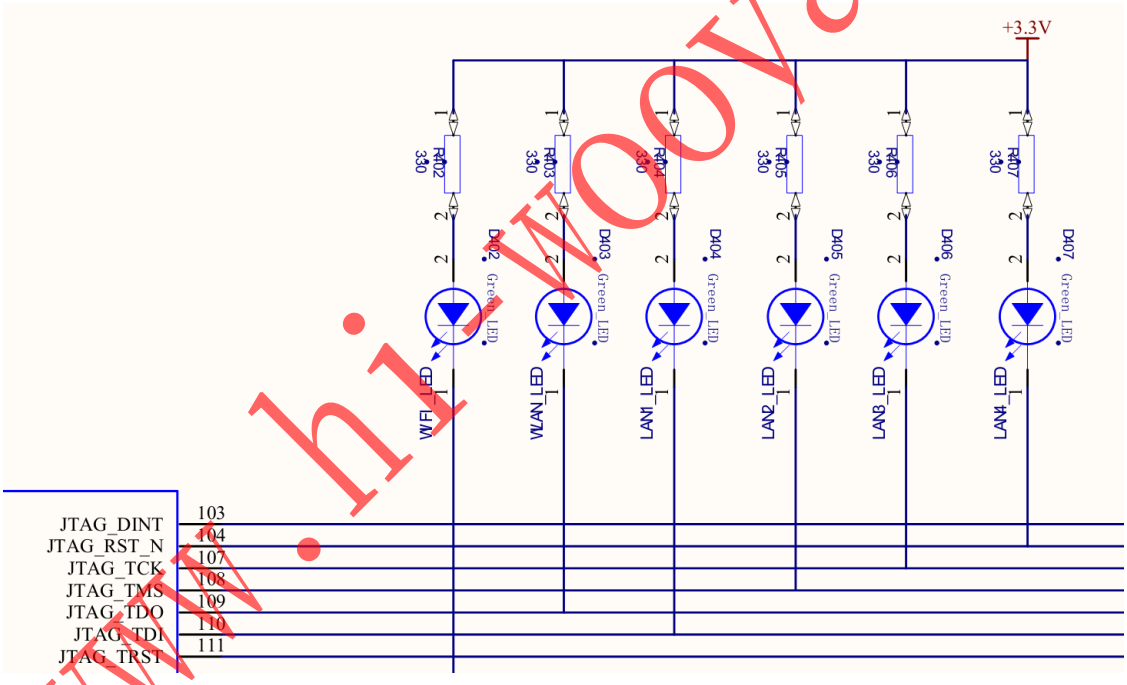
第 20 章　驱动开发之 LED 驱动程序_编写编译

本章目标

- 掌握嵌入式开发的步骤：编程、编译、烧写程序、运行
- 通过 GPIO 的操作了解软件如何控制硬件

20.1　硬件原理



由于发光二级管单向导电特性，即只有在正向电压（二极管的正极接正，负极接负）下才能导通发光。如图所示，如果 GPIO 输出高电平，LED 就会被点亮，如果 GPIO 输出低电平，LED 就会熄灭。

对于我们的驱动开发，无论是单片机、还是 ARM、或者是我们的 MIPS，核心思想，都是读写某个地址，即操作某个寄存器。

20.2　寄存器介绍

MT7620A 一共有 72 个 GPIO 管脚，这 72 个 GPIO，除了 GPIO0，其他全部是与其他功能引脚复用的。比如与 I2C、SPI、UARTF 等复用。

