第6章 具体单板的 LED 驱动程序

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

录制视频时，我的 source insight 里总是使用某个版本的内核。这没有关系，驱动程序中调用的内核函数，在这些 4.x 版本的内核里都是一样的。

6.1 怎么写 LED 驱动程序？

详细步骤如下：

① 看原理图确定引脚，确定引脚输出什么电平才能点亮/熄灭 LED

② 看主芯片手册，确定寄存器操作方法：哪些寄存器？哪些位？地址是？

③ 编写驱动：先写框架，再写硬件操作的代码

注意：在芯片手册中确定的寄存器地址被称为物理地址，在 Linux 内核中无法直接使用。

需要使用内核提供的 ioremap 把物理地址映射为虚拟地址，使用虚拟地址。

6.1.1 ioremap 函数的使用：

① 函数原型：

```
void __iomem *ioremap(resource_size_t res_cookie, size_t size)
```

使用时，要包含头文件：

```
#include <asm/io.h>
```

② 作用：

把物理地址 phys_addr 开始的一段空间(大小为 size)，映射为虚拟地址；返回值是该段虚拟地址的首地址。

```
virt_addr  = ioremap(phys_addr, size);
```

实际上，它是按页(4096 字节)进行映射的，是整页整页地映射的。

假设 phys_addr = 0x10002，size=4，ioremap 的内部实现是：

a) phys_addr 按页取整，得到地址 0x10000

b) size 按页取整，得到 4096

c) 把起始地址 0x10000，大小为 4096 的这一块物理地址空间，映射到虚拟地址空间，

假设得到的虚拟空间起始地址为 0xf0010000

d) 那么 phys_addr = 0x10002 对应的 virt_addr = 0xf0010002

③ 不再使用该段虚拟地址时，要 iounmap(virt_addr)：

```
void iounmap(volatile void __iomem *cookie)
```

6.1.2 volatile 的使用：

① 编译器很聪明，会帮我们做些优化，比如：

```
int   a;
a = 0;   // 这句话可以优化掉，不影响 a 的结果
a = 1;
```

② 有时候编译器会自作聪明，比如：

```
int *p = ioremap(xxxx, 4);  // GPIO 寄存器的地址
*p = 0;   // 点灯，但是这句话被优化掉了
*p = 1;   // 灭灯
```

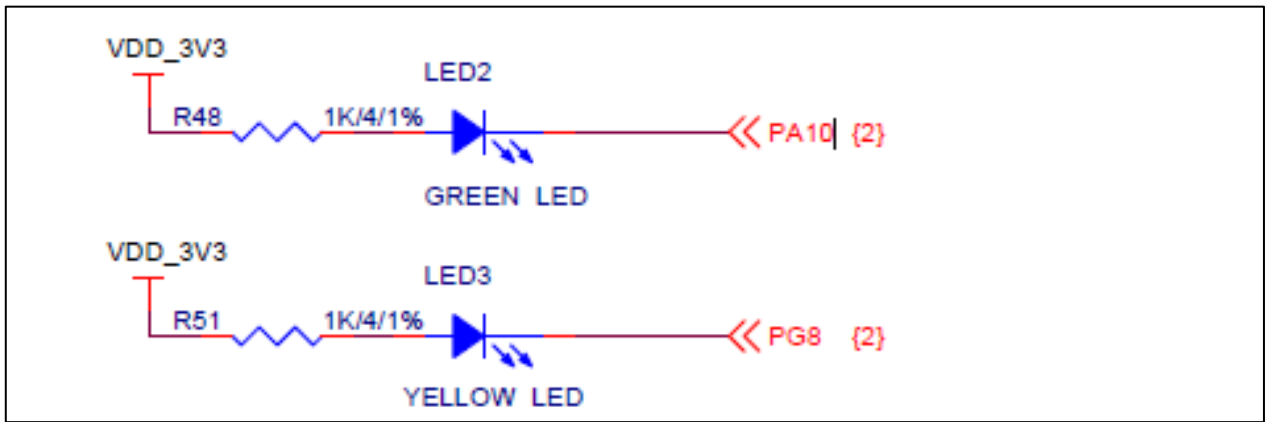
③ 对于上面的情况，为了避免编译器自动优化，需要加上 volatile，告诉它“这是容易出错的，别乱优化”：

```
volatile  int *p = ioremap(xxxx, 4);  // GPIO 寄存器的地址
*p = 0;   // 点灯，这句话不会被优化掉
*p = 1;   // 灭灯
```

