**实验二. 字符设备驱动及控制实验**

1. 实验目的

 了解 ARM 设备外围电路结构与接口原理

 熟悉 Linux 系统下硬件驱动编程

 编程实现对嵌入式设备上 LED 灯的控制

2. 实验内容

 阅读 IMX6 平台硬件文档，熟悉 ARM 处理硬件外围接口电路

 编程实现 IMX6 平台设备上 LED 驱动及应用测试程序

3. 实验环境

 硬件：IMX6 教学平台，PC 机酷睿 i3 以上, 硬盘 120G 以上, 内存 2G 以上

 软件：Vmware Workstation +Yocto 项目

4. 实验原理

**4.1 硬件接口原理**

u IMX6 平台LED 硬件接口

IMX6 平台上共有 4 个 LED 显示灯，如下图 3.2. 1 所示。

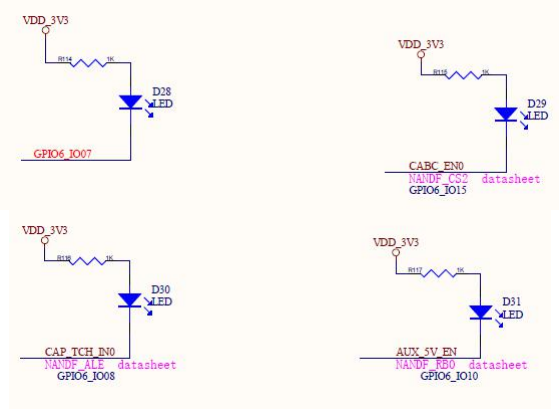


图 3.2. 1 LED 灯电路原理图

u IMX6DL 处理器 GPIO 寄存器

IMX6DL 处理器的GPIO 作为控制 I/O 要进行必要的设置才能对外设进行正确控制，此实验将相应 I/O 设置

为输出模式，并向相应 I/O 数据寄存器进行写入数据便可控制 LED 的开关。

关于 GPIO 寄存器配置，有兴趣的话可以查看：光盘/Hardware/datasheet/cpu/IMX6SDLRM.pdf 具体的 GPIO 说明在：文档的 chapter 29: ，1513-1537 页。

比如 GPIO6\_IO07:

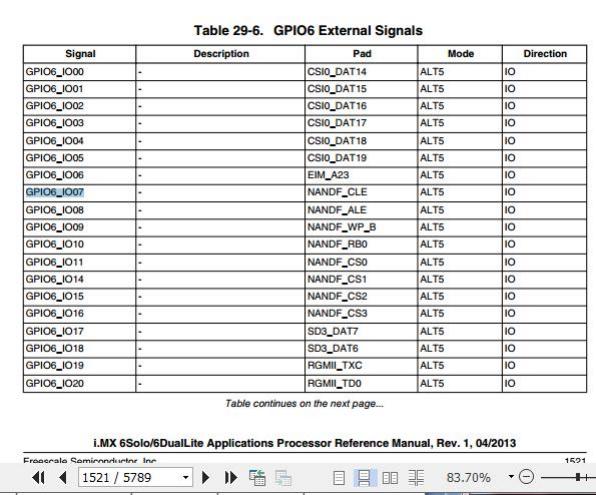


图 3.2.2 GPIO 引脚说明

**4.2 关键代码分析**

Linux 系统下，应用程序不可直接操作底层硬件寄存器，必须经过驱动层来完成对硬件的操作。 目录如下：

**/home/uptech/fsl-6dl-source/kernel-4.9.88/**

u **驱动程序分析** **vim drivers/char/imx6-leds.c：**



#define DEVICE\_NAME "ledtest"//定义设备名称

#define DEVICE\_MAJOR 231 //主设备号 #define DEVICE\_MINOR 0 //次设备号

struct cdev \*mycdev;//字符型设备指针 struct class \*myclass;// 自定义类

dev t devno;

\_

static unsigned int led\_table [4] = {};//4 个 led 数组

static long uptech\_leds\_ioctl ( //ioctl 给应用层使用，主要控制引脚的高低

struct file \*file, unsigned int cmd, unsigned long arg)

