

作者：粤嵌·温子祺
一、概念
二、驱动程序存在的形式
三、如何将一个驱动程序编译进内核
（一）思路
（二）步骤
四、测试内核镜像的方法
五、快速定位驱动的位置
六、烧写内核
6.1 方式1-fastboot
6.2 方式2-ext4write
拓展
1.内核配置与编译过程
（一）内核配置过程详解
（二）内核编译过程详解
2、menu包含多个配置选项
3、int、hex、string、choice、depends on、select关键字
4、替换开机Logo
5、内核裁剪
附件

一、概念

简单地说，Linux内核里有许多功能或驱动是我们用不上的，我们应该把这些不需要的东西去掉，这样可以让操作系统系统占用内存小，启动速度快。一般嵌入式Linux操作系统需要裁剪。裁剪方法：常用menuconfig配合使用。

内核裁剪的过程，从了解内核所有的支持项，有的是互相依赖（就是选中了该功能，还得选中其他功能，才能实现前者，因为前者功能是依赖于后者），有的支持是互斥的（就是选中了该功能，另外的功能是不允许使用），要根据设计需要选定。内核可裁剪的主要原因是节省硬件资源（首先是flash和内存），裁剪后的内核运行效率也高。另外，开发内核的人不知道你的硬件设计，所以他们尽可能的大而全。

内核默认大小：

1 Created:	Tue Oct 30 19:14:55 2018	//创建时间
2 Image Type:	ARM Linux Kernel Image (uncompressed)	
3 Data Size:	5532560 Bytes	= 5402.89 kB = 5.28 MB //镜像的大小

根据需要裁剪后的大小：

1 Created:	Mon Nov 8 12:34:10 2021	
2 Image Type:	ARM Linux Kernel Image (uncompressed)	
3 Data Size:	4828640 Bytes	= 4715.47 kB = 4.60 MB

二、驱动程序存在的形式

驱动存在形式要么内建于内核，即一直存在于内核，要么是以模块的形式存在。内建内核提高了整体的稳定性，但是增加了内核体积，同时增加了内核引导时间，提高了对内核对内存的占用。使用内核模块利于减少内核体积，可靠性没有内建于内核这么好。

- 1. 将驱动成编译成一个ko，ko是一个独立的module

驱动程序安装：

```
1 #insmod led_drv.ko
```

卸载：

```
1 #rmmod led_drv.ko
```

- 2. 如何开机自动安装驱动，运行应用程序。

```
1 /test/led_drv.ko
2 /test/test
```

修改/etc/profile:

```
1 #vi /etc/profile
2 cd /test
3 insmod led_drv.ko
4 ./test &
5 cd /
```

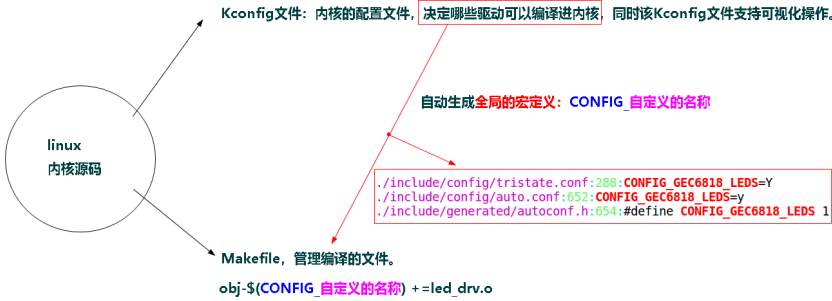
- 3. 将驱动程序编译到linux内核中，使驱动程序和linux内核成为一个整体。当linux内核启动的过程中，会自动的安装驱动。

内核的编译过程如下：

- 1) Image (未压缩的内核映像)
- 2) zImage (压缩)
- 3) mkimage (uboot生成工具) 将zImage加个头
- 4) uImage
- 5) ext4文件系统打包，生成boot.img

三、如何将一个驱动程序编译进内核

(一) 思路



(二) 步骤

1. 将驱动程序放入内核源码

```
1 /kernel/drivers/char/leds/led_drv.c
```

2. 创建并编辑一个Kconfig文件

详细语法参考文件：\Documentation\kbuild\kconfig-language.txt

CONFIG宏变量参数

bool: 表示该CONFIG宏只能选择y(编译内核)或者n(不编译), 不能选择m(编译为模块)

tristate: 表示该CONFIG宏可以设置y/m/n三种模式(tristate)

string: 表示该CONFIG宏可以设为一串字符, 比如#define CONFIG_XXX "config test"

hex: 表示该CONFIG宏可以设为一个十六进制, 比如#define CONFIG_XXX 0x1234

int: 表示该CONFIG宏可以设为一个整数, 比如#define CONFIG_XXX 1234

常用参数

default y: 表示默认是勾上的, 当然也可以写为default m或者default n

help: 帮助提示信息

depends on: 依赖项, 比如depends on XXX 表示当前宏需要CONFIG_ XXX宏打开的前提下, 才能设置它 (注意依赖项的config参数只有bool或tristate才有效)

select : 反依赖项, 和depends on刚好相反, 比如 selecton XXX表示当前宏如果是y或者m, 则会自动设置XXX=y或者m(注意参数只有bool或tristate才有效)

choice: 会生成一个单选框, 里面通过多选一方式选择config, 需要注意choice中的config参数只能bool或tristate

prompt: 提示信息, 如果对于choice而言, 则会用来当做一个单选框入口点的标签

range : 设置用户输入的数据范围, 比如range 0 100表示数据只能位于0~100

menuconfig: menuconfig XXX和config XXX类似, 唯一不同的是该选项除了能设置y/m/n外, 还可以实现菜单效果(能回车进入该项内部)

编辑/kernel/drivers/char/leds/Kconfig (可参考\Documentation\kbuild\kconfig-language.txt 行201开始) 如下:

```
1 menu "GEC6818 LED Driver"
2
3 config GEC6818_LEDS
4     tristate "led drivers for gec6818"
5     depends on ARCH_S5P6818
6     default y
7 help
8     this is a driver for GEC6818, D7(GPIOE13) D8(GPIOC17) D9(GPIOC8) D10(GPIOC7)
9
10 endmenu
```

Kconfig作用: 定义一个条件编译选项和配置菜单。当make menuconfig弹出Kconfig定义的菜单, 对条件编译选项赋值。

分析Kconfig文件:

1) 菜单的开始和结束

```
1 menu "GEC6818 LED Driver"
2
3 endmenu
```

- menu: 菜单的开始
- "GEC6818 LED Driver": 菜单的名字
- endmenu: 菜单的结束

当执行make menuconfig时会看到该菜单配置。

2) config GEC6818_LEDS

定义一个条件编译选项，条件编译选项在使用时候会加一个前缀：CONFIG_。

make menuconfig是对条件编译选项进行赋值，在Makefile编译驱动的时候，会使用该条件编译选项。

3) tristate "led drivers for gec6818"

条件编译选项的值是tristate:

- Y ---> 驱动程序编译进内核
- M ---> 驱动程序不编译进内核，但是在驱动程序的源码目录下，会生成一个ko
- N ---> 驱动程序不编译

扩展:

tristate

Y--<*>,
M--<M>,
N--< >

bool

Y--[*]
N--[]

除了有bool、tristate类型，还有string、hex、int类型的使用。

类型	说明	示例
bool	布尔型，可能值为 0 或者 1，只有选中与不选中两种状态	config DEVKMEM bool "/dev/kmem virtual device support"
tristate	三态型，可能值为 0、1 或者 2，有选中、模块和不选 3 种状态	config IKCONFIG tristate "Kernel .config support"
string	字符串，用于填入字符串，如设置交叉编译器，或者内核命令行参数等	config CMDLINE string "Default kernel command string"
hex	十六进制，常用于填写地址信息	config PAGE_OFFSET hex default 0x40000000 if VMSPLIT_1G default 0x80000000 if VMSPLIT_2G default 0xC0000000
int	整型，用于填写数目，如 CPU 处理器个数、系统 Hz 数等	config NR_CPUS int "Maximum number of CPUs (2-32)" range 2 32 depends on SMP default "4"

