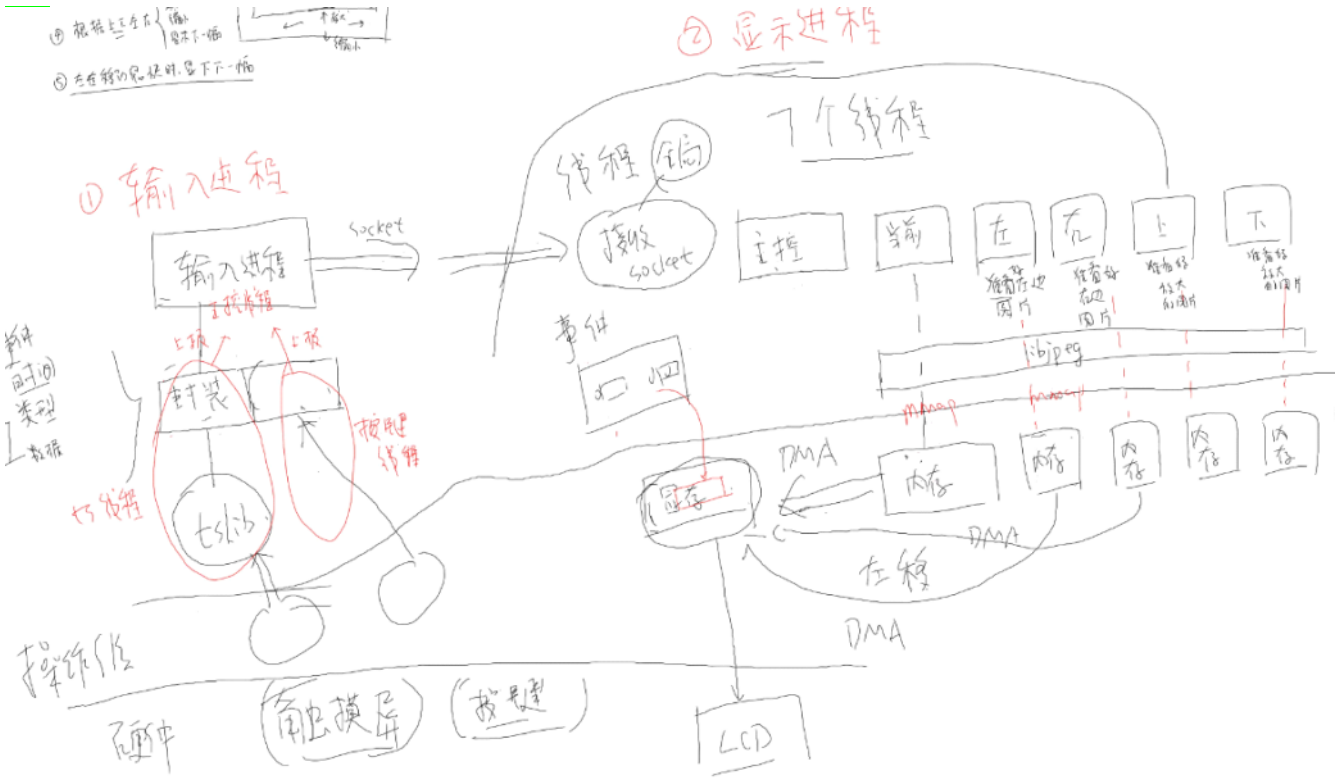


图2

总结

① 输入进程
- 主控线程：得到上报的事件，用socket发出
- ts线程：使用ts库读TS，封装事件，上报
- 按键线程：读按键，封装事件，上报

② 显示进程
- socket线程：接收socket
- 放大线程：准备好当前图片的放大数据
- 缩小线程：准备好当前图片的缩小数据
- 上一幅线程：准备好上一幅图片
- 下一幅线程：准备好下一幅图片
- 当前图片线程：准备好当前图片
- 主控：根据得到的事件，决定显示哪一幅

