第14章 具体单板的按键驱动程序(查询方式)

14.1 GPIO 操作回顾

参考章节

- 《>>>>>>>>第五篇第 3 章普适的 GPIO 引脚操作方法》
- 《>>>>>>>>第五篇 3.3 具体单板的 GPIO 操作方法》。

① 使能电源/时钟控制器；

② 配置引脚模式；

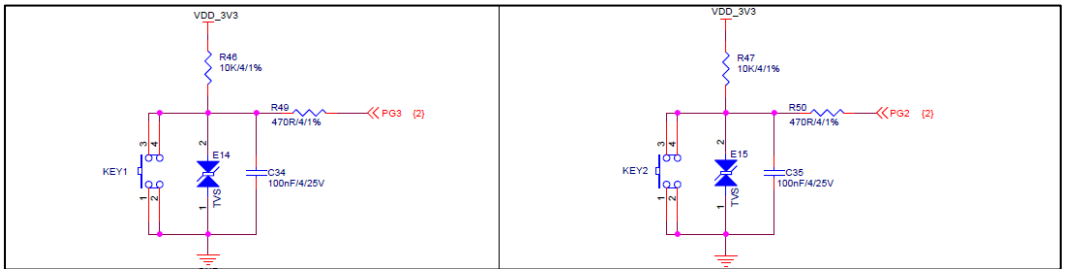
③ 配置引脚方向——输入/输出；

④ 输出电平/读取电平；

14.2 百问网 STM32MP157 的按键驱动程序(查询方式)

在 STM32MP157 开发板上，我们为它设计了 2 个按键。

14.2.1 先看原理图确定引脚及操作方法



平时按键电平为低，按下按键后电平为高。

按键引脚为 GPIOG_IO03、GPIOG_IO02。

14.2.2 15.2.2 再看芯片手册确定寄存器及操作方法

- **步骤 1**：使能 GPIOG

下图为针对 APU 的 GPIOA 至 K 的时钟使能寄存器，低 11 位有效。为了使用 GPIOG，我们需要将对应的 b[6]位设置为 1。

10.7.155 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

- **步骤 2**：设置 GPIOG_IO03、GPIOG_IO02 为 GPIO 输入模式。

GPIOx_MODER 用于配置 GPIO 的模式，包括输入、通用输出、多功能和模拟共四种模式。该寄存器共 32 位,，涉及 16 个 GPIO，每个 GPIO 对应 2 位。GPIOx_MODER 的各位定义如下，我们在这里分别选择 00 和 01 两种，各自对应输入和输出模式。（上电默认为输入悬空模式）。其中 00 对应输入功能，01 对应输出功能。

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[7:6]为 0 就可以配置 GPIOG_IO03 为输入模式，配置 b[5:4]为 0 就可以配置 GPIOG_IO02 为输入模式。

- **步骤 3**：读取 GPIOG_IO02 GPIOG_IO03 引脚电平

寄存器地址为：

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

读取 IDR 寄存器获取引脚状态寄存器，得到引脚电平：

13.4.5 GPIO port input data register (GPIOx_IDR) (x = A to K, Z)

Address offset: 0x10

Reset value: 0x0000 XXXX

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
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

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

Bits 15:0 **IDR[15:0]:** Port x input data I/O pin y (y = 15 to 0)
These bits are read-only. They contain the input value of the corresponding I/O port.

14.2.3 编程

1 程序框架

使用 GIT 下载所有源码后，本节源码位于如下目录

注意：git 仓库内可能暂时没有此目录，需要学员参考如下示例进行修改

```
01_all_series_quickstart\
05_嵌入式 Linux 驱动开发基础知识\source\
04_button_drv\02_button_drv_for_boards\05_button_drv_for_100ask_stm32mp157-pro
```

2 硬件相关的代码

主要看 board_100ask_stm32mp157-pro.c。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。