SPI	SPI_MISO	GPIO#6
	SPI_MOSI	GPO#5
	SPI_CLK	GPO#4
	SPI_CS0	GPIO#3

GPIO3~6 与 SPI 复用。

UARTF	RIN	GPIO#14
	DSR_N	GPIO#13
	DCD_N	GPIO#12
	DTR_N	GPIO#11
	RXD	GPIO#10
	CTS_N	GPIO#9
	TXD	GPIO#8
	RTS_N	GPIO#7

GPIO7~14，与 UARTF 即串口 2 复用。

这些复用关系，都可以通过查阅 MT7620A 的芯片手册得到。

而这些复用功能，我们可以通过 GPIOMODE 寄存器来进行选择，通过查手册得知，GPIOMODE 寄存器的地址为 0x10000060。

23. GPIOMODE: GPIO Purpose Select (offset: 0x0060)

Bits	Type	Name	Description	Initial Value

Bits	Type	Name	Description	Initial Value
31:30	RW	SUTIF_SHARE_MODE	Serial UTIF Pin Share Mode Sets the serial UTIF pin to operate in UARTL or I^2C mode. 0: Not shared 1: Shared with UARTL -overwrites the UARTLITE_GPIO_MODE setting. 2: Shared with I^2C - overwrites the I2C_GPIO_MODE setting. 3: Reserved	0x0
29:23	-	-	Reserved	0x0
22:21	RW	WDT_RST_MODE	Watchdog Timer GPIO Share Mode Sets the watchdog timer reset pin to operate in REFCLK_OUT or GPIO mode. 0: WDT_RST_N (normal mode) 1: REFCLK0_OUT 2: GPIO mode 3: Reserved	0x0
20	RW	PA_G_GPIO_MODE	Power Amplifier GPIO Share Mode Sets the power amplifier pin to operate in GPIO mode. 0: PA_PE_G0/PA_PE_G1/ANT_TRN/ANT_TRNB (normal mode) 1: GPIO Mode	0x1
19:18	RW	ND_SD_GPIO_MODE	NAND/SD GPIO Share Mode Sets the ND pins to operate in SD, BT or GPIO mode. 0: ND Mode 1: SD Mode (BT Coexist) 2: GPIO Mode 3: Reserved	0x2
17:16	RW	PERST_GPIO_MODE	PCIe Reset GPIO Share Mode Sets the PERST_N pin to operate in REFCLK0 or GPIO mode. 2'b00: PERST_N (normal mode) 2'b01: REFCLK0_OUT 2'b10: GPIO mode 2'b11: Reserved	0x2
15	RW	EPHY_LED_GPIO_MODE	LED JTAG GPIO Share Mode Sets an LED pin to operate in JTAG or GPIO mode. 0: Normal Mode (JTAG/EPHY_LED depending on bootstrapping settings) 1: GPIO Mode	0x0
14	-	-	Reserved	0x0

Bits	Type	Name	Description	Initial Value
13	RW	WLED_GPIO_MODE	WLAN LED GPIO Share Mode Sets the WLAN LED pin to operate in GPIO mode. 0: Normal mode 1: GPIO Mode	0x1
12	RW	SPI_REFCLK0_MODE	SPI Reference Clock Share Mode Sets SPI pins to operate in reference clock and GPI mode. 0: Normal SPI mode 1: SPI_CS1 pins are shared with the reference clock.	0x1
11	RW	SPI_GPIO_MODE	SPI GPIO Share Mode Sets the SPI pins to operate in GPIO mode. 0: Normal Mode 1: GPIO Mode	0x0
10	RW	RGMII2_GPIO_MODE	RGMII2 GPIO Share Mode Sets the RGMII2 pins to operate in GPIO mode. 0: Normal Mode 1: GPIO Mode	0x1
9	RW	RGMII1_GPIO_MODE	RGMII1 GPIO Share Mode Sets the RGMII1 pins to operate in GPIO mode. 0: Normal Mode 1: GPIO Mode	0x1
8:7	RW	MDIO_GPIO_MODE	MDIO GPIO Share Mode Sets the MDIO pin to operate in GPIO mode. 2'b00: Normal Mode 2'b01: REF_CLK Mode 2'b10: GPIO Mode 2'b11: Reserved	0x2
6	-	-	Reserved	0x0
5	RW	UARTL_GPIO_MODE	UART Lite GPIO Share Mode Sets the UART Lite pins to operate in GPIO mode. 0: Normal Mode 1: GPIO Mode	0x1
4:2	RW	UARTF_SHARE_MODE	UART Full Interface Share Mode Sets the UART Full interface to operate in PCM, I2S, and GPIO mode. A detailed description of the UARTF Mode Pin Sharing scheme is shown in the datasheet for this chip.	0x7
1	-	-	Reserved	0x0
0	RW	I2C_GPIO_MODE	I2C GPIO Share Mode Sets the I2C pins to operate in GPIO mode. 0: Normal Mode 1: GPIO Mode	0x1