6.2 百问网 STM32MP157 的 LED 驱动程序

在 STM32MP157 开发板上，我们为它设计了 2 个 LED。

7.2.1 看原理图确定引脚及操作方法



从上图可知，这 2 个 LED 用到了 PA10 PG8 2 个引脚。

在芯片手册里，这些引脚的名字是：PA10 PG82。可以根据名字搜到对应的寄存器。

当这些引脚输出低电平时，对应的 LED 被点亮；输出高电平时，LED 熄灭。

6.2.1 所涉及的寄存器操作

● 使能 GPIOA、GPIOG 时钟给 MPU，参考资料：CPU 开发手册 DM00327695.pdf《10: Reset and clock control (RCC)》10.7 RCC registers.

10.7.157 RCC AHB4 Periph. Enable For MPU Set Register (RCC_MP_AHB4ENSETR)

Address offset: 0xA28

Reset value: 0x0000 0000

This register is used to set the peripheral clock enable bit of the corresponding peripheral to '1'. It shall be used to allocate a peripheral to the MPU. Writing '0' has no effect, reading will return the effective values of the corresponding bits. Writing a '1' sets the corresponding bit to '1'.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	GPIOKEN	GPIOJEN	GPIOIEN	GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
					rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs

设置 b[22] 、b[16]为 bit1 就可以使能 GPIOG GPIOA 外设时钟给 MPU

- **步骤 2**：设置 GPIOA_10 GPIOG_08 为 GPIO 输出模式,参考资料：CPU 开发手册 DM00327695.pdf《10: General-purpose I/Os(GPIO)》。。

① 对于 GPIOA_IO10，设置如下寄存器：

GPIOA 的寄存器基地址如下图所示，参考 CPU 开发手册 DM00327695.pdf《2 Memory and bus architecture》

	0x50002000 - 0x500023FF	1 KB	GPIOA	*GPIO registers*

13.4.1 GPIO port mode register (GPIOx_MODER) (x = A to K, Z)

Address offset:0x00

Reset value: 0xFFFF FFFF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **MODER[15:0][1:0]:** Port x configuration I/O pin y (y = 15 to 0)
These bits are written by software to configure the I/O mode.
00: Input mode
01: General purpose output mode
10: Alternate function mode
11: Analog mode

设置 b[21:20]为 1 就可以配置 GPIOA_IO10 为输出模式。

② 对于 GPIOG_IO08，设置如下寄存器：

GPIOG 的寄存器基地址如下图所示，参考 CPU 开发手册 DM00327695.pdf《2 Memory and bus architecture》

		0x50008000 - 0x500083FF	1 KB	GPIOG	*GPIO registers*

13.4.1 GPIO port mode register (GPIOx_MODER) (x = A to K, Z)

Address offset:0x00

Reset value: 0xFFFF FFFF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **MODER[15:0][1:0]:** Port x configuration I/O pin y (y = 15 to 0)
These bits are written by software to configure the I/O mode.
00: Input mode
01: General purpose output mode
10: Alternate function mode
11: Analog mode

设置 b[17:16]为 1 就可以配置 GPIOG_IO08 为输出模式。

- **步骤 3**：设置 GPIOA_IO10、GPIOG_IO08 设置其输出电平,向相应的 bit 位写入 0/1 表示输出低/高。

GPIOA_IO10 寄存器地址为 基地址+偏移地址=0x50002000 + 0x14：

0x50002000 - 0x500023FF	1 KB	GPIOA	*GPIO registers*

如下图所示，设置 GPIOA_IO10 需要操作 b[10]，写入 1/0 来控制输出高低。

13.4.6 GPIO port output data register (GPIOx_ODR) (x = A to K, Z)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODR[15:0]:** Port output data I/O pin y (y = 15 to 0)
These bits can be read and written by software.
Note: For atomic bit set/reset, the ODR bits can be individually set and/or reset by writing to the GPIOx_BSRR or GPIOx_BRR registers (x = A..F).

GPIOG_IO08 寄存器地址为 基地址+偏移地址=0x50008000 + 0x14：

0x50008000 - 0x500083FF	1 KB	GPIOG	*GPIO registers*

如下图所示，设置 GPIOG_IO08 需要操作 b[10]，写入 1/0 来控制输出高低。

13.4.6 GPIO port output data register (GPIOx_ODR) (x = A to K, Z)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODR[15:0]:** Port output data I/O pin y (y = 15 to 0)
These bits can be read and written by software.
Note: For atomic bit set/reset, the ODR bits can be individually set and/or reset by writing to the GPIOx_BSRR or GPIOx_BRR registers (x = A..F).