③ 驱动：分配5块内存，DMA操作，mmap

图3

第一课第3.1节_数码相框额外项目_电子书之代码阅读及编写

2017年8月22日 10:17

讲的内容如图1所示。存在的问题是啥叫“**注册结构体**”啊！

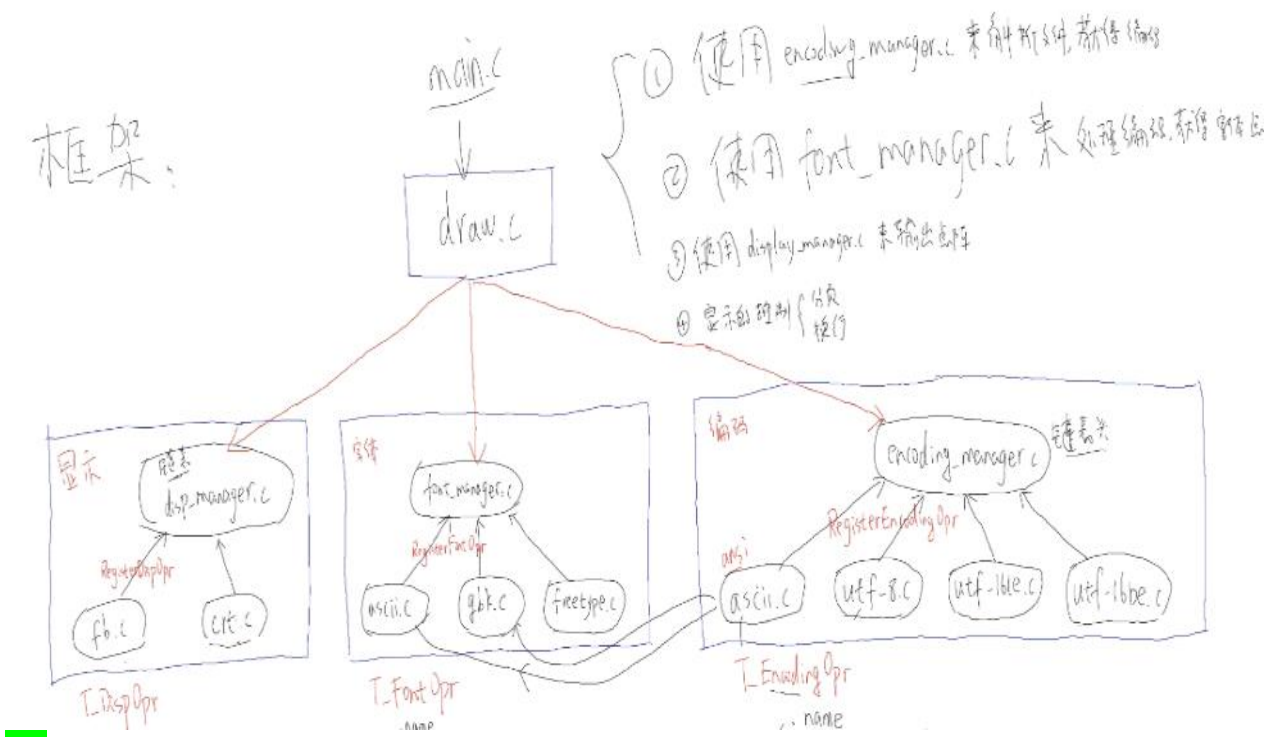
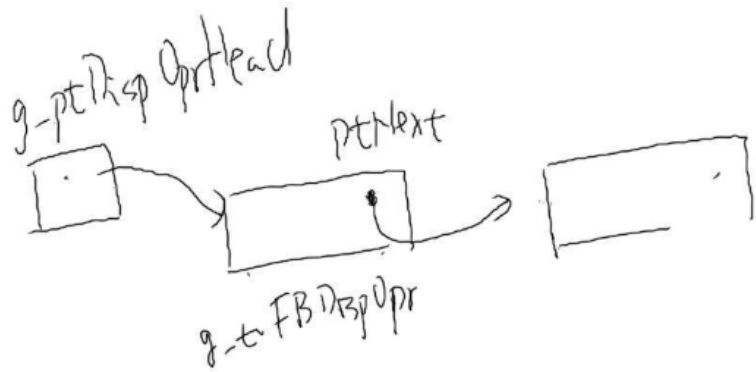


图1



注册结构体：

这里用了一个很高级的用法就是“注册结构体”，它的意义在于：例如板子上可以接多个输出设备时每个设备相同货不同那么这些设备需要统一的管理，“注册结构体”这个方法可以很好地解决这个问题，每个设备都定义其用法，并用结构体来实现类似于面向对象的编程，通过成员函数来定义各项底层操作，随后通过“管理文件.c”来收集所用的设备，通过数组或其他的数据结构来方便管理。例如可以定义一个数组来管理这些输出设备如OutPutDev[10]，但是问题就来了，数组并不灵活，各项数组元素的添加删除都很麻烦，所以这里用了链表这个东西管理设备，通过全局变量g_ptXxxYyyZzzHead这种东西来向更上层的“*功能实现.c*”传递信息。