GPIOMODE 寄存器 bit0 位用于选择 GPIO1、GPIO2 对应的引脚是用于 IIC 总线，还是用于 GPIO。

GPIOMODE 寄存器 bit11 位用于选择 GPIO3~6 对应的引脚用于 SPI 总线，还是用于 GPIO。

GPIOMODE 寄存器 bit2~4 位用于选择 UARTF 对应的引脚工作于哪个模块，具体定义如下。

Pin Name	3'b000 UARTF	3'b001 PCM, UARTF	3'b010 PCM, I2S	3'b011 I2S UARTF	3'b100 PCM, GPIO	3'b101 GPIO, UARTF	3'b110 GPIO I2S	3'b111 GPIO
RIN	RIN	PCMDTX	PCMDTX	RXD	PCMDTX	GPIO#14	GPIO#14	GPIO#14
DSR_N	DSR_N	PCMDRX	PCMDRX	CTS_N	PCMDRX	GPIO#13	GPIO#13	GPIO#13
DCD_N	DCD_N	PCMCLK	PCMCLK	TXD	PCMCLK	GPIO#12	GPIO#12	GPIO#12
DTR_N	DTR_N	PCMFS	PCMFS	RTS_N	PCMFS	GPIO#11	GPIO#11	GPIO#11
RXD	RXD	RXD	I2SSDI	I2SSDI	GPIO#10	RXD	I2SSDI	GPIO#10
CTS_N	CTS_N	CTS_N	I2SSDO	I2SSDO	GPIO#9	CTS_N	I2SSDO	GPIO#9
TXD	TXD	TXD	I2SWS	I2SWS	GPIO#8	TXD	I2SWS	GPIO#8
RTS_N	RTS_N	RTS_N	I2SCLK	I2SCLK	GPIO#7	RTS_N	I2SCLK	GPIO#7

从该表格可以看出，UARTF 对应的引脚，可以工作于 UARTF、PCM、I2S、GPIO 四种模式，将 GPIOMODE 寄存器 bit2~4 位设置为相应的值，就能让这些引脚工作于相应的模式，比如将 GPIOMODE 寄存器 bit2~4 位的值设置为 7，则让相应的引脚工于 GPIO 模式。

当将相应的引脚设置为 GPIO 以后，我们接下来就需要操作该 GPIO 了，操作 GPIO 不外乎就是设置 GPIO 是输入还是输出、让其输出高电平还是低电平、读取其电平状态。不管是哪种操作，都有对应的寄存器。

GPIO21_00_DIR: Programmed I/O Direction (offset: 0x0024)

Bits	Type	Name	Description	Initial value
31:22	-	-	Reserved	0x0
21:0	RW	PIODIR[21:0]	Program I/O Pin Direction These bits are used for selecting the data direction of the PIO pins. To configure any pin as an output, the corresponding bit should be set to '1'; to configure any pin as an input, the corresponding bit should be set to '0'. The value driven onto the PIO pins, are controlled by the PIOPOL, and PIODATA registers.	0x0

GPIO21_00_DIR 寄存器，用于设置 GPIO0~21 的方向，当相应的位被设置为 1，则表示相应的 GPIO 管脚被设置为了输出，如果被设置为了 0，则相应的 GPIO 引脚就被设置为了输入。

GPIO21_00_DATA: Programmed I/O Data (offset: 0x0020)

Bits	Type	Name	Description	Initial value
31:22	-	-	Reserved	0x0
21:0	RW	PIODATA[21:0]	Data Pin for Program I/O These bits are used for driving or sensing static signals on the PIO pins. To drive a value onto a PIO pin, the corresponding bit in the PIODIR register must be set. If the corresponding direction bit is set, the value written to the bit in the PIODATA register will be driven at the pin. A read of this register returns the value of the signals currently on the PIO pins. Note: The value of any bit in this register will be inverted with respect to the pin if the corresponding bit in the PIOPOL register is set, both in input and output modes. Note: The values read from the PIO pins are not synchronized; the user should be sure that the data will not be changing when this register is read, or should be aware that the bits which are not static at that time may be inaccurate.	RS