4) depends on ARCH_S5P6818

条件编译选项GEC6818_LEDS是依赖于条件编译选项ARCH_S5P6818的。只有ARCH_S5P6818的值为Y的时候，才有机会对ARCH_S5P6818条件编译选项进行赋值，即进入该菜单时变为空白。

5) default y

条件编译选项的值默认是Y

6) help

this is a driver for GEC6818
D7(GPIOE13) D8(GPIOC17) D9(GPIOC8) D10(GPIOC7)

帮助说明，能够在内核配置的时候进入该驱动配置的<help>选项，能够显示如下：

```
CONFIG_GEC6818_LEDS:

this is a driver for GEC6818
D7(GPIOE13) D8(GPIOC17) D9(GPIOC8) D10(GPIOC7)
Symbol: GEC6818_LEDS [=y]
Type : tristate
Prompt: led drivers for gec6818
Defined at drivers/char/leds/Kconfig:3
Location:
-> Device Drivers
-> Character devices
-> GEC6818 LED Driver
```

3. 创建一个Makefile文件

文件创建路径: /kernel/drivers/char/leds/Makefile

编辑内容如下:

```
1 obj-$(CONFIG_GEC6818_LEDS) += led_drv.o
```

4. 修改上一级目录下Kconfig

文件修改路径: /kernel/drivers/char/Kconfig

添加以下内容:

```
1 source "drivers/char/leds/Kconfig"
```

5. 修改上一级目录下的Makefile

文件修改路径: /kernel/drivers/char/Makefile

添加以下内容:

```
1 obj-$(CONFIG_GEC6818_LEDS) += leds/
```

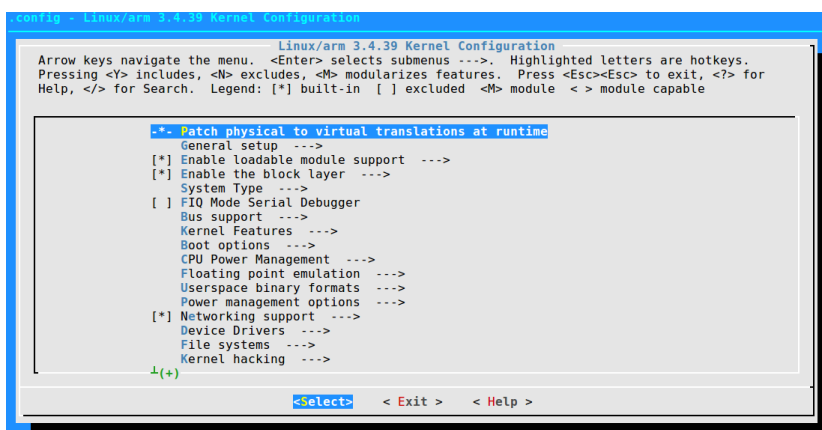
6. 配置内核

1) 拷贝配置文件

```
1 gec@ubuntu:~/6818GEC/kernel$ cp arch/arm/configs/GEC6818_defconfig .config
```

2) 执行make menuconfig命令

```
1 gec@ubuntu:~/6818GEC/kernel$ make menuconfig
```



并在上述菜单找到配置的位置: 搜索条件编译选项

```
1 Device Drivers --->
2   Character devices --->
3     GEC6818 LED Driver --->
4       <*> led drivers for gec6818 (NEW)
```

保存、退出

补充说明:

- 1) 如果使用make menuconfig有以下出错信息, 请安装libncurses5-dev。

```
1 root@ubuntu:~/6818GEC/kernel# make menuconfig
2 *** Unable to find the ncurses libraries or the
3 *** required header files.
4 *** 'make menuconfig' requires the ncurses libraries.
5 ***
6 *** Install ncurses (ncurses-devel) and try again.
7 ***
```

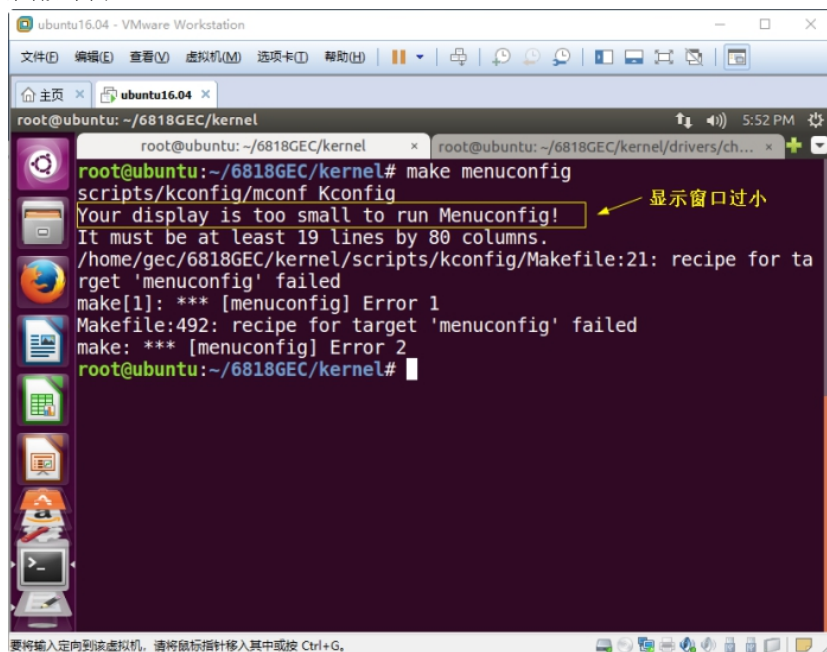
解决方法:

```
1 apt-get install libncurses5-dev
```

- 2) 如果使用make menuconfig有以下出错信息, 请调整终端窗口的大小。

```
1 root@ubuntu:~/6818GEC/kernel# make menuconfig
2 scripts/kconfig/mconf Kconfig
3 Your display is too small to run Menuconfig!
4 It must be at least 19 lines by 80 columns.
5 /home/gec/6818GEC/kernel/scripts/kconfig/Makefile:21: recipe for target 'menuconfig' failed
6 make[1]: *** [menuconfig] Error 1
7 Makefile:492: recipe for target 'menuconfig' failed
8 make: *** [menuconfig] Error 2
9
```

详细如下图:



7. 保存配置

```
1 #cp .config arch/arm/configs/GEC6818_defconfig
```

注:

为什么每次配置完内核之后，都需要拷贝.config文件去覆盖GEC6818_defconfig，原因在于mk脚本文件编译内核的时候，会使用GEC6818_defconfig的配置文件，关键脚本源码如下：

```
1 .....
2 BS_CONFIG_KERNEL=GEC6818_defconfig
3
4 build_kernel()
5 {
6 export PATH=${BS_DIR_UBOOT}/tools:$PATH
7 # Compiler kernel
8 cd ${BS_DIR_KERNEL} || return 1
9 make ${BS_CONFIG_KERNEL} ARCH=arm CROSS_COMPILE=${BS_CROSS_TOOLCHAIN_KERNEL} || return 1
10 .....
```

8. 编译

./mk -k

编译输出是看到的关键信息：

```
1 CC      drivers/char/leds/led_drv.o
2
3 OBJCOPY arch/arm/boot/Image
4 Kernel: arch/arm/boot/Image is ready
5
6 OBJCOPY arch/arm/boot/zImage
7 Kernel: arch/arm/boot/zImage is ready
8
9 UIMAGE  arch/arm/boot/uImage
10 Image arch/arm/boot/uImage is ready
11
12 /home/gec/6818GEC/out/release/boot.img
```

注意事项1:

生成boot.img文件出现错误，make_ext4fs说没有这个文件或目录，问题如下：

```
1 Image arch/arm/boot/uImage is ready
2 boot.img -> /home/arm/6818GEC/out/release
3 '/home/arm/6818GEC/kernel/arch/arm/boot/uImage' -> '/home/arm/6818GEC/out/target/product/GEC6818//boot/uImage'
4 make_ext4fs -s -T -1 67108864 -a boot /home/arm/6818GEC/out/target/product/GEC6818//boot.img /home/arm/6818GEC/out/target/product/GE
5 /home/arm/6818GEC/out/host/linux-x86/bin/mkuserimg.sh:
6 line 84: /home/arm/6818GEC/out/host/linux-x86/bin/make_ext4fs: No such file or directory
7 '/home/arm/6818GEC/out/target/product/GEC6818//boot.img' -> '/home/arm/6818GEC/out/release/boot.img'
```