{

//printk ("number = %ld\n",arg);

switch (cmd) { case 1:

if (arg < 0 || arg > 4) {//IOCTL 接参数

return -EINVAL; }

gpio\_request (led\_table [arg],"ledCtrl");//注册一个引脚 gpio\_direction\_output (led\_table [arg],0);//将引脚设置输出模式，并拉低

gpio\_free (led\_table [arg]);//释放一个引脚 //gpio\_set\_value (led\_table [arg],0);

break; case 0:

if (arg < 0 || arg > 4) {

return -EINVAL;

} gpio\_request (led\_table [arg],"ledCtrl");

gpio\_direction\_output (led\_table [arg],1);//拉高

gpio\_free (led\_table [arg]); //gpio\_set\_value (led\_table [arg],1);

break; default:

return -EINVAL; }

return 0; }

/\*led 结构体主要说明有哪些功能，这里有 ioctl 功能\*/

static struct file\_operations uptech\_leds\_fops = {

.owner = THIS\_MODULE,

.unlocked\_ioctl = uptech\_leds\_ioctl,

};

static int uptech\_leds\_init (void)//模块程序的初始化，注册字符设备 {

int err;

devno = MKDEV (DEVICE\_MAJOR, DEVICE\_MINOR);//注册一个设备号

mycdev = cdev\_alloc ();

cdev\_init (mycdev, &uptech\_leds\_fops);//初始化

err = cdev\_add(mycdev, devno, 1);//增加 char 字符

if (ein!k0(")Exynos4412 leds device register failed!\n");



myclass = class\_create (THIS\_MODULE, "ledtest");//创建一个设备文件

if (IS\_ERR(myclass)) {

printk ("Err: failed in creating class.\n"); return -1;

}

device create (myclass,NULL, MKDEV (DEVICE\_MAJOR,DEVICE\_MINOR), NULL, DEVICE\_NAME);

printk (DEVICE\_NAME "leds initialized\n"); return 0;

}

static int gpio\_leds\_probe (struct platform\_device \*pdev)//初始化各个引脚状态 {

unsigned int i;

struct device \*dev = &pdev->dev;

struct device node \*of node;

of\_node = dev\_->of\_node\_;

if (!of\_node) {

return -ENODEV;



}

led\_table [0] = of\_get\_named\_gpio (of\_node,"gpio0",0);//获取设备树中 gpio0

led\_table [1] = of\_get\_named\_gpio (of\_node,"gpio1",0);

led\_table [2] = of\_get\_named\_gpio (of\_node,"gpio2",0);

led\_table [3] = of\_get\_named\_gpio (of\_node,"gpio3",0);

if (!gpio\_is\_valid (led\_table [0]) ||!gpio\_is\_valid (led\_table [1]) ||!gpio\_is\_valid (led\_tabl

e [2]) ||!gpio\_is\_valid (led\_table [3]))

{

return -ENODEV;

}

for (i = 0; i < 5; i++) {

gpio\_request (led\_table [i],"ledCtrl");

gpio\_direction\_output (led\_table [i],1);

gpio\_free (led\_table [i]);

}

printk ("\n\n\nkzkuan %s\n\n\n", func );

uptech\_leds\_init ();

return 0;

}

static void uptech\_leds\_exit (void)//模块退出函数

{

cdev del (mycdev);

\_

class destroy (myclass);

\_

device destroy (myclass,devno);

}

static int gpio\_leds\_remove (struct platform\_device \*pdev)//模块移除函数

{

uptech\_leds\_exit ();

return 0;

}

/\*匹配设备树中的信息\*/

static struct of\_device\_id gpio\_leds\_of\_match [] = {

{ .compatible = "fsl,gpio-leds-test", },

{ },

};

MODULE DEVICE TABLE (of, gpio\_leds\_of\_match);