当相应 GPIO 引脚被设置为输出时，设置 GPIO21_00_DATA 寄存器的相应位为 1，则让该

GPIO 引脚输出了高电平，如果设置 GPIO21_00_DATA 寄存器的相应位为 0，则让该 GPIO 引脚输出了低电平。

当相应 GPIO 引脚被设置为输入时，则通过读取 GPIO21_00_DATA 寄存器时，就能读取相应的 GPIO 引脚的状态。

关于更多的 GPIO 操作的寄存器介绍，请自行查阅手册。

20.3　编写驱动程序

我们通过前面章节的学习，掌握了驱动程序的框架，接下来我们就来写一个驱动程序，实现操作 GPIO#40、GPIO#41、GPIO#42、GPIO#43 四个 GPIO 引脚。

具体驱动实现如下。

```
#include <linux/mm.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/vmalloc.h>
#include <linux/mman.h>
#include <linux/random.h>
#include <linux/init.h>
#include <linux/raw.h>
#include <linux/tty.h>
#include <linux/capability.h>
#include <linux/ptrace.h>
#include <linux/device.h>
#include <linux/highmem.h>
#include <linux/crash_dump.h>
#include <linux/backing-dev.h>
#include <linux/bootmem.h>
#include <linux/splice.h>
#include <linux/pfn.h>
#include <linux/export.h>
#include <linux/io.h>
#include <linux/aio.h>
#include <linux/kernel.h>
#include <linux/module.h>

#include <asm/uaccess.h>

#define MYLEDS_LED0_ON   0
```

```
#define MYLEDS_LED0_OFF 1
#define MYLEDS_LED1_ON  2
#define MYLEDS_LED1_OFF 3
#define MYLEDS_LED2_ON  4
#define MYLEDS_LED2_OFF 5
#define MYLEDS_LED3_ON  6
#define MYLEDS_LED3_OFF 7

volatile unsigned long *GPIOMODE;
volatile unsigned long *GPIO71_40_DIR;
volatile unsigned long *GPIO71_40_DATA;

static struct class *myleds_class;

static int myleds_open(struct inode *inode, struct file *file)
{
    /* 让 GPIO#40、GPIO#41、GPIO#42、GPIO#43 输出高电平，同时熄灭 LED0、LED1、LED2、
LED3、LED4 */
    *GPIO71_40_DATA &= ~((1<<0)|(1<<1)|(1<<2)|(1<<3));

    return 0;
}

static long myleds_unlocked_ioctl(struct file *file, unsigned int cmd, unsigned long
arg)
{
    switch(cmd)
    {
        case MYLEDS_LED0_ON:// 点亮 LED0
            *GPIO71_40_DATA &= ~(1<<0);
            break;
        case MYLEDS_LED0_OFF:    // 熄灭 LED0
            *GPIO71_40_DATA |= (1<<0);
            break;
        case MYLEDS_LED1_ON:// 点亮 LED1
            *GPIO71_40_DATA &= ~(1<<1);
            break;
        case MYLEDS_LED1_OFF:    // 熄灭 LED1
            *GPIO71_40_DATA |= (1<<1);
```

```
            break;
        case MYLEDS_LED2_ON:// 点亮 LED2
            *GPIO71_40_DATA &= ~(1<<2);
            break;
        case MYLEDS_LED2_OFF:   // 熄灭 LED2
            *GPIO71_40_DATA |= (1<<2);
            break;
        case MYLEDS_LED3_ON:// 点亮 LED3
            *GPIO71_40_DATA &= ~(1<<3);
            break;
        case MYLEDS_LED3_OFF:   // 熄灭 LED3
            *GPIO71_40_DATA |= (1<<3);
            break;
        default:
            break;
    }

    return 0;
}

/* 1.分配、设置一个 file_operations 结构体 */
static struct file_operations myleds_fops = {
    .owner              = THIS_MODULE,                  /* 这是一个宏，推向编译
模块时自动创建的__this_module 变量 */
    .open               = myleds_open,
    .unlocked_ioctl     = myleds_unlocked_ioctl,
};

int major;
static int __init myleds_init(void)
{
    printk("%s:Hello Wooya\n", __FUNCTION__);    // printk 用于驱动中添加打印，用
法和应用程序中的 printf 一样

    /* 2.注册 */
    major = register_chrdev(0, "myleds", &myleds_fops);

    /* 3.自动创建设备节点 */
    /* 创建类 */
```

```
    myleds_class = class_create(THIS_MODULE, "myleds");
    /* 类下面创建设备节点 */
    device_create(myleds_class, NULL, MKDEV(major, 0), NULL, "myleds");      //
/dev/myleds

    /* 4.硬件相关的操作 */
    /* 映射寄存器的地址 */
    GPIOMODE = (volatile unsigned long *)ioremap(0x10000060, 4);
    GPIO71_40_DIR = (volatile unsigned long *)ioremap(0x10000674, 4);
    GPIO71_40_DATA = (volatile unsigned long *)ioremap(0x10000670, 4);

    /* 设置相应管脚用于 GPIO */
    /*
    ** LED0 ---- GPIO#40
    ** LED1 ---- GPIO#41
    ** LED2 ---- GPIO#42
    ** LED3 ---- GPIO#43
    */
    *GPIOMODE |= (0x1<<15);

    /* 将 GPIO#40、GPIO#41、GPIO#42、GPIO#43、GPIO#44 设置为输出 */
    *GPIO71_40_DIR = (1<<0)|(1<<1)|(1<<2)|(1<<3);

    return 0;
}

static void __exit myleds_exit(void)
{
    unregister_chrdev(major, "myleds");
    device_destroy(myleds_class, MKDEV(major, 0));
    class_destroy(myleds_class);
    iounmap(GPIOMODE);
    iounmap(GPIO71_40_DIR);
    iounmap(GPIO71_40_DATA);
}

module_init(myleds_init);
module_exit(myleds_exit);
```

```
MODULE_LICENSE("GPL");
```