解决如下：

能够从上面看到make_ext4fs为32位程序，所以我们须要让64位机支持执行32位应用。

运行以下命令就可以解决这个问题：

```
1 sudo apt-get install lib32c-dev lib32stdc++6
2 sudo apt-get install gcc-multilib
```

接着重新编译内核，显示如下正确的信息：

```
1 Image arch/arm/boot/uImage is ready
2 boot.img -> /home/arm/6818GEC/out/release
```

```
3 '/home/arm/6818GEC/kernel/arch/arm/boot/uImage' -> '/home/arm/6818GEC/out/target/product/GEC6818//boot/uImage'
4 make_ext4fs -s -T -1 -l 67108864 -a boot /home/arm/6818GEC/out/target/product/GEC6818//boot.img /home/arm/6818GEC/out/target/product/GE
5 Creating filesystem with parameters:
6   Size: 67108864
7   Block size: 4096
8   Blocks per group: 32768
9   Inodes per group: 4096
10  Inode size: 256
11  Journal blocks: 1024
12  Label:
13  Blocks: 16384
14  Block groups: 1
15  Reserved block group size: 7
16 Created filesystem with 18/4096 inodes and 4134/16384 blocks
17 '/home/arm/6818GEC/out/target/product/GEC6818//boot.img' -> '/home/arm/6818GEC/out/release/boot.img'
```

注意事项2:

若不需要重复拷贝配置文件操作，可以在mk脚本文件中的build_kernel()函数添加以下shell命令语句，详细如下：

```
build_kernel()
{
    export PATH=${BS_DIR_UBOOT}/tools:$PATH
    # Compiler kernel
    cd ${BS_DIR_KERNEL} || return 1
    cp ${BS_DIR_KERNEL}/.config ${BS_DIR_KERNEL}/arch/arm/configs/${BS_CONFIG_KERNEL}
    make ${BS_CONFIG_KERNEL} ARCH=arm CROSS_COMPILE=${BS_CROSS_TOOLCHAIN_KERNEL} || return 1
}
```

四、测试内核镜像的方法

测试内核的方法有两种：第一种方法使用uboot下载uImage到内存启动内核；第二种方法使用fastboot或恢复卡固化到emmc。

方法一：uboot下载uImage并启动内核，这种方法是临时运行内核。

```
1 tftp 0x40008000 uImage
2 bootm 0x40008000
```

方法二：将boot.img烧写到emmc

1. 固化boot.img有以下两种方法

1) fastboot

2) 恢复卡

注：fastboot烧写时候，电脑提示安装驱动，可以使用驱动精灵来安装。

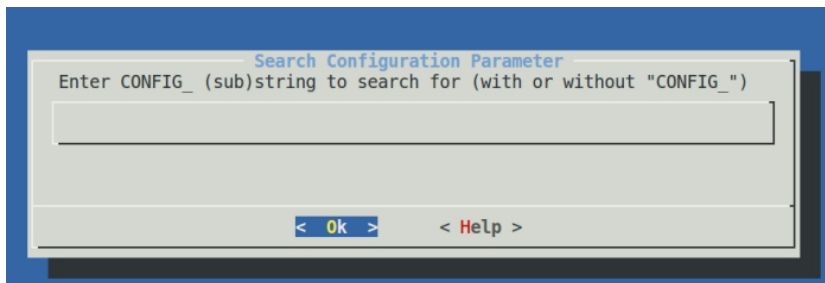
```
1 [root@GEC6818 /]#dmesg | head -50
2 [ 0.000000] Booting Linux on physical CPU 0
3 [ 0.000000] Initializing cgroup subsys cpu
4 [ 0.000000] Linux version 3.4.39-gec (gec@ubuntu) (gcc version 4.8 (GCC) ) #1 SMP PREEMPT Fri Feb 14 15:51:34 CST 2020
```

```
[root@GEC6818 /]#dmesg | grep "gec6818 led"
```

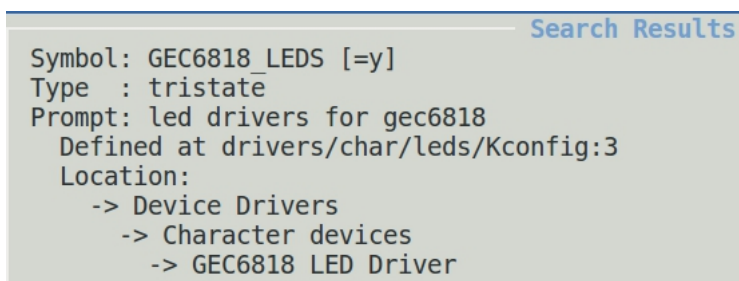
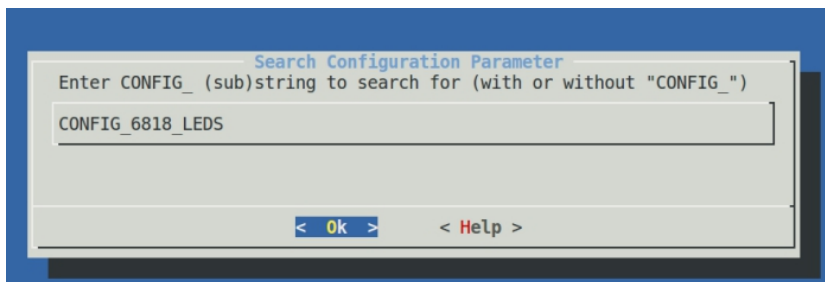
```
[ 1.469000] gec6818 led init
```

五、快速定位驱动的位置

1. 当进入内核配置菜单后，输入‘/’后，会弹出以下界面。



2. 例如要定位刚添加的led驱动，可以在该界面输入“CONFIG_GEC6818_LEDS”，并选中“<ok>”后回车。



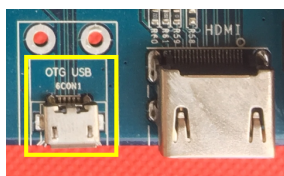
注：

关于此前冲突的LED与ADC驱动，也可以搜索CONFIG_LEDS_GPIO与CONFIG_NXP_ADC该宏定义，快速定位到对应的位置，去掉两个驱动的编译。

六、烧写内核

6.1 方式1-fastboot

1. 开发板通过该



接口，使用micro usb线（支持数据传输功能，不能是充电线）连接到电脑。

2. 重启开发板进入uboot状态，并输入命令“fastboot”。

GEC6818# fastboot

提示：

这个时候，如果电脑没有安装好相应的驱动，会提示安装驱动，安装驱动使用“fastboot驱动.rar”进行安装，安装成功后在设备管理器当中显示以下结果：

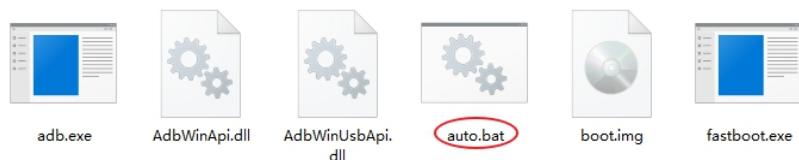
[win7]



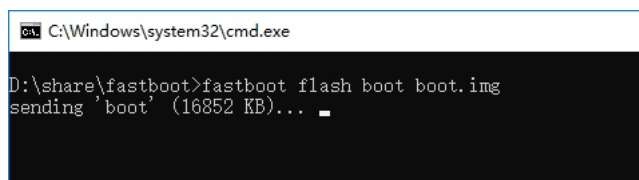
[win10]



3. 保证boot.img在fastboot文件夹当中是最新编译好的linux内核镜像，接着双击运行“auto.bat”脚本。



4. 这个时候会弹出命令窗口且uboot会不断刷新烧录进度如下：



```
-----
Starting download of 17256680 bytes
downloading 10354688 -- 60% complete.
```

注：

auto.bat内容如下：

```
1 #通过fastboot工具将boot.img烧写到开发板的boot分区
2 fastboot flash boot boot.img
3
4 #烧写成功后，执行复位操作
5 fastboot reboot
```