6.2.2 写程序

使用 GIT 下载所有源码后，本节源码位于如下目录：

```
01_all_series_quickstart\
05_嵌入式 Linux 驱动开发基础知识\
source\02_led_drv\02_led_drv_for_boards\100ask_stm32mp157_src_bin
```

硬件相关的文件是 board_100ask_stm32mp157-src.c，其他文件跟 LED 框架驱动程序完全一样。

涉及的寄存器挺多，一个一个去执行 ioremap 效率太低。

先定义结构体，然后对结构体指针进行 ioremap，这些结构体在。

对于 GPIO，可以如下定义：

```
struct stm32mp1_gpio {
    volatile unsigned int moder;
    volatile unsigned int otyper;
    volatile unsigned int pupdr;
    volatile unsigned int idr;
    volatile unsigned int odr;
    volatile unsigned int bsrr;
};
/* enable GPIOA,GPIOG */
static volatile unsigned int *RCC_MP_AHB4ENSETR;

static struct stm32mp1_gpio *gpioA;
static struct stm32mp1_gpio *gpioG;
```

开始详细分析 board_100ask_stm32mp157-src.c。

它首先构造了一个 led_operations 结构体，用来表示 LED 的硬件操作：

```
176 static struct led_operations board_demo_led_opr = {
177     .num  = 4,
178     .init = board_demo_led_init,
179     .ctl  = board_demo_led_ctl,
180 };
```

led_operations 结构体中有 init 函数指针，它指向 board_demo_led_init 函数，在里面将会初始化 LED 引脚：使能、设置为 GPIO 模式、设置为输出引脚。

对于寄存器要先使用 ioremap 得到它的虚拟地址，以后使用虚拟地址访问寄存器：

```
static int board_demo_led_init (int which) /* 初始化 LED, which-哪个 LED */
{

    if (!RCC_PLL4CR)
    {
        // RCC_PLL4CR 地址：0x50000000 + 0x894
        RCC_PLL4CR = ioremap(0x50000000 + 0x894, 4);

        // RCC_MP_AHB4ENSETR 地址：0x50000000 + 0xA28
        RCC_MP_AHB4ENSETR = ioremap(0x50000000 + 0xA28, 4);

        // GPIOA_MODER 地址：0x50002000 + 0x00
        GPIOA_MODER = ioremap(0x50002000 + 0x00, 4);
```

```
        // GPIOA_BSRR 地址:  0x50002000 + 0x18
        GPIOA_BSRR = ioremap(0x50002000 + 0x18, 4);
    }

    if (which == 0)
    {
        /* enalbe PLL4, it is clock source for all gpio */
        *RCC_PLL4CR |= (1<<0);
        while ((*RCC_PLL4CR & (1<<1)) == 0);

        /* enable gpioA */
        *RCC_MP_AHB4ENSETR |= (1<<0);

        /*
         * configure gpa10 as gpio
         * configure gpio as output
         */
        *GPIOA_MODER &= ~(3<<20);
        *GPIOA_MODER |= (1<<20);
    }

    return 0;
}
```

led_operations 结构体中有 ctl 函数指针，它指向 board_demo_led_ctl 函数，在里面将会根据参数设置 LED 引脚的输出电平：

```
static int board_demo_led_ctl (int which, char status) /* 控制 LED, which-哪个 LED, st
atus:1-亮,0-灭 */
{
    if (which == 0)
    {
        /* to set gpio register: out 1/0 */
        if (status)
        {
            /* set gpa10 to let led on */
            *GPIOA_BSRR = (1<<26);
        }
        else
        {
            /* set gpa10 to let led off */
            *GPIOA_BSRR = (1<<10);
        }
    }
    return 0;
}
```

下面的 get_board_led_opr 函数供上层调用，给上层提供 led_operations 结构体：

```
182 struct led_operations *get_board_led_opr(void)
183 {
184     return &board_demo_led_opr;
185 }
```

6.2.3 上机实验

先启动 stm32mp157 开发板，挂载 NFS 文件系统。

然后执行以下命令安装驱动、执行测试程序：

```
[root@100ask:~]# insmod  100ask_led.ko
[root@100ask:~]# ./ledtest  /dev/100ask_led0  on
[root@100ask:~]# ./ledtest  /dev/100ask_led0  off
```

6.2.4 课后作业

a.在驱动里有 ioremap，什么时候执行 iounmap？请完善程序

b.视频里我们只实现了点一个 LED，修改代码支持两个 LED。