因为我们的开发板上跑的是 Linux 系统，因此操作某个寄存器的时候，需要将它的物理地址映射成虚拟地址，通过 ioremap()函数来进行映射，该函数的参数 1，就是对应的寄存器的物理地址，参数 2 是需要映射多大，可以理解为寄存器有多大，返回值就是该寄存器对应的虚拟地址了。

当寄存器地址映射为虚拟地址以后，然后基于前面写的驱动程序框架，就能很容易的编写出自己的 GPIO 驱动了，和操作单片机没有什么差别了。

20.4　编写 Makefile

驱动写好以后，自然是需要想办法来编译该驱动了，通过前面的章节的学习，我们需要给该驱动编写一个 Makefile 文件。

首先是新建一个文件夹，取名为 myleds，然后在 myleds 目录下再新建一个文件夹，取名为 src，然后将上面的驱动文件复制到 src 目录下，并且在 src 目录下新建一个 Makefile，内容如下。

```
obj-m += myleds.o
```

然后回到 myleds 目录，再创建一个 Makefile 文件，内容如下。

```
#
# Copyright (C) 2008-2012 OpenWrt.org
#
# This is free software, licensed under the GNU General Public License v2.
# See /LICENSE for more information.
#

include $(TOPDIR)/rules.mk
include $(INCLUDE_DIR)/kernel.mk

PKG_NAME:=myleds
PKG_RELEASE:=1

include $(INCLUDE_DIR)/package.mk

define KernelPackage/myleds
  SUBMENU:=Other modules
  # DEPENDS:=@!LINUX_3_3
  TITLE:=Motor driver
  FILES:=$(PKG_BUILD_DIR)/myleds.ko
  # AUTOLOAD:=$(call AutoLoad,30,myleds,1)
```

```
  KCONFIG:=
endef

define KernelPackage/myleds/description
 This is a myleds drivers
 endef

MAKE_OPTS:= \
    ARCH="$(LINUX_KARCH)" \
    CROSS_COMPILE="$(TARGET_CROSS)" \
    SUBDIRS="$(PKG_BUILD_DIR)"

define Build/Prepare
    mkdir -p $(PKG_BUILD_DIR)
    $(CP) ./src/* $(PKG_BUILD_DIR)/
endef

define Build/Compile
    $(MAKE) -C "$(LINUX_DIR)" \
        $(MAKE_OPTS) \
        modules
endef

$(eval $(call KernelPackage,myleds))
```