图2结构的理解：

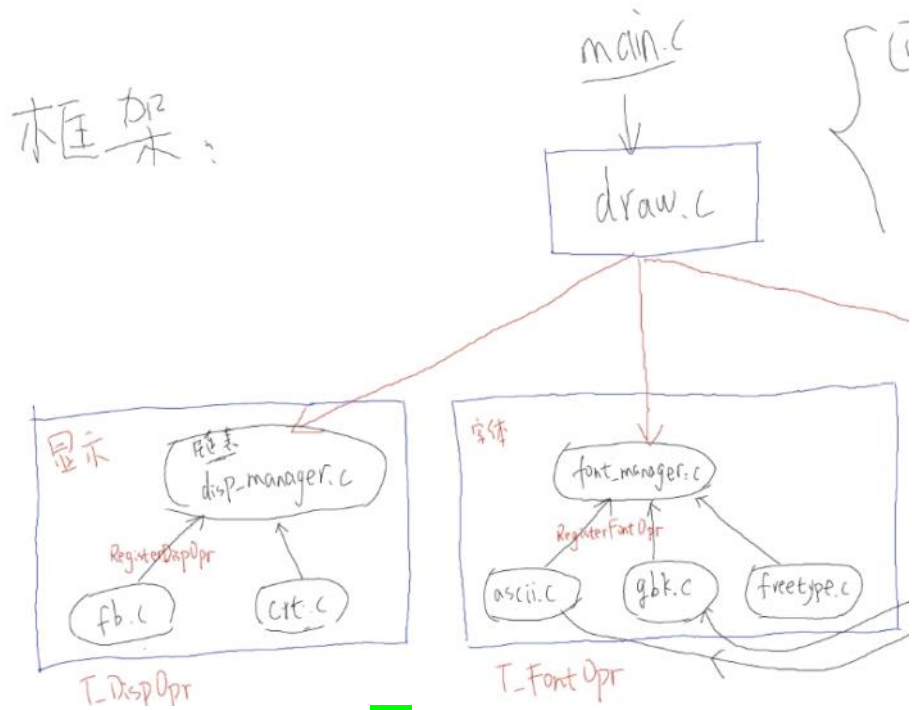


图2

从程序结构上说所有如图2所示的程序结构都可以按照图3的文件结构来进行组织，disp_maneger.c和font_maneger.c分别是是“显示”和"字体"这两个框架的整理者，在这两个文件中提供了对外的函数调用，以及以全局变量的形式提供更下层如fb.c、crt.c、ascii.c、gbk.c、freetype.c等“模块”的链表头，从而方便上层函数调用；draw.c调用各个xxx_maneger.c中的函数和链表头，同时实现另外一些函数功能并被main.c调用，如此便实现了逐级调用。但是如上述的讨论并没有体现fb.c、crt.c、ascii.c、gbk.c、freetype.c这些文件的作用，在XXX_maneger.c中实现RegisterXxxOpr()函数，这个函数将各个“模块”放入相应的链表中，各个“模块”通过定义结构体来实现初始化，在结构体中使用成员函数来实现底层的操作。以上描述可以理解为在fb.c、crt.c、ascii.c、gbk.c、freetype.c这些文件中实现了底层的函数，并定义了XxxOperator结构体将这些底层函数整合；通过其对应的上层xxx_maneger.c给出RegisterXxxOpr()函数来定义不同的链表对各种XxxOperator结构体进行管理，也就实现了对各底层函数的管理；draw.c通过xxx_maneger.c的各种链表操作调用fb.c、gbk.c等中的底层函数，并通过这些函数来实现更为复杂的函数以便完成各种操作；main.c将draw.c中的复杂操作进行整合，实现“逻辑”操作。(**这段写的不好以后会更正——Edit By GYG，20170822**)

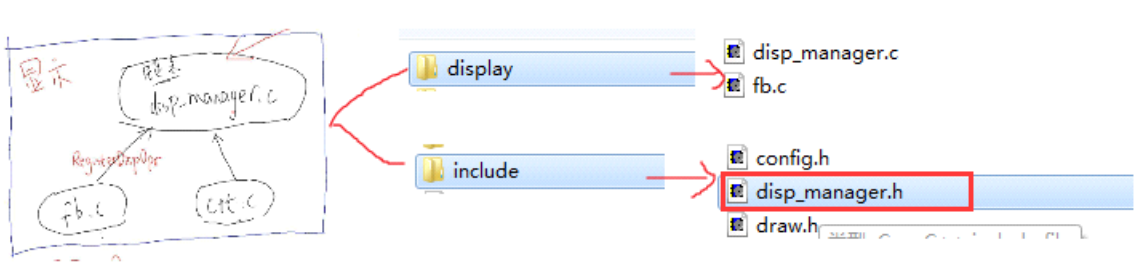


图3

简单地说：

main.c的任务是大面上的方针，比如：我按按钮A,就执行“把大象放冰箱里”的操作。

draw.c的任务是实现方针，比如就要实现“把大象放冰箱里”的操作，需要三个动作“打开冰箱门”、“把大象放进去”、“关上冰箱门”，要将这三个动作有序地组织在一起。

OpenFridgeDoor.c的任务是具体实现动作，比如“打开冰箱门”需要多大的力气、开多大的角度等细枝末节的操作，并通

过XXXX_maneger.c进行整理。

第一课第3.2节_数码相框额外项目_电子书之效果及框架

2017年8月22日 10:16