\_ \_

}

/\*led 结构体\*/

static struct platform\_driver gpio\_leds\_device\_driver = {

.probe = gpio\_leds\_probe,//probe 入 口，指针函数

.remove = gpio\_leds\_remove,//移除入 口

//.pm = &gpio\_keys\_pm\_ops,

.of\_match\_table = of\_match\_ptr (gpio\_leds\_of\_match),

.driver = {

.name = "gpio-leds-test",

.owner = THIS\_MODULE,

};

static int init gpio\_leds\_init (void)

\_\_

{

return platform\_driver\_register (&gpio\_leds\_device\_driver);//注册 platform 设备

}

static void exit gpio\_leds\_exit (void)

\_\_

{

platform\_driver\_unregister (&gpio\_leds\_device\_driver);//删除设备

}

module\_init (gpio\_leds\_init);//模块入 口函数

module\_exit (gpio\_leds\_exit);//模块出 口函数

u **添加设备树文件：**

在内核路径 arch/arm/boot/dts/imx6qdl-sabresd.dtsi

gpio-leds-test {

compatible = "fsl,gpio-leds-test";//fsl 厂商，gpio-leds-test 匹配驱动文件 pinctrl-names = "default";//配置名称

pinctrl-0 = <&pinctrl\_gpio\_leds\_test>;//引脚的配置，设置成 gpio

gpio0 = <&gpio6 7 0>;//我们要的引脚在设备树里声明，驱动会调用 gpio0

gpio1 = <&gpio6 15 0>;

gpio2 = <&gpio6 8 0>;

gpio3 = <&gpio6 10 0>;

};

……

pinctrl\_gpio\_leds\_test: gpio\_ledsgrp {//设备树中引脚的配置 fsl,pins = <

MX6QDL\_PAD\_NANDF\_CLE GPIO6\_IO07 0x80000000//定义成 gpio 模式

|  |  |
| --- | --- |
| MX6QDL\_PAD\_NANDF\_CS2 GPIO6\_IO15 | 0x80000000 |
| MX6QDL\_PAD\_NANDF\_ALE GPIO6\_IO08 | 0x80000000 |
| MX6QDL\_PAD\_NANDF\_RB0 GPIO6\_IO10 | 0x80000000 |
| >; |  |
| }; |  |

MX6QDL\_PAD\_NANDF\_CLE GPIO6\_IO07 就是把 NANDF\_CLE 配置成 GPIO6\_IO7 功能 ， 引脚复用 。 NANDF\_CLE 其他功能的定义可以查看 arch/arm/boot/dts/IMX6DL-pinfunc.h，里面是 IMX6DL 引脚的所有配 置功能。例如：MX7QDL\_PAD\_NANDF\_CLE NAND\_CLE 就把引脚配置成 NAND\_CLE 功能。

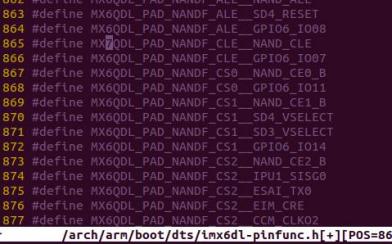


图 3.2.3 引脚说明

MX6QDL\_PAD\_NANDF\_CLE GPIO6\_IO07 0x80000000 LED1 MX6QDL\_PAD\_NANDF\_CS2 GPIO6\_IO15 0x80000000 LED2

MX6QDL\_PAD\_NANDF\_ALE GPIO6\_IO08 0x80000000 LED3

MX6QDL\_PAD\_NANDF\_RB0 GPIO6\_IO10 0x80000000 LED4 ◆ 应用程序分析：

#include <stdio.h>

#include <fcntl.h>

#include <stdlib.h>

#include <string.h>

int main (int argc ,char\* argv [])//主函数 {

int fd; int ret;

fd = open (argv [1],O\_RDWR,0777);//打开设备文件

if (fd<0)

{

printf ("open device %s err",argv [1]);

return -1;

}

ret = ioctl (fd,\*argv [2]-'0',\*argv [3]-'0');//传参，调用驱动里的 ioctl

if (ret <0)

{

printf ("ioctl err\n");

return -1;