对于 GPIO，可以如下定义：

```
struct stm32mp157_gpio {
  volatile unsigned int MODER;    /*!< GPIO port mode register,               Addres
s offset: 0x00      */
  volatile unsigned int OTYPER;   /*!< GPIO port output type register,        Addres
s offset: 0x04      */
  volatile unsigned int OSPEEDR;  /*!< GPIO port output speed register,       Addres
s offset: 0x08      */
  volatile unsigned int PUPDR;    /*!< GPIO port pull-up/pull-down register,  Addres
s offset: 0x0C      */
  volatile unsigned int IDR;      /*!< GPIO port input data register,         Addres
s offset: 0x10      */
  volatile unsigned int ODR;      /*!< GPIO port output data register,        Addres
s offset: 0x14      */
  volatile unsigned int BSRR;     /*!< GPIO port bit set/reset,               Addres
s offset: 0x18      */
  volatile unsigned int LCKR;     /*!< GPIO port configuration lock register, Addres
s offset: 0x1C      */
  volatile unsigned int AFR[2];   /*!< GPIO alternate function registers,     Addres
s offset: 0x20-0x24 */
} ;
```

看一个驱动程序，先看它的入口函数，下列代码向上层驱动注册一个 button_operations 结构体,代码如下。

```
static struct button_operations my_buttons_ops = {
    .count = 2,
```

```
    .init = board_stm32mp157_button_init,
    .read = board_stm32mp157_button_read,
};

int board_stm32mp157_button_drv_init(void)
{
    register_button_operations(&my_buttons_ops);
    return 0;
}
```

button_operations 结构体中有 init 函数指针，它指向 board_stm32mp157_button_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。代码如下。

值得关注的下列代码中对 ioremap 函数的使用，它们是得到寄存器的虚拟地址，以后使用虚拟地址访问寄存器。

```
/* RCC_PLL4CR */
static volatile unsigned int *RCC_PLL4CR;

/* RCC_MP_AHB4ENSETR */
static volatile unsigned int *RCC_MP_AHB4ENSETR;

static struct stm32mp157_gpio *gpiog; /* KEY1: PG3, KEY2: PG2 */

static void board_stm32mp157_button_init (int which) /* 初始化 button, which-哪个 butt
on */
{
    if (!RCC_PLL4CR)
    {
        RCC_PLL4CR = ioremap(0x50000000 + 0x894, 4);
        RCC_MP_AHB4ENSETR = ioremap(0x50000000 + 0xA28, 4);

        gpiog = ioremap(0x50008000, sizeof(struct stm32mp157_gpio));
    }

    if (which == 0)
    {
        /* 1. enable PLL4
         * CG15, b[31:30] = 0b11
         */
        *RCC_PLL4CR |= (1<<0);
        while((*RCC_PLL4CR & (1<<1)) == 0);

        /* 2. enable GPIOG */
        *RCC_MP_AHB4ENSETR |= (1<<6);

        /* 3. 设置 PG3 为 GPIO 模式，输入模式
         */
        gpiog->MODER &= ~(3<<6);

    }
    else if(which == 1)
    {
        /* 1. enable PLL4
         * CG15, b[31:30] = 0b11
         */
```

```
        *RCC_PLL4CR |= (1<<0);
        while((*RCC_PLL4CR & (1<<1)) == 0);

        /* 2. enable GPIOG */
        *RCC_MP_AHB4ENSETR |= (1<<6);

        /* 3. 设置 PG2 为 GPIO 模式，输入模式
         */
        gpiog->MODER &= ~(3<<4);
    }

}
```

button_operations 结构体中还有有 read 函数指针，它指向 board_stm32mp157_button_read 函数，在里面将会读取并返回按键引脚的电平。代码如下。

```
static int board_stm32mp157_button_read (int which) /* 读 button，which-哪个 */
{
    //printk("%s %s line %d, button %d, 0x%x\n", __FILE__, __FUNCTION__, __LINE__, w
hich, *GPIO1_DATAIN);
    if (which == 0)
        return (gpiog->IDR & (1<<3)) ? 1 : 0;
    else
        return (gpiog->IDR & (1<<2)) ? 1 : 0;
}
```

14.2.4 测试

先启动 STM32MP157，挂载 NFS 文件系统。

安装驱动程序之后执行测试程序，观察它的返回值(执行测试程序的同时操作按键)：

```
[root@100ask:~]# insmod button_drv.ko
[root@100ask:~]# insmod board_drv.ko
[root@100ask:~]# insmod board_100ask_stm32mp157-pro.ko
[root@100ask:~]# ./button_test  /dev/100ask_button0
[root@100ask:~]# ./button_test  /dev/100ask_button1
```

14.2.5 课后作业

修改 button_test.c，使用按键来点灯