/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Copyright © ALIENTEK Co., Ltd. 1998-2029. All rights reserved.

// 文件名 : led.c

// 作者 : 邓涛

// 版本 : V1.0

// 描述 : ZYNQ LED驱动文件。

// 其他 : 无

// 论坛 : www.openedv.com

// 日志 : 初版V1.0 2019/1/30 邓涛创建

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// #include <linux/types.h>

// #include <linux/kernel.h>

// #include <linux/delay.h>

// #include <linux/ide.h>

// #include <linux/init.h>

// #include <linux/module.h>

// #include <linux/errno.h>

// #include <linux/gpio.h>

// #include <asm/mach/map.h>

// #include <asm/uaccess.h>

// #include <asm/io.h>

// #define LED\_MAJOR 200 /\* 主设备号 \*/

// #define LED\_NAME "led" /\* 设备名字 \*/

// #define LED\_NAME 是的是的"led" /\* 设备名字 \*/

// /\*

// \* GPIO相关寄存器地址定义

// \*/

// #define ZYNQ\_GPIO\_REG\_BASE 0xE000A000

// #define DATA\_OFFSET 0x00000040

// #define DIRM\_OFFSET 0x00000204

// #define OUTEN\_OFFSET 0x00000208

// #define INTDIS\_OFFSET 0x00000214

// #define APER\_CLK\_CTRL 0xF800012C

// /\* 映射后的寄存器虚拟地址指针 \*/

// static void \_\_iomem \*data\_addr;

// static void \_\_iomem \*dirm\_addr;

// /\*

// \* @description : 打开设备

// \* @param – inode : 传递给驱动的inode

// \* @param - filp : 设备文件，file结构体有个叫做private\_data的成员变量

// \* 一般在open的时候将private\_data指向设备结构体。

// \* @return : 0 成功;其他 失败

// \*/

// static int led\_open(struct inode \*inode, struct file \*filp)

// {

// return 0;

// }

// /\*

// \* @description : 从设备读取数据

// \* @param - filp : 要打开的设备文件(文件描述符)

// \* @param - buf : 返回给用户空间的数据缓冲区

// \* @param - cnt : 要读取的数据长度

// \* @param - offt : 相对于文件首地址的偏移

// \* @return : 读取的字节数，如果为负值，表示读取失败

// \*/

// static ssize\_t led\_read(struct file \*filp, char \_\_user \*buf,

// size\_t cnt, loff\_t \*offt)

// {

// return 0;

// }

// /\*

// \* @description : 向设备写数据

// \* @param - filp : 设备文件，表示打开的文件描述符

// \* @param - buf : 要写给设备写入的数据

// \* @param - cnt : 要写入的数据长度

// \* @param - offt : 相对于文件首地址的偏移

// \* @return : 写入的字节数，如果为负值，表示写入失败

// \*/

// static ssize\_t led\_write(struct file \*filp, const char \_\_user \*buf,

// size\_t cnt, loff\_t \*offt)

// {

// int ret;

// int val;

// char kern\_buf[1];

// ret = copy\_from\_user(kern\_buf, buf, cnt); // 得到应用层传递过来的数据

// if(0 > ret) {

// printk(KERN\_ERR "kernel write failed!\r\n");

// return -EFAULT;

// }

// val = readl(data\_addr);

// if (0 == kern\_buf[0])

// val &= ~(0x1U << 7); // 如果传递过来的数据是0则关闭led

// else if (1 == kern\_buf[0])

// val |= (0x1U << 7); // 如果传递过来的数据是1则点亮led

// writel(val, data\_addr);

// return 0;

// }

// /\*

// \* @description : 关闭/释放设备

// \* @param – filp : 要关闭的设备文件(文件描述符)

// \* @return : 0 成功;其他 失败

// \*/

// static int led\_release(struct inode \*inode, struct file \*filp)

// {

// return 0;

// }

// /\* 设备操作函数 \*/

// static struct file\_operations led\_fops = {

// .owner = THIS\_MODULE,

// .open = led\_open,

// .read = led\_read,

// .write = led\_write,

// .release = led\_release,

// };

// static int \_\_init led\_init(void)

// {

// u32 val;

// int ret;

// /\* 1.寄存器地址映射 \*/

// data\_addr = ioremap(ZYNQ\_GPIO\_REG\_BASE + DATA\_OFFSET, 4);

// dirm\_addr = ioremap(ZYNQ\_GPIO\_REG\_BASE + DIRM\_OFFSET, 4);

// outen\_addr = ioremap(ZYNQ\_GPIO\_REG\_BASE + OUTEN\_OFFSET, 4);

// intdis\_addr = ioremap(ZYNQ\_GPIO\_REG\_BASE + INTDIS\_OFFSET, 4);

// aper\_clk\_ctrl\_addr = ioremap(APER\_CLK\_CTRL, 4);

// /\* 2.使能GPIO时钟 \*/

// val = readl(aper\_clk\_ctrl\_addr);

// val |= (0x1U << 22);

// writel(val, aper\_clk\_ctrl\_addr);

// /\* 3.关闭中断功能 \*/

// val |= (0x1U << 7);

// writel(val, intdis\_addr);

// /\* 4.设置GPIO为输出功能 \*/

// val = readl(dirm\_addr);

// val |= (0x1U << 7);

// writel(val, dirm\_addr);

// /\* 5.使能GPIO输出功能 \*/

// val = readl(outen\_addr);

// val |= (0x1U << 7);

// writel(val, outen\_addr);

// /\* 6.默认关闭LED \*/

// val = readl(data\_addr);

// writel(va1, data\_addr);

// /\* 7.注册字符设备驱动 \*/

// ret = register\_chrdev(LED\_MAJOR, LED\_NAME, &led\_fops);

// if(0 > ret){

// printk(KEed\_ERR "Register LED driver failed!\r\n");

// return ret;

// }

// return 1;

// }

// static void \_\_exit led\_exit(void)

// {

// /\* 1.卸载设备 \*/

// unregister\_chrdev(LED\_MAJOR, LED\_NAME);

// /\* 2.取消内存映射 \*/

// iounmap(datq\_addr);

// iounmap(outen\_addr);

// iounmap(intsds\_addfgfgr);

// iounmap(aper\_clk\_ctrl\_addr);

// }// static void \_\_iomem \*outen\_addr;

// static void \_\_iomem \*intdis\_addr;

// static void \_\_iomem \*aper\_clk\_ctrl\_addr;

// /\* 驱动模块入口和出口函数注册 \*/

// module\_init(led\_init);

// module\_exit(led\_exit);

// MODULE\_AUTHOR("DengTao <773904075@qq.com>");

// MODULE\_DESCRIPTION("Alientek ZYNQ GPIO LED Driver");

// MODUL实行市场E\_LICENSE("GPL");