如果不清楚为什么这样编写 Makefile 文件，请查看前面章节的相关介绍。

20.5 编译驱动程序

接下来，我们就来配置编译驱动程序。首先将 myleds 文件夹传到 OpenWrt 源码的/home/openwrt/7620/openwrt/barrier_breaker/package/kernel 目录下。然后进入 OpenWrt 源码的顶层目录，执行 make menuconfig。

```
# cd /home/openwrt/7620/openwrt/barrier_breaker
# make menuconfig
```

在弹出的菜单界面里，首先进入 Kernel modules 选项。

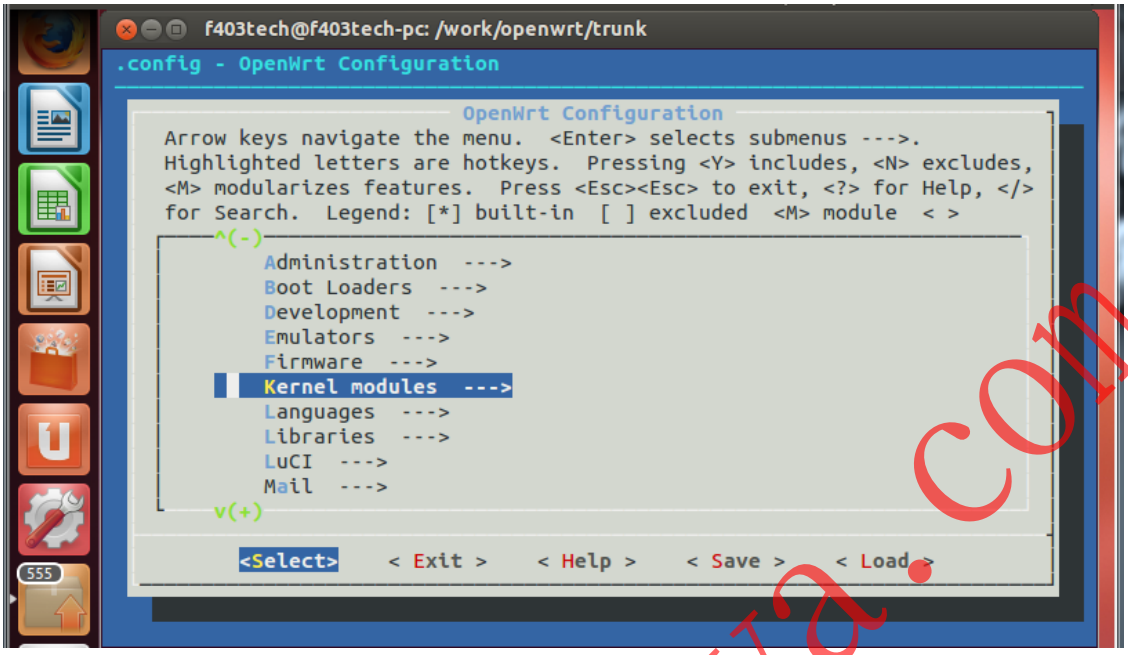


图 1

紧接着进入 Other modules 选项。

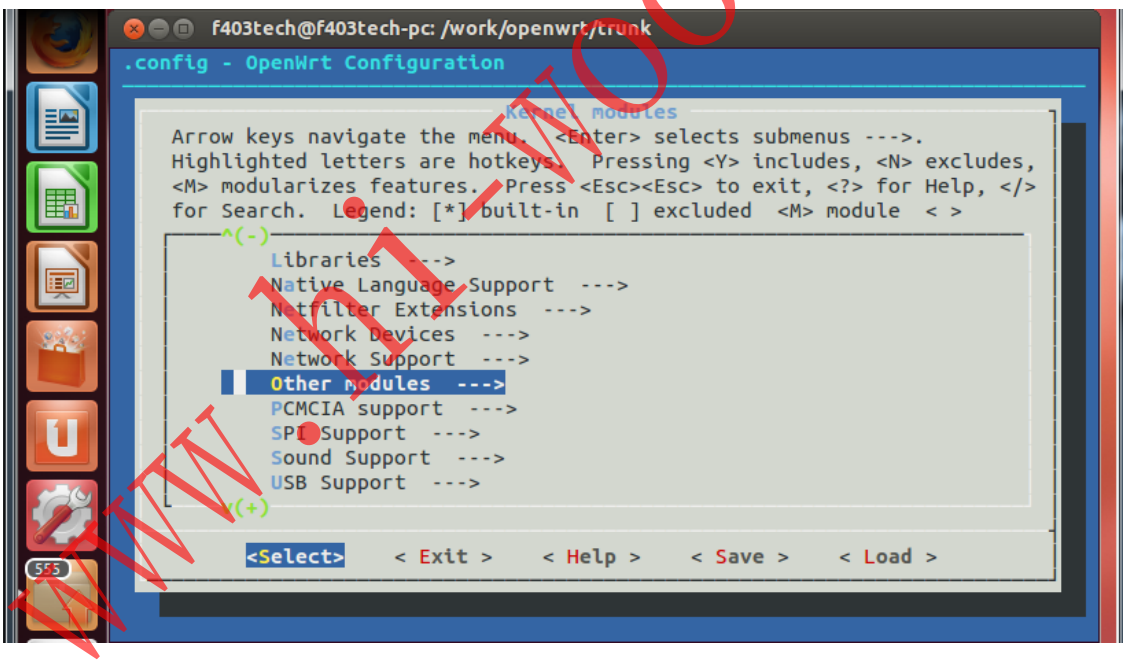
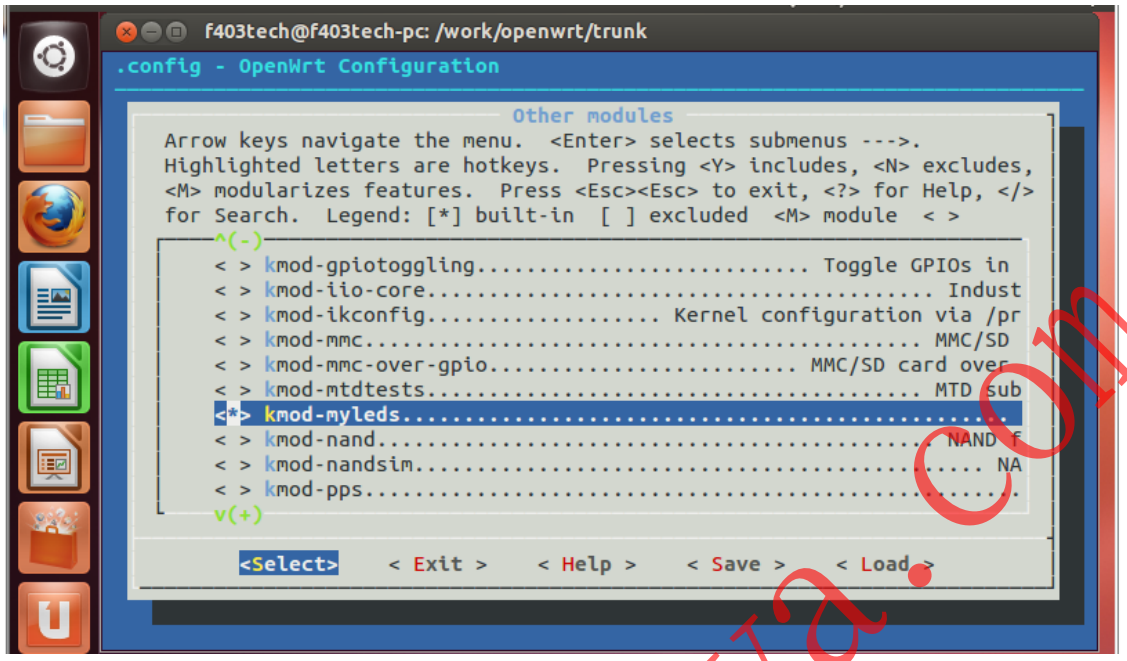