5. 烧录成功后会自动启动linux内核。

6.2 方式2-ext4write

1. uboot下载uImage到内存地址0x40008000。

```
1 tftp 0x40008000 uImage
```

2. 在uboot命令行模式下，执行ext4write命令。

```
1 # ext4write mmc 2:1 40008000 /uImage uImage文件大小
2
3 File System is consistent //文件系统一致
4 file found deleting //重名文件删除
5 update journal finished //更新日志完成
6 File System is consistent //文件系统一致
7 update journal finished //添加新的文件后，更新日志完成
```

3. 执行ext4ls mmc 2:1命令可以看到当前分区文件。

```
1
ext4ls mmc 2:1
2 <DIR> 4096 .
3 <DIR> 4096 ..
4 <DIR> 4096 lost+found
5 230456 battery.bmp
6 3798425 debug-ramdisk.img
7 1152054 logo.bmp
8 1213716 ramdisk-recovery.img
9 611566 root.img.gz
10 5598176 uImage
11 280854 update.bmp
```

拓展

1. 内核配置与编译过程

（一）内核配置过程详解

1. Linux内核的配置系统由三个部分组成，分别是：

- **Makefile**——分布在 Linux 内核源代码根目录及各层目录中，定义 Linux 内核的编译规则；
- **Kconfig**——分布在 Linux 内核源代码根目录及各层目录中，给用户提供了配置选择的功能；
- **scripts**——分布在 Linux内核源代码根目录下，包括配置命令解释器（对配置脚本中使用的配置命令进行解释）和配置用户界面（提供基于字符界面、基于 Ncurses 图形界面以及基于 Xwindows 图形界面的用户配置界面，各自对应于 Make config、Make menuconfig 和 make xconfig）。

2. 当我们使用**make menuconfig**这个命令时（其它配置命令类似）：

- 首先由make编译生成**scripts/kconfig/mconf.c**生成**scripts/kconfig/mconf**。（xconfig对应qconf，gconfig对应gconf，config对应conf）
 - 然后执行**scripts/kconfig/mconf Kconfig**
 - mconf程序读取内核根目录下的Kconfig文件，Kconfig载入了**arch/\$SRCARCH/Kconfig**，**arch/\$SRCARCH/Kconfig**又分别载入各目录下的Kconfig文件，以此递归下去，最后生成主配置界面以及各级配置菜单。**\$SRCARCH**是由顶层Makefile中定义的，它等于**\$ARCH**，而**\$ARCH**由Makefile或make的命令行参数指定。
 - 在完成配置后，mconf会将配置保存在Linux内核源代码根目录下的**.config**文件中。
3. 当我们使用**make defconfig**这个命令时：
- 系统直接将**arch/\$SRCARCH/configs**（该目录存放内核的默认配置文件）下的对应的默认配置文件拷贝到Linux内核源代码根目录下的**.config**文件。

（二）内核编译过程详解

1. 在输入编译命令后，make首先调用脚本来读取**.config**文件，并根据内容载入对应文件到**include/config/**，并将一些配置项（配置符号）写入**include/config/auto.conf**。

2. 脚本程序将**include/config/auto.conf**中的配置项（配置符号）**CONFIG_XXX=y|m|xxx**翻译为宏定义**#define CONFIG_XXX[_MODULE] 1|xxx**，并写入**include/generated/autoconf.h**中。

3. autoconf.h作用就是将.config翻译为C语言中能识别的头文件，以便在以后使用的时候作为宏定义出现，以实现条件编译。

4. make根据Makefile执行编译。

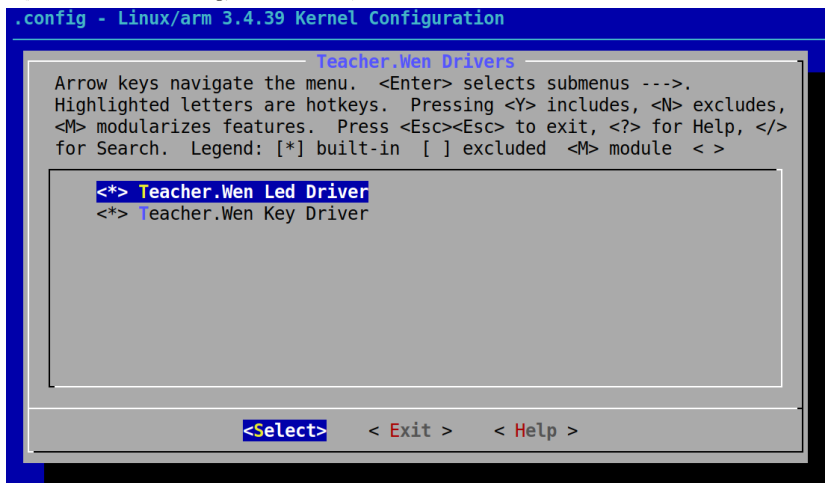
2、menu包含多个配置选项

一、基于Kconfig，创建menu包含多个配置选项。

1.Kconfig内容如下：

```
1 menu "Teacher.Wen Drivers"
2 config MY_LED
3     tristate "Teacher.Wen Led Driver"
4     default y
5     help
6         d7(gpioe13) d8(gpioc17) d9(gpioc8) d10(gpioc7)
7 config MY_KEY
8     tristate "Teacher.Wen Key Driver"
9     default y
10    help
11        k2(gpioa28) k3(gpiob9) k4(gpiob31) k6(gpiob30)
12 endmenu
```

2. 执行make menuconfig, 进入指定的菜单。



3、int、hex、string、choice、depends on、select关键字

官方参考例子：6818GEC\kernel\kernel\Kconfig.hz

在项目中的用途：可以通过该配置选项来控制硬件的访问。

. 通过int、hex类型从配置选项来控制超声波测量的距离、手势识别模块能够识别手势的数量。通过系统升级来决定的，通过4S店这种服务最常见。

. 通过string类型的配置选项可以控制显示信息，如LCD、OLED显示开机的信息（企业信息、版本信息、其他信息）

一、基于Kconfig，实现宏定义数值，以CONFIG_MY_LED_N为例。

示例1：直接使用int、hex、string关键字。

1.Kconfig内容如下：

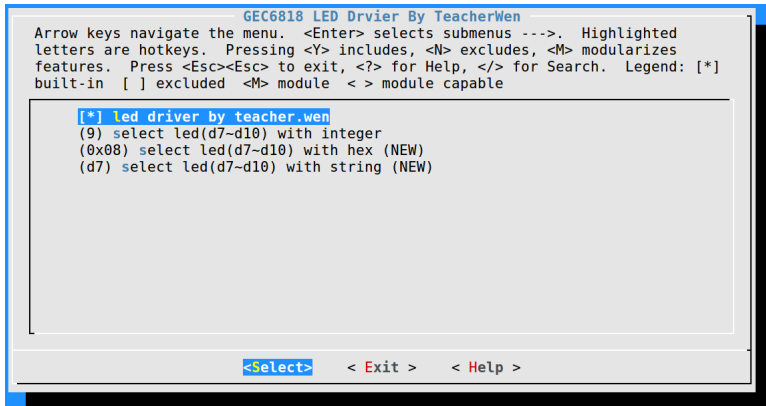
```
1 menu "GEC6818 LED Drvier By TeacherWen"
2
3 config MY_LED
```

```

4     bool "led driver by teacher.wen"
5     default y
6     help
7         d7(gpioe13) d8(gpioc17) d9(gpioc8) d10(gpioc7)
8
9 config MY_LED_N
10    int "select led(d7~d10) with integer"
11    range 7 10
12    default 8
13    help
14        d7(gpioe13) d8(gpioc17) d9(gpioc8) d10(gpioc7)
15
16 config MY_LED_HEX
17    hex "select led(d7~d10) with hex"
18    range 0x07 0x0A
19    default 0x08
20    help
21        d7(gpioe13) d8(gpioc17) d9(gpioc8) d10(gpioc7)
22
23 config MY_LED_STR
24    string "select led(d7~d10) with string"
25    default "d7"
26    help
27        d7(gpioe13) d8(gpioc17) d9(gpioc8) d10(gpioc7)
28 endmenu

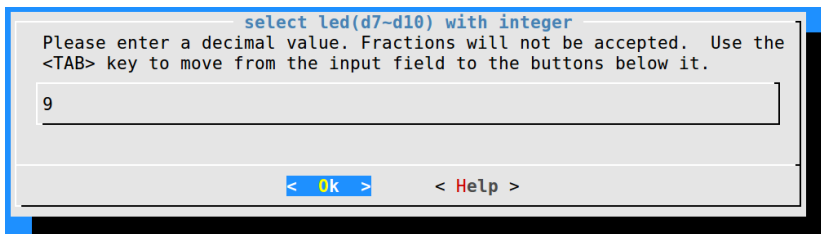
```