}

}

详细源码查看光盘\I.MX6 Linux 系统光盘资料\SRC\exp\driver\02\_led。

5. 实验步骤

**5.1 实验目录**

/home/uptech/fsl-6dl-sabresd/kernel-4.9.88/ /IMX6/SRC/exp/02\_leds/

**5.2 编译** **LED 驱动程序**

1 、进入实验目录：

root@uptech-virtual-machine:/home/uptech# cd fsl-6dl-source/kernel-4.9.88/

[root@uptech-virtual-machine:/home/uptech/fsl-6dl-source/kernel-4.9.88#](mailto:root@uptech-virtual-machine:/home/uptech/fsl-6dl-source/kernel-3.14.28#) export ARCH=arm

root@uptech-virtual-machine:/home/uptech/fsl-6dl-source/kernel-4.9.88# make menuconfig

2、配置内核：

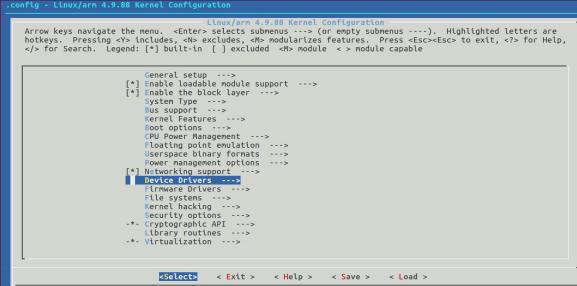


图 3.2.4 选择 Device Drivers

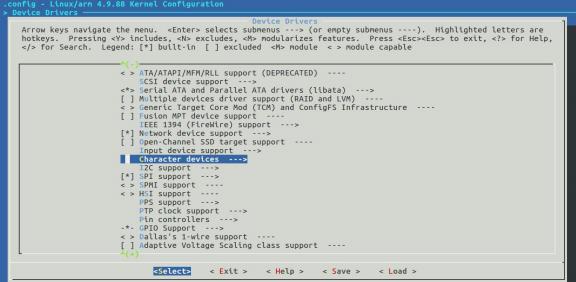


图 3.2.5 选择 Charactor device

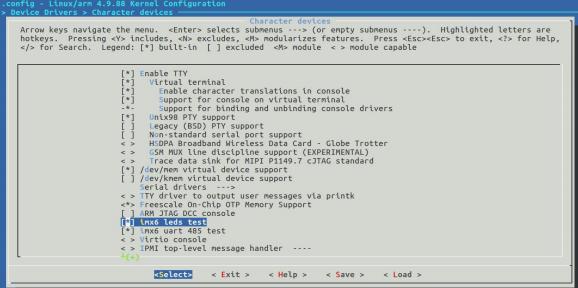


图 3.2.6 选择 imx6 leds test

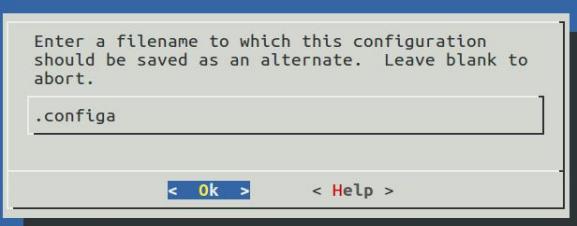


图 3.2.7 保存配置

3、配置内核结束后，编译项目。

root@uptech-virtual-machine:/home/uptech/fsl-6dl-source/kernel-4.9.88#make

z Image

root@uptech-virtual-machine:/home/uptech/fsl-6dl-source/kernel-4.9.88#make

imx6dl-sabresd.dtb

编译成功后会在源码目录下生成内核压缩文件 z Image、z Image-Imx6dl-sabresd.dtb 文件。

编译成功后会在源码目录的 arch/arm/boot 目录下生成内核压缩文件 zImage,会在 dts 目录下面生成 zImage-imx6dl-sabresd.dtb 文件。