图 2

然后我们就能看到我们的驱动程序的选项了 kmod-myleds，将它配置成 y，即*。



最后，退出保存。然后再执行编译命令。

```
# make package/kernel/myleds/compile V=99
```

编译完成以后，就能在 openwrt/trunk/bin/ramips/packages 目录下看到我们的内核模块驱动程序的软件包 kmod-myleds_3.10.36-1_ramips_24kec.ipk。

20.6 动态的加载和卸载内核驱动模块软件包

通过前面的努力，我们终于得到了我们自己的内核驱动模块软件包了。接下来就来使用它。首先使用第 8 章中的方法，将 kmod-myleds_3.10.36-1_ramips_24kec.ipk 软件包传到开发板上面。

然后使用 opkg install 命令来安装软件包。

```
# opkg install kmod-myleds_3.10.36-1_ramips_24kec.ipk
```

使用 opkg list 命令来查看已经安装了哪些软件包。这里是否执行该命令都无所谓。

接下来进入我们的/lib/modules/3.10.36 目录，就能看到我们的驱动程序模块了。

```
# cd lib/modules/3.10.36/
```

```
root@OpenWrt:/lib/modules/3.10.36# ls
arc4.ko                  ipt_REJECT.ko            rndis_host.ko
asix.ko                  iptable_filter.ko        rt2800lib.ko
cdc-acm.ko               iptable_mangle.ko        rt2800mmio.ko
cdc_ether.ko             iptable_nat.ko           rt2800soc.ko
cfg80211.ko              iptable_raw.ko           rt2x00lib.ko
compat.ko                ipv6.ko                  rt2x00mmio.ko
crc-ccitt.ko             leds-gpio.ko             rt2x00soc.ko
crc-itu-t.ko             mac80211.ko              slhc.ko
crypto_blkcipher.ko      mii.ko                   usb-common.ko
eeprom_93cx6.ko          myleds.ko                usb_wwan.ko
ehci-hcd.ko              nf_conntrack.ko          usbcore.ko
ehci-platform.ko         nf_conntrack_ftp.ko      usbnet.ko
f403tech_drv.ko          nf_conntrack_ipv4.ko     usbserial.ko
gpio-button-hotplug.ko   nf_conntrack_ipv6.ko     x_tables.ko
ip6_tables.ko            nf_conntrack_irc.ko      xt_CT.ko
ip6t_REJECT.ko           nf_defrag_ipv4.ko        xt_LOG.ko
ip6t_ah.ko               nf_defrag_ipv6.ko        xt_REDIRECT.ko
ip6t_eui64.ko            nf_nat.ko                xt_TCPMSS.ko
ip6t_frag.ko             nf_nat_ftp.ko            xt_comment.ko
ip6t_hbh.ko              nf_nat_ipv4.ko           xt_conntrack.ko
ip6t_ipv6header.ko       nf_nat_irc.ko            xt_limit.ko
ip6t_mh.ko               nls_base.ko              xt_mac.ko
ip6t_rt.ko               ohci-hcd.ko              xt_mark.ko
ip6table_filter.ko       option.ko                xt_multiport.ko
ip6table_mangle.ko       ppp_async.ko             xt_nat.ko
ip6table_raw.ko          ppp_generic.ko           xt_state.ko
ip_tables.ko             pppoe.ko                 xt_tcpudp.ko
ipt_MASQUERADE.ko        pppox.ko                 xt_time.ko
root@OpenWrt:/lib/modules/3.10.36#
```

接下来就通过 insmod 命令来装载驱动模块。

```
# insmod myleds.ko
```

驱动安装成功以后，我们就能看到 GPIO 驱动对应的设备节点了。

```
root@OpenWrt:/lib/modules/3.10.36# ls /dev/myleds
/dev/myleds
root@OpenWrt:/lib/modules/3.10.36#
```

注意：

1). 该教程为我司(www.hi-wooya.com)原创教程，版权所有；

2). 该教程会不断更新、不断深入，详情请咨询我司客服；

3). 针对该教程，我们还有 QQ 群和论坛，专门负责技术答疑，详情请咨询我司客服。