2. 执行make menuconfig, 进入指定的菜单。

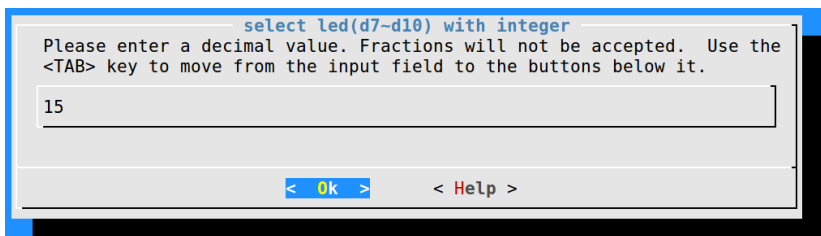


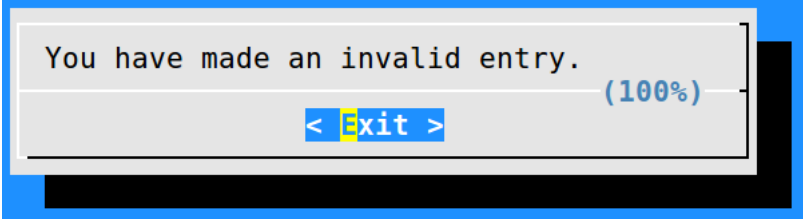
3. int-配置数值

1) 合理范围。



2) 非法范围。

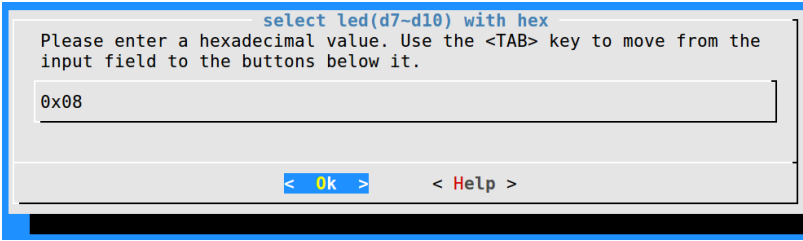




You have made an invalid entry. (100%)
< Exit >

4. hex-配置数值

1) 合理范围。

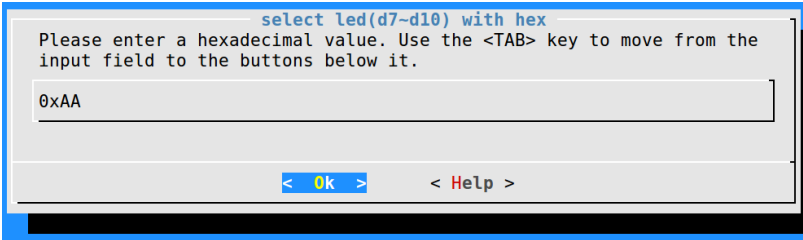


select led(d7-d10) with hex
Please enter a hexadecimal value. Use the <TAB> key to move from the input field to the buttons below it.

0x08

< Ok > < Help >

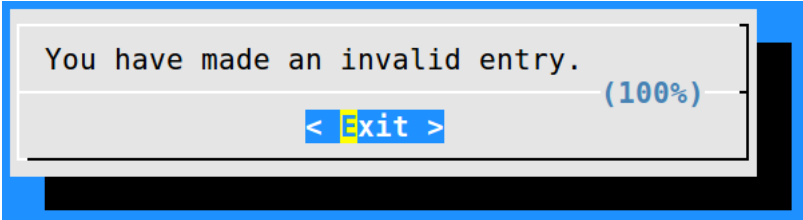
2) 非法范围。



select led(d7-d10) with hex
Please enter a hexadecimal value. Use the <TAB> key to move from the input field to the buttons below it.

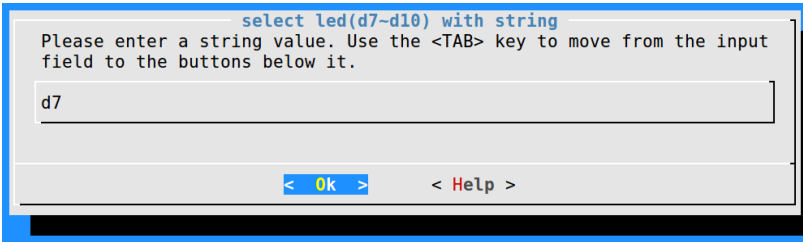
0xAA

< Ok > < Help >



You have made an invalid entry. (100%)
< Exit >

5. string-配置字符串



select led(d7-d10) with string
Please enter a string value. Use the <TAB> key to move from the input field to the buttons below it.

d7

< Ok > < Help >

6. 执行make menuconfig后，找到CONFIG_MY_LED_N宏定义

```
1 gec@ubuntu:~/6818GEC/kernel$ grep "CONFIG_MY_LED_N" -r -n .  
2 ../config:1458:CONFIG_MY_LED_N=7
```

7. 重新编译内核，生成新的内核，为模块的使用提供新的配置信息。

```
1 gec@ubuntu:~/6818GEC/kernel$ grep "CONFIG_GEC6818_MY_N" -r -n .  
2 ./include/config/auto.conf:134:CONFIG_MY_LED_N=7  
3 ./include/generated/autoconf.h:136:#define CONFIG_MY_LED_N 7  
4 ../config:1458:CONFIG_MY_LED_N=7
```

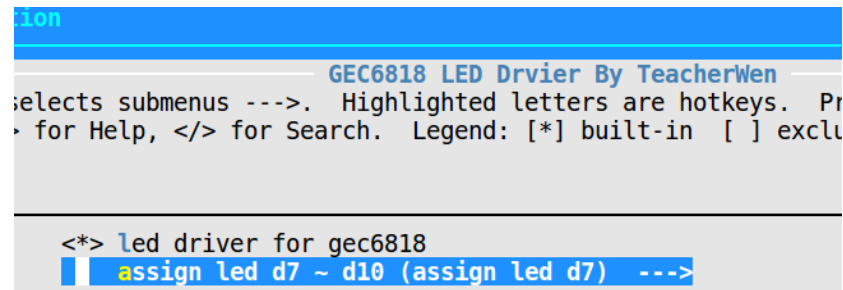
8. 重新编译内核模块源码。

示例2：使用choice关键字，显示选择菜单，提高交互。

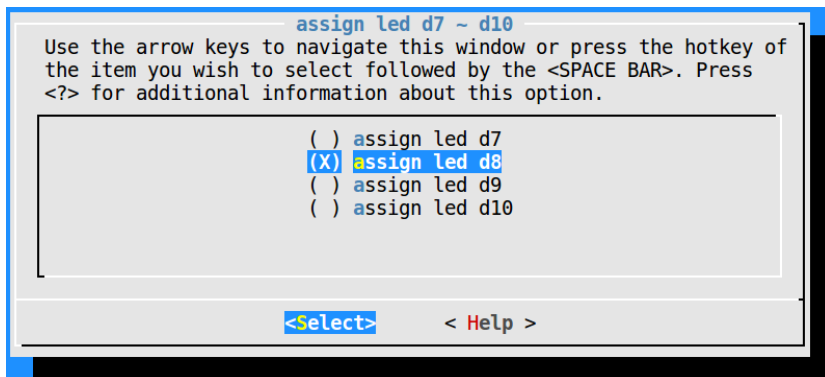
1. Kconfig内容如下：

```
1 menu "GEC6818 LED Drvier By TeacherWen"
2
3 config GEC6818_LEDS
4     tristate "led driver for gec6818"
5     default y
6     help
7         this is a driver for gec6818,d7(gpioe13)d8(gpioc17)d9(gpioc8)d10(gpioc7)
8 choice
9     prompt "assign led d7 ~ d10"
10    help
11        assign led d7 ~ d10
12
13    config LED_IS_D7
14        bool "assign led d7"
15
16    config LED_IS_D8
17        bool "assign led d8"
18
19    config LED_IS_D9
20        bool "assign led d9"
21
22    config LED_IS_D10
23        bool "assign led d10"
24 endchoice
25
26 config GEC6818_LED_N
27     int
28     default 7 if LED_IS_D7
29     default 8 if LED_IS_D8
30     default 9 if LED_IS_D9
31     default 10 if LED_IS_D10
32 endmenu
```