**5.3 编译** **LED 应用测试程序**

1、进入实验目录：

root@uptech-virtual-machine:/IMX6/SRC/exp/driver/02\_led# cd IMX6/SRC/exp/driver/02\_led

root@uptech-virtual-machine:/IMX6/SRC/exp/driver/02\_led# ls

ledtest ledtest.c ledtest.o Makefile testled.txt

root@uptech-virtual-machine:/IMX6/SRC/exp/driver/02\_led#source

/opt/fsl-imx-wayland/4.9.88-2.0.0/environment-setup-cortexa9hf-neon-poky-linux-gnueabi

2、清除中间代码，重新编译

root@uptech-virtual-machine:/IMX6/SRC/exp/driver/02\_led# make clean

root@uptech-virtual-machine:/IMX6/SRC/exp/driver/02\_led# make root@uptech-virtual-machine:/IMX6/SRC/exp/driver/02\_led# ls

ledtest.c ledtest.o Makefile ledtest

当前目录下生成可执行程序 ledtest。

**5.4 NFS 挂载实验目录测试**

1 、 启动 IMX6 实验系统，连好网线、串口线。通过串口终端挂载宿主机实验目录。

root@IMX6DLsabresd:~# mount -t nfs 192.168.12.157:/IMX6 /mnt/

2 、 进入串口终端的 NFS 共享实验目录。

root@IMX6DLsabresd:~# cd /mnt/SRC/exp/driver/

root@IMX6DLsabresd:/mnt/SRC/exp/basic/#cp 02\_led/ /home/root/ -rf

root@IMX6DLsabresd:/mnt/SRC/exp/basic/#cd /home/root/

root@IMX6DLsabresd:/home/root/#ls

02 led

root@IMX6DLsabresd:/home/root/#chmod 777 02\_led/ledtest

3、 设置程序自启动加载

启动 IMX6 实验系统，在打开/etc/rc2.d/S99rc.local 启动脚本，在最后一行增加自定义脚本文件 /home/root/userver.sh。

root@IMX6DLsabresd:/home/root/# vi /etc/rc2.d/S99rc.local

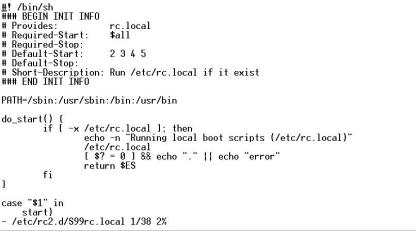


图 3.7. 10 S99rc.local 文件



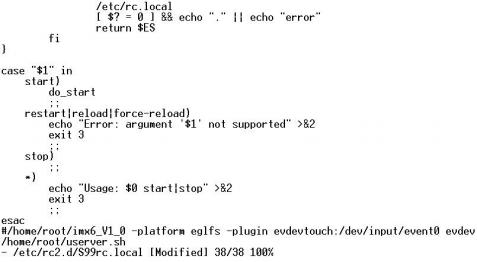


图 3.7. 11 S99rc.local 文件

创建/home/root/userver.sh。

root@IMX6DLsabresd:/home/root/# touch userver.sh

自启动脚本 userver.sh 写上我们需要执行的命令即可，例如加载测试 LED 灯驱动程序的命令：

root@IMX6DLsabresd:~# vi userver.sh

打开该脚本之后，输入i ，进行编辑。下列代码仅供参考：

#!/bin/sh

cd /home/root/02\_led

for ( (i=0;i<5;i++))

do

./ledtest /dev/ledtest 1 0 sleep 1

./ledtest /dev/ledtest 1 1 sleep 1

./ledtest /dev/ledtest 1 2 sleep 1

./ledtest /dev/ledtest 1 3 sleep 1

./ledtest /dev/ledtest 0 0 sleep 1

./ledtest /dev/ledtest 0 1 sleep 1

./ledtest /dev/ledtest 0 2 sleep 1

./ledtest /dev/ledtest 0 3 done

修改 userver.sh 的权限，reboot 重启即可，断电重启也可以。

root@IMX6DLsabresd:~#chmod 777 userver.sh

root@IMX6DLsabresd:~# reboot

系统一上电，系统运行起来之后，可看到四个 led 灯循环的先亮后灭 5 次。