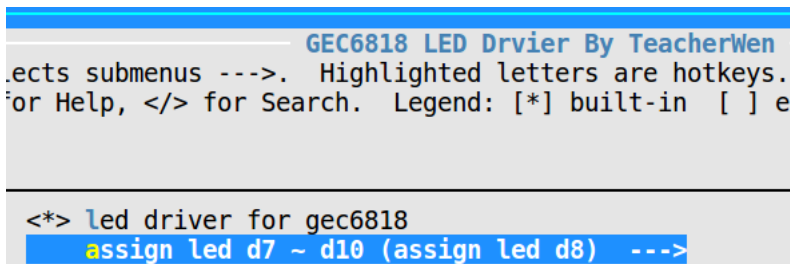
2. 执行make menuconfig,进入指定的菜单，看到提示：当前指定的led为d7灯。



3. 进入“assign led d7 ~ d10”选择菜单，并选中“(X) assign led d8”。



4. 返回后，看到提示：当前指定的led为d7灯。



4. 执行make menuconfig后，找到CONFIG_GEC6818_LED_N宏定义

```
1 gec@ubuntu:~/6818GEC/kernel$ grep "CONFIG_GEC6818_LED_N" -r -n .
2 ./config:1458:CONFIG_GEC6818_LED_N=7
```

5. 重新编译内核，生成新的内核，为模块的使用提供新的配置信息。

```
1 gec@ubuntu:~/6818GEC/kernel$ grep "CONFIG_GEC6818_LED_N" -r -n .
2 ./include/config/auto.conf:134:CONFIG_GEC6818_LED_N=7
3 ./include/generated/autoconf.h:136:#define CONFIG_GEC6818_LED_N 7
4 ./config:1458:CONFIG_GEC6818_LED_N=7
```

二、参考驱动关键源码，指定对应的LED灯运行。

```
1 long led_ioctl (struct file *fp, unsigned int cmd, unsigned long arg)
2 {
3     //arg:7~10
4     switch(cmd)
5     {
6         case GEC6818_LED_ON:
7             {
8                 #if CONFIG_GEC6818_LED_N == 7
9                     if(arg == 7)gpio_set_value(PAD_GPIO_E + 13,0);
10                #endif
11
12                #if CONFIG_GEC6818_LED_N == 8
13                    if(arg == 8)gpio_set_value(PAD_GPIO_C + 17,0);
14                #endif
15
16                #if CONFIG_GEC6818_LED_N == 9
17                    if(arg == 9)gpio_set_value(PAD_GPIO_C + 8,0);
18                #endif
19
20                #if CONFIG_GEC6818_LED_N == 10
```



```

21         if(arg == 10)gpio_set_value(PAD_GPIO_C + 7,0);
22     #endif
23     }break;
24
25     case GEC6818_LED_OFF:
26     {
27     #if CONFIG_GEC6818_LED_N == 7
28         if(arg == 7)gpio_set_value(PAD_GPIO_E + 13,1);
29     #endif
30
31     #if CONFIG_GEC6818_LED_N == 8
32         if(arg == 8)gpio_set_value(PAD_GPIO_C + 17,1);
33     #endif
34
35     #if CONFIG_GEC6818_LED_N == 9
36         if(arg == 9)gpio_set_value(PAD_GPIO_C + 8,1);
37     #endif
38
39     #if CONFIG_GEC6818_LED_N == 10
40         if(arg == 10)gpio_set_value(PAD_GPIO_C + 7,1);
41     #endif
42     }break;
43
44     default:
45         return -ENOIOCTLCMD;
46     }
47     return 0;
48 }

```

示例3：直接使用depends on关键字。

```

1 menu "GEC6818 LED Drvier By TeacherWen"
2
3 config GEC6818_LEDS
4     tristate "led driver for gec6818"
5     default y
6     help
7         this is a driver for gec6818,d7(gpioe13)d8(gpioc17)d9(gpioc8)d10(gpioc7)
8 choice
9     prompt "assign led d7 ~ d10"
10    depends on GEC6818_LEDS
11    help
12        assign led d7 ~ d10
13
14    config LED_IS_D7
15        bool "assign led d7"
16
17    config LED_IS_D8
18        bool "assign led d8"
19
20    config LED_IS_D9
21        bool "assign led d9"
22
23    config LED_IS_D10
24        bool "assign led d10"
25 endchoice
26
27 config GEC6818_LED_N

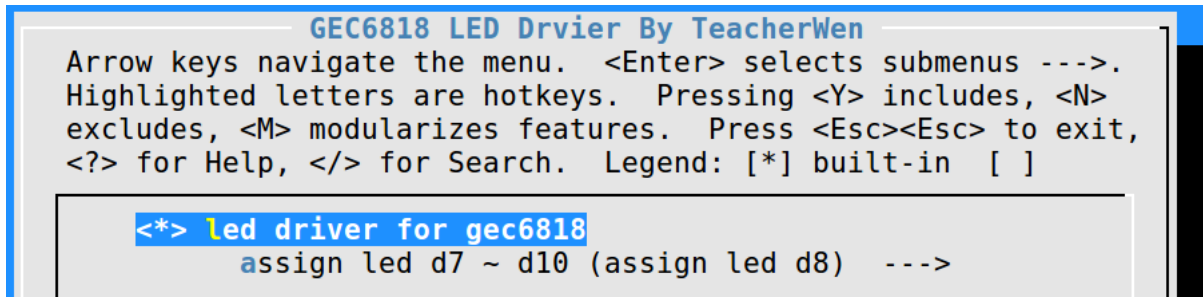
```

```

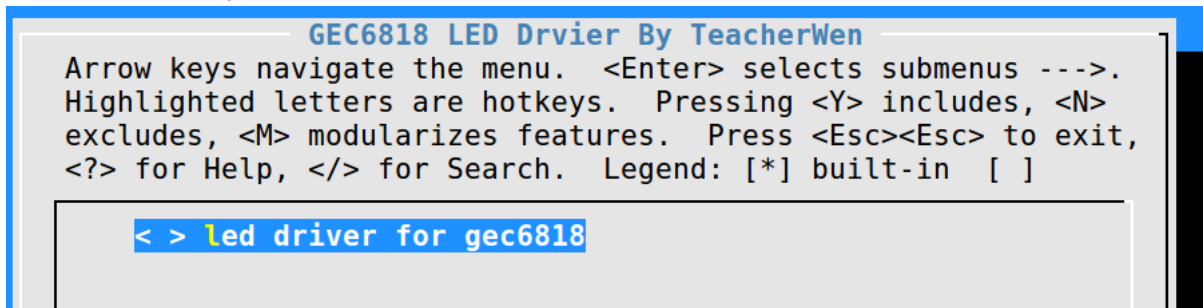
28     int
29     default 7  if LED_IS_D7
30     default 8  if LED_IS_D8
31     default 9  if LED_IS_D9
32     default 10 if LED_IS_D10
33 endmenu

```

- 选中“led driver for gec6818”，则展开详细的配置。



- 不选中“led driver for gec6818”，则不展开详细的配置。



示例4：直接使用select关键字。

```

1 menu "GEC6818 LED Drvier By TeacherWen"
2
3 config GEC6818_KEYS
4     tristate "key driver for gec6818"
5     select GEC6818_LEDS
6     help
7         this is a driver for gec6818
8
9 config GEC6818_LEDS
10    tristate "led driver for gec6818"
11    default y
12    help
13        this is a driver for gec6818,d7(gpioe13)d8(gpioc17)d9(gpioc8)d10(gpioc7)
14
15 choice
16     prompt "assign led d7 ~ d10"
17     depends on GEC6818_LEDS
18     help
19         assign led d7 ~ d10
20
21     config LED_IS_D7
22         bool "assign led d7"
23
24     config LED_IS_D8
25         bool "assign led d8"
26

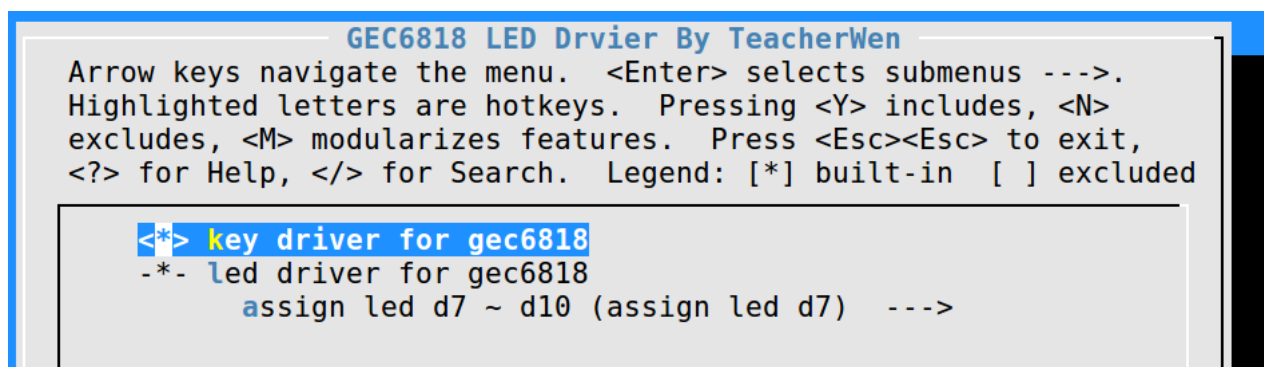
```

```

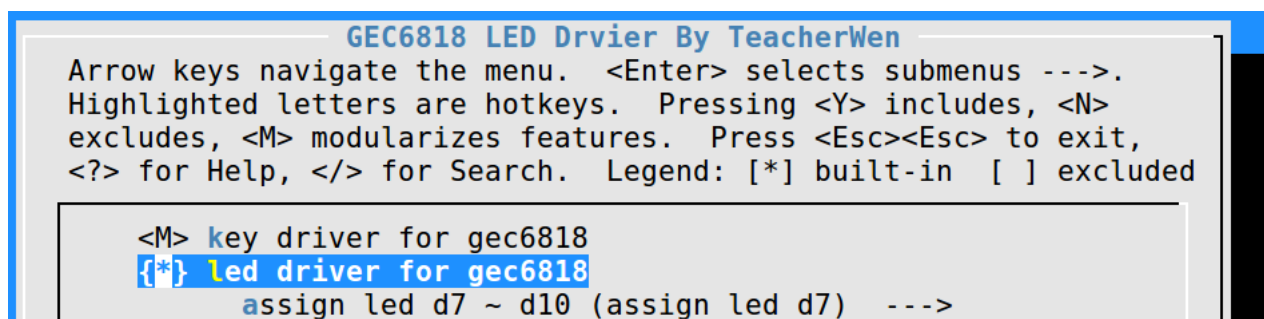
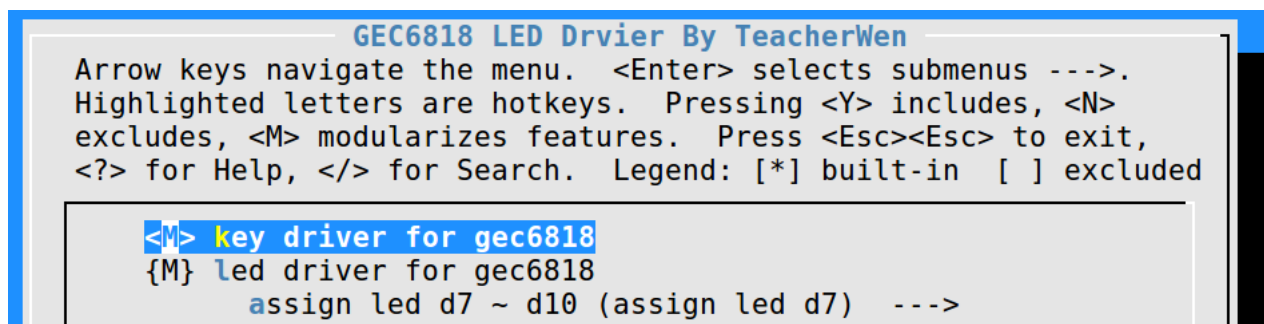
27     config LED_IS_D9
28         bool "assign led d9"
29
30     config LED_IS_D10
31         bool "assign led d10"
32 endchoice
33
34 config GEC6818_LED_N
35     int
36     default 7 if LED_IS_D7
37     default 8 if LED_IS_D8
38     default 9 if LED_IS_D9
39     default 10 if LED_IS_D10
40 endmenu

```

- 选中“key driver for gec6818”，会同步选中“led driver for gec6818”



- 选中“key driver for gec6818”为内核模块，会同步选中“led driver for gec6818”为内核模块，也可以编译进内核。



注：

本来“led driver for gec6818”有三种选择，由于“key driver for gec6818”的select的关系，只能在编译进内核与编译为内核模块来回选择。

4、替换开机Logo

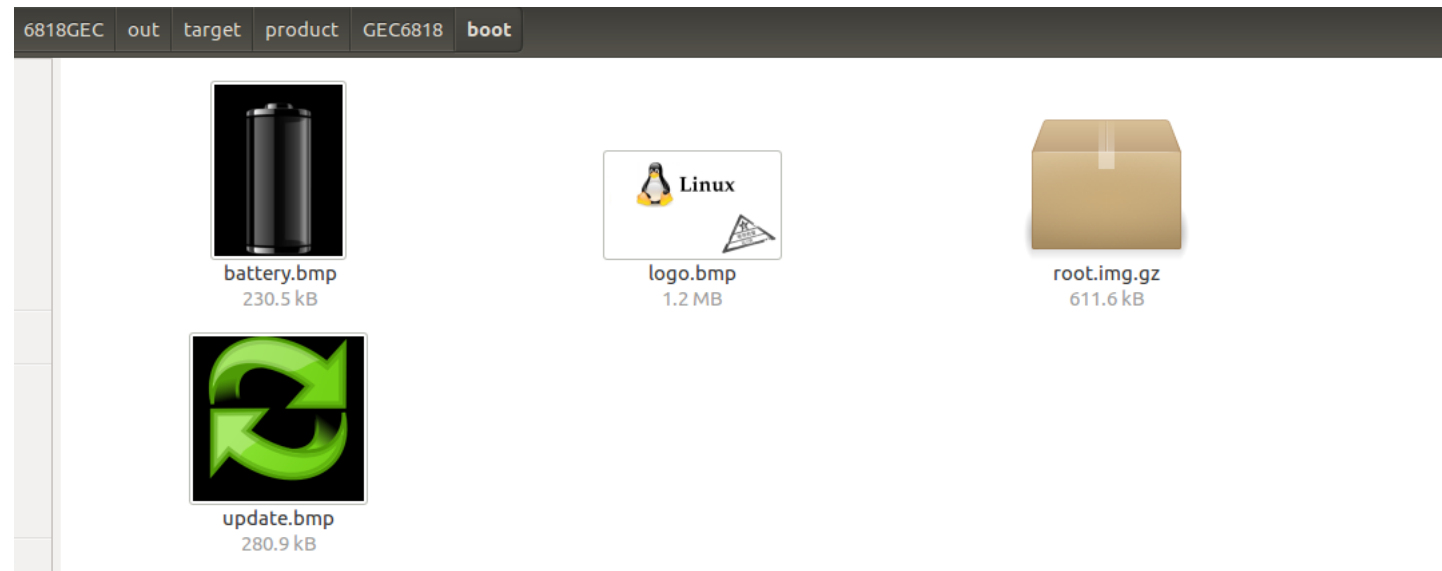
替换开机Logo。

- 1) 装备好一张800x480大小的bmp图片，格式为24位色，文件名为:logo.bmp

2) 拷贝到6818GEC/out/target/product/GEC6818/boot/目录下，覆盖原有的logo.bmp图片。

3) 重新编译内核

4) 使用fastboot烧录到开发板



原理

- **uboot.lds文件**

u-boot.lds决定了u-boot可执行映像的连接方式，以及各个段的装载地址（装载域）和执行地址（运行域）。

```
1 OUTPUT_FORMAT("elf32-littlearm", "elf32-littlearm", "elf32-littlearm")
2 OUTPUT_ARCH(arm)
3 ENTRY(_stext)
4 SECTIONS
5 {
6     . = 0x00000000;
7     . = ALIGN(4);
8     .text :
9     {
10        *(__image_copy_start)
11        #第一个执行的文件
12        arch/arm/cpu/slsiap/s5p6818/start.o (.text*)
13        arch/arm/cpu/slsiap/s5p6818/vectors.o (.text*)
14        *(.text*)
15    }
16
17    .....
18 }
```

- **执行流程**

6818GEC\GEC6818uboot\arch\arm\cpu\slsiap\s5p6818\start.S

```
1 _stext:
2     .....
3     /* call board_init_r */
4     ldr    pc, =board_init_r    /* this is auto-relocated! */
```

6818GEC\GEC6818uboot\arch\arm\lib\board.c

```
1 void board_init_r(gd_t *id, ulong dest_addr)
2 {
3     .....
4     #ifdef CONFIG_BOARD_LATE_INIT
```

```

5     board_late_init();
6 #endif
7 .....
8 }
9

```

6818GEC\GEC6818uboot\board\s5p6818\GEC6818\board.c

```

1 int board_late_init(void)
2 {
3 #if defined(CONFIG_SYS_MMC_BOOT_DEV)
4     char boot[16];
5     //指定boot分区位置
6     sprintf(boot, "mmc dev %d", CONFIG_SYS_MMC_BOOT_DEV);
7     run_command(boot, 0);
8 #endif
9
10    //初始化lcd
11    GEC6818_lcd_select();
12
13    .....
14 #if defined(CONFIG_DISPLAY_OUT)
15     //显示开机Logo
16     bd_display_run(CONFIG_CMD_LOGO_WALLPAPERS, CFG_LCD_PRI_PWM_DUTYCYCLE, 1);
17 #endif
18
19    .....
20    return 0;
21 }

```

思考：只要更新开发板开机Logo，就得fastboot一遍，甚是麻烦，能否有更好的方法？

答案：可以将logo.bmp放到根文件系统，并修改uboot源码。

操作步骤：

(1) 将要显示的logo.bmp下载到开发板的根文件系统



(2) 修改uboot源码指定logo.bmp加载位置，即从根文件系统加载logo.bmp，源码文件路径：6818GEC/GEC6818uboot/include/configs/GEC6818.h

```

1 #define CONFIG_CMD_LOGO_WALLPAPERS    "ext4load mmc 2:1 0x47000000 logo.bmp; drawbmp 0x47000000"

```

“2:1” 修改为 “2:2”，如下。

```

1 #define CONFIG_CMD_LOGO_WALLPAPERS    "ext4load mmc 2:2 0x47000000 logo.bmp; drawbmp 0x47000000"

```

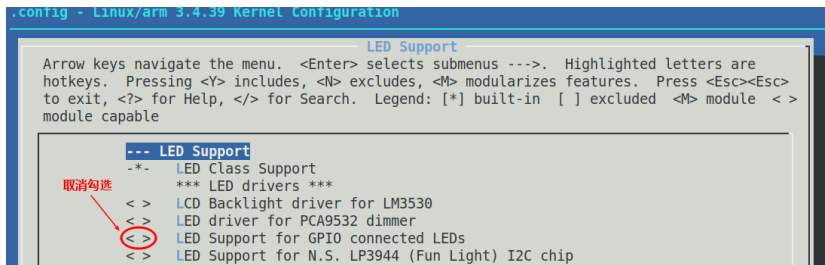
(3) 执行./mk -u命令编译uboot源码。

(4) 将生成的GECuboot.bin文件覆盖fastboot目录下的GECuboot.bin文件。

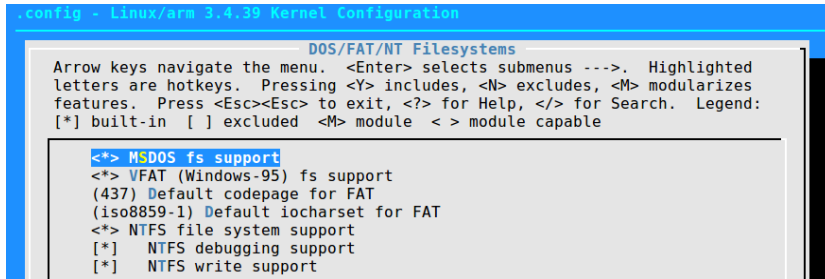
(5) 使用fastboot烧录到开发板

5、内核裁剪

任务1：裁剪内核中LED的驱动。



任务2: 裁剪多余的文件系统, 例如将MSDOS、NTFS、VFAT去掉。



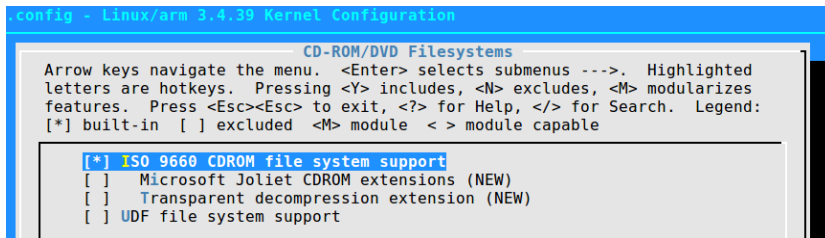
最后将配置好的内核放到开发板运行, 并输入命令:

```
1 cat /proc/filesystems
```

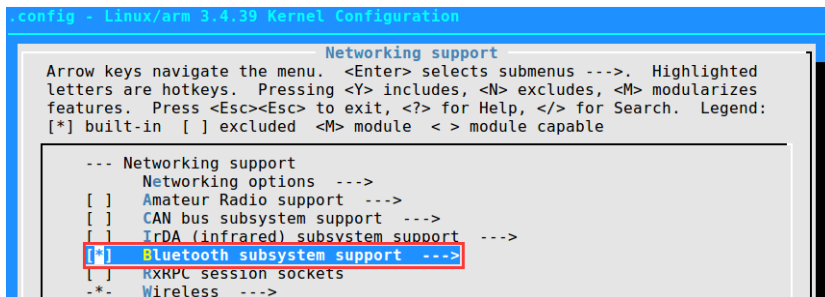
可以发现相应的文件系统被裁剪掉。

任务3: 裁剪多有的硬件, 例如CD-ROM、蓝牙、Wireless。更多的硬件裁剪同学们可以继续深挖。

- CD-ROM



- 蓝牙



- 无线WiFi

```
.config - Linux/arm 3.4.39 Kernel Configuration

Wireless
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted
letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes
features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend:
[*] built-in [ ] excluded <M> module < > module capable

--- Wireless
<*> cfg80211 - wireless configuration API
[*] nl80211 testmode command
[ ] enable developer warnings
[ ] cfg80211 regulatory debugging
[ ] enable powersave by default
[ ] cfg80211 DebugFS entries
[ ] use statically compiled regulatory rules database
[*] cfg80211 wireless extensions compatibility
[ ] Wireless extensions sysfs files
< > Common routines for IEEE802.11 drivers
[ ] Allow reconnect while already connected
<*> Generic IEEE 802.11 Networking Stack (mac80211)
[ ] PID controller based rate control algorithm
[*] Minstrel
[*] Minstrel 802.11n support
Default rate control algorithm (Minstrel) --->
[ ] Enable mac80211 mesh networking (pre-802.11s) support

+
(+)
```

任务4：裁剪内核模块中不需要的功能。

```
.config - Linux/arm 3.4.39 Kernel Configuration

Linux/arm 3.4.39 Kernel Configuration
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted
letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes
features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend:
[*] built-in [ ] excluded <M> module < > module capable

-*- Patch physical to virtual translations at runtime
General setup --->
[ ] Enable loadable module support --->
[*] Enable the block layer --->
System Type --->
[ ] FIQ Mode Serial Debugger
```

更多的裁剪可根据附件自行裁剪。

附件

-  menuconfig 配置菜单.pdf
559.56KB
-  linux-2.6.28内核配...解.pdf
